Scaling Up Search with Partial Initial States in Optimization Crosswords

Adi Botea,[1] **Vadim Bulitko**[2]
[1]Eaton
[2]Department of Computing Science, University of Alberta, Edmonton, AB, Canada
adibotea@eaton.com, bulitko@ualberta.ca

Abstract

Heuristic search remains a leading approach to difficult combinatorial optimization problems. Search algorithms can utilize pruning based on comparing a target score with an admissible (optimistic) estimate of the best score that can be achieved from a given state. If the former is larger they prune the state. However, when the target score is too high the search can fail by exhausting the space without finding a solution. In this paper we show that such failed searches can still be valuable. Specifically, best partial solutions encountered in such failed searches can often bear a high similarity to the corresponding part of a full high-quality or even optimal solution. Thus, a new search for a full solution, with a lower target score, can start with a best known partial solution, rather than starting from scratch. We demonstrate our ideas in a constraint optimization problem modelled on the Romanian Crosswords Competition, a challenging problem where humans perform much better than computers. Utilizing partial solutions produced by a failed search cuts down the running time of an existing state-of-the-art system by orders of magnitude on competition-level crosswords puzzle instances and allows to solve more instances.

Introduction

Search remains a powerful approach to difficult combinatorial tasks including NP-hard problems. Many search problems also allow for solutions of varying quality thereby requiring a solver not only to find a solution but a solution of a higher quality. Such search problems can be framed as optimization problems with solution quality (also called a *score*) being the objective function.

In this paper we present a two-stage approach to improve search for a high-score solution. In the first stage we use heuristic search to generate a *partial* solution which ideally bears a high similarity to a yet undiscovered high-score full solution. In the second stage we use this partial solution as an initial search state to search for a *full* solution.

Our two-stage approach can be used on top of any search algorithm that takes a target score as a parameter and uses it during the search to prune search states whose heuristics values are below the target score. Thus a higher target score induces more pruning.

Such a baseline search algorithm is used in both stages of our approach as follows. In the first stage, the target score is set to a value significantly higher than an actually achievable score. The resulting search is thus called an *overestimation search*. It exhausts the search space quickly due to the aggressive pruning but does not find a solution. However, the best partial solution encountered in an overestimation search is frequently similar to the corresponding part of a high-quality or even optimal full solution. Several overestimation searches can be performed, each with a different target score, until an acceptable partial solution is found.

In the second stage we set the target score to an achievable value and start the search with the partial solution found in the first stage. Then the second-stage search effectively completes the partial solution to a high-score full solution. If an achievable score value is not known *a priori* then we can run the second-stage search multiple times, starting with an upper bound on the optimal score and progressively reducing it until a solution is found.

Despite having two stages of heuristic search, their combined effort can be orders of magnitude smaller than the effort to search for a full solution starting with an empty solution. Intuitively, the first-stage search is fast due to aggressive pruning while the second-stage search is fast due to seeding it with a relevant partial solution.

To demonstrate the viability of the two-stage approach we apply it to the Romanian Crosswords Competition, a problem that has spurred over five decades of annual national-level competitions. This is a challenging problem where human creativity allows them to defeat the current state of the art in Artificial Intelligence.

The task is to construct a 13×13 grid adding both black cells and letters. Every cell needs to be filled with either a letter or a black cell. Word slots of length 3 or higher need to be filled with a valid word from one or several dictionaries (list of words) given as input. Slots of length one can be filled with any letter, and slots of length two can be filled with any two-letter combination. A small fraction of the valid words, called *thematic words*, have a score corresponding to their number of letters. The score of a full grid is the sum of the lengths of all thematic words.

In this paper we assume that the black cells are given as an input and focus on filling up the grid with words with the objective of maximizing the puzzle score. We call crosswords grid generation the task of finding a valid way to fill a grid with words from a list, given a configuration of black cells. As such, we address an optimization variant of crosswords grid generation, in line with the national annual competition. This is in contrast to the typical textbook framing of crosswords grid generation as a constraint-satisfaction problem, without attempting to maximize a score.

To apply our approach to this optimization problem we take an existing competitive puzzle generator based on heuristic search. As it prunes search states using a target score it is amenable to the two-stage approach outlined above. We empirically evaluate the approach on instances from the human competition, using black cell configurations from top-level human-designed crosswords puzzles. The two-stage approach improves the performance of the baseline system by orders of magnitude which, in turn, results in significantly more instances being solved.

We make the following contributions. First we introduce a simple but promising approach to significantly improving the performance of a baseline heuristic search based system for optimization problems. Second, we empirically demonstrate the potential of this approach by reporting an orders-of-magnitude improvement over a state-of-the-art heuristic search system. Finally, we introduce the long-running Romanian Crosswords Competition to the search community, calling for its use as a challenging problem for heuristic search research. Achieving and exceeding human performance in games have been major drivers in Artificial Intelligence research for decades (Schaeffer et al. 1996; Schaeffer 2008; Campbell, Hoane, and Hsu 2002; Buro 1997, 1999; Silver et al. 2017). We hope that using this problem will lead to interesting advances in heuristic search eventually netting another victory for Artificial Intelligence over humans. To get started we can make our code and data available.

Related Work

Existing work in crosswords grid generation focuses more on the constraint-satisfaction variant ignoring the score-optimization aspect. Early work is due to Mazlack (1976), with a letter-by-letter filling approach. Ginsberg et al. (1990) adopt a word-by-word filling strategy. Meehan and Gray (1997) perform a comparison of these two strategies and conclude that a word-by-word approach is more effective. COMBUS is a more modern solver (Botea 2007; Anbulagan and Botea 2008). Part of the functionality offered in COMBUS is reused in the baseline search algorithm used in our work.

Botea and Anbulagan (2009) analyse the crosswords grid generation problem in terms of phase transition and easy-hard-easy behaviors. Such phenomena are observed when varying parameters such as the size of a dictionary and the percentage of black cells. Their work is limited to the standard problem, with no optimization component.

A simplified version of The Romanian Crosswords Competition problem has been posed in the 2018 XCSP Competition (Lecoutre and Roussel 2019). 13 instances are featured in the competition, with grid sizes ranging within $3 \times 3, 4 \times 4, \ldots, 15 \times 15$. 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. The relaxation used in such work allows word repetition and isolating parts of the grid from each other with black cells. While we avoid such relaxations, we use a different simplification, requiring a configuration of black cells as input.

Search algorithms often perform pruning based on upper bounds on the solution cost. Branch and bound (Land and Doig 1960) is a prominent example where lower and upper bounds are explicitly used for pruning, as part of a search for a full solution. We do not limit our approach to pruning based on upper bounds. We additionally run dedicated searches with target score upper bounds to obtain a partial initial state for a subsequent search for a full solution.

Problem Definition

Our ideas apply to combinatorial search problems where states are characterized by a score function, and the task is to compute a solution with a high score. However, for clarity, we apply our ideas to a specific such problem, namely an optimization variant of a constraint satisfaction problem.

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

- $X = \{X_1, \ldots, X_n\}$ is a set of variables;
- $D = \{D_1, \ldots, D_n\}$ is a set of finite domains, with D_i corresponding to variable X_i;
- $C = \{C_1, \ldots, C_m\}$ is a set 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 $\overline{D}$ as an extension of D that also contains a special symbol $\perp$ in each domain, to represent variables with no actual value assigned yet. In other words, $\overline{D}$ allows representing partial assignments.

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

We say that a partial assignment $s \in \overline{D}$ is consistent if it violates no constraint. A full assignment that is consistent is called a solution. A solution is optimal if no other solution has a larger score.

Romanian Crosswords Competition

The Romanian Crosswords Competition is an annual contest, started in 1965. Participants submit crosswords grids with a score defined as the sum of the lengths of all words in the grid coming from a special list of words called the *thematic* list (dictionary). The competition is organized by Rebus, a crosswords-dedicated Romanian publication. This includes publishing the thematic list and the rules at the beginning, and the results at the end, including the top 12 grids in that year.

More specifically, the task is to construct a 13×13 grid filled with valid words and no more than 26 black cells. In a solution, every cell needs to be filled with either a letter or a black cell. Words from two lists (dictionaries) can be used to fill the grid. One is a generic Romanian dictionary,

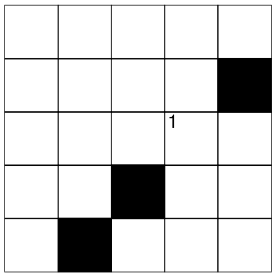

Figure 1: Semiclosure example on a toy grid. Adding one more black cell, at position 1, would partition the grid into two disjoint areas, each having more than one white cell.

with about 135 thousand words, which we call the *regular* dictionary. Using words from the regular dictionary gives no points. The second list, namely the thematic dictionary, is much smaller. It changes every year, and the size is a few hundred words. For example, in the nine-year dataset that we use in our study in this work, the thematic dictionary varies from 278 to 481 words, with the median size being 387. Using a word from the thematic dictionary gives a point for every letter. Thus, every cell in the grid can give at most two points, when it lies at the intersection of two thematic words.

Given a configuration of black cells, we call a word slot a set of contiguous white cells, on the same row or column, where each end of the slot is adjacent to either the border of the grid or a black cell.

Slots need to be filled with words from the two dictionaries, as said earlier. Furthermore, length-two slots can be filled with any combination of two letters. Length-one slots can be filled with any letter. Words cannot be repeated in the grid, and neither can two-letter combinations. No solution can contain two or more words from the same so-called family of words (e.g., WORK and WORKER).[1] In a full solution, every cell needs to contain a letter or a black cell.

Two black cells cannot share a common edge. Black cells cannot partition the grid into disjoint areas. Furthermore, *semiclosures* are forbidden. A semiclosure is a pattern of black cells with the property that adding one more cell would partition the grid into disjoint parts, and each disjoint part would have two or more white cells. Figure 1 shows an example.

The thematic dictionary that applies to a given year is released when the competition starts. Participants have around three months to submit their entries.

Constructing a competition grid requires both adding black cells and words. In this paper we assume that the configuration of black cells is given as input. We focus on adding the words, aiming at maximizing the score.

Standard crosswords grid generation, with no optimization component, is a textbook example of a CSP. Thus, not

[1]In this work, we do not impose the family-of-words constraint, as no mapping of words into their families was available. However, a post-mortem analysis confirms that solutions obtained in the empirical evaluation actually do respect this constraint.

Algorithm 1: OVEREST

```
Input: Instance: optimization CSP
Overestimation max target threshold $\Theta$
Overestimation min threshold $\theta$
Decrement step $\ell$
Max target for full-solution searches $T$
Min number of instantiated variables $k$
Output: Best full solution found if any
1  $s_0 \leftarrow \varepsilon$
   /* Overestimation searches */
2  $t \leftarrow \Theta$
3  while $t \geq \theta$ do
4      bps $\leftarrow$ SEARCH($\varepsilon, t$)
5      if bps has at least $k$ instantiated variables then
6          $s_0 \leftarrow$ TRIM(bps)
7          break
8      $t \leftarrow t - \ell$
   /* Search for a full solution */
9  for $t \leftarrow T$ downto 1 do
10     $s \leftarrow$ SEARCH($s_0, t$)
11     if $s$ is full solution then
12         return $s$
13 return failure
```

surprisingly, with the black points predefined, the Romanian Crosswords Competition problem can be formalized as an Optimization CSP as follows. Each slot defines a variable whose initial domain has all words of the corresponding length. Constraints impose that two intersecting slots agree on the letter at their intersecting cell, and that no word or two-letter group repeats on the grid. The score function for a partially instantiated state is the sum of the lengths of all words from the thematic list placed on the grid.

Combining Overestimation Search with a Search for a Full Solution

In this section we present our main algorithmic contribution, a two-stage approach to speed up the search for a solution. The idea is simple to understand and easy to implement. In the first stage, we run overestimation searches. The objective is to obtain a partial state that could be used as an initial state in the searches in the second stage. Searches in the second stage aim at finding a full solution. Algorithm 1 shows our two-stage approach in pseudocode.

As we will show in the empirical evaluation, often, the combined search effort is much smaller than running standard searches, for the following two reasons. First off, overestimation searches benefit from an aggressive pruning. Secondly, second-stage searches benefit from starting from a partially instantiated initial state, rather than starting from scratch. See more details later in this section.

A key factor that enables the effectiveness of the approach is the observation that, often, the best partial solution found in an overestimation search can be the basis for a high-quality full solution. We illustrate this in Figure 2, and we discuss in more detail in the section focused on trimming a partial solution. The empirical evaluation presented in this paper further shows that best partial solutions greatly overlap (on the subset of instantiated variables) with the best known solutions.

OVEREST invokes a search algorithm at lines 4 and 10. We require that such an algorithm implements pruning based on a given target score. For example, take the sum of the score of a given partial state, and an admissible (optimistic) estimation of the best score that could be achieved in the rest of the state. If this sum is below the current target, prune the state at hand.

The parameters to method SEARCH shown in the pseudocode are the initial state of a search and the target score. We denote an empty initial state (i.e., a state with no variables instantiated) by ε. We assume that the range of scores is discretized. For simplicity, in this paper we assume that the score takes positive integer values.

Overestimation Search

As seen in Algorithm 1, overestimation searches are just standard searches. However, we set the target score to a larger value than an achievable score. An overestimation search can exhaust the search space relatively fast, as a large target score can result in aggressive pruning. We start from a target score t set to a maximum threshold Θ, and each new iteration (i.e., new overestimation search) uses a t value smaller by ℓ than the previous. The iterations at lines 3–8 in Algorithm 1 stop when one of the following two conditions holds: either i) an overestimation search has found a best partial solution with at least k instantiated variables (line 5); or t reaches a minimum threshold θ (line 3). As explained later in this section and in the experimental setup section, imposing a threshold k for the number of instantiated variables will ensure in turn that the partial initial state used when searching for a full solution satisfies some minimum size condition. Otherwise, a very small partial initial state would have a much more reduced impact on scalability. See the experimental setup section for a discussion on how parameters mentioned in this section are set in the experiments.

Partial Solution Trimming

After validating (accepting) a best partial solution (line 5 in Algorithm 1), the best partial solution is trimmed (line 6), to obtain the state s_0 that will be used as an initial state in the second-stage searches.

The use of trimming is illustrated in Figure 2. The best partial solution discovered in the overestimation search (left) has a good but not perfect overlap with a full high-quality solution. Trimming is an attempt to improve the overlap. In the example shown, trimming achieves a perfect overlap (the middle grid in the figure).

Intuitively, we want to trim away words that are less coupled (interleaved) with the rest of the words in the partial grid. The less coupled with other words the more likely that a given word could be irrelevant. Take, for instance, word ALOR in the grid in Figure 2 on the left. Ignoring, for simplicity, any constraints on this word slot due to constraint propagation, it is coupled with the rest of the grid through only one letter. Several other 4-letter words with L on their second position could potentially match in there.

Our trimming strategy is simple: ignore the last z variable instantiations in the order they were performed, as the most recently added words might be less coupled with the rest of the grid. Selecting an actual z value is discussed in the experimental setup section. As seen in the empirical evaluation this simple trimming works well, leading to a high-quality solution (e.g., with a best known score) in a majority of cases.

There is no guarantee that a partial best solution selected in the first stage is a part of an optimal full solution. In other words, our combined approach offers no guarantee that it will return an optimal solution.

Searching for a Full Solution

In the second stage, the target score t starts from an upper bound on the optimal score. The target score t is decreased by 1 at each iteration, until a full solution is found. Each search in the second stage is initialized to s_0, rather than starting from scratch (i.e., from an initial state with no variable initialized). This can be a substantial benefit since a partial state with m instantiated variables is equivalent to getting the first m moves in the search for free. The search explores only the corresponding subtree at depth m with a size exponentially smaller than the original search tree. In a sense, starting from a partial state is similar to using opening books in two-player games such as Chess and Checkers which provide good moves in the early stages of a game with no search effort.

Baseline Search Algorithm

As shown in the previous section, our approach invokes a baseline search algorithm at each of its two stages (lines 4 and 10 in Algorithm 1). We use a search algorithm available in Botea's WOMBAT system. WOMBAT is a system for optimization crosswords grid generation that requires a configuration of black points as input. This section gives an overview of the baseline search algorithm used in our experiments. This is needed because no report on WOMBAT is available in the literature.

In this paper we call the baseline algorithm WOMBAT-DFS. It is based on depth-first search (DFS) and implements several enhancements as follows.

WOMBATDFS treats each word slot of length greater than 2 as a variable. Instantiating a new slot (variable) involves selecting the slot to fill and generating words that fit into the slot, given the constraints in the state at hand. In the initial state WOMBATDFS selects the first slot to instantiate with a heuristic that quantifies the influence of that slot across the entire state. For a given slot, its influence is computed as the sum of the lengths of all intersecting slots of length greater than 2. The larger the value, the larger the influence. A highest-influence slot is selected, breaking ties at random.

In non-initial states the slot to instantiate is selected with a popular heuristic in constraint satisfaction: prefer variables (slots) with a smaller branching factor. In particular,

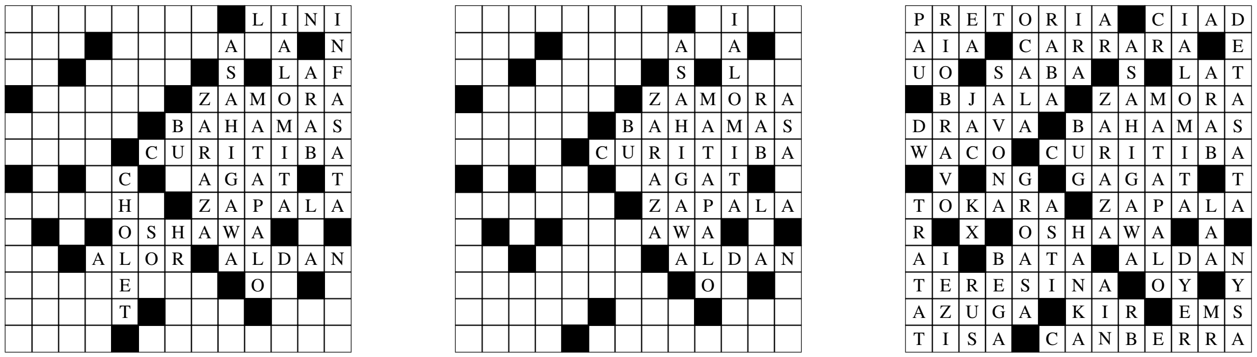

Figure 2: An illustration of similarity scores. On the right is a human-designed full puzzle. The partially filled grid on the left has 63 letters, 50 of which match the human-designed puzzle. Its similarity score is thus $50/63 \approx 0.79$. The grid in the middle loses 18 letters including all 13 mismatching letters. Its remaining 45 letters match the human-designed puzzle, yielding the perfect similarity score of $45/45 = 1$.

an empty variable domain shows a dead end, allowing to prune an exploration path as soon as a dead end is detected.

However, unlike typical constraint satisfaction problems, where the branching factor is the size of the domain of the variable selected for instantiation, WOMBATDFS reduces the branching factor as follows. It leverages the fact that the domain of a given variable (slot) has two types of values, namely thematic words and regular words. Thematic words are relatively few in comparison to regular ones (e.g., in the competition the regular dictionary is three orders of magnitude larger than the thematic dictionary). Instantiating a slot considers either thematic words only or regular words only. Specifically, every slot has a boolean flag initialized to true. Assume that in a given state s the slot v to instantiate has the flag set to true (never before instantiated), and it has m thematic words and n regular words in the current domain. WOMBATDFS generates $m+1$ successors, with a significant reduction in the branching factor relative to the full branching factor of $m+n$. The first m successors correspond to the cases when the slot is instantiated with a thematic word. The last successor does not instantiate the slot, but sets the flag to false to indicate that thematic words have already been considered in v. Let s' be this last successor of state s. When s' is instantiated, a slot with a smallest branching factor will be selected, as for any other state. Given that v's branching factor in this new state will be n, which can be large, it is likely that another slot, with a much smaller branching factor will be selected for instantiation in s'. Subsequently, in the subtree rooted at s', v may eventually need to be instantiated when its branching factor is the smallest. However, by then, constraint propagation – discussed later in this section – could reduce v's branching factor to values much smaller than n. In effect, using thematic words at first and regular words later effectively reduces the branching factor.

WOMBATDFS performs constraint propagation in the fashion implemented in the COMBUS system (Botea 2007). Every time a slot (variable) is instantiated with a word, constraint propagation shrinks the domains of other variables in the problem, iteratively, until a fixed point is reached. See Botea's work (2007) for a detailed example of the constraint propagation technique that is reused in WOMBATDFS.

Our two-stage approach presented in the previous section requires that the baseline search algorithm implements a pruning technique based on the target score. The higher the target score, the more aggressive the pruning. In WOMBATDFS score pruning works as follows. Given a partial state s, its partial score $g(s)$ is the sum of the lengths of all thematic words in the partially filled grid. In addition, the method implements an admissible (optimistic) heuristic $h(s)$ for the score that could be achieved in the rest of the grid. The heuristic is computed as follows: for each slot that is not fully instantiated, check if its current domain (after constraint propagation is applied) contains at least one thematic word. If so then add the length of that slot to $h(s)$. Now, given a target score t, state s is pruned when $g(s)+h(s) < t$.

Experimental Setup

We run experiments on instances obtained from the Romanian Crosswords Competition. Every year, the best top 12 solutions (grids with black cells and words) are published by Rebus, the competition organizers. We have collected human-designed solutions from nine years: 2007, 2008, 2011, 2013, 2014, 2016, 2017, 2018, and 2019. We then created a testbed with $9 \times 12 = 108$ instances. A given instance takes a human-designed crosswords puzzle and discards all letters, replacing them with blanks. The black cell configuration is preserved. The result is used as a starting point for an AI crosswords puzzle generator such as the methods described in this paper.

We compare our approach OVEREST to a baseline system BASEITER. The latter performs standard searches for a full solution, with initial state set to ε (grid with black cells but no letters). Similarly to OVEREST, BASEITER iteratively runs baseline searches, with decreasing target scores, until a solution if found. See Algorithm 2 for the pseudocode.

Experiments are run in the Compute Canada[2] environ-

[2]https://www.computecanada.ca/

```
Algorithm 2: BASEITER
  Input: Instance with black cell configuration but no
         letters
  Output: Best full solution found, or failure
  /* Search for a full solution       */
1 for $t \leftarrow T$ downto $1$ do
2     $s \leftarrow$ SEARCH$(\varepsilon, t)$
3     if $s$ is full solution then
4         return $s$
5 return failure
```

ment. Each solving process is a serial run on a CPU at 2.10GHz. Each baseline search had a timeout of 3 hours. Solving an instance via iteration of the baseline search had a timeout of 100 hours.

Algorithms 1 and 2 have a number of parameters to set. We set $\Theta = 240, \theta = 180, \ell = 5, T = 215$. That is, target scores in overestimation searches start from 240, decrease by 5 at a time and go down to 180, unless a partial solution is returned earlier (which is typically the case). Searches for a full solution start from a target score of 215, after which it iteratively decreases by 1 at a time. These values are based on the observation that, depending on the year, good scores can start at around 170 and exceed 210. For example, best known scores in our dataset range from 173 to 215.

Setting Θ above 215 (i.e., $\Theta = 240$) allows overestimation searches to build a partial solution. Setting Θ to values above 240 does not add a significant overhead. The higher the target score, the more aggressive the pruning. Thus, the highest target scores induce the smallest search overhead.

Algorithm 1 further requires setting k, the minimum number of variables instantiated in a satisfactory best partial state. We set $k = 15$. Trimming removes the 40% most recent moves which implies that a partial initial state has at least 9 variables instantiated.

The data used in experiments is publicly available at url https://github.com/adibotea/rom-comp-data-wombat. The source code of the systems used in experiments is available on request from the authors.

Empirical Evaluation

Figure 3 compares OVEREST (dual-stage search) to the baseline in terms of CPU time on the subset of instances where either both systems compute a solution with the same score or only OVEREST succeeds. There were no cases where BASEITER succeeded and OVEREST failed. Thus, all remaining cases (not captured in Figure 3) are instances where the two systems compute solutions of different scores. We will address these separately later in this section.

Two main conclusions can be drawn from Figure 3. First, OVEREST is 7 to 224 times faster on the 19 instances where both approaches find solutions of equal score. Secondly, OVEREST finds a solution in many more instances than the baseline. This is quantified more precisely in Table 1.

Table 1 gives a summary of the solved instances, with and without achieving the best known score. The best known score is the maximum among the score submitted in the competition by the human author of the instance at hand, and the two values computed by the two systems, when they find a solution. Remarkably, OVEREST improves the human score in two instances. One of these new records is also matched by BASEITER.

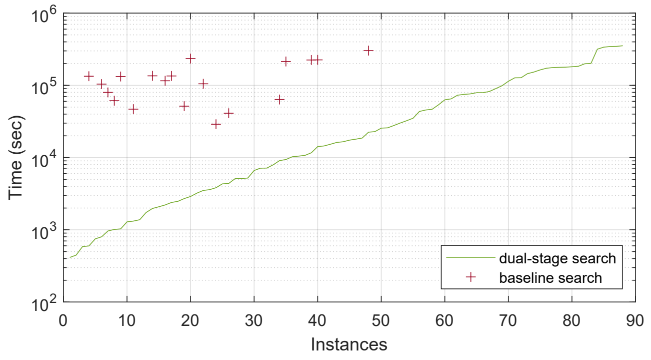

Figure 3: Timings to generate solutions when both systems find solutions with equal scores, or only OVEREST succeeds.

	OVEREST	BASEITER
Solved with best known score	58	23
Solved with weaker score	34	5
Total solved	92	28

Table 1: Summary of solutions found, with and without a best known score. In all but two instances the best known score coincides with the score submitted in the competition.

Table 1 shows that a majority of OVEREST's solutions have a best known score. Compared to human-computed solutions, OVEREST's average score degradation is 1.74%. OVEREST solves 92 instances, compared to 28 by BASEITER.

While OVEREST convincingly outperforms the baseline approach there are five instances for which the baseline produces better solutions than OVEREST. This is due to the fact that the partial initial states found in the overestimation-search stage are not guaranteed to be a subset of a top-quality state. Next we analyse partial initial states in more depth.

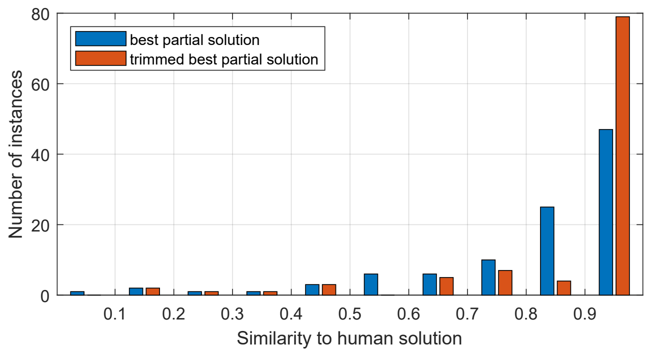

Figure 4: Grid similarities to human-designed grids.

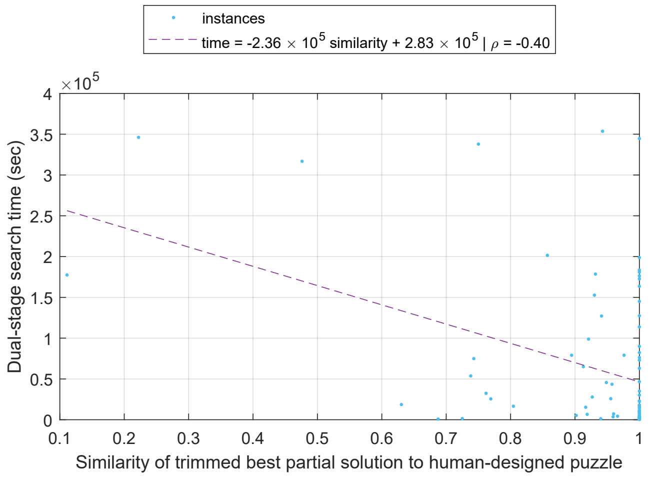

Figure 5: Correlation between the similarity ratio after trimming and the dual-stage search time. ρ is Pearson's linear correlation coefficient. The dashed line shows the least-squares linear fit between similarity and time.

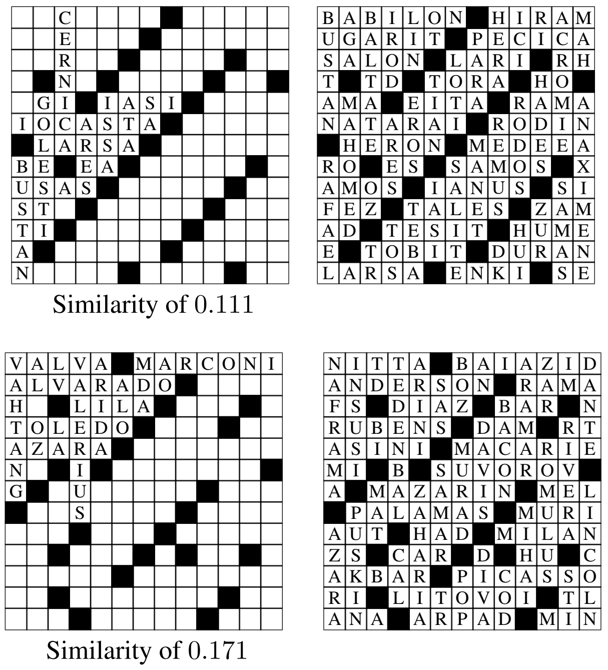

Figure 6: Trimmed best partial solutions (left column) with lowest similarity ratios to human solutions (right column).

Figure 4 plots the distribution of the similarity ratios between a best partial state and a best known full solution. Two variants of such data are plotted, namely with the best partial state before and after the trimming. Given a (trimmed) best partial state and a full state, their similarity ratio is defined as the number of cells where the letter coincides in the partial state and the full state, divided by the number of letters in the partial state. For example, a score of 1 indicates that the partial state is a subset of the full state. In the figure, each histogram bar corresponds to a bucket representing a score range of size 0.1 (i.e., from 0 to 0.1, from 0.1 to 0.2, etc.)

Of course, the system has no knowledge of a best known solution before or during the solving process. The similarity ratios are computed after the system finishes its computation. Therefore, they are a metric that we apply in retrospect to evaluate the quality of the best partial states before and after trimming.

The similarity ratios of the best partial solutions, before trimming, are interesting due to the fact that such best partial solutions are the basis to obtaining the trimmed states (i.e., the initial state in a search for a full solution). If the quality were poor before trimming, it would be difficult or even impossible to obtain a high-quality partial state after trimming. The figure shows that 70.6% of all instances have a similarity ratio of 0.8 or higher before trimming.

Recall that trimmed best partial states are used to initialize a search for a full solution. A similarity ratio of 1 for a trimmed state corresponds to the case where the search for a full solution starts from a subset of the best known solution. We get a similarity ratio of 1 after trimming in 58.8% of instances (not explicitly shown in Figure 4 as the figure groups the data into buckets).

The differences between the similarity ratios before and after trimming further show that trimming works well too, significantly increasing similarity ratios.

Figure 5 shows the correlation between the search time and the similarity ratio after trimming. The larger the similarity, the smaller the search effort. This can be explained as follows. With a higher similarity (e.g., a perfect similarity ratio of 1), the search is more likely to find a solution with a higher score. This implies that the method finishes after fewer iterations over steps 9–12 in Algorithm 1.

Current Shortcomings and Future Work

Figure 4 overall shows good results, but it also points out a few poor cases that require a closer analysis. Indeed, these results are obtained with simple techniques to generate best partial states and to trim these. For example, in an overestimation search, ties among best partial scores are broken at random. Trimming simply erases a percentage of the most recent moves.

Figure 6 shows the smallest two similarity ratios after trimming. In both cases, the low similarity stems from the fact that the best partial solution is a local optimum (i.e., a partial solution with a best possible score for area of the grid filled with letters in that partial solution, that does not necessarily expand into an optimal full solution).

At the top of Figure 6, words IOCASTA and CERNICA in the best partial solution are thematic. Their counterparts in the full best known solution are NATARAI and BALTATE, both being regular. Thus, the best partial solution found is a local optimum. The example at the bottom is similar. Word VAHTANG in the best partial solution found is thematic, whereas word NAFRAMA, the counterpart in the best known full solution, is regular.

A simple way to address such cases in the future is to run two or more independent overestimation searches, each selecting a different variable (slot) to instantiate in the initial state. This could result in partial states with different scores in each search, allowing to select the best one among them. As Figure 4 indicates that such poor similarity ratios are rather exceptional, trying to build several best partial states, in different areas of the same instance, increases the chances that at least one partial state will end up being more useful.

Trimming could potenially be improved by introducing a measure of how tightly or loosely coupled a word is with the rest of the grid. Then, we can trim away loosely coupled words. Another idea is to consider ties among best partial states encountered in an overestimation search. If such tied best partial states have a high similarity with each other, their common part could constitute the trimmed partial state.

We further plan to study what properties of the domain and the test data correlate with the performance of the approach presented, and how these lessons could be used to apply our ideas to other combinatorial optimization problems. Finally, we plan to explore efficient techniques to generating good configurations of black cells.

Complexity

The decision problem based on standard crosswords grid generation (i.e., given an instance consisting of a grid size, a dictionary and a black cell configuration, decide whether the instance admits a solution) is NP-complete (Garey and Johnson 1979; Engel et al. 2012).

It immediately follows that the optimization variant (i.e., given an instance where some words are thematic, and a number n, decide whether the problem admits a solution whose score is at least n) is also NP-complete. Indeed, the problem is in NP, as solutions can be verified in polynomial time. The problem is also NP-hard, being a generalization of the standard one (e.g., if all words are thematic then any valid solution is optimal).

Conclusions

Heuristic search is a leading approach to solving difficult combinatorial optimization problems. We have presented an effective approach to speeding up a baseline search algorithm that can perform pruning based on a target score given as a parameter. Our dual-stage approach first finds a partial state, which can then be used to initialize the search for a full solution. We also presented The Romanian Crosswords Competition, a challenging problem where human champions currently outperform AI. We demonstrated our ideas in instances obtained from the competition top entries, by preserving their black cell configuration and discarding the letters inside. We convincingly speed up a competitive system, solving more instances, and achieving a speedup that can exceed two orders of magnitude.

References

Anbulagan; and Botea, A. 2008. Crossword Puzzles as a Constraint Problem. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming CP-08*, 550–554.

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. 2007. Crossword Grid Composition with A Hierarchical CSP Encoding. In *Proceeding of the 6th CP Workshop on Constraint Modelling and Reformulation, ModRef-07*.

Botea, A.; and Anbulagan. 2009. Analysing the Behaviour of Crossword Puzzles. In *Proceedings of the Symposium on Combinatorial Search, SoCS-09*.

Buro, M. 1997. The Othello Match of the Year: Takeshi Murakami vs. Logistello. *Journal of International Computer Games Association* 20(3): 189–193.

Buro, M. 1999. How Machines Have Learned to Play Othello. *IEEE Intelligent Systems Journal* 14(6): 12–14.

Campbell, M.; Hoane, A. J.; and Hsu, F.-h. 2002. Deep Blue. *Artificial Intelligence* 134(1–2): 57–83.

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.

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.; Frank, M.; Halpin, M. P.; and Torrance, M. C. 1990. Search Lessons Learned from Crossword Puzzles. In *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90*, 210–215.

Land, A. H.; and Doig, A. G. 1960. An Automatic Method of Solving Discrete Programming Problems. *Econometrica* 28(3): 497–520.

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

Mazlack, L. J. 1976. Computer Construction of Crossword Puzzles Using Precedence Relationships. *Artificial Intelligence* 7(1): 1–19.

Meehan, G.; and Gray, P. 1997. Constructing Crossword Grids: Use of Heuristics vs Constraints. In *Proceedings of Expert Systems 97: Research and Development in Expert Systems XIV, SGES*, 159–174.

Schaeffer, J. 2008. *One Jump Ahead: Computer Perfection at Checkers*. Springer Publishing Company, 2nd edition.

Schaeffer, J.; Lake, R.; Lu, P.; and Bryant, M. 1996. CHINOOK The World Man-Machine Checkers Champion. *AI Magazine* 17(1).

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T. P.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the Game of Go Without Human Knowledge. *Nature* 550(7676): 354–359.