University of Leeds

MATH3001: Project in Mathematics

School of Mathematics

Automatic Puzzle Solving

Author:
Matthew Gordon
SID - [redacted]

Supervisor:
Dr Adrian Martin

March 28, 2024

Contents

1 Preliminaries

1.1 Abstract

Killer Sudoku puzzles present a captivating amalgamation of mathematical operations and deductive reasoning, offering enthusiasts an intellectually stimulating challenge within the realm of grid-based puzzles.

The primary objective of this report is twofold: firstly, to devise a sophisticated algorithm capable of crafting Killer Sudoku puzzles of unparalleled quality, characterized by their intricate yet solvable nature. Secondly, to undertake a systematic examination of various solving methodologies, aiming to discern the most effective strategies for untangling these complex puzzles.

Harnessing computational tools and mathematical insights, this report seeks to contribute invaluable insights to both puzzle enthusiasts and computational researchers, illuminating the intricacies of Killer Sudoku puzzles.

1.2 Acknowledgments

I would like to thank Dr Adrian Martin for his guidance and support throughout this project.

1.3 Ethical Implications

The development of a program capable of generating and solving Killer Sudoku puzzles presents several ethical considerations that merit thoughtful examination. Below are key points outlining the ethical issues associated with such a program, paired with my contentions for each point raised.

1. **Intellectual Property Rights:**
   While accrediting third party puzzle generators for their contributions is essential, the use of randomly generated puzzles in this report mitigates concerns regarding intellectual property rights.

2. **Impact on Human Creativity:**
   Automating puzzle generation may raise concerns about diminishing opportunities for human puzzle creators and solvers. However, existing platforms like Sudoku.com [2] already employ generators, contributing to the puzzle's popularity and allowing users to experience the joy of solving. Thus, creating a generator could further enhance puzzle popularity without significantly impeding human creativity, given the precedent set by established platforms.

3. **Fairness in Competitive Settings:**
   In competitive puzzle solving environments, the reliance on automated tools could potentially create disparities between participants. However, the program's availability does not inherently compromise fairness, as competitors can opt for manual methods. It's crucial to note that reputable competitions typically prohibit the use of technological aids, ensuring fairness through strict regulations and oversight by adjudicators.

4. **Dependency and Skill Development:**
   While there's a risk of over reliance on the program leading to a decline in manual puzzle solving skills, the overall impact is likely positive. Exposure to puzzles through the program may motivate more individuals to engage with and enjoy them, potentially outweighing any reduction in manual solving proficiency.

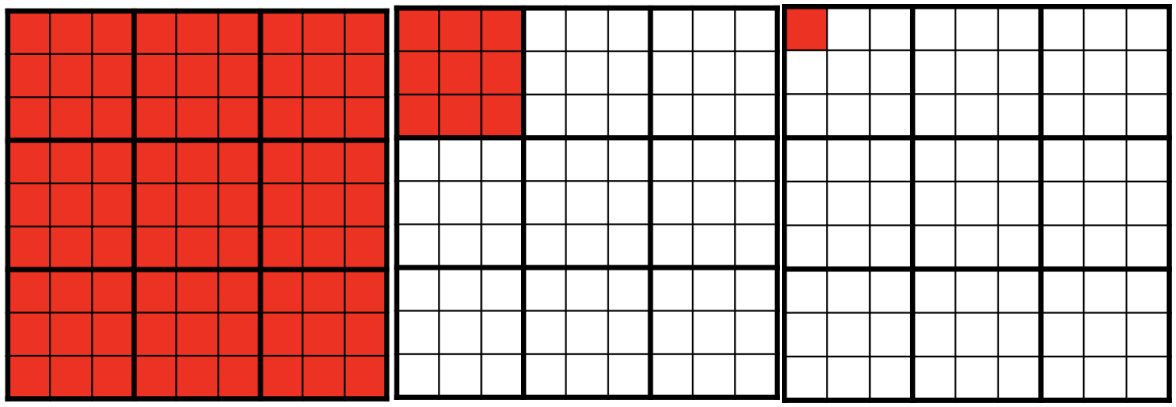
5. **Ethical Use:**
   Designing the program with a focus on responsible use is paramount, prioritising user well-being and mitigating potential addictive tendencies. However, given the recreational nature of Killer Sudoku puzzles, the risk of addiction is likely minimal compared to more inherently addictive forms of entertainment.

In conclusion, although the development of a program aimed at generating and solving Killer Sudoku puzzles raises various ethical considerations, I contend that these concerns are sufficiently mitigated within the scope of this project to justify its creation.

1.4 Nomenclature

In this report, there is a noticeable volume of terms lacking precise definitions, potentially causing confusion. For instance, the term "cage" remains ambiguous and could lead to varied interpretations. In order to properly describe Sudoku puzzles, we need some consistent terminology.

Notation 1.1 The full 9×9 Sudoku puzzle grid will be called the "Grid".
Notation 1.2 The 3×3 mini-grid below will be referred to as a "box".
Notation 1.3 The individual piece pictured below will be referred to as a "cell".



(a) Grid. (b) Box. (c) Cell.

Figure 1: Grid Box Cell.

Notation 1.4 A "list of potentials" is a list that contains all possible values that the cell in question can take.
Clarification: Although Sudoku can come in a multitude of grid sizes, 9x9, 16x16 etc. For the purposes of this project we only focus on the 9x9 grid size case as showcased in the figures above.

2 Background

2.1 Sudoku

Sudoku is a highly popular puzzle game characterized by a square grid structure in an n×n format. The objective is to fill the grid with numerical digits from 1 to n, ensuring that each row, column, and $\sqrt{n} \times \sqrt{n}$ sub grid contains each digit exactly once. This principle of non-repetition within rows, columns, and sub grids forms the core of Sudoku game play.

Any integer value of n with an integer square root can be utilized to generate a valid Sudoku puzzle. However, in this project, we primarily focus on n=9, which is the most commonly employed value [9].

At the start of a Sudoku puzzle, a portion of the cells are filled in advance with digits, serving as initial clues for the player. These clues typically occupy about one-fourth to one-third of the total grid.

Players tackle Sudoku puzzles by employing deductive reasoning and systematic elimination techniques to determine the correct placement of digits within the grid. By logically deducing and recognizing patterns, players progressively fill in empty cells with the appropriate numerical values.

One of Sudoku's key features is its adaptability to various levels of difficulty. Puzzles range from straightforward challenges suitable for beginners to intricate configurations that challenge even seasoned players. Despite these differences in difficulty, all Sudoku puzzles adhere to the same fundamental rules and mechanics. The following is an example of a typical Sudoku grid.

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

Figure 2: A typical Sudoku puzzle [9].

2.2 Killer Sudoku

In Killer Sudoku, participants are confronted with a grid partitioned into distinct regions termed "cages", each comprising a predefined target sum and a unique subset of cells. The primary objective mirrors that of conventional Sudoku: to populate the grid with integers ranging from 1 to n (typically 9), ensuring that every row, column, and sub grid contains each digit precisely once[2].

However, in Killer Sudoku, additional criterion are introduced: players must guarantee that the numerical values within each cage, when subjected to the designated arithmetic operation, result in the specified target sum assigned to that particular cage. This operation is addition for the purposes of this report and is the operation most commonly used amongst Killer Sudoku puzzles. The second criterion is that no numerical value can appear more than once within the cells of a cage.

Some Killer Sudoku puzzles provide starting numbers or "clues" that initially populate the grid. For this project we look at puzzles with no such starting clues[3]. As such the puzzles we will be investigating will have a format similar to the one shown below.

Notion 1.5 The various coloured boxes on the grid below are referred to as "cages" throughout.

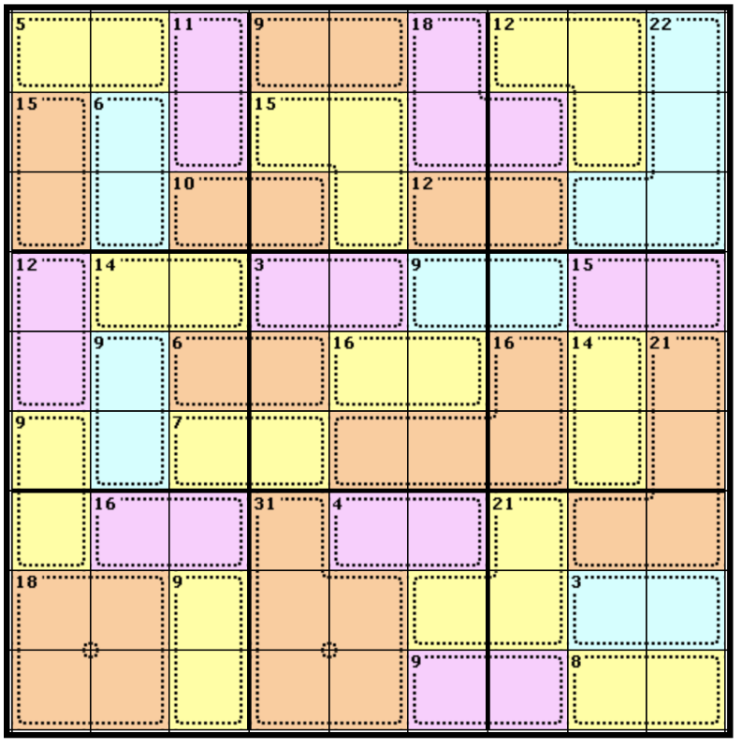


Figure 3: A typical Killer Sudoku puzzle[3].

3 Generator

Inputting Killer Sudoku puzzles without a Graphical User Interface (GUI) can be an arduous task. To streamline this process and facilitate testing of the Solver, a puzzle generator capable of producing random puzzles of various difficulties was developed. This generator not only expedites the tedious puzzle input process, but also enables the testing of the Solver on numerous puzzles efficiently. Otherwise, manually inputting hundreds of puzzles would be time-consuming, requiring verification of unique solutions and substantial effort in sourcing and inputting a sufficiently large sample size for meaningful analysis.

The design of the Killer Sudoku puzzle generator can be deconstructed into four key stages:

1. **Generation of Seed Cells:**
   Initially, the generator must randomly select a set of seed cells from the grid. These cells serve as the starting points for building the cages.

2. **Growing the Cages to Fill the Grid:**
   Using the selected seed cells as anchors, the generator expands each cage by adding adjacent cells. This expansion continues until the entire grid is filled, ensuring that each cell belongs to exactly one cage.

3. **Assigning Cage Values:**
   After the cages are formed, the generator assigns meaningful target values to each cage. These values are determined in a way that adheres to the rules of Killer Sudoku, ensuring that each cage's total is achievable through the numbers in its cells without violating any constraints.

4. **Verifying a Unique Solution:**
   Finally, the generator must validate that the puzzle it has created possesses only one possible solution. This involves employing techniques such as constraint propagation and backtracking to ensure that no alternative solutions exist beyond the intended unique solution.

By carefully executing these four stages, the generator can produce high-quality Killer Sudoku puzzles that can rigorously test the Solver.

The puzzle Generator exclusively produces grids devoid of starting clues, opting instead to utilise cages of size 1. In this method, a cell within a cage of size one inherently serves as a clue, as it must be equal to the value assigned to the cage. Consequently, these solitary cages function identically to traditional clues, effectively initialising the grid.

To establish a connection between cage positions and corresponding cells on the starting grid, a list named "cages" was introduced for storing this crucial information. Each entry within "cages" represents a distinct "cage", comprising a list of coordinates denoting the cells contained within that specific cage. This strategic approach facilitates easy retrieval of cells within each cage from the starting grid, enabling streamlined operations such as accessing cell values via commands like "grid[cage[0]][cage[1]]".

```
grid = [[0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

(a) Starting Grid.

```
cells = [[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5], [0, 6], [0, 7], [0, 8],
         [1, 0], [1, 1], [1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [1, 7], [1, 8],
         [2, 0], [2, 1], [2, 2], [2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, 8],
         [3, 0], [3, 1], [3, 2], [3, 3], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8],
         [4, 0], [4, 1], [4, 2], [4, 3], [4, 4], [4, 5], [4, 6], [4, 7], [4, 8],
         [5, 0], [5, 1], [5, 2], [5, 3], [5, 4], [5, 5], [5, 6], [5, 7], [5, 8],
         [6, 0], [6, 1], [6, 2], [6, 3], [6, 4], [6, 5], [6, 6], [6, 7], [6, 8],
         [7, 0], [7, 1], [7, 2], [7, 3], [7, 4], [7, 5], [7, 6], [7, 7], [7, 8],
         [8, 0], [8, 1], [8, 2], [8, 3], [8, 4], [8, 5], [8, 6], [8, 7], [8, 8]]
```

(b) Cell Coordinate Grid.

Figure 4: Starting Grid & Cell Coordinate Grid.

3.1 Generation of Seed Cells

During the cage creation process, the generator faces the initial challenge of determining the optimal number of cells to select and whether specific locales should be targeted. This prompts considerations regarding the selection criteria and the ideal quantity of cells to ensure puzzle quality. Analysis of methodologies employed by respected generators, such as those showcased on [2] and [7], revealed a common trend of generating puzzles with 28 to 34 cages. Drawing inspiration from this observation, the generator was configured to produce puzzles with a slightly broader range of cages, spanning from 35 to 46.

Regarding the selection of seed cells' locales, the current approach of random selection occasionally resulted in clustering issues, hindering grid filling due to the concentration of seeds in close proximity. To address this, the generator incorporated a predefined list named "starter seeds", comprising cells positioned around the edges of the grid. However, upon reflection, a more effective strategy would involve ensuring a more even spread of seed cells. This could potentially be achieved by randomly selecting one seed cell from each row and column in a systematic manner, thereby enhancing the distribution across the grid.

To prevent the selection of duplicate cells, the generator removes chosen cells from the sample list once they are designated as seed cells. The coordinates of the chosen cell are then added to both a list named "seeds" and a list named "cages", facilitating their utilisation for further growth in subsequent stages.

3.2 Growing the cages

To correctly achieve the growth of cages, it's essential to identify the neighbouring cells each cage can potentially absorb, considering that cages can only encompass adjacent cells. The function "find neighbours" accomplishes this task by taking a cage as input and outputting all the eligible cells it could grow to.

Subsequently, the process involves randomly expanding each cage until the entire grid is filled. This task is achieved by the function "generate cages", which randomly selects a cage, expands it to an eligible neighbouring cell, and repeats this process until the grid reaches full occupancy.

3.3 Assigning cage values

To assign meaningful values to the cages, the process involves generating a filled grid that adheres to the rules of Sudoku. Subsequently, the values within each cage are summed, and this total is assigned as the cage value.

Ensuring the absence of duplicate values within each cage is crucial. In the event of duplicate detection, the current methodology involves regenerating both the cages and the grid from scratch. However, an alternative optimisation could involve regenerating only the grid, thus conserving computational resources. This approach was not initially chosen because solely regenerating the grid may not guarantee a proper solution, as the issue could lie with the cages. Similarly, regenerating just the cages while retaining the grid may not ensure a valid puzzle. Thus, starting afresh was deemed the most efficient way to obtain a functional puzzle.

3.4 Unique Solution Verification

In Killer Sudoku, each cell can accommodate a number from 1 to 9, and through a combination of the rules of Killer Sudoku and logical deduction, the list of potential numbers for each cell can ideally be narrowed down to a single possibility. However, some puzzles may possess multiple solutions, undermining the effectiveness of the solver as they rely on guesswork rather than logical deduction. To maintain the integrity of the solver and ensure puzzles with unique solutions, a brute force Backtracking approach is employed to verify the uniqueness of the solution.

The Backtracking algorithm functions by systematically exploring potential solutions, making educated guesses at each step. If a dead end is reached, where no viable options are available for the next cell, the algorithm backtracks to the last decision point and explores an alternate path. This process continues recursively until a solution is found. Despite its inherent inefficiency, as the correct path may be among the last tried, the backtracking method guarantees eventual discovery of a solution.

A helpful analogy for understanding Backtracking is envisioning oneself navigating through a maze. Each decision represents a choice of direction, and when a chosen path leads to a dead end, one retraces steps to the last decision point and explores alternative routes. This process continues until a viable pathway out of the maze is discovered.

Consider the maze depicted below, with stars denoting decision points and red circles marking dead ends. At the initial decision point represented by the first red star, there are three potential paths: cyan, green, and purple. Let's assume we opt for the cyan path, leading to another decision point at the cyan star, where we must choose between left and right. If we select the left path and encounter a dead end, we logically backtrack to the cyan star, (the last decision point) and explore the right path. Upon discovering another dead end, we backtrack further to the initial red star, having exhausted all possibilities along the cyan pathway. At this juncture, we proceed to explore alternative paths along the green or purple pathways, thus illustrating the logic behind the backtracking process.

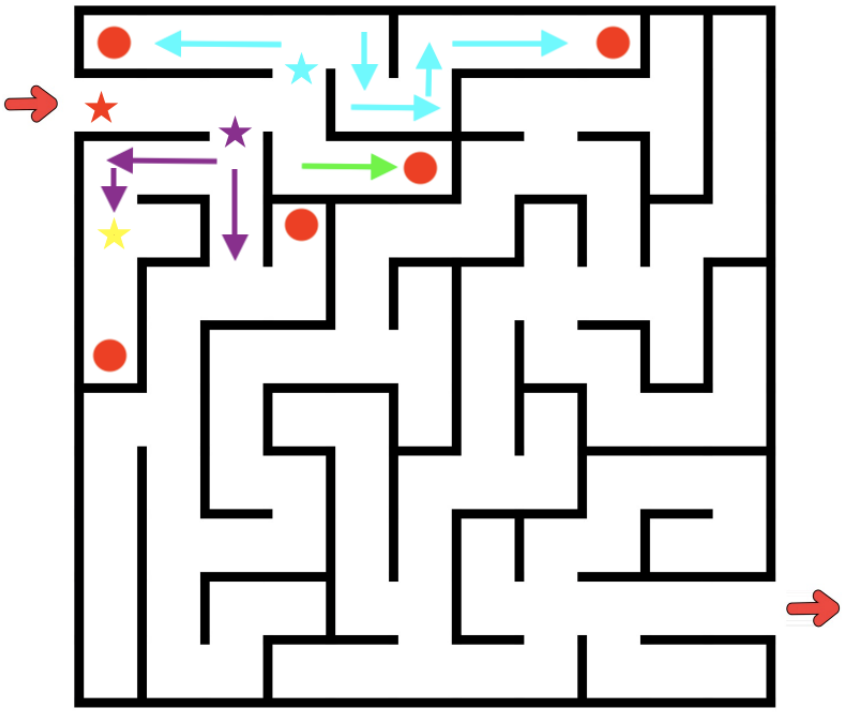


Figure 5: Maze, edited with pathways, blank maze acquired from: [10].

To confirm the uniqueness of a solution, the Backtracking algorithm is executed both forwards (1-9) and backwards (9-1), akin to traversing each pathway from left to right and then from right to left in the maze analogy. By doing so, the outputs of both forward and backward runs are compared. In the context of the maze with a single solution, traversing from left to right or from right to left would ultimately lead to the same solution. Similarly, in Sudoku puzzles, both forward and backward executions of the Backtracker should yield identical solutions if the puzzle possesses a unique solution. This dual approach ensures robust verification of solution uniqueness.

As one can imagine, the Backtracking process can be time consuming in finding a viable solution or pathway. To expedite puzzle generation, an iteration limit was implemented within the Backtracking algorithm. With each decision made by the Backtracker, the iteration counter increments. If the Backtracker fails to find a solution within the specified number of iterations, the current puzzle is discarded, and a new one is generated. This approach ensures efficiency in puzzle creation by preventing prolonged processing times.

This process underwent further optimisation in a later stage with the integration of the solver, as discussed in the subsequent section.

Additional improvements were made to the generator's processes to maximize the likelihood

of a generated puzzle having a unique solution, in an effort to increase the speed at which such puzzles were created. One such enhancement was the addition of "Ungrowables".

"Ungrowables" comprise a list of seeds for which no growth occurs. They are exempt from the cage growth process to remain at size 1 in the completed puzzle. This ensures that they serve as initial clues on the starting grid to facilitate solving for the Backtracker and eventual solver.

Research findings from [4], proved conclusively via computational exhaustion that no regular Sudoku puzzle with fewer than 17 clues possess a unique solution. Given the similarities between regular Sudoku and its Killer variant, a similar threshold for Killer Sudoku puzzles is potentially implied. Through trial and error, 10 "Ungrowables" was identified as a suitable number to expedite the generation of puzzles with unique solutions. This integration into the generation of seed cells further optimised the puzzle generation process.

Further research is required to ascertain the precise minimum number of initial clues needed for a Killer Sudoku puzzle to possess a unique solution. Nevertheless, based on the findings presented in this report, it can be inferred that this number is less than 10.

In an earlier section of the report, it was mentioned that the generator was configured to create between 35 and 46 seed cells, which would then be developed into 35 to 46 cages. The rationale behind this range lies in the desired average cage size for the generated puzzles. Larger cages tend to pose more challenge in terms of solvability due to the increased number of potential values each cell can take, stemming from the lower constraints imposed on each cell within a larger cage.

To aid the Solver and Backtracker in efficiently finding puzzles, it's important to maintain a relatively low average cage size. However, the goal is still to generate interesting and challenging Killer Sudoku puzzles. Given that 10 seeds are fixed, resulting in 10 cages of size 1, in a puzzle with 35 cages, there are 25 cages that have the potential to grow. With each cell having an equal likelihood of growth and a total of 71 cells to fill with growable cages, this yields an average cage size of approximately 2.84 for the growable cages and a total average cage size of approximately 2.61 across the full puzzle.

If we consider the scenario of 46 seeds, resulting in 36 growable cages, the average cage size for growable cages would be approximately 1.97, leading to a total average cage size of approximately 1.85 across the full puzzle. These ranges of average cage size allow for the generation of complex puzzles while ensuring they remain solvable with relative speed.

4 Solver

Various pen and paper techniques are employed by the Solver to methodically narrow down potential values for each cell in a Killer Sudoku grid. These techniques mirror the logical deductions utilised by human solvers, adhering strictly to the rules of Killer Sudoku.

4.1 Killer Combinations

The initial step, akin to human solvers, involves creating a list of potential values for each cell. This is achieved through the "killer combinations" function. By considering all possible combinations of numbers that can fulfil a given cage value, this function generates a list of potential values for each cell. For example, if a cage with a value of 6 spans 3 cells, potential values for each cell would include 1, 2, or 3, ensuring no repetition within the same cage. This approach significantly reduces the potential values for each cell across the grid. An exhaustive list of all possible cell values for each cage size and value can be found here [6].

4.2 Simple Elimination

Subsequently, the Solver employs simple elimination techniques, guided by the core rules of Killer Sudoku. It systematically scans the grid for cells with only one potential value and proceeds to remove this value from corresponding rows, columns, boxes, and cages. This iterative process continues until no further deductions can be made, steadily refining the possibilities, and driving towards the eventual solution.

4.3 Hidden Singles

Another algorithm employed by the Solver is called hidden singles. This algorithm iterates through numbers 1 to 9 and examines how many times each number appears within the lists of potential values for each row, column, and box. If a number is found to appear only once within any of these contexts, then the cell it corresponds to must take on that value. This deduction is based on the principle that in Killer Sudoku, each number from 1 to 9 must appear exactly once in every row, column, and box.

Below is an example demonstrating the efficacy of hidden singles. Consider the top left box of the Sudoku grid. Upon examination, it becomes apparent that the number 4 can potentially appear in cells B1, B2, B3, or A2. Employing hidden singles, one can deduce that in the A row, cell A2 stands as the sole location capable of accommodating a 4. Consequently, it follows that A2 must indeed contain the value 4, thereby eliminating 4 from all lists of potentials in column 2 and the top left box. This instance exemplifies how hidden singles prove invaluable in the process of refining lists of potentials.

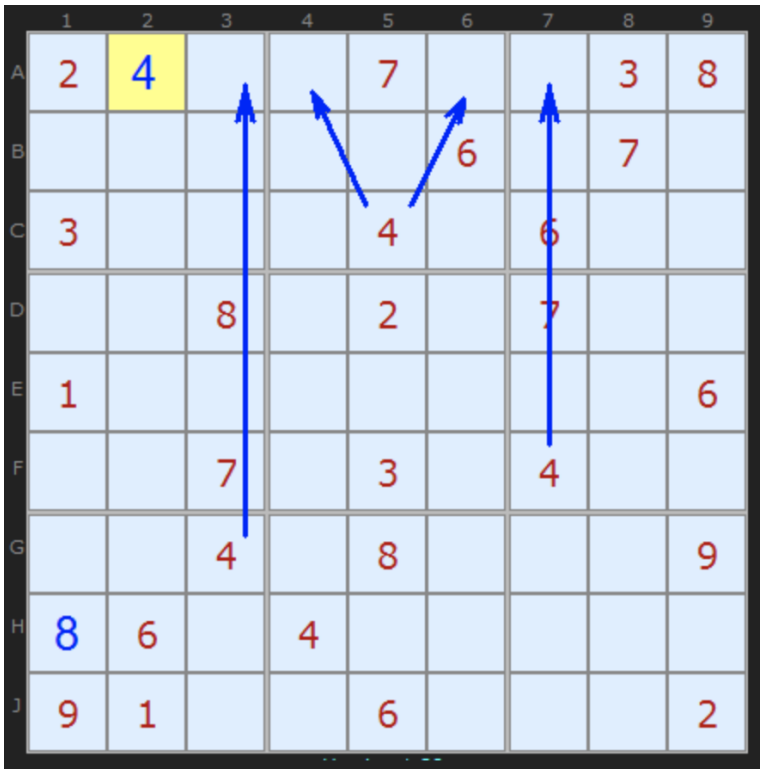


Figure 6: Hidden Singles example, image from [8].

4.4 Intersection Removal

Intersection removal is commonly split into 2 sections, Pointing Pairs/ Triples and Box Line Reduction.

4.4.1 Pointing Pairs/ Triples

Pointing Pairs/Triples is a technique employed by human solvers to refine the lists of potential candidates for each cell in Sudoku puzzles. The concept revolves around identifying situations where a number exclusively appears within the potential candidate lists of 2 or 3 cells that reside in the same box. If these cells are aligned either in a row or column, it indicates that one of these aligned cells must contain that particular number. Since each box in Sudoku must encompass every number once, the presence of this number in those aligned cells suggests its exclusion from other cells outside the box in the same row or column, depending on alignment. This deduction aids in further narrowing down potential values and advancing towards the solution.

In the example provided, attention is directed to the top right box initially. Observing this box reveals that the number 3 appears in the lists of potentials only twice and that these instances are aligned in a row, specifically cells B7 and B9. Given the constraint that each box must contain every number once, it becomes evident that the top right box must house a 3, and it will occupy either cell B7 or B9. Consequently, the number 3 cannot appear in any

other cell within the rest of the B row.

Similarly, in the bottom middle box, focus is directed to cells G4 and G5. As it is known that a 4 must occur in this box and that it will be located in either cell G4 or G5, it follows that no other cell in the G row can contain the number 4. These examples vividly illustrate the application of Pointing Pairs/Triples in narrowing down potential values within Sudoku puzzles.

Figure 7: Pointing Pairs example, image from [5].

4.4.2 Box Line Reduction

Box Line Reduction is another pen and paper technique in the solver's toolkit. It employs a similar logic to Pointing Reduction but focuses on the occurrences of numbers within rows and columns. By analysing the distribution of numbers in each row and column, if it's observed that a particular number can only occur in 2 or 3 specific cells within either a row or a column, and these cells happen to belong to the same box, it's deduced that this number must appear in one of those cells within that box. Consequently, this number can be eliminated from the lists of potential candidates for every other cell in that box.

For instance, consider the example below. Initially, column 1 is examined, revealing that the number 3 appears solely in cells G1, H1, and J1. Since every column must encompass each number once, it follows that the number 3 must appear in one of these cells. Therefore, 6 can be removed from the lists of potential candidates within the rest of the box if it occurs

elsewhere. Similarly, in column 2, it's observed that the number 6 can only appear in cells D2, E2, and F2. Consequently, it can be eliminated from the lists of potential candidates in cells D3, E3, and F3, as it's certain to already be present in that box, and each number can only occur once per box.

Figure 8: Box Line Reduction example, image from [5].

4.5 Running the Solver

The Solver function operates recursively, systematically iterating through each algorithm until no further changes can be made by any method. Initially, it creates a deep copy of the grid using the "copy" module. Then, it executes the first algorithm, Simple Elimination. If a change occurs, the Solver function is called recursively, restarting the loop from the beginning. This ensures that Simple Elimination continues to run until it can no longer make any further changes before proceeding to more complex algorithms.

After Simple Elimination, the Solver proceeds to execute Hidden Singles. Again, a deep copy is made and compared to the puzzle's state after hidden singles have been applied. If Hidden Singles yield a change, the Solver calls itself recursively, restarting with Simple Elimination.

Following this approach, the Solver avoids unnecessary execution of more complicated solving algorithms unless required. Furthermore, the Solver progresses through the algorithms in

order of complexity, a strategy observed in the Killer Sudoku solver on [7].

After Hidden Singles, Pointing Reduction is executed, followed by Box Line Reduction. This sequential execution ensures thorough exploration of all logical deductions before resorting to more complex solving strategies.

4.6 Integration of the Solver into the Backtracker

As previously mentioned, the task of Unique Solution verification can be time consuming due to the Backtracker's brute force strategy. However, Pen & Paper methods rely solely on logical deductions based on the rules of Killer Sudoku and do not resort to "guesses".

Pen & Paper methods only eliminate a number from a cell's list of potentials if it is impossible for the cell to take that value without violating the rules of Killer Sudoku. Running the solver on the starting grid before initiating the Backtracker has the potential to significantly reduce the workload for the Backtracker without compromising its aim, which is to determine whether a unique solution exists.

Therefore, they can be utilised to solve the grid partially or potentially fully before the Backtracker is executed, thereby decreasing or even eliminating the volume of work left for it.

In the case of the Generator, the Solver is executed on the starting grid to find some initial numbers to populate the grid. After the Solver completes its run, any list of length one is converted to its numerical value, while lists longer than size one are reverted back to zeros.

In an ideal scenario, the Backtracker would not only utilise the lists of length one narrowed down by the Solver but also make use of the lists of potentials for each cell, trying potential values in each one. This area presents an opportunity for further development within the project and is discussed in more detail in the "Further Work" section.

5 User Interface

The user interface incorporated into the puzzle-solving system empowers users to engage seamlessly with the platform by either inputting personalised puzzles or generating challenges for individual solving endeavours. Engineered to optimise user interaction, this interface streamlines the puzzle solving process, prioritizing user experience and efficacy. Stringent validation protocols are applied to all inputs to ensure compliance with prescribed criteria, thereby enhancing accuracy and feasibility for subsequent solving attempts. Moreover, the system meticulously crafts puzzles with unique solutions, enriching user satisfaction through the presentation of diverse challenges. Noteworthy is the user-initiated solution unveiling feature, which grants users control over the timing of solution revelation, fostering a sense of autonomy within the interactive framework.

5.1 User Interface Function

The User Interface function is responsible for handling user input when they opt to input their own puzzle. It prompts the user to enter the desired number of cages for their puzzle. Subsequently, it prompts them to enter the coordinates of the first cage. Cage coordinates can be entered as "12" for [1,2], providing a user-friendly method for inputting puzzles.

After entering the coordinates for each cage, the user is prompted to assign a value to that cage. This process continues until all cages have been assigned coordinates and values.

Once the puzzle is entered, it undergoes a validity check which examines several criteria: whether all cells have been filled by cages, if the sum of all cage values equals 405, if any cells feature in multiple cages, and if the puzzle has a unique solution.

If the puzzle fails to meet any of these criteria, the user is informed of the specific criteria it fails on. Otherwise, the puzzle is displayed for the user.

At this point, the user is prompted whether they would like to see the solution. If they input "y", the solution is displayed, and the program terminates. This allows the user to attempt to solve the puzzle before seeing the solution.

```
Hello, welcome to Matthew's Killer Sudoku Solver
Would you like me to generate you a Killer Sudoku puzzle, or would you prefer to enter your own?
Press 'G' to generate a puzzle, or I to input a puzzle i
Input
How many cages are you wanting in your puzzle? 40
Please input cage coordinates as 12 to represent [1,2] where 1 is the row number and 2 is the column
number.
To add multiple coordinates to a cage simply add a space between cell coordinates.
EG:
Type 12 34 56 78, to give the cage,
[[1,2],[3,4],[5,6],[7,8]].
Once you have finished entering the coordinates for a particular cage, press enter to move onto the
next.
Please input the coordinates of cage number 1:
```

Figure 9: Message during input.

```
Cage 32 contains cells [[6, 5], [7, 5]] and has value 12
Cage 33 contains cells [[6, 6], [6, 7]] and has value 13
Cage 34 contains cells [[6, 8], [7, 8]] and has value 9
Cage 35 contains cells [[7, 0]] and has value 3
Cage 36 contains cells [[7, 1], [7, 2]] and has value 10
Cage 37 contains cells [[7, 3], [7, 4]] and has value 3
Cage 38 contains cells [[7, 6], [8, 6]] and has value 14
Cage 39 contains cells [[7, 7]] and has value 5
Cage 40 contains cells [[8, 0]] and has value 8
Cage 41 contains cells [[8, 1], [8, 2]] and has value 11
Cage 42 contains cells [[8, 3], [8, 4]] and has value 11
Cage 43 contains cells [[8, 5]] and has value 5
Cage 44 contains cells [[8, 7]] and has value 1
Cage 45 contains cells [[8, 8]] and has value 3
Grid after solver:
7 0 0 0 2 0 0 0 0
9 0 0 0 3 0 0 0 0
1 4 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 9 0 0 0 0 0 0 0
5 0 0 0 0 3 0 0 2
3 6 4 2 1 0 0 5 7
8 2 9 0 0 5 6 1 3
Would you like to see the full solution?
Enter 'Y' for the full solution, or enter 'N' to continue y
Full Solution:
7 5 3 1 2 6 9 8 4
9 8 6 7 3 4 1 2 5
1 4 2 9 5 8 7 3 6
2 7 5 6 9 1 3 4 8
4 3 1 5 8 7 2 6 9
6 9 8 3 4 2 5 7 1
5 1 7 8 6 3 4 9 2
3 6 4 2 1 9 8 5 7
8 2 9 4 7 5 6 1 3
Goodbye
```

Figure 10: Message after successful input.

5.2 The Main Menu function

The main menu function seamlessly integrates the User Interface and Generator functions into a unified delivery system. Utilising while loops effectively prevents unauthorised entries into prompted inputs and ensures continuous program operation even if incorrect keys are pressed.

Although not essential for the generation of the results analysed in this report, the Main Menu enhances the overall functionality of the code, facilitating a streamlined demonstration of the program's capabilities.

6 Results

As previously outlined, this report's objective extends beyond the creation of a program capable of generating advanced Killer Sudoku puzzles. A fundamental aspect is the evaluation of the effectiveness of employed solving strategies.

The Generator function, embedded within a method named "results generator", systematically executes the puzzle Generator. Each time a valid puzzle is generated, it is appended to a designated list, referred to as "killers." This process iterates until a total of 100 puzzles have been produced.

Moreover, enhancements were made to the unique solution function, enabling the extraction of analytics data, which is then stored in a list named "analytics list". These analytics encompass various performance metrics of the solver on each specific puzzle, including time taken for solution, percentage of the puzzle solved, and a comprehensive breakdown of the frequency of utilisation for each solving algorithm.

Of particular note is the method employed to gauge puzzle difficulty, which hinges on the examination of the puzzle's structural components, particularly the number of cages. A higher count of cages suggests a smaller average cage size, indicative of an easier puzzle, while conversely, harder puzzles tend to feature fewer cages.

The analytics conducted provide a comprehensive assessment of the solver's efficiency, shedding light on its performance across a spectrum of puzzle complexities. This multifaceted analysis not only facilitates a deeper understanding of the solver's capabilities but also offers valuable insights into the dynamics of the puzzle solving strategies employed.

6.1 Analysis

The analysis was structured to segregate elements according to puzzle difficulty levels: easy, medium, and hard, allowing for comparative assessment across different difficulty tiers. This approach aimed to discern potential disparities in the utilisation frequency of various solving strategies across different difficulty levels.

For each difficulty category, the average time taken for the Solver to complete the puzzle was computed, providing insights into the Solver's efficiency under varying difficulty levels. Additionally, the average percentage of the puzzle solved by the Solver within each difficulty category was calculated, offering a measure of Solver proficiency across different puzzle complexities.

Moreover, the average utilisation rate of each solving strategy was determined for each difficulty tier. This analysis provided valuable insights into the relative effectiveness of different solving strategies across different difficulty levels, shedding light on their respective contributions to puzzle resolution.

By structuring the analysis in this manner, the study aimed to elucidate potential correlations

between puzzle difficulty and Solver performance, thereby facilitating a nuanced understanding of the interplay between difficulty levels and solving strategies.

6.2 Average Time Taken

The average time taken by the solver until it reached a point where no further changes could be made did not yield the anticipated results. Contrary to expectations, the solver exhausted its available moves more swiftly on the easy puzzles compared to the harder ones. This outcome defies conventional assumptions, as it would be logical to expect the solver to encounter more opportunities for progress on easier puzzles, leading to a quicker depletion of available moves on harder puzzles.

Notably, it's important to emphasise that none of the generated puzzles were solved solely by the solver. Therefore, this unexpected outcome cannot be attributed to the solver swiftly completing the grid. This anomaly warrants further investigation to uncover potential underlying factors contributing to this deviation from anticipated performance.

One plausible factor worth considering is the method used to assess puzzle difficulty. Currently, difficulty is determined solely by the number of cages, disregarding the complexity arising from cage positions and shapes. In reality, the difficulty of a Killer Sudoku puzzle is more closely linked to the arrangement and contours of its cages. Cages that traverse boxes and exhibit irregular shapes tend to pose greater challenges compared to regularly shaped cages contained within a single box, even if the latter are slightly larger. This oversight in assessing difficulty could potentially explain the unexpected disparity in solver performance across different puzzle difficulties.

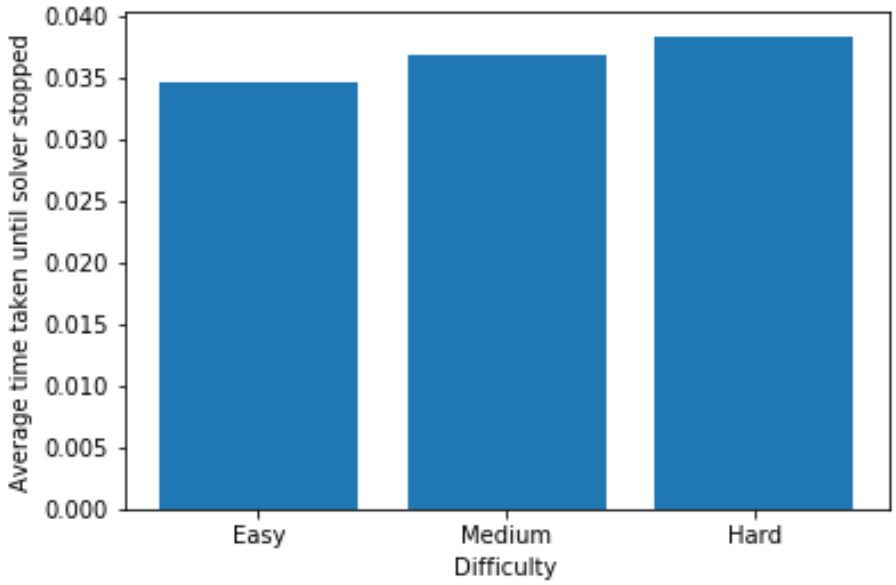


Figure 11: Graph comparing the average time taken until the Solver ran out for varying difficulties.

6.3 Average Percentage Solved

In contrast, the average percentage of the grid solved by the Solver followed the anticipated pattern, with the Solver achieving higher completion rates on easier puzzles compared to medium and hard ones, as illustrated in the accompanying graph. This observation validates the notion that while the number of cages may not be the sole determinant of difficulty, it still holds a measurable influence on puzzle complexity and solvability.

The consistent trend observed in the Solver's performance across varying difficulty levels underscores the significance of considering puzzle structure and configuration, particularly in relation to cage placement and shape, when assessing difficulty. Despite the complexity introduced by factors beyond the mere count of cages, such as irregular cage shapes and their intersections with grid boundaries, the overall correlation between cage count and puzzle difficulty remains evident in the Solver's performance metrics. This reinforces the utility of cage count as a foundational metric in gauging the complexity and solvability of Killer Sudoku puzzles.

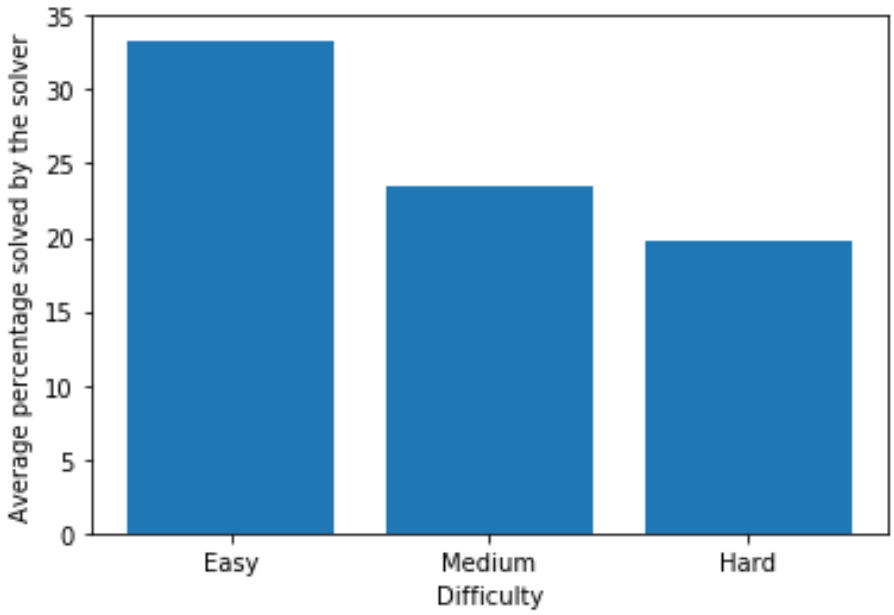
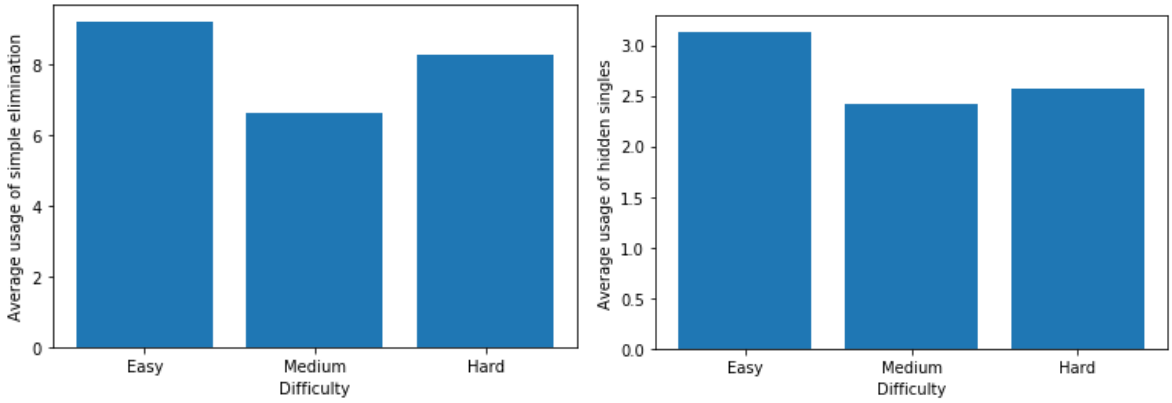


Figure 12: Graph comparing the average percentage of the grid filled by the Solver.

6.4 Average usage of various solving strategies across difficulties

The primary aim of this project was to compare the efficacy of different solving strategies across varying difficulties of Killer Sudoku puzzles. Presented below are four graphs illustrating the usage of each algorithm.

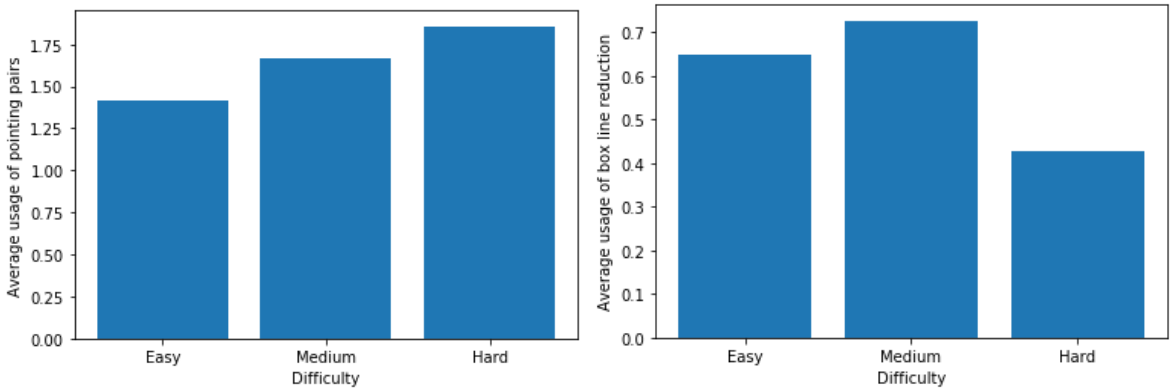
These visual representations serve as a critical component of the project's analysis, providing insights into the relative utilisation rates of different solving strategies across different puzzle complexities.



(a) Simple Elimination Usage.

(b) Hidden Singles Usage.

Figure 13: Simple Elimination & Hidden Singles Usage across difficulties.



(a) Pointing Reduction Usage.

(b) Box Line Reduction Usage.

Figure 14: Pointing Reduction & Box Line Reduction Usage across difficulties.

6.4.1 Simple Elimination - Analysis

The performance of the Simple Elimination function did not align with initial expectations. It was hypothesised that the trend would demonstrate a negative gradient moving from easy to hard difficulty puzzles. However, a notable dip in usage across medium difficulty puzzles was observed. This unexpected outcome could be attributed to several factors, including the imperfect difficulty measures mentioned earlier or the limited utilisation of multiple solving strategies within the solver. Additionally, the lack of a substantial dataset of puzzles analysed may have contributed to this deviation.

Among the 100 generated puzzles, the distribution across difficulty levels was skewed, with 60 categorized as easy, 33 as medium, and only 7 as hard. This imbalance in puzzle difficulty distribution may have led to the disproportionately high usage of the Simple Elimination function across hard puzzles. With only 7 hard puzzles in the dataset, the sample size might not

have been sufficient to accurately gauge the usage patterns of solving strategies, particularly across the more challenging puzzles. Therefore, the observed trend warrants further investigation to ascertain the underlying factors influencing the utilisation of the Simple Elimination function across different difficulty levels.

6.4.2 Hidden Singles - Analysis

Similarly, the trend observed in the usage of Hidden Singles mirrored that of the Simple Elimination function, with high utilisation for easy puzzles, slightly lower for hard puzzles, and a further decline in usage for medium difficulty puzzles. Once again, this pattern may be attributed to the aforementioned imperfections in difficulty measures, the inadequate sample size, and the limited variety of solving strategies employed.

6.4.3 Pointing Reduction - Analysis

In contrast, pointing reduction performed exactly as hypothesised. Being a more advanced solving strategy than the former, it was expected to have minimal impact on easy level puzzles but become increasingly relevant for more challenging ones. The observed positive gradient across difficulty levels aligns with this hypothesis, validating the effectiveness of pointing reduction as a sophisticated solving technique, particularly for puzzles of higher complexity.

6.4.4 Box Line Reduction - Analysis

Box Line Reduction, however, did not perform as expected, with its highest usage observed in medium difficulty puzzles, followed closely by easy puzzles, and a significant drop in usage for hard puzzles. This excessive utilisation in medium puzzles may contribute to the dips observed in usage for Simple Elimination and Hidden Singles across medium difficulty puzzles. The relatively low usage of Box Line Reduction across harder puzzles could be explained by the high usage of pointing reduction, which shares similarities with Box Line Reduction in solving approach. These unexpected results may stem from issues related to difficulty measurement, sample size, and the variety of employed solving strategies, as previously discussed. Further investigation is necessary to ascertain the underlying factors influencing the utilisation of Box Line Reduction across different difficulty levels.

7 Further Work

This section is dedicated to a critical evaluation of the project's shortcomings and proposes potential avenues for further development. Below are areas in which the project could be enhanced, in addition to other comments made throughout.

7.1 Bulking up the Solver

Broadening the scope of employed solving strategies within the Solver framework could improve performance analysis. Introducing additional advanced solving techniques may yield more nuanced results, particularly for challenging puzzles.

Moreover, a more sophisticated Solver could produce a greater number of initial results before employing the Backtracker algorithm for solving. This enhancement holds promise for significantly reducing generation time, thereby enabling the generation of a larger dataset.

7.2 Spreading out the Seed Cells

Implementing a more refined distribution of seed cells could eliminate the need for Starter Seeds and potentially serve as a pivotal factor in achieving a better balance of difficulty in puzzle generation. By ensuring a broader coverage of seed cells, it becomes feasible to reduce the number of cages while still attaining unique solutions. This approach holds the promise of generating a greater number of hard level puzzles, thereby addressing the skewed dataset currently being produced.

7.3 Refinement of Data Analysis

Implementing more sophisticated data analysis techniques could uncover hidden patterns and correlations. Utilising statistical methods to discern significant trends and relationships among solving strategies and puzzle difficulties could enhance the validity of findings.

7.4 Upgrading the Backtracker

Upgrading the Backtracker to iterate through reduced lists of potentials outputted by the Solver, rather than processing all numbers 1-9, presents a promising avenue for improvement. Leveraging non-mutable entities like lists, akin to how the iteration counter operates, could streamline this process. However, further exploration is required to ascertain the precise implementation details.

A more advanced Backtracker holds the potential to significantly accelerate generation times, facilitating the generation of larger datasets. Additionally, an enhanced Backtracker may demonstrate improved capability in solving harder puzzles, potentially leading to a reduction in the volume of cages required in a puzzle to achieve a unique solution. Paired with a more robust solver, this advancement could enable exhaustive testing of the minimum number of clues needed for Killer Sudoku puzzles.

7.5 Advanced Difficulty Measures

Enhancements in the methodology for assessing puzzle difficulty could provide more accurate categorisation. Incorporating factors beyond cage count, such as cage placement and shape intricacies, may offer a more comprehensive understanding of puzzle complexity.

7.6 Diversification and enlargement of sample data set

Expanding the dataset to include a more balanced representation of puzzle difficulties would enable a more robust analysis. A larger sample size, particularly for hard level puzzles, would provide deeper insights into solving strategy utilisation across various difficulty tiers.

7.7 Further Research into Unique Solutions

Further research into the factors contributing to a unique solution in Killer Sudoku puzzles could offer valuable insights that might be leveraged to refine the generation process, resulting in the creation of more challenging puzzles at a faster pace. As mentioned in an article by [1], there are over six sextillion Sudoku puzzles with a unique solution. However, the Killer Sudoku puzzle generator developed for this project may go through numerous iterations before finding one, hinting at a potentially lower number for Killer Sudoku puzzles. This observation underscores the need for deeper investigation into the specific characteristics that distinguish a unique solution in Killer Sudoku puzzles, which could inform the development of more efficient generation algorithms.

7.8 Community Engagement and Collaboration

Engaging with the puzzle-solving community and collaborating with experts in the field could enrich the project's insights. Leveraging collective expertise and perspectives may uncover novel approaches and foster collaborative problem-solving.

References

[1] Conroy,T. 2016. "Will We Ever Run Out of Sudoku Puzzles?". Encyclopedia Britannica website. [Online] [Accessed on 19 March 2024]. Available from https://www.britannica.com/story/will-we-ever-run-out-of-sudoku-puzzles

[2] Easybrain. 2018. Sudoku.com Website. [Online]. [Accessed 20 March 2024]. Available from: https://sudoku.com/killer

[3] Killersudoku.com. 2019. How to play Killer Sudoku. [Online]. [Accessed 16 March 2024]. Available from https://killersudoku.com/pages/how-to-play-killer-sudoku

[4] McGuire,G, Tugemann,B and Civario,G. 2014. There Is No 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration. Experimental Mathematics. **Volume** 23(2), pp.190-217.

[5] Stuart,A. 2008. SudokuWiki.org website. [Online]. [Accessed 19 March 2024]. Available from: https://www.sudokuwiki.org/Intersection_Removal

[6] Stuart,A. 2008. SudokuWiki.org website. [Online]. [Accessed 19 March 2024]. Available from: https://www.sudokuwiki.org/Killer_Combinations

[7] Stuart,A. 2009. SudokuWiki.org website. [Online]. [Accessed 19 March 2024]. Available from: https://www.sudokuwiki.org/killersudoku.aspx

[8] Stuart,A. 2012. SudokuWiki.org website. [Online]. [Accessed 19 March 2024]. Available from: https://www.sudokuwiki.org/Getting_Started

[9] Wilson,R. 2007.Sudoku. [Online]. [Accessed 22 March 2024]. Available from: https://www.britannica.com/topic/sudoku

[10] Veder,O. 2023. Vecteezy.com website. [Online]. [Accessed on 18 March 2024]. Available from: https://www.vecteezy.com/vector-art/2416178-square-maze-game-for-kids