Quick generation of crosswords using concatenation

Jakub Dakowski
Faculty of Psychology & Cognitive Sciences
Adam Mickiewicz University
Poznań, Poland
https://orcid.org/0000-0002-7811-4580

Piotr Jaworski
Faculty of Psychology & Cognitive Sciences
Adam Mickiewicz University
Poznań, Poland
piojaw@amu.edu.pl

Waldemar Wojna
Faculty of Law & Administration
Adam Mickiewicz University
Poznań, Poland
walwoj@st.amu.edu.pl

***Abstract*—We propose two crossword generation methods based on a crossword concatenation, word addition and crossword rotation operation. This can be viewed as an alternative to the method proposed by Bonomo, Lauf and Yampolskiy or Bulitko and Botea, who focus on generating matrices filled with letters and mutating them in order to make them into actual crosswords. The first one uses a combination of *first improvement* and *best improvement* local search methods. The choice on which one to use is made using the temperature calculated for a given turn. Second algorithm is a simulated annealing algorithm which uses *best improvement* search and word removal operation. The crosswords are evaluated using a goal function that includes the amount of intersections in a crossword and the density of letters in the crossword. Unfortunately, both of these solutions, while producing decent results, create puzzles unsolvable for humans in reasonable time. Because of that, we plan on implementing: a better goal function, targeted word removal and targeted word addition. We also plan to switch simulated annealing for a cuckoo search algorithm.**

***Index Terms*—artificial intelligence, optimization, crosswords, puzzles, local search, simulated annealing, cuckoo search, content generation**

I. Introduction

While the crossword generation problem has been studied for some time, it has recently received incredible attention due to the rise of new AI methods. Bonomo, Lauf and Yampolskiy [1] proposed using a genetic algorithm for crossword generation. Similarly, Bulitko and Botea [2] use an approach with the additional usage of neural networks for crossword quality estimation. Unfortunately, their crosswords were meant for a competition-level AI solver and because of that it is safe to assume they were unsolvable to humans. Rigutini, Diligenti and Gori [3] propose an NLP algorithm for clue generation.

Harris [4] proposes dividing these efforts into unconstrained and constrained crossword generators. The latter use a predetermined black field placement and crossword generators that do not need this assumption are unconstrained. Harris [4] observed that most crossword generators described in the literature are constrained. While his work is from 1990, this rings true to this day[1].

This paper employs a different approach. We introduce an inductive method of crossword generation based upon three operations: crossword concatenation, rotation and word removal. These methods are utilized in two AI algorithm designs based on simulated annealing and local search. Next, we present a summary of their performance and discuss our plans for this project. These are focused on adjusting the difficulty level of crossword solving and include an improved goal function, targeted crossword modification and a cuckoo search algorithm [5].

To create a common understanding of what we will consider crosswords, we invite the reader to look at the figure I. One can define the crosswords:

Definition 1: A crossword is a two dimensional matrix filled with empty (black) fields, clue fields and letter (white) fields. Every white field can be filled with one letter in such way that in every direction pointed by the clue field the white fields make up a word related to the clue via the dictionary. This word ends at the first black field encountered in a given direction.

II. Method

This section presents the crossword representation, the old goal function and AI algorithms used. We used Python 3 with multiprocessing support, allowing quicker crossword generation.

A. Crossword representation

The software stores crosswords in *dynamic slot tables* [4], allowing the system to leave specific fields empty. The letters are stored in a dictionary, with keys being their coordinates. Words are stored as "pointers" to their letters. The words are stored in two separate structures - one for horizontal words and one for vertical ones. Clues are stored similarly to the letters.

1) The coordinate system: The most exciting feature of this class is its possibility to perform three operations on crosswords – transposition, crossover and word removal. However, the crossover operation introduced the need to combine two crosswords while appropriately managing their coordinates. Because of this, the concept of relative coordinates is introduced.

The algorithm starts from the absolute coordinates. These are identical to the ones found in slot tables (every column and row is assigned a natural number and the coordinate is a pair $\langle column, row \rangle$). These can be easily transformed into

[1]Bonomo, Lauf and Yampolskiy [1] use a set of three predefined grids and while Bulitko and Botea [2] use random grids, the algorithm still only tries to satisfy the black fields provided by a random generator. The system developed by Rigutini [3] also seems to employ this design.

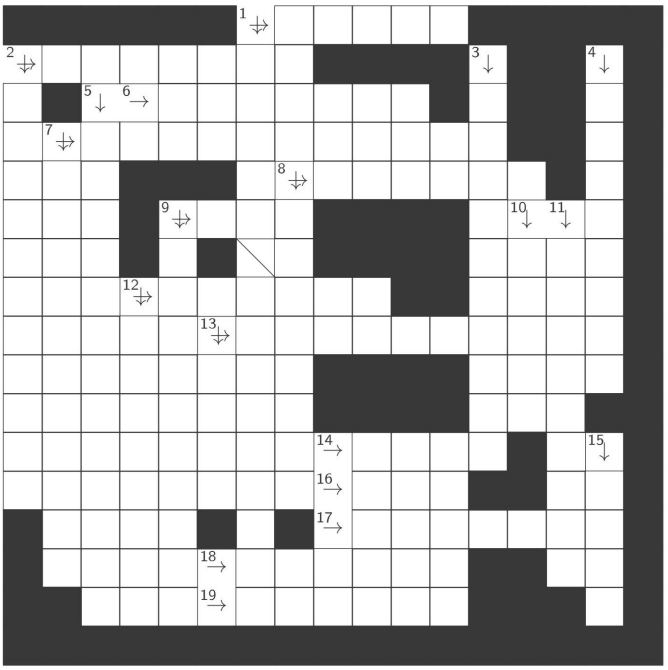

Fig. 1. An example of the crossword generated using this algorithm. The crosswords are compiled using LaTeX. Numbers direct the user to the provided clues. Arrows symbolize whether the word is horizontal or vertical. One can also see the crossed out field - this symbolises a whitespace.

a coordinate *relative* to another field (whose coordinates are $\langle x_0, y_0 \rangle$) using the formula:

$$relative(\langle x, y \rangle), \langle x_0, y_0 \rangle) = \langle x - x_0, y - y_0 \rangle$$

This operation can be reversed using the same function (min returns the lowest possible coordinate in this axis):

$$relative(\langle x - x_0, y - y_0 \rangle, \langle min(x), min(y) \rangle) = \langle x, y \rangle$$

The relative coordinates can be used to unify the coordinates of two crosswords before merging. When the software finds a possible intersection of two crosswords, it can relativize them in regard to the position of intersectable fields. From this point on, the two crosswords can be easily compared. The unification process is also presented on figure 2.

2) Crossover operation: When crossing over two crosswords, the system chooses a random pair of fields with the same letters and tries to create an intersection. This relies on an original letter collision detection algorithm. After the coordinate relativization stage, merged crosswords are scanned for fields with the exact coordinates. This way, every collision can be assessed by checking if a letter would be overwritten. If any collisions are found, the system looks for another possible intersection. The merging process is a set union operation. At the end the crosswords go back to the absolute representation.

3) Rotation operation: While the crossover operation is complex, the crossword rotation could not be simpler. It works

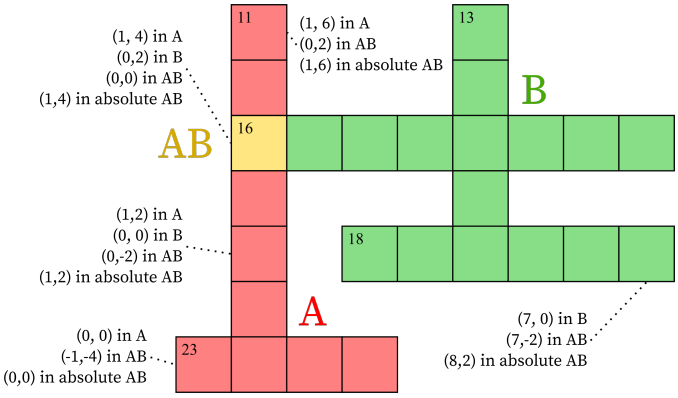

Fig. 2. An example of the coordinate unification process. The coordinates have been marked using parenthesis instead of angled brackets. Notice how the yellow field is the intersection, and its relative coordinates in AB are $(0, 0)$. This changes when the coordinates are turned back into their absolute counterpart.

just like matrix transposition. Here is an example to illustrate it:

$$\{\langle 1, 2 \rangle : a, \langle 2, 6 \rangle : b\}^T = \{\langle 2, 1 \rangle : a, \langle 6, 2 \rangle : b\}$$

The crossword has only two possible configurations preventing the creation of words that go from right to left or from the bottom to the top.

4) Word removal operation: The word removal operation is also quite simple. As the words stored in the crosswords are also pointers to the coordinates of their letters and the `Crossword` class also stores the coordinates of all word intersections in order to ease this process.

5) Crossword attributes: As the crosswords are simple data structures, we can define certain functions that will describe their traits. These functions will be used in the incoming sections:

- $intersections(a)$ returns the amount of intersections in a crossword. This is the size of aforementioned intersection set.
- $width(a)$ and $height(a)$ return the x-axis and y-axis values of the highest coordinate (in the absolute coordinates)
- $letters(a)$ returns the amount of white fields in a crossword. This is calculated using the combined sets of letters.

B. Crossword generation

How can one use this operation to generate puzzles? We can take a random word and transform it into a crossword by placing it in an empty crossword slot table. Then the same is done for another word, and they are combined using the proposed concatenation operation. This is repeated until some constraint is satisfied, and the crossword is ready.

Unfortunately, it will leave the user with a very sparse (and frankly, quite challenging) crossword. So, how to create *good* crosswords? This is the case of optimizing the goodness of a crossword. However, how does one define a good crossword?

1) Goal function and simple local search: We have proposed a temporary goal function for crosswords:

$$f(x) = intersections(x) * \frac{letters(x)}{width(x) * height(x)}$$

Initially, we only used the ratio of letters to crossword size (the second part of the equation). This, however, introduced huge crosswords with quite a small amount of words in them. We introduced the intersection amount to this function, as they give the user more hints regarding the words in the crosswords and promote more compact crosswords.

Using a simple *best improvement* and *first improvement* local search [6], one can now take multiple words and check which one fits best into the crossword. Ironically, this approach can produce undesirable results, as in the beginning, the puzzles should focus more on expanding and not on being dense. This produces small and sparse crosswords, as there is no room for additional words. We have found three approaches to solve this issue:

- Simulated annealing-inspired local search with algorithm seeding,
- Simulated annealing [7] with algorithm seeding,
- Cuckoo Search [5] (this one will be described in the discussion section).

2) Algorithm seeding: As the algorithm performed much better when it was tasked with optimizing an already existing crossword, we have decided to "seed" every crossword. This way, the starting puzzles are treated as the skeleton for the result. For now, the seeding algorithm combines multiple random words.

3) Simulated annealing-inspired local search: Since the *best improvement* local search will continually search for the highest goal function value, it can quickly get stuck in a local optimum. Seeding allows the production of puzzles with a high ratio of black fields to the filled-in ones, but it does not solve this problem.

This is quite similar to the problem solved by simulated annealing, and so we have modified the concept of slowly increasing the cautiousness of changes. The system starts at a high temperature, which allows it to conduct risky *first improvement* searches. With every turn, this value is lowered, and the system will have fewer chances to perform them, focusing on *best improvement* searches instead. These searches are performed on a specified number of random words not included in the puzzle.

This paints the image of an algorithm that starts by creating a randomized puzzle and then tries to fill it in being slightly more strict with every coming turn.

4) Simulated annealing: We have also introduced a more typical simulated annealing technique which uses local search and the word removal method. The algorithm creates a randomized puzzle and tries filling it in using the *best improvement* search while also having the possibility to remove a random word from the puzzle. This chance decreases with the temperature.

C. Web scrapping

To create a dictionary, the author has downloaded the words available in [8]. This was done using a script written in Python using the *BeautifulSoup 4* package [9] and the *lxml* parser. The data was quite low quality (repetitions, lack of the clues for certain words or their unhelpfulness), so it was curated manually. It is worth mentioning that the dictionary can be easily switched for another one and the software will still work. Therefore, one can also use it to generate crosswords in other languages. In some cases, the clues had multiple versions. In these situations all of them are printed. In the case of homographs, the software disallows them in one crossword.

III. Algorithm evaluation method

The first algorithm was prepared to be tested by setting the following hyperparameters:

- Starting temperature of 1 – this means that at the beginning, there will be a 100% chance of using the *first improvement* search.
- Temperature change speed of 0.015 (this means that every turn, 1.5% of the temperature dissipates).
- The amount of words in the seed is set to 12.
- The maximum size of a crossword is set to 33 by 25 (this perfectly fits an A4 page).
- The algorithm was set to stop if the filled-in ratio reached 80%.

The second algorithm used a much lower starting temperature of 0.2. Other parameters were left unchanged.

We generated 110 puzzles using both methods. The experiment was done using an Intel Core i9-10850 processor with the clock speed set to 4.8 GHz. All of the data was processed using the R language [10] and the *dplyr* [11] library used for easier data transformation.

IV. Results

The first algorithm's average puzzle generation time is 3.94 seconds (Standard Deviation (SD) = 1.71 s, Coefficient of Variation (CV) = 59%). This seems like a fair value, as even 8 seconds (the maximum generation time) should not be problematic for the end-user. This was also significantly lower (t=6.926, df=181, p<0.001) then the generation time of the second algorithm (Mean (M) = 5.07s, SD=2.78s, CV=54%).

The crosswords generated by the first algorithm had the mean letters to fields ratio ($\frac{letters(x)}{width(x)*height(x)}$) of 68.87% (SD=15%, CV=25%). The variation here is low, but the value could be improved. It is still better than what Bonomo, Lauf and Yampolskiy [1] achieved, as, in many of their grids, a great number of letters would be discarded. The first and second algorithm (M=0.66, SD=8%, CV=13%) didn't differ significantly when it comes to white field ratio (t = -1.6096, df = 165.52, p = 0.1094).

The mean ratio of intersections to letters ($\frac{intersections(x)}{letters(x))}$) in the first algorithm was 13.95% (SD=2.29%, CV=16%). It is hard to compare this to the work of other authors. We hypothesize this value should be improved, as it seems that the

word intersection can be used as additional clues. This idea is expanded on in the discussion. Unfortunately, the second algorithm isn't useful here, as its result here (M=12.07%, SD=1.58%, CV=13%) is significantly lower (t = -7.0472, df = 193.7, p<0.001).

Unfortunately, both generators created crosswords too complex for humans.

V. Discussion

The main limitation of this work is its current ineffectiveness at creating solvable puzzles and leaving local optima. We presume the reason for this is the vast randomness of our approach. Difficulty threshold of crosswords tends to be based on interaction between clues and structure of cells [12]. Due to randomness there is no possibility to form specific patterns and crosswords' themes. It seems improbable for the crosswords to be good when we need to rely on pseudorandom functions.

Besides minor improvements, like better algorithm seeding procedure and code optimization, there are three main improvements authors would like to focus on in the future. These changes aim to improve the solving experience for the users.

1) Goal function improvements: It is worth focusing on the frequency of a word in a language - the more often it is used, the easier it is to recall from memory, and the structure of the word - complexity refers to the number of letters building it. In order to keep the difficulty of the clues at the right level we plan to use the collocation extraction tool [13].

A vital element that constitutes a component of the difficulty of the word is its location in the crossword. We would like to expand on the idea of filled-in letters being clues and assess the possibility of guessing the words when certain letters are provided. This is related to the psycholinguistic model of lexical search, i.e. the Cohort model [14]. According to this model, when a person hears speech segments in real-time, each speech segment "activates" every word in the lexicon that begins with that segment. As more segments are added, more words are excluded until only one word remains that still matches the input. Similarly, here with each letter typed, the set of remaining possibilities narrows. This hypothesis could be tested empirically in a paradigm similar to the hangman game. The respondent would be presented with a word with letters in different positions and asked to guess the word as quickly as possible. Such an experiment would help determine which letters in which positions are most helpful in guessing passwords.

2) Targeted word elimination and addition: With the cohort-inspired crossword quality assessment comes the idea of assessing the quality of single words and crossword areas. This can be used in three ways:

- The software can eliminate words straying from the intended difficulty level.
- The system can search for the best and worst areas in the puzzles.
- The worst areas could be subjected to a targeted word search, which would find the best possible fits from the dictionary.

3) Cuckoo search: The procedure for finding the best areas in puzzles could be used to implement the Cuckoo search (as described in [5]). Similarly to how cuckoo birds lay eggs in the nests of other birds, the crossword would be able to place its best parts in other crosswords. While they would be able to "destroy the egg" using the word elimination, the fragment could also incorporate into the puzzle. Of course, at every turn, some crosswords would not survive.

VI. Conclusion

This work presented our current results in unconstrained crossword generation. After a brief overview of the field, we proposed a representation for crosswords in our software, three crossword operations and tested two crossword generation algorithms based around local search and simulated annealing. We discussed our plans on making our system less dependent on randomness using better goal function, word search and cuckoo search. This is a response to the main limitation of our work to this day.

References

[1] D. Bonomo, A. P. Lauf, and R. Yampolskiy, "A crossword puzzle generator using genetic algorithms with wisdom of artificial crowds," in *2015 Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games (CGAMES)*, 2015, pp. 44–49.

[2] V. Bulitko and A. Botea, "Evolving romanian crossword puzzles with deep learning and heuristic search," in *2021 IEEE Conference on Games (CoG)*. IEEE, 2021, pp. 1–5.

[3] L. Rigutini, M. Diligenti, M. Maggini, and M. Gori, "A fully automatic crossword generator," in *2008 Seventh International Conference on Machine Learning and Applications*. IEEE, 2008, pp. 362–367.

[4] G. Harris, "Generation of solution sets for unconstrained crossword puzzles," in *Proceedings of the 1990 Symposium on Applied Computing*. IEEE, 1990, pp. 214–219.

[5] M. Kochenderfer and T. Wheeler, *Algorithms for Optimization*. MIT Press, 2019. [Online]. Available: https://books.google.pl/books?id=uBSMDwAAQBAJ

[6] P. Hansen and N. Mladenović, "First vs. best improvement: An empirical study," *Discrete Applied Mathematics*, vol. 154, no. 5, pp. 802–817, 2006, iV ALIO/EURO Workshop on Applied Combinatorial Optimization. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X05003070

[7] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019. [Online]. Available: https://books.google.pl/books?id=HnetDwAAQBAJ

[8] polski-slownik.pl, "Wszystkie wyszukiwane hasła do krzyżówek," 2021. [Online]. Available: https://polski-slownik.pl/hasla-do-krzyzowek.php?pokaz=wszystkie

[9] L. Richardson, "Beautiful Soup documentation," *crummy.com*, 2020.

[10] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2021. [Online]. Available: https://www.R-project.org/

[11] H. Wickham, R. François, L. Henry, and K. Müller, *dplyr: A Grammar of Data Manipulation*, 2021, r package version 1.0.7. [Online]. Available: https://CRAN.R-project.org/package=dplyr

[12] J. K. McSweeney, "Analysis of crossword puzzle difficulty using a random graph process," in *The Mathematics of Various Entertaining Subjects*. Princeton University Press, 2015, pp. 105–126.

[13] P. Pęzik, *Narodowy Korpus Języka Polskiego*. Wydawnictwo Naukowe PWN SA, 2012, ch. Język mówiony w NKJP, pp. 25–49.

[14] W. D. Marslen-Wilson and A. Welsh, "Processing interactions and lexical access during word recognition in continuous speech," *Cognitive Psychology*, vol. 10, no. 1, pp. 29–63, 1978. [Online]. Available: https://www.sciencedirect.com/science/article/pii/001002857890018X