Searching for Explainable Solutions in Sudoku

Yngvi Björnsson
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
yngvi@ru.is

Sigurður Helgason
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
sigurdurhel15@ru.is

Aðalsteinn Pálsson
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
adalsteinn19@ru.is

***Abstract*—Explainable AI is an emerging field that studies how to explain the rationality behind the decisions of intelligent computer-based systems in human-understandable terms. The research-focus so far has though almost exclusively been on model interpretability, in particular, on trying to explain the learned concepts of (deep) neural networks. However, for many tasks, constraint- or heuristic-based search is also an integral part of the decision-making process of intelligent systems, for example, in planning and game-playing agents. This paper explores how to alter the search-based reasoning process used in such agents to generate more easily human-explainable solutions, using the domain of Sudoku puzzles as our test-bed. We model the perceived human mental effort of using different familiar Sudoku solving techniques. Based on that, we show how to find an explanation understandable to human players of varying expert levels, and evaluate the algorithm empirically on a wide range of puzzles of different difficulty.**

***Index Terms*—XAI, heuristic search, Sudoku**

I. Introduction

The field of explainable AI (XAI) is concerned with developing techniques that are useful for explaining the reasons behind the decisions of intelligent computer systems, preferably in a way easily understandable to humans. The research focus of XAI has so far mostly been on model interpretability, in particular, providing insights into concepts learned by (deep) neural networks. Given how prevailing such machine-learning techniques have become, this is unsurprising. However, for many tasks, heuristic search is also an integral part of the decision-making process of intelligent systems.

Heuristic search is one of the fundamental problem-solving techniques in AI and computer science. It provides computer-based agents the means of reasoning, or "thinking ahead." One of the main appeals of the approach is its general applicability to decision making in a wide range of disparate problem domains, including manufacturing optimization, automated scheduling and planning, and game-playing. Such informed search techniques are often capable of producing high-quality (real-time) answers to complex decision problems when provided with proper guidance.

Although current research into model interpretability may be applicable for explaining learned heuristic models used by the search, the better part of the heuristic-search reasoning process remains unexplained. The solution to heuristic-search problems typically takes the form of a sequence of actions. For example, a human to fully appreciate the solution might need to know not only how each step came about, but also why that particular sequence is preferable to an alternative one.

In this paper, we take initial steps into making the reasoning process of heuristic search more transparent to humans, using Sudoku puzzles as our test-bed. There are fundamentally two different approaches to achieve this. The former is to treat the solver as a black-box and then try to explain the resulting solution post-mortem; in contrast, the latter alters the solver to produce more human-understandable solutions. Both approaches have their pros and cons, but the latter typically offers greater flexibility in delivering human-explainable solutions, although possibly at the cost of solution quality or run-time performance. We opt for the second approach here as the goal is first and foremost to generate highly informative solutions for humans instead of, e.g., run-time performance. The task of the search process has thus become more intricate: it is looking for not only an acceptable solution to the problem (quality-wise) but also an easily explainable one. Such a solving approach is relevant in settings where a human is expected to verify or execute the resulting solution as well as in settings where a human is hoping to learn from the solving process.

The primary contributions of the paper are: *(i)* we present a heuristic-search-based Sudoku solver that returns solutions understandable to humans with different levels of expertise; *(ii)* a study of numerous problem characteristics that relate to the solution difficulty in our test-bed domain is provided; *(iii)* although the test-bed here is Sudoku, many of the underlying ideas are transferable to a wider-range of problem domains with similar characteristics; *(iv)* and finally, this is one of few recent works emphasizing explainability in heuristic search, and issue that hopefully will attract added research attention.

The paper is organized as follows. Section II introduces both the necessary terminology and preliminaries. Section III explains Sudoku and the basic strategies humans use for solving such puzzles. Related to that, in Section IV, we develop a metric for (objectively) measuring the perceived difficulty humans have in understanding solutions using the human-like strategies. In Section V, we introduce our algorithm for finding the most instructive solution paths, followed by an empirical evaluation of the algorithm in Section VI. Finally, we discuss related work in Section VII and conclude and discuss future work in Section VIII, respectively.

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

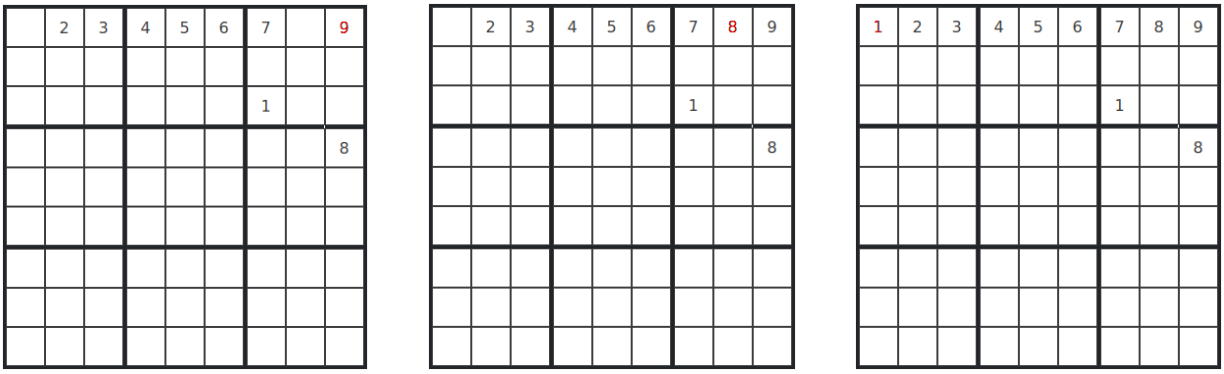

Figure 1. Sole candidates: In the topmost row: 9 is a sole candidate (left), 8 becomes a sole candidate (middle), and finally the 1 becomes one too (right).

II. Background

Heuristic (or *informed*) search methods encompass a wide range of search algorithms. The term is commonly used to refer to state-space-based search methods that rely on heuristics for guiding a search process, such as *A**, *Minimax*, *MCTS*, and backtracking-based *CSP* search. The state-space is typically explored by growing a search tree representing the different possible continuations to take. The problem domains can be single- or multi-agent based, deterministic or non-deterministic, fully observable or not, and the heuristics largely domain-specific (hand-crafted or learned) or largely domain-independent (like variable and action selections in CSP). For example, classic heuristic-search-based planning typically employs single-agent search in deterministic and fully observable domains.

Different search domains face dissimilar explainability challenges. In an adversary-search domain like chess, a human observer might need to realize why a seemingly promising continuation is not taken, for example, because of an unanticipated refutation, whereas in other domains, like Sudoku, the human must first and foremost understand each step in the proposed solving sequence.

Constraint satisfaction problems (CSPs) are a class of problems that can be described by a set of variables, $V = \{v_1, v_2, ...v_i\}$, a set of domains $D = \{d_1, d_2, ..., d_i\}$, where d_j is the set of values that variable v_j can take, and lastly a set of binary constraints $C = \{c_1, c_2, ..., c_n\}$, where $c_j = (v_a, v_b, relationaloperator)$.

A variable v_i is *arc-consistent* with another variable v_j if (and only if), for every value a in d_i there exists a value b in d_j such that (a, b) satisfies the binary constraint between the two variables. A problem is arc-consistent if (and only if) every variable is arc-consistent with every other variable. Although there exist several related algorithms for making a problem arc-consistent, AC-3 [11] is the most widely used.

Sudoku, our test-bed puzzle game, is naturally formalized as a CSP, and Sudoku puzzles are generally effectively solvable using a CSP solver. However, when searching not only for a solution but the most explainable one, then the problem is more naturally formalized as a heuristic-search-based single-agent problem. One distinguishing feature between CSP and heuristic search is that, in the former, the goal state itself represents the solution, whereas in the latter the exact sequence of actions taken that lead to the goal state is the solution.

III. Sudoku

The game of Sudoku is played on a rectangular grid, usually at order 3, meaning that the grid is of size $3^2 \times 3^2 = 9 \times 9$, and overlaid by mutually exclusive boxes (also called blocks or regions) of size 3×3 (see Figure 1). We interchangeably refer to grid cells as variables, as this is how they would be represented in the CSP formalism. The domain of each variable (grid-cell) is 1 to 9 (3^2).

Initially, a partially filled puzzle is provided and the task of the player is to fill out the remaining grid cells such that each number occurs exactly once in each row, column, and box. Sudoku puzzles are traditionally generated such that they have exactly one solution. The puzzles can be of a varied level of difficulty, ranging from being easily solved by humans using only trivial deduction techniques to requiring quite sophisticated levels of reasoning.

A. Human Solving Strategies

Humans do apply various deduction schemes to solve Sudoku puzzles and a rich literature exists (e.g., see [4]) describing a wide array of strategies of different complexity, including with names such as "X-wing", "Swordfish" and "Unique Rectangle". Novice players do typically rely only on a couple of simple deduction strategies, however, as they become more proficient the more elaborate strategies they incorporate into their arsenal of solving techniques. Consequently, when presenting a puzzle solution to a novice player, the solution must be at an appropriate sophistication level for the player to fully comprehend it, for example, by using only familiar deduction strategies. As the players develop their problem-solving skills, more advanced strategies can be incorporated.

We will borrow from the Sudoku literature some of the deduction strategies that are typical for beginner (to intermediate) level players, as presented in the following subsections. We refer to the (other) cells in the same row, column, or box as a cell c as *row-*, *column-*, and *box-neighbours* of cell c,

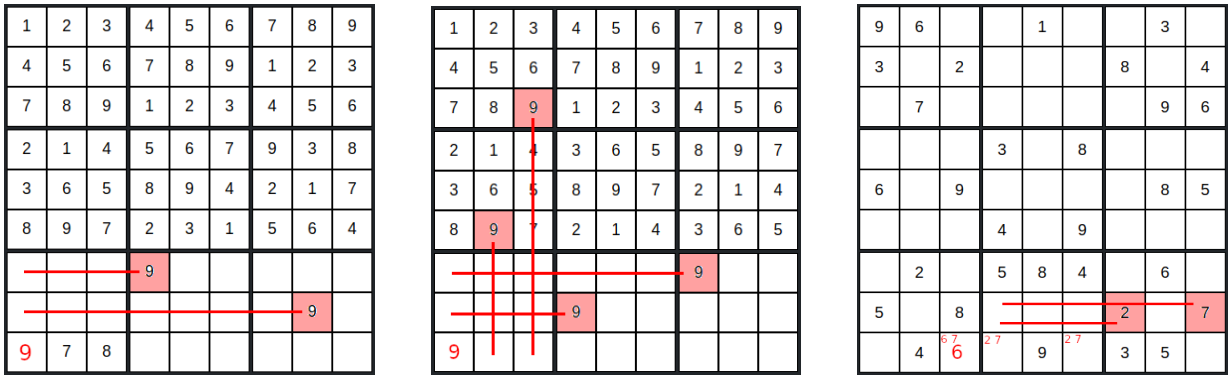

Figure 2. One-dimensional (left) and two-dimensional (middle) unique candidate, and naked-double candidate (right)

respectively. The *neighbours* of a cell c is the union of the row-, column- and box neighbours.

B. Sole Candidate

The *sole candidate strategy* detects a grid cell whose domain has been reduced to a single candidate, for example, as seen in Figure 1. More generally, in detecting a sole candidate all neighbours of a cell c may play a role in restricting the cell's domain. Note, that in the CSP framework sole-candidate domain reduction would be performed via arc-consistency.

C. Unique Candidates

If a value, say v, has been eliminated from the domains of all row-, column- or box-neighbours of a cell c, then we know that cell c must be assigned the value v. The *unique candidate strategy* tries to do such an elimination, for example, as depicted in Figure 2.

We remove values from the box cell's domain, as in Figure 2, by scanning its neighbours. The unique candidate can be found with two levels of difficulty, they can be found by scanning ether one or two dimensions. We refer to candidates found by scanning two dimensions as *two-dimensional unique candidates*, and those found by scanning only one of the two dimensions, we refer to them as *one-dimensional unique candidates*. The reason for this distinction is that it typically requires somewhat less mental effort to scan only one type of neighbours. Analogously, when finding a unique candidate in a row (or a column), we use the box-neighbours and the col-neighbours (or row-neighbours) of the remaining row (or col) cells. All one-dimensional unique candidates are also two-dimensional unique candidates, but not the reverse.

D. Naked Doubles Candidate

Naked doubles is a pair of neighbour cells, say c_1 and c_2, having an identical domain D of size two. This means that each of two values remaining in D must be assigned to one of the two cells. Although, we do not know (yet) which cell should be assigned which value, we know that the two domain values must be reserved for those two cells only. This implies that if c_1 and c_2 are, for example, row-neighbours, then no other cell in that row can take on the two values in D and we can remove the values from their respective domains if present. Analogous arguments apply if c_1 and c_2 are col- or box-neighbours, as seen in Figure 2.

IV. Estimating the Cost of an Explanation

How understandable is a solution to a human player? Ultimately, the answer to this is subjective and depends on the person. Here we try to quantify explainability in somewhat objective terms based on different solution strategies and general principles. Assuming different strategies exist for achieving individual steps of a multi-step plan, for example, the different strategies for deducting an assignment of a Sudoku grid-cell we saw in the previous section.

We define a cost function for measuring a plan's (solution's) explainability — the higher the cost the less understandable the plan is — where we assume the following general principles:

(i) Different strategies may impose different non-negative step costs.
(ii) The same strategy may impose a different step cost depending on the context it is used in.
(iii) Repeatedly applying the same strategy yields a lowered step cost at each successive step (other things being equal). Switching between strategies may involve additional costs.
(iv) Using a strategy not previously used in the solving process may impose an additional (one-time) cost. Thus, using a smaller set of disparate strategies is always preferable (other things being equal).
(v) The total cost of a plan (solution) is the summed cost of its steps.

More formally, taking the above principles into account, we specify our Sudoku solution cost metric as:

$$W_{total} = \sum_{i=1}^{N} \left(\left(1 + \left(1 - \frac{a_{m,i}}{U_i} \right) \right)^{\delta} \theta^{n_{m,i}} w_{m,i} + \xi_{m,i} w_{m,i} \right) \tag{1}$$

where m indicates the strategy used, w_m is the cost of the strategy used (see Table I), N is the number of unassigned variables at the beginning of the puzzle, U_i is the number

Table I
STRATEGY SPECIFIC COST, w_m

	w_m
One-Dimensional Unique Candidate	1
Sole Candidate	3
Two-Dimensional Unique Candidate	5
Naked Double Candidate	10

of unassigned variables at step i, $a_{m,i}$ is the number of available moves for the chosen strategy, δ is the power of the transformation function, θ is a cost decaying factor, $n_{m,i}$ indicates the count of similar moves done before our move m and finally $\xi_{m,i}$ indicates the cost of learning a strategy.

In the first term of our equation, $\left(1 + \left(1 - \frac{a_{m,i}}{U_i}\right)\right)^{\delta}$, we capture the context specific cost (see (ii)) for strategy m at step i, indicating a higher cost with lower availability. The strategy specific cost (see (i)) is lowered by repetitive use of the same strategy through the decaying factor $\theta^{n_{m,i}}$ (see (iii)). The value for the learning factor (see (iv)) $\xi_{m,i}$ was arbitrarily set to 10 if we are using strategy m for the first time at step i, otherwise it is 0.

One can think of the strategy-specific cost in Table I as approximate of the different techniques' sophistication level: novice players easily apply the less costly ones, whereas the more costly require added skill. Solutions, i.e., action sequences, comprised chiefly of simple strategies, are understandable even to less skilled players. In contrast, more advanced players would also comprehend a solution that uses more sophisticated steps. The exact costs listed in the table are per se inconsequential; instead, it is worth noting that different strategies have different associate costs, which may vary depending on the targeted user. For example, given knowledge of the expert level of a user observing the solution, unfamiliar strategies are made costly (even infinite) whereas ones familiar to the user are cheap, thus ensuring that the generated solution uses only strategies familiar to the user.

In this framework, when given a solution, it is naturally explained as the sequence of actions taken annotated with the strategy used and the cells involved. For example, in Figure 1, the explanations would be along the lines of: *9 added as a sole-candidate (using all neighbor-constraints), 8 added as a sole-candidate (using row- and box-constraints), and 1 added as a sole neighbor (using a row-constraint)*.

It is important to note that our solver is not dependent on using this exact cost-function formulation for evaluating its solution paths. In theory, any metric will do that is monotonically non-decreasing with an added number of actions. In practice, however, we would like to capture the idea of varying mental-efforts required for the human to discover and/or understand different solutions, and thus we will model the cost on the above-mentioned principles.

V. METHODS

A backtracking-based CSP solver can be used to solve Sudoku puzzles efficiently. However, here the task is not only to solve the puzzles but to do so in a manner explainable to humans with different levels of expertise. One approach would be to use a standard solver and then try to explain the outcome post-mortem. Unfortunately, this is not guaranteed to succeed as some of the steps taken in solving the puzzle might be too intricate given the ability of the human observer, possibly even based on multiple trial and error as opposed to human-like deduction strategies. The solver must instead be biased towards finding the simplest (best explainable) solution while solving the puzzle using only human-like deduction strategies. Here we present such a solver.

A. Strategy Abstraction

Our approach selects a (variable, value) pair iff that assignment can be deducted using one of the previously mentioned human-like Sudoku solving strategies. Although we lose the ability to solve Sudoku boards that cannot be solved using only those strategies, that is inconsequential as such solutions would be non-explainable. Furthermore, once a strategy has been chosen, say a sole candidate, that strategy is applied repeatedly before potentially switching to a new strategy. A parameter *batch size* controls the number of repetitions.

One can think of a chosen strategy and batch size as a *macro-action*, that is, instead of the action filling out a single grid cell, it fills out multiple cells at once. Not only does such an approach imitate how humans solve the puzzles (as supported by both the Sudoku literature, e.g. [17], and results from a general human-style solving model [13]), but it also reduces the search space significantly (roughly $s^{m/b}$ where s is the amount of Sudoku strategies and m is the amount of empty cells, and b is the batch size). However, using a batch size larger than one does potentially sacrifice optimality; that is, the algorithm might overlook a lowest-cost solution (according to our metric). For example, when exploring a higher-cost action, the algorithm is forced to repeatedly apply that high-cost action, even if lesser-cost actions become available as an aftereffect of a previous application of the high-cost action.

B. The Search Algorithm

The pseudo-code in Algorithm 1 describes the search, where each recursive call to the search function does the following: (1) for each available strategy we apply (up to) *batch size* many moves (not transitively considering those that arise from using the strategy); (2) the cost of the (partial) solution is computed using our cost function; (3) we recurse into the search once more with the changes from applying that strategy; (5) we undo everything from applying the strategy and try the next.

The search is depth-first branch-and-bound-like, keeping track of the least costly solutions found so far and pruning where applicable. It also uses a transposition table, for avoiding reexploring the same state if reached via two different paths, and macro-actions to reduce the search space. A significant characteristic is that the actions allowed for the algorithm are always explainable in terms of Sudoku strategies. Given that the algorithm has at its disposal sufficiently advanced

Algorithm 1 Searching for a least-cost path

map tt {transposition_table}
$f_{best} = \infty$
function Search(s, g, $strategies$)
if Terminal(s) **then**
  {Keep track of best solution so far}
  if $g < f_{best}$ **then**
    $f_{best} = g$
  return 0
if $g \geq f_{best}$ **then**
  {A more costly path, no need to explore further}
  return ∞
if s in tt **then**
  {h* already found for s}
  if $g + tt(s) < f_{best}$ **then**
    $f_{best} = g + tt(s)$
  return tt(s)
$subtree\_costs = []$
for all $strategy$ in $strategies$ **do**
  {Get batch many moves for a given strategy}
  $actions$ = GetActions($strategy$)
  ApplyActions(s, $actions$)
  c = CostFunction(s, $actions$)
  h = Search(s, $g + c$, $strategies$)
  UndoActions(s, $actions$)
  $subtree\_costs.append(c + h)$
$tt(s) = h* = min(subtree\_costs)$
return $h*$

Table II
LEAST COSTLY PATH FOUND TO SOLVE THE BOARD IN FIGURE 3.
COST 33

Available Moves				Chosen Moves		
sole	1d unique	2d unique	naked	Strategy	Value	Square
3	2	5	1	1d unique	5	I1
4	3	6	1	1d unique	5	E3
4	3	7	1	1d unique	8	E1
4	3	6	1	1d unique	2	H1
4	2	6	0	1d unique	8	H3
3	1	5	0	1d unique	3	H7
3	2	4	0	1d unique	3	E8
3	2	3	0	1d unique	2	E7
2	2	2	0	1d unique	2	I8
1	1	1	0	1d unique	7	I7

Table III
MOST COSTLY PATH FOUND TO SOLVE THE BOARD IN FIGURE 3.
COST 342

Available Moves				Chosen Moves		
sole	1d unique	2d unique	naked	Strategy	Value	Square
3	2	5	1	naked	7	I7
2	2	4	0	sole	8	H3
3	3	5	1	naked	5	E3
3	3	4	1	naked	8	E1
2	2	3	0	2d unique	5	I1
2	2	4	0	sole	2	H1
2	2	4	0	sole	3	H7
2	2	3	0	1d unique	3	E8
2	2	2	0	2d unique	2	E7
1	1	1	0	2d unique	2	I8

strategies to progress in each step, it is complete (i.e., always finds a solution).

VI. RESULTS

Here we provide an empirical evaluation of our solver, both in terms of solution understandability and run-time performance. We also look at search-space properties of interest.

A. Experimental Setup

To evaluate our solver, we used an online tool *qqwing* [12] to generate Sudoku puzzles of varying difficulty (simple, easy, and intermediate), the difference being the sophistication level of the strategies required to solve the puzzles. We gathered 50 puzzles for each of the three chosen difficulty levels. All experiments were run on a AMD Ryzen 5950 CPU.

B. Solution Understandability

We start with an example, depicted in Figure 3. The Sudoku board is close to completion with multiple ways to continue. Tables II and III show two such alternative continuations: the former — proposed by our algorithm — uses only a single relatively straightforward strategy, whereas the latter alternates between several strategies of varied sophistication levels, which our algorithm estimates to be the most costly according to our cost function.

Which continuation is more logical? Although there is no definite answer, then the former is easier to comprehend and can be explained to humans of even novice solving abilities (as it requires only one relatively straightforward strategy). There are two one-dimensional unique candidates available to start with, I1 and H7[1]. Our solver proposes to fill I1, then E3 becomes a new one-dimensional unique candidate, and so on until the puzzle is solved. The solution shown in the latter table is much less approachable, requiring jumping between four different strategies and is possibly even incomprehensible to most novice human solvers. On the contrary, expert human solvers might find such an explanation preferable.

[1]We use a board coordinate system where the rows are indexed top-to-bottom by the letters A–I, and the columns left-to-right by digits 1–9.

3	5	7	4	2	1	9	8	6
4	2	1	8	9	6	5	7	3
6	8	9	7	3	5	4	1	2
1	9	3	2	8	7	6	4	5
	6		9	1	4			7
7	4	2	6	5	3	1	9	8
9	7	6	3	4	2	8	5	1
	1		5	7	9		6	4
	3	4	1	6	8			9

Figure 3. Unsolved board with 10 remaining moves

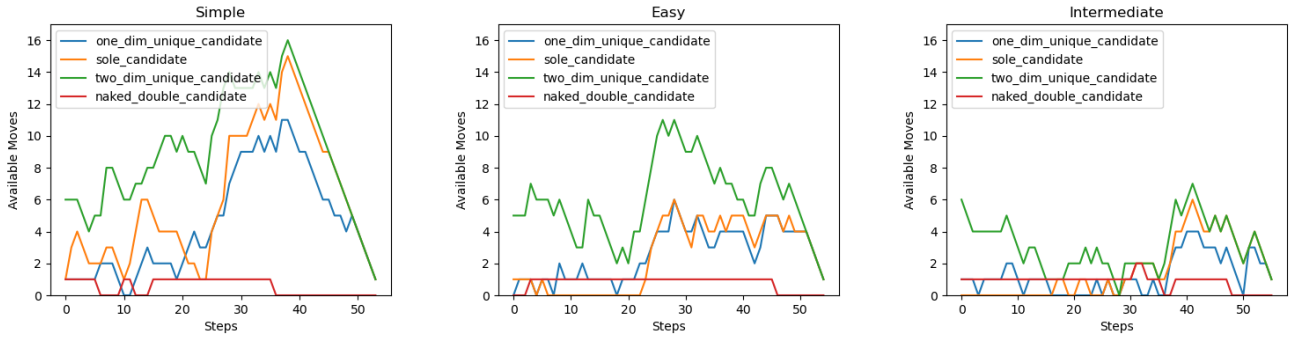

Figure 4. Number of available moves for the least-cost path for three puzzles with different move-availability: simple (left), easy (middle), intermediate (right)

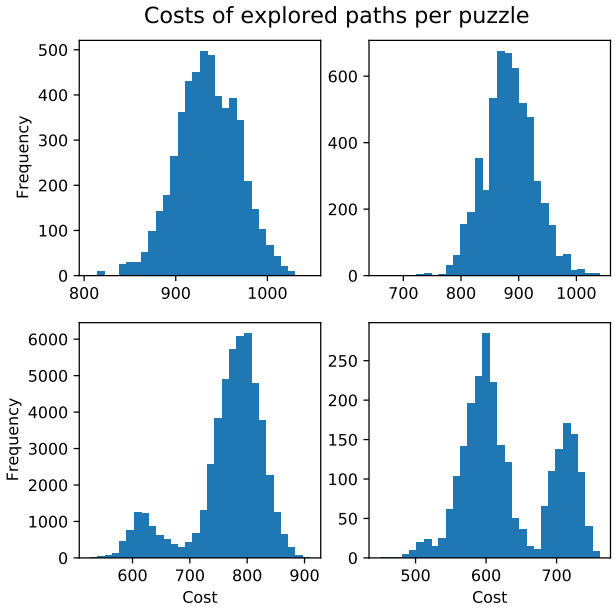

Figure 5. Costs of the various paths explored during the search, for four puzzles.

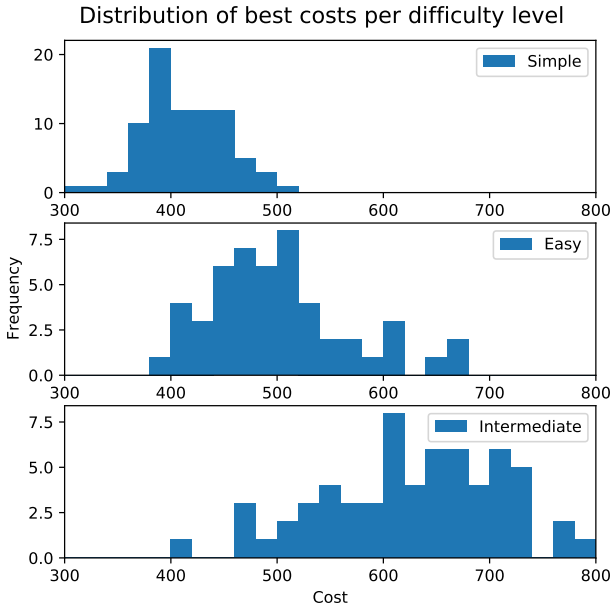

Figure 6. Resulting distributions of the best costs found from solving 50 tables of each difficulty levels: Simple(top), Easy(middle), and Intermediate(bottom)

C. Solution Cost Analysis

The understandability of the possible solutions to a puzzle may vary greatly, as we saw. Figures 4 and 5 cast some more light on this. The former shows the availability of different strategies as the search progresses (the board fills up), whereas the latter shows the cost distributions of all possible solutions to four different puzzle instances. The distributions take different shapes, although in all four the tails are slim, that is, there are only a few very low-cost and high-cost cases.

The availability of strategies (moves) varies throughout the search, this gives us insight into the difficulty of the given solution. A solution that has few available moves throughout the search can in general be considered harder than one with an abundance of possible continuations at each step. For the more simple puzzles the board is open, that is, there are multiple simple moves available, whereas for the intermediate board there is frequently only a few possible continuation. For example, we see in Figure 4 rightmost graph that at one point the only continuation is to use a Naked Double strategy, highlighting the difficulty of that puzzle.

D. Puzzle Difficulty vs. Cost Metric

The puzzles we used for our evaluation were all pre-labeled with a difficulty ranking of simple, easy, or intermediate. Figure 6 shows the distribution of best solution costs, as measured by our cost metric, over all puzzles, and how it relates to the pre-labeled difficulty ranking. The puzzles in the more difficulty-ranked puzzle categories have higher solution costs in general, although there is some overlap. Table IV gives the mean- and median-cost for each category.

Table IV
COST OF DIFFICULTY CATEGORIES

Difficulty	mean	median	std
Simple	411	402	38
Easy	501	496	66
Intermediate	630	632	86

Table V
OVERVIEW OF STRATEGIES BEING USED IN LOWEST AND HIGHEST BEST COSTS FOUND FOR GIVEN DIFFICULTY

	Lowest Best Cost			Highest Best Cost		
	Simple	Easy	Inter	Simple	Easy	Inter
1d unique	17	12	19	13	18	11
sole	15	36	19	11	8	11
2d unique	0	0	0	15	1	14
naked	0	0	0	0	0	1
Cost	319	384	400	500	671	806

To illustrate what the variability in the different costs in Figure 6 might indicate, we show in Table V the solutions statistics for the highest and lowest costs for each difficulty level. The solutions with a lower cost are able to solve the puzzle by using fewer strategies and using the simpler strategies relatively ofter.

E. Run-Time Analysis

One of the configuration parameters to the algorithm is the batch size, that is, the maximum number of actions to perform of a single strategy before considering switching to a different one. A batch size of one indicates that one can freely switch between strategies (as in regular search), a batch size of two makes the algorithm apply that strategy twice (as a single macro-action), if available, before trying the next one, etc. A batch size of infinity means that all available application of a given strategy are played as a single macro-action. The main purpose of the batched actions is both to improve the algorithm's run-time efficiently, but also to have it mimic better how humans typically solve the puzzles (repeatedly apply the same strategy while available). Figure 7 shows the average run-time of the algorithm using different batch sizes. The general trend is that the run-time decreases with increased batch size, however, there is an interesting anomaly when going from batch size one to two, which increases the run-time. [2]

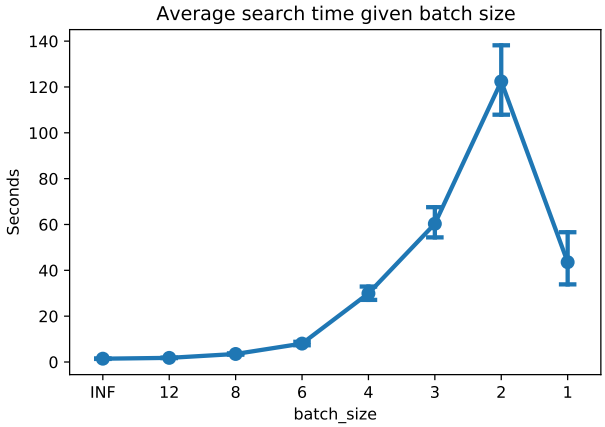

Figure 7. Time / batch size runtime

VII. RELATED WORK

There exists an extensive research literature on model interpretability (e.g., see [8] for a survey) whereas research into the explainability of (heuristic) search is still in its infancy.

In the context of intelligent autonomous agents, particularly in planning [2], the ability of an agent to explain the reasoning behind its decisions has been labeled *explainable agency* [10], and which requires four distinct abilities: (i) the agent must be able to explain decisions made during plan generation, (ii) report which actions it executed at different levels of abstraction, (iii) show how actual events diverged from planned ones and what adaptions were necessary, and, finally, (iv) communicate its decisions and reasoning effectively in a formalism natural to humans. Furthermore, work on explainable agency in systems based on heuristic search tends to distinguish between two types of self-explanations: *process* vs. *preference* oriented. The former emphasizes the (thought) process leading to finding the solutions, whereas the latter focuses on the solutions themselves without concerns about how they were found [10]. In this paper, we focused on the former.

The challenges arising in explaining planning systems are outlined in [5], and the gap between current planning algorithms and human problem-solving acknowledged. Some concrete preliminary steps towards plan explainability are also taken. Interactive (or human-in-the-loop) planning, where computer agents actively participate with humans in the plan generation, do use both some plan-recognition and human-understandable explanation capabilities [3]. Also, recent work on a fully autonomous planning system, which is capable of some rudimentary plan explanations, uses a greedy algorithm to generate so-called minimal plan explanations by searching over the space of abstract models. [16]

For CSPs, the main focus has been on deriving efficient constraint propagation techniques (see e.g. [14] for an overview), with little attention paid to how a solution is found. In many cases, this is perfectly acceptable as the goal is simply to find a good solution (performance-based approach), whereas in other cases —like the one we studied here— such an approach is not helpful because understanding the solution process is the main goal (process-based approach). A few notable exceptions falling into the latter category, include using domain-specific constraints to solve logic puzzles to find more human-like solutions [15], and using an user-understandable tree-hierarchy in an explanation-based constraint programming system [9].

There has also been some recent work in theorem proving systems in generating more human-style proofs for elementary

[2]We have investigated this further, including ruling out (the best we can) that this is caused by incorrect program behaviour. As of now we do not have an bullet-proof explanation for this somewhat unexpected behaviour.

mathematical problems [7].

In Sudoku, the MITS system [1] is an intelligent autonomous tutor that acts interactively with a student to complete a puzzle. It retains a model of acceptable next actions using a dynamic strategy graph that is continuously adapted throughout the game. Either the tutor or the student can direct how to play (mixed-initiative). Good Sudoku [6] is a commercially available Sudoku app. It uses a custom puzzle generator to generate puzzles with varying difficulty, e.g., requiring specific strategies. These puzzles are accompanied by an algorithm that runs in the background and provides hints while the player solves the puzzle.

VIII. Conclusions and Future Work

Many intelligent decision-making systems employ both models and search as an integral part of their decision-making process. The decisions made by such systems must ideally be transparent, verifiable, and, where applicable, instructive for humans. The research focus of XAI has so far mostly been limited to model interpretability, in particular, providing insights into concepts learned by (deep) neural networks.

In this work, we put the focus on the search part of the decision-making process. We use Sudoku as our test-bed and provide a solver — a hybrid of a heuristic- and constrained-based — that biases the search towards finding solutions that are easily explainable to humans with different levels of domain expertise. We provide a concrete method for achieving this and show that such a solver is feasible in our test domain. We also provide an analysis of various aspects of the search- and solution space to better reveal the challenges faced by the solver.

There are several avenues for taking this work further. In particular, it would be of interest to compare more quantitatively the solutions generated by our strategy-aware solver to those generated by a traditional constraint-based Sudoku solver. For example, how understandable are the solutions as judged by our perceived mental-cost metric or, more ideally, by human players. The model we currently use for estimating solution difficulty could also be further refined, for example, based on observing better how humans play. Finally, the domain of Sudoku is just our first stop on the journey into exploring explainability issues in constraint- and heuristic-based search.

The plan also is to generalize the proposed techniques to be more widely applicable. Coarsely speaking, the algorithm, as is, applies to other state-based problem domains where: (i) the search task is to find a solution, as opposed to finding an optimal one, and; (ii) a monotonically non-decreasing cost metric applies for evaluating the perceived understandability of the different solutions path, and (iii) there is a close correspondence between the understandability of individual actions in the solution path and how easily the solution can be explained as a whole (for example, we assume that explaining the solution is achievable by explaining each individual action in the solution independently). Sudoku conforms nicely to the above restrictions and, therefore, is ideal as a first domain for research into explaining heuristic-search processes. A natural next step is to move to a somewhat more challenging search-based problem domain where some of the above conditions are relaxed. For example, such an approach could potentially be useful in planning, in particular for tasks where some subsets of actions are preferable over others from a human-style solving point of view, even though all suffice to solve the planning task.

References

[1] Allan Caine and Robin Cohen. Tutoring an entire game with dynamic strategy graphs: The mixed-initiative sudoku tutor. *JCP*, 2(1):20–32, 2007.

[2] T. Chakraborti, A. Kulkarni, S. Sreedharan, D. E. Smith, and S. Kambhampat. Explicability? legibility? predictability? transparency? privacy? security? the emerging landscape of interpretable agent behavior. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)*, pages 86–96. AAAI Press, 2019.

[3] T. Chakraborti1, K. P. Fadnis, K. Talamadupula, M. Dholakia, B. Srivastava, Jeffrey O. Kephart, and R. K. E. Bellamy. Visualizations for an explainable planning agent. In *Proceedings of the Twenty-Eight International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pages 5820–5822. AAAI Press, 2018.

[4] Conceptis. Conceptis Sudoku solving techniques, 2019. Accessed on 19-11-2019, https://www.conceptispuzzles.com/index.aspx?uri=puzzle/sudoku/techniques.

[5] M. Fox, D. Long, , and D. Magazzeni. Explainable planning. In *IJCAI 2017 Workshop on Explainable Artificial Intelligence (XAI)*, 2017.

[6] Zach Gage and Jack Schlesinger. Good Sudoku is software for generating and solving Sudoku puzzles, 2008. Accessed on 29-06-2021, www.playgoodsudoku.com.

[7] M. Ganesalingam and W. T. Gowers 0001. A fully automatic theorem prover with human-style output. *J. Autom. Reasoning*, 58(2):253–291, 2017.

[8] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv*, 51(5):93:1–93:42, 2019.

[9] Narendra Jussien and Samir Ouis. User-friendly explanations for constraint programming. In Anthony J. Kusalik, editor, *Proceedings of the Eleventh Workshop on Logic Programming Environments (WLPE'01), Paphos, Cyprus, December 1, 2001*, 2001.

[10] Pat Langley, Ben Meadows, Mohan Sridharan, and Dongkyu Choi. Explainable agency for intelligent autonomous systems. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 4762–4764. AAAI Press, 2017.

[11] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[12] Stephen Ostermiller. QQwing is software for generating and solving Sudoku puzzles, 2005. Accessed on 19-11-2019, www.qqwing.com.

[13] Radek Pelánek. Difficulty rating of sudoku puzzles: An overview and evaluation. *CoRR*, abs/1403.7373, 2014.

[14] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[15] Mohammed H. Sqalli and Eugene C. Freuder. Inference-based constraint satisfaction supports explanation. In William J. Clancey and Daniel S. Weld, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 1*, pages 318–325. AAAI Press / The MIT Press, 1996.

[16] S.Sreedharan, S.Srivastava, and S. Kambhampati. Hierarchical expertise-level modeling for user specific robot-behavior explanations. In *Proceedings of the Twenty-Eight International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pages 4829–4836. AAAI Press, 2018.

[17] Paul Stephens. *Mastering Sudoku week by week*. Duncan Baird Publishers, 2007.