IE523 B,OB,ONL: Financial Computing
Fall, 2020
Programming Assignment 3: Finding all Solutions to a Sudoku Puzzle using Integer Linear Programs
Due Date: 25 September 2020

A Sudoku Puzzle consists of a 9×9 grid, that is subdivided into nine 3×3 blocks. Each entry in the grid must be filled with a number from $\{1, 2, \ldots, 9\}$, where some entries are already filled in. The constraints are that every number should occur exactly once in each row, each column, and each 3×3 block.

The easiest (from a modeling standpoint, at least) way to solve this is to use BILP formulation. Specifically, we have $x_{i,j,k} \in \{0, 1\}$ where

$$x_{i,j,k} = \begin{cases} 1 & \text{if the } i\text{-th row, } j\text{-th column entry is } k \\ 0 & \text{otherwise.} \end{cases}$$

Technically, we do not need an objective function (but lpsolve will require you to invent something). We are looking to find a member in the feasible set defined by

$$\begin{gathered} (\forall i, j), \textstyle\sum_k x_{i,j,k} = 1 \\ (\forall i, k), \textstyle\sum_j x_{i,j,k} = 1 \\ (\forall j, k), \textstyle\sum_i x_{i,j,k} = 1 \\ (\forall k, \forall (3 \times 3) \text{ block } b), \textstyle\sum_{(i,j) \in b} x_{i,j,k} = 1 \end{gathered}$$

The first constraint says that each entry should have one number in it. The second, third and fourth constraints say that each number should occur once in each row, column, and block, respectively.

First Part: Finding a solution using lp_solve

Write a C++ program that takes the initial board position as input and presents the solved puzzle as the output. Your program is to take the name of the input file as a command-line input, print out the initial- and final-board poistions (cf. figure 1 for an illustration). The input file is formatted as follows – there are 9 rows and 9 numbers, where the number 0 represents the unfilled-value in the initial board. Figure 2 contains an illustrative sample.

You will need to make sure you have the Lpsolve API is installed on your machine. This was covered in the first week of class, and there is an handout on *Compass* that tells you how you can go about doing it. I have also uploaded sudoku hint.cpp that takes care of the nitty-gritty C++ stuff – all you have to do, is to figure out how the constraints can be inserted at the appropriate spots in the code. You do not have to use my sample code if you are well-versed/comfortable with C++.

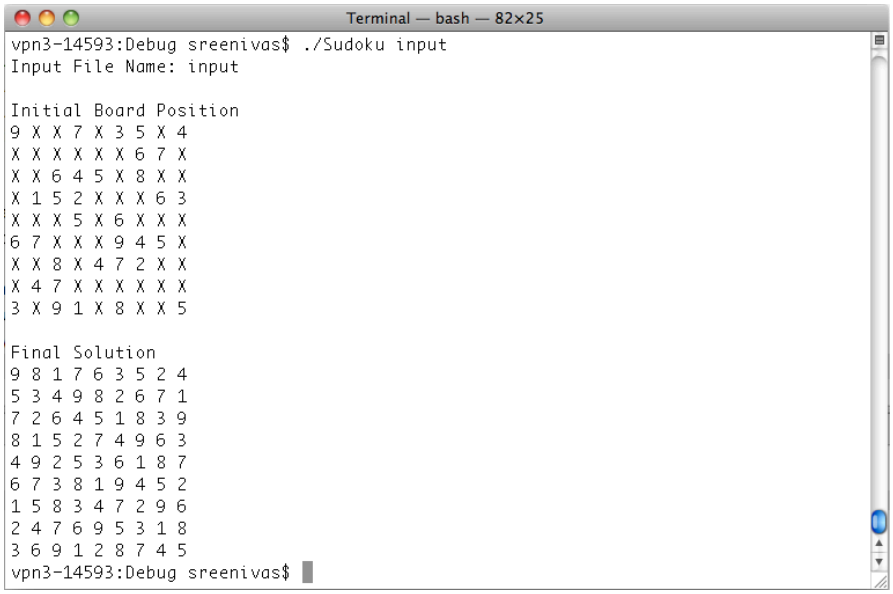

Figure 1: Sample output.

```
9 0 0 7 0 3 5 0 4
0 0 0 0 0 0 6 7 0
0 0 6 4 5 0 8 0 0
0 1 5 2 0 0 0 6 3
0 0 0 5 0 6 0 0 0
6 7 0 0 0 9 4 5 0
0 0 8 0 4 7 2 0 0
0 4 7 0 0 0 0 0 0
3 0 9 1 0 8 0 0 5
```

Figure 2: Sample input file called input, which was used in the illustrative example of figure 1. The 0's represent the incomplete board positions/values.

Second Part: Finding all solutions using lp_solve

Using the lpsolve API, write a C++ program that takes the initial board position as input and presents *all* solutions to a Sudoku puzzle. The format of the input file is exactly the same as the previous programming assignment. I am looking for an output that looks something like what is shown in figure 3.

Essentially, each time lpsolve finds a solution to the puzzle, you have to add an extra constraint that eliminates/precludes the solution. You use lpsolve to find a solution of the newly constructed ILP to find a different solution (if one exists). This process is repeated till no more new solutions can be found.

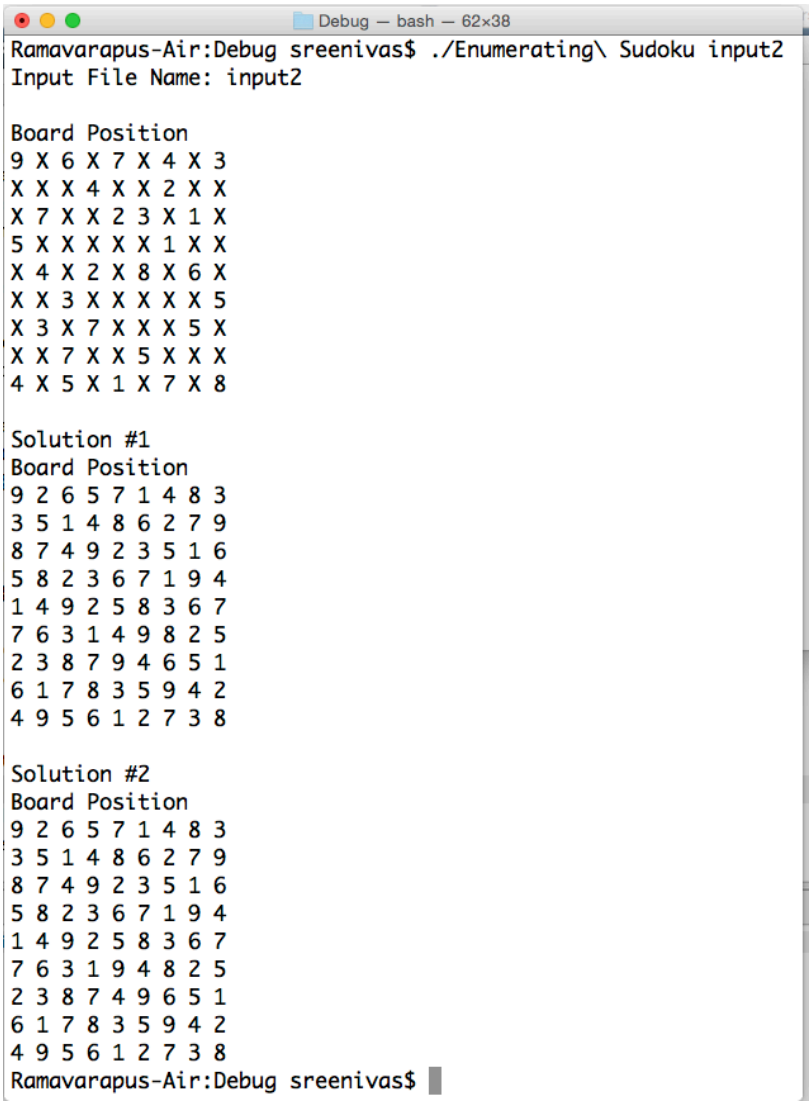

Figure 3: Sample output.