International Journal on Artificial Intelligence Tools

Automatic Generation of Crossword Puzzles

LEONARDO RIGUTINI, MICHELANGELO DILIGENTI, MARCO MAGGINI, MARCO GORI
Dipartimento di Ingegneria dell'Informazione Universiy of Siena, Via Roma 56
Siena, 53100, Italy
{*rigutini, diligmic, marco, maggini*}*@dii.unisi.it*

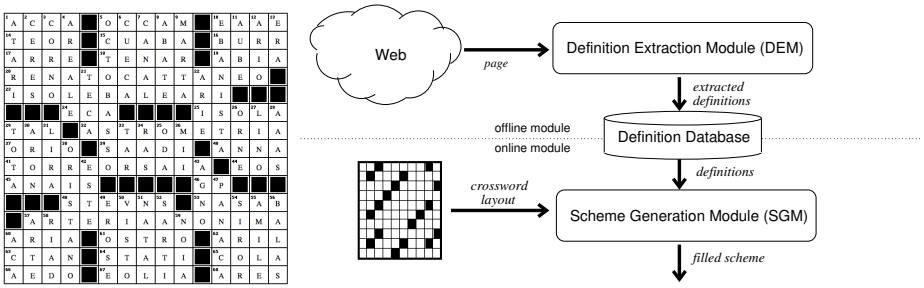
Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Crossword puzzles are used everyday by millions of people for entertainment, but have applications also in educational and rehabilitation contexts. Unfortunately, the generation of ad-hoc puzzles, especially on specific subjects, typically requires a great deal of human expert work. This paper presents the architecture of *WebCrow-generation*, a system that is able to generate crosswords with no human intervention, including clue generation and crossword compilation. In particular, the proposed system crawls information sources on the Web, extracts definitions from the downloaded pages using state-of-the-art natural language processing techniques and, finally, compiles the crossword schema with the extracted definitions by constraint satisfaction programming. The system has been tested on the creation of Italian crosswords, but the extensive use of machine learning makes the system easily portable to other languages.

Keywords: Crossword generation; information extraction; Natural Language Processing

1. Introduction

Because of the involvement of any kind of knowledge, language ambiguities, and deceiving tricks, crosswords are often thought of as truly human puzzles. Beside entertainment, crosswords are used in educational and rehabilitation contexts. For example, some studies claim that simple thematic puzzles are amongst the most effective instruments to support the process of student learning[1] and crossowords can be a powerful tool to reinforce terminology and improve the learning of definitional items[2]. Interestingly enough, Littman et al.[3,4] broke the belief that crossword solving is a task that can be approached only by humans with the implementation of Proverb, the first automatic crossword solver. Proverb exploits a simple architecture, in which the search for candidate answers is followed by a constraint satisfaction process to fill out the schema. Candidate words are discovered by a complex retrieval system in a huge database of solved crosswords and encyclopedia entries. A few years later, using most of the architectural solutions adopted in Proverb, another system, called WebCrow[5,6], contributed to dismiss the belief that crosswords cannot be cracked by computers. It early became quite popular because of



(a) (b)

Fig. 1. (a) An example of crossword automatically generated by WebCrow-generator. All terms have been automatically extracted from the Web along with their corresponding definitions. (b) The crossword generator crawls a set of relevant data sources from the Web and extracts the newly discovered definitions by applying NLP techniques on the downloaded pages. Finally, the extracted definitions are used to compile a given crossword layout.

its massive exploitation of the Web to download candidate terms for clues[7] and for having started, with interesting achievements, the man-machine crossword challenge (http://www.newscientist.com/article/dn9888).

Crossword generation seems to be even more challenging for machines and so far, to the best of our knowledge, it has not been attacked in a fully automatic way. Most of the studies on the subject have restricted attention to the task of compiling a crossword given a schema. The goal is to select a subset of words from a selected vocabulary so as to place them on the grid until no white cells are left empty. Crossword compilation is a combinatorial problem that has been approached by constraint satisfaction programming with good results[23,8,9,4]. Some real–world solutions have been devised that are based on human-edited collections of terms with the corresponding clues and sub-sequent compilation of the schema, but the human intervention is still crucial.

This paper presents WebCrow-generation, a system that generates crosswords with no human intervention, including definition/clue generation and crossword compilation. In Figure 1 an overall view of the system is shown that is essentially based on two modules aimed at carrying out the definition extraction and the schema generation, respectively. The input to the system is a predefined set of information sources, including Wiki pages and Web dictionaries that are efficiently imported by crawling[10]. Once the pages have been downloaded and the raw text has been extracted, the system extracts definitions from the text using state-of-the-art natural language processing techniques (NLP) and, finally, it compiles a given crossword layout by constraint satisfaction programming (CSP). The information sources do not have to meet any special requirement, since the proposed system works directly on the text extracted from any HTML page. Similarly, the system does not impose any predefined structure to the text on the pages, since the system processes and understands sentences which are written in a natural human

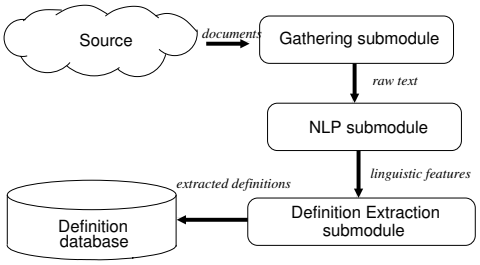


Fig. 2. The architecture of the Definition Extractor. The data sources are imported, then the documents are processed using an NLP analyzer and, finally, the linguistic features are used to extract the definitions.

language. Therefore, any Web site can be used as input to the system, even if it is better to use Web sites which are rich in encyclopedic knowledge, as the extracted definitions are more relevant.

The system has been evaluated in the generation of Italian crosswords. However, all the employed NLP modules are based on machine learning allowing to easily port the system to other languages with minor adaptations. In particular, it would be easy to cover other Indo-European languages based on the Latin alphabet (French, English, German, Spanish, etc.), as they all share the same syntactic and morphological structure (subject, verb, complements).

The outline of the paper is the following. In section 2 we describe the architecture of the definition extraction process, while in section 3 we present the architecture of the crossword compilation module. Some experimental result are reported in section 4, while in section 5 we draw some conclusions and delineate the possible future developments.

2. Definition Extraction

The definition extraction module has three components: the information gatherer, the natural language analyzer and the definition extractor (see Figure 2). The gatherer downloads the pages from the information sources and processes the content of each page discarding the formatting information and retaining the raw text. The information sources do not have to meet any special requirement beside to contain pages written using the HTML language. Therefore, any Web site can be used as input to the system. It would be relatively easy to add support for pages written using different standards, even if this is beyond the scope of this paper.

The extracted text is then passed to the Natural Language Processing analyzer. The NLP analysis is organized into layers as sketched in Figure 3. Each layer processes the output of the previous layer, with the exception of the first one that directly works on the input text. Finally, the extractor processes the linguistic features provided by the NLP analyzer and extracts the definitions. Since the system attempts at understanding text written in a human language, the text does not have

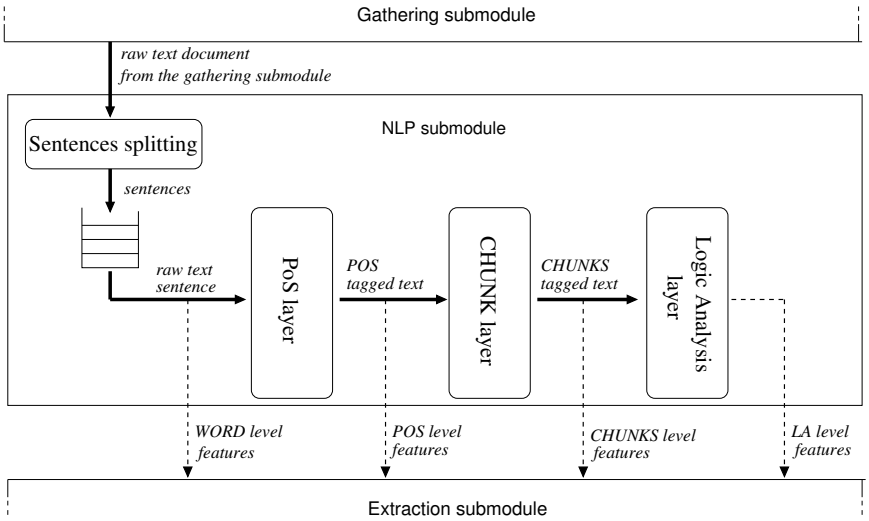


Fig. 3. The NLP analyzer is the core of the crossword generator. Due to its complexity it is subdivided into layers. The first layer performs low level linguistic tasks like sentence splitting and PoS tagging. The subsequent layer performs chunking and, finally, a simple logic analyzer extracts the subject and the action from the sentence.

to follow any predefined structure, making it easy to add new information sources.

The first layer is the *Part-of-Speech* (PoS) tagger, which identifies the grammatical properties of each word in a sentence. In the second layer, the PoS tags are used to assign the *chunk* tags. Phrase chunking is a well studied task in the NLP literature and it aims at grouping contiguous words representing a single semantic entity. The next layer performs the Logic Analysis and, starting from the previously extracted chunks, it identifies the subject, the object and the predicate of the sentence. Other NLP layers (Clause Identification, Anaphora Resolution, etc.) are implemented by the developed library, but they are not used in the system. We now describe the implementation of each linguistic layer.

2.1. *The Part-of-Speech (PoS) layer*

The first NLP layer is the *Part-of-Speech* (PoS) tagging. The main PoS tag summarizes the grammatical role of the word (noun, proper noun, adjective, adverb, verb, etc.). Furthermore, the PoS tagger can assign other morphological attributes that depend on the specific grammatical class of the word, such as the number (singular or plural) and the gender (male or female) for nouns, and the person, tense (present, past, etc.) and mode for verbs. The PoS tagger processes the input text, sentence by sentence, and attaches a PoS tag to each word.

PoS tagging is a well studied task in the NLP literature. Machine learning approaches are very popular because they can provide state-of-the-art results with

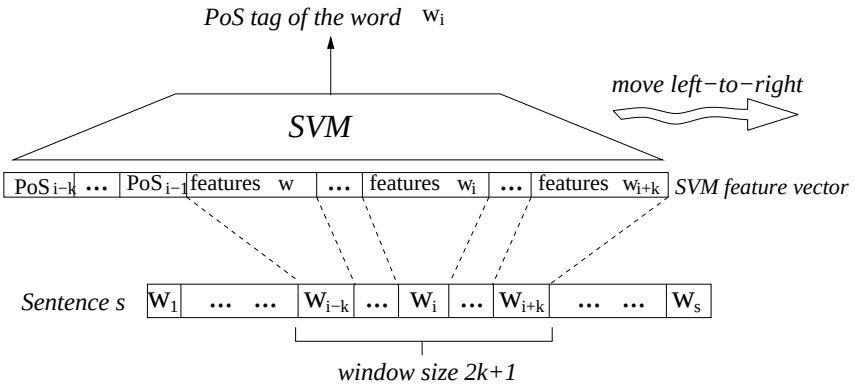


Fig. 4. The PoS tagger employed in the proposed system scans the input text using a sliding window. Each window is represented by a set of binary features that are processed using an SVM.

little tuning. The tagging rules can be directly learned from a set of examples[11], or it is possible to devise purely statistical PoS taggers like *Trigrams'n'Tags* (TnT)[12]. Other remarkable approaches are based on Hidden Markov Models (HMMs)[13,14], or on stochastic methods based on Maximum Entropy Markov Models (MEMM)[15]. The PoS tagger employed in our system is implemented using a machine learning approach very similar to the model proposed in Gimenez et al.[16], where the authors propose an effective tagger based on Support Vector Machines (SVMs). The SVM-based PoS tagger (see Figure 4) scans the input text using a sliding-window with size $2k + 1$. The tag of the i-th word w_i in the sentence is predicted by using the words $w_{i-k}, ..., w_i, ...w_{i+k}$ surrounding w_i. After processing w_i, the window is moved rightwards to the next term. A set of binary features is computed for each word in the window.

In our implementation, the following features are inserted in the window representation,

- *Word features.* These binary features represent the properties of the raw characters in a word. For example, they account for the case of the characters ("Does the word start with an uppercase letter?", "Is it all uppercase?", etc.), or for the presence of digits ("Is it a number?", "Does it contain a number?", etc.) and of non-alphanumeric symbols ("Does the word contain a dot?", etc.) in the word.
- *PoS features.* Since we use a sliding window approach, the terms preceding the currently processed word w_i are already assigned a PoS tag. Hence, the tags already attached to $w_{i-k}, ..., w_{i-1}$ can be used as features for w_i. If the PoS tag list contains $|T_{PoS}|$ possible labels, $t_1, ..., t_{|T_{PoS}|}$, the set of features for the word w_p, $p = i - k, ..., i - 1$, is represented as a $|T_{PoS}|$-dimensional binary vector where the j-the entry is 1 if t_j is the PoS tag of the word w_p, 0 otherwise.
- *Memory-based features.* Let V be a vocabulary of words. Each word in

the vocabulary is assigned a unique *id* and can be represented as a $|V|$-dimensional vector where the j-th element is 1 if j is the id of the current word and 0 otherwise. The prefix and the suffix of each term are also represented as a binary vector, using the same encoding based on a fixed dictionary of prefixes and suffixes.

- *The Ambiguity class.* It stores all the tags which are assigned at least once to a given word in the training set. The ambiguity class for the term is a $|T_{PoS}|$-dimensional binary vector where the j-th element is 1 is and only if the word has the tag t_j in its ambiguity class. This feature represents the set of possible tags that are suitable for the term under analysis. In other words, the ambiguity class answers to the question "Is the tag t_j likely to be predicted for this given word?". The ambiguity class has been shown to improve the accuracy of the predictions in PoS tagging tasks as shown by Daelemans et al.[17].

All the above features are merged into a single feature vector which represents the window under consideration. A set of SVM models is then used to process the feature vectors. Each SVM is associated to one PoS tag. In the training phase, the window is provided to all the SVMs: the SVM associated to the PoS tag of the central term of the window is trained to output 1 as target value. All other SVMs are instead trained to predict 0 given the input vector. When tagging a unseen sentence, the feature vector is extracted and processed by the SVMs. The SVM that predicts a 1 with maximum confidence determines the tag for the central word of the window (w_i).

As an illustrative example, suppose to have the phrase "Us and them and after all we are only ordinary men" and to use a window of 5 terms ($k = 2$). We indicate with f_W, f_P, f_M, f_A the word, PoS, memory-based and ambiguity class features, respectively. The first term ("Us") is processed using a sliding window that takes into account w_{-2}, w_{-1}, w_0, w_1 and w_2. Words w_{-2} and w_{-1} do not exist, so the corresponding feature vectors are defaulting to a vector of all zeros. All the words have the f_P features set to zero since no PoS tags have been predicted yet. The target label provided to the SVM corresponding to the PoS class of 'pronouns' is set to 1, whereas all the other SVMs get 0 as target output. At the next iteration, the model takes into consideration the word w_1 ('and') based on the features representing the words w_{-1}, w_0, w_1, w_2 and w_3. All the features of the w_{-1} are set to 0, since the word does not exist. The f_P features for w_1, w_2, w_3 are set to zero, since no predicted PoS tag exists for such words, whereas the features f_P of the word w_0 contain the tag predicted at the previous step (the pronoun tag). Finally, the features f_W, f_M and f_A for the tokens w_0, w_1, w_2, w_3 have non-null values. The target tag for the word w_1 is 'conjunction'. The procedure is repeated for all the words in the sentence and, then, for all the sentences in the training set.

The performances of the tagger have been measured using the Treebank (TUT)

Table 1. Results of the SVM and MEMM PoS taggers on the TUT dataset.

	SVM window-based	MEMM
Accuracy	$91.5\% \pm 0.5$	$92.3\% \pm 0.3$
Recall	$91.2\% \pm 0.8$	$92.5\% \pm 0.4$
Precision	$91.4\% \pm 0.3$	$92.0\% \pm 0.3$

dataset[a], a collection of morphologically, syntactically and semantically annotated Italian sentences, released by the Turin University and used as benchmark at *EVALITA 2007*[b]. The TUT dataset adopts a Penn-like annotation but differentiates from Penn[c] because of the set of PoS tags[d]. The set of PoS tags is composed by about 80 tags. In our implementation, the sliding window has size 5, i.e. ($k = 2$) and $20,000$ features are extracted for each word. Hence, the resulting vector, associated for each window, collects 10^5 binary features. 5-fold cross validation was performed by using $2,000$ and $1,000$ tagged sentences as learning and test set, respectively. The performances of the model have been compared with the MEMM-based tagger described by Ratnaparkhi[15], which reports state-of-the-art results on the TUT English dataset (accuracy equal to $96,6\%$).

As shown in Table 1, both models show similar performances on the TUT dataset. As suggested by Gimenez et al.[16], the SVM-based approach is preferable over most of the other state-of-the-art techniques because of its simplicity, flexibility and efficiency. Therefore, the SVM tagger was employed in our system.

2.2. *The chunking layer*

The second layer is the *chunker* (CHK) or *shallow parser* which groups the terms and identifies the base constituents of a sentence. In grammatical theory, the main constituent types are noun phrase (NP), prepositional phrase (PP), verbal phrase (VP) and adverbial phrase (ADVP). A noun phrase is a set of contiguous words whose head is a noun (common noun or person noun), a pronoun or an adjective. Similarly, a prepositional phrase is a phrase whose head is a preposition, a verbal phrase is a phrase which contains only verbs and an adverbial phrase is a phrase consisting of one o more adverbs. The adverbial phrases are often not considered and the adverbs are considered part of a VP.

Chunking has been extensively studied in the last decade. Previous work showed that it is possible to achieve state-of-the-art performances using an ensemble of classifiers each of which implements a base chunker[18]. The chunker employed by WebCrow-generation (TRI-CHK) exploits an ensemble of three SVMs based on

[a]http://www.di.unito.it/∼tuttreeb
[b]http://evalita.itc.it/
[c]http://www.cis.upenn.edu/∼treebank/
[d]The use of the Penn-TUT PoS tags, that are derived by reduction from the TUT original PoS tags, has been preferred because they better represent the inflectional richness of Italian.

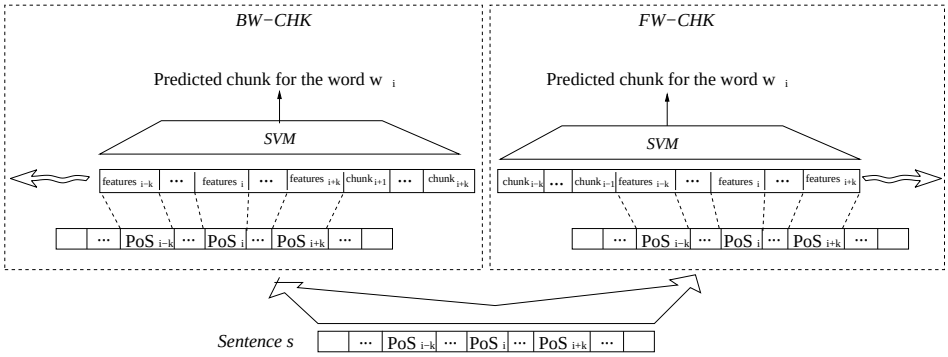


Fig. 5. The forward (FW-CHK) and backward (BW-CHK) chunkers. The forward chunker scans the input data from left to right while the backward chunker analyses the sentence in the opposite direction. The two chunkers can predict different chunk labels for a given token, leaving to a third chunker the resolution of the conflicts.

a sliding window processing schema: a forward chunker which moves its window from left to right (FW-CHK), a backward chunker that moves it from right to left (BW-CHK), and a third chunker (R-CHK) that combines the outputs of FW-CHK and BW-CHK. Hence, the TRI-CHK chunker learns from examples how to merge the outputs of the base chunkers, without relying on any pre-encoded combination function. This improves what proposed in previous work in the literature[19,20], where the outputs of the different SVM-based chunkers are merged using a predefined weighted majority voting.

The Forward and Backward chunkers

Using the same notation that we used for the PoS tagger, we assume w_i to be the current word and to use a window of size $2k+1$ words centered around w_i. For each word in the window, the following set of features is extracted.

- *PoS features.* While the PoS tagger uses features derived from the raw analysis of text (the words), the shallow parser predicts the chunks by observing the outputs of the PoS layer (the predicted PoS tags). This follows the general pattern of the adopted layered NLP analysis, where the output of each layer is the input for the subsequent one. Hence, a $|T_{PoS}|$-dimensional binary vector is created for each term w_p in the window, where the j-th entry is equal to 1 if the PoS tagger assigns w_p to the j-th entry in the PoS tag list, 0 otherwise. These features are the same for both the forward and the backward chunker.
- *Chunk features.* The chunk tags have been already predicted by the chunker for the words in the windows preceding the current word w_i (from w_{i-k} to w_{i-1}) for FW-CHK or following w_i (from w_{i+1} to w_{i+k}) for BW-CHK. Hence, if $|T_{CHK}|$ is the size of the chunk tag list, a $|T_{CHK}|$-dimensional

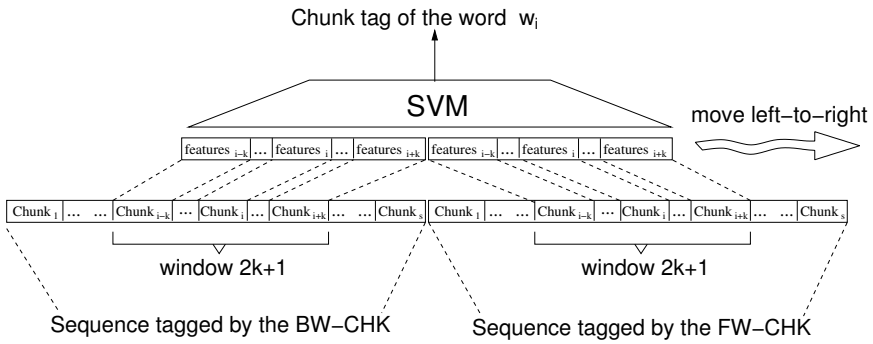


Fig. 6. The root chunker (R-CHK) is an SVM based chunker that processes the outputs of the forward and backward chunkers and takes a final decision on the chunk tag to be assigned to the central token of the window.

binary vector is assigned to each term w_p preceding (FW-CHK) or following (BW-CHK) w_i, with a *one-hot* vector encoding similar to that adopted for the other binary features.

Hence, the pattern extracted for each word is composed by $|T_{PoS}| \times (2k+1) + |T_{CHK}| \times k$ binary features. When $i < k$ or $i + k > length(s)$, being $length(s)$ the number of words in the current sentence s, some words in the window are missing and the features related to these words are set to zero. The processing schema of these two chunkers is sketched in Figure 5.

The Root chunker

The forward and backward chunkers output two tag sequences that are analyzed by the *root chunker* (R-CHK), that performs the final prediction for each word. R-CHK is implemented as a window-based multi-class SVM that processes the $2 \cdot (2k+1)$ tags predicted by the forward and backward chunkers for w_i and the k preceding and following terms. Each tag is encoded by a $|T_{CHK}|$-dimensional binary vector as described before. The resulting feature vector contains $2 \cdot (2k+1) \cdot |T_{CHK}|$ entries and is fed to the pool of SVMs, one for each chunk class. The architecture of the R-CHK chunker is sketched in Figure 6.

We tested the proposed chunker using the TUT dataset. The set of the considered tags for the experiments are NP, PP and VP. The _S suffix is appended to indicate a START occurrence, the _I suffix to indicate an INTERMEDIATE element and the _E suffix stand for the END event of the chunk. When the chunk is composed by a single word, no suffixes are added (NP,VP or PP are directly used). The tag O (Other) is used to identify unclassified chunks.

Table 2 summarizes some experiments we performed to verify the improvement of TRI-CHK compared to the use of the single BW-CHK, FW-CHK. The results show that TRI-CHK corrects some predictions of the single–direction chunkers and increases the overall accuracy of chunking by $2\% - 3\%$ with respect to BW-CHK and

Table 2. The chunking results on the TUT dataset using as features the PoS tags computed by the PoS tagger. The chunker results can be negatively affected by errors of the underlying PoS tags.

	BW-CHK	FW-CHK	TRI-CHK
Accuracy	$88.5\% \pm 0.9$	$89.6\% \pm 0.5$	91.4 ± 0.3
Recall	$86.4\% \pm 0.8$	$88.9\% \pm 0.4$	89.8 ± 0.3
Precision	$90.6\% \pm 0.6$	$90.5\% \pm 0.2$	91.3 ± 0.3

Table 3. The chunking results on the TUT dataset using the human edited PoS Tags provided in the TUT dataset. This highlights the accuracy of the chunker by isolating it from the errors of the PoS tagger.

	BW-CHK	FW-CHK	TRI-CHK
Accuracy	$89.4\% \pm 0.6$	$90.7\% \pm 0.4$	92.3 ± 0.3
Recall	$87.6\% \pm 0.5$	$89.8\% \pm 0.2$	90.4 ± 0.5
Precision	$90.3\% \pm 0.5$	$92.1\% \pm 0.2$	92.5 ± 0.2

1.5% with respect to FW-CHK. Since the chunking layer depends on the quality of the PoS layer, we performed a test using the PoS tags directly provided in the ground-truth TUT annotations to identify the impact of the R-CHK, assuming to have no errors in the input features. The results are summarized in Table 3. Comparing the results in Tables 2 and 3, we note that the chunker results are only slightly influenced by the PoS tagging step (about 1%).

2.3. *The logic analysis (LA) layer*

The final NLP layer processes the output of the previous two layers to perform Logic Analysis on the input text. Logic analysis detects the logic roles of the chunks, detecting the subject, the predicate and the complements. In the implementation, a simple logic analyzer was exploited, that identifies the subject and the predicate in each sentence without taking care of the complements. The logic-analyzer is currently implemented as a finite state automaton, even if machine learning approaches have been considered for future developments. The automaton processes the chunk sequence predicted by the previous layer and assigns one of the following four syntactic tags to each chunk: subject, nominal predicate, verbal predicate, other. The finite-state automaton has 4 states, one for each tag and the transitions between states are activated by the following sets of features.

- *Chunk type.* The label of the current chunk. For example, only a VP chunk can be labeled as 'verbal predicate'. Similarly, only a NP chunk can be labeled as 'subject';
- *Chunk-level boolean features.* A set of boolean predicates representing the composition of the current chunk. For example, "Does the chunk contains

a punctuation mark?".

- *Sentence-level boolean features.* A set of boolean predicates representing the overall structure of the sentence. For example: "Does the sentence already contain a predicate?", "Does the sentence already contain a subject?".

2.4. *Extracting definitions from labeled sentences*

The Definition Extractor identifies and extracts the definitions in the sentence, processing the linguistic features extracted by the NLP analysis. From an analysis made on a dataset of clues from Italian published crosswords, it results that around 80% of definitions follow a "nominal structure", that consists of a subject, followed by a nominal predicate and by the complements. The definition extractor is implemented as a finite state automaton which observes the sequence of the PoS, chunk and logic analysis tags and decides whether a sentence is a definition. If the phrase is classified as a definition, the subject is removed from the sentence and the pair <subject,definition> is inserted into the database. In the opposite case, the sentence is discarded and the next sentence is analyzed. The definitions stored in the DB are then used by the Scheme Generation Nodule (SGM) to fill an empty schema and create a new crossword puzzle.

3. Crossword Compilation

Crossword compilation from a given vocabulary is a challenging Constraint Satisfaction Programming problem. Let us assume to have available a vocabulary of definitions (a definition is a word/clue pair). We refer to a *black cell* to indicate an element of a crossword grid that should never be overwritten by a word. Instead, a *white cell* indicates an element that can accommodate a character. A horizontal or vertical sequence of consecutive white cells forms a *slot*. We will refer to the *layout* of a crossword as the skeleton of the crossword with its boundaries and black cells. We define a *schema* as a layout partially filled with words.

A new word can be inserted into the schema either horizontally or vertically but only if it starts and ends at a slot boundary or at a black cell (*layout constraints*) and it does not modify any previously inserted word (*schema constraints*). A *compatible* word for a given slot on a schema is a word that respects the actual constraints on the crossword in the horizontal and vertical directions and it could be therefore written on the slot.

The core of the solving system is a priority queue which stores the set of partial solutions (schemas). At each step, a new partial solution is popped from the queue and an insertion slot is selected. A set of candidate words is obtained by querying the index to determine the words in the vocabulary that are satisfying the constraints on the selected slot of the schema. Since the index is queried at each iteration, it is essential to make it very fast to avoid it becoming the bottleneck of the search process.

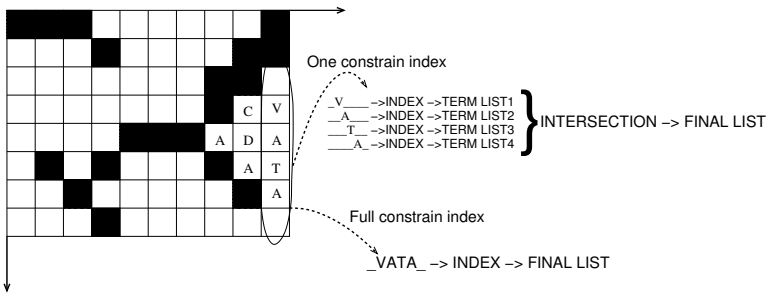


Fig. 7. A *one-constraint* index fetches a list of compatible words for each constraint imposed by the schema. List intersection is performed among the returned lists. A *full-constraint* index allows the direct look up of the list of compatible words for any combination of constraints.

Different implementations of a constraint index are possible. A *term scan* index (TS) is the simplest implementation. All terms of the desired length g are scanned checking if they satisfy the constraints. This solution has an average time complexity equal to $O(n_g \times m)$ where m is the number of constraints and n_g is the number of definitions in the vocabulary satisfying the layout constraints. A *one-constraint* index (OC) represents each constraint as a pair (character, position-in-the-term). This implementation of the index creates an inverted index associating each constraint to a list of compatible terms. When multiple constraints are imposed by the schema, each single constraint is matched against the index and, then, the returned lists are intersected. Since a term of length g appears in g different lists, this requires to store up to $\hat{g} \times n$ word instances in the index, where $\hat{g}$ is the average length of a term. Assuming that list intersection is performed in linear time by keeping the lists alphabetically sorted, the time complexity is reduced to $\hat{g} \times n_{\hat{g}}$ where $n_{\hat{g}}$ is the average length of a list which is typically much smaller than the vocabulary size[e]. Finally, a *full-constraint* index (FC) stores the list of compatible words for any combination of constraints. The computational complexity of a full index is reduced to $O(1)$ because any set of constraints requires a simple lookup. A query to a full index corresponds to a lookup of a string of the same length of the slot to be filled. The string is in the form $t_1 t_2 \ldots t_g$ where $t_i \in \{a, \ldots, z, \_\}$. The presence of the symbol $'\_'$ in a given position is used to indicate that a constraint is not set for that position (meaning that a white cell is present in that position in the schema). On the other hand, the presence of an alphabetical character in a given position indicates that the character must appear in the same position in the returned words. Unfortunately, a single word of length g that needs to be stored by the index corresponds to 2^g entries. Therefore, it can be very expensive to create a full index for long terms. In Figure 7 we show the difference between a full and an one constraint index. In our implementation, a full-constraint index is used to

[e]This analysis assumes the probability distribution over the constraints and over the word length to be uniform, which is a very crude assumption for real vocabularies of definitions.

look up terms with length lower than 10. For longer terms, our system keeps a *one-constraint* index that associates each constraint to a list of compatible terms.

Even a fast index does not suffice to find good solutions when the search space is large. Therefore, the heuristics to guide the search are crucial. Every time that a new partial solution is popped from the queue, the solver selects the slot with most white cells left to be filled. In particular, this strategy selects the longest slot on the schema as the starting slot at the first iteration. This strategy has been directly borrowed from Ginsberg et al.[21], where the authors notice that better compilation performances are obtained by restricting the search space as early as possible. A certain degree of randomization is allowed to be able to generate different solutions for different runs. The definition of a heuristic to sort the partial solutions is crucial for the success of the crossword compilation, since only a tiny portion of the search space can be explicitly expanded. Every time that a new term is written, a score is assigned to the newly obtained schema accounting for the "goodness" of the newly inserted word and how close to a full solution the schema is. The distance from a full solution is easy to determine, as it only requires to count the number of non-filled cells. In order to assess the goodness score for an insertion, *one step lookahead* is performed to check how many definitions would be compatible with the newly inserted term in the crossing slots at the next search iteration. Let $p(c)$ be the probability of finding a future match involving a character c of the word in the crossing slot passing for c. We assume that $p(c)$ is proportional to the number of terms that would be compatible in the crossing slot. Assuming term independence across different crossing slots, the probability of finding a set of future matches involving all the characters of the inserted term is proportional to the product of the probabilities for each symbol:

$$p(w) \propto \prod_{i=0}^{g} p(c_i) \propto \prod_{i=0}^{g} num\_compatible\_words(c_i)$$

where w is a word of length g and c_i is its i-th character. Please note that the independence assumption is incorrect since the terms correlate through the constraints. However, the goodness score performs well in practice, similarly to what happens for Naive Bayes classifiers in text categorization tasks[22]. In the extreme case of a term that introduces a constraint that can not be satisfied by any definition, $p(w)$ becomes 0. This avoids expanding a term w that would lead to a dead-end at the next step.

4. Experimental Results

Two set of experiments have been performed to validate the definition extraction and the crossword compilation modules.

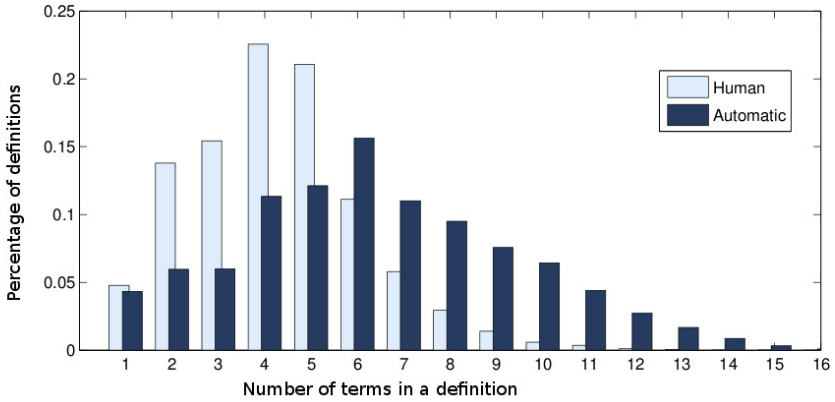


Fig. 8. The distribution of the number of terms in the definitions: extracted from real crosswords (light bars) and automatically generated definitions (dark bars). The automatically generated definitions are only slightly longer than the very compact ones created by human editors.

Definitions Extraction

To validate the definition extraction module, we looked at the quality of the extracted definitions. In particular, the Definition Extractor was run over the pages in the Italian Wikipedia[f]. Please note that the system can easily integrate other inputs (online dictionaries, search engines as Google, etc.), since any Web site containing HTML pages can be crawled and processed by the system. Excluding disambiguation and redirection pages, the Italian Wikipedia was composed by 409000 pages when it was downloaded. To speed up the processing, only the first 5 sentences per page have been analyzed by the NLP modules and the definitions have been extracted from them. The system discovered about 91000 definitions from the Italian Wikipedia (after removing the definitions including numbers and, more in general, any non-alphabetic character that can not be used in a crossword puzzle). This means that 22% of the processed pages contained a definition that was considered to be suitable for insertion in a crossword puzzle.

Table 4 shows a small sample of extracted word/definition pairs (after being literally translated from Italian). It is clear that the system is mainly extracting encyclopedic definitions, which are nevertheless ambiguous and challenging.

The quality of a crossword can be measured by the length of its definitions (clues): human edited clues are usually very compact. This means that a "realistic" crossword puzzle should not use too many long definitions, since short definitions are often more enigmatic. Crossword experts usually refer to this compactness as an important feature in defining the crossword esthetic. Figure 8 compares the distribution of the number of terms in the automatically discovered definitions against

[f]http://it.wikipedia.org. The Wikipedia data collection is provided as a database under the GNU Free Documentation License (GFDL).

Table 4. Some examples of (word, definition) pairs that have been automatically extracted by our system (literally translated from Italian).

word	clue
zelda	Nintendo character
pamtillis	country singer
dernier	card game
leonardmccoy	Star Trek character
sandracretu	German singer
pastoralodon	big extinct mammal
musket	weapon
johnfante	American writer
rishdan	city in the Fergana province in Uzbekistan
heap	data structure used in computer science
student	someone that is learning something

the same metric measured over a set of 30000 human edited definitions that are used by *Corrado Tedeschi Editore*[g] to generate crosswords for their magazines. The automatically generated crosswords are slightly longer than the human edited ones but they are still quite compact, meaning that the resulting crossword will be realistic and visually similar to a human generated one.

Using a pool of three human raters, we measured the percentage of correct and crossword-suitable definitions returned by the system. All the selected human raters are expert crossworders, compiling crossword puzzles on a regular basis. As shown in Table 5, the high accuracy of the NLP analysis allowed keeping the percentage of wrong definitions to a very low value. However, there is a higher portion of definitions that are semantically correct but that can not be used in a real crossword puzzle, since they are either too long, or they contain terms too similar to the solution itself, or they are too ambiguous.

Please note that we did not attempt at measuring recall directly, since this would require the human raters to specify the full set of definitions which can be extracted from a given sample of pages. This is a very hard task to perform and it would lead to very inconsistent results.

A crossword puzzle should contain a mix of ambiguous (admitting a large set of possible solutions) and punctual definitions. We manually classified as punctual or ambiguous 1000 definitions extracted from a database of crosswords published by Corrado Tedeschi in various magazines and 1000 definitions extracted by the automatic system. Table 6 compares the obtained ambiguity distributions for the human and automatic generated crosswords. Surprisingly, the two distributions look very similar and confirm that WebCrow-generation is able to generate realistic crosswords.

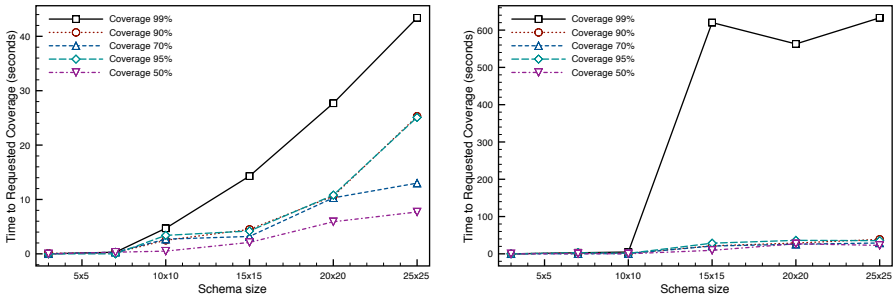
[g]http://www.tedeschi-net.it/ — *Corrado Tedeschi* is the second most popular and oldest crossword publisher in Italy.

Table 5. Percentage of correct and wrong definitions extracted by WebCrow-generation.

correct	wrong	correct but non usable
81%	3%	16%

Table 6. Ambiguity level for the automatic extracted definitions and the definitions in a set of human edited crosswords.

	Ambiguous	Punctual
Automatic	52%	48%
Human	55%	45%

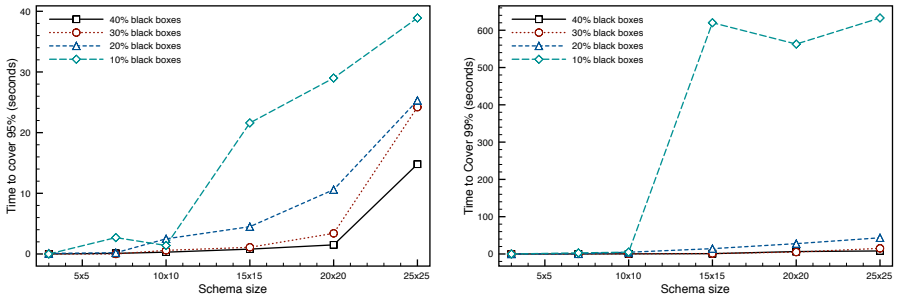


(a) (b)

Fig. 9. Average time (seconds) needed to complete 95% cells for each schema size for a crossword with (a) 20% and (b) 10% black cells.

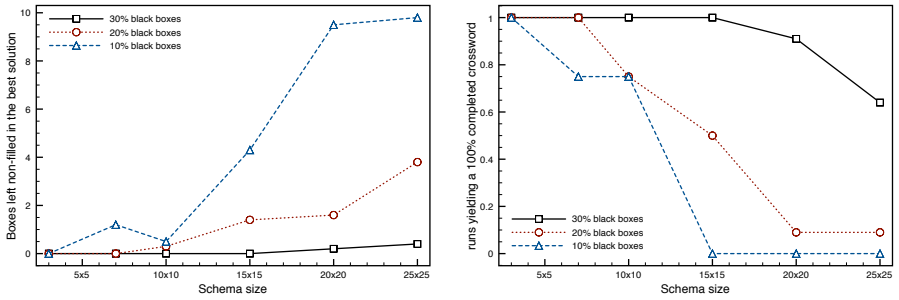
Crossword Compilation

The crossword compilation module has been evaluated by compiling a set of random layouts using the 91000 definitions generated by the definition extractor. The layouts were controlled by two parameters: the percentage of white cells (schema difficulty D) and the vertical/horizontal size N. Please note that this produces the classical square-shaped schemas. We decided to test the system on these shapes as they are the standard and, given number of cells, they feature the highest number of constraints, making them the hardest to be compiled. However, the system can work on schemas of arbitrary shape. Larger values of D and N correspond to schemas that are harder to compile due to the higher number of constraints. The performance of the crossword compiler was evaluated as the time needed to discover an $n\%$ filled solution, the percentage of times in which the solver finds a complete solution in a maximum of 10000 iterations and the percentage of completed slots in the best solution discovered in 10000 iterations. Single experiments are noisy due to the random generation of the layouts. For this reason, the results reported in this



(a) (b)

Fig. 10. Time (seconds) needed to complete (a) 95% and (b) 99% cells for each schema size in the case of a crossword with 20% of black cells.



(a) (b)

Fig. 11. (a) Average number of non-filled boxes left on the best discovered solution. (b) Percentage of runs returning a 100% completed crossword.

sections have been averaged over 30 different randomly generated layouts for each selection of the parameters D and N.

As shown in Figure 9 and 10, the solver reliably and quickly converges to good partial solutions. However, going very close to a full solution becomes more complicated when the number of constraints is high (e.g. schemas bigger than 15×15 with few black cells).

As shown in Figure 11-(b), the solver is very likely to discover a full solution for small crosswords or large ones with not too many constraints. When these conditions are not met, many runs can be needed before a full solution is found. However, as shown in Figure 11-(a), even if a full solution is not found, the solver goes very close to it, discovering schemas with only a tiny non-filled portion.

4.1. *Topic-specific crosswords*

A *topic-specific crossword* is a crossword having most of the definition/answer pairs belonging to a given topic T. In a real-word scenario, a user specifies a set of topics of expertise, and the automatic system generates a crossword that is targeted to his/her skills. It is clear that the manual generation of topic-specific crosswords is a challenge, as the number of possible topics is large and it is usually expensive to find a human expert for each possible topic.

In this section we evaluate the accuracy of the automatic crossword generator when building topic-specific crosswords. In particular, three topics have been taken into account: movies, music and sport. A sample of 1000 documents available in the Italian DMOZ section[h] for each target topic has been extracted. Another sample of 10000 URLs has been extracted from all the categories available in DMOZ. All the sampled pages have been downloaded from the Internet and parsed to extract the contained terms. For each term t_k and category c_i, the odd ratio score is computed as:

$$OR(t_k, c_i) = \frac{p(t_k|c_i) \cdot (1 - p(t_k|\overline{c_i}))}{(1 - p(t_k|c_i)) \cdot p(t_k|\overline{c_i})} \ ,$$

where $p(t_k|c_i)$ is the probability that t_k is selected when drawing a random term from a document of class c_i. $p(t_k|c_i)$ can be estimated as $p(t_k|c_i) \approx \frac{n(t_k,c_i)}{\sum_j n(t_j,c_i)}$, where $n(t_k, c_i)$ is the number of occurrences of t_k in the documents of class c_i. $p(t_k|\overline{c_i})$ indicates the probability that t_k is selected when drawing a random term from a document in $\overline{c_i}$ (the set of documents not belonging to class c_i). The odd ratio is often used in the text categorization literature as a feature selection method to detect the most discriminative terms to represent a class [24].

In the experimental setting, we selected the keywords with a score above 5 for each class, keeping a few tens of candidates for each class. The list has been manually pruned to remove noise, keeping between 20 and 80 topical words per class. The 91000 extracted definitions matching at least a topical word have been assigned to the corresponding class, resulting into 7218, 5536 and 4580 definitions for the category movies, music and sport, respectively. Please note that it can happen that the same definition is assigned to multiple categories. A manual rating of 600 classified pairs showed that the classifier is correct in 98.7% of cases. This simple classification schema has been preferred to a standard text categorization approach. Indeed, we tried to train a SVM-based text classifier build over DMOZ documents, and then to apply it over the definitions. However, the different nature of the two sets of documents made generalization very hard for the SVM classifier. On the other hand, we wanted to to avoid generating a training set of definitions for each considered class. The simple approach based on keywords probably features a low

[h]http://www.dmoz.org/World/Italiano/. The Open Directory Project (ODP but mainly known as DMOZ) is a multilingual semantic taxonomy of Web documents. It is constructed and maintained by a community of volunteer editors.

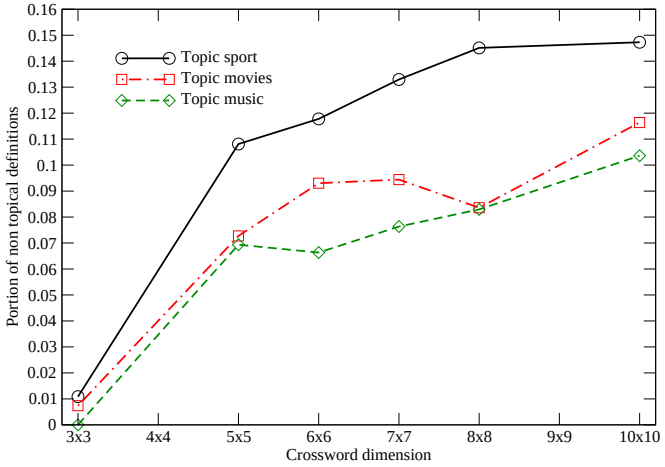


Fig. 12. Portion of non-topical definition/answer pairs that are selected in the automatically generated crossword for the categories "movies", "music" and "sport", while varying the crossword dimension.

recall (not measured) but, as previously reported, also provides a very high precision, which would be hard to achieve using standard text classifiers.

The topical and non-topical definition/answer pairs have been integrated in the search heuristic by adding the number of topical definitions to the score used to sort the schemas in the priority queue. This means that schemas with a higher partial number of topical definitions get a higher score and should be selected first, when this does not come at the cost of a too strong reduction of the space of possible future expansions.

Figure 12 reports the number of non-topical definition/answer pairs that are selected in the automatically generated crossword for each target category for different crossword dimensions (20% of black boxes). Ideally, the number of non-topical definitions should be 0, however it is often necessary to select a small number of definitions from the larger non-topical set to complete the crosswords. Nevertheless, the number of non-topical definitions remains around 10% also for relatively big crosswords. This shows that the proposed methodology is effective in building crosswords that can be really considered as focused on a target topic by an end-user.

5. Conclusions and future developments

Crosswords have relevant applications in entertainment, educational, and rehabilitation contexts. However, their generation requires many hours of expert human work. For this reason, crosswords are usually obtained from magazines or text books

and personalization is very expensive. This paper presented a system that is able to automatically generate crosswords, including automatic generation of the clues. The proposed system crawls a predefined set of information sources on the Web. Then, it extracts a vocabulary with the associated definitions from the downloaded pages using state-of-the-art NLP techniques and, finally, it compiles a crossword layout using a CSP solver.

The proposed system is fully independent on the source of information provided as input. Thanks to the extensive use of NLP techniques, the system does not simply extract a fixed set of sentences from an encyclopedia. This greatly increases the number of extracted definitions and it helps in discarding phrases that would not be suitable for a crossword. It is also possible to create topic-specific crosswords, where most of the clues/answer are about a given topic. The experimental results show that the system is able to generate very efficiently large encyclopedic crosswords with human-like quality.

The software has been used to generate square-like crosswords, but it is by no means limited to this shape and we plan to test the generation of crosswords with more creative patterns in the future. The system has been tested on the generation of Italian crosswords. However, the extensive use of machine learning techniques throughout the whole architecture would allow covering other Indo-European languages based on the Latin alphabet (French, English, German, Spanish, etc.), as they all share the same syntactic and morphological structure (subject, verb, complements). It is beyond the scope of this work to study the generation of crossword puzzles on languages like Chinese, which have a much larger character set and very different syntactic rules.

Finally, we also plan to start controlling the difficulty of the generated crosswords in order to generate crosswords requiring a specific skill level to be completed.

Acknowledgements

We thank *Corrado Tedeschi Editore* for providing the human edited data used to assess the quality of the automatic generator and for introducing us to the secrets of the trade needed to generate realistic and visually appealing crosswords. Special thanks to Luigi Rizzi, Cristiano Chesi, and Vincenzo Moscati from the Computational Linguistic group of the University of Siena[i] for the insightful discussions on linguistic aspects of crosswords. The project has been supported by Google Inc., under the Google Research Awards program.

References

1. A. Wise, Web-Based Puzzle Program to Assist Students' Understanding of Research Methods, *Active Learning in Higher Education*, vol. 4, no. 2 (2003) p. 193.

[i]http://www.ciscl.unisi.it/

2. J. M. D. Hill, C. K. Ray, J. R. S. Blair, and C. A. Carver, Puzzles and games: addressing different learning styles in teaching operating systems concepts, in *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, (New York, NY, USA) (ACM)(2003) 182–186.
3. G. A. Keim, N. M. Shazeer, M. L. Littman, S. Agarwal, C. M. Cheves, J. Fitzgerald, J. Grosland, F. Jiang, S. Pollard, and K. Weinmeister, PROVERB: The probabilistic cruciverbalist, in *American Conference on Artificial Intelligence (AAAI/IAAI-99)*, (1999) 710–717.
4. M. Littman, G. Keim, and N. Shazeer, A probabilistic approach to solving crossword puzzles, *Artificial Intelligence*, vol. 134, no. 1-2 (2002) 23–55.
5. M. Ernandes and M. Gori, WebCrow: a WEB-based system for CROssWord solving, in *American Conference on Artificial Intelligence (AAAI-05)* (2005) 1412–1417.
6. M. Ernandes, G. Angelini, and M. Gori, A web-based agent challenges human experts on crosswords, *AI-Magazine*, vol. 29, no. 1, (2008) 77–90.
7. F. Castellani, Program cracks crosswords, *Nature news*, vol. 431, (2004) p. 620.
8. A. Aherne and C. Vogel, Wordnet enhanced automatic crossword generation, in *Proceedings of the 3rd International Wordnet Conference* (C. F. Petr Sojka, Key-Sun Choi and P. Vossen, eds.), (Seogwipo, Korea), (2006) 139–145.
9. A. Beacham, X. Chen, J. Sillito, and P. van Beek, Constraint Programming Lessons Learned from Crossword Puzzles, *Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence* (2001) 78–87.
10. J. Cho, H. Garcia-Molina, and L. Page, Efficient crawling through URL ordering, *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, (1998) 161–172.
11. E. Brill, A simple rule-based part-of-speech tagger, in *Proceedings of the 3rd Conference on Applied Natural Language Processing (ANLP)*, (Trento, IT) (1992) 152–155.
12. T. Brants, Tnt – a statistical part-of-speech tagger, in *Proceedings of the 6th Applied NLP Conference (ANLP)*, (Seattle (WA)), (2000) 224–231.
13. H. Schutze and Y. Singer, Part-of-speech tagging using a variable memory markov model, in *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, (Morristown, NJ, USA), (1994) 181–187.
14. S. Thede and M. Harper, A second-order Hidden Markov Model for part-of-speech tagging, in *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics* (1999) 175–182.
15. A. Ratnaparkhi, A maximum entropy model for part-of-speech tagging, in *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (E. Brill and K. Church, eds.), (Somerset, New Jersey: Association for Computational Linguistics) (1996) 133–142
16. J. Gimenez and L. Marquez, Fast and accurate part-of-speech tagging: The SVM approach revisited, in *Proceedings of RANLP* (2003) 158–165.
17. W. Daelemans, J. Zavrel, and S. Berck, Mbt: A memorybased part of speech tagger-generator, in *Proceedings of the 4th Workshop on Very Large Corpora*(1996) 14–27.
18. E. T. K. Sang, Text chunking by system combination, in *Proceedings of the Fourth Conference on Computational Natural Language Learning and of the Second Learning Language in Logic Workshop* (C. Cardie, W. Daelemans, C. Nédellec, and E. T. K. Sang, eds.), (Somerset, New Jersey: Association for Computational Linguistics) (2000) 151–153.
19. T. Kudo and Y. Matsumoto, Use of support vector learning for chunk identification, in *Proceedings of the 4th CoNLL and LLL Conference*, (2000) 142–144.
20. T. Kudo and Y. Matsumoto, Chunking with support vector machines, in *Proceedings*

of the Second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies (NAACL), (2001) 1–8.
21. M. L. Ginsberg, M. Frank, M. P. Halpin, and M. C. Torrance, Search lessons learned from crossword puzzles, in *AAAI*, (1990) 210–215.
22. A. McCallum and K. Nigam, A comparison of event models for Naive Bayes text classification, in *AAAI/ICML-98 Workshop on Learning for Text Categorization*, (1998) 41–48.
23. H. Berghel, Crossword compilation with Horn clauses, in *The Computer Journal* vol.30, n. 2, Br Computer Soc (1987) pg.183
24. M. Caropreso, S. Matwin, and F. Sebastiani, "A learner-independent evaluation of the usefulness of statistical phrases for automated text categorization," *Text databases and document management: Theory and practice*, pp. 78–102, 2001.