Title: Solving Sudoku game by using Linear Programming **Date**:5/17/19

Group members: Yanxin Wang, Yaoqi Li, Qingcheng Zhang, Zhaoyue Wang

- State your method, mathematically, indicate clearly the notations and symbols and their meanings. Also try to explain why do you think your method

To solve the sudoku problem, we choose to use the Linear Programming method which treat the problem as min$f(X)$ subject to the linear constraints $AX=B$. A is the matrix construct by the constraints of sudoku which consists of row, column, box and cell. In the sudoku game, there are 9*9 cells. In each cell, we treated it as a 9*1 vector, which is o's except 1 on the position of the digit. So the shape of CELL is (81,729). For each row in this matrix, we defined a number(from 1 to 9). For example, 100000000 means 1; 010000000 means 2, and so on. As for the ROW in array A, it is a array which has 9 rows. Each row has 9 different choices of numbers(from 1 to 9) which represented by 9*9 matrix, therefore, the shape of ROW is (81, 729). Similarly, the COL uses the same way to contribute an array and the shape of COL is same as the ROW, which is (81,729). For the BOX part, we consider one box as a 3*3 box. In one cell, there are three choices:100,010,001. One row has three cells, and there a three rows in the box. CLUE is a constraint which constructed by the given clues and the size of CLUE is L*729, the value of L will base on the given clues. Finally, A=[ROW COL BOX CELL CLUE] and X is our solution which has 81*9=729 (81 numbers, one number represents by 9 bits)rows. So B is a (324+L)*1 matrix and all entries are equal to 0.

When solving sudoku problem, we use the constraint to find the feasible solution at first, but there may have different feasible solutions for one cell which satisfies one constraint. Also, the feasible set may be large enough to contain an infinite number of points, but the sudoku problem will have a unique solution which we prefer. In this case our method can get the feasible solution and we need to consider the possibility or desirability of each feasible solution to find the unique solution.

To solve the issue which we described in the previous part, we need to consider a new constraint. The new constraint makes sure we can get the higher probability solution in the whole feasible solution set. And by using the new constraint, to let the feasible set and the solution set we get in the new constraint equal, we can improve the rate of correction in finding the unique solution in sudoku problem.

In this correspondence, the elements of $\mathbf{x}$ are viewed as probabilities with $0 \leq \mathbf{x} \leq 1$, the inequality applying element wise. The element of $\mathbf{x}$ giving the probability of the i^{th} digit filling the $(j,k)^{\text{th}}$ cell in the puzzle is denoted x_{ijk}. A relaxation of the hard constraints in (2) which represents our model of $\mathbf{x}$ as probabilities is

$$\mathbf{A}\mathbf{x} = \mathbf{1},\ \mathbf{x} \geq 0. \tag{3}$$

Define the 9×9 matrices formed from x_{ijk}, when one of i, j or k is held fixed, to be slices of the cube x_{ijk}: $\mathbf{X}_{i::}$ is an i-slice, $\mathbf{X}_{:j:}$ is a j-slice, and $\mathbf{X}_{::k}$ is a k-slice. Constraints (3) may be interpreted as saying that all slices of the cube x_{ijk} along any of its dimensions are doubly stochastic matrices, i.e., their rows and columns sum to one.

Since sudoku is a NP-hard question. It's basically a question that want to find x such that Ax = 1, x is zero or one. We can do the relaxation on this problem that transfers our problems into solve Ax = 1, x>=0. This is a polyhedron feasible region. We want to move this feasible region to the optimal solution. At optimal solution, our x is an array with each entry is zero or one. So this can be viewed as a binary classification problem. And our loss function is the cross entropy that measure the loss between x with each entry between 0-1 and x with each entry zero or one. So our goal for sudoku game is to minimize the cross entropy function.

The algorithm presents as follows:

The cross entropy function is:

Algorithm INIT: Find Initial Feasible Point

1) Solve the feasibility problem:

$$\text{minimize } 0,\ \text{subject to } \mathbf{A}\hat{\mathbf{x}} = \mathbf{1},\ \hat{\mathbf{x}} \geq 0.$$

2) If $\mathbf{A}[\hat{\mathbf{x}}] = \mathbf{1}$, then stop. The puzzle is solved.

We find want to solve the minimization problem:

Minimize 0, subject to Ax = 1, x>= 0

This is to obtain the first feasible solution. [x] is the round up, [x] = 1 if 0.5<=x<=1, 0 otherwise.

If A[x] = 1, we obtain the optimal solution [x]

This is the cross entropy function:

$$H(\mathbf{x}) = -\sum_{i,j,k} x_{ijk} \log x_{ijk}. \qquad (9)$$

This is the gradient of cross entropy function.

$$\mathbf{y} = \nabla H(\check{\mathbf{x}}) = \left.\frac{\partial H(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\check{\mathbf{x}}} = -\log \check{\mathbf{x}} - \mathbf{1} \qquad (12)$$

On the second step of the algorithm. We compute the gradient with respect to x(feasible point).

Then we set a = 10, and we want to minimize ||ay + z|| with respect to z. Subject to A(x+z) = 1, x+z >0. If norm(z) not equal to 0, it means we are on the descent direction. Then we update

X ← X + Z

And we then check the stopping criterion A[x] = 1 again. If met, we return the optimal solution [x], otherwise we still iteratively do the algorithm using while loop.

Algorithm DOWN: Entropy Descent

1) Compute an ascent direction $\mathbf{y}$ at $\check{\mathbf{x}}$ using **(12)** or **(13)**.
2) Find a feasible step $\mathbf{z}$ by solving

$$\begin{aligned} \text{minimize} \quad & \mathbf{y}^T\mathbf{z} \text{ or } \|\alpha\mathbf{y} + \mathbf{z}\|, \\ \text{subject to} \quad & \mathbf{A}(\check{\mathbf{x}} + \mathbf{z}) = \mathbf{1},\ \check{\mathbf{x}} + \mathbf{z} \geq 0. \end{aligned}$$

3) Update: $\check{\mathbf{x}} \leftarrow \check{\mathbf{x}} + \mathbf{z}$.
4) If $\mathbf{A}[\check{\mathbf{x}}] = \mathbf{1}$, then stop. The puzzle is solved.

If norm(z) = 0, it means we step across the minimum. Therefore, we should adjust the direction from ascent to descent. To do this, we just simple transform our problem from min ||ay + z|| to min -||ay +z||, subject to A(x+z) = 1, (x+z) >= 0. Then we solve this problem, and we update the X ← X + Z. Then we check the stopping criterion A[x] = 1 again. If met, we return the optimal solution [x], otherwise we still iteratively do the algorithm using while loop.

We also add counter in while loop. If eventually we cannot solve this problem, says after 100 iteration we still cannot have the A[x] = 1, we break the loop, and we our algorithm cannot solve this problem.

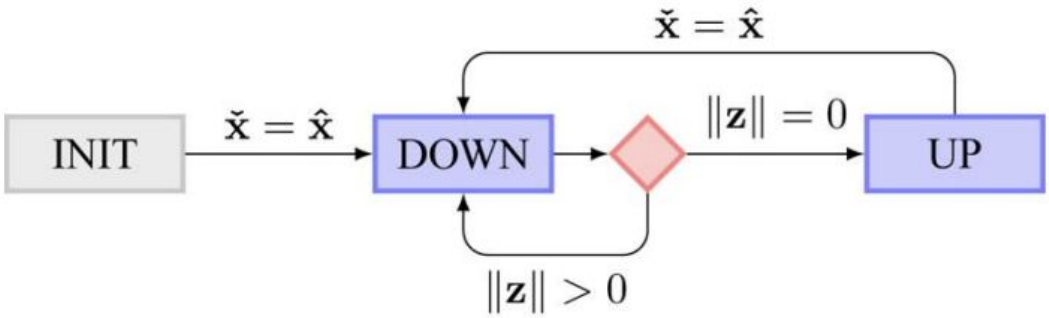
Algorithm UP: Entropy Ascent

1) Compute an ascent direction $\mathbf{y}$ at $\hat{\mathbf{x}}$ using **(12)** or **(13)**.
2) Find a feasible step $\mathbf{z}$ by solving

$$\begin{aligned} \text{minimize} \quad & -\mathbf{y}^T\mathbf{z} \text{ or } \|\alpha\mathbf{y} - \mathbf{z}\|, \\ \text{subject to} \quad & \mathbf{A}(\hat{\mathbf{x}} + \mathbf{z}) = \mathbf{1},\ \hat{\mathbf{x}} + \mathbf{z} \geq 0. \end{aligned}$$

3) Update: $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} + \mathbf{z}$.
4) If $\mathbf{A}[\hat{\mathbf{x}}] = \mathbf{1}$, then stop. The puzzle is solved.

Algorithm: Sudoku Solver



- State the success rate on data sets A, B with a table. Explain why or why not the result match your group's expectation.

For small1,large1 and large2 the success rate is approximately 100%

When p(iteration times)= 100, the success rate and average time looks like(small2):

```
Aver Time:   3.44 secs. Success rate: 11 / 20
Aver Time:   2.60 secs. Success rate: 27 / 40
Aver Time:   2.64 secs. Success rate: 40 / 60
Aver Time:   2.37 secs. Success rate: 56 / 80
Aver Time:   2.42 secs. Success rate: 69 / 100
Aver Time:   2.40 secs. Success rate: 83 / 120
Aver Time:   2.42 secs. Success rate: 96 / 140
Aver Time:   2.39 secs. Success rate: 110 / 160
Aver Time:   2.44 secs. Success rate: 122 / 180
Aver Time:   2.45 secs. Success rate: 135 / 200
Aver Time:   2.43 secs. Success rate: 149 / 220
Aver Time:   2.42 secs. Success rate: 163 / 240
Aver Time:   2.49 secs. Success rate: 174 / 260
Aver Time:   2.57 secs. Success rate: 184 / 280
Aver Time:   2.61 secs. Success rate: 195 / 300
Aver Time:   2.63 secs. Success rate: 207 / 320
Aver Time:   2.66 secs. Success rate: 218 / 340
Aver Time:   2.72 secs. Success rate: 228 / 360
Aver Time:   2.71 secs. Success rate: 241 / 380
Aver Time:   2.66 secs. Success rate: 257 / 400
Aver Time:   2.70 secs. Success rate: 267 / 420
```

When p(iteration times)= 300, the success rate and average time looks like(small2):

```
Aver Time:   9.39 secs. Success rate: 11 / 20
Aver Time:   6.92 secs. Success rate: 27 / 40
Aver Time:   7.09 secs. Success rate: 40 / 60
Aver Time:   6.38 secs. Success rate: 56 / 80
```

When p(iteration times)= 500, the success rate and average time looks like(small2):

```
Aver Time:  15.57 secs. Success rate: 11 / 20
Aver Time:  11.33 secs. Success rate: 27 / 40
```

- If there are issues or additional future work, please state that as well.

There is a trade-off between average time and success rate. Even though here the success rate seems the same, but when the population size is large enough, the success rate while improve when we increase the max iteration.

References:

https://www.math.uci.edu/~brusso/entropyminim2012.pdf