Core Expansion in Optimization Crosswords

Adi Botea[1], **Vadim Bulitko**[2]

[1]Eaton
[2]Department of Computing Science, University of Alberta
adibotea@eaton.com, bulitko@ualberta.ca

Abstract

In constraint optimization many problem instances remain challenging to current technology. We focus on the Romanian Crosswords Competition Problem. It is a challenging, NP-hard constraint optimization problem where state-of-the-art AI has been lagging significantly behind top human performance. We present an approach that first builds a core, a portion of the problem that will have a high contribution to the objective function. A core is grown into a seed, a partial solution with a subset of variables defined and instantiated. Seeds are further extended into full solutions. Our approach takes as input the size of a rectangular core to consider, and the locations of zero or more black cells inside the core. The results advance state-of-the-art substantially. We report a boost in the scores obtained, bringing our top solutions in the vicinity of top human entries.

1 Introduction

Many instances of constraint optimization problems remain challenging to current technology. In such problems, the task is to find a solution (e.g., assignment to variables that define a state) that satisfies a number of constraints specified in the definition of the problem, while optimizing an objective function (e.g., maximizing a score or minimizing a cost).

We focus on the Romanian Crosswords Competition Problem (ROCO for short), introduced in the search literature by Botea and Bulitko (2021). The application has a decades-long history of annual national-level competitions among humans. The problem is challenging to Artificial Intelligence (AI) which has been lagging significantly behind the performance of top human contestants, despite recent progress (Botea and Bulitko 2021; Bulitko and Botea 2021; Botea and Bulitko 2022).

In ROCO the input includes two lists of words (the thematic list and the regular list) and a 13×13 grid with white cells. The task is to fill the grid with criss-crossing words and no more than 26 black cells. Each thematic word gives a number of points equal to its length. The objective is to obtain as many points as possible.

We present an approach that first builds a *core*, a part of the problem that will have a strong contribution to the score (objective function) of the solution. A core is utilized to build a partial solution called a *seed*. A seed is further evolved into a *full solution*, in a way that aims at maximizing the overall score.

ROCO generalizes a well-known problem that we call the standard crosswords grid generation. The latter is a textbook example of a constraint satisfaction problem. The input is a list of words, and a grid with black cells and white cells. The task is to fill the white-cell area with criss-crossing words.

ROCO generalizes the standard crosswords grid generation in two ways and the resulting differences amplify the computational difficulty of ROCO in practice. Firstly, in ROCO, solutions are ranked by their score, whereas in the standard problem any correct solution will do. Often in practice finding a high-quality solution is much more challenging than finding any solution. Secondly, the task in ROCO includes finding a configuration of black cells (that will allow achieving a high score) whereas in standard grid generation the configuration of black cells is given as input. The absence of a pre-existing black cell configuration combinatorially increases the search space and good configurations (i.e., that allow achieving a high score) are difficult to find.

Our approach takes as input the size of a rectangular core and the locations of zero or more black cells inside the core. We report a substantial improvement of the scores achieved compared to previous work. Our system is capable of generating scores in the vicinity of human champion scores. We feel that the approach presented might be a breakthrough towards achieving the long-standing goal of outperforming human champions in this challenging domain.

2 Related Work

A simplified version of The Romanian Crosswords Competition problem was posed in the 2018 XCSP Competition (Lecoutre and Roussel 2019). Thirteen instances are featured in the competition, with square grids whose sizes range 3×3 to 15×15. Eight instances remained unsolved (Lecoutre and Roussel 2019). Audemard, Lecoutre, and Maamar (2020) use the same representation as a testbed for using segmented tables to encode constraints.

Botea and Bulitko (2021; 2022) focus on a subproblem inspired from the ROCO. Specifically, they assume that a good configuration of black cells, that would allow a high score, is given *a priori* as a part of the input. When a good configuration of black cells, such as a configuration extracted

from top human entries, is given as input, AI can successfully generate high-quality scores, such as reproducing top human scores (Botea and Bulitko 2021).

However, giving the black cell configuration as input is a massive help to an AI system. Automatically finding a good configuration of black cells has been challenging. Bulitko and Botea (2021) present a method to automatically search for black cell configurations. In their work scores obtained based on automatically generated black cell configurations have been significantly lower than top human scores. As mentioned in the introduction, our work in this paper brings scores to the vicinity of top human performance for the first time.

The literature has seen work on additional types crosswords problems, such as solving crosswords puzzles. A problem instance is defined by a grid with black cells but no letters, and a list of clues. The task is then to fill the grid with words that match the clues (Littman, Keim, and Shazeer 2002; Ernandes, Angelini, and Gori 2005; Ginsberg 2011; Chen et al. 2022). This is a significantly different problem compared to ROCO.

3 Background

We present how standard crosswords grid generation and its important generalizations can be mapped to constraint-satisfaction and constraint-optimization problems.

3.1 Constraint Satisfaction and Constraint Optimization

We consider constraint-optimization problems obtained from constraint satisfaction problems (CSPs) by adding a score function on partial or full solutions.

Definition 1. A *constraint-satisfaction problem (CSP)* is a tuple $\mathcal{P} = \langle X, D, C \rangle$, where:

- $X = \{v_1, \ldots, v_n\}$ is a collection of variables;
- $D = \{D_1, \ldots, D_n\}$ is a collection of finite domains with D_i corresponding to variable v_i;
- $C = \{C_1, \ldots, C_m\}$ is a collection of constraints. Each constraint C_j is a pair $\langle t_j, R_j \rangle$, where $t_j \subset X$ is a subset of p variables and R_j is a p-ary relation on the corresponding subset of domains.

We define D^* as an extension of D that contains a special symbol $\perp$ in each domain for variables with no actual value assigned yet. D^* can be used to represent partial assignments.

Definition 2. An *optimization CSP* is a tuple $\langle \mathcal{P}, f \rangle$, where $\mathcal{P}$ is a CSP and $f : D^* \to \mathbb{R}^+$ is a score function.

An assignment $s \in D^*$ is *consistent* if it violates no constraint. A consistent partial assignment is called a *partial solution*. A consistent full assignment is a *full solution* (or *solution* for short). A solution is optimal if no solution has a higher score.

3.2 Crosswords Grid Generation

Crosswords grid generation is a well-known, textbook example of a CSP. In a crosswords grid generation instance, the input is a list of words, and a grid with black cells and white cells. Define a word slot as a sequence of contiguous cells, either on a row or a column, bordered at each end of the sequence with either a black cell or the border of the grid. A popular way to model crosswords grid generation as a CSP is to define one CSP variable for each word slot. Its domain is the list of words of the corresponding length. The crosswords grid generation problem is NP-complete (Garey and Johnson 1979; Engel et al. 2012).

3.3 Optimization Crosswords

Optimization crosswords introduce a scoring function on top of the standard crosswords grid generation. Similarly to crosswords grid generation, the problem is NP-complete (Botea and Bulitko 2021). However, in practice, optimization crosswords instances often are much more challenging compared to standard grid generation. Intuitively, this is because in standard grid generation any correct solution will do, whereas in optimization crosswords we are interested in solutions with a high score. Observe that optimization crosswords are an example of an optimization CSP.

3.4 Crosswords with Dynamic Black Cells

In standard grid generation problems the placement of black cells is given as an input. In contrast, in crosswords with dynamic black cells the initial grid contains only white cells. The task is to fill the grid with a combination of black cells and crisscrossing words from the input list. Black cells added to the grid need to satisfy a number of constraints, depending on the exact problem formulation. A maximum number of black cells (as a percentage of the total number of cells) is a typical example of such a constraint. The next section shows in detail the constraints related to black cells that we consider in this work.

The need to add black cells as part of the solving process has two major implications. Firstly, the state space becomes much larger due to the combinatorics. For instance, on a 13×13 grid with 26 black cells, the total number of combinations is $\binom{169}{26}$ though not all of which are legal due to the constraints specified in the next section.

Secondly, adding black cells dynamically has the important consequence that word slots are introduced gradually. The number of word slots is initially zero and it grows as more and more black cells are placed on the grid. In other words, the CSP variables of the problem instance are introduced gradually, rather than being all defined from the beginning. Before all black cells are placed on the grid, the constraint problem modelling has only a *partial* problem definition, making the solving more challenging. For example, constraint propagation can be less effective at detecting infeasible values in the domains of state variables.

As mentioned in the introduction and also seen in the next section, the ROCO problem features both an optimization component and dynamic black cells, on top of the standard crosswords grid generation problem, inheriting a combination of the computational challenges outlined earlier.

As said earlier in this section, when black cells are already placed on the grid, the problem is an example of an

optimization CSP. The positions of the black cells define the word slots, which can be modelled as variables in the optimization CSP.

On the other hand, ROCO requires adding the black cells dynamically. When part of the black cells are missing from the grid, the definition of the word slots (i.e., the definition of the instance variables) is incomplete. Thus, in ROCO, we view the solving process as interleaving the *construction* of an instance of an optimization CSP (Definition 2) with *solving* the instance. Adding new black cells help define more and more variables (word slots) in the instance, until the definition is complete. Adding words instantiate the variables.

Seen from this perspective, we believe that our generic approach could be generalized to additional optimization CSP problems. In generic terms, our approach identifies a core that will have a strong contribution to the score. Then it builds variables, starting from the core, and instantiates the variables. The process continues until an instance that allow a high-score solution is fully built and solved.

4 Romanian Crosswords Competition

In this section we describe the application domain in detail, for a self-contained paper. The Competition is an annual contest with human participants, started in 1965. The task is to create a 13×13 grid filled with words and at most 26 black cells. The input includes two lists of words (dictionaries), called the *thematic list* and the *regular list*. The thematic list changes every year and is on the order of a few hundred words. The regular list contains approximately 135 thousand Romanian words. If the two lists overlap, the common words are considered to be thematic, and removed from the regular list. Black cells cannot have common edges but they are allowed to have common corners. White cells have to form a cardinally connected region. Black cells cannot create semi-closures — configurations of black cells such that adding one more black cell would partition the open/white area of the grid into two or more areas with the disconnected areas having more than one open cell each.

Recall that a word slot is a set of contiguous white cells in a row or a column where each end of the slot is adjacent to either the border of the grid or a black cell. Slots of length 1 can be filled with any letter. Slots of length 2 can be filled with any combination of two letters but no two-letter combination can be repeated in the grid. Slots of length 3 or higher can be filled with words from the two lists. For simplicity we say that singleton letters are words of length 1 and two-letter combinations are words of length 2. We treat them as regular words unless such a combination is in the thematic list.

Words of length 2 or higher cannot be repeated in a grid. So-called families of words are forbidden (e.g., WRITER and WRITING cannot both be placed on the same grid).[1] All cells in a (full) solution must contain a letter or a black cell.

The score of a grid is the sum of the lengths of all thematic words included in the solution. This is equivalent to saying that, in a solution each cell at the intersection of two thematic words contributes two points. A cell at the intersection of one thematic word and one regular word contributes one point. A cell at the intersection of two regular points contributes zero points, and so does every black cell.

[1] We do not observe the family-of-words constraint as no mapping of words into their families is available to us. We manually inspected our solutions reported in this paper, confirming that our solutions satisfy this constraint as well.

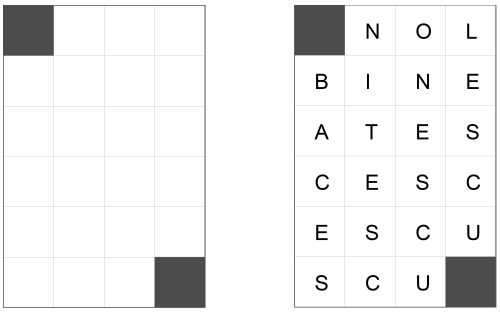

Figure 1: Left: A pattern defined as a 6×4 grid with two black cells, in the top-left corner and the bottom-right corner. Right: A core based on the pattern at hand.

Algorithm 1: GENERATING FULL SOLUTIONS.

Input: Pattern p, thematic dictionary and regular dictionary

```
C ← GenerateCores(p)
E ← ExpandCores(C)
S ← GenerateSeeds(E)
Rank S and put top k elements into a list S'
for s ∈ S' do
    Evolve(s)
```

5 Approach

Algorithm 1 summarizes the main steps of our approach. In this section we present these steps in detail and give an end-to-end example.

In high-level terms, we build an area with a high density of points, decide where to place it on the grid, and then build the rest of the solution around it. Human experts can sometimes take a strategy that in high-level terms fits this informal description.

As seen in the pseudocode, our approach takes as input a *pattern* p, besides the thematic dictionary and the regular dictionary. A pattern is a rectangular grid of a given size, with zero or more black cells and no letters.

Figure 1 (left) shows an example of a pattern. It is a 6×4 grid with two black cells, one in the top-left corner and one in the bottom-right corner. Observe that the pattern is significantly smaller than the 13×13 size of the full-size grid. The pattern will end up being integrated inside the full-size solutions generated with our approach as we will show in this section.

5.1 Generating Cores

Step 1 in Algorithm 1 generates a collection of so-called *cores* from the pattern p. A core is a pattern filled with let-

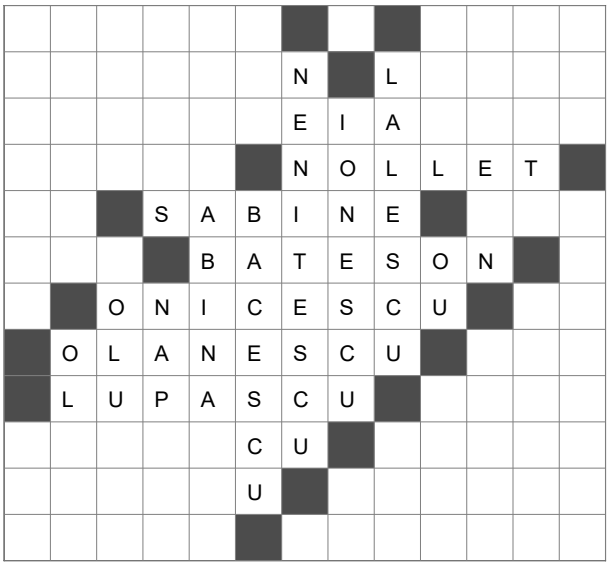

Figure 2: An expanded core based on the core shown in Figure 1 (right).

ters on the white cells. Figure 1 (right) shows an example of a core.

We require that in a full solution returned by Algorithm 1, the sub-area corresponding to the pattern will have a perfect score. That is each white cell in the pattern must contribute two points.

To achieve this we construct a small crosswords puzzle instance and solve it with WOMBAT, a freely available C++ solver for crosswords optimization. WOMBAT takes as input up to two dictionaries (one thematic and one regular), and a grid with the black cells in place, and zero or more letters. The program is very effective at computing high scores when such input is provided, including all black cells (Botea and Bulitko 2021, 2022).

In the small instance we build, the grid is the pattern p. There is only one dictionary, which contains substrings of the thematic words.[2] It is sufficient to consider only substrings of the lengths that correspond to the lengths of the word slots in the pattern.

In our running example, the lengths of the pattern word slots are 3, 4, 5 and 6. For example, when generating substrings of length 4, if IONESCU and POPESCU are thematic words, the substrings generated are IONE, ONES, NESC, ESCU, POPE, OPES, PESC and ESCU. Duplicates such as ESCU are kept only once.

We then run WOMBAT to enumerate all solutions to the small instance, which are precisely the cores computed at step 1 in Algorithm 1.

5.2 Expanding Cores

At step 2 each core is expanded by placing full thematic words on top of the substrings present in the core at hand.

[2]No regular dictionary is needed here, as we seek a perfect score for this instance.

For each thematic word added to an expanded core, we add two black cells at its corresponding endpoints. Figure 2 shows an example of an expanded core. The cells on the top row, bottom row, leftmost column and rightmost column form the *border* of the expanded core. The rest of the cells are the *inner part* of the expanded core. We define the *inner size* as $h \times w$, where h is the number of rows of the inner part and w is the number of columns. In Figure 2, the inner size is 10×11. Observe that the border can only contain black cells and empty cells. The inner part can contain letters, black cells and empty cells.

As shown later in this section, the inner part of an expanded core will be fit inside a full solution. All or part of the border may or may not be included in a full solution, as explained later on.

More than one expanded core can exist for a given core (e.g., a substring such as ESCU can be expanded into multiple thematic words such as IONESCU and POPESCU). We generate expanded cores with a depth-first search.

We say that an expanded core is a deadend (or, equivalently, it is dead) if no full valid solution can contain the expanded core at hand. We label an expanded core as dead, and filter it away, if any of the following conditions is met: it contains a word repetition; it contains a sequence of letters that cannot possibly be expanded into a full word; or it contains adjacent black cells within the inner part. Note that, at this stage, we do not check for adjacent black cells on the border. Such a test will be performed later, after it is decided which part of the border, if any, will be kept in a full-size solution. We detail this later in this section.

5.3 Generating Seeds

When the inner size of an expanded core e is 13×13, it fits inside a full 13×13 grid in a unique way. However, if the inner height and/or width are below 13, e can fit inside a full grid in multiple positions, obtained by shifting it inside the full grid vertically and/or horizontally.

Each fitting of an expanded core inside a full grid is called a *seed*. As such, a seed is a full-size 13×13 grid, with letters, black cells and empty cells. Figure 3 shows three seeds generated from an expanded core.

When the first column of the inner part of e coincides with the first column of the seed, the leftmost column of e's border is left out from the seed. Otherwise, it is kept within the seed. Other parts of the border (top row, bottom row, rightmost column) are left out or kept inside in a similar way. For example, in Figure 3 (left), the leftmost border column and the top border row are left out. The seed in Figure 3 (middle) leaves out the leftmost border column. The seed at the right of Figure 3 leaves out the leftmost border column and the bottom border row.

A seed is post processed with two objectives: to detect, if possible, dead seeds (e.g., seeds that cannot be possibly expanded into a valid full solution); and, for seeds that are still alive, to detect, if possible, forced black cells and forced letters that could be added to the seed. Given a position (cell) in a seed, we say that a black cell or a letter is forced onto a given position if every possible full solution based on that

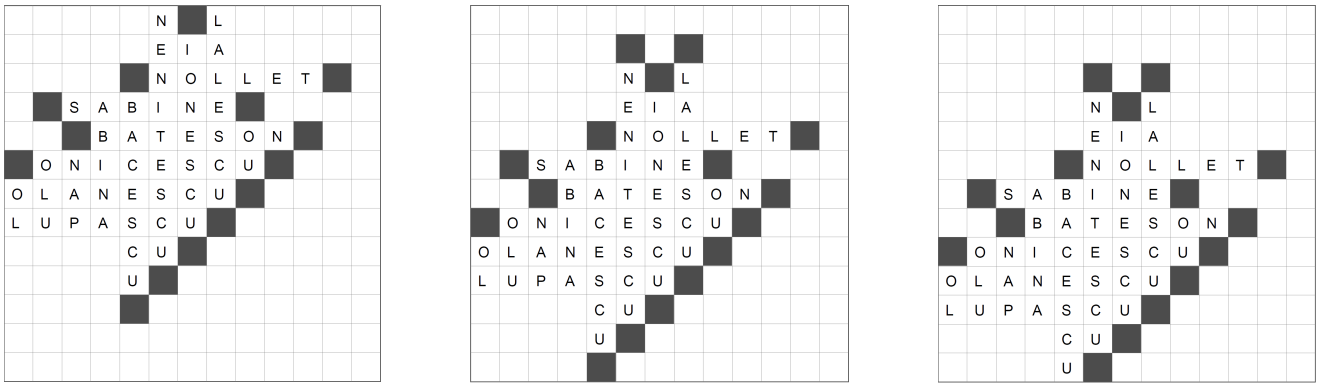

Figure 3: Seeds generated from the expanded core shown in Figure 2.

Algorithm 2: FIXPOINT COMPUTATION.

Input: Seed s labelled alive
repeat
  if *deadend detected* **then**
    Label s as dead
    Break
  Add forced moves (black cells and words) to s
until *reaching fixpoint*
return s and label (dead/alive)

seed must contain the black cell or letter at hand on the corresponding position.

Algorithm 2 shows seed post-processing in pseudocode. Deadend detection includes checking for closures and semi-closures; adjacent black cells;[3] word repetitions; and sequences of letters that do not match with any word from the dictionaries.

Forced black cells are detected when all dictionary words matching a given sequence of letters in the seed would start in the same cell, forcing the placement of a black cell on the cell before. Similarly, forced black cells can be detected when all matching words finish on the same cell, forcing a black cell on the cell after. Forced letters are detected when all matching words have the same letter on a given position, and it is not possible to insert a black cell between the sequence of letters and the cell where the forced letter is being added. This includes the case where exactly one word matches the sequence of letters at hand, forcing the addition of the entire word, and its corresponding black cells to border the word.[4]

5.4 Ranking Seeds

Given a seed (i.e., a 13×13 grid with some letters and n black cells) we need to complete it with an additional $26 - n$ black cells[5] so that WOMBAT can subsequently attack the instance and fill the remaining white area with words. The placement of the black cells should allow WOMBAT to obtain a high score on the resulting instance.

There are two principally different ways of placing the remaining $26 - n$ black cells: one-shot and incremental modifications. Indeed, one can place all $26 - n$ black cells in one shot and then submit the resulting grid to WOMBAT to fill in the words and compute the score. Alternatively, one can take an existing grid with 26 black cells and move its black cells around (protecting the letters and black cells from the original seed) in an attempt to improve the score. A grid modified (or mutated) in such a way is then submitted to WOMBAT again to compute its new score and the process repeats. One can view this process as an evolution of grids with WOMBAT being the (slow) fitness function.

The first way of completing a seed to a full grid is faster since WOMBAT is invoked only once. The second way is slower but can result in higher-score grids since the placement of the additional black cells is incremental and guided by multiple WOMBAT score computations.

After some experimentation we adopted the second approach, as detailed in Section 5.5. However, given the computational cost of evolving each seed into a full grid and the large number of seeds produced in the previous steps, we cannot afford to evolve every single seed. So instead we estimate the score potential of each seed with one-shot competitions and then evolve only the most promising seeds.

The ranking process worked as follows. We took N seeds from the previous step[6] and completed each seed multiple times to a full grid by placing $26 - n$ back cells randomly in the empty space of the seed. Here n is the number of black cells already present in the seed. We made sure each completion produced a legal grid which passes all constraints. Each such completion was scored with WOMBAT (with the hyperparameters found in Table 1). The process continued until a

[3] As this test is applied to the full grid, it covers the part of the expanded core border, if any, that was kept in the seed.

[4] No black cell needs to be added at a given end of the forced word if the word is adjacent to the border of the grid at that endpoint.

[5] While having 26 black cells is not a requirement for a legal crossword puzzle, having all allowed black cells (26 for a 13×13 grid) tends to maximize the score once the words are placed onto the grid.

[6] Each of them was checked to be a legal grid. Illegal seeds were discarded.

hyperparameter	ranking	evolving
WOMBAT target score	0	0
WOMBAT time limit	1 minute	3 sec to 3 min
WOMBAT weight	0.64	0.5 to 0.7
Time limit per seed	1 minute	4 hours
Number of maps m		3
Parent elites per map B		256

Table 1: Hyperparameters used for ranking/evolving seeds.

certain time limit. The maximum of the resulting WOMBAT scores was returned as the score estimate of the seed. All N seeds were processed in parallel on a 32-core CPU.

5.5 Evolving Seeds

Given the N seeds ranked as described in Section 5.4 we took K of them with the top score estimates. We then evolved each of the K seeds with a time limit listed in Table 1. The number K was chosen in a such a way that the total time of the ranking plus evolving would be around two days (Table 2). All K evolutions runs were in parallel on a 32-core CPU.

After experimenting with plain evolution (i.e., the simple genetic algorithms) we felt that the risk of population collapsing to a local maximum is too high. Once a population collapses to variants of a single individual (i.e., the local maximum) the evolution tends to stall.

Thus we adapted MAP-Elites algorithm (Mouret and Clune 2015) that explicitly maintains diversity of the population via the use of *maps* – tables whose rows and columns are labeled with feature values and whose cells can be empty or can contain a single individual (i.e., a crossword grid). An individual in a cell is called an *elite*. Grids compete only within a map cell to become an elite. Elites from different map cells co-exist without competition.

Each map cell is defined by a vector of feature values. We used the two features to define a map: the *maximum word slot length* (f_1) and the *number of walls* (f_2) where a wall is a group of (diagonally) adjacent black cells. The maximum word slot length (Bulitko and Botea 2021) is important because word slots which are too long have less chance of being filled with a thematic word and thus contribute fewer points to the overall score.

Resolution of each feature is an influential design choice. For instance, one can create a map cell for each possible value of each feature. This creates the least amount of competition since elites from different cells do not compete. On the other hand, by not distinguishing between several different values of each feature (i.e., value abstraction via aggregation) there will be fewer cells which increases the competition. We did not *a priori* know the optimal degree of feature value aggregation. Instead we ran MAP-Elites with m maps, each with different feature resolution (Algorithm 3).

We start by creating m maps in line 1. As we have two features each map is a two-dimensional table of cells. Each cell can contain a single elite: a grid together with its score.

Algorithm 3: MULTI-MAP MAP-ELITES.

Input: seed grid s
Output: champion grid c, its score $\text{score}(c)$

```
1  create maps M_1, ..., M_m at different feature resolutions
2  place seed s on each map M_i
3  score(c) = −∞
4  repeat
5      for i = 1, ..., m do
6          e^max ← argmax_{e∈M_i} score(e)
7          if score(e^max) > score(c) then
8              c ← e^max
9              score(c) ← score(e^max)
10     P ← ∅
11     for i = 1, ..., m do
12         P ← P ∪ select(M_i, B)
13     C ← ∅
14     for j = 1, ..., |P| do
15         C ← C ∪ mutate(P|_j)
16     for j = 1, ..., |C| do
17         compute score(C|_j) with WOMBAT
18     for i = 1, ..., m do
19         for j = 1, ..., |C| do
20             place C|_j on map M_i
21 until time limit is reached
```

Initially all cells are empty. The input seed grid s is then placed on each of the m maps in line 2. To place a grid on a map we first compute the grid's features f_1 and f_2. We then locate the map's cell with feature values closest to those of the grid. To illustrate, suppose there are three maps ($m = 3$) and their feature values are listed in Table 3. Then a grid with the maximum word slot length of eight and three walls ($f_1 = 8, f_2 = 3$) will place in the cell with coordinates $(2, 2)$ on map 2. Indeed the value of $f_1 = 8$ is closest to the second f_1 value for that map (i.e., 8.3333). Similarly, the value of $f_2 = 3$ is equally close to the second and third f_2 values for that map (i.e., 2 and 4). We break ties towards lower values.

Note that the input seed grid s has not been evaluated by WOMBAT at this point. So its $\text{score}(s)$ is computed merely from the letters already in the seed.

In lines 5 – 9 we update the global champion c: the grid with the highest $\text{score}(c)$ seen so far by comparing its score to that of the highest-score elite $e^{\max}$ from each map. Lines 10 – 12 pick B random elites from each map (with replacement) and add it to the set of all parents P. We then mutate each parent $P|_j$ by adding, removing or moving around its black cells in lines 13 – 15. We make sure that the initial seed's black cells and letters are unaffected and that the total number of black cells is 26. The resulting mutated grids, offspring, are put in the set C with duplicates removed.

Each offspring $C|_j$ gets scored by WOMBAT in lines 16 – 17. If the offspring $C|_j$ is not a legal grid (since a ran-

year	cores	expanded cores	seeds before postprocessing	seeds ranked (N)	ranking time	seeds evolved (K)	evolving time
2013	12560	2754	1259	932	30 minutes	352	1.9 days
2021	332062	1005114	122579	38035	21 hours	224	1.2 days
2023	123218	1042	1087	446	16 minutes	352	2.0 days

Table 2: Generating, ranking and evolving seeds.

map	f_1 values	f_2 values	map size
1	$\{6, 9.5, 13\}$	$\{0, 13, 26\}$	3×3
2	$\{6, 8.3333, 10.667, 13\}$	$\{0, 2, 4, \ldots, 26\}$	4×14
3	$\{6, 7, \ldots, 13\}$	$\{0, 1, \ldots, 26\}$	8×27

Table 3: Feature values in our multi-map MAP-Elites algorithm.

dom mutation can break legality) then its $\text{score}(C|_j) = -1$. Finally in lines 18 – 20 we update each map M_i with each offspring $C|_j$. When placing the offspring $C|_j$ on map M_i we first compute the corresponding cell of the map using the offspring's features and then compare $\text{score}(C|_j)$ to the score of the elite grid in that cell (if the cell is not empty). If the offspring has a higher score then it replaces the elite in that cell of the map and becomes the cell's new elite. An empty cell automatically receives the offspring.

A single iteration of the loop in lines 4 – 21 is called a generation. If it does not increase the champion $\text{score}(c)$ then we say that the evolution stalled on that generation. The longer the evolution stalls the higher the time limit and the weight are in WOMBAT in line 17. A generation on which a new highest champion score is obtained resets the time limit and the weight to their lowest settings (Table 1).

6 Empirical Evaluation

We applied the approach presented above to synthesize grids for three years of the competition: 2013, 2021 and 2023. Each year uses its own thematic dictionary. Note that competition results and human-designed grids for year 2023 are not yet announced. Our code and data are available at https://www.dropbox.com/s/d1g9l7vthpw35sh/code-data-socs23.tgz?dl=0.

For each of the three years we generated and expanded cores (lines 1 and 2 in Algorithm 1), and further generated seeds (line 3) that were post-processed and then ranked (line 4). Top-ranked seeds were then evolved (line 6). Summary statistics about such steps are listed in Table 2.

The ranking and evolution steps were repeated four times for the same seed set for each year. Each run used a different random-number generator initialization. As per Section 5.5, each ranking + evolution run took about two days on a 32-core CPU. We used four 32-core CPU nodes on a grid and thus were able to execute all four runs in parallel. Out of the four runs the highest score grids synthesized for each year are shown in Figure 4. On the left we show evolved seeds with the contents of the original seed shown in tinted background. Additional black cells added during the evolution are shown without tinting.

The human competition publishes the top 12 entries every year. Years 2013 and 2021 feature in their top-12 entries at least one entry that contains the 6×4 pattern considered in our experiments (Figure 1, left). In 2013 scores in the top 12 list ranged from 184 (first place) to 181 (second place) and so on down to 175 on the twelfth place. Our best score out of the four runs for that year is 182 with the mean and standard deviation of $4 \cdot 352$ scores of 159.6 ± 20.83. One human entry from 2013 features the 6×4 pattern mentioned. Our solution has the pattern in a different position, as well as part of the letters different inside the pattern. This further leads to significant differences between these two solutions. In 2021 human scores in the top-12 list range from 195 down to 187 whereas our top score out of the four runs is 190 (162.6 ± 25.02). Two top-12 entries in 2021 feature the perfect-score pattern considered in our work. One has 194 points and one has 189 points. Our best entry from the four runs has 190 points and turns out to be similar to the 194-point entry. In particular, the core has the same letters in both cases and the same position in the grid. In contrast, the 189-point human score differs significantly from our solution. In particular, the 6×4 pattern is in a different position and the letters inside the pattern differ in part. Year 2023 human results are not announced yet. Our top score of the four runs is 186 with the mean and standard deviation being 147.9 ± 36.11.[7]

Thus with additional input as simple as a pattern to use which in our experiments is a 6×4 rectangle with two black cells in its corners, our approach can reach scores that are competitive with those from top human performers. Part of the previous work utilized much more detailed additional input such as the entire configuration of black cells (Botea and Bulitko 2021). When no such input is used, the best results reported were significantly behind top-12 results in that

[7]While it was previously known that starting from an empty grid tends to result in scores of evolved grids below that of champion-level human-created grids, we did run some experiments with MAP-Elites. The empirical setup was somewhat different from the that used for the main experiments yet the scores of grids MAP-Elites synthesized from an empty grid were indeed lower than those when starting with seeds: 154 versus 190 for year 2021 and 141 versus 186 for year 2023.

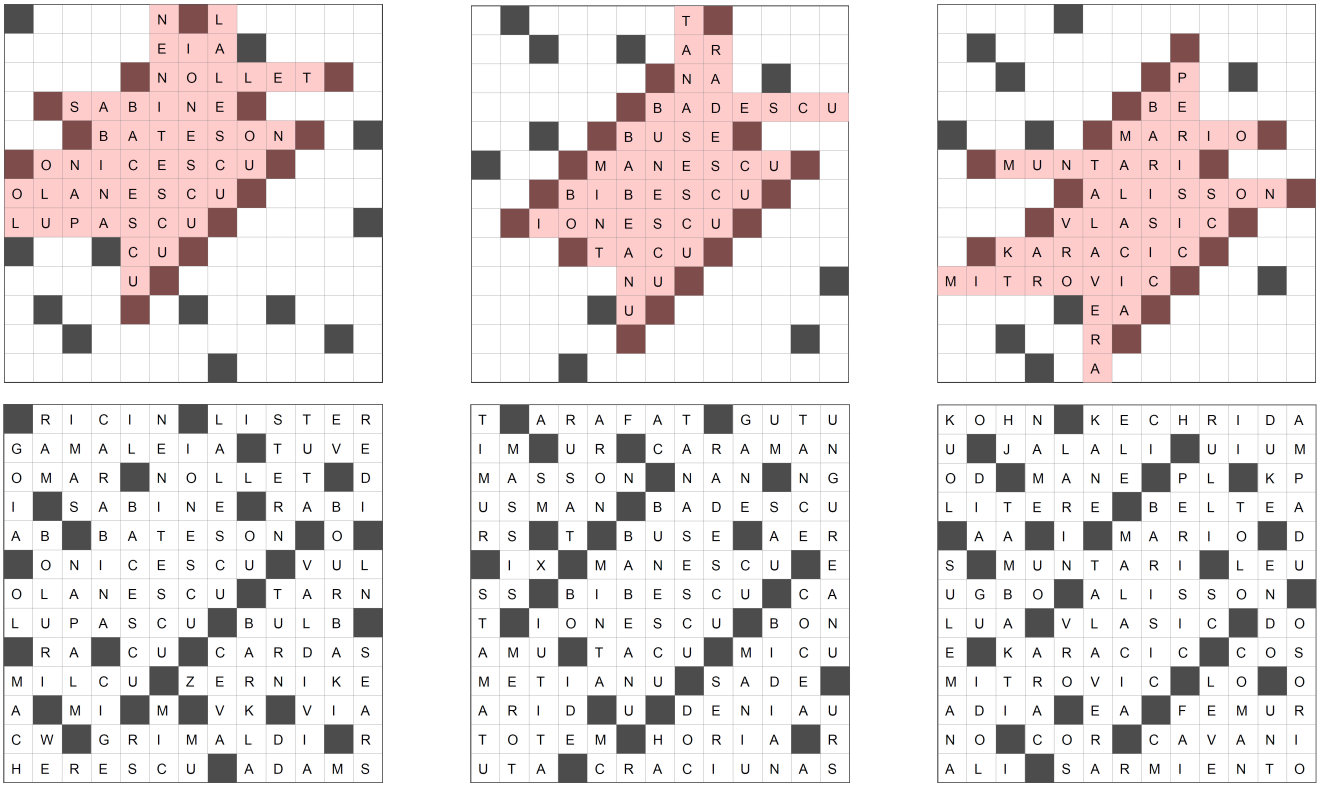

Figure 4: Our top results for years 2013 (left column), 2021 (middle column) and 2023 (right column). Top: seeds after placing additional black cells. Bottom: WOMBAT solutions.

year (Bulitko and Botea 2021; Botea and Bulitko 2022).

7 Future Work

In the future we plan to utilize multiple patterns rather than just one. We further plan to synthesize grids with higher scores via increasing the amount of computation as well as handling illegal evolved grids in a more sophisticated fashion. For the former, with more computational resources we can increase the number of seeds ranked and evolved, increase the time for estimating each seed's score during ranking, increase the time limit in WOMBAT to get higher scores for a candidate grid, maintain more maps inside MAP-Elites, increase the evolution time limit and simply run more independent ranking + evolution runs on the same seeds.

For the latter we currently allow illegal grids to be placed on a map. This may help the evolution by storing intermediate steps or missing links towards a legal high-score grid. However, because each such illegal grid automatically gets a score of -1 there is no meaningful measure of quality for such infeasible grids (Kimbrough et al. 2008). Thus one illegal grid can never replace another one on a map and thus no latent mutations can be accumulated over time. Future work will remedy this by introducing a meaningful measure of quality for illegal grids so that the evolution can improve them over time. One possibility is to consider legal offspring score potential (Gallotta, Arulkumaran, and Soros 2022).

8 Conclusion

Many types of constraint optimization problems remain challenging to current technology. In this work we focused on the Romanian Crosswords Competition Problem, a challenging domain where AI performance has been lagging significantly behind top human performance. We presented an approach that first builds a core with a high contribution to the score function, and then builds the rest of the solution around it. Using as input the size of a rectangular core and the placement of two black cells inside the core, we build full solutions that are competitive to top human performance, boosting previous AI results. We believe that our results might give a basis to exceeding top human performance with fully automated solutions in future work.

Acknowledgments

We appreciate support from Compute Canada, funding from NSERC and consultation from Jonathan Schaeffer who independently proposed the idea of starting from a high score density cluster of words and building walls around them.

References

Audemard, G.; Lecoutre, C.; and Maamar, M. 2020. Segmented Tables: An Efficient Modeling Tool for Constraint Reasoning. In *Proceedings of the European Conference on Artificial Intelligence, ECAI-20*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, 315–322.

Botea, A.; and Bulitko, V. 2021. Scaling Up Search with Partial Initial States in Optimization Crosswords. In Ma, H.; and Serina, I., eds., *Proceedings of the Fourteenth International Symposium on Combinatorial Search, SOCS 2021, Virtual Conference [Jinan, China], July 26-30, 2021*, 20–27. AAAI Press.

Botea, A.; and Bulitko, V. 2022. Tiered State Expansion in Optimization Crosswords. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 18(1): 79–86.

Bulitko, V.; and Botea, A. 2021. Evolving Romanian Crossword Puzzles with Deep Learning and Heuristic Search. In *2021 IEEE Conference on Games (CoG), Copenhagen, Denmark, August 17-20, 2021*, 1–5. IEEE.

Chen, L.; Liu, J.; Jiang, S.; Wang, C.; Liang, J.; Xiao, Y.; Zhang, S.; and Song, R. 2022. Crossword Puzzle Resolution via Monte Carlo Tree Search. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Engel, J.; Holzer, M.; Ruepp, O.; and Sehnke, F. 2012. On Computer Integrated Rationalized Crossword Puzzle Manufacturing. In *Proceedings of the International Conference on Fun with Algorithms, FUN-12*, 131–141.

Ernandes, M.; Angelini, G.; and Gori, M. 2005. WebCrow: A Web-Based System for Crossword Solving. In Veloso, M. M.; and Kambhampati, S., eds., *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, 1412–1417. AAAI Press / The MIT Press.

Gallotta, R.; Arulkumaran, K.; and Soros, L. B. 2022. Surrogate Infeasible Fitness Acquirement FI-2Pop for Procedural Content Generation. In *2022 IEEE Conference on Games (CoG)*. IEEE.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman.

Ginsberg, M. L. 2011. Dr.Fill: Crosswords and an Implemented Solver for Singly Weighted CSPs. *Journal of Artificial Intelligence Research*, 42: 851–886.

Kimbrough, S. O.; Koehler, G. J.; Lu, M.; and Wood, D. H. 2008. On a Feasible–Infeasible Two-Population (FI-2Pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research*, 190(2): 310–327.

Lecoutre, C.; and Roussel, O. 2019. Proceedings of the 2018 XCSP3 Competition. *CoRR*, abs/1901.01830.

Littman, M. L.; Keim, G. A.; and Shazeer, N. M. 2002. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134(1-2): 23–55.

Mouret, J.; and Clune, J. 2015. Illuminating search spaces by mapping elites. *CoRR*, abs/1504.04909.