WebCrow: a WEB-based system for CROssWord solving

Marco Ernandes, Giovanni Angelini and **Marco Gori**
Dipartimento di Ingegneria dell'Informazione,
Via Roma 56, 53100 Siena, ITALY,
[ernandes|angelini|marco]@dii.unisi.it,
WWW home page: http://airgroup.dii.unisi.it

Abstract

Language games represent one of the most fascinating challenges of research in artificial intelligence. In this paper we give an overview of WebCrow, a system that tackles crosswords using the Web as a knowledge base. This appears to be a novel approach with respect to the available literature. It is also the first solver for non-English crosswords and it has been designed to be potentially multilingual. Although WebCrow has been implemented only in a preliminary version, it already displays very interesting results reaching the performance of a human beginner: crosswords that are "easy" for expert humans are solved, within competition time limits, with 80% of correct words and over 90% of correct letters.

Introduction

Crosswords are probably the most played language puzzles worldwide. Until recently, the research in artificial intelligence focused its interest only on the NP-complete *crossword generation* problem (Mazlack 1976) (Ginsberg *et al.* 1990) which can now be solved in a few seconds for reasonable dimensions (Ginsberg 1993).

Conversely, problems like solving crosswords from clues have been defined as AI-complete (Littman 2000) and are extremely challenging for machines since there is no closed-world assumption and they require human-level knowledge.

Interestingly, AI has nowadays the opportunity to exploit a stable nucleus of technology, such as search engines, information retrieval and machine learning techniques, that can enable computers to enfold with semantics real-life concepts.

We will present here 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 an extremely rich and self-updating repository of human knowledge[1]. This paper gives a general overview of the modules and the subproblems present in the project.

[1]The WebCrow project, with the emphasis on the power coming from searching the Web, was briefly described in the Nature magazine(News-Nature 2004) and mentioned in the AAAI Web site concerning games and puzzles http://www.aaai.org/AITopics/html/crosswd.html

To the best of our knowledge the only significant attempt reported in the literature to tackle this problem is the Proverb system, which has reached human-like performances (Keim, Shazeer, & Littman 1999). Unlike Proverb, WebCrow does not possess any knowledge-specific expert module nor a crossword database of great dimensions. WebCrow, instead, in order to stress the generality of its knowledge and language-independence, has in its core a module performing a special form of question answering on the Web that we shall call clue-answering. The second goal of the system, that is filling the crossword grid with the best set of candidate answers, has been tackled as a Probabilistic-CSP.

Italian crosswords are structurally similar to Americans, but they differ in two aspects: two-letter words[2] are allowed and, more importantly, they contain stronger and more ambiguous references to socio-cultural and political topics. The latter is a phenomenon especially present in newspapers and it introduces an additional degree of complexity in crossword solving, since it requires the possession of a knowledge that is extremely broad, fresh and also robust to volunteer vagueness and ambiguity. Neologisms and compound words are often present.

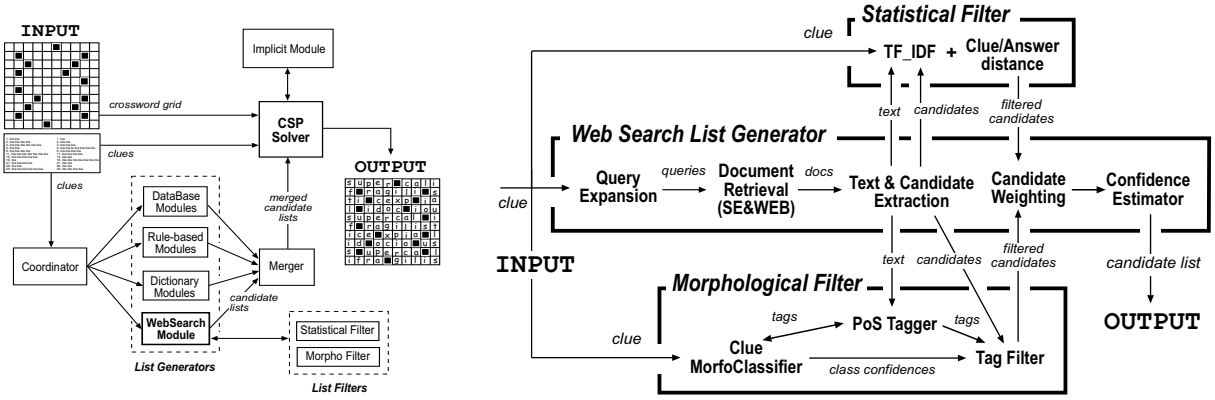
We have collected 685 examples of Italian crosswords. The majority was obtained from two sources: the main Italian crossword magazine *La Settimana Enigmistica* and the main on-line newspaper's crossword section *La Repubblica*. 60 puzzles were randomly chosen to form the experimental test suite. The remaining part constituted a database of clue-answer pairs used as an aid in the answering process.

In this paper the challenge of WebCrow is to solve Italian crosswords within a 15 minutes time limit (running the program on a common laptop), as in real competitions. The version of WebCrow that is discussed here is preliminary but it has already given very promising results. WebCrow averaged around 70% words correct and 80% letters correct in the overall test set. On the examples that expert players consider "easy" WebCrow performed with 80% words correct (100% in one case) and over 90% letters correct.

Architectural overview

WebCrow has been designed as a modular and data-driven system. Two are the main sources that it applies to: Web

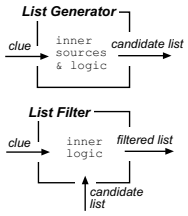
[2]Name initials, acronyms, abbreviations or portions of a word.



(A) General Overview **(B) The Web Search Module**

Figure 1: **(A)** A general overview of WebCrow's architecture (inspired by Proverb's). **(B)** A sketch of the internal architecture of the Web Search Module, with its three main submodules: the web-based list generator, the statistical and the morphological list filter.

documents and previously solved crosswords. It is also possible to plug in additional *ad hoc* information (i.e. word play rules) in order to increase the system's performances. We designed WebCrow's architecture (fig. 1A) using some insights from Proverb's. The information flow is similar: in a first phase the clues are analyzed by a set of candidate-list generators (the expert modules). Among these the preliminary role is played by the Web Search Module (WSM). Each list contains a global confidence score and a set of words ordered by probability. These candidate lists are passed to a merger system that produces a unique list for each clue. In a second phase, a constraint-satisfaction algorithm carries forward the grid-filling, in order to place the correct words within the slots.



In addition to the list generators we introduce here the concept of list filter. This sort of expert module takes as an input a clue and a candidate list, and returns a new candidate list filtered by some specific logic or knowledge. Filtering includes re-ranking or deletion of the candidates and re-weighting of the confidence score associated to the list.

At the current stage we implemented a statistical filter, based on IR techniques, and a morphological filter, based on machine learning and NLP techniques. These have been embedded in the WSM. Our future objective is to implement a full battery of filters: stylistic, morpho-syntactical, lexical and logical.

The web-search module (WSM)

The Web Search Module represents the heart of WebCrow and the main innovation with respect to the literature.

The goal of this important module recalls that of a Web-based Question Answering system. The idea is to exploit the Web and search engines (SE) in order to find sensible answers to questions expressed in natural language. However, with crossword clues, the nature of the problem changes sensibly, often becoming more challenging than classic QA[3]. The main differences are:

- clues are mostly formulated in a non-interrogative form (i.e: ≺*La voce narrante del Nome della Rosa:* **adso**≻[4]) making the task of determining the answer-type more subtle.
- clues can be voluntarily ambiguous and misleading (i.e: ≺*Quello liquido non attacca:* **scotch**≻[5])
- the topic of the questions can be both factoid[6] and non-factoid (i.e: ≺*Ci si va al buio:* **cinema**≻[7])
- there is a unique and precise correct answer: a single or a compound word (i.e. ≺*Vi si elegge il papa:* **cappellasistina**≻[8]), 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.

The inner architecture of the WSM is sketched in figure 1B. It is based on three main sub-modules: a web-based list generator that extracts possible answers to a clue from web documents by picking the terms (and compound words) of the correct length, a statistical filter that ranks the candidates using statistical information retrieval techniques and a morphological filter that ranks the words according to their morphological category. The confidence estimator at the back end of the module has been implemented by a multilayer neural network. It was trained on a set of 2000 candidate lists, to associate confidence scores to the lists.

[3] The main reference for standard QA is the TREC competition (Voorhees & Tice 2000).

[4] ≺*The background narrator of the Name of the Rose:* **adso**≻

[5] ≺*The liquid one does not stick:* **scotch**≻, the clue ambiguously refers to two different senses: scotch-whisky and scotch-tape.

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

[7] ≺*We go there in the darkness:* **cinema**≻

[8] ≺*Where the Pope is elected:* **cappellasistina**≻

Module	Cover	1-pos	5-pos	100-pos	Len
WSM	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 WSM	98.4	34.0	43.6	52.3	$>10^4$
ALL	99.3	36.5	50.4	72.1	$>10^4$

Table 1: **Cover** (coverage) reports 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 WSM used a maximum of 30 documents per clue. Test Set: 60 crosswords (3685 clues).

Although the current design and implementation of the WSM are only at an initial stage, this is already able to produce impressive answering performances. It is in fact the module that provides the best coverage/precision balance, as it can be observed in tab. 1. In over half of cases the correct answer can be found within the first 100 candidates within a list containing more than 10^4 words. The contribution of the WSM can be appreciated in the last two rows of tab. 1 where we 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, but the introduction of the WSM is fundamental to increase sensibly the rank of the correct answer. Interesting results are reported in fig. 2, where we take into account the length of the target. It can easily be observed that the WSM guarantees very good performances even with long word targets, which are of great importance in the CSP phase.

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

Seeking answer candidates

Retrieving the 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 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.

A classic QA approach is to make use only of the snippets in order to stress time efficiency. The properties of the clues sensibly reduce the probability of finding the answer within a snippet and we decided for a full-document approach.

The SE interrogation and the successive downloads are the most time consuming tasks of the clue-answering process. Therefore we implemented them in a highly parallel manner, similarly to web-crawlers, making tens of contemporary http requests with short time-outs.

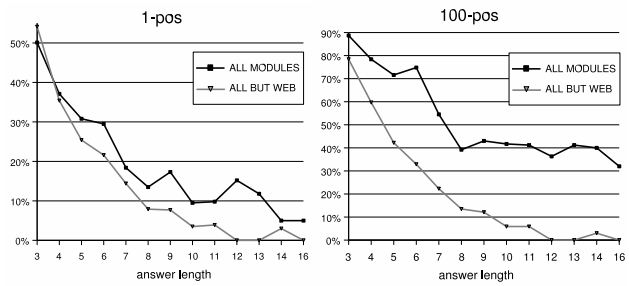


Figure 2: The frequency of the target in first position (left) and within the first 100 candidates (right) related to its word length. The black curve represents the overall performance of the system, the gray curve reports the results of the system without the WSM.

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*.

Extracting useful text The extraction of the text goes through two different steps. Primarily, the documents are processed by a parser that converts them to plain text. Secondarily this text is passed to a list generator that extracts the words of the correct length and produces an unweighted candidate list. Both outputs are then passed to the statistical and morphological list filter.

Ranking of the candidates The candidates are ranked exploiting the information provided by the two list filters. The score associated to each word candidate w is given by

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

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.

Statistical filtering

A score is associated to each triple (w, Q^n, D_i), where w is a word belonging to the document D_i given as output to the query Q^n (n-th reformulation of clue C). A rank $rank(D_i, Q^n)$ is given to each document provided by the

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

Table 2: The performances of the WSM. The number of documents used is reported in brackets. SF=statistical filter, MF=morphological filter. Time is reported in min:secs.

search engine (SE) using the query Q^n. Finally, the global score of a word extracted by the documents using the query Q^n is given by the weighted sum showed below:

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

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. Each occurrence of a word counts $1/dist(w, Q, D_i)$, whereas in normal TF each occurrence counts equally:

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

$occ(w) \in D_i$ represents all the occurrences of the word w in the document D_i. The distance between word w and query Q (eq. 4) is computed as the square-root-mean distance between w and each term w_{Q_t} of the query. To enhance the relevance of the words that are close to the more informative terms in the query we tune the exponential factor with the idf value of w_{Q_t} (normalized between 1 and 3).

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

$dist(w, w_{Q_t}, D_i)$ is given by the minimum number of words separating w and w_{Q_t} in document D_i. Eq. 4 is a slightly different version of the distance measure presented in (Kwok, Etzioni, & Weld 2001) that resulted experimentally more effective for our goals.

Morphological filtering

The module is based on two different parts: a part-of-speech (PoS) tagger and a clue-classifier. The first one associates a morphological class to each word of a sentence, while the second one determines, for each clue, the most probable morphological classes of the answer. During the filtering process each document is tagged using the PoS tagger and the clues are classified by the morpho-classifier. For each word of the candidate list the following morphological score is given:

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

$p(tag_i|w)$ 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 (Schmid 1995). We used 26 different classes to distinguish: articles, nouns and adjectives, verbs, adverbs, particles, interlocutory words, numbers and others.

The clue classifier was built using multi-class Kernel-based Vector Machine (Tsochantaridis *et al.* 2004) (Crammer & Singer 2002). A set of about 7000 clue-tag pair was extracted from the CWDB and tagged by the TreeTagger. Each example was then automatically mapped to a feature vector $\bar{x} \in \mathbb{R}^n$ and a class $i \in \{1, \ldots, k\}$ was associated to it. The features extracted from each clue-answer pair are: 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. We made use of 21 different target classes (almost all the morphological ones).

The multi-class Kernel-based Vector Machine learns a linear function $H{:}X{\rightarrow}Y$ of the type $H(\bar{x}, M){=}\langle M, \Phi(\bar{x})\rangle{=}\bar{y}$, where the predicted class is given by the function $f(\bar{x}){=}\text{argmax}_{i\in\{1,\ldots,k\}} H_i(\bar{x}, M)$. $H_i(\bar{x}, M){=}y_i$ is the i-th entry of the vector $\bar{y}{=}H(\bar{x}, M)$, corresponding to the score given to the class i.

Using a cross validation test over the training set described above, we obtain with a linear kernel an accuracy of 54.3% on the predicted class and a coverage of 91.38% on the first five predicted classes. If we consider the great number of different target classes this can be considered as a remarkable result, which turns out to increase sensibly the WSM's performance.

The other modules

The data-base modules (CWDB) 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 $3.5{\times}10^5$ 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 candidate 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.

The rule-based module (RBM) 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 occurs in two-letter and three-letter answers. The 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**≻[9] the RBM works as follows: pattern → *ai confini*; object → *mondo*; rule → extract first and last letter from the object. Hence, answer → **mo**.

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 of the dictionaries. To do so we used tetragrams. The global score of a letter sequence results by the product of all the inner tetragram probabilities. Following a data-driven approach the tetragram probabilities were computed from the CWDB answers.

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. We made use of two Italian dictionaries respectively containing word lemmas and word forms. The output of this module is given by the list of terms with the correct length, ranked by the implicit module.

The Merger The merger module has been implemented in a very straightforward way. The final score of each term w is computed as: $p(w) = c\sum_{i=0}^{m}(p_i(w) \times \text{conf}_i)$ 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.

The grid-filling

As demonstrated by (Shazeer, Keim, & Littman 1999) crossword solving can be successfully formalized as a Probabilistic-CSP problem. In this framework the slots of the puzzle represent the set of variables and 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*[10], which means that we search for the complete assignment 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 can be faced using heuristic search techniques as A*. 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* (Pohl 1970). The following cost function was used:

$$f(X) = \gamma(d_X)(g(X) + wh(X)) \quad (6)$$

where the path cost is $\sum_{i=1}^{d}$-log$(p_{x_i}(v_i))$, the heuristic estimation is $\sum_{j=1}^{q}$-log$(\text{argmax}_{k=1}^{\#D_j}(p_{x_j}(v_j^k))$ and w is the weighting constant that makes A* more greedy, as in the classic definition of WA*. $\gamma(d_X)$ represents an additional score, based on the number of assigned values d_X (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 candidate list. Whenever a variable x_i remains with no available values then a heuristic score is computed by taking the tetragram probability of the pattern present in x_i. The same technique is used when a slot is indirectly filled with a term that is not present within the initial candidate list.

	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
Answer 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 3: Statistics of the test subsets.

Experimental results

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 of subsets 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 (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 3. 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 (tab. 2). 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.

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

[10] A more efficient metric, the *maximum expected overlap* function , has been proposed in (Shazeer, Keim, & Littman 1999). We will include this feature in our further works.

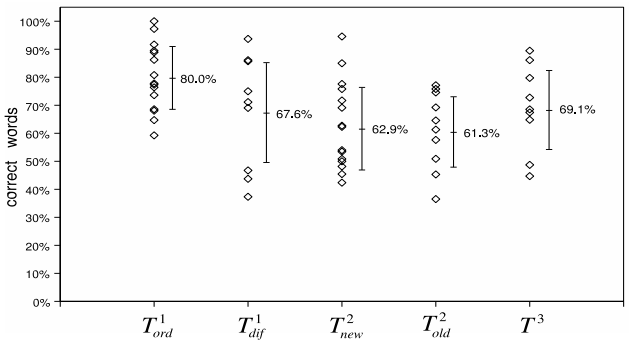


Figure 3: WebCrow's performance on the five subsets. The average and the variance of the correct words are also reported.

The results[11] that we obtained indicated the different difficulty inherent in the five subsets. Figure 3 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 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.

Conclusions

In this paper we have given an overview of WebCrow, a system conceived for solving crosswords. The main innovation consists of having shown, for the first time, the effectiveness of searching the Web to crack crosswords. WebCrow neither possesses knowledge-specific expert modules nor a big crossword database. Instead, it makes massive use of machine learning and IR techniques and, therefore, it is easily extensible to different languages and problems. The dynamic updating and growth of the Web makes possible to reach better performance on crosswords containing clues on recent events. The system discussed here is basic but it has given very promising results which can be easily improved. The WebCrow architecture is conceived in such a way that *ad hoc* modules can be plugged to increase the performance. We believe that for similar multilingual games, it can become a serious competitor even for human experts.

[11] 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.

Acknowledgments

We thank Google™ which makes the project possible and especially M. Diligenti for his crucial advises concerning the submission of automatic query sessions. We are indebted to M. L. Littman who gave us precious suggestions on the system's architecture and to G. Bartoli and G. Canessa for the preliminary implementation of the grid-filling module.

References

Crammer, K., and Singer, Y. 2002. On the algorithmic implementation of multiclass kernel-based vector machines. J. Mach. Learn. Res. 2:265–292.

Ginsberg, M. L.; Frank, M.; Halping, M. P.; and Torrance, M. 1990. Search lessons learned from crossword puzzles. In Proceeding of AAAI '90, 210–215.

Ginsberg, M. L. 1993. Dynamic backtracking. Journal of Artificial Intelligence Research 1:25–46.

Keim, G. A.; Shazeer, N. M.; and Littman, M. L. 1999. Proverb: the probabilistic cruciverbalist. In Proceeding of AAAI '99, 710–717.

Kwok, C.; Etzioni, O.; and Weld, D. S. 2001. Scaling question answering to the web. ACM Trans. Inf. Syst. 19(3):242–262.

Littman, M. L. 2000. Review: computer language games. Computer and Games 134:396–404.

Mazlack, L. J. 1976. Computer construction of crossword puzzles using precedence relationships. Artificial Intelligence 7:1–19.

News-Nature. 2004. Program crosses web to fill in puzzling words. Nature 431:620.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. Artificial Intelligence 1:193–204.

Schmid, H. 1995. Improvements in part–of–speech tagging with an application to german. In Proceedings of the EACL SIGDAT Workshop.

Shazeer, N. M.; Keim, G. A.; and Littman, M. L. 1999. Solving crosswords as probabilistic contraint satisfaction. In Proceeding of AAAI '99, 156–152.

Tsochantaridis, I.; Hofmann, T.; Joachims, T.; and Altun, Y. 2004. Support vector machine learning for interdependent and structured output spaces. In Proceedings of ICML '04. ACM Press.

Voorhees, E. M., and Tice, D. M. 2000. Overview of the TREC-9 question answering track. In Proceedings of TREC-9. Department of Commerce, National Institute of Standards and Technology.