Search Algorithm Expedition

Project Report

25 April, 2021

Valorant

203050085 | Sawan
203050083 | Sandeep
203050075 | Snehlata
170050003 | Jigar

Table of Content

Problem Statement

Searching is the universal technique of problem-solving in AI. The search algorithms help you search for a particular position in single-player games such as tile games, Sudoku, crossword, etc.

Our aim was to explore the search algorithms and compare them by implementing them in a single-player game. We learned new search algorithms from theoretical and practical perspectives through implementation. We also learned properties, features and limitations of each algorithm implemented.

We further extended our problem to create a Sudoku from scratch and then solve using different algorithms to compare results and performance. We also developed GUI to show algorithms' working which helps others understand more clearly. Our final goal was to compare the algorithms on the following metrics:

- Space Complexity − The maximum number of nodes that are stored in memory.
- Time Complexity − The maximum number of nodes that are created.
- Admissibility − A property of an algorithm to always find an optimal solution.

Software Architecture

We implemented three search algorithms and compared them based on the problem statement metrics. We have implemented all three algorithms on Sudoku as a single-player game and report a brief analysis. We have also built a GUI to visualize the working of the algorithms. Following is the list of algorithms that we have implemented.

1. Backtracking Algorithm
2. Simulated Annealing Algorithm
3. Genetic Algorithm

Libraries and Technology used:

Library used: NumPy, matplotlib, time, statistics, maths, random

Code built-in Jupyter Notebook and Spyder, python version - 3.7.4

For GUI: PyGame

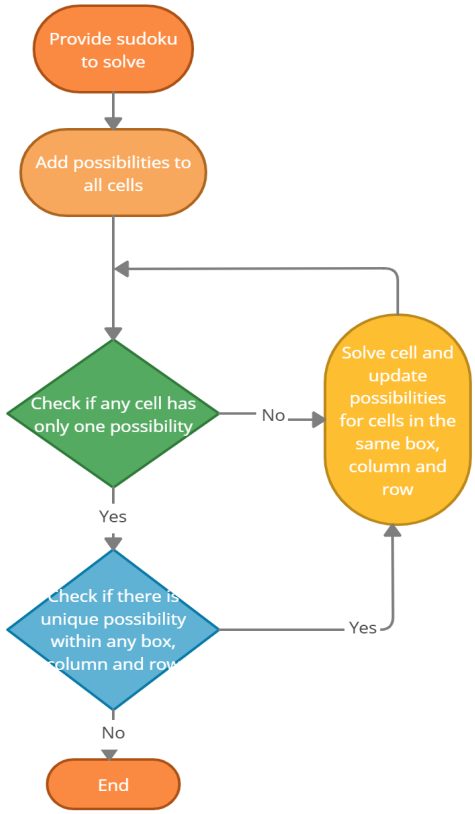
Backtracking Algorithm

A depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. The backtracking algorithm repeatedly determines an unassigned variable and then tries all values in that variable's domain, in turn, trying to find a solution.

It is a basic uninformed search algorithm that eliminates unnecessary permutations in the search tree and significantly reduces search space.

Algorithm

1. Find the empty cells.
2. If the number is safe, assign.
3. If a safe entry then recursively checks for the assignment that leads to a solution.
4. If not a safe entry, then check with a possibly different number in the exact location.
5. If none of the numbers leads to a solution, backtrack, try different values for the previous location.



Enhancements

- Recursively filled the single possible value in the cell and filled the place in order where block/row/column almost filled.
- Used Deep copy of Sudoku Board as a part of Forward Checking and Lookahead and recursively filled single values and hidden single values.
- After enhancement, we have improved our algorithm and reduced the time taken and random guess by large numbers.

Advantages & Disadvantages

- It is easy to first develop an algorithm,&then convert it into a flowchart &then into a computer program.
- Backtracking is effective for constraint satisfaction problems.
- Time Complexity: O(9^(n*n))
- Space Complexity: O(n*n)

Simulated Annealing Algorithm

Annealing is the process of heating and cooling metal to change its internal structure for modifying its physical properties. When the metal cools, its new structure is seized, and the metal retains its newly obtained properties. In the simulated annealing process, the temperature is kept variable. It is a meta-heuristic algorithm.

Initiate with candidate solution, an exploration of the search space is conducted by iteratively applying the above neighborhood operator. Given a candidate solution **s**, a neighbor **s'** is then accepted if

1. **s'** is better than **s** (concerning the cost function), or
2. With a probability: **exp(−δ/t)**

where δ is the proposed change in the cost function and **t** is a control parameter, known as the temperature. Reset the moves that meet neither of the above two conditions.

In general, the way t is altered during the run is essential to the SA algorithm's success and expense.

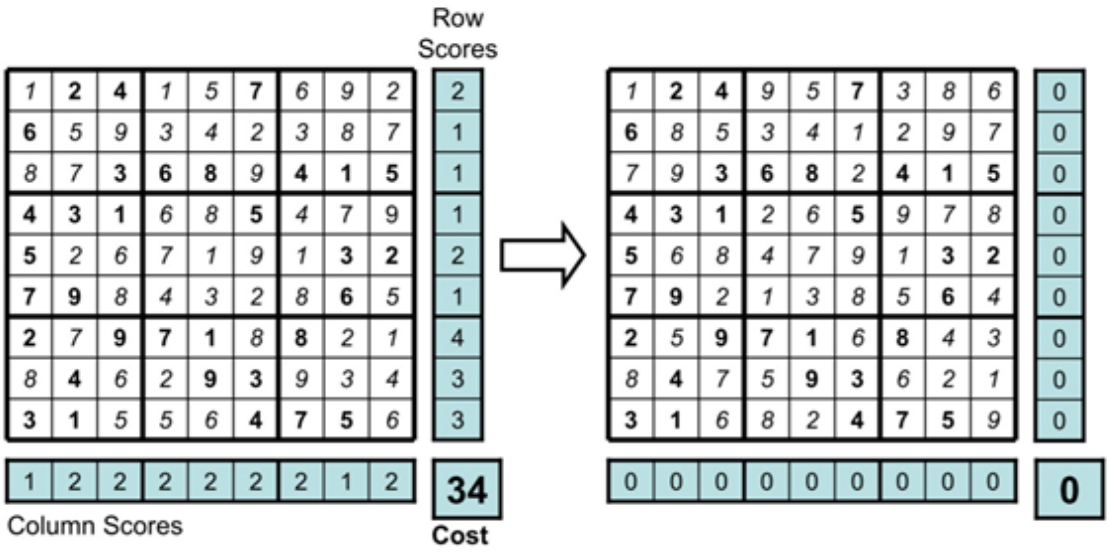
However, we should choose $\mathbf{t_0}$ carefully: a too high value will lead to a waste of computation time because the algorithm is likely to conduct an unhelpful random walk through the search space. On the other hand, a too low value for $\mathbf{t_0}$ will also negatively impact, as it will make the search too greedy from the outset, making it more susceptible to getting stuck in local minima.

One method for calculating this measures the variance in cost for a small sample of neighborhood moves. Then set the initial temperature t_0 to the standard deviation of the cost during these moves.

During the run, the temperature is reduced using a simple geometric cooling schedule where the current temperature ti is modified to a new temperature $\mathbf{t_{i+1}}$ via the formula:

$$\mathbf{t_{i+1} = \alpha \cdot t_i}$$

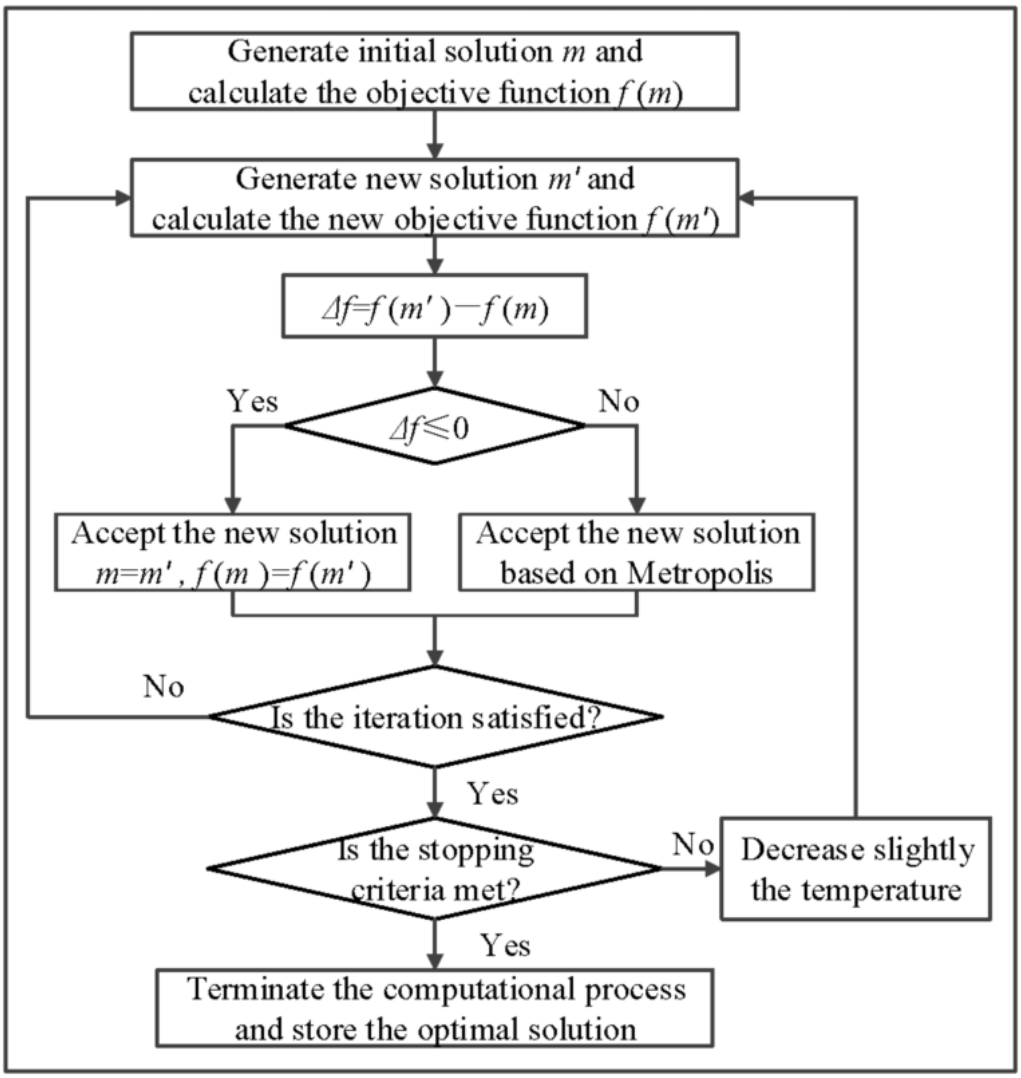
where **α** is a control parameter known as the cooling rate, and $\mathbf{0 < \alpha < 1}$. A significant value for **α**, such as 0.999, will cause the temperature to drop very slowly, while a set of, say, 0.5 will cause a much quicker cooling.



Approach

1. Fill the elements in all the boxes with possible random values within the box.
2. Calculate the cost for each row/column and total cost.
3. Pick one block randomly and swap any two randomly filled elements within the block and calculate the cost.
4. If the cost is better than the previous cost, accept it; otherwise, accept it with probability exp(−δ/t), here t is initially set to the std deviation of the cost during the move.
    a. High Temperature -> Worse state **{Tmax = 0.5}**
    b. Lower Temperature -> Better state **{Tmin = 0.05}**
    c. t is decreased in geometric progression (with a rate of **0.9**)
5. Repeat the step 3 and 4 until the cost is 0

Our objective is to minimize the sudoku board's cost using SA, where cost is defined by the duplicate element in row/column. Accept worse states more often when T is higher; otherwise, always accept better states.



Advantages & Disadvantages

- Quickly finds the minimum, and less likely to stuck in local minimum
- It is relatively easy to code, even for complex problems
- May not find Global Minimum (Best Solution)
- The method cannot tell whether it has found an optimal solution.
- Repeatedly annealing with a **1/log k** schedule is very slow, especially if the cost function is expensive to compute
- Increasing the Temperature makes the algorithm slower but less prone to get stuck in the Local Minimum.

Genetic Algorithm

Genetic algorithm (GA) is based on the fundamental concepts of natural evaluation in biology, such as mutation, selection, crossover, etc. It is a search technique used to find exact or approximate solutions to search problems.

Some terminologies used in GAs:

1. Gene is a fundamental parameter (A cell in a sudoku puzzle).
2. A chromosome or individual is a sequence of genes.
3. The population is a set of chromosomes.
4. Fitness function is the ability of an individual to compete with others in a way, The one which is more close to the solution is more fit.
5. Pair of individuals having best fitness score is selected for reproduction in the Selection Process
6. Genes of selected parents get exchanged till random crossover point is chosen in Crossover
7. The mutation is performed on recently generated individuals by transforming their values into different legal values.

Algorithm

- Start
- Generate the initial population.
- Compute the fitness of the population.
- repeat
    - Perform Selection.
    - Choose a crossover point randomly.
    - Perform Crossover.
    - Perform Mutation.
    - Compute fitness value of newly created genes.
- until the population has converged
- Stop.

Advantages

1. Easily parallelized
2. Provides a list of some reasonable solutions instead of a single solution
3. Useful for huge search space and when the large number of parameters involved

Disadvantages

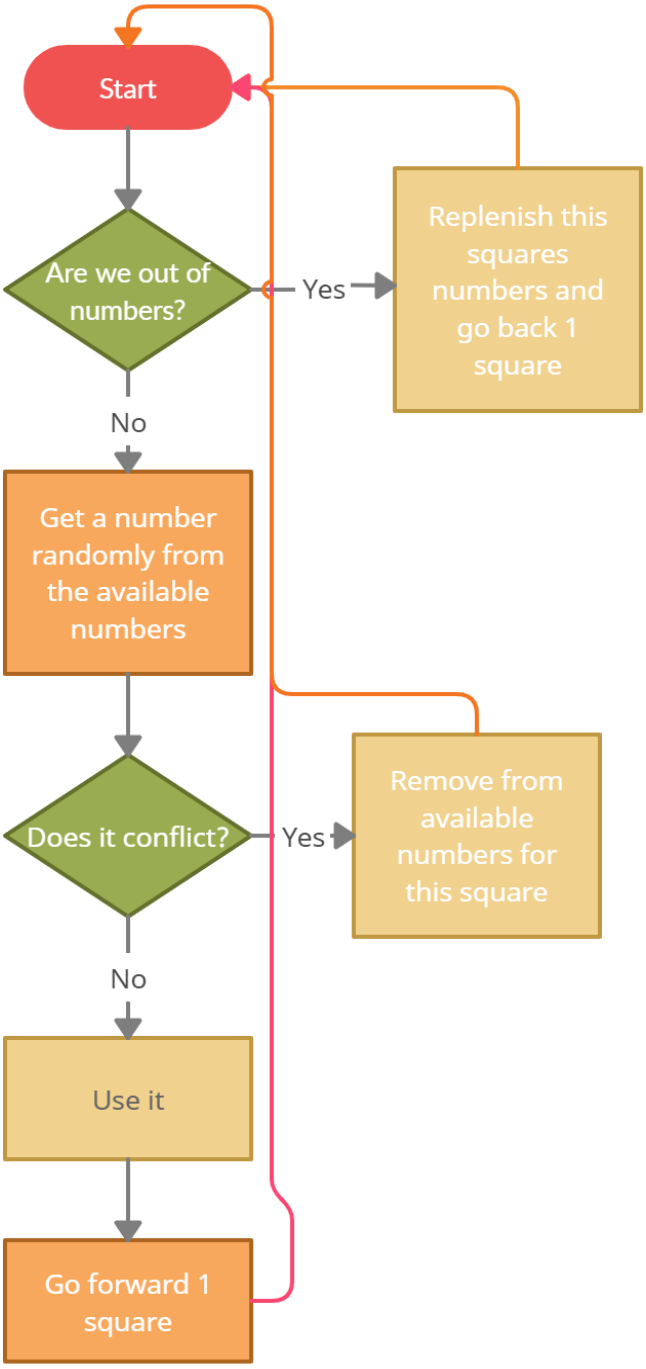
1. GAs are not suited for many simple problems and problems having derivative information.
2. There are no guarantees on the optimality of solutions.
3. In many problems, it stucks on local optima rather than the global optimum of the problem.
4. Evaluation of fitness function repeatedly is often the most prohibitive and limiting for complex problems.

Sudoku Generator

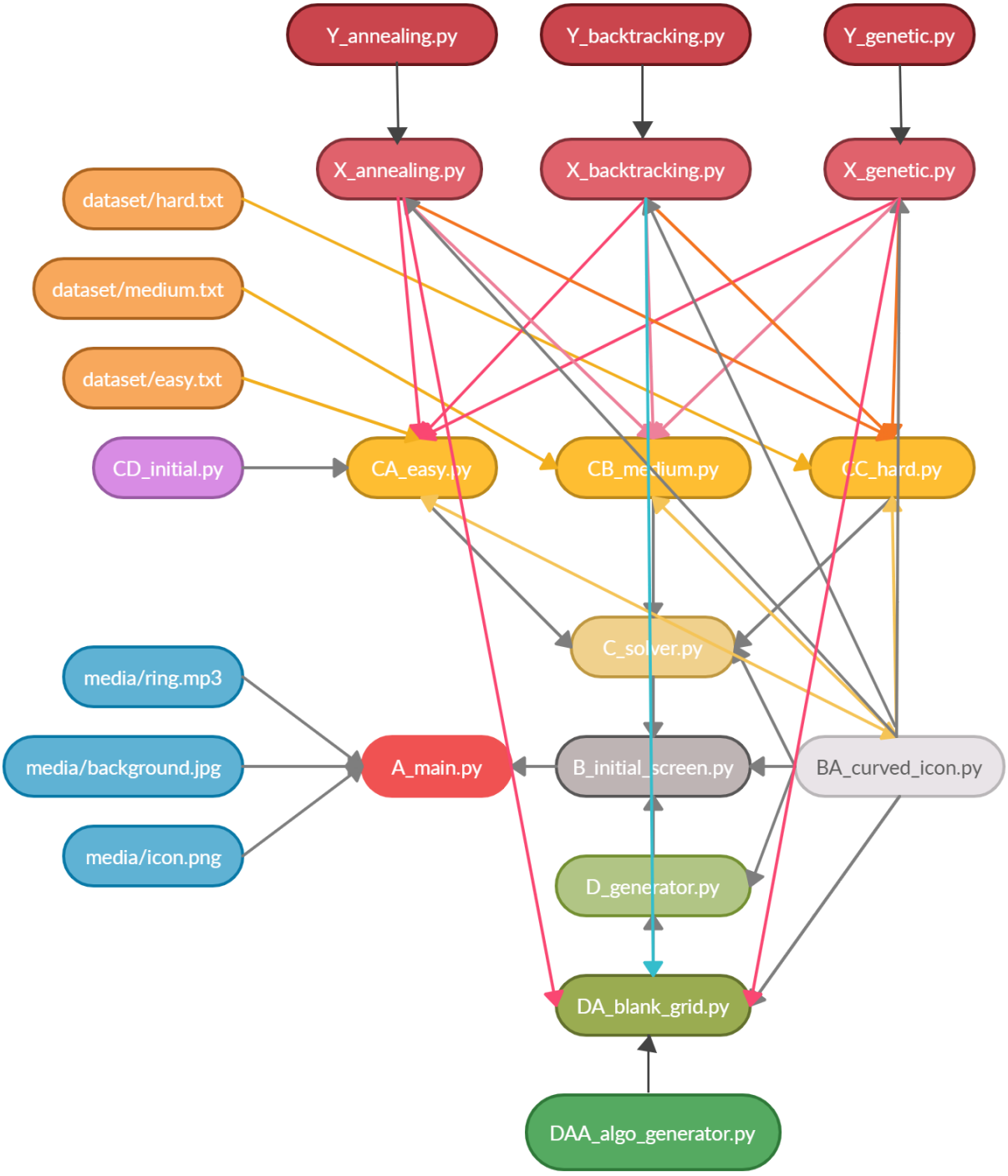
It is nearly impossible to produce a valid sudoku by randomly plotting numbers and trying to make them fit. Similarly, backtracking with a random placement method is also equally ineffective because it works best in a linear method. Backtracking is fast, reliable and effective.

Steps to generate a sudoku:

1. Generate a full grid using backtracking, keeping unique numbers in row, column and sub-grid(3*3) for the grid of 9*9.
2. We will then remove 1 value at a time from our full grid and replace it with 0.
3. Each time a value is removed, we will apply a sudoku solver algorithm to see if the grid can still be solved and to count the number of solutions it leads to.
4. If the resulting grid only has one solution we can carry on the process from step 2. If not then we will have to put the value we took away back in the grid.
5. We can repeat the same (from step 2) several times using a different value each time to try to remove additional numbers, resulting in a more difficult grid to solve. The number of attempts we will use to go through this process will have an impact on the difficulty level of the resulting grid.
    - for easy - filled cell should lie in range (36, 50)
    - for medium - filled cells in range (27, 35)
    - for hard - filled cells in range (19, 26)



Module hierarchy flowchart

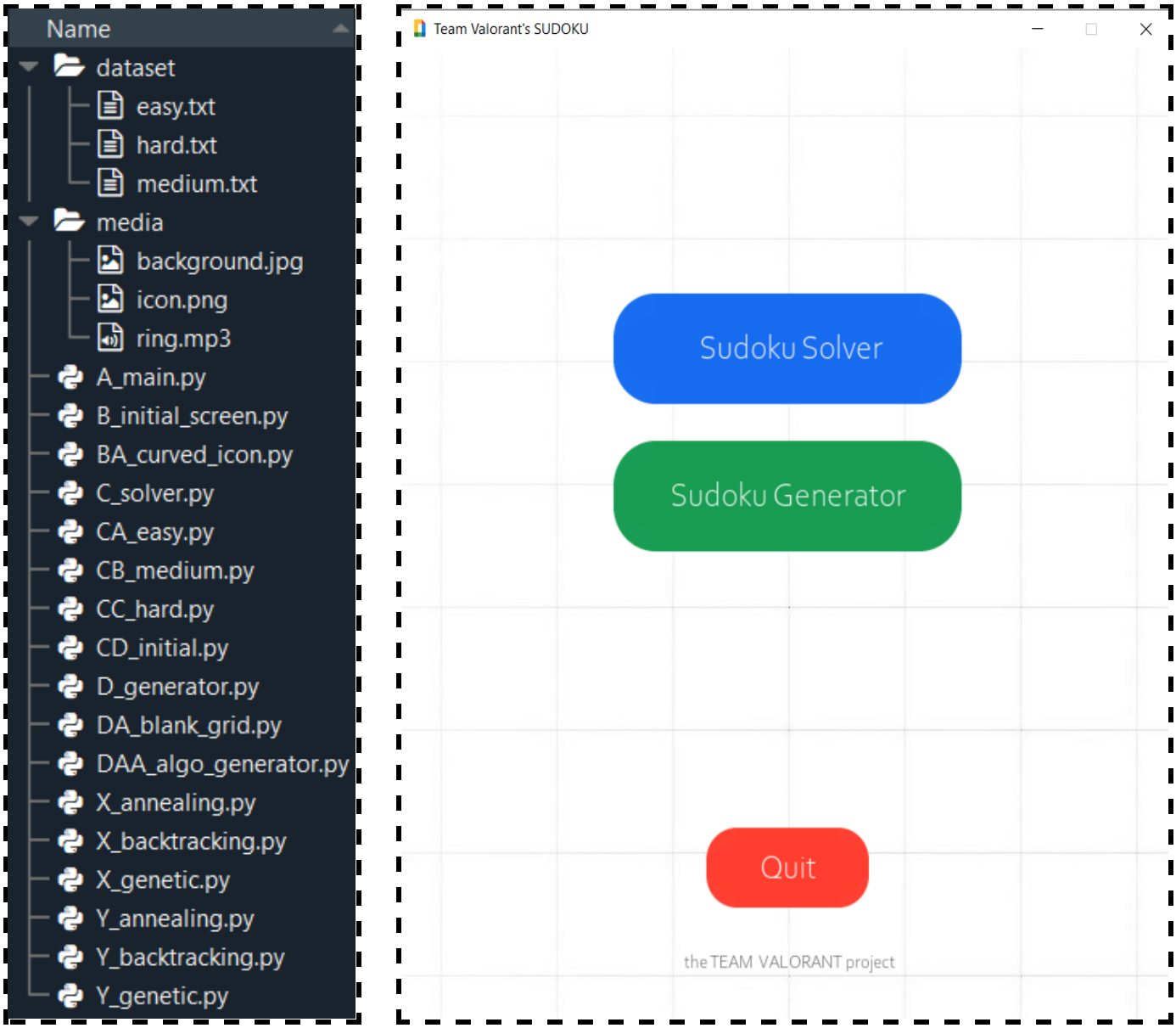


All the modules are in python files. The dotted sudoku test cases are saved in .txt files of the dataset folder as per the levels.

GUI for Algorithms

The GUI is developed using the PyGame framework of the Python language.

Initial screen

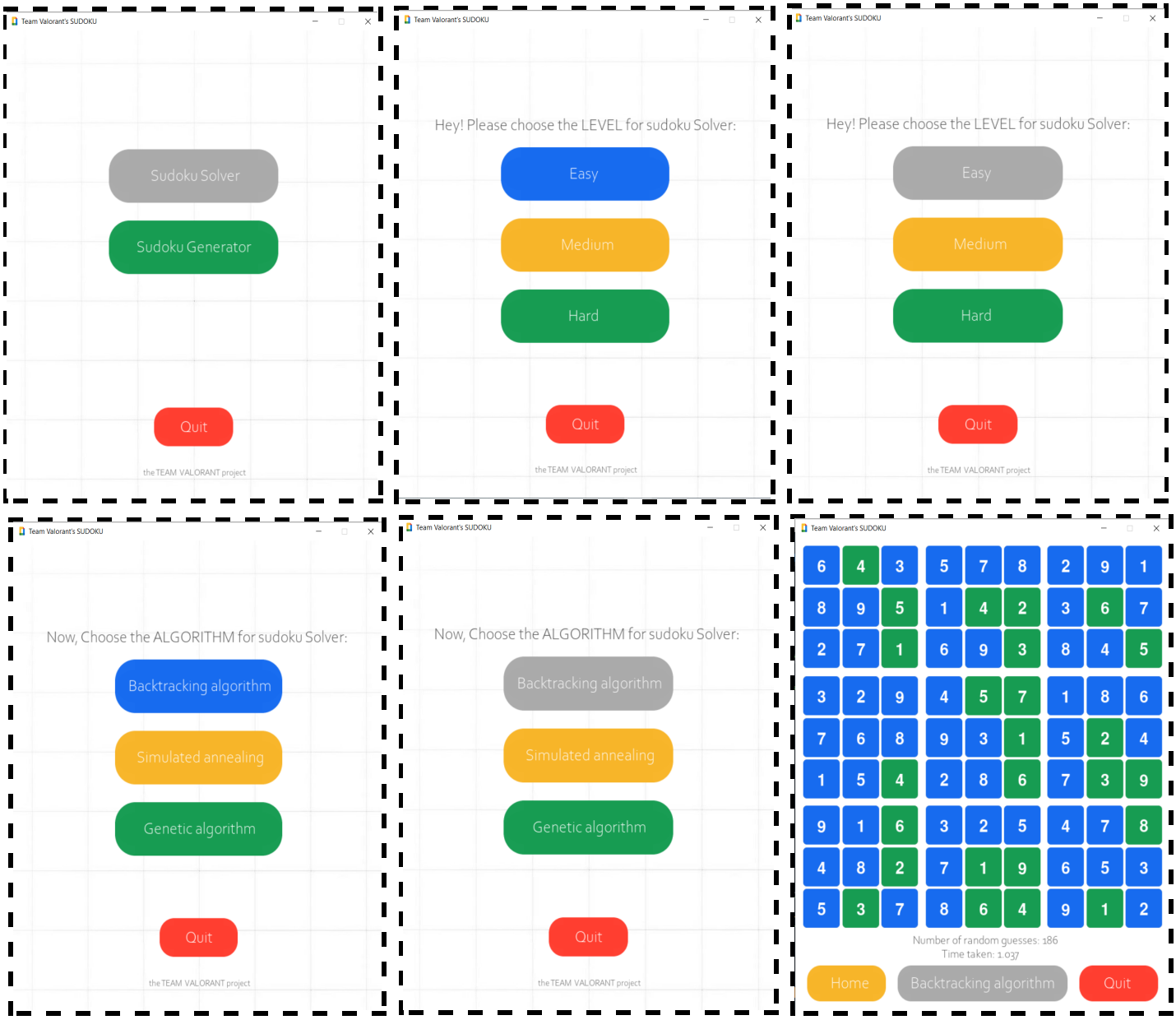


The initial screen of GUI has two main icon, which are:

1. **Sudoku Solver:** It uses the randomly extracted sudoku from the dataset.
2. **Sudoku Generator**: While it generates sudoku from scratch using a backtracking algorithm.

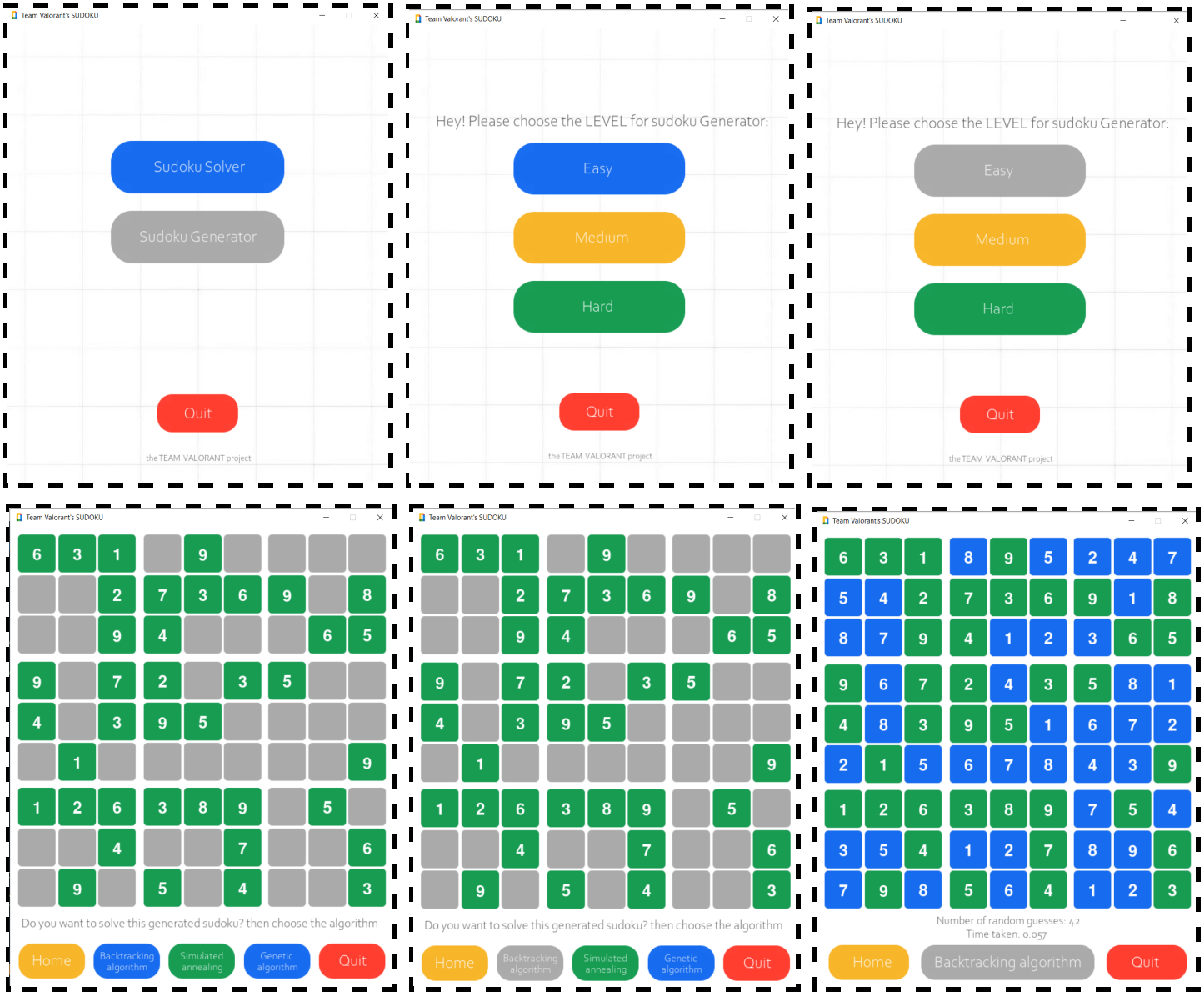
Sudoku Solver

The user will have to choose level (easy, medium, hard) and then algorithm (Backtracking algorithm, simulated annealing and genetic algorithm). As per the level chosen, it will extract a random line from the level specific .txt file from the dataset folder as **initial_matrix**, then it will apply the chosen algorithm on initial_matrix to output the final **new_matrix.** We have also displayed the time taken by generator to generate sudoku, time taken by respective algorithm and main judging factor (energy for simulated annealing, number of random guesses for backtracking algorithm and generation number for genetic algorithm).



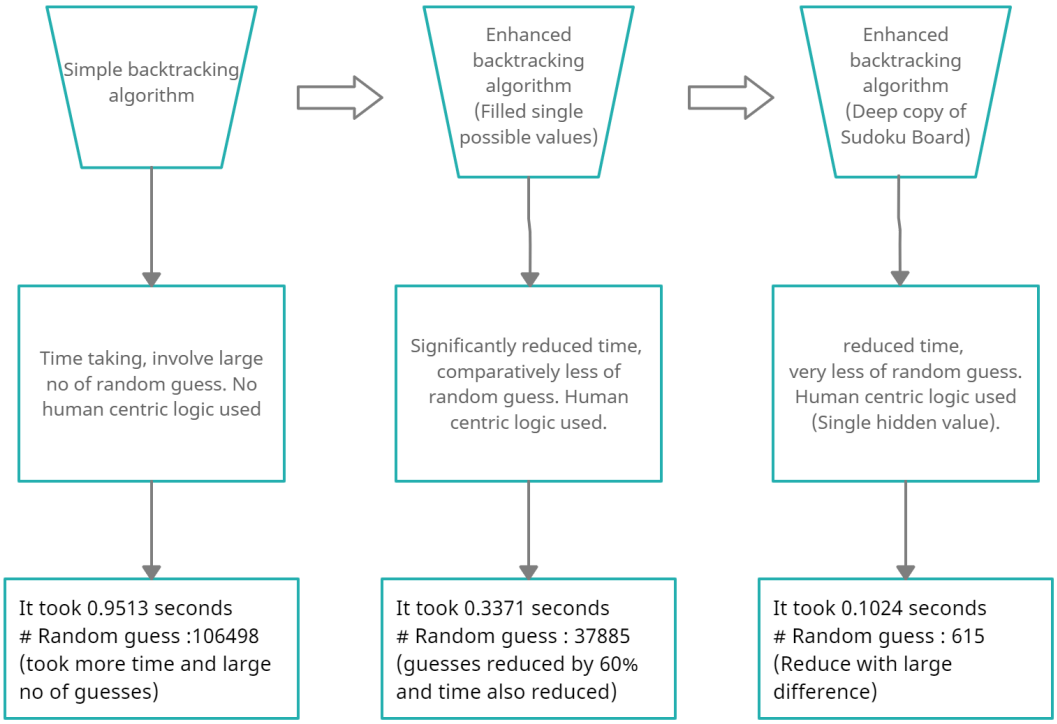
Sudoku Generator

In generator GUI, It will first ask about the level of sudoku the user wants to generate and on the basis of difficulty level, the resulting grid will be displayed. After getting the generated sudoku, the user can directly choose the algorithm (Backtracking algorithm, simulated annealing and genetic algorithm) to solve this generated sudoku. We have also displayed the time taken by generator to generate sudoku, time taken by respective algorithm and main judging factor (energy for simulated annealing, number of random guesses for backtracking algorithm and generation number for genetic algorithm).



Critical Evaluation

Backtracking



*Results may vary problem to problem. This evaluation was drawn by taking one puzzle into consideration

Finally we focus on reducing the random guess. We also introduced hidden pairs , hidden singles technique which increased time but significantly reduced the guesses from **615 to 235** and solve in just 5 backtracks and **0.785** seconds. There is a trade off between time and random guesses.

Simulated Annealing

The main challenge in this algorithm was to choose the correct value of temperature and cooling rate; we tried so many combinations and found that :

- Starting with high temperature and cooling rate close to 1 leads to slow convergence.
- Cooling rate little far from 1 (0.9) may give an optimal solution
- When the probability of choosing a neighbor becomes 0.5, it behaves like Hill Climbing.

Since it is a meta-heuristic algorithm, the results are not always the same for the same configuration. We have found the solution and noticed that it is not always giving the correct solution. We are getting the Energy cost of -160 instead of -162. We are getting an error of 2. To solve the sudoku completely we can use other algorithms to correct it. Since SA is a metaheuristic, a lot of choices are required to turn it into an actual algorithm. There is a clear tradeoff between the quality of the solutions and the

time required to compute them.

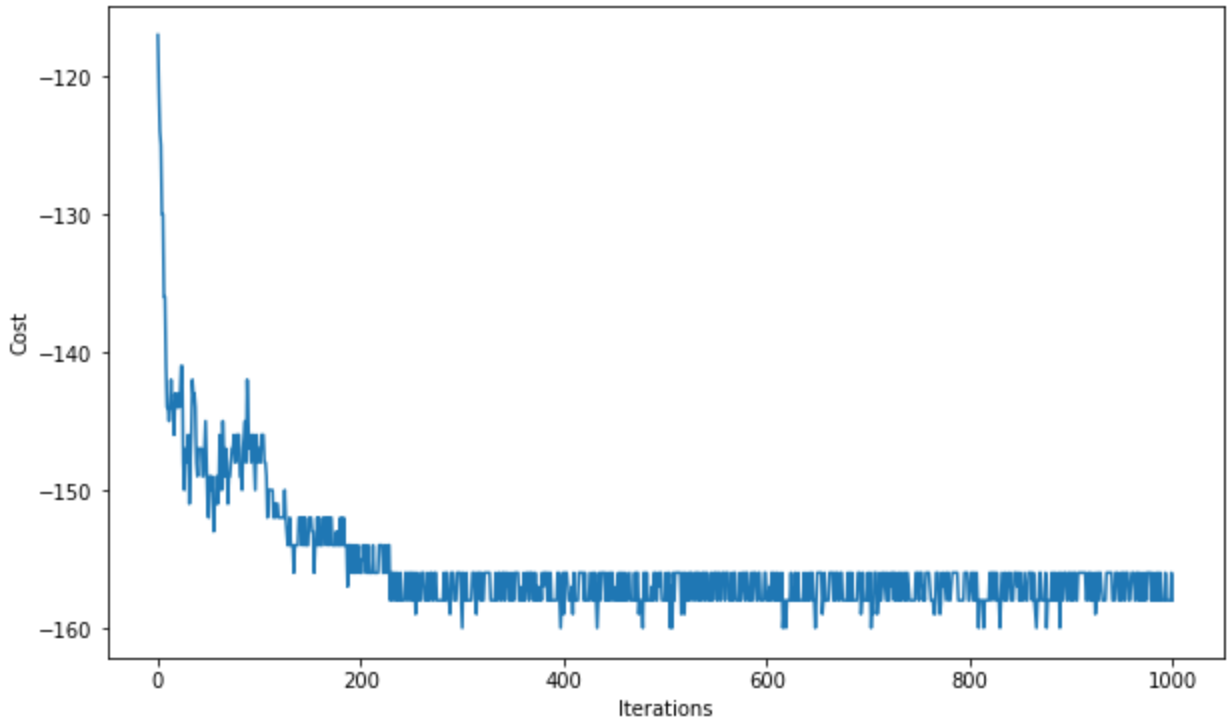


Figure shows (Cost vs. # of Iterations (scaled down by 10)) Total iterations: 10,000

We started from Maximum temperature of **0.5** and minimum temperature of **0.05**. We analyse that after **2000** iterations the solver is fluctuating to get **-162** energy cost but failed to do so. We stopped at 10000 iterations and we got an error of **2**. In future we can apply Adaptive Simulated annealing which is a variant of SA, see if the result improves.

Genetic Algorithm

- When the algorithm gets stuck, means now fittest two candidates always having the same fitness, re-seeding or restarting the algorithm often works although it might take some more re-seedings before getting the correct solution.
- We tried changing this re-seeding parameter (stalecount), i.e. after how many generations we should re-seed. We found that it gets re-seed frequently in case of too small value of stalecount, and time inefficient in case of too large value of it.
- In general, changing the parameters like population size, selection rate, and mutation rate had a profound impact upon performance, but these were problem-specific.

Solver Evaluation

	Backtracking Algorithm		Simulated Annealing		Genetic Algorithm	
	Time (in secs)	No. of Guess	Time	Energy	Time	Generation
Easy	0.0522	462	0.6359	-160	11.7155	32
Medium	0.0387	549	0.6285	-160	6.4117	17
Hard	0.0683	641	0.6065	-160	49.0934	138

Generator Evaluation

	Time to Generate Sudoku	Backtracking Algorithm		Simulated Annealing		Genetic Algorithm	
		Time	No. of Guess	Time	Energy	Time	Generation
Easy	0.0372	0.0832	307	0.6077	-160	0.5351	1
Medium	0.1804	0.0861	438	0.0657	-162	5.8739	17
Hard	0.6248	0.0903	773	0.328	-162	6.0582	17

*Results may vary problem to problem.

We can clearly see backtracking wins here. Since simulated annealing is a metaheuristic algorithm it involves a lot of iterations and takes a large amount of time to solve the problem.

Genetic algorithm is also a search optimization technique and takes a large amount of time to solve hard problems. GA is computationally expensive i.e. time-consuming.

Project Deliverables

We planned to deliver a brief report on implementation, working, performance, and comparison among these three algorithms and a GUI to analyze and visualize all three algorithms' working.

- Improvement in Simulated Annealing and draw proper GUI to show the results and analysis (temperature versus iteration, cost vs. iteration graph).
- We have solved the sudoku using these algorithms and showed the variation on the values of parameters caused by the respective algorithm.
- Develop a Sudoku generator and check whether the sudoku is good or bad based on the number of Solutions.
- Introduced a more AI oriented method like hidden pair in the Backtracking.
- Finally, the project is presented through Graphical Interface and shows all the steps to solve the sudoku and results.

Contributions

Name	Contribution	Time Spent
Sandeep Parihar	Backtracking Algorithm, Simulated Annealing, Documentation	~ 70 hrs
Snehlata Yadav	Sudoku Generator,Backtracking, GUI	~ 70 hrs
Sawan	Genetic Algorithm, Analysis,GUI	~ 65 hrs
Jigar Makwana	Genetic Algorithm, Analysis,GUI, Documentation	~ 65 hrs

* time spent does not include time to explore, reading material and team discussion.

- Sudoku Solver using Backtracking is fully implemented through scratch. No code has been copied from any other source.
- The Sudoku Solver using Simulated Annealing is implemented using the simanneal python package. The code is referenced from this blog. We have not completely copied the code. We have made significant changes and used some other approach to implement our code.
- This was helpful for the Sudoku Solver using Genetic algorithm and the code is referenced from here, but we did not copy the code, we have used some other approaches and alternatives to implement the algorithm that makes our code significantly different from the original code.

Future projects

Sudoku is an interesting Game. We can apply various algorithms to solve sudoku. Another approach to solve Sudoku is by Deep Neural network. Sudoku can be solved by multiple search algorithms and optimized using optimization algorithms. Instead of trying the traditional unconstrained continuous minimization problem, we can go with an unnatural choice of a constrained combinatorial problem. We look forward to applying those techniques and comparing the results and improvements.

References

1. https://github.com/to92me/sudoku-SimulatedAnnealing/blob/master/main.py
2. https://pypi.org/project/simanneal/
3. https://www.geeksforgeeks.org/building-and-visualizing-sudoku-game-using-pygame/
4. https://www.adrian.idv.hk/2019-01-30-simanneal/
5. Lewis, R. Metaheuristics can solve sudoku puzzles. J Heuristics 13, 387–401 (2007). https://doi.org/10.1007/s10732-007-9012-8
6. http://micsymposium.org/mics_2009_proceedings/mics2009_submission_66.pdf
7. https://github.com/ctjacobs/sudoku-genetic-algorithm
8. https://github.com/roshniRam/Sudoku-Solver
9. https://link.springer.com/content/pdf/10.1007/s10732-007-9012-8.pdf
10. Dataset from https://www.kaggle.com/radcliffe/3-million-sudoku-puzzles-with-ratings
11. https://docs.google.com/drawings/ for Flow charts and diagram.
12. Google doc for writing the report.