Rossey Charleston
Vijaya Kumar V
I1800
Nov 24, 2013

Solving Sudoku using Simulated Annealing

For solving a Sudoku using Simulated Annealing , first we choose a Sudoku board which has some numbers pre filled as a part of the problem. We then fill the blank spaces in the board with unavailable numbers(1-9) such that all columns have distinct numbers in them. Now the columns are perfect however the rows and sub squares are not. Our goal is to make the sub square and rows perfect i.e. it should have distinct number in them.

We introduce a way to calculate the score of the board. We add -1 to score for each distinct number in each row and sub-square. In this way a perfect Sudoku board would have a score of -168. Our program goal is to achieve a score as close as possible to -168 and this is done using Simulated Annealing. In the process of Simulated Annealing we swap two randomly selected numbers from a randomly selected column . We then verify whether the swap which is to occur is a valid one i.e. the random numbers selected should not be the numbers which were present at the initial stage of the problem. If the swap is a valid one we calculate the score for the board after the swap. We use a function which gives us the probability of acceptance of a score . Based on the probability of acceptance calculated the two numbers are swapped giving rise to a new board.

The probability of acceptance(POA) is calculated using:

function= e^(currscore-prevscor)/maxcount

We mimic the SA's concept of temperature with a variable called maxcount which decreases gradually after each iterations.The maxcount is reduced based on the score obtained .If the score obtained is better than the previous score then maxcount is reduced with a large magnitude else its reduced by a small fraction.. We set the POA as 0.8 for this problem. Initial stages of iteration when maxcount is high(we used 3000 as initial value of maxcount) the probability of acceptance of a bad score is more than 0.8 . Thus even the bad swap get accepted . However in the later stages when maxcount has reduced considerably the probability of acceptance of a bad score reduces.

Finally , after 3000 iterations we obtain a board which has a score varying between -135 to -152. Thus we never achieve the best solution for the problem rather we get an optimum score closer to the best score.

Below is a sample output :

```
<terminated> SudokuBoard [Java Application] C:\Program Fil
Initial Board :

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

Final Board :

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


Final score:-142
Best Score:-168
Elapsed milliseconds: 887
```