The Need to Visualize Sudoku

Navin K. Ipe and Raghavendra V. Kulkarni, *Senior Member, IEEE*

Abstract—Sudoku puzzles are easily solved using backtracking algorithms. Yet, literature is scattered with various shy and often opaque attempts at using evolutionary algorithms and hybridizing them with known strategies of solving the puzzle. Evolutionary methods are serendipitous in nature, and this paper demonstrates the behaviour of such serendipity under constraints, using visuals that depict the sheer magnitude of the problem space and the nature in which intertwined constraints affect the scope for locating a solution, with the hope that it could inspire a new way of looking at the problem. We propose a method of visualizing the sudoku fitness landscape, the vastness and complexity of even partially brute forcing the puzzle, and a unique method of mutating puzzle states using circular swaps. These insights could potentially serve as a link to comprehend the problem space when designing solutions for vast, multidimensional problems. Additionally, finding the optimal solution for some puzzles was notably harder, compared to puzzles in the same category of given clues. A short investigation was conducted into this phenomenon, which revealed hints that compel us to propose that the direction of research that should be taken, is in discovering more about puzzle states and definitive mathematical properties of the puzzle, rather than merely designing brute-force, stochastic or hybrid approaches of finding solutions.

Index Terms—Combinatorics, Constraint Satisfaction Problems, Sudoku

I. INTRODUCTION

Sudoku is a number puzzle, originally consisting of a square grid of 9×9 cells, where each cluster of contiguous 3×3 cells (a sub-grid) contain unique numbers ranging from 1 to 9, such that each row and column of the 9×9 grid also contain unique numbers ranging from 1 to 9. The puzzle is pre-filled with numbers called “clues”, where the number of clues given may range from 17 to 77 [1] [2]. Sudoku resembles a puzzle created in the middle ages, studied in 1782 by the mathematician Euler who christened it the “Latin Square” [3]. Sudoku in its modern form was introduced in 1979 by Howard Garns as a game named “Number Place”. Prior to this, in 1802, puzzle creators in France had attempted removing numbers from magic squares with the same number of squares as the modern sudoku, but with double digit numbers [4]. When modern sudoku got introduced in Japan in 1984, [5], it was christened “*suji wa dokushin ni kagiru*”, meaning “the digits must remain single”, and was shortened to “*sudoku*”, meaning “single number”. Sudoku is an exact-cover type problem that

can be modeled as a constraint satisfaction problem (CSP), and is NP complete. Even the most difficult puzzles can be solved by exhaustive backtracking and pivot-based heuristics.

The purpose of this paper is not about providing an efficient method of definitively solving sudoku puzzles and being better than other algorithms presented in literature. Solving sudoku is easily done via recursive backtracking algorithms or the dancing links algorithm [6] which can solve even 17-clue puzzles in a fraction of a second. This paper does not claim to present an “evolutionary” algorithm either. Evolutionary algorithms are considered good enough if they can provide a near-optimal solution, but sudoku requires an optimal solution. This paper presents the realities of utilizing stochastic approaches in a vast domain. This is achieved by providing a fresh perspective that visualizes the density of puzzle states, the proximity of non-optimal states to the solution and examining how a judicious use of techniques similar to minimum-conflicts [7] CSP, can narrow down and channel the search toward a solution, even when complex heuristics are not used. These visualizations have the potential to assist future work on sudoku algorithms or CSP algorithms where properties of the search space are not well known.

The remainder of the paper is organized as follows: Evolutionary algorithms that attempt solving sudoku and concepts that utilize similar techniques for CSP are referenced in Sect. II. Definitions used, concept introductions and algorithm design are presented in Sect. III. Visualization of the fitness landscape is presented in Sect. III-B4. Trial runs and results of solving the puzzle are presented in Sect. IV. The paper concludes with Sect. V.

II. RELATED WORK

The Soft Computing (SC) paradigm, having the human mind as its role-model has also been referred to as Computational Intelligence (CI) and although it does not have a precise definition, it has been defined in general by Fuzzy Logic, Evolutionary Computation (EC), Machine Learning and Probabilistic Reasoning. These are a class of algorithms that operate on constraints by utilizing a soft stochastic approach that converges to a desired solution. Despite evolutionary algorithms not being appropriate for solving sudoku, various evolutionary and logical inference techniques have been attempted. These range from SAT [8] and simulated annealing [9], to various Evolutionary Computing (EC) techniques like Particle Swarm Optimization (PSO) [10]–[13], genetic algorithms [14] and variants of harmony-search [15]. However, most algorithms do not attempt solving 17-clue puzzles due to the vastness of fitness landscapes containing approximately 3×10^{34} points (the number of permutations of a 17-clue

Authors acknowledge the support received from M S Ramaiah University of Applied Sciences, Bengaluru, India.

N. K. Ipe is with the Department of Computer Science and Engineering, M S Ramaiah University of Applied Sciences, Bengaluru, India (e-mail: navinipe@gmail.com).

R. V. Kulkarni is with the Department of Electronics and Communication Engineering, M S Ramaiah University of Applied Sciences, Bengaluru, India (e-mail: arvie@ieee.org).

puzzle). Among the most promising approaches that solved Sudoku via CI, was the hybridization that used human heuristics and genetic algorithms (GA) [16]. Some authors also utilized logical deduction to augment GA and solved even 21-clue puzzles with 100% success in a matter of seconds [17]. However, it is important to note that even among tough 21-clue or 17-clue puzzles, some puzzles are easier to solve than others. Therefore, it is important to verify the algorithm for various categories of puzzles before making conclusions about its efficacy.

In the constraint satisfaction domain, the minimum conflicts heuristic which solved the eight queens problem, could solve even a one million queens problem in an average of less than fifty steps [7]. However, it was prone to getting stuck at a local optima and required execution of all possible iterations if all queens were initially positioned on the first row. Perhaps a more intelligent approach would be to exploit features of the problem domain, as encouraged by [18] and [9], who also emphasized the importance of initialization, to reduce the search space. An important observation to note was the manner in which random perturbations [19] were capable of solving sudoku puzzles, sometimes offering quicker solutions than evolutionary approaches. The paper questioned the relevance of EC, when a purely random approach could solve even 21-clue puzzles. The utility of uniform randomness to locate optima in vast fitness landscapes was further validated in [20], for a fitness landscape that was similar in vastness to sudoku. Various authors [17] have already remarked on the futility of utilizing evolutionary approaches for solving sudoku, when simpler, quicker techniques like backtracking are sufficient. Despite this, efforts at utilizing hybrid techniques to improve on evolutionary approaches to solve sudoku continue, while tending to avoid difficult puzzles and offering minimal information on how the solution was obtained or how complex the algorithm was. Additionally, the mathematics and computation of sudoku is often presented in a complex symbolic form that provides insufficient perceptual information about the problem space, to anyone who freshly attempts solving the puzzle.

Although this paper began as a challenge to find a new evolutionary method of solving sudoku, the vast literature we came across and our own observations while attempting to solve puzzles, exposed a research gap in the need to be able to visualize the magnitude of the problem space being tackled, the proximity of the optimal solution state to non-optimal states, puzzle densities, categories of solution difficulty and the hard truth that even puzzles that may appear to be simple (due to having a large number of given clues) may in-fact be difficult to solve.

III. DESIGN

A dataset was created, comprising of 9×9 puzzles categorized by the given number of clues. A few famous puzzles like the “easter monster”, “2012’s World’s hardest puzzle by Arto Inkala” and “golden nugget” were manually added to the dataset. Puzzles were auto-selected in the order

they were listed in the one million puzzle dataset [21] and the supersudokulib dataset [22]. A maximum of thirty puzzles were selected for each category. In a few categories, fewer than thirty puzzles were found, so the dataset comprised of a total of 438 puzzles. Figure 9 depicts number of puzzles per category. Results presented in this paper are obtained by attempting to solve puzzles from this assimilated dataset.

A. Definitions

Fig. 1: Some definitions

- **Puzzle** (S , S_E or S_F): The given “empty” Sudoku puzzle (S_E) in which some cells are pre-filled with numbers called “clues”. When S_E is completely filled with non-zero numbers, it is denoted as S_F . The puzzle is referred to as S in contexts where the state of the matrix is known.
- **Individual** (I_i): Each non-intersecting 3×3 group of cells is an individual (Fig. 1a). The puzzle has $n = 9$ individuals, each containing unique numbers ranging from 1 to 9. Individuals I_1 to I_n are depicted in Fig. 1b.
- **Team** (T_i): Although any “leader” individual I_i and the individuals toward the left, right, top and bottom of I_i form a team, any reference to a team in this paper encompasses the individuals other than the leader. For example, in a team with I_1 as the leader, I_1 ’s team members are $T_1 = \{I_2, I_3, I_4, I_7\}$. Similarly, $T_5 = \{I_2, I_8, I_4, I_6\}$ and $T_6 = \{I_3, I_9, I_5, I_4\}$.
- **Frozen cell** (F_i): Any cell in I_i containing a value that is temporarily or permanently fixed in position. Cells with clues given at the start of the puzzle are always kept frozen. Algorithms can temporarily freeze any other cell when required. Frozen cells of a team are depicted with F_T .
- **Non-frozen cell** ($\neg F_i$): Any cell in I_i that is not frozen.
- **No conflict cell** ($\neg C$): A cell containing a number that does not match any frozen number in its row and column, spanning the entire puzzle.
- **Conflicting cell** (C): A cell containing a number that matches the number in any frozen cell along the same row or column of the cell, spanning the entire puzzle (thus violating the rule of requiring unique numbers in every row and column).

- **Obvious cell** (Ω): If any of the non-frozen numbers of an I_i can be inserted into only one $\neg F_i$ (due to being the only cell in which the number would not conflict with any F_T). An S_E matrix filled with Ω 's but is not yet S_F , will be considered as the new S_E . Obvious cells are often referred to as “naked single”, in literature.
- **Cardinality**: The number of elements in a set is the cardinality of the set. For example, the phrase “the number of F_i cells is two” is denoted mathematically as: $|F_i| = 2$.
- **Fitness of an individual** (f_i): The cardinality of C cells in I_i .
- **Total fitness** ($\sum_{i=1}^n f_i$): The sum of fitness values of all individuals. A fully solved puzzle will have $\sum_{i=1}^n f_i = 0$.
- **Circular swap/mutation** (${}_c\Delta_i$): If a number in $\neg F_i$ makes it a conflicting cell, the number is swapped with the number in another $\neg F_i$, but only if after the swap, the swapped cells would be $\neg C$. Figure 1c shows the pattern in which three cell values are swapped. The operation is represented as ${}_3\Delta_i$, where i is the individual id and $c = 3$ (the maximum number of cells undergoing swaps). When $c = 3$, either three cells can be circularly swapped or the numbers of two cells can be swapped. If $c = 4$, the number of cells involved in a circular swap can be 2, 3 or 4. The range of c is $2 \leq c \leq 9$. Swaps can also be referred to as “2-way swaps” or “3-way swaps”.
- **Entropy** (e): The number of times I_i performs ${}_c\Delta_i$.
- **Permutations** ($p \in {}^mP_r$): Permutations of a set of m numbers taken r at a time. Any one of those permutations can be referred to as p .
- **Bruted puzzles** (B , with β indicating the number of bruted individuals in any puzzle in B): Bruted matrices are created when permutations of $\neg F_i$ (generated as $|\neg F_i|P_{|\neg F_i|}$) are inserted into β number of individuals. This is explained in Sect. III-B5. References to “ n brutes” mean that n number of individuals were bruted.
- **Epochs** (E): Each puzzle solution is attempted E times to check if a solution can be found reliably.
- **Generations** (G): The number of times in each epoch, that all teams perform circular swaps (described in Sect. III-B2).

B. Technique

In some puzzles, the number of possible permutations of just one individual may be as high as 3,62,880, with the chances of locating the correct permutation being a mere $1/362880 = 0.0000027$. Considering the entire puzzle, the chances of finding a solution are lower. Therefore, a prudent approach was to reduce the search space by utilizing a few common-sense optimization techniques. The first, was by filling obvious cells (Ω). Once a few obvious cells were filled and frozen, a few more cells often became obvious and they could be filled too. This iteration was continued until no more obvious cells were found. The second optimization was to insert numbers into S_E by inserting numbers only if it would not conflict with any frozen cell of the puzzle. If no such

optimal cells were found, the number was inserted into any available empty cell of the individual. In puzzles with 29 or more clues, the puzzle would often get fully solved during these optimization operations (Fig. 10). The puzzle was then run for $G = 2500$ generations in each epoch of $E = 100$ epochs. Circular swaps were performed on individuals that were not fully frozen, and the individuals were processed in ascending order of the cardinality of non-frozen cells it possessed. Uniform pseudo-random numbers (Mersenne Twister [23]) were used to simulate stochasticity throughout the algorithm. The only exception was during the selection of paths to traverse in the circular swap depth-first search (Fig. 3), where a linear congruential algorithm (from glibc [24]) generated pseudo-random numbers. The random seed was generated when the program was started. The algorithms proposed in this paper are essentially teams of individuals attempting to avoid conflicts with others.

1) *What Makes Puzzles Difficult?*: Literature on evaluating the difficulty of puzzles [25], presents a “refutability” score and a “Richter-type” score [26]. Arto Inkala also presents a formula [27] that incorporates the techniques to make eliminations, the number of links needed to make eliminations and the difficulty of the next step, relative to the prior step. These measures were based on tactics that humans use to solve puzzles. However, evolutionary or random approaches do not depend on such tactics, and are capable of solving puzzles christened “world’s most difficult puzzle”. Moreover, initial trials revealed that locating a solution in some puzzles was significantly more difficult than others, as evident from the outliers in Fig. 11. Initial trials also showed that when any sudoku puzzle with a certain number of given clues is considered as a two dimensional (2D) array, where empty cells are assigned a zero value and cells with clues are assigned the number 1, the resulting 2D array can be represented as a kernel density estimate plot, as shown in Fig. 2. In any given puzzle categorized by the given number of clues, there initially appeared to be a relationship between the number of obvious cells and the density of given clues, the number of peaks of dense areas and even the spread of the given clues in the puzzle.

An investigation was performed, to check if a correlation existed between the chances of obtaining a successful solution versus the spatial arrangement of frozen cells. It was observed that the given number of clues were not necessarily a good predictor of success, and the number of constraints of all team members on an individual could not predict it either. A row and column proximity histogram was generated based on the proximity of each frozen cell to its nearest frozen cell (to obtain a metric of the density and the spread of frozen cells), but a strong correlation could not be obtained even when puzzles were examined by category of given cells (this was examined for frozen cells consisting of just the given clues and also examined for frozen cells after filling obvious cells). Even the number of possible permutations per individual could not provide a strong correlation. Therefore, the difficulty of a puzzle may not necessarily be a factor of how many given

Fig. 2: Kernel density estimate plots of frozen cells. “giv” and “obv” are the given number of clues and the number of obvious cells, respectively.

clues or how many obvious cells can be filled or even the density of the clues scattered across the puzzle. The difficulty appears to be a function of how each individual influences its team members. This relationship could perhaps be derived mathematically, but is not within the scope of this paper. These observations helped design the sub-grids of the puzzles as individuals that were affected directly by the frozen cells of their team.

2) *Designing Team Behaviour*: During multiple initial runs, it was observed that good fitness values were not necessarily an indicator of proximity to an optimal solution (visually depicted in Fig. 8). Therefore, standard evolutionary techniques of retaining the fittest individual and discarding other individuals, was counterproductive, since often, it was necessary for the fitness of S to worsen a little, before it improved and reached the optimal solution. This phenomenon was also observed by other authors [26] who remark on the stochastic continuous time dynamics. Hence, a decision was taken to not have any conditions in the algorithm that retained only the fittest individual. Three behaviors were designed:

- 1) *Be the change*: For each individual I_i , all cells of the team members were temporarily frozen and circular swaps were performed on I_i . Temporarily frozen cells of team members are then un-frozen. The procedure is shown in Alg. 2. This behaviour was designed to check if a greater number of constraints could help an individual converge to a solution quicker.
- 2) *Follow the leader (FTL)*: Each leader individual I_i was frozen and circular swaps were performed on the team members T_i . Temporarily frozen cells were then un-frozen. The procedure is shown in Alg. 1. Since at any given point of time there could be one or more leaders which were closer to the optimal fitness, this algorithm was designed to allow team members to adjust values according to the leader, thus gradually bringing the entire puzzle closer to a solution. Starting the algorithm with individuals that possessed the least

number of non-frozen cells reduced the exploration space since such individuals had fewer permutations of numbers, thus being more likely to reach optimum fitness.

- 3) *Follow the leaders (FLS)*: This algorithm is the same as FTL, but without freezing any individuals. The leader individual and team members were all leaders in their own right, and their circular swaps were based on the already-frozen cells of each row and column they belonged to. This approach was designed to check if mere stochasticity would help find a solution, and also designed to compare FTL with algorithms in literature that utilized pure randomness [19] or evolutionary algorithms. The difference being, that this algorithm utilized circular swaps, thus being less susceptible to a drastic worsening of fitness, compared to a purely random or conventional evolutionary approach.

3) *Designing Circular Swaps*: During initial runs, it was observed that simple “mutations” between any two cells (swapping the numbers in non-frozen cells) of an individual were insufficient to converge difficult puzzles toward a solution. Even after checking whether the two swapped cells would conflict with frozen cells, it was observed that there often were situations where the only way a swap could be performed (without causing conflicts), was if the number in one of the cells could be moved to some other cell. This resulted in the design of swaps that took place in a “circular” manner, as explained in Sect. III-A and Fig. 1c. In order to perform circular swaps effectively, a pre-processing step was performed, where a list of non-frozen numbers were generated for each non-frozen cell, based on whether the number could be inserted into the cell without becoming a conflicting cell. The generated list formed a directed graph, as shown in Fig. 3, where $\{6, 9, 4, 8\}$ are frozen, and $\{1, 5, 2, 7\}$ are non-frozen. The number 1 could be placed in the cell that contains 5 or in the cell containing 2 (hence the arrows from node 1 of the graph, directed to nodes 2 and 5). This could be done without conflicting with any other frozen cell in the puzzle (the entire puzzle is not shown). The number 1 cannot be placed in the cell containing 7, because it would conflict with the frozen cell of another individual (part of the puzzle but not shown in the diagram) that contains 1 in that row or column. It can be observed that 7 and 2 form a loop. This means that a 2-way circular swap is possible between cells 2 and 7. Nodes 1, 2 and 7 also form a loop, revealing that a 3-way circular swap is possible between them. Ideally, some nodes of the graph should have been depicted with self-loops, but it is omitted to avoid cluttering the image. The graphs were created as simple adjacency lists, and circular swaps were found using a depth-first search. Circular swaps were performed only on individuals belonging to teams having a fitness $f_i + f_{T_i} > 0$. The improved fitness when using a larger c is evident in Fig. 4. This is an important point to note when designing algorithms that require escaping local optima.

Fig. 3: No-conflicts graph of an individual

Fig. 4: Circular swap efficacy

4) *Proposed Fitness Landscape Visualization*: Visualizing the fitness landscape for a problem space can help estimate the complexity of finding a solution or assist in designing evolutionary algorithms that could converge to a solution, based on the gradients of the landscape. An attempt was made to visualize the sudoku fitness landscape using various states of S_F . When a puzzle undergoes circular swaps, the puzzle transitions between various states of existence. Each such unique state was recorded. Additionally, puzzle states were also generated by filling multiple copies of S_E with permutations of non-frozen numbers of all individuals. This procedure was performed for three puzzles of 17, 28 and 29 clues each. To keep the quantity of generated data within limits that could be processed, only approximately 15% of the permutations of each individual in the 17 clue puzzle were considered (it generates approximately 4×10^7 puzzle states). For the 28 and 29 clue puzzles, approximately half of 4×10^7 permutations were generated. To generate the fitness landscape points for each puzzle state, the numbers of its first individual were taken in order as shown in Fig. 5. A two dimensional point in space is considered, where, starting from the origin, the first number “1” is used to create a line of length 1 which is generated in the positive x direction at zero angle. The second number “3” is used to generate a line of length 3, starting from the point where the previous line ended. However, from the second number onward, all lines are drawn at an angle $\theta = 360/81$ degree with respect to the previous line, as depicted in Fig. 5 (81 is the number of cells in the

puzzle). After all numbers of the first individual are used in the order $\{1, 3, 2, 5, 4, 9, 6, 8, 7\}$, the numbers in the next individual are used until all numbers of the puzzle have been used. The x, y position of the end of the final line drawn, is considered as the point that the puzzle state occupies on the fitness landscape. The fitness value of the puzzle is calculated and multiplied with -1 , to obtain the position of the point in the z dimension. Thus, the solved puzzle state would be visible as the highest point in the landscape, and the combination of the line lengths and angles ensure that each puzzle state is represented as a unique point in space. Each x and y value was multiplied by 10000, to spread the dense cluster of points in 2D space. The points were also assigned red, green and blue (RGB) codes. The point with $\sum_{i=1}^{i=n} f_i = 0$ was assigned a red colour, points with $1 \leq \sum_{i=1}^{i=n} f_i \leq 4$ were bright green, points with $5 \leq \sum_{i=1}^{i=n} f_i \leq 20$ were dark green, points with $\sum_{i=1}^{i=n} f_i \geq 21$ were shades of grey, where darker shades indicated lower fitness values. The Cloud Compare viewer [28] software was used to visualize the generated points, and the size of each point was increased a little, for better visibility.

The fitness landscape generated is shown as a point cloud in Fig. 6, where in that particular figure, each z value is multiplied by 5000, to visualize the density of points at each fitness level. The top view of the same points (but with z values multiplied by 500) is shown in Fig. 7a. There are more than one red pixels (optimal solutions) in Figs. 6 and 7a since it is a puzzle which has more than one optimal solution (shown in Fig. 12). In Fig. 6, the green and grey points at the higher fitness levels are fewer than the number of points in the lower fitness levels because the fitter S_F states were generated by running the puzzle for 2500 generations of 100 epochs (thus yielding unique puzzle states ≤ 250000), rather than by generating a vast number of permutations. The scarcity of the green points may be an illusion, since all permutations were not generated. An important point to note from Fig. 7 is that the bright green points are not necessarily close to the red points. Also, the light grey and dark grey points are in very close proximity to each other as well as with the green and red points, when considered in the x, y dimension (shown in Fig. 8). The phenomenon of puzzle states appearing to be close to the solution when they were actually not, was also observed by [26]. Fitness value is a poor indicator of proximity to a solution. However, when the puzzle reaches a fitness of approximately 20 or lower, the landscape shows that it may help to explore within close proximity to the existing puzzle state (perhaps by utilizing a lower value of c for circular swaps. Additionally, comparing the variations in puzzle states of Fig. 12 to the positions of red pixels as puzzle states of Fig. 7a, can provide some insight into how the variations in different cells of the puzzle correspond to positions on the fitness landscape. A visual examination of the green points, hinted at the potential for locating patterns that could reveal a fitness gradient. However, the number of points currently generated were insufficient. A small weakness of this method of generating the fitness landscape, is that the

Fig. 5: Fitness landscape points generation

Fig. 6: Side view of point cloud density for 28 clue puzzle

dependence on the order of selecting individuals, can cause large variations in distance in the x, y dimension, for small variations of numbers in the first few individuals considered. For example, if individuals are considered in the order 1 to 9, and a point is generated based on the puzzle state, the point generated by another puzzle state which has a slight variation in numbers of individual 1, will end up reasonably far from the earlier point. Since the variation in the puzzle was small, ideally, the point should be generated closer to the initial point. Perhaps a nine or ten dimensional fitness landscape could mitigate this problem.

5) *Partial Brute Forcing*: In 21-clue puzzles, the sheer number of permutations of non-frozen numbers (calculated as per Eq. 1), is of the order 10^{31} . In 17-clue puzzles, it can reach 3×10^{34} . Therefore, it is insufficient to depend purely on constraint-based stochasticity of the given clues. Figure 10 showed that when the given clues were fewer than 23, the initial optimizations and circular swaps were also insufficient to find a solution. In order to guide the puzzle toward a solution, a strategy of partially brute-forcing a few individuals was considered. To minimize the number of permutations, three individuals with the least number of non-frozen cells (such that $|\neg F_i| \neq 0$) were selected for brute forcing. A copy of S_E is created, and permutations of non-frozen cells are inserted into the first individual selected, as shown in Alg. 3. During the insertion, if any inserted number conflicts

TABLE I: Number of possible brutes vs. valid brutes

Given clues	$ \Omega $	β	possible $ B $	valid $ B $	% valid
30	6	1	6	3	50
30	6	2	36	12	33.33
30	6	3	216	24	11.11
17	2	1	120	20	16.66
17	2	2	14400	400	2.77
17	2	3	72576000	320	0.0004
17	0	1	720	16	2.22
17	0	2	3628800	3840	0.10
17	0	3	18289152000	31104	0.0001

with frozen cells, the permutation is discarded and the next permutation of numbers is attempted. All cells of the newly filled individual are frozen and the remaining empty cells of the puzzle are filled with non frozen numbers. The puzzle is run for E epochs. Next, another copy of S_E is created and the next permutation of non-frozen cells of the individual is inserted and frozen. This is continued until all permutations are exhausted. Then the same process is repeated, but using permutations of the first and second individual. The procedure is repeated again with permutations of all three individuals. Table I shows how only a small fraction of permutations form valid brutes, since most of the permutations would consist of conflicting cells. Among all valid brutes generated, at least one can lead to an optimal solution. Table I, also shows how drastically the number of permutations reduce for puzzles with obvious cells filled. Nevertheless, running the valid brutes is an expensive operation.

An example of brute forcing two individuals: If the $\neg F$ of two individuals I_5 and I_7 are populated with numbers and the cells are frozen, β represents the number of these individuals being brute forced. In this case, $\beta = 2$. If the permutations of non-frozen numbers of I_5 are $\{\{3, 5, 6, 8, 9\}, \{3, 5, 6, 9, 8\}, \dots\}$ and permutations of non-frozen numbers I_7 are $\{\{1, 3, 4, 5, 8, 9\}, \{1, 3, 4, 5, 9, 8\}, \dots\}$, then a copy of S_E is created and $\neg F_5$ is filled with values $\{3, 5, 6, 8, 9\}$ in the same order as listed in the set. The cells $\neg F_7$ are filled with $\{1, 3, 4, 5, 8, 9\}$. The cells filled with permutations are frozen. This newly created puzzle is now a bruted puzzle S_B . If any number contained in the cells of bruted individuals conflict with frozen cells of the puzzle, the bruted puzzle is discarded and the next permutation is attempted. A non-conflicting bruted puzzle has its remaining empty cells filled with non-frozen numbers and an attempt is made to solve the puzzle. To create the next bruted puzzle, another copy of S_E is created and cells $\neg F_5$ and $\neg F_7$ are filled with the next permutation pair $\{3, 5, 6, 8, 9\}, \{1, 3, 4, 5, 9, 8\}$ and so on until all possible permutations are exhausted or until a solution is found. The set of bruted puzzles is B .

$$Total\ permutations = \prod_{i=1}^n |\neg F_i| P_{|\neg F_i|} \quad (1)$$

(a) 28 clue puzzle landscape

(b) 29 clue puzzle landscape

(c) 17 clue puzzle landscape

Fig. 7: Top view of fitness landscapes

Algorithm 1 FollowTheLeader

```

function FOLLOWTHELEADER( $i$ )
  Temporarily freeze  $\neg F_i$ 
  for each  $I$  in  $T_i$  do
    for  $e$  iterations do
      Perform  ${}_c\Delta_I$ 
    end for
  end for
  Unfreeze temporarily frozen cells
end function

```

Algorithm 2 Be the change

```

function BETHECHANGE( $i$ )
  Temporarily freeze  $\neg F$  of  $T_i$ 
  for  $e$  iterations do
    Perform  ${}_c\Delta_i$ 
  end for
  Unfreeze temporarily frozen cells
end function

```

Algorithm 3 Brute permutations

```

function BRUTEPERMUTATIONS( $\beta$ ,  $S$ ,  $B$ )
  if  $\beta == 0$  then return  $B$ ; end if
  Select  $I_i$  for  $\beta$ , based on  $|\neg F_i|$ 
  for each  $p \in {}^{|\neg F_i|}P_{|\neg F_i|}$  do
     $I_i$  of  $S$  is filled with  $p$ 
    if filled cells are  $\neg C$  then
      Filled cells of  $S$  are frozen
       $B = B \cup S$ 
    else exit for loop
    end if
  end for
   $\beta = \beta - 1$ 
  return BrutePermutations( $\beta$ ,  $S$ ,  $B$ )
end function

```

Fig. 8: Zoomed fitness landscape of puzzle from Fig. 7c

Algorithm 4 Main

```

Initialize  $S_E$ . Initialize  $\beta = 0$ .
while  $\Omega$  newly found in  $S_E$  do
   $S_E = S_E$  with newly found  $\Omega$  cells frozen.
  if  $\sum_{i=1}^{i=n} f_i == 0$  then exit program end if
end while
while  $\beta \leq 3$  do
   $B = \{\emptyset\}$ ;  $\beta_{temp} = \beta$ 
   $B = \text{BrutePermutations}(\beta_{temp}, S_E, B)$ 
  if ( $B == \{\emptyset\}$ ) then  $B = \{S_E\}$  end if
  for each  $S$  in  $B$  do
    Insert  $\neg F$  in  $S$ 
    if  $\sum_{i=1}^{i=n} f_i == 0$  then exit  $B$  loop end if
    for each epoch of  $E$  do
      for each generation of  $G$  do
        for each  $I_i$  in order of least  $|\neg F_i|$  do
          if  $f_{T_i} + f_i > 0$  then
            Run Alg. 1 or Alg. 2 for  $i$ .
          end if
        end for
        if  $\sum_{i=1}^{i=n} f_i == 0$  then exit  $G$  loop end if
      end for
    end for
  end while
  if  $\sum_{i=1}^{i=n} f_i == 0$  for all  $E$  then exit while end if
   $\beta = \beta + 1$ 
end while

```

IV. TRIALS

Each of the 438 puzzles were run with the number of brutes β ranging from 0 to 3. The challenge was to see if even the most difficult puzzles could be solved within 500 generations using constraints and stochasticity, while obtaining a realistic picture of how stochasticity within the puzzle manifests itself under constraints. Solutions were considered under four categories, as shown in Fig. 10. These were puzzles solved during Ω cell computation and filling up S_E , puzzles solved using no brutes, one brute, two brutes and three brutes.

A. Trial 1, With All Puzzles

The first trial was run with $c = 6$, $e = 5$, $E = 2500$ and $G = 100$. Individuals in a puzzle were allowed to influence their teams or be influenced using frozen cells as constraints. Sections IV-A1, IV-A2 and IV-A3 list three such methods that attempted to influence $\neg F$ numbers to switch positions until they did not conflict with F cells. Any team with fitness equal to zero, was not subjected to circular swaps. Among the algorithms attempted, “follow the leader” was most successful, as evident in Fig. 9.

1) *Be The Change*: In this method (Alg. 2), the team members of each of the 9 individuals chosen in a generation were fully frozen, and the individual attempted circular swaps within the entropy limit. This method was not very successful since the large number of frozen team members often left very few or no cells for the individual to perform circular swaps. Additionally, the number of swaps performed were lesser than FTL and FLS, thus providing lesser opportunity to locate a solution.

2) *Follow The Leaders (FLS)*: In this method (similar to Alg. 1, but without freezing any individuals), for each of the 9 individuals chosen in a iteration, all team members including the leader performed circular swaps. Every member of the team was therefore a leader that adjusted values according to the frozen cells of all team members.

3) *Follow The Leader (FTL)*: In this method (Alg. 1) for each of the 9 individuals chosen in an iteration, the chosen individual was considered the leader, and all its cells were temporarily frozen. As shown in Alg. 4, the leader individuals were chosen in the order of the individual having the least number of non-frozen cells being selected earlier than the others ($|\neg F_i|$). The team members then performed circular swaps. Although FTL was capable of solving more puzzles, it was interesting to note that there were 36-clue and 30-clue puzzles that did not get solved. In Sect. IV-A1, a large number of frozen members resulted in fewer cells being swappable. Similarly, in many FTL puzzles, the puzzle had a high success rate for 2 brutes and then an extremely small success rate at 3 brutes.

It was also interesting to note that solved puzzles with given clues 17 to 23 were entirely dependent on brute forcing (shown

in Fig. 10). Puzzles with given clues 24 to 28 were solved with or without brute forcing, and a majority of puzzles with given clues ranging from 29 to 37 could be solved as early as at the obvious cells (Ω) stage. It was however surprising to note that even in the 36 clue category, there was a puzzle (Fig. 9c) that required brute forcing. The 36 clue category also had a puzzle that remained unsolved until $G = 10000$ was used (Sect. IV-B). On examining the number of generations it took for puzzles to reach a solution (Fig. 11), it was observed that utilizing 3 brutes, a majority of even the 17 clue puzzles could be solved within 500 generations. The figures also depict how the probability of finding a solution within fewer generations increases with a greater number of brutes (solutions obtained during Ω computation are considered as having being solved in generation zero, as depicted in Fig. 11a). In Fig. 11, the “no brutes” attempt (Fig. 11a) is performed with all puzzles. Puzzles that could not be solved with “no brutes”, are run using one brute (Fig. 11b). Puzzles that remained unsolved are run with two brutes (Fig. 11c) and so on.

Among the 438 puzzles, 45 remained unsolved in trial 1. Based on the number of given clues, the unsolved puzzles were in the given clue categories: {17, 19, 21, 22, 23, 24, 25, 26, 27, 28, 30, 36}, and the cardinality of unsolved puzzles were {12, 2, 2, 1, 1, 4, 5, 5, 2, 8, 2, 1} respectively. There is a lot of literature where authors tend to present only results that achieved a 100% success rate. However, in the true spirit of research, we believe that it is important to also present results that were unsuccessful, since among the unsuccessful results, there are successes which can be compared and utilized to derive further insights.

4) *Unique Solutions*: Ideally, puzzles should have only one solution (referred to as a “well formed” puzzle), but it was interesting to note that some puzzles had more than one unique solution. One such set of solutions is shown in Fig. 12. The fact that these unique solutions were found after ensuring that obvious cells were found and frozen at the start of the algorithm, is perhaps a hint at the need to design and standardize sudoku generation algorithms.

B. Trial 2, With Unsolved Puzzles

Unsolved puzzles of trial 1 were selected and re-run with algorithms FLS and FTL. Entropy was $e = 20$ for both algorithms, where each was run with $c = 6$ and $c = 9$. It was observed that it was not necessary for a fitness value to be close to zero to reach a solution. Puzzles were reaching optimal fitness directly from a fitness values of 16 or 12. As evidenced in Table II, FTL consistently proved to be a better algorithm than FLS, and utilizing a larger c value was not necessarily beneficial (as observed from the number of puzzles solved). However, the success from using a larger entropy value (compared to the value used in trial 1), demonstrated that

Fig. 9: Successful solutions in trial 1, with $c = 6$

TABLE II: Trial 2

Algorithm	c	e	% of puzzles solved
FLS	6	20	2.22
FLS	9	20	2.22
FTL	9	20	44.44
FTL	6	20	46.66

Fig. 10: Categories in which solution was reached first

greater opportunity for each individual to adjust its values to avoid conflicts, benefits such a stochastic search for solutions. Among the 45 puzzles run for FTL with $c = 6$, there were 24 puzzles that remained unsolved. They were in the given clue categories $\{17, 19, 24, 25, 26, 27, 28, 36\}$, with the cardinality of puzzles being $\{9, 1, 2, 4, 3, 1, 3, 1\}$ respectively. Although $c = 9$, it was observed that circular swaps with 8 or 9-way swaps were almost negligible, since for such swaps to take place, it was necessary for individuals to have 8 or 9 non-frozen cells and constraints from team members that allowed such a swap to take place.

An additional trial was run with the 24 unsolved puzzles using FTL, to check if a greater number of generations $G = 10000$, a low entropy $e = 5$ and large circular swaps $c = 9$ would solve the puzzles. Unsurprisingly, 10 puzzles were solved, with unsolved puzzles being in the given clue categories $\{17, 19, 25, 26, 27, 28\}$ with cardinality of puzzles being $\{5, 1, 2, 2, 1, 3\}$, respectively. The remaining 14 puzzles could have been solved if the algorithm was simply run a few more times. However, the objectives of the trials were to obtain a clear picture of the extent of success (under various puzzle categories) an algorithm can achieve.

V. CONCLUSION

Various decisions were taken during the literature review and initial trial runs, that shaped the design of the objectives and algorithms presented in this paper. Rather than merely present successful results, it was important to investigate the properties of the puzzle that influenced the capability to locate a solution. Although the pattern of arrangement of clues or the quantity of clues may be assumed to play a role, we conclude that it may not necessarily be true. It was when puzzles were put under a higher order of constraints by bruteing, that we observed anomalies like the 36-clue and 28-clue puzzles remaining unsolved even with three brutes, while a majority of the 17-clue puzzles were solved. So rather than present only the successful solutions, we believe it is important to candidly present the manner in which puzzles remained unsolved, since they are an anomaly that potentially validates our initial suspicion that it is not merely the pattern of arrangement of clues in the puzzle that matters, but the pattern in which the clues place each team along a path on the fitness landscape, where the swapping of numbers is more likely to find a direct route toward the solution. There was a consistency with which some puzzles ended up with a solution while some didn't, and merely the fitness values could not predict the probability of reaching a successful solution. Any future work on sudoku would benefit from examining this property of finding the right path on the fitness landscape.

A. Primary Contributions of This Paper

- The utility of constraint-aware c -way circular swaps as a mutation technique that could potentially improve the

(a) Using no brutes

(b) Using 1 brute

(c) Using 2 brutes

(d) Using 3 brutes

Fig. 11: Trial 1: Generations needed to find a solution during various epochs

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

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

Fig. 12: Multiple unique solutions for a 28-clue puzzle

- Sudoku fitness landscape visualizations that clearly depict the vast, dense problem space and the proximity of sub-optimal solutions to the optimal solution.
- Visually and numerically demonstrating the extent by which the problem space is narrowed down (box-plots in Fig. 11, Table I and Fig. 10) by utilizing partial brute forcing.

Other important observations noted were the large number of outliers in Fig. 11 and the few puzzles that remained unsolved even after 10000 generations were run (surprisingly even with 28-clue puzzles). The need for partial brute forcing also showed that a purely evolutionary approach is insufficient to solve hard sudoku puzzles. It is imperative to couple the algorithm with heuristics or improved logic to create a hybrid algorithm. Sudoku rules and heuristics have been studied by academics and are well known. However, when requiring to solve any other vast constraint-based NP complete problem whose properties have not been studied, the methods presented

capability of evolutionary algorithms (not only for solving sudoku) that need to escape local optima.

in this paper can provide some basic insight into how to tackle the problem and reach a solution.

B. Future Work

- Designing a gradient calculation or a path, based on an improved fitness landscape and given clues, could potentially assist in improving algorithms that could use the gradient or path to converge toward an optimal solution.
- Starting from an optimal puzzle state, a fitness landscape could be reverse-generated by varying puzzle states and observing the properties of the variations. It may very well be possible to also design an algebraic solution to solving the puzzle, by observing such patterns.
- Sudoku solving algorithms could additionally be compared based on the Big- O notation, rather than merely based on number of clues or fitness or time taken to solve the puzzle.
- A “tiredness” feature could allow the algorithm to decide when to stop using stochastic search and begin brute forcing. Ideally, such a program should be designed to understand the problem domain and be given sufficient time to formulate its own solution tactics. Similarly, the number of c -way swaps could initially be kept high, and as the puzzle approaches closer to a solution, the value could be decreased, to allow exploring within a limited range.
- Some 36-clue and 28-clue puzzles that were hard to solve, could be compared with other puzzles of the same category to analyze what makes them different. The 17-clue puzzles that were simple to solve could also be compared to 17-clue puzzles that did not get solved.

On an anecdotal note, a philosophical conclusion that could be derived from the results of running the algorithms in this paper is that individuals who try to “be the change” they would like to see in society (Alg. 2) would not be as successful in improving a group or society, as compared to a society that follows a succession of leaders (Alg. 1) who bring about changes, and although the leader need not always be correct, the interconnectedness of society and the presence of strong values (the given clues) among individuals leads to an eventual convergence to an optimal state.

REFERENCES

- [1] G. McGuire, B. Tugemann, and G. Civario, “There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration,” *Experimental Mathematics*, vol. 23, Jan. 2012. I
- [2] D. A. P. (https://math.stackexchange.com/users/31718/daniel-a-pelsmaecker), “Maximum number of clues in a sudoku game that does not produce a unique solution,” Mathematics Stack Exchange, uRL:https://math.stackexchange.com/q/345255 (version: 2013-03-29). [Online]. Available: https://math.stackexchange.com/q/345255 I
- [3] H. F. MacNeish, “Euler squares,” *Annals of Mathematics, Second Series*, vol. 23, pp. 221–227, 1922. I
- [4] B. Christian, “Supplément de l’article ‘Les ancêtres français du Sudoku,’” Pour la Science, 2006, archived from the original. [Online]. Available: http://cboyer.club.fr/multimagie/SupplAncetresSudoku.pdf I
- [5] J.-P. Delahaye, “The science behind Sudoku,” *Scientific American*, pp. 80–87, 2006. I
- [6] D. E. Knuth, “Dancing links,” *Millennial Perspectives in Computer Science*, vol. 1, pp. 1–26, Jan 2000. I
- [7] M. D. Johnston and H.-M. Adorf, “Learning in stochastic neural networks for constraint satisfaction problems,” in *Proceedings of the NASA Conference on Space Telecommunications*, 1989. I, II
- [8] T. Weber, “A SAT-based Sudoku solver,” in *Proceedings of the 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, 2005, pp. 11–15. II
- [9] R. Lewis, “Metaheuristics can solve Sudoku puzzles,” *Journal of Heuristics*, vol. 13, no. 4, pp. 387–401, Aug. 2007. II
- [10] T. Mantere and J. Koljonen, “Sudoku solving with cultural swarms,” in *Proceedings of the 13th Finnish Artificial Intelligence Conference (AI and Machine Consciousness)*, 2008, pp. 60–67. II
- [11] A. Moraglio and J. Togelius, “Geometric particle swarm optimization for the Sudoku puzzle,” in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2007, pp. 118–125. II
- [12] J. M. Hereford and H. Gerlach, “Integer-valued particle swarm optimization applied to Sudoku puzzles,” in *Proceedings of the IEEE Swarm Intelligence Symposium*, Sep. 2008, pp. 1–7. II
- [13] S. McGerty, “Solving Sudoku puzzles with particle swarm optimisation,” 01 2019. II
- [14] M. Nicolau and C. Ryan, “Solving Sudoku with the GAUGE system,” in *Genetic Programming*, P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 213–224. II
- [15] O. Abdel-Raouf, M. Abdel-Baset, and I. Henawy, “A novel hybrid flower pollination algorithm with chaotic harmony search for solving Sudoku puzzles,” *International Journal of Engineering Trends and Technology*, vol. 7, pp. 126–132, 03 2014. II
- [16] N. Pillay, “Finding solutions to Sudoku puzzles using human intuitive heuristics,” *South African Computer Journal*, vol. 49, pp. 25–34, 2012. II
- [17] M. Becker and S. Balci, “Improving an evolutionary approach to sudoku puzzles by intermediate optimization of the population,” in *International Conference on Information Science and Applications*. Springer, 2018, pp. 369–375. II
- [18] S. Jones, P. Roach, and S. Perkins, “Construction of heuristics for a search-based approach to solving Sudoku,” in *Proceedings of the 27th SGAI International Conference on Artificial Intelligence Cambridge, England*, 01 2007, pp. 37–49. II
- [19] S. McGerty and F. Moisiadis, “Are evolutionary algorithms required to solve Sudoku problems?” in *Computer Science & Information Technology*, vol. 4, 02 2014, pp. 365–377. II, 3
- [20] N. Ipe, “An in-memory physics environment as a world model for robot motion planning,” 2020. II
- [21] K. Park, “1 million sudoku games,” Dec 2016. [Online]. Available: https://www.kaggle.com/bryanpark/sudoku III
- [22] Y. Cao, “Benchmarking Sudoku solvers,” 2008. [Online]. Available: https://in.mathworks.com/matlabcentral/fileexchange/18921-benchmarking-sudoku-solvers. III
- [23] M. Matsumoto and T. Nishimura, “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator,” *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998. III-B
- [24] “The gnu c library (glibc).” [Online]. Available: https://www.gnu.org/software/libc/ III-B
- [25] R. Pelanek et al., “Difficulty rating of sudoku puzzles by a computational model,” in *FLAIRS Conference*. Citeseer, 2011. III-B1
- [26] M. Ercsey-Ravasz and Z. Toroczkai, “The chaos within Sudoku,” *Scientific Reports*, vol. 2, pp. 1–8, 2012. III-B1, III-B2, III-B4
- [27] A. Inkala, “Ai sudoku puzzle difficulty,” [Online]. Available: http://www.aisudoku.com/index_en.html III-B1
- [28] “Cloud compare.” [Online]. Available: https://www.danielgm.net/cc/ III-B4