Soduku Solver

Aug 29, 2021

Fangyang Zhang, Jiaxin Li, Junyu Sui

Method description

We used deterministic optimization. Firstly, we express the constraints of Soduku as matrix form:
We use binary variable x_{ijk} to represent that whether number k is shown

- Column constraint:

$$\sum_{i=1}^{9} x_{ijk} = 1 \; for \; 1 \leq j,k \leq 9$$

Column constraint matrix will have shape 81*729 since it obviously has variables to be determined. Namely for each x_{ijk} , we use the ((i-1)*81+(j-1)*9+k)-th column to represent it. Since $1 <= j,k <= 9$, we have 81 constraints.
col = np.kron(np.ones((1,9)), np.eye(9)) will give 9 rows with 81 columns. Each row the 1 appears every 9 items, and each row has 1 starting in a different position. This corresponds to summing index i with rolling index j. Finally, np.kron(col, np.eye(N)) makes the interval from 9 to 81, since index i appears only every 81 items.

- Row constraint:

$$\sum_{j=1}^{9} x_{ijk} = 1 \; for \; 1 <= i,k <= 9$$

row = np.kron(np.ones((1,9)), np.eye(9)) will give 9 rows with 81 columns. Each row the 1 appears every 9 items, and each row has 1 starting in a

different position. This corresponds to summing index j with rolling index k.
Now we use np.kron(np.eye(N), row) to row over index i.
Note that inside np.kron, np.eye(N) is added to the left in column constraint to extend i,j patterns with k, but is added to the right in the row constraint to extend j,k patterns with i.

- Block constraint:

$$\sum_{j=3p-2}^{3p} \sum_{i=3q-2}^{3q} x_{ijk} = 1 \; for \; 1 <= k <= 9, and \; 1 <= p, q <= 3$$

we use box = np.kron(np.eye(M), np.block([np.eye(M), np.eye(M) ,np.eye(M)])) to get the sum over i distributed over k. Finally, we use np.kron(box, np.block([np.eye(N), np.eye(N) ,np.eye(N)])) to extend to j.

- Cell constraint:

$$\sum_{k=1}^{9} x_{ijk} = 1 \; for \; 1 <= i, j <= 9$$

We use np.eye(N**2) to get the i,j pattern and np.ones((1,N)) to get the sum of k (9 consecutive of 1's)

scs.csr_matrix(np.block([[ROW],[COL],[BOX],[CELL]])) is used to combine all 4 types of constraints. redundant rows will be combined automatically.

Finally, we use convex optimization solver to solve the above constraints, since linear programming has convex constraints and a convex objective function. Note the objective function we choose is just the L1 norm, since constraint satisfaction is the main part to focus on.

Result Summary

Part A (small1.csv + small2.csv)
we obtain around 33.7198% success rate on dataset A.

Part B (large1.csv + large2.csv)
we obtain around 90.9% success rate on dataset B.

Additional Work

We know that the solution may not be unique in the sudoku problem, so the method which we use in the above codes is not valid enough. Thus, our group creates a check function to find out whether the solution is valid but it is not shown in the solutions part of dataset.