# Artificial Intelligence Project

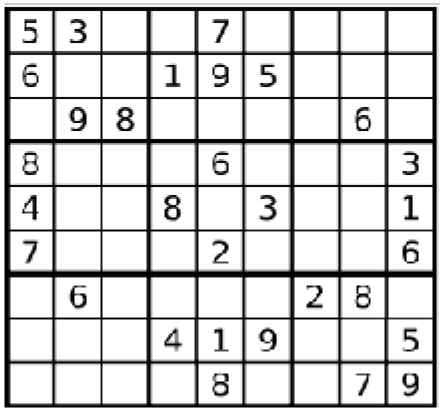
## Sudoku Solvers

**By:** Eran & Shahar

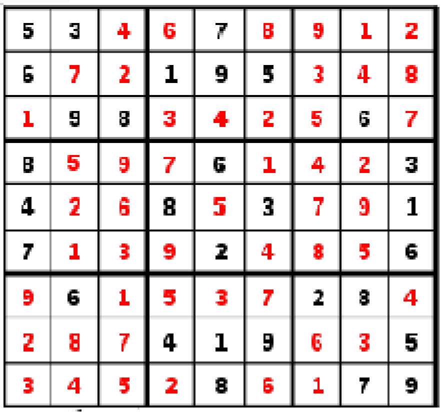
## Introduction

” Sudoku is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub-grids that compose the grid (also called "boxes", "blocks", "regions", or "sub-squares") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which typically has a unique solution.” - Wikipedia

Sudoku puzzle & solution example:



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



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

A typical Sudoku puzzle grid, with nine rows and nine columns that intersect at square spaces. Some of the spaces are pre-filled with one number each; others are blank spaces for a solver to fill with a number.

Sudoku puzzles have various difficulty levels, and while easy Sudoku can be solved relatively fast by simple logical algorithms, harder ones must be solved by “guessing” some possible solutions in order to obtain a good running time. In order for the time to solve the Sudoku not to be depended on its logical hardness, we've decided to explore stochastic methods, which simply search the space of solutions stochastically until an adequate solution is found, and not according to actual values given to it within the puzzle.

Moreover, the stochastic methods avoid local maximum solutions and continue the search in order to reach the global maximum. This quality is fundamental for the Sudoku problem since the frequency of finding a local maximum is high, when satisfying only a part of the constrains and not all of them.

In this project we've studied two different algorithms and compared their running times while working on different inputs:

1. Simulated Annealing.
2. Genetic Algorithm.

Some definitions we have used in both the algorithms:

1. Fixed cell - a cell that is given filled in the original input Sudoku puzzle.
2. Contradiction – a situation where a value appears more than once in a row, a column or a block.
3. legal value (of a cell in the Sudoku board) – is a value v which satisfies the following demands:
    a. v>=1 and v<=9
    b. v has no contradiction with values within fixed cells

# Genetic Algorithm as a Sudoku Solver

” A genetic algorithm is a search technique used in order in computing to find exact or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection and crossover” - Wikipedia.

Some definitions first:

1. Gene – an empty cell in the original Sudoku puzzle, filled with some legal value .
2. Individual/Chromosome – an object representing a potential solution of the Sudoku problem. An individual actually consists of a sequence of genes, each of which has a value within it.
3. Population – a set of individuals.
4. Fitness function – a function from the space of individuals to R which represents the number of contradictions

## Our implementation of the genetic algorithm:

In our implementation, each individual represents a solution of the Sudoku puzzle, where each gene represents a cell which isn't fixed.

First we've created a random population of Sudoku solutions– where each individual consists only with randomly selected legal values - the value is within legal range and has no contradiction with fixed cells.

Once created, the fitness value of each member of the population is computed and the population is sorted in ascending order of fitness – when zero is the best value and represents a solution to the problem (since no contradictions to the constrains are found.)

While there is no individual of value zero (and therefore the problem isn't solved yet), we perform selection of half of the best-valued individuals in our population, in order for them to create the next generation of solutions.
We choose randomly which two individuals mate, and create a new generation using crossover and mutation.
Given two parents we choose one random point of crossover and create two children which are a combination of their parents' genes according to crossover point.
A mutation is also performed randomly, on a recently created individual, on a random gene, by transforming the gene's value into a different but legal value.
The act of mutation helps the algorithm in avoiding local minimum solutions.
However it isn't perfect, and as a result the algorithm should be restrained with a maximal number of generations created.
This cycle of life is being performed until a solution is found or until the algorithm has reached its limit regarding to number of generations created.

## Simulated annealing based algorithm as a Sudoku Solver:

"Simulated annealing (SA) is a generic probabilistic metaheuristic for the global optimization problem of applied mathematics, namely locating a good approximation to the global minimum of a given function in a large search space… each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability... The allowance for "uphill" moves saves the method from becoming stuck at local minima. "
– Wikipedia.

A few definitions first:

1. Approximated solution – a Sudoku board, where its non-fixed cells are filled with randomly selected legal values.
2. A neighbor solution – a solution S' which is close to the given solution S, differs from it only by one parameter ( in our implementation – only by a fitness value of one cell)
3. An improving value- a legal value which is given to a cell and improves its fitness value.
4. A downhill move - moving to an approximated solution which has a lower fitness value than the currant solution. (In our implementation – a value of a cell is changed to a different legal value, but with a worse fitness value).

### Our implementation of the simulated annealing based algorithm:

Even though a search of a solution will be discussed as a climb up
in the solutions plane, in fact a solution for a Sudoku game, in the plane which is defined by the fitness of a placement is actually 0 which is the minimum fitness, and not some maximum fitness - so the search is not really a climb but a decent.

Given a Sudoku board we create an approximated solution – S.
Once created, the fitness value of S is being calculated alongside with the fitness value of each of the cells.

While the solution's fitness isn't zero we search for the next best step, an approximated solution which has a better fitness value.
In order to achieve this goal, we try to find an improving value to the worst cell (the one with the worst fitness value) in the board, this by testing all possible legal values.
If no legal value improves the cell's fitness, it is marked as tested, and we continue to next worst-valued cell. In case we haven't been able to find an improving value for any of the non fixed cells the algorithms outputs failure.
In case that several improving values are found – the value which improves the board's fitness the most is selected.
Once an improving value has been reached, a neighbor solution S' is created – in which the improving value is placed instead of the prior value in that cell.
At this stage our algorithm selects whether to move to the neighbor state or allow a <u>downhill move</u>. This choice is being made in a probabilistic way, in order to <u>avoid local minima</u> and expand the search space. The probability of selecting a downhill move lowers as the search advances, due to the thought that as the search advances we are closer to solving the problem.
After performing the probabilistic choice and moving to some other approximated solution, the loop of trying to find a neighbor solution with better fitness value is being performed, until a solution is found.
This loop continues until a solution to the problem is found.

## Comparison between the two algorithms

### Average running time comparison over 40 solving attempts:

| **Board level** | **Genetic Algorithm** Avg. running time (In millisec.) | **Genetic Algorithm** Avg. running time (In millisec.) |
|---|---|---|
| Easy | 39 | 14 |
| Medium | 337 | 26 |
| Medium (2) | 478 | 28 |
| Hard | 18379 | 15295 |

### Average running time comparison over 80 solving attempts:

| **Board level** | **Genetic Algorithm** Avg. running time (In millisec.) | **Simulated Annealing** Avg. running time (In millisec.) |
|---|---|---|
| Easy | 15 | 7 |
| Medium | 147 | 11 |
| Medium (2) | 154 | 14 |
| Hard | 8304 | 6316 |

## Project conclusions

Both algorithms successfully solve easy, medium and hard leveled Sudoku boards, however with a noticeable difference in running time. We have noticed regardless the input (the difficulty level of the given Sudoku) the simulated annealing algorithm has a significantly better running time comparing to the genetic algorithm (especially noticeable in hard leveled Sudoku boards), a fact that is easy to see in the graph presented above.
This situation might occur as a result of a not sufficiently varied population created at each solution generation. We believe that in order to improve the algorithm's running time we must work with a more varied population at each solution generation.
Several updates to the genetic algorithm might reach this goal:

1. Creating an initial larger population.
2. Selecting mating partners according to fitness-based-probability, and not according to equally distributed probability among the best half population. This action will allow another method to avoid local minima, since it will allow choosing (with low probability) individuals with a worse fitness function in the selection process.
3. Applying several crossover points instead of only one. Due to this the new individual created will have a more varied genome, since it came from several parents (and not only two).

Some ideas to improve the running time of the simulated annealing: