Solving Sudoku using heuristics

I wrote my code in the Programming Language Java. To run my code simply import this zip file into eclipse or put all my Java files (located in the src folder): Node.java, LinkedList.java, MyArrayList.java, and SudokuSolver.java into one directory and RUN the SudokuSolver.java file. When my code is run, it writes the solution to the Sudoku Puzzle to the console and to a text file called "output.txt."

Note: my code does not prompt the user for any input. At the top of my SudokuSolver.java is my main method, the first line in this method is

```
int [][] givenSquares={
            {1,0,3,0},
            {0,0,0,0},
            {2,0,4,0},
            {0,0,0,0}
      };
```

This 2D array represents the sudoku puzzle. The zeros in the array correspond to the squares in the puzzle that are not given, aka the spots that must be filled in to complete the puzzle. The non-zeros in the array correspond to the squares in the puzzle that are given, aka the completed puzzle must have those same numbers in the same locations.

****My algorithm finds the solution to the sudoku puzzle represented by this 2D array.

Explanation of my algorithm:

The algorithm I chose to solve the sudoku puzzle with is called "local beam search"

- ❖ The search begins with k (10 in this case) randomly generated states/ randomly generated sudoku boards
- ❖ At each iteration, successors (a successor is simply the same board, except in one row, 2 non-fixed values are switched) of all 10 boards are generated (in my

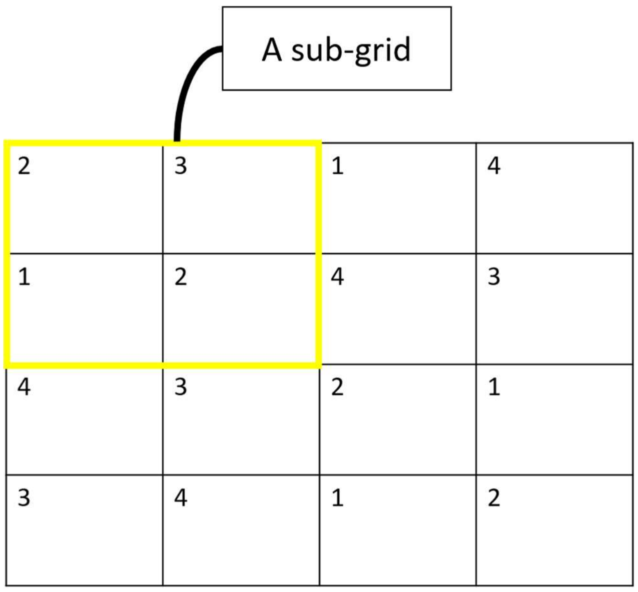
implementation each board gets n successors) n being one of the dimensions of the board, 4 in this case because the board is 4 by 4

- If any one of the successors is the desired solution, the algorithm stops

- Otherwise, it selects the 10 best boards from both the current list and successor list and repeats until the solution is found or if the algorithm gets stuck at a local minimum

- I judge how "good" a particular sudoku board is using a heuristic. The heuristic value of a board is the number of duplicates which occur in every column and every sub grid aka the inner smaller grids (in this case the 4, 2 by 2 grids. the smaller the value the "better" the board is

Example Sudoku board



2	3	1	4
1	2	4	3
4	3	2	1
3	4	1	2

- If the algorithm gets stuck at a local minimum. It starts over. (generates k (10 in this case) random sudoku boards) and performs local beam search again. This way it will eventually find the solution.

- The aim of my implementation of local beam search was so that unfruitful searches/board with lots of duplicate numbers will be abandoned and it would focus on the generating successors of sudoku boards that are "better"/do not have a lot of duplicates