WebCrow

Solving Italian crosswords using the Web

Technical Report

Giovanni Angelini Marco Ernandes Marco Gori

April 2005

Dipartimento di Ingegneria dell'Informazione,
Via Roma 56,
53100 Siena, ITALY,
[angelini|ernandes|marco]@dii.unisi.it,
Home page: http://airgroup.dii.unisi.it

Abstract

We designed and implemented a software system, called WebCrow, that represents the first solver for Italian crosswords and the first system that tackles a language game using the Web as knowledge base. Its core feature is the Web Search Module that produces a special form of web-based question answering that we call clue-answering. This paper will focus its attention on this task.

The web-search approach has proved itself to be very consistent: using a limited set of documents (30 for each clue) the clue-answering process is able to retrieve over two thirds of the correct answers. In many cases the targeted word is given in output among the very first most probable candidates and in nearly 15% of clues the correct answer appears in first position.

To complete the crossword solving problem the system has to fill the grid with the best set of word answers. Currently, WebCrow performances are interesting: crosswords that are "easy" for expert humans (i.e. crosswords from the cover pages of *La Settimana Enigmistica*™) are solved, in a 15 minutes time limit, with 80% of correct words and over 90% of correct letters. Crosswords that are designed for experts (i.e. examples by S. & A. Bartezzaghi both in *La Settimana Enigmistica* and in *La Repubblica*) WebCrow places correctly two thirds of the words and around 80% of the letters.

1 Introduction

1.1 Motivations and relevant literature.

Crosswords is probably the most played language puzzle worldwide and provides a very challenging game for human intelligence. *La Settimana Enigmistica*, the main Italian crossword magazine, sells over one million copies weekly. It is estimated that over 50 million Americans solve crosswords[1] with frequency.

Problems like solving crosswords from clues are reputed as AI-complete [4]. This enormous complexity is due to its semantics and the large amount of encyclopedic knowledge required.

AI started developing an interest for crossword solving only recently. The first experience reported in the literature is the Proverb system [2] that reached human-like performances on American crosswords using a great number of knowledge-specific expert modules and a crossword database of great dimensions[2].

We believe that recent developments in computer technology, such as the Web, search engines, information retrieval and machine learning techniques, can enable computers to enfold with semantics real-life concepts. With this in mind we designed a software system, called WebCrow, whose major assumption is to attack crosswords (within competition time limits) making use of the Web as its primary source of knowledge, being this the most extremely rich and self-updating repository of human knowledge. This represents a different approach with respect to Proverb, because WebCrow does not possess any knowledge-specific expert module. Nevertheless, in order to assure the system robustness to all sorts of clues, WebCrow makes also use of a strict set of other useful modules, which includes a dictionary and a small CrossWord DataBase[3] (CWDB).

The web-based clue-answering paradigm aspires to stress the generality of WebCrow's knowledge and its language-independence. We will show in this paper that Web Search, thanks to the fact that in clue-answering we priorly know the exact length of the correct answer, can produce extremely effective results providing the most important source of knowledge for the clue-answering process.

[1] It has been recently observed that this sort of activity helps to prevent from developing mental decline

[2] Before Proverb, AI limited its analysis to the *crossword generation* problem [5]. This makes a closed-world assumption by requiring a predefined dictionary of legal words and results to be an NP-complete task that can be solved in a few seconds.

[3] The database used by Proverb was about one order of magnitude greater than ours.

1.2 Problem setting and results.

Italian crosswords tend to be extremely difficult to handle because they contain a great quantity of word plays, neologisms, compound words, ambiguities and a deep involvement in socio-cultural and political topics, often treated with irony. The latter is a phenomenon, especially present in newspapers, that introduces an additional degree of complexity in crossword solving since it requires the possession of a very broad and fresh knowledge that is also robust to volunteer language vagueness and ambiguity.

We have collected 685 examples of solved Italian crosswords, each one containing an average of 62.7 clues. These examples were mainly obtained from two sources: the main Italian crossword magazine *La Settimana Enigmistica* (due to its popularity and history, this publisher sets a probable standard for Italian crosswords) and an important on-line newspaper's crossword section, *La Repubblica*. Other examples were downloaded from crossword-dedicated web sites. Sixty crosswords (3685 clues, avg. 61.4 each) were randomly extracted from these subsets in order to form the experimental test suite. The remaining crosswords constituted a database (CWDB) of clue-target pairs that was used as an aid for the generation of the candidate-answer lists.

Given this test set WebCrow's challenge was to answer all the clues and to subsequently fill the slots with the highest percentage of correct words. As in many human competitions a 15 minutes time limit was given for each example.

The version of WebCrow that is discussed here is basic but it has already given very promising results. In over two thirds of the clues the correct answer was found by the Web Search Module within the downloaded documents and in some cases (nearly 15%) this answer was the most probable (i.e., appearing a the top of the list). The addition of the other modules has raised the coverage to 99% and the probability of having the targeted word in first position to over 35%. Finally, solving the Constraint Satisfaction problem by filling the crossword puzzle, WebCrow averaged on the overall test set around 70% words correct and 80% letters correct. On the examples that experts consider "easy", as the examples from the cover pages of *La Settimana Enigmistica*, WebCrow performed with 80,0% words correct (100% in one case) and 90.1% letters correct. On more difficult examples the percentage of correct words was steadily above 60%: 67,6% with la *La Settimana Enigmistica* (81% letters) and 62.9% with *La Repubblica* (73% letters).

2 The system architecture

WebCrow is a modular-based system. Therefore, it is also possible to plug in additional *ad hoc* modules in order to increase the system's performances. A sketch of WebCrow's architecture is given in figure 1.

The WebCrow solving process can be divided in two phases. During the first one, all the clues of a puzzle are passed by the coordinator to all the "List Generator" modules. Each of them returns for each clue a list of possible solutions. All lists are then merged by the "Merger", using the confidence values of each list and the probabilities associated to each candidate of a list. At the end of this phase, a unique list of candidate-probability pairs is generated for each clue. Finally, WebCrow has to face a constrain-satisfaction problem. From each clue list a candidate has to be chosen and inserted in the crossword-puzzle, trying to satisfy the intrinsic constrains. The aim of this phase is to find an admissible solution which maximize the number of correct words inserted.

3 Using the Web for clue-answering: the Web Search Module.

The objective of the Web Search Module (WSM) is to find sensible answers to crossword clues, that are expressed in natural language, by exploiting the Web and search engines (SE). This task recalls that of a Web-based Question Answering system. However, with crossword clues, the nature of the problem changes sensibly, often becoming more challenging than classic QA[4]. The main differences are:

- clues are mostly formulated in a non-interrogative form (i.e: $\prec$*La voce narrante del Nome della Rosa:* **adso**$\succ$[5]) making the task of determining the answer-type more subtle.
- clues can be voluntarily ambiguous and misleading (i.e: $\prec$*Quello liquido non attacca:* **scotch**$\succ$[6])

[4] The main reference for standard QA is the TREC competition [11].

[5] $\prec$*The background narrator of the Name of the Rose:* **adso**$\succ$

[6] $\prec$*The liquid one does not stick:* **scotch**$\succ$, the clue ambiguously refers to the two senses of the target: scotch-whisky and scotch-tape.

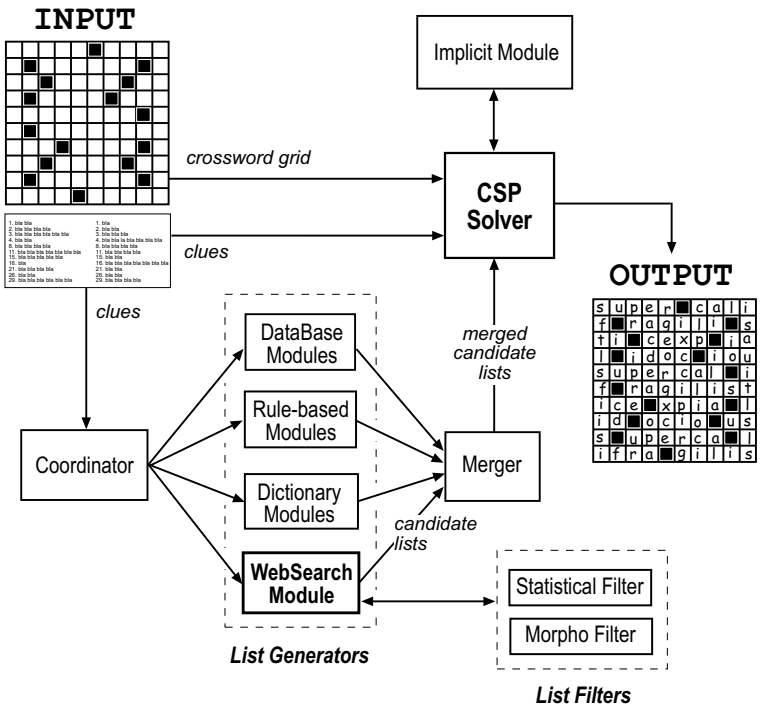

Figure 1: **WebCrow.** A general overview of WebCrow's architecture (inspired by Proverb's).

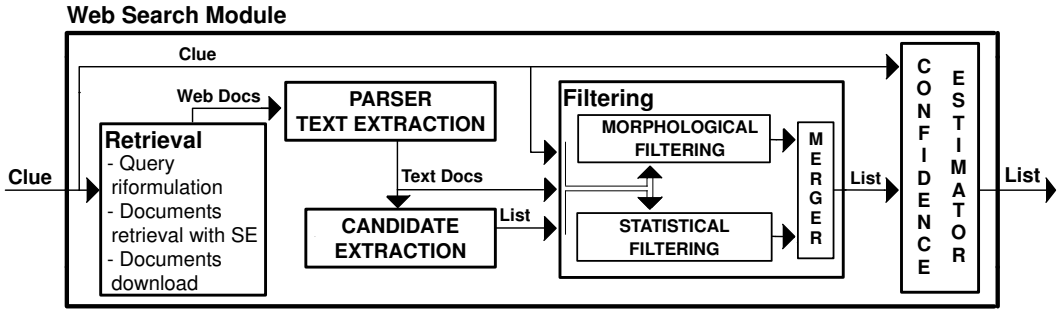

Figure 2: **Web Search Module.** A sketch of the internal architecture of the Web Search Module.

- the topic of the questions can be both factoid[7] and non-factoid (i.e: $\prec$*Ci si va al buio:* **cinema**$\succ$[8])
- there is a unique and precise correct answer: a single word or a compound word (i.e. $\prec$*Ha cambiato il linguaggio della tv:* **ilgrandefratello**$\succ$[9]), whereas in QA the answers are usually a 50/250-byte-passage in which the target has to be recognizable by humans.
- the list of candidate answers requires also a global confidence score, expressing the probability that the target is within the list.

The only evident advantage in crossword solving is that we priorly know the exact length of the words that we are seeking. We believe that, thanks to this property, web search can be extremely effective and produce a strong clue-answering.

[7] As TREC questions, like *Who was the first American in space?*

[8] $\prec$*We go there in the darkness:* **cinema**$\succ$

[9] $\prec$*It changed the language used in television:* **thebigbrother**$\succ$

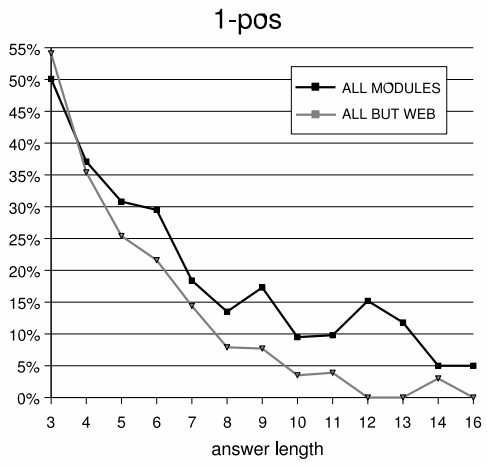

Figure 3: **Target in first position.** The frequency of the target in first position in relation to its length with and without the WSM.

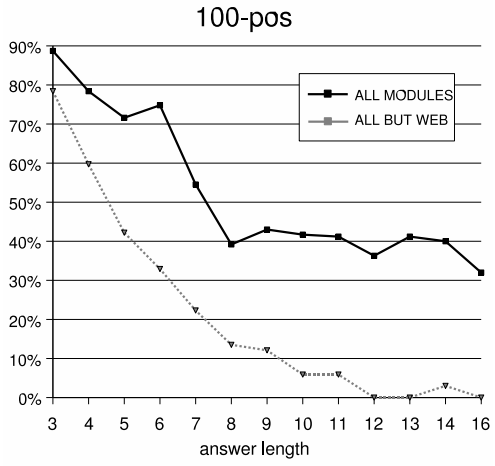

Figure 4: **Target in first 100 positions.** The frequency of the target in the first 100 positions in relation to its length with and without the WSM.

The inner architecture of the WSM is sketched in figure 2. There are four task that have to be accomplished by the WSM: the retrieval of useful web documents, the extraction of the answer candidates from these documents, the scoring/filtering of the candidate lists and, finally, the estimation of the list confidence. In this section all these components will be presented and analyzed. Despite the fact that the WSM has been implemented only in a basic version, it is clear that this module, among the set of expert modules used by WebCrow, produces the most impressive answering performances, with the best coverage/precision balance. This is evident if we observe tab. 3 (first two columns). In over half of cases the correct answer is found within the first 100 candidates inside a list containing more than 10^5 words.

The contribution of the WSM can be appreciated in the last three rows of tab. 3 where we can observe the loss of performance of the whole system when the WSM is removed. The overall coverage of the system is mainly guaranteed by the dictionary module (sec. 4.4), but the introduction of the WSM is fundamental to increase sensibly the rank of the correct answer. Also interesting is fig. 3 and fig. 4 where we take into consideration the length of the target. It can easily be observed that the WSM guarantees the system to well perform even with long word targets, which are of great importance in the CSP phase.

Module	**Coverage**	**1-pos**	**5-pos**	**100-pos**	**Length**
WEB (30 docs)	68.1	13.5	23.7	53.2	499
WEB (50 docs)	71.7	13.6	24.0	54.1	735
CWDB-EXACT	19.8	19.6	19.8	19.8	1.1
CWDB-PARTIAL	29.0	10.6	20.1	28.4	45.5
CWDB-DICTIO	71.1	0.4	2.1	21.5	$>10^3$
RULE-BASED	10.1	6.9	8.3	10.1	12.4
DICTIONARY	97.5	0.3	1.6	21.3	$>10^4$
ALL BUT WEB	98.4	34.0	43.6	52.3	$>10^4$
ALL (30 docs)	**99.3**	**36.5**	**50.4**	**72.1**	$\mathbf{>10^4}$
ALL (50 docs)	99.4	36.6	52.2	73.0	$>10^4$

Table 1: **Module's coverage.** It is reported the frequency with which the target word can be found within the candidate list. **1-pos** gives the freq. of the target in first position, **5-pos** the freq. within the first five candidates, **100-pos** within the first hundred. **Len** is the avg. list length. The number of documents used the WSM is reported in brackets. Test Set: 60 crosswords (3685 clues).

As it can be seen in table 2 the coverage of the WSM's lists grows sensibly with the first increments in the number of retrieved documents. This growth is imperceptible after 100 docs. We found that optimal balance in the trade off between precision, coverage and time cost is reached using 30 docs. We took this as the standard quantity of sources to be used in the experiments because it allows WebCrow to fulfill the time limit of 15 minutes.

Table 3 gives an insight of the quality of WSM's answers. Several examples of clues are reported along with a small portion of the correspondent candidate list.

3.1 Retrieving useful documents

The first goal of the answering process is to retrieve the documents that are better related to the clue. This can be done thanks to the fundamental contribution of search engine's technology (Google™ was used in our testing). In order to increase the informativeness of the search engine the clues go through a reformulation/expansion step. Each clue $C = \{t_1 t_2 ... t_n\}$ generates a maximum of 3 queries: $Q_1 =< t_1 \wedge t_2 \wedge ... t_n >$, $Q_2 =< t_1 \vee t_2 \vee ... t_n >$ and $Q_3 =< (t_1^1 \vee t_1^2 \vee ...) \wedge (t_2^1 \vee t_2^2 \vee ...) \wedge ... (t_n^1 \vee t_n^2 \vee ...) >$ where t_n^i is the i-th derivation (i.e. changing the number, the gender, ...) of term t_n. Q_3 has not been implemented yet. Non informative words are removed from the queries.

A classic QA approach is to make use only of the document snippets in order to stress time efficiency. Unfortunately the properties of the clues make this approach useless (the probability of finding the correct answer to a crossword clue within a snippet has been experimentally observed below 10%) and we decided for a full-document approach.

The interrogation of the search engine and the download the documents represent two tasks that are extremely time consuming, absorbing easily over 90% of time in the entire clue-answering process. Therefore we have implemented it in a highly parallel manner: the WSM simultaneously downloads tens of documents (for one or more clues at a time) adopting a strict time-out for each http request (20 secs.). If a request reaches the time-out then the WSM asks the search engine for a cached copy of the document. If this is unavailable then the document is declared missed and an additional link is requested to the SE.

For each example of our test suite we have produced a full retrieval session with a maximum of 200 docs per clue (max. 30 docs with Q_2). 615589 docs were downloaded in 44h 36min (bandwidth: 1Mb/s, effective ≈100KB/sec, avg. 230 docs/min, 167 docs/clue, 25.6KB/doc). All the test sessions were subsequently made offline exploiting this *web image*.

3.2 Extracting and ranking the candidates

The process of generating a list of candidate answers given a collection of relevant documents goes through two important steps. First, the documents are analyzed by a parser which produces as output plain ASCII text[10]. Second, this text is passed to a list generator that extracts the words of the correct length, eliminates doubles and produces an

[10]Currently, the parser handles only HTML scripts. We are planning to implement a PDF parser in the next future.

#docs + filters	**Cover**	**1-pos**	**5-pos**	**100-pos**	**Time**
5+SF	46.4	11.1	19.1	41.7	1:25
10+SF	56.2	12.2	21.6	47.5	2:45
20+SF	63.7	12.3	22.1	50.3	5:30
30+SF	67.9	12.3	22.2	52.3	8:10
50+SF	71.6	12.2	22.0	53.4	13:30
100+SF	74.5	11.9	21.5	53.2	26:50
30+SF+MF	**68.1***	**13.5**	**23.7**	**53.2**	**8:45**
50+SF+MF	71.7*	13.6	24.0	54.1	14:15

Table 2: **WSM's performance.** The performances of the WSM are here reported. The number of documents used is reported in brackets. SF=statistical filter, MF=morphological filter. Time is reported in min:secs. *The growth of the coverage is due the NI submodule.

"Easy" clues
≺*Confina con l'Abruzzo:* **molise**≻ 1:*molise* 2:aquila 3:marche 4:umbria
≺*Il von Klein scrittore:* **heinrich**≻ 1:*heinrich* 2:giovanni 3:kohlhaas
≺*Atomi elettrizzati:* **ioni**≻ 1:*ioni* 2:poli 3:essi 4:sali 5: rame
≺*Mal d'orecchi:* **otite**≻ 1:*otite* 2:ictus 3:otiti 4:edemi 5:gocce
≺*Lo parlano anche in Austria:* **tedesco**≻ 1:inglese 2:*tedesco* 3:milione 4:skiroll
≺*Un film di Nanni Moretti:* **carodiario**≻ 1:palombella 2:portaborse 3:*carodiario*
≺*Il piú famoso dei Keaton:* **buster**≻ 1:comico 2:cinema 3:grande 4:*buster*
≺*Il Giuseppe pittore di Barletta:* **denittis**≻ 1:leontine 2:molfetta 3:ritratto 4:*denittis*

"Tough" clues
≺*Caratteristica del burlone:* **giocosita**≻ 1:simpatica 2:sicurezza 3:compagnia
≺*Documenti per minorenni:* **patentini**≻ 1:necessari 2:richiesti 3:organismi
≺*Il verbo di chi ha coraggio:* **lanciarsi**≻ 1:interiore 2:predicato 3:idealismo
≺*Lasciare un segno:* **intaccare**≻ 1:passaggio 2:possibile 3:segnalare
≺*Non ha gusto in bocca:* **insapore**≻ 1:dialetto 2:prodotto 3:zucchero
≺*Larga e comoda:* **ampia**≻ 1:bella 2:sella 3:barca 4:scala 5:valle
≺*Sembrano ridere:* **iene**≻ 1:anni 2:loro 3:rane 4:rami 5:voci
≺*Una sciagura attraente:* **calamita**≻ 1:passione 2:alcolico 3:fardello

Table 3: **Some examples.** On the left: some examples of clues that are "easy" to answer for the WSM. The correct answer is present in the very first candidates. The easiest examples are usually the clues where the topic is directly addressed and where the answer is a factoid. On the write there is a list of "tough" clues. These are typically very general or ambiguous and the WSM fails to place the correct answer at the head of the list. Nevertheless, all the answers that the system produces tend to be semantically related to the target and to the clue.

unweighted candidate list. In order to increase the coverage, a list of compound words (i.e., a sequence of adjacent words fulfilling the length requirement) is generated from each document. To avoid noisy information, compound words which occurs only once are omitted.

Both outputs are then passed to two submodules: a statistical filter, based on IR techniques, and a morphological filter, based on machine learning and NLP techniques. Both have been embedded in the WSM.

The candidates are ranked by merging together the information provided by the two list filters. The score-probability associated to each word candidate w is given by

$$p(w, C) = c\,(sf\text{-}score(w, C) \times mf\text{-}score(w, C)) \tag{1}$$

where $sf\text{-}score(w, C)$ is the score attributed to word w by the statistical filter, $mf\text{-}score(w, C)$ is the score provided by the morphological filter, c is the normalizing factor that fulfills the probability requirement $\sum_{i=0}^{n} p(w_n, C) = 1$.

In QA systems it is important to produce very high precision only in the very first (3-5) answer candidates, since a human user will not look further down in the list. For this reason NLP techniques are typically used to remove those answers that are not likely correct. This answer selection policy is not well suited for clue-answering, a more conservative approach is required because the lack of the correct answer makes a greater damage than a low precision. The eq. 1 serves this goal: words that have low scores will appear at the bottom of the list but will not be dropped.

Our future objective is to implement a full battery of filters that can be added to the two already implement: stylistic, morpho-syntactical, lexical and logical. We believe that a robust NLP system could be of great impact in the answering of the clues. In addition to this we are designing a clue classifier that will enable WebCrow's module coordinator to understand when the web search can really be fruitful and when, conversely, this should not be triggered.

3.3 The Statistical Filtering

This submodule makes use of three types of information: a query (generated by the reformulation of a clue), a collection of ranked documents (parsed and cleaned) provided by the search engine and a list of candidate answers extracted from the documents. We represent this information with the triple (w, Q^n, D_i) where w is a word of the correct length,

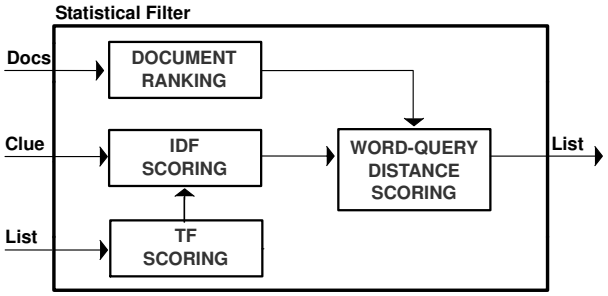

Figure 5: **Statistical Filter.** A sketch of the internal architecture of the Statistical Filter.

Q^n (n-th reformulation of clue C) is the query that is given as input to the SE and D_i is the i-th document (containing word w) provided as output by the SE. An additional element is used, $rank(D_i, Q^n)$: the document ranking. To obtain this score we use the position of D_i in the Google's output and then compute $\log(1/pos(D_i))$. It has to be noted that i does not strictly correspond to $pos(D_i)$ because whenever a document is missed for some reasons (non parsable format, http errors, etc.), the systems looks further down in the list in order to maintain constant the quantity of usable documents.

Finally, we attribute a global score to each word extracted from the documents in the following way:

$$sf\text{-}score(w, Q^n) = \sum_{i=0}^{\#\text{docs}} \left(\frac{score(w, Q^n, D_i)}{length(D_i)} rank(D_i, Q^n) \right) \tag{2}$$

where $length(D_i)$ is the number of words in D_i. The score of a word within a single document is computed in a TF-IDF fashion. TF has been modified in order to take into account the inner-document distance between the word and the query. As shown in eq. 3, each occurrence of a word counts $1/dist(w, Q, D_i)$, whereas in normal TF each occurrence counts equally.

$$score(w, Q^n, D_i) = idf(w) \sum_{w_k \in occ(w, D_i)} \frac{1}{dist(w_k, Q^n, D_i)} \tag{3}$$

$idf(w)$ is the classic inverse document frequency, which provides an immediate interpretation of term specificity. For compound words we take the highest idf value of the word components. $occ(w, D_i)$ represents the set of all the occurrences of the word w in the document D_i. The distance between the word w_k and query Q^n is computed as a modified version of the square-root-mean distance between w_k and each term w_{t,Q^n} of the query, suggested by [10]. The main bias of the original formula was to weight equally all the words of the query without taking into account that some words are more informative than others. As shown in eq. 4, we decided to overcome this problem by tuning the exponential factor of the square-root-mean distance using the idf value of w_{t,Q^n} (normalized between 1 and 3). This increases the relevance of those answer candidates that are close to the more informative terms in the query. This novel contribution has resulted experimentally more effective for our goals.

$$dist(w_k, Q, D_i) = \frac{\sqrt{\sum_{t=0}^{\#\text{terms} \in \text{Q}} (dist(w_k, w_{t,Q^n}, D_i))^{idf(w_{t,Q^n})}}}{\#\text{terms} \in \text{Q}} \tag{4}$$

$dist(w_k, w_{t,Q^n}, D_i)$ denotes the distance between the word w_k and word w_{t,Q^n} in document D_i. In our implementation the distance between two words, within a single document, is given by the minimum number of words that separate them. After a preliminary testing we decided to limit to 150 words the maximum word-word distance. A default distance of 300 is assigned to those words that exceed this limit. It is legitimate to believe that outside a certain window the semantic link between two words is unpredictable.

This distance metric could be further improved (i.e. taking into account sentences, paragraphs, titles, punctuations, etc.) but it already provides a very informative tool.

Other improvements could be obtained using a crossword-focused idf function (the idf values used here were obtained through a non-focused crawling session) or making use of the context in which each candidate appears.

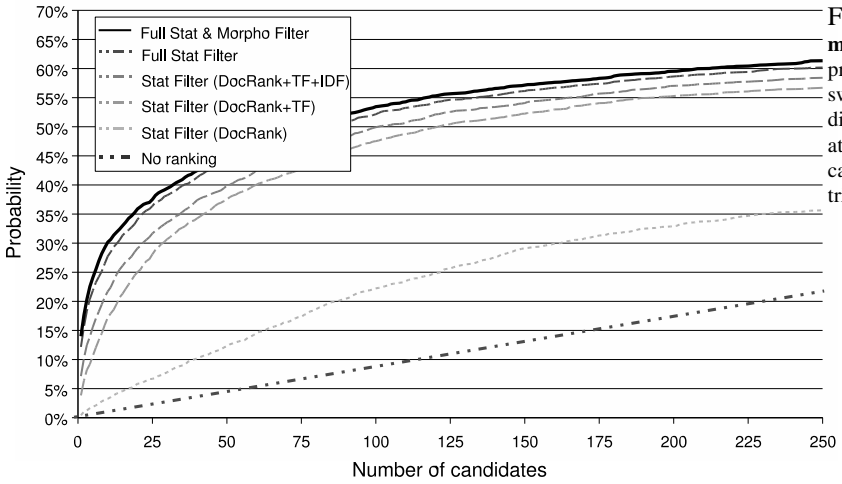

Figure 6: **Filtering performances.** The graphic represents the probability of finding the correct answer in relation to the number of candidates that are taken into consideration. Looking further down in the candidate list, the probability of retrieving the target answer increases.

Figure 6 shows the contribution of all the elements used within the statistical filter. In a non ranked list the probability of finding the correct answer increases linearly with the number of candidates taken into consideration. If we rank the candidates for their TF value the probability increases for those words that are better placed in the list. It is easy to observe in figure 6 how the performances increase shifting from a basic filter to the full one which includes both the statistical and morphological information.

3.4 The Morphological Filtering

The aim of this filter is to rank the candidates according to the morphological class they belong to. For this reason we made use of a Part-of-Speech (PoS) tagger, which associates a morphological class to each word of a sentence. Figure 7 shows the information flow of the morphological filter. The PoS tagger is used to tagged both the clue and each document related to it. Afterwords, the clue is processed by a multiclass classifier, which returns a weighted vector of the possible morphological classes the solution can belong to. Finally, for each word of the candidate list its morphological score is calculated by:

$$mf\text{-}score(w, C) = \sum_{i=0}^{\#\text{tags}} p(tag_i|w) score(tag_i, C) \tag{5}$$

$p(tag_i|w)$ is the information provided by the PoS-Tagger, $score(tag_i, C)$ is computed using the output of the classifier with the addition of a spread factor in order to enhance the impact of the classification.

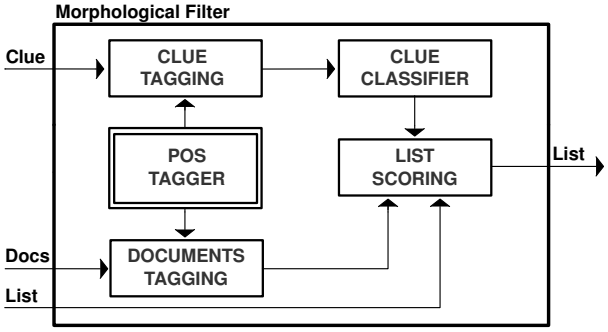

Figure 7: **Morphological Filter.** A sketch of the internal architecture of the Morphological Filter.

class	description
MS	Noun or Adj. or Pron., masc. sing.
FS	Noun or Adj. or Pron., fem. sing.
MP	Noun or Adj. or Pron., masc. pl.
FP	Noun or Adj. or Pron., fem. pl.
NP	Proper Noun
VS	Verb, cong. singular
VP	Verb, cong. plural
VI	Verb, base form
VOTHER	Verb, other
AMS	Article, masculine singular
AFS	Article, feminine singular
AMP	Article, masculine plural

class	description
AFP	Article, feminine plural
AV	Adverb
PART	Particle
NUM	Number
EP	Interlocutory words
ABBR	Abbreviation
PC	Compound Words
SCRIPT	Script words in html doc.
SENT2	Punctuation, all the others
SENT	Punctuation a the end of a sentence
OTHER	all the rest

Table 4: **Morphological classes.** This is the full list of the morphological classes used in our PoS Tagger. It differs from usual PoS tagging lists as the choice was to stress information relevant for finding the solution of a clue.

With the attempt to maintain a strong language-independence we chose an automatic trainable PoS tagger, called TreeTagger [14] [15], which is an extension of a basic Markov Model tagger. The TreeTager is based on two parts: a Lexicon and a Decision tree. Each word is first tagged using the Lexicon, which makes use also of a Prefix tree and a Suffix tree. This two trees are binary decision trees, generated by the training examples, which infer the possible tag of a word by examining, respectively, its beginning or ending. Finally, a binary decision tree is used. This takes into account the tags of the k preceding words and returns a vector of the probable tags, based on the examples seen in the training corpus. We used 23 different classes to distinguish: articles, nouns and adjectives, verbs, adverbs, particles, interlocutory words, numbers, punctuation marks, abbreviations and others. A detailed list is given in table 3.4. At first, the TreeTagger was trained using an automatically extracted corpus form TUT [16]. The tagger was then used to tag a new corpus based on some CWDB's clues and documents from the web. This new corpus was corrected and added to the first one. Finally, the TreeTagger was retrained, obtaining an accuracy of about 93% on a cross validation test set.

The clue classifier was built using multiclass Kernel-based Vector Machine [13] [12]. First, a training set was created by extracting about 7000 clue-target pairs from the CWDB. Each clue was tagged by the TreeTagger and a feature vector $\bar{x} \in \mathbb{R}^n$ was then automatically generated for each example. The features extracted from each clue-answer pair were: the length of the target, the number of words in the clue, the number of capital letters in the clue, a set of the 250 most frequent clue-words and the probability tag vector associated to each word of the clue. Finally, a target class $i \in \{1, \ldots, k\}$ was associated to each example. We made use of 21 different target classes: almost all the morphological ones with the addition of name initials (IP) and non-semantic words (NS). A detailed list is shown in table 6.

Our classifier is based on a multiclass Kernel-based Vector Machine, whose aim is to learn a linear function H : X $\rightarrow$ Y of the type $H(\bar{x}, M) = \langle M, \Phi(\bar{x}) \rangle$, where the predicted class is given by the function

$$f(\bar{x}) = \underset{i \in \{1,\ldots,k\}}{\text{argmax}} \; H_i(\bar{x}, M) \tag{6}$$

$H_i(\bar{x}, M) = y_i$ is the i-th entry of the vector $\bar{y} = H(\bar{x}, M)$, corresponding to the score given to the class i. The goal is to minimize the empirical risk over all the training examples

$$\mathcal{R}(f) = \sum_t \Delta(y_t, f(\bar{x}_t)) \tag{7}$$

where $\Delta(y_t, \hat{y_t})$ is the loss associated to the predicted class $\hat{y_t} = f(\bar{x}_t)$. $\Delta(y_t, \hat{y_t}) = 0$ if $y_t = \hat{y_t}$. Instead, $\Delta(y_t, \hat{y_t}) = pos\_loss + c \sum_{j:(y_j - y_t)>0} (y_j - y_t)$ if $y_t \neq \hat{y_t}$, where pos_loss is the distance in positions of y_t from the first value $\hat{y_t}$ and c is a normalization parameter.

Using a cross validation test over the training set described above, we obtain with a linear kernel an accuracy of 54,30% on the predicted class. The accuracy is not very high as there are many clues where it is hard, also for humans, to determine the exact class of the solution. This ambiguity occurs mainly between the classes of these two subset: {MS,FS,NP} and {MP,FP} [11]. For the latter reason and taking into account that no candidate is pruned but just re-weighted, we considered as a more significant value the coverage of the classifier on the first n predicted classes. As shown in table 5, the coverage increases very rapidly and it is equivalent to 91,38% if we look over the first 5 predicted classes. Thus, as the number of different target classes is large, this can be considered a very good result. In fact, the use of the output of the clue classifier causes an increment in the WSM performance.

Table 6 shows the occurrence of each class in the data set, which should be similar to the one in the whole CWDB. No re-balancing has been made, as the learning algorithm, during each loop, process the "most violated" constraint using a cutting plane method. It can be seen also that there are several classes whose accuracy is high, such as IP, NS, VI, NP and MP.

Moreover, the two non-morphological classes (IP and NS) were introduced in order to better exploit the morphological classifier. A submodule (NI) was implemented which generates name initials [12] when two subsequent proper nouns are found in a sentence. The NS class, instead, is associated to all those clues where the solution does not generally belong to the dictionary, but it can be inferred from the clue itself [13]. At this moment, this type of clues is mainly covered by the rule-based module. In future, a specific module will be implemented which will generate appropriate solutions in a more machine learning fashion. This means by inferring likely solutions from previously seen examples.

3.5 Estimating a confidence on the lists

After generating a candidate, each module has to estimate the probability that this list contains the correct answer. This information is then processed by the merger, in order to correctly join the lists produced by the modules.

The confidence estimator of the Web Search Module has been implemented using a standard MLP neural network. This was trained on a set of 2000 candidate lists, using a cross validation set of 500 examples. The main features used for the description of a candidate list example include: the length of the query, the idf values of its words, the length of the list and the scores of the candidates. The output target was set to 1 when the list contained the correct answer, 0 when this was absent. At the end of the training the estimator produced on the validation set an average square error of 0,08.

[11] For example, in some clues is not possible to determine the gender of the solution, such as ≺*Ricopre i vialetti:* **ghiaia, FS**≻ (≺*It can cover a drive:* **gravel**≻) or ≺*Si cambiano ad ogni portata:* **piatti, MP**≻ (≺*You use different ones at each course:* **plates**≻).

[12] E.g., ≺*Iniziali di Celentano:* **ac, IP**≻ (≺*Name initials of Celentano:* **ac**≻). The celebrity we are talking about is Adriano Celentano, with name initials A.C.

[13] E.g., ≺*Trasformano la forza in norma:* **nm, NS**≻ (≺*they change the force in norm:* **nm**≻, it should be read ≺*they change the word 'forza' in 'norma':* **nm**≻).

Position	Coverage
1st pos	54.30%
2nd pos	73.01%
3rd pos	82.67%
4th pos	87.77%
5th pos	91.38%
6th pos	93.60%
7th pos	95.01%
8th pos	96.16%
9th pos	96.91%
10th pos	97.20%

Table 5: **Coverage.** Here is reported the probability of finding the correct answer in the first k positions.

class	P of ex.	acc.
MS	24.82%	50.23%
NP	18.68%	68.17%
FS	13.68%	32.36%
MP	11.17%	65.12%
NS	9.04%	84.64%
FP	5.18%	18.99%
ABBR	3.67%	67.18%
IP	2.86%	92.16%
VI	2.64%	67.39%
PC	2.33%	34.62%
PART	1.28%	25.45%

class	P of ex.	acc.
AV	1.22%	16.28%
EP	1.01%	5.00%
OTHER	0.89%	12.50%
NUM	0.81%	38.71%
AMS	0.36%	21.43%
VS	0.16%	0.00%
AMP	0.10%	20.00%
AFP	0.06%	33.33%
AFS	0.03%	33.33%
VP	0.01%	0.00%

Table 6: **Class accuracy.** For each class it is given the percentage of examples inside the training set and the accuracy of the classifier.

4 The other modules

4.1 The data-base modules

Three different DB-based modules have been implemented in order to exploit the 42973 clue-answer pairs provided by our crossword database. As a useful comparison, the CWDB used by Proverb contained around 3.5×10^5 clue-answer pairs.

CWDB-EXACT simply checks for an exact clue correspondence in the clue-entries. For each answer to a clue C the score-probability is computed using the number of occurrences in the record C. CWDB-PARTIAL employs MySQL's partial-match functions, query expansion and positional term distances to compute clue-similarity scores. The number of answer occurrences and the clue-similarity score are used to calculate the candidates probabilities. CWDB-DICTIO simply returns the full list of words with the correct length, using the number of total occurrences to rank the candidates. Finally, the confidence estimation of the CWDB lists is an entropy function based on the probabilities and occurrences of the candidates.

4.2 The rule-based module

Italian crosswords often contain a limited set of answers that have no semantic relation with their clues, but that are cryptically hidden inside the clue itself. This especially occurs in two-letter and three-letter-answers. Some of the cryptic jokes that crossword authors apply are more or less standard. The rule-based module (RBM) has been especially designed to handle these cases. We have defined eighteen rules for two-letter words and five rules for the three-letter case.

For example, with a clue like ≺*Ai confini del mondo:* **mo**≻ [14] the RBM works as follows: pattern → *ai confini*; object → *mondo*; rule → extract first and last letter from the object. Hence, answer → **mo**.

4.3 The implicit module

The goal of the implicit module is to give a score to sequences of letters. The implicit module is used in two ways: first, within the grid-filling algorithm, to guarantee that the slots that have no candidate words left during the solving process are filled with the most probable sequence of characters; second, as a list filter to rank the terms present in the dictionaries. To do so we used tetra-grams. The global score of a letter sequence results by the product of all the inner tetra-gram probabilities. Following a data-driven approach the tetra-gram probabilities were computed from the CWDB answers.

4.4 The dictionary module

Dictionaries will never contain all the possible answers, being crosswords open to neologisms, acronyms, proper names and colloquial expressions. Nevertheless these sources can help to increment the global coverage of the clue-answering.

Two Italian dictionaries were used. The first one containing 127738 word lemmas, and the second one containing 296971 word forms. The output of this module is given by the list of terms with the correct length, ranked by the implicit module.

5 Merging the candidate lists

The merger module has been implemented in a very straightforward way (a more sophisticated version will be required in the future). The final score of each term w is computed as: $p(w) = c\sum_{i=0}^{m}(p_i(w)\times \mathrm{conf}_i)$ where m is the number of modules used, conf_i is the confidence evaluation of module i, $p_i(w)$ is the probability score given by module i and c is a normalizing factor.

[14] ≺*At the edge of the world:* **wd**≻, **wd** is the fusion of the first and the last letter of object *world*.

6 Filling the crossword puzzle

As demonstrated by [3] crossword solving can be successfully formalized as a Probabilistic-CSP problem. In this framework the slots of the puzzle represent the set of variables, the lists of candidates provide the domain of legal values for the variables. The goal is to assign a word to each slot in order to maximize the similarity between the final configuration and the target (defined by the crossword designer). This similarity can be computed in various ways. We adopted the *maximum probability function*, described by the following equation.

$$\underset{\forall \text{sol:} v_1,\ldots v_n}{\text{argmax}} \prod_{i=1}^{n} p_{x_i}(v_i) \tag{8}$$

where $p_{x_i}(v_i)$ is the probability that the value v_i is assigned to the variable x_i in the target configuration.

This means that given all the possible legal solutions we search for the one that maximizes the probability of the entire configuration.

A more efficient metric has been proposed in [3], the *maximum expected overlap* function[15]. We will include this feature in our further work.

Finding the maximum probability solution is an NP-complete problem that can be faced using heuristic search techniques as A*. In our implementation the path cost function is the product of the probabilities of the already assigned variables and the heuristic function is the product of the best remaining values of the unassigned variable. Taking the negative log probability, as in eq. 9 and 10, we transform the grid filling into a minimization problem that can be attacked using the classic A* cost function $f(X) = g(X) + h(X)$. Given d the number of already assigned variables in X, q the number of unassigned variables, $\#D_j$ the number of legal values for each unassigned variable x_j and v_j^k) the k-th legal value for x_j, we have the following:

$$g(X) = \sum_{i=1}^{d} -\log(p_{x_i}(v_i)) \tag{9}$$

$$h(X) = \sum_{j=1}^{q} -\log(\underset{k=1}{\overset{\#D_j}{\text{argmax}}}(p_{x_j}(v_j^k)) \tag{10}$$

Due to the competition time restrictions and to the complexity of the problem the use of standard A* was discarded. For this reason we adopted as a solving algorithm a CSP version of WA* [6]. Our new cost function is given by:

$$f(X) = \gamma(d)(g(X) + wh(X)) \tag{11}$$

w is the weighting constant that makes A* more greedy, as in the classic definition of WA*, and $\gamma(d)$ represents an additional score, based on the number of assigned values d (the depth of the current node), that makes the algorithm more depth-first, which is preferable in a CSP framework. This depth score increases the speed of the grid-filling, but it also causes $f(X)$ to be non-admissible.

The grid-filling module works together with the implicit module in order to overcome the missing of a word within the candidates list. Whenever a variable x_i remains with no available values then a heuristic score is computed by taking the tetra-gram probability of the pattern present in x_i. The same technique is used when a slot is indirectly filled (by the insertion of a crossing word) with a term that is not present within the initial candidates list.

To produce a fast node consistency computation, whenever a variable is selected for expansion, we calculate the remaining legal words using the pointer technique proposed in [5].

7 Experimental results

[15] The aim here is to maximize the number of words that coincide with the target, and not the overall probability

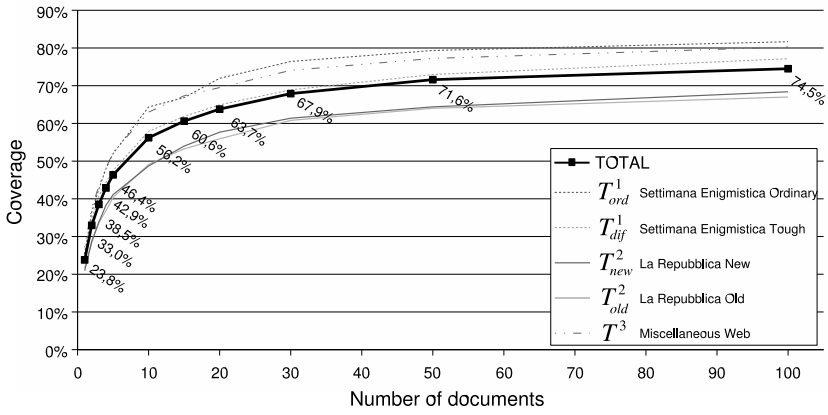

Figure 8: **The coverage of the WSM in relation to the number of documents used.** The WSM can increase its coverage by using more documents for each clue. This sensibly slows the answering process.

The whole crossword collection has been partitioned in five subsets. The first two belong to *La Settimana Enigmistica*, S^1_{ord} containing examples of ordinary difficulty (mainly taken from the cover pages of the magazine) and S^1_{dif} composed by crosswords especially designed for skilled cruciverbalists. An other couple belong to *La Repubblica*, S^2_{new} and S^2_{old} respectively containing crosswords that were published in 2004 and in 2001-2003. Finally, S^3 is a miscellaneous of examples from crossword-specialized web sites.

Sixty crosswords of the test set (3685 clues, avg. 61.4 each) were randomly extracted from these subsets in order to form the experimental test suite: T^1_{ord} (15 examples), T^1_{dif} (10 exs.), T^2_{new} (15 exs.), T^2_{old} (10 exs.) and T^3 (10 exs.). Some statistics about the test set are shown in table 7.

To securely maintain WebCrow within the 15 minutes time limit we decided to gather a maximum of 30 documents per clue. To download the documents, parse them and compute the statistical filtering an average of 8 minutes are required. An additional 35 secs are needed by the morphological filter. Thus, in less than 9 minutes WSM's work is completed. The other modules are much faster and the global list generation phase can be terminated in less than 10 minutes. To fulfill the competition requirements we limited the grid-filling execution time to 5 minutes. If a complete solution is not found within this time limit the best partial assignment is returned.

The results[16] that we obtained manifested the different difficulty inherent in the five subsets. Figure 8 reports WebCrow's performance on each example. On T^1_{ord} the results were quite impressive: the average number of targets in first position was just above 40% and the CSP module raised this to 80.0%, with 90.1% correct letters. In one occasion WebCrow perfectly completed the grid. With T^1_{dif} WebCrow was able to fill correctly 67.6% of the slots and 81.2% of the letters (98.6% in one case) which is more or less the result of a beginner human player. On T^2_{new} WebCrow performs with less accuracy averaging 62.9% (72% letters). On T^2_{old} (old crosswords), due to the constant refreshing of Web's information, the average number of correct

[16]WebCrow has been implemented mainly in Java with some parts in C++ and Perl. The system has been compiled and tested using Linux on a Pentium IV 2GHz with 2GB ram.

	T^1_{ord}	T^1_{dif}	T^2_{new}	T^2_{old}	T^3
# Letters	160.7	229.5	156.6	141.1	168.5
# Blanks	69.4	37.5	29.8	31.3	37.8
Clues	59.7	79.5	59.5	61.4	50.5
Avg. Length	4.99	5.53	5.04	4.96	4.61
Target in 1-pos	40.3%	37.3%	37.3%	33.3%	31.2%

Table 7: **Statistics of the test subsets.** T^1_{ord} is the subset of "easy" crosswords with short answers, high number of blanks, limited number of clues. T^1_{dif} provides a tough challenge, having a high number of clues and long answers. T^2_{new} and T^2_{old} are extremely difficult because they contain a great quantity of socio-political references. T^3 is a miscellaneous of average difficulty.

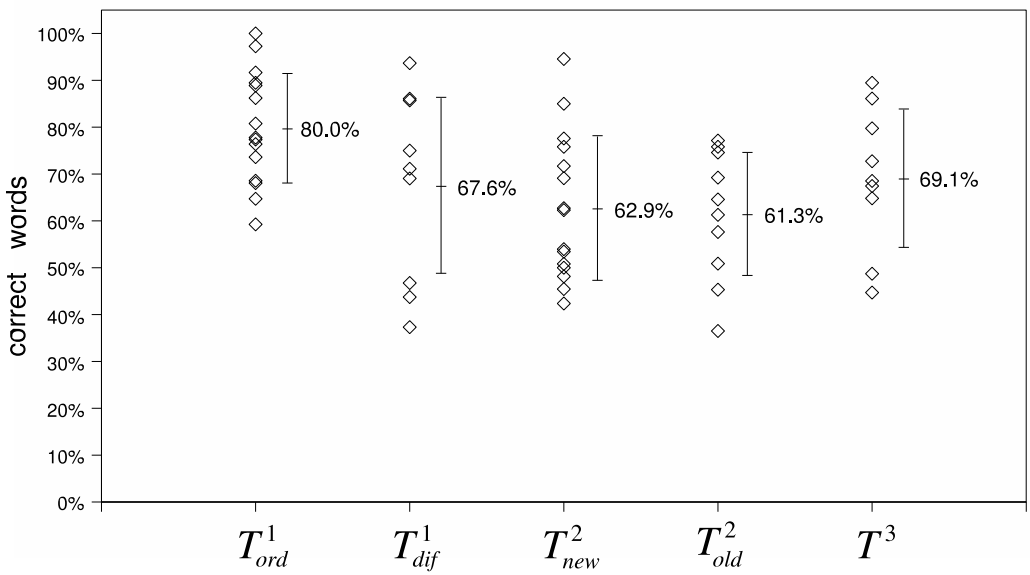

Figure 9: **WebCrow's performance on the five subsets.** The average and the variance of the correct words are also reported.

words goes down to 61.3% (72.9% letters). The last subset, T^3, contains crosswords that belong to completely different sources, for this reason the contribution of the CWDB is minimal (the coverage and the precision of CWDB-EXACT are more than halved). Nevertheless, the WSM still assures a good clue-answering and the solving module is able to reach 69.1% words correct and 82.1% letters correct.

Altogether, WebCrow's performance over the test set is of 68.8% (ranging from 36.5% to 100%) correct words and 79.9% (ranging from 48.7% to 100%)correct letters.

From preliminary tests we observed that allowing an extended time limit of 45 minutes and using more documents from the Web (i.e. 50 per clue) the system's performances increase by a 7% in average.

8 Conclusions

The version of WebCrow that is discussed here is basic but it has already given very promising results. WebCrow's overall architecture allows to plug in several expert modules in order to increase the system's performances. The web-search approach has proved to be very consistent. We believe it could suite all those problems in which semantics and interpretation play an important role

In our future work we believe that a robust NLP system could be of great impact in the answering of the clues. This can be done by adding several other list filters: stylistic, morpho-syntactical, lexical and logical. Moreover, we will improve the grid-filling algorithm. Thanks to its modular design and to the promising results of the current implementation we believe that WebCrow can become a consistent Italian and multilingual crossword solver.

9 Acknowledges

We are extremely thankful to the scientific magazine *Nature* that confirmed the significance of the topic and manifested a great interest on WebCrow [9]. We thank to Google^{TM} for its fundamental support and to Michael Littman that gave us precious suggestions. We also thank Guido Bartoli and Giovanni Canessa for their aid.

References

[1] Michael L. Littman, Greg A. Keim and Noam M. Shazeer: A probabilistic approach to solving crossword puzzles. Journal of Artificial Intelligence. 134 (2002) 23–55

[2] Greg A. Keim, Noam M. Shazeer and Michael L. Littman: PROVERB: the probabilistic cruciverbalist. AAAI '99: Proceedings of the sixteenth national conference on Artificial Intelligence. (1999) 710–717

[3] Noam M. Shazeer, Greg A. Keim and Michael L. Littman: Solving crosswords as probabilistic contraint satisfaction. AAAI '99: Proceedings of the sixteenth national conference on Artificial Intelligence. (1999) 156–152

[4] Michael L. Littman: Review: computer language games. Journal of Computer and Games. 134 (2000) 396–404

[5] M. L. Ginsberg, M. Frank, M. P. Halping and M.C. Torrance: Search lessons learned from crossword puzzles. Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI–90). (1990) 210–215

[6] I. Pohl: Heuristic search viewed as path finding in a graph. Journal of Artificial Intelligence. 1 (1970) 193–204

[7] L. J. Mazlack: Computer construction of crossword puzzles using precedence relationships. Journal of Artificial Intelligence 7 (1976) 1–19

[8] Matthew L. Ginsberg: Dynamic Backtracking. Journal of Artificial Intelligence Research. 1 (1993) 25–46

[9] News–Nature: Program crosses web to fill in puzzling words. Nature 431 (2004) 620

[10] Cody Kwok, Oren Etzioni and Daniel S. Weld: Scaling question answering to the web. ACM Trans. Inf. Syst. 19,3 (2001) 242–262

[11] Ellen M. Voorhees and Dawn M. Tice: Overview of the TREC–9 Question Answering Track. Proceedings of the Ninth Text REtrieval Conference (TREC–9). (2000)

[12] Koby Crammer and Yoram Singer: On the algorithmic implementation of multiclass kernel–based vector machines. Journal of Machine Learning Res. 2 (2002) 265—-292

[13] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims and Yasemin Altun: Support vector machine learning for interdependent and structured output spaces. ICML '04: Twenty–first international conference on Machine learning (2004)

[14] Helmut Schmid: Probabilistic Part–of–Speech Tagging Using Decision Trees. International Conference on New Methods in Language Processing. (1994)

[15] H. Schmid: Improvements in Part-of-speech Tagging with an Application to German. Proceedings of the EACL SIGDAT Workshop. (1995)

[16] C. Bosco, V. Lombardo and D. Vassallo and L. Lesmo: Building a Treebank for Italian: a Data–driven Annotation Schema. Proceedings of LREC. (2002)