Solving Italian Crosswords Using the Web

Giovanni Angelini, Marco Ernandes, and Marco Gori

Dipartimento di Ingegneria dell'Informazione,
Via Roma 56, 53100 Siena, ITALY,
[angelini|ernandes|marco]@dii.unisi.it,
WWW 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 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 (15% of correct answers in first position).
To complete the crosswords solving problem the system has to fill the grid with the best set of word answers. Currently, WebCrow's 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. With crosswords that are designed for experts, WebCrow places correctly two thirds of the words and around 80% of the letters.

1 Introduction

Motivations and Relevant Literature. Crosswords are puzzles that engage millions of people everyday in a very challenging game for human intelligence. This problem is reputed as AI-complete [4]. The complexity is due to its semantics and the large amount of encyclopedic knowledge required. AI developed an interest for crosswords 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 crosswords database of great dimensions[1].

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 making use of the Web as its primary source of knowledge, being this the most extremely rich and self-updating repository of human knowledge. With respect to Proverb WebCrow does not possess any knowledge-specific expert module, but only a limited set of useful modules which includes a dictionary and a small database[2].

[1] Before Proverb, AI limited its analysis to the *crossword puzzles 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 now in a few seconds.

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

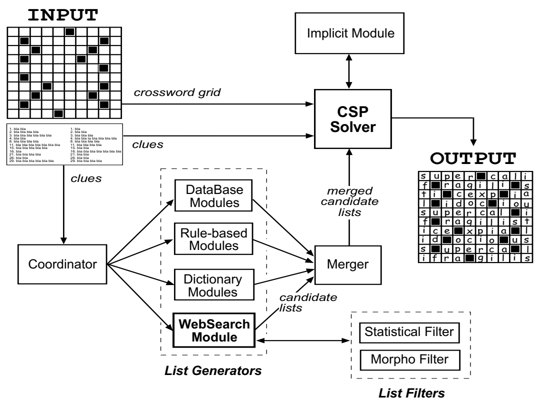

Fig. 1. WebCrow. A general overview of WebCrow's architecture.

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 can produce extremely effective results providing the most important source of knowledge for the clue-answering process.

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. Hence, the system requires the possession of a very broad and fresh knowledge that is also robust to volunteer language vagueness and ambiguity.

We have collected a dataset of 685 solved Italian crosswords. These examples were mainly obtained from two sources: the main Italian crosswords magazine *La Settimana Enigmistica* (this publisher sets, as matter of fact, a standard for Italian crosswords) and an important on-line newspaper's crosswords section, *La Repubblica*.

Given a test set of 60 crosswords, 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 and in nearly 15% this answer was 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, filling the puzzle WebCrow averaged on the overall test set around 70% words correct and 80% letters correct.

2 The System Architecture

WebCrow is a modular-based system (fig. 1). Therefore, it is also possible to plug in additional *ad hoc* modules in order to increase the system's performances.

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. Afterwards the candidate lists are merged into a unique list 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

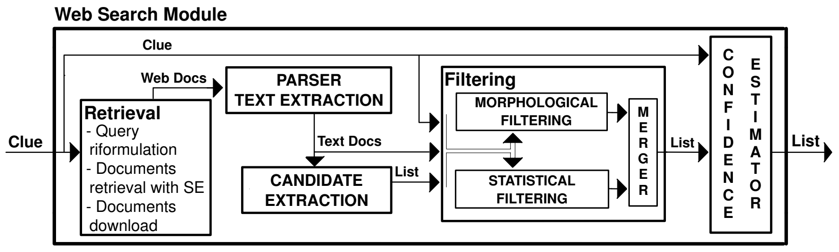

Fig. 2. Web Search Module. A sketch of the internal architecture of the Web Search Module.

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 [9]. The main differences are:

– clues are mostly formulated in a non-interrogative form
– clues can be voluntarily ambiguous and misleading
– the topic of the questions are not limited to factoids.
– there is a unique and precise answer which is a word or a compound word. Instead, in QA the answer is sequence of words in which the target has to be recognizable by humans.

The only evident advantage in crosswords solving is that we priory 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.

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 analysed.

Although 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

Table 1. Modules coverage. Cov reports the frequency with which the target word can be found within the candidate list. **n-pos** ($n = 1, 5, 100$) gives the frequency within the first n candidates. **Len** is the average list length. ALL(30 docs) was used in the final tests.

Module	**Cov**	**1-pos**	**5-pos**	**100-pos**	**Len**
WEB (30 docs)	68.1	13.5	23.7	53.2	499
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**	$\mathbf{>10^4}$

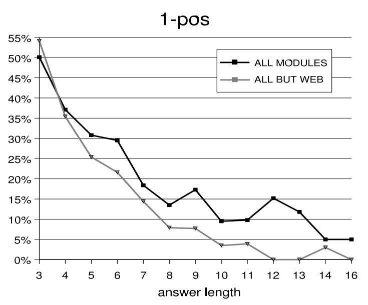

Fig. 3. Target in first position. The frequency of the target in first position in relation to its length with and without the WSM.

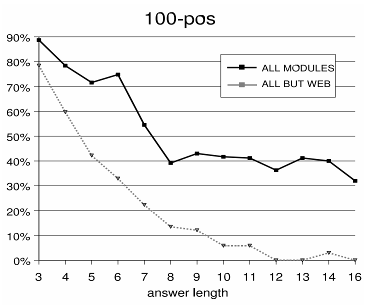

Fig. 4. Target in first 100 positions. The frequency of the target in the first 100 positions in relation to its length with and without the WSM.

impressive answering performances, with the best coverage/precision balance. This is evident if we observe tab. 1 (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 two rows of tab. 1 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), 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 be observed that the WSM guarantees the system to well perform with long word targets, of great importance in the CSP phase.

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$^{\text{TM}}$ was used in our testing). In order to increase the information retrieved through the search engine the clues go through a reformulation/expansion step. Each clue $C = \{t_1 t_2 ... t_n\}$ generates two queries: $Q^1 = < t_1 \wedge t_2 \wedge ... t_n >$ and $Q^2 = < t_1 \vee t_2 \vee ... t_n >$. Non informative words are removed from the queries.

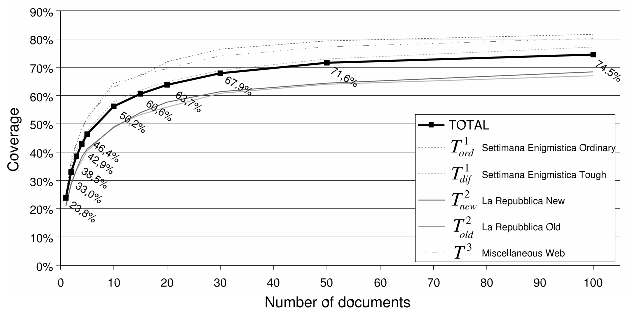

Fig. 5. WSM coverage. The coverage of the WSM in relation to the number of documents retrieved for each clue. The coverage increases very rapidly until about 30 documents. After this limit, in order to increse the coverage we have to sensibly slow down the answering process.

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 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 adopting a strict timeout for each http request (20 secs.) and using SE's cached copies when necessary.

For each example of our test suite we have retrieved and downloaded a maximum of 200 docs per clue (max. 30 docs with Q^2). 615589 docs were downloaded in 44h 36m[3]. All the test sessions were subsequently made off-line using 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 analysed by a parser which produces as output plain ASCII text. Second, this text is passed to a list generator that extracts the words of the correct length, eliminates doubles and produces an 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 (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 $\sum_{i=0}^{n} p(w_i,C) = 1$.

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

[3] Bandwidth: 1Mb/s, effective ≈100KB/sec, avg. 230 docs/min, 167 docs/clue, 25.6KB/doc).

Table 2. Clue-Answering samples. The "easy" examples are usually those where the topic is directly addressed. Instead, the "tough" ones are usually very general or ambiguous.

"Easy" clues for the WSM
≺*Confina con l'Abruzzo:* **molise**≻ 1:*molise* 2:aquila 3:marche 4:umbria
≺*Mal d'orecchi:* **otite**≻ 1:*otite* 2:ictus 3:otiti 4:edemi 5:gocce
≺*Un film di Nanni Moretti:* **carodiario**≻ 1:palombella 2:portaborse 3:*carodiario*
≺*Il Giuseppe pittore di Barletta:* **denittis**≻ 1:leontine 2:molfetta 3:ritratto 4:*denittis*

"Tough" clues for the WSM
≺*Documenti per minorenni:* **patentini**≻ 1:necessari 2:richiesti 3:organismi
≺*Il verbo di chi ha coraggio:* **lanciarsi**≻ 1:interiore 2:predicato 3:idealismo
≺*Larga e comoda:* **ampia**≻ 1:bella 2:sella 3:barca 4:scala 5:valle
≺*Una sciagura attraente:* **calamita**≻ 1:passione 2:alcolico 3:fardello

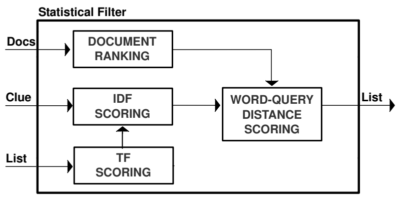

Fig. 6. Statistical Filter. A sketch of the internal architecture of the Statistical Filter.

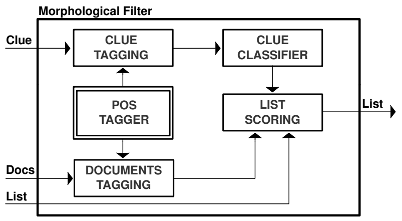

Fig. 7. Morphological Filter. A sketch of the internal architecture of the Morphological Filter.

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.

3.3 The Statistical Filtering

This submodule makes use of three types of information: the query given as input to the SE (Q^n: the n-th reformulation of clue C), the documents provided by the search engine (D_i is the i-th document of the SE's output) and a list of candidate answers w extracted from the documents. An additional element is used, $rank(D_i, Q^n)$: the document ranking, obtained using Google's output. Finally, we attribute a global score to each triple (w, Q^n, D_i) 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 $1/dist(w_k, Q, D_i)$, whereas in normal TF each occurrence counts equally.

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

$idf(w)$ 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 all the occurrences of the word w in the document D_i. The distance between word w_k and query Q is computed as a modified version of the square-root-mean distance between w_k and each term w_{Q_t} of the query, suggested by [8]. 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 a normalized idf value of w_{Q_t}. 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.

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

$dist(w_k, w_{Q_t}, D_i)$ denotes the minimum number of words that separate w_k and an occurance of the clue word w_{Q_t} in document D_i. After a preliminary testing we decided

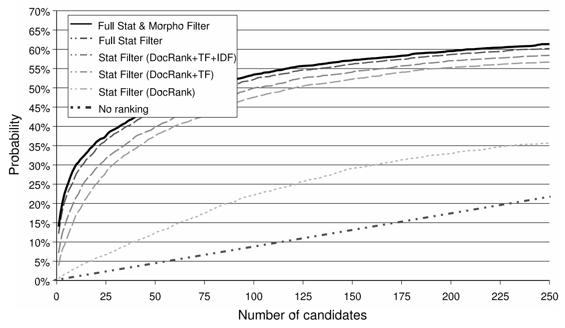

Fig. 8. Filtering perfomaces. The graphic represents the probability of finding the correct answer in relation to the number of candidates that are taken into consideration.

to limit to 150 words the maximum word-word distance. A default penalty distance of 300 is assigned to those words that exceed this limit, as we assume that the semantic link between two words is weaker.

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 8 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. It is easy to observe in figure 8 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. Afterwards, the clue is processed by a multi-class 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.

With the attempt to maintain a strong language-independence we chose an automatic trainable PoS tagger, called TreeTagger [12], 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. At first, the TreeTagger was trained using an automatically extracted corpus form TUT [13]. 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 multi-class Kernel-based Vector Machine [11] [10]. 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 5.

The classifier learns a linear function $H : \mathrm{X} \rightarrow \mathrm{Y}$ of the type $H(\bar{x}, M) = \langle M, \Phi(\bar{x}) \rangle$, where

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

is the predicted class and the i-th entry of the vector $\bar{y} = H(\bar{x}, M)$ corresponds to the score given to the class i. The goal is to minimize the empirical risk over all the training examples $\mathcal{R}(f) = \sum_t \Delta(y_t, f(\bar{x}_t))$ 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.

Table 3. Morphological classes. This is the full list of the morphological classes used in our PoS Tagger. The choice was to stress information relevant for finding the solution of a clue.

class	description	class	description
MS	Noun or Adj. or Pron., masc. sing.	AFP	Article, feminine plural
FS	Noun or Adj. or Pron., fem. sing.	AV	Adverb
MP	Noun or Adj. or Pron., masc. pl.	PART	Particle
FP	Noun or Adj. or Pron., fem. pl.	NUM	Number
NP	Proper Noun	EP	Interlocutory words
VS	Verb, cong. singular	ABBR	Abbreviation
VP	Verb, cong. plural	PC	Compound Words
VI	Verb, base form	SCRIPT	Script words in html doc.
VOTHER	Verb, other	SENT	Punctuation a the end of a sentence
AMS	Article, masculine singular	SENT2	Punctuation, all the others
AFS	Article, feminine singular	OTHER	all the rest
AMP	Article, masculine plural		

Table 4. Coverage. Here is reported the probability of finding the correct answer in the first k positions.

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%

Table 5. Class accuracy. For each class it is given the percentage of examples inside the training set and the accuracy of the classifier.

class	P ex.	acc.	class	P ex.	acc.	class	P ex.	acc.
MS	24.8%	50.2%	IP	2.9%	92.2%	NUM	0.8%	38.7%
NP	18.7%	68.2%	VI	2.6%	67.4%	AMS	0.4%	21.4%
FS	13.7%	32.4%	PC	2.3%	34.6%	VS	0.2%	0.0%
MP	11.2%	65.1%	PART	1.3%	25.5%	AMP	0.1%	20.0%
NS	9.0%	84.6%	AV	1.2%	16.3%	AFP	0.1%	33.3%
FP	5.2%	19.0%	EP	1.0%	5.0%	AFS	0.1%	33.3%
ABBR	3.7%	67.2%	OTHR	0.9%	12.5%	VP	0.1%	0.0%

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} [4]. 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 4, 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 5 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.

In order to better exploit the morphological classifier, a submodule (NI) which generates name initials[5] was implemented.

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.

[4] 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**≻).

[5] E.g., ≺*Iniziali di Celentano:* **ac, IP**≻ (≺*Name initials of Celentano:* **ac**≻).

4 The Other Modules.

Four different typologies of additional modules are present in WebCrow's design, namely the data-base, the rule-based, the implicit and the dictionary module.

The Data-Base Module. Three different DB-based modules have been implemented in order to exploit the 42973 clue-answer pairs provided by our crosswords database: CWDB-EXACT, that checks for exact clue correspondences in the clue-entries, CWDB-PARTIAL, that checks for partial matches by computing clue-similarity scores, CWDB-DICTIO, that simply returns the full list of words with the correct length.

The Rule-Based Module. Italian crosswords frequently contain answers that have no semantic relation with their clues, like $\prec$*Ai confini del mondo:* **mo**$\succ$, but that are cryptically hidden inside the clue itself. This especially occurs in two-letter and three-letter-answers. With these clues the Web does not provide any help. Therefore we have implemented a rule-based module (RBM), containing eighteen rules for two-letter words and five rules for the three-letter case.

The Implicit Module. The implicit module attributes scores to sequences of letters. It is used in two ways. First, to help the grid-filling algorithm when there are no candidate words left for a certain slot during the solving process and as 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 probabilities that were computed from the CWDB.

The Dictionary Module. Dictionaries are used 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 is given by the list of terms with the correct length, ranked by the implicit module.

5 Merging the Lists and Filling the Puzzle

The grid-filling phase requires to have unique lists for each slot. Hence, the first step is to merge all the lists produced by the different modules into one. The merger module attributes a probability $p(w)$ to all the words w that appear in the collection of candidate lists: $p(w) = c\sum_{i=0}^{m}(p_i(w) \times \text{conf}_i)$ 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.

Crossword solving can be successfully formalized as a Probabilistic-CSP problem [3]. 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 crosswords designer). To compute this similarity we adopted the *maximum probability function*[6]. We search among the various solutions for the one that maximizes: $\prod_{i=1}^{n} p_{x_i}(v_i)$ 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.

Finding the maximum probability solution is an NP-complete problem that can be faced using heuristic search techniques as A*. Due to the time restrictions and to the complexity of the problem we chose as a solving algorithm a CSP version of WA* [6] with cost function: $f(X) = \gamma(d)(g(X) + wh(X))$ where w is the weighting constant

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

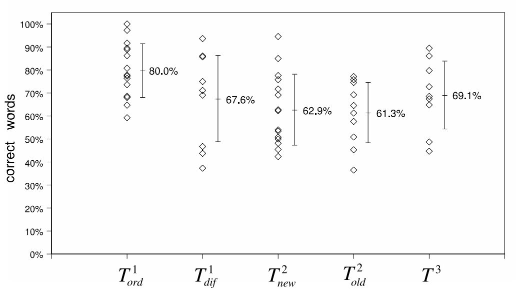

Fig. 9. WebCrow's performance on the five subsets. The performance over the full test set is of 68.8% correct words and 79.9% correct letters. Allowing an extended time limit of 45 minutes and using more documents, the system's performances increase by a 7% in average.

that makes A* more greedy, as in classic WA*, and $\gamma(d)$ represents an additional score, based on the number of assigned values d that makes the algorithm more depth-first, which is preferable in a CSP framework. This depth score speeds up the grid-filling but it also causes non-admissibility.

6 Experimental Results

The whole crosswords 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.).

The quality of the clue-answering provided by the Web Search Module can be observed in figure 5. By increasing the number of documents used, the coverage of the system can be augmented sensibly reaching 74,5% with 100 documents. The coverage is higher for the examples belonging to T^1_{ord} (up to 81,7%), T^1_{dif} (up to 77,4%) and T^3 (up to 80,6%). The clues from *La Repubblica* are objectively more difficult to answer, having a coverage of nearly 69%.

The coverage of the WSM's lists grows sensibly with the first increments in the number of retrieved documents. We found that an optimal balance in the trade off between precision, coverage and time cost is reached using 30 docs[7]. 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. If a complete solution is not found by the grid-filling algorithm within this time limit the best partial assignment is returned.

Figure 9 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% (90.1% correct letters). With T^1_{dif} WebCrow was

[7] In our testing it took an avg. of 8 minutes to answer all the clues of a crossword using 30 docs.

able to fill correctly 67.6% of the slots (81.2% letters). On T^2_{new} WebCrow performs with less accuracy averaging 62.9% (72% letters). On T^2_{old}, due to the constant refreshing of Web's information, the average number of correct words goes down to 61.3% (72.9% letters). In T^3 WebCrow reached 69.1% words correct (82.1% letters).

7 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. With these additions we are confident that WebCrow can become a strong Italian and multilingual crosswords solver.

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. Proc. AAAI '99. (1999) 710–717
3. Noam M. Shazeer, Greg A. Keim and Michael L. Littman: Solving crosswords as probabilistic contraint satisfaction. Proc. AAAI '99. (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. Proc. 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. Matthew L. Ginsberg: Dynamic Backtracking. Journal of Artificial Intelligence Research. **1** (1993) 25–46
8. Cody Kwok, Oren Etzioni and Daniel S. Weld: Scaling question answering to the web. ACM Trans. Inf. Syst. **19,3** (2001) 242–262
9. Ellen M. Voorhees and Dawn M. Tice: Overview of the TREC–9 Question Answering Track. Proc. TREC-9. (2000)
10. Koby Crammer and Yoram Singer: On the algorithmic implementation of multiclass kernel–based vector machines. Journal of Machine Learning Res. **2** (2002) 265—-292
11. Ioannis Tsochantaridis et al.: Support vector machine learning for interdependent and structured output spaces. Proc. ICML 04. (2004)
12. H. Schmid: Improvements in Part-of-speech Tagging with an Application to German. Proc. EACL SIGDAT Workshop. (1995)
13. C. Bosco, V. Lombardo and D. Vassallo and L. Lesmo: Building a Treebank for Italian: a Data–driven Annotation Schema. Pro. LREC. (2002)