Learning to Rank Answer Candidates for Automatic Resolution of Crossword Puzzles

Gianni Barlacchi
University of Trento
38123 Povo (TN), Italy
gianni.barlacchi@gmail.com

Massimo Nicosia and **Alessandro Moschitti**
Qatar Computing Research Institute
5825 Doha, Qatar
m.nicosia@gmail.com, amoschitti@qf.org.qa

Abstract

In this paper, we study the impact of relational and syntactic representations for an interesting and challenging task: the automatic resolution of crossword puzzles. Automatic solvers are typically based on two answer retrieval modules: (i) a web search engine, e.g., Google, Bing, etc. and (ii) a database (DB) system for accessing previously resolved crossword puzzles. We show that learning to rank models based on relational syntactic structures defined between the clues and the answer can improve both modules above. In particular, our approach accesses the DB using a search engine and reranks its output by modeling paraphrasing. This improves on the MRR of previous system up to 53% in ranking answer candidates and greatly impacts on the resolution accuracy of crossword puzzles up to 15%.

1 Introduction

Crossword puzzles (CPs) are probably the most popular language games played around the world. It is very challenging for human intelligence as it requires high level of general knowledge, logical thinking, intuition and the ability to deal with ambiguities and puns. CPs normally have the form of a square or rectangular grid of white and black shaded squares. The white squares on the border of the grid or adjacent to the black ones are associated with clues. The goal of the game is to fill the sequences of white squares with words answering the clues.

There have been many attempts to build automatic CP solving systems, which have also participated in competitions such as The American Crossword Puzzle Tournament (ACPT). This is the oldest and largest CP tournament for crossword experts held in the United States. The goal of such systems is to outperform human players in solving crosswords more accurately and in less time.

Automatic CP solvers have been mainly targeted by the artificial intelligence (AI) community, who has mostly focused on AI techniques for filling the puzzle grid, given a set of answer candidates for each clue. The basic idea is to optimize the overall probability of correctly filling the entire grid by exploiting the likelihood of each candidate answer, fulfilling at the same time the grid constraints. After several failures in approaching the human expert performance, it has become clear that designing more accurate solvers would not have provided a winning system. In contrast, the Precision and Recall of the answer candidates are obviously a key factor: a very high value for both of them would enable the solver to quickly find the correct solution.

This basically suggests that, similarly to the Jeopardy! challenge case (Ferrucci et al., 2010b), the solution relies on Question Answering (QA) research. However, although some CP clues are rather similar to standard questions, as for example, in the clue/answer pair: ≪What keeps a camera rolling?: *dolly*≫, some specific differences hold: (i) clues can be in interrogative form or not, e.g., ≪Capital of USA: *Washington*≫; (ii) they can contain riddles or be deliberately ambiguous and misleading (e.g., ≪It's green at first: *orange*≫); (iii) the exact length of the answer keyword is known in advance; and (vi) the confidence in the answers is an extremely important input for the CP solver.

In this paper, we study methods for improving the quality of automatic extraction of answer candidate lists for automatic CP resolution. For this purpose, we designed learning to rank models for reordering the answers produced with two different techniques typically used in CP systems: (i) searching the Web with clue representations, e.g.,

Proceedings of the Eighteenth Conference on Computational Language Learning, pages 39–48,
Baltimore, Maryland USA, June 26-27 2014.

exploiting Bing search engine[1]; and (ii) querying the DB of previously resolved CP clues, e.g., using standard SQL techniques.

We rerank the text snippets returned by Bing by means of SVM preference ranking (Herbrich et al., 2000) for improving the first technique. One interesting contribution is that our model exploits a syntactic representation of clues to improve Web search. More in detail, we use structural kernels (e.g., see (Moschitti, 2006; Moschitti, 2008)) in SVMs applied to our syntactic representation of pairs, formed by clues with their candidate snippets. Regarding the DB approach, we provide a completely novel solution by substituting it and the SQL function with a search engine for retrieving clues similar to the target one. Then, we rerank the retrieved clues by applying SVMs and structural kernels to the syntactic representation of clue pairs. This way, SVMs learn to choose the best candidate among similar clues that are available in the DB. The syntactic representation captures clue paraphrasing properties.

In order to carry out our study, we created two different corpora, one for each task: (i) a snippets reranking dataset and (ii) a clue similarity dataset. The first includes 21,000 clues, each associated with 150 candidate snippets whereas the latter comprises 794,190 clues. These datasets constitute interesting resources that we made available to the research community[2].

We compare our methods with one of the best systems for automatic CP resolution, WebCrow (Ernandes et al., 2005). Such system does use the two approaches mentioned before. Regarding snippet reranking, our structural models improve on the basic approach of WebCrow based on Bing by more than 4 absolute percent points in MRR, for a relative improvement of 23%. Concerning the similar clues retrieval, our methods improve on the one used by WebCrow, based on DBs, by 25% absolute, i.e., about 53% of error reduction whereas the answer accuracy at first position improves up to 70%.

Given such promising results, we used our clue reranking method in WebCrow, and obtained an average improvement of 15% in resolving complete CPs. This demonstrates that advanced QA methods such as those based on syntactic structures and learning to rank methods can help to win the CP resolution challenge.

In the reminder of this paper, Sec. 2 introduces the automatic CP resolution task in the context of the related work, Sec. 3 introduces WebCrow, Sec. 4 illustrates our models for snippets reranking and similar clue retrieval using kernel methods, syntactic structures, and traditional feature vectors, Sec. 5 describes our experiments, and finally, Sec. 6 derives the conclusions.

2 Related Work

Proverb (Littman et al., 2002) was the first system for the automatic resolution of CPs. It includes several modules for generating lists of candidate answers. These lists are merged and used to solve a Probabilistic-Constraint Satisfaction Problem. Proverb relies on a very large crossword database as well as several expert modules, each of them mainly based on domain-specific databases (e.g., movies, writers and geography). In addition, it employs generic-word list generators and clue-specific modules to find solutions for particular kinds of clues like ≪Tel ____ (4): *aviv* ≫. Proverb's modules use many knowledge sources: databases of clues, encyclopedias and Web documents. During the 1998 ACPT, Proverb placed 109th out of 251 contestants.

WebCrow (Ernandes et al., 2005) is based on Proverb. It incorporates additional knowledge sources, provides a solver for the Italian language and improves the clues retrieval model from DB. In particular, it enables partial matching to retrieve clues that do not perfectly overlap with the query. WebCrow carries out basic linguistic analysis such as Part-Of-Speech tagging and lemmatization. It takes advantage of semantic relations contained in WordNet, dictionaries and gazetteers. Its Web module is constituted by a search engine, which can retrieve text snippets or documents related to the clue. Answer candidates and their confidence scores are generated from this content. WebCrow uses a WA* algorithm (Pohl, 1970) for Probabilistic-Constraint Satisfaction Problems, adapted for CP resolution. The solver fills the grid entries for which no solution was found by the previous modules. It tries combinations of letters that satisfy the crossword constraints, where the letters are derived from words found in dictionaries or in the generated candidate lists. WebCrow participated in international competitions with good results.

[1] https://www.bing.com/

[2] http://projects.disi.unitn.it/iKernels/projects/webcrow/

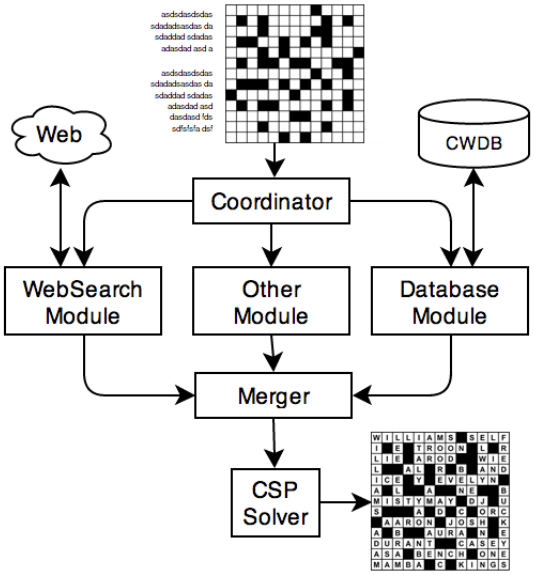

Figure 1: Overview of WebCrow's architecture.

Dr. Fill (Ginsberg, 2011) targets the crossword filling task with a Weighted-Constraint Satisfaction Problem. Constraint violations are weighted and can be tolerated. It heavily relies on huge databases of clues. It was placed 92nd out of more than 600 opponents in the 2013 ACPT.

Specifically for QA using syntactic structures, a referring work for our research is the IBM Watson system (Ferrucci et al., 2010a). This is an advanced QA pipeline based on deep linguistic processing and semantic resources. It demonstrated that automatic methods can be more accurate than human experts in answering complex questions.

More traditional studies on passage reranking, exploiting structural information, were carried out in (Katz and Lin, 2003), whereas other methods explored soft matching (i.e., lexical similarity) based on answer and named entity types (Aktolga et al., 2011). (Radlinski and Joachims, 2006; Jeon et al., 2005) applied question and answer classifiers for passage reranking. In this context, several approaches focused on reranking the answers to definition/description questions, e.g., (Shen and Lapata, 2007; Moschitti et al., 2007; Surdeanu et al., 2008; Severyn and Moschitti, 2012; Severyn et al., 2013b).

3 WebCrow Architecture

Our research focuses on the generation of accurate answer candidate lists, which, when used in a CP resolution systems, can improve the overall solution accuracy. Therefore, the quality of our modules can be assessed by testing them within such systems. For this purpose, we selected WebCrow as it is rather modular, accurate and it was kindly made available by the authors. Its architecture is illustrated in Figure 1.

The solving process is divided in two phases: in the first phase, the coordinator module forwards the clues of an input CP to a set of modules for the generation of several candidate answer lists. Each module returns a list of possible solutions for each clue. Such individual clue lists are then merged by a specific *Merger* component, which uses list confidence values and the probabilities of correctness of each candidate in the lists. Eventually, a single list of candidate-probability pairs is generated for each input clue. During the second phase WebCrow fills the crossword grid by solving a constraint-satisfaction problem. WebCrow selects a single answer from each candidate merged list, trying to satisfy the imposed constraints. The goal of this phase is to find an admissible solution maximizing the number of correct inserted words. In this paper, we focus on two essential modules of WebCrow: the Web and the DB modules, described in the next sections.

3.1 WebSearch Module (WSM)

WSM carries out four different tasks: (i) the retrieval of useful text snippets (TS) and web documents, (ii) the extraction of the answer candidates from such text, (iii) the scoring/filtering of the candidates, and (iv) the estimation of the list confidence. The retrieval of TS is performed by the Bing search engine by simply providing it the clue through its APIs. Then, the latter again are used to access the retrieved TS. The word list generator extracts possible candidate answers from TS or Web documents by picking the terms (also multiwords) of the correct length. The generated lists are merged and sorted using the candidate confidence computed by two filters: the statistical filter and the morphological filter. The score associated with each candidate word w is given by the following heuristic formula:

$$p(w, C) = k(score_{sf}(w, C) \times score_{mf}(w, C)),$$

where (i) C is the target clue, (ii) k is a constant tuned on a validation set such that $\sum_{i=0}^{n} p(w_n, C) = 1$, (iii) $score_{sf}(w, C)$ is computed using statistical information extracted from the text, e.g., the classical TF×IDF, and (iv) $score_{mf}(w, C)$ is computed using morphological features of w.

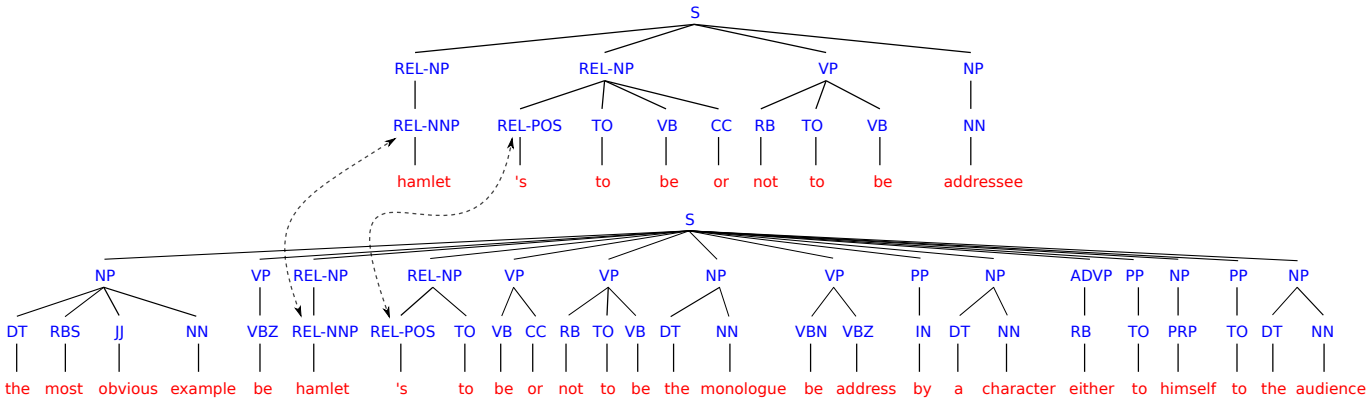

Figure 2: Shallow syntactic trees of clue (upper) and snippet (lower) and their relational links.

3.2 Database module (CWDB)

The knowledge about previous CPs is essential for solving new ones. Indeed, clues often repeat in different CPs, thus the availability of a large DB of clue-answer pairs allows for easily finding the answers to previously used clues. In order to exploit the database of clue-answer pairs, WebCrow uses three different modules:

CWDB-EXACT, which simply checks for an exact matching between the target clue and those in the DB. The score of the match is computed using the number of occurrences of the matched clue.

CWDB-PARTIAL, which employs MySQL's partial matching function, query expansion and positional term distances to compute clue-similarity scores, along with the Full-Text search functions.

CWDB-DICTIO, which simply returns the full list of words of correct length, ranked by their number of occurrences in the initial list.

We improve WSM and CWDB by applying learning-to-rank algorithms based on SVMs and tree kernels applied to structural representations. We describe our models in detail in the next section.

4 Learning to rank with kernels

The basic architecture of our reranking framework is relatively simple: it uses a standard preference kernel reranking approach (e.g., see (Shen and Joshi, 2005; Moschitti et al., 2006)). The structural kernel reranking framework is a specialization of the one we proposed in (Severyn and Moschitti, 2012; Severyn et al., 2013b; Severyn et al., 2013a). However, to tackle the novelty of the task, especially for clue DB retrieval, we modeled innovative kernels. In the following, we first describe the general framework and then we instantiate it for the two reranking tasks studied in this paper.

4.1 Kernel framework

The framework takes a textual query and retrieves a list of related text candidates using a search engine (applied to the Web or a DB), according to some similarity criteria. Then, the query and candidates are processed by an NLP pipeline. The pipeline is based on the UIMA framework (Ferrucci and Lally, 2004) and contains many text analysis components. The latter used for our specific tasks are: the tokenizer[3], sentence detector[1], lemmatizer[1], part-of-speech (POS) tagger[1], chunker[4] and stopword marker[5].

The annotations produced by such processors are used by additional components to produce structural models representing clues and TS. The structure component converts the text fragments into trees. We use both trees and feature vectors to represent pairs of clues and TS, which are employed to train kernel-based rerankers for reordering the candidate lists provided by a search engine. Since the syntactic parsing accuracy can impact the quality of our structure and thus the accuracy of our learning to rank algorithms, we preferred to use shallow syntactic trees over full syntactic representations. In the next section, we first describe the structures we used in our kernels, then the tree kernels used as building blocks for our models. Finally, we show the reranking models for both tasks, TS and clue reranking.

[3] http://nlp.stanford.edu/software/corenlp.shtml

[4] http://cogcomp.cs.illinois.edu/page/software_view/13

[5] Based on a standard stoplist.

Rank	Clue	Answer
1	Kind of support for a computer user	tech
2	Kind of computer connection	wifi
3	Computer connection	port
4	Comb users	bees
5	Traveling bag	grip

Table 1: Clue ranking for the query: *Kind of connection for traveling computer users (wifi)*

4.2 Relational shallow tree representation

The structures we adopt are similar to those defined in (Severyn et al., 2013b). They are essentially shallow syntactic trees built from POS tags grouped into chunks. Each clue and its answer candidate (either a TS or clue) are encoded into a tree having word lemmas at the leaves and POS tags as pre-terminals. The higher tree level organizes POS tags into chunks. For example, the upper tree of Figure 2, shows a shallow tree for the clue: *Hamlet's "To be, or not to be" addressee*, whereas the lower tree represents a retrieved TS containing the answer, *himself*: *The most obvious example is Hamlet's "To be or not to be ... the monologue is addressed by a character either to himself or to the audience.*

Additionally, we use a special REL tag to link the clue/snippet trees above such that structural relations will be captured by tree fragments. The links are established as follows: words from a clue and a snippet sharing a lemma get their parents (POS tags) and grandparents, i.e., chunk labels, marked by a prepending REL tag. We build such structural representations for both snippet and similar clue reranking tasks.

4.3 Tree kernels

We briefly report the different types of kernels (see, e.g., (Moschitti, 2006) for more details).

Syntactic Tree Kernel (STK), also known as a subset tree kernel (Collins and Duffy, 2002), maps objects in the space of all possible tree fragments constrained by the rule that the sibling nodes from their parents cannot be separated. In other words, substructures are composed by atomic building blocks corresponding to nodes along with all of their direct children. These, in case of a syntactic parse tree, are complete production rules of the associated parser grammar.

STK$_b$ extends STK by allowing leaf nodes to be part of the feature space. Leaf in syntactic trees are words, from this the subscript b (bag-of-words).

Subtree Kernel (SbtK) is one of the simplest tree kernels as it only generates complete subtrees, i.e., tree fragments that, given any arbitrary starting node, necessarily include all its descendants.

Partial Tree Kernel (PTK) (Moschitti, 2006) can be effectively applied to both constituency and dependency parse trees. It generates all possible connected tree fragments, e.g., sibling nodes can also be separated and be part of different tree fragments. In other words, a fragment is any possible tree path, from whose nodes other tree paths can depart. Thus, it can generate a very rich feature space resulting in higher generalization ability.

4.4 Snippet reranking

The task of snippet reranking consists in reordering the list of snippets retrieved from the search engine such that those containing the correct answer can be pushed at the top of the list. For this purpose, we transform the target clue in a search query and retrieve candidate text snippets. In our training set, these candidate text snippets are considered as positive examples if they contain the answer to the target clue.

We rerank snippets using preference reranking approach (see, e.g., (Shen and Joshi, 2005)). This means that two snippets are compared to derive which one is the best, i.e., which snippet contains the answer with higher probability. Since we aim at using kernel methods, we apply the following preference kernel:

$$P_K(\langle s_1, s_2\rangle, \langle s'_1, s'_2\rangle) = K(s_1, s'_1) +$$

$$+K(s_2, s'_2) - K(s_1, s'_2) - K(s_2, s'_1),$$

where s_r and s'_r refer to two sets of candidates associated with two rankings and K is a kernel applied to pairs of candidates. We represent the latter as pairs of clue and snippet trees. More formally, given two candidates, $s_i = \langle s_i(c), s_i(s)\rangle$ and $s'_i = \langle s'_i(c), s'_i(s)\rangle$, whose members are the clue and snippet trees, we define

$$K(s_i, s'_i) = TK(s_i(c), s'_i(c)) + TK(s_i(s), s'_i(s)),$$

where TK can be any tree kernel function, e.g., STK or PTK. Finally, it should be noted that, to add traditional feature vectors to the reranker, it is enough to add the product $(\vec{x}_{s_1} - \vec{x}_{s_2}) \cdot (\vec{x}_{s'_1} - \vec{x}_{s'_2})$ to the structural kernel P_K, where $\vec{x}_s$ is the feature vector associated with the snippet s.

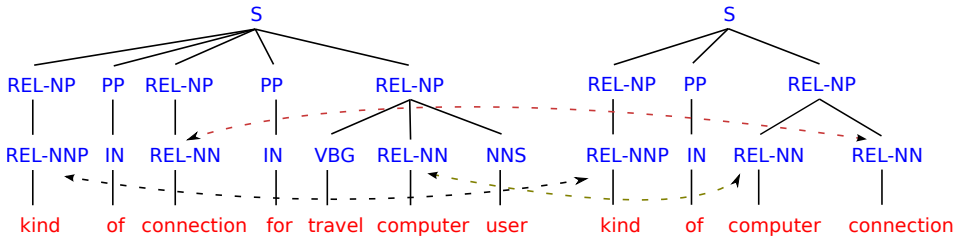

Figure 3: Two similar clues leading to the same answer.

4.5 Similar clue reranking

WebCrow creates answer lists by retrieving clues from the DB of previously solved crosswords. It simply uses the classical SQL operator and full-text search. We instead verified the hypothesis that a search engine could achieve a better result. Thus we opted for indexing the DB clues and their answers with the open source search engine Lucene (McCandless et al., 2010), using the state-of-the-art BM25 retrieval model. This alone significantly improved the quality of the retrieved clue list, which could be further refined by applying reranking. The latter consists in (i) retrieving a list of similar clues using a search engine and (ii) moving those more similar, which more probably contain the same answer to the clue query, at the top. For example, Table 1 shows the first five clues, retrieved for a query built from the clue: *Kind of connection for traveling computer users*. The search engine retrieves the wrong clue, *Kind of connection for traveling computer users*, at the top since it overlaps more with the query.

To solve these kinds of problems by also enhancing the generalization power of our reranking algorithm, we use a structural representation similar to the one for TS that we illustrated in the previous section. The main difference with the previous models is that the reranking pair is only constituted by clues. For example, Fig. 3 shows the representation of the pairs constituted by the query clue and the correct clue ranked in the second position (see Table 1). The relational arrows suggest a syntactic transformation from `connection for * computer` to `computer connection`, which can be used by the reranker to prefer the correct clue to the wrong one. Note that such transformation corresponds to the pair of tree fragments: `[S [REL-NP[REL-NN]][PP][NP[VBG][REL-NN]]]` → `[S [REL-NP[REL-NN][REL-NN]]]`, where the node pairs, ⟨`REL-NN`, `REL-NN`⟩ define the arguments of the syntactic transformation. Such fragments can be generated by PTK, which can thus be used for learning clue paraphrasing.

To build the reranking training set, we used the training clues for querying the search engine, which draws candidates from the indexed clues. We stress the fact that this set of clues is disjoint from the clues in the training and test sets. Thus, identical clues are not present across sets. At classification time, the new clue is used as a search query. Similar candidate clues are retrieved and used to form pairs.

4.6 Feature Vectors

In addition to structural representations, we also used features for capturing the degrees of similarity between clues within a pair.

DKPro Similarity. We used similarity features from a top performing system in the Semantic Textual Similarity (STS) task, namely DKPro from the UKP Lab (Bär et al., 2013). These features were effective in predicting the degree of similarity between two sentences. DKPro includes the following syntactic similarity metrics, operating on string sequences, and more advanced semantic similarities:

– *Longest common substring measure* (Gusfield, 1997). It determines the length of the longest substring shared by two text segments.

– *Longest common subsequence measure* (Allison and Dix, 1986). It extends the notion of substrings to word subsequences by applying word insertions or deletions to the original input text pairs.

– *Running-Karp-Rabin Greedy String Tiling* (Wise, 1996). It provides a similarity between two sentences by counting the number of shuffles in their subparts.

– *Resnik similarity* (Resnik, 1995). The WordNet hypernymy hierarchy is used to compute a measure of semantic relatedness between concepts expressed in the text. The aggregation algorithm by Mihalcea et al. (Mihalcea et al., 2006) is applied to extend the measure from words to sentences.

– *Explicit Semantic Analysis* (ESA) similarity

(Gabrilovich and Markovitch, 2007). It represents documents as weighted vectors of concepts learned from Wikipedia, WordNet and Wiktionary.
– *Lexical Substitution* (Biemann, 2013). A supervised word sense disambiguation system is used to substitute a wide selection of high-frequency English nouns with generalizations. Resnik and ESA features are then computed on the transformed text.

New features. Hereafter, we describe new features that we designed for CP reranking tasks.
– *Feasible Candidate*. It is a binary feature signaling the presence or absence of words with the same length of the clue answer (only used for snippet reranking).
– *Term overlap features*. They compute the cosine similarity of text pairs encoded into sets of n-grams extracted from different text features: surface forms of words, lemmas and POS-tags. They are computed keeping and removing stop-words. They complement DKPro features.
– *Kernel similarities*. These are computed using (i) string kernels applied to sentences, or PTK applied to structural representations with and without embedded relational information (REL). This similarity is computed between the members of a $\langle clue, snippet \rangle$ or a $\langle clue, clue \rangle$ pair.

5 Experiments

Our experiments aim at demonstrating the effectiveness of our models on two different tasks: (i) Snippet Reranking and (ii) Similar Clue Retrieval (SCR). Additionally, we measured the impact of our best model for SCR in the WebCrow system by comparing with it. Our referring database of clues is composed by 1,158,202 clues, which belong to eight different crossword editors (downloaded from the Web[6]). We use the latter to create one dataset for snippet reranking and one dataset for clues retrieval.

5.1 Experimental Setup

To train our models, we adopted SVM-light-TK[7], which enables the use of structural kernels (Moschitti, 2006) in SVM-light (Joachims, 2002), with default parameters. We applied a polynomial kernel of degree 3 to the explicit feature vectors, as we believe feature combinations can be valuable. To measure the impact of the rerankers as well as the baselines, we used well known metrics for assessing the accuracy of QA and retrieval systems, i.e.: Recall at rank 1 (R@1 and 5), Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), the average Recall (AvgRec). R@k is the percentage of questions with a correct answer ranked at the first position. MRR is computed as follows: $MRR = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{rank(q)}$, where *rank(q)* is the position of the first correct answer in the candidate list. For a set of queries Q, MAP is the mean over the average precision scores for each query: $\frac{1}{Q}\sum_{q=1}^{Q} AveP(q)$. AvgRec and all the measures are evaluated on the first 10 retrieved snippets/clues. For training and testing the reranker, only the first 10 snippets/clues retrieved by the search engine are used.

Model	MAP	MRR	AvgRec	REC@1	REC@5
Bing	16.00	18.09	69.00	12.50	24.80
V	18.00	19.88	76.00	14.20	26.10
SbtK	17.00	19.6	75.00	13.80	26.40
STK	18.00	20.44	76.00	15.10	27.00
STK_b	18.00	20.68	76.00	15.30	27.40
PTK	19.00	21.65	77.00	16.10	28.70
V+SbtK	**20.00**	**22.39**	**80.00**	**17.20**	29.10
V+STK	19.00	20.82	78.00	14.90	27.90
STK_b	19.00	21.20	79.00	15.60	28.40
V+PTK	19.00	21.68	79.00	16.00	**29.40**
V+DK	18.00	20.48	77.00	14.60	26.80
V+DK+SbtK	**20.00**	22.29	**80.00**	16.90	28.70
V+DK+STK	19.00	21.47	79.00	15.50	28.30
V+DK+STK_b	19.00	21.58	79.00	15.4	28.60
V+DK+PTK	**20.00**	22.24	**80.00**	16.80	29.30

Table 2: Snippet reranking

Model	MAP	MRR	AvgRec	REC@1	REC@5
MB25	69.00	73.78	80.00	62.11	81.23
WebCrow	-	53.22	58.00	39.60	62.85
SbtK	52.00	54.72	69.00	36.50	64.05
STK	63.00	68.21	77.00	54.57	76.11
STK_b	63.00	67.68	77.00	53.85	75.63
PTK	65.00	70.12	78.00	57.39	77.65
V+SbtK	68.00	73.26	80.00	60.95	81.28
V+STK	71.00	76.01	82.00	64.58	83.95
V+STK_b	70.00	75.68	82.00	63.95	83.77
V+PTK	71.00	76.67	82.00	65.67	84.07
V+DK	71.00	76.76	81.00	65.55	84.29
V+DK+SbtK	72.00	76.91	82.00	65.87	84.51
V+DK+STK	**73.00**	**78.37**	**84.00**	**67.83**	**85.87**
V+DK+STK_b	**73.00**	78.29	**84.00**	67.71	85.77
V+DK+PTK	**73.00**	78.13	83.00	67.39	85.75

Table 3: Reranking of similar clues.

5.2 Snippet Reranking

The retrieval from the Web is affected by a significant query processing delay, which prevents us

[6] http://www.crosswordgiant.com
[7] http://disi.unitn.it/moschitti/Tree-Kernel.htm

to use entire documents. Thus, we only considered the text from Bing snippets. Moreover, since our reranking approach does not include the treatment of special clues such as anagrams or linguistic games, e.g., *fill-in-the blank clues*, we have excluded them by our dataset. We crawled the latter from the Web. We converted each clue into a query and downloaded the first 10 snippets as result of a Bing query. In order to reduce noise from the data, we created a black list containing URLs that must not be considered in the download phase, e.g., crossword websites. The training set is composed by 20,000 clues while the test set comprises 1,000 clues.

We implemented and compared many models for reranking the correct snippets higher, i.e., containing the answer to the clue. The compared systems are listed on the first column of Table 2, where: V is the approach using the vector only constituted by the new feature set (see Sec. 4.6); DK is the model using the features made available by DKPro; the systems ending in TK are described in Sec. 4.3; and the plus operator indicates models obtained by summing the related kernels.

Depending on the target measure they suggest slightly different findings. Hereafter, we comment on MRR as it is the most interesting from a ranking viewpoint. We note that: (i) Bing is improved by the reranker based on the new feature vector by 2 absolute points; (ii) DK+V improves on V by just half point; (iii) PTK provides the highest result among individual systems; (iv) combinations improve on the individual systems; and (v) overall, our reranking improves on the ranking of paragraphs of Bing by 4 points in MRR and 5 points in accuracy on the first candidate (REC@1), corresponding to about 20% and 50% of relative improvement and error reduction, respectively.

5.3 Similar clue retrieval

We compiled a crossword database of 794,190 unique pairs of clue-answer. Using the clues contained in this set, we created three different sets: training and test sets and the database of clues. The database of clues can be indexed for retrieving similar clues. It contains 700,000 unique clue-answer pairs. The training set contains 39,504 clues whose answer may be found in database. Using the same approach, we created a test set containing 5,060 clues that (i) are not in the training set and (ii) have at least an answer in the database.

Model	MRR	REC@1	REC@5	REC@10
WebCrow	41.00	33.00	51.00	58.00
Our Model	46.00	39.00	56.00	59.00

Table 4: Performance on the word list candidates averaged over the clues of 10 entire CPs

Model	%Correct words	%Correct letters
WebCrow	34.45	49.72
Our Model	39.69	54.30

Table 5: Performance given in terms of correct words and letters averaged on the 10 CPs

We experimented with all models, as in the previous section, trained for the similar clue retrieval task. However, since WebCrow includes a database module, in Tab. 3, we have an extra row indicating its accuracy. We note that: (i) BM25 shows a very accurate MRR, 73.78%. It largely improves on WebCrow by about 20.5 absolute percent points, demonstrating the superiority of an IR approach over DB methods. (ii) All TK types do not improve alone on BM25, this happens since they do not exploit the initial rank provided by BM25. (iii) All the feature vector and TK combinations achieve high MRR, up to 4.5 absolute percent points of improvement over BM25 and thus 25 points more than WebCrow, corresponding to 53% of error reduction. Finally, (iv) the relative improvement on REC@1 is up to 71% (28.23% absolute). This high result is promising in the light of improving WebCrow for the end task of solving complete CPs.

5.4 Impact on WebCrow

In these experiments, we used our reranking model of similar clues (more specifically, the V+DK+STK model) using 10 complete CPs (for a total of 760 clues) from the New York Times and Washington Post. This way, we could measure the impact of our model on the complete task carried out by WebCrow. More specifically, we give our reranked list of answers to WebCrow in place of the list it would have extracted with the CWDB module. It should be noted that to evaluate the impact of our list, we disabled WebCrow access to other lists, e.g., dictionaries. This means that the absolute resolution accuracy of WebCrow using our and its own lists can be higher (see (Ernandes et al., 2008) for more details).

The first result that we derive is the accuracy of the answer list produced from the new data, i.e., constituted by the 10 entire CPs. The results are reported in Tab. 4. We note that the improvement of our model is lower than before as a non-negligible percentage of clues are not solved using the clue DB. However, when we compute the accuracy in solving the complete CPs, the impact is still remarkable as reported by Tab. 5. Indeed, the results show that when the lists reordered by our reranker are used by WebCrow, the latter improves by more than 5 absolute percent points in both word and character accuracy.

6 Conclusions

In this paper, we improve automatic CP resolution by modeling two innovative reranking tasks for: (i) CP answer list derived from Web search and (ii) CP clue retrieval from clue DBs.

Our rankers are based on SVMs and structural kernels, where the latter are applied to robust shallow syntactic structures. Our model applied to clue reranking is very interesting as it allows us to learn clue paraphrasing by exploiting relational syntactic structures representing pairs of clues.

For our study, we created two different corpora for Snippet Reranking Dataset and Clue Similarity Dataset on which we tested our methods. The latter improve on the lists generated by WebCrow by 25 absolute percent points in MRR (about 53% of relative improvement). When such improved lists are used in WebCrow, its resolution accuracy increases by 15%, demonstrating that there is a large room for improvement in automatic CP resolution. In the future, we would like to add more semantic information to our rerankers and include an answer extraction component in the pipeline.

Acknowledgments

We are deeply in debt with Marco Gori and Marco Ernandes for making available WebCrow, for helping us with their system and for the useful technical discussion regarding research directions. This research has been partially supported by the EC's Seventh Framework Programme (FP7/2007-2013) under the grants #288024: LIMOSINE – Linguistically Motivated Semantic aggregation engiNes. Many thanks to the anonymous reviewers for their valuable work.

References

Elif Aktolga, James Allan, and David A. Smith. 2011. Passage reranking for question answering using syntactic structures and answer types. In *ECIR*.

L Allison and T I Dix. 1986. A bit-string longest-common-subsequence algorithm. *Inf. Process. Lett.*, 23(6):305–310, December.

Daniel Bär, Torsten Zesch, and Iryna Gurevych. 2013. Dkpro similarity: An open source framework for text similarity. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (System Demonstrations) (ACL 2013)*, pages 121–126, Stroudsburg, PA, USA, August. Association for Computational Linguistics.

Chris Biemann. 2013. Creating a system for lexical substitutions from scratch using crowdsourcing. *Lang. Resour. Eval.*, 47(1):97–122, March.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 263–270, Stroudsburg, PA, USA. Association for Computational Linguistics.

Marco Ernandes, Giovanni Angelini, and Marco Gori. 2005. Webcrow: A web-based system for crossword solving. In *In Proc. of AAAI 05*, pages 1412–1417. Menlo Park, Calif., AAAI Press.

Marco Ernandes, Giovanni Angelini, and Marco Gori. 2008. A web-based agent challenges human experts on crosswords. *AI Magazine*, 29(1).

David Ferrucci and Adam Lally. 2004. Uima: An architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10(3-4):327–348, September.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. 2010a. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3).

David A. Ferrucci, Eric W. Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John M. Prager, Nico Schlaefer, and Christopher A. Welty. 2010b. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79.

Evgeniy Gabrilovich and Shaul Markovitch. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 1606–1611, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Matthew L. Ginsberg. 2011. Dr.fill: Crosswords and an implemented solver for singly weighted csps. *J. Artif. Int. Res.*, 42(1):851–886, September.

Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA.

R Herbrich, T Graepel, and K Obermayer. 2000. Large margin rank boundaries for ordinal regression. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 115–132, Cambridge, MA. MIT Press.

Jiwoon Jeon, W. Bruce Croft, and Joon Ho Lee. 2005. Finding similar questions in large question and answer archives. In *CIKM*.

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA. ACM.

Boris Katz and Jimmy Lin. 2003. Selectively using relations to improve precision in question answering.

Michael L. Littman, Greg A. Keim, and Noam Shazeer. 2002. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134(12):23 – 55.

Michael McCandless, Erik Hatcher, and Otis Gospodnetic. 2010. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., Greenwich, CT, USA.

Rada Mihalcea, Courtney Corley, and Carlo Strapparava. 2006. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 775–780. AAAI Press.

Alessandro Moschitti, Daniele Pighin, and Roberto Basili. 2006. Semantic role labeling via tree kernel joint inference. In *Proceedings of CoNLL-X*, New York City.

Alessandro Moschitti, Silvia Quarteroni, Roberto Basili, and Suresh Manandhar. 2007. Exploiting syntactic and shallow semantic kernels for question/answer classification. In *ACL*.

Alessandro Moschitti. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *ECML*, pages 318–329.

Alessandro Moschitti. 2008. Kernel methods, syntax and semantics for relational text categorization. In *CIKM*.

Ira Pohl. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(34):193 – 204.

Filip Radlinski and Thorsten Joachims. 2006. Query chains: Learning to rank from implicit feedback. *CoRR*.

Philip Resnik. 1995. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 448–453, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Aliaksei Severyn and Alessandro Moschitti. 2012. Structural relationships for large-scale learning of answer re-ranking. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval (SIGIR)*, pages 741–750. ACM.

Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. 2013a. Building structures from classifiers for passage reranking. In *CIKM*, pages 969–978.

Aliaksei Severyn, Massimo Nicosia, and Alessandro Moschitti. 2013b. Learning adaptable patterns for passage reranking. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 75–83, Sofia, Bulgaria, August. Association for Computational Linguistics.

Libin Shen and Aravind K. Joshi. 2005. Ranking and reranking with perceptron. *Machine Learning*, 60(1-3):73–96.

D. Shen and M. Lapata. 2007. Using semantic roles to improve question answering. In *EMNLP-CoNLL*.

M. Surdeanu, M. Ciaramita, and H. Zaragoza. 2008. Learning to rank answers on large online QA collections. In *Proceedings of ACL-HLT*.

Michael J. Wise. 1996. Yap3: Improved detection of similarities in computer program and other texts. In *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '96, pages 130–134, New York, NY, USA. ACM.