SACRY: Syntax-based Automatic Crossword puzzle Resolution sYstem

Gianni Barlacchi[†], **Massimo Nicosia**[†] and **Alessandro Moschitti**
[†]Dept. of Computer Science and Information Engineering, University of Trento
38123 Povo (TN), Italy
ALT, Qatar Computing Research Institute, Hamad Bin Khalifa University
5825 Doha, Qatar
{gianni.barlacchi, m.nicosia, amoschitti}@gmail.com

Abstract

In this paper, we present our Crossword Puzzle Resolution System (SACRY), which exploits syntactic structures for clue reranking and answer extraction. SACRY uses a database (DB) containing previously solved CPs in order to generate the list of candidate answers. Additionally, it uses innovative features, such as the answer position in the rank and aggregated information such as the min, max and average clue reranking scores. Our system is based on WebCrow, one of the most advanced systems for automatic crossword puzzle resolution. Our extensive experiments over our two million clue dataset show that our approach highly improves the quality of the answer list, enabling the achievement of unprecedented results on the complete CP resolution tasks, i.e., accuracy of 99.17%.

1 Introduction

Crossword Puzzles (CPs) are the most famous language games played around the world. The automatic resolution of CPs is an open challenge for the artificial intelligence (AI) community, which mainly employs AI techniques for filling the puzzle grid with candidate answers. Basic approaches try to optimize the overall probability of correctly filling the grid by exploiting the likelihood of each candidate answer, while satisfying the grid constraints.

Previous work (Ernandes et al., 2005) clearly suggests that providing the solver with an accurate list of answer candidates is an important step for the CP resolution task. These can be retrieved using (i) the Web, (ii) Wikipedia, (iii) dictionaries or lexical databases, e.g., WordNet or, (iv) most importantly, recuperated from the DBs of previously solved CP. Indeed, CPs are often created reusing the same clues of past CPs, and thus querying the DB with the target clue allows for recuperating the same (or similar) clues of the target one. It is interesting to note that, for this purpose, all previous automatic CP solvers use standard DB techniques, e.g., SQL Full-Text query. Existing systems for automatic CP resolution, such as Proverb (Littman et al., 2002) and Dr. Fill (Ginsberg, 2011), use several different modules for generating candidate answer lists. These are merged and used for defining a Constraint Satisfaction Problem, resolved by the CP solver.

Our CP system, SACRY, is based on innovative QA methods for answering CP clues. We employ (i) state-of-the-art IR techniques to retrieve the correct answer by querying the DB of previously solved CPs, (ii) learning to rank methods based on the syntactic structure of clues and structural kernels to improve the ranking of clues that can potentially contain the answers and (iii) an aggregation algorithm for generating the final list containing unique candidate answers. We implemented a specific module based on these approaches and we plugged it into an automatic CP solver, namely WebCrow (Ernandes et al., 2005). The latter is one of the best systems for CP resolution and it has been kindly made available by the authors.

We tested our models on a dataset containing more than two million clues and their associated answers. This dataset is an interesting resource that we will make available to the research community. It can be used for tasks such as: (i) similar clue retrieval/reranking, which focuses on improving the rank of clues c_i retrieved by a search engine, and (ii) answer reranking, which targets the list of a_{c_i}, i.e., their aggregated clues. We tested SACRY on an end-to-end task by solving ten crossword puzzles provided by two of the most famous CP editors from The New York Times and the Washington Post. SACRY obtained an impressive CP resolution accuracy of 99.17%.

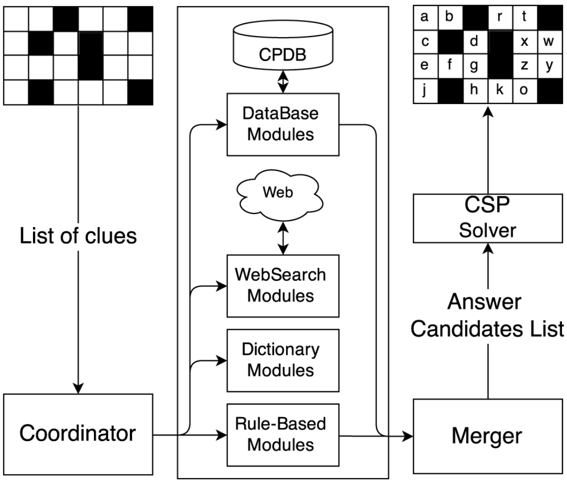

Figure 1: The architecture of WebCrow

In the reminder of this paper, Sec. 2 introduces WebCrow and its architecture. Our models for similar clues retrieval and answers reranking are described in Sec. 3 while Sec. 4 illustrates our experiments. Finally, the conclusions and directions for future work are presented in Sec. 5.

2 The WebCrow Architecture

Our approaches can be used to generate accurate candidate lists that CP solvers can exploit to improve their overall accuracy. Therefore, the quality of our methods can be also implicitly evaluated on the final resolution task. For this purpose, we use an existing CP solver, namely WebCrow, which is rather modular and accurate and it has been kindly made available by the authors. Its architecture is illustrated in Figure 1. In the following, we briefly introduce the database module of WebCrow, which is the one that we substituted with ours.

WebCrow's solving process can be divided in two phases. In the first phase, the input list of clues activate a set of answer search modules, which produce several lists of possible solutions for each clue. These lists are then merged by a specific *Merger* component, which uses the confidence values of the lists and the probability that a candidate in a list is correct. Eventually, a single list of answers with their associated probabilities is built for each input clue. In the second phase WebCrow fills the crossword grid by solving a constraint-satisfaction problem. WebCrow selects a single answer from each merged list of candidates, trying to satisfy the imposed constraints. The goal of this phase is to find an admissible solution that maximizes the number of correctly inserted words. It is done using an adapted version of the WA* algorithm (Pohl, 1970) for CP resolution.

2.1 CrossWord Database module (CWDB)

Gathering clues contained in previously published CPs is essential for solving new puzzles. A large portion of clues in new CPs has usually already appeared in the past. Clues may share similar wording or may be restated in a very different way. Therefore, it is important to identify the clues that have the same answer. WebCrow uses three different modules to retrieve clues identical or similar to a given clue from the database: the CWDB-EXACT module, which retrieves DB clues that match exactly with a target clue, and weights them by the frequency they have in the clue collection. The CWDB-PARTIAL module, which uses the MySQL's partial matching function, query expansion and positional term distances to compute clue-similarity scores, along with the Full-Text search functions. The CWDB-DICTIO module, which simply returns the full list of words of correct length, ranked by their number of occurrences in the initial list.

We outperform the previous approach by applying learning-to-rank algorithms based on SVMs and tree kernels on clue lists generated by state-of-the-art passage retrieval systems.

3 Crossword Puzzle Database (CPDB) Module

WebCrow creates answer lists by retrieving clues from the DB of previously solved crosswords. It simply uses the classical SQL operators and full-text search. We instead index 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 improves the quality of the retrieved clue list, which is further refined by applying reranking. The latter consists in promoting the clues that potentially have the same answer of the query clue.

We designed a relatively complex pipeline shown in Fig. 2. We built a training set using some training clues for querying our search engine, which retrieves correct and incorrect candidates from the indexed clues. At classification time, the new clues are used as a search query and the retrieved similar candidate are reranked by our models. The next sections show our approach for building rerankers that can exploit structures for solving the ineffectiveness of the simple word representation.

3.1 Reranking framework based on Kernels

The basic architecture of our reranking framework is relatively simple: it uses a standard preference kernel reranking approach and is similar to the one we proposed in (Severyn and Moschitti, 2012) for QA tasks. However, heare, we modeled different kernels suitable for clue retrieval.

The framework takes a query clue and retrieves a list of related candidate clues using a search engine (applied to the CPDB), according to some similarity criteria. Then, the query and the candidates are processed by an NLP pipeline. Our pipeline is built on top of the UIMA framework (Ferrucci and Lally, 2004) and contains many text analysis components. The components used for our specific tasks are: the tokenizer[1], sentence detector[1], lemmatizer[1], part-of-speech (POS) tagger[1] and chunker[2].

The annotations produced by these standard processors are input to our components that extract structures as well as traditional features for representing clues. This representation is 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 structures and consequently the accuracy of our learning to rank algorithms, we preferred to use shallow syntactic trees over full syntactic representations.

We used the Partial Tree Kernel (PTK) (Moschitti, 2006) to build our reranker. Given an input tree, PTK 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 (which is a feature) 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.

We combined the structural features with other traditional ones. We used the following groups:

iKernels features (iK), which include similarity features and kernels applied intra-pairs, i.e., between the query and the retrieved clues:

– *Syntactic similarities*, i.e., cosine similarity measures computed on n-grams (with $n = 1, 2, 3, 4$) of word lemmas and part-of-speech tags.

– *Kernel similarities*, i.e., string kernels and tree kernels applied to structural representations.

DKPro Similarity (DKP), which defines features used in the Semantic Textual Similarity (STS) challenge. These are encoded by the UKP Lab (Bär et al., 2013):

– *Longest common substring measure* and *Longest common subsequence measure*. They determine the length of the longest substring shared by two text segments.

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

– *Resnik similarity*. The WordNet hierarchy is used to compute a measure of semantic relatedness between concepts expressed in the text.

The aggregation algorithm in (Mihalcea et al., 2006) is applied to extend the measure from words to sentences.

– *Explicit Semantic Analysis* (ESA) similarity (Gabrilovich and Markovitch, 2007), which 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 computed on the transformed text.

WebCrow features (WC), which are the similarity measures computed on the clue pairs by WebCrow (using the Levenshtein distance) and the Search Engine score.

Kernels for reranking, given a query clue q_c and two retrieved clues c_1, c_2, we can rank them by using a reranking model, namely (**RR**). It uses two pairs $P = \langle p_q^1, p_q^2 \rangle$ and $P' = \langle p_{q'}^1, p_{q'}^2 \rangle$, the member of each pair are clues from the same list generated by q and q', respectively. In this case, we use the kernel, $K_{RR}(P, P') = PTK(\langle q, c_1 \rangle, \langle q', c_1' \rangle) + PTK(\langle q, c_2 \rangle, \langle q', c_2' \rangle) - PTK(\langle q, c_1 \rangle, \langle q', c_2' \rangle) - PTK(\langle q, c_2 \rangle, \langle q', c_1' \rangle)$, which corresponds to the scalar product between the vectors, $\big(\phi(p_q^1) - \phi(p_q^2)\big) \cdot \big(\phi(p_{q'}^1) - \phi(p_{q'}^2)\big)$, in the fragment vector space generated by PTK.

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

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

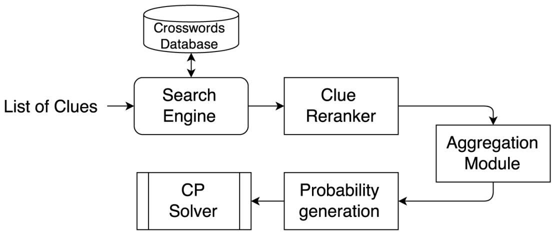

Figure 2: The architecture of our system

3.2 Learning to rank aggregated answers

Groups of similar clues retrieved from the search engine can be associated with the same answers. Since each clue receives a score from the reranker, a strategy to combine the scores is needed. We aim at aggregating clues associated with the same answer and building meaningful features for such groups. For this purpose, we train an SVM^{rank} with each candidate answer a_{c_i} represented with features derived from all the clues c_i associated with such answer, i.e., we aggregate them using standard operators such as average, min. and max.

We model an answer a using its set of clues $C_a = \{c_i : a_{c_i} = a\}$ in SVM^{rank}. The feature vector associated with a must contains information from all $c \in C_a$. Thus, we designed novel aggregated features that we call **AVG**: (i) we average the feature values used for each clue by the first reranker, i.e., those described in Sec. 3.1 as well as the scores produced by the clue reranker. More specifically, we compute their sum, average, maximum and minimum values. (ii) We also add the term frequency of the answer word in CPDB.

Additionally, we model the occurrences of the answer instance in the list by means of positional features: we use n features, where n is the size of our candidate list (e.g., 10). Each feature corresponds to the positions of each candidate and it is set to the reranker score if $c_i \in C_a$ (i.e., for the target answer candidate) and 0 otherwise. We call such features (**POS**). For example, if an answer candidate is associated with clues appearing at positions 1 and 3 of the list, Feature 1 and Feature 3 will be set to the score calculated from the reranker. We take into account the similarity between the answer candidate and the input clues using a set of features, derived from word embeddings (Mikolov et al., 2013). These features consider (i) the similarities between the clues in a pair, (ii) the target clue and the candidate answer and (iii) the candidate clue and the candidate answer. They are computed summing the embedding vectors of words and computing the cosine similarity. This way we produce some evidence of semantic relatedness. We call such features (**W2V**).

3.3 Generating probabilities for the solver

The aggregation and reranking steps provide as a set of unique candidate answers ordered by their reranking scores. Using the raw scores in WebCrow generally produces a decrease of its accuracy since it expects probabilities (or values ranging from 0 to 1). The summed votes or the scores produced by the reranker can have a wider range and can also be negative. Therefore, we apply logistic regression (**LGR**) to learn a mapping between the reranking scores and values ranging from 0 to 1.

4 Experiments

In our experiments we compare our approach with WebCrow both on ranking candidate answers and on the end-to-end CP resolution.

4.1 Database of previously resolved CPs

The most commonly used databases of clues contain both single clues taken from various crosswords (Ginsberg, 2011) and entire crossword puzzle (Ernandes et al., 2008). They refer to relatively clean pairs of clues/answers and other crossword related information such as date of the clue, name of the CP editor and difficulty of the clue (e.g., clues taken from the CPs published on *The Sunday* newspaper are the most difficult). Unfortunately, they are not publicly available.

Therefore, we compiled a crossword corpus combining (i) CPs downloaded from the Web[3] and (ii) the clue database provided by Otsys[4]. We removed duplicates, fill-in-the-blank clues (which can be better solved by using other strategies) and clues representing anagrams or linguistic games. We collected over 6.3 millions of clues, published by many different American editors. Although this is a very rich database, it contains many duplicates and non-standard clues, which introduce significant noise in the dataset. For this reason we created a filtered dataset of 2,131,034 unique and standard clues, with associated answers. It excludes the *fill-in-the-blank* clues mentioned above.

4.2 Experimental Setup

To train our models, we adopted SVM-light-TK[5], which enables the use of the Partial Tree Kernel

[3] http://www.crosswordgiant.com

[4] http://www.otsys.com/clue

[5] `http://disi.unitn.it/moschitti/Tree-Kernel.htm`

(PTK) (Moschitti, 2006) in SVM-light (Joachims, 2002), with default parameters. We applied a polynomial kernel of degree 3 to the explicit feature vectors (FV). To measure the impact of the rerankers as well as the CWDB module, we used well-known metrics for assessing the accuracy of QA and retrieval systems, i.e.: Recall at different ranks (R@1, 5, 20, 50, 100), Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP). R@1 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: $\frac{1}{Q} \sum_{q=1}^{Q} AveP(q)$.

To measure the complete CP resolution task, we use the accuracy over the entire words filling a CP grid (one wrong letter causes the entire definition to be incorrect).

4.3 Clue reranking experiments

Given an input clue, BM25 retrieves a list of 100 clues. On the latter, we tested our different models for clue reranking. For space constraints, we only report a short summary of our experiments: kernel-based rerankers combined with *traditional features* (PTK+FV) relatively improve standard IR by 16%. This is an interesting result as in (Barlacchi et al., 2014), we showed that standard IR greatly improves on the DB methods for clue retrieval, i.e., we showed that BM25 relatively improves on SQL by about 40% in MRR.

4.4 Answer aggregation and reranking

Reranking clues is just the first step as the solver must be fed with the list of unique answers. Thus, we first used our best model (i.e., PTK+FV) for clue reranking to score the answers of a separate set, i.e., our answer reranking training set. Then, we used these scores to train an additional reranker for aggregating identical answers. The aggregation module merges clues sharing the same answer into a single instance.

Models	MRR	R@1	R@5	R@10	R@20	R@50	R@80
WebCrow	39.12	31.51	47.37	54.38	58.60	63.34	64.06
Raw voting	41.84	33.0	52.9	58.7	62.7	66.6	67.5
LGR voting	43.66	35	53.7	59.3	63.4	67.4	67.7
SVM^{rank}							
AVG	43.5	35.3	53.5	59.4	63.9	67.4	67.7
AVG+POS	44.1	36.3	53.6	58.9	63.9	67.4	67.6
AVG+W2V	43.2	35	53.3	58.8	63.9	67.4	67.7
AVG+POS+FV	44.4	36.7	54.2	60	64.3	67.4	67.7
AVG+FV+W2V	44.1	35.8	54.4	60	64.4	67.4	67.7
AVG+POS+ FV+W2V	**44.6**	**36.8**	**54.2**	**59.8**	**64.4**	**67.4**	**67.7**

Table 1: Answer reranking on the dev. set.

Tab. 1 shows the results for several answer reranking models tested on a development set: the first row shows the accuracy of the answer list produces by WebCrow. The second row reports the accuracy of our model using a simple voting strategy, i.e., the score of the clue reranker is used as a vote for the target candidate answer. The third row applies Logistic Regression (LGR) to transform the SVM reranking scores in probabilities. It uses Lucene score for the candidate answer as well as the max and min scores of the entire list. From the fourth column, the answer reranker is trained using SVM^{rank} using FV, AVG, POS, W2V and some of their combinations. We note that: (i) voting the answers using the raw score improves WebCrow but the probabilities computed by LGR perform much better, i.e., about 2 percent points better than Raw voting and 4.5 points better than WebCrow; (ii) the SVM^{rank} aggregation model can provide another additional point, when positional features and standard feature vectors are used along with aggregated and W2C features. (iii) The overall relative improvement of 14% over WebCrow is promising for improving the end-to-end CP resolution task.

4.5 Evaluation of the CP resolution

In order to test the effectiveness of our method, we evaluated the resolution of full CPs. We selected five crosswords from *The New York Times* newspaper and other five from the *Washington Post*. Fig. 3 shows the average resolution accuracy over the ten CP of the original WebCrow compared to WebCrow using our reranked lists. We ran the solver by providing it with lists of different size. We note that our model consistently outperforms WebCrow. This means that the lists of candidate answers generated by our models help the solver, which in turn fills the grid with higher accuracy. In particular, our CP system achieves an average accuracy of 99.17%, which makes it competitive with international CP resolution challenges.

Additionally, WebCrow achieves the highest accuracy when it uses the largest candidate lists (both original or reranked). A large list size negatively impacts on the speed of the solver, which in a CP competition is critical to beat the other competitors (if participants obtain the same score, the solving time decides who is ranked first). Thus, our approach also provides a speedup as the best accuracy is reached for just 50 candidates (in con-

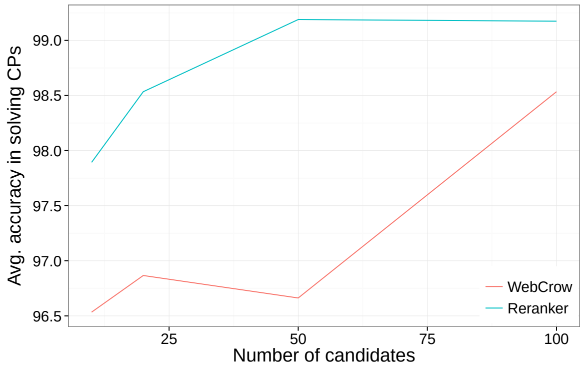

Figure 3: Average accuracy over 10 CPs.

trast with the 100 candidates needed by the original WebCrow).

5 Final Remarks

In this paper, we describe our system SACRY for automatic CP resolution. It is based on modeling rerankers for clue retrieval from DBs. It achieves a higher accuracy than WebCrow thanks to the use of rerankers based on SVMs and structural kernels, where the latter are applied to robust shallow syntactic structures. Our structural models applied to clue reranking enable us to learn clue paraphrasing by exploiting relational syntactic structures representing pairs of clues.

We collected the biggest clue dataset ever, which can be also used for QA tasks since it is composed by pairs of clue/answer. The dataset includes 2,131,034 unique pairs of clue/answers, which we are going to make available to the research community. The experiments show that our methods improve the quality of the lists generated by WebCrow by 14% in MRR. When used in WebCrow solver with its best setting, its resolution error relatively decreases by 50%, achieving almost a perfect resolution accuracy, i.e., 99.17%. In the future, we would like to release the solver to allow researchers to contribute to the project and make the system even more competitive.

Acknowledgments

This work has been supported by the EC project CogNet, 671625 (H2020-ICT-2014-2, Research and Innovation action). This research is part of the Interactive sYstems for Answer Search (IYAS) project, conducted by the ALT group at Qatar Computing Research Institute within the Hamad Bin Khalifa University and Qatar Foundation.

References

Daniel Bär, Torsten Zesch, and Iryna Gurevych. 2013. Dkpro similarity: An open source framework for text similarity. In *Proceedings of ACL*.

Gianni Barlacchi, Massimo Nicosia, and Alessandro Moschitti. 2014. Learning to rank answer candidates for automatic resolution of crossword puzzles. In *Proceedings of CoNLL*.

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

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.

Evgeniy Gabrilovich and Shaul Markovitch. 2007. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*.

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

Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of ACM SIGKDD*, New York, NY, USA. ACM.

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 AAAI*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

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

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

Aliaksei Severyn and Alessandro Moschitti. 2012. Structural relationships for large-scale learning of answer re-ranking. In *SIGIR*, New York, NY, USA.