CS593 Final Project Proposal

Backtracking

Team Members: Binglin Xie, Yingbin Zheng, Zeyu Ni

Project Description:

We develop highly efficient solutions based on backtracking for Magic Squares, n-queens, and sudoku. We focus on improving the efficiency of Algorithm by skipping illegal solutions. Testing and evaluation can be conducted to evaluate the performance of backtracking Algorithm.

Problem and Possible Solution:

(1) Magic Squares

A magic square is an arrangement of distinct numbers (i.e. each number is used once), usually integers, in a square grid, where the numbers in each row, and in each column, and the numbers in the main and secondary diagonals, all add up to the same number.

As the size of square become bigger, the complexity and difficulty to solve the problem become even harder. And we can find it is easier to solve the odd magic square problem, but to the even magic square problem, it need different strategy.

The possible way to solve these problem is to skip large segments of the tree if we detect that a solution does not work. For example, if sum of the first several number is over or less than a special value, we can figure out it is not possible right solution. And we can calculate the last number based on the first several numbers to avoid unnecessary loop.

The APIs of the preliminary solution for Magic Squares in Java are given as follows:

```
class MagicSquare{
class Set_MagicSquare(){};
public Set_NumPossibili(){};
public Stamp_MagicSquare(){};
public Calc_MagicSquare(){};
public SearchEmpty(){};
public CheckSum(){};
public CheckRows(){};
public CheckCols(){};
public CheckDiags(){};
}
```

(2) n-queens

The problems we want to solve:

The n-queens is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.

The challenging about the solution:

The problem can be quite computationally expensive as there are 4,426,165,368 (i.e., 64C8)

possible arrangements of eight queens on an 8×8 board, but only 92 solutions. How to skip large segments of the tree if I detect that a solution does not work.

The way to solve problems:
The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes then we backtrack and return false.

(3) Sudoku
-What problem we need to solve?
Using backtracking to solve the sudoku as quickly as possible.

-What is going to be new and challenging about it?
We may find due to sudoku rules we have no grid to fill numbers in it, we're going to backtracking at this time that cancel the previous step and choose another number to fill in the grid. Then continue to backtracking, if still not have right numbers. We need to backtracking too much because of certain constraints so that the solution space is too big, we can generate the appropriate solution may take a long time. This algorithm cannot meet the needs of the rapidly developed a complete solution.

-How you will try to solve the problems?
We are going to reduce the solution space of backtracking by modify the algorithm that divide the whole big sodoku into some small sodoku. Ruled out the restrictions between numbers so that the backtracking space greatly narrowed. Certain Numbers do not have to backtracking, for example the number "1" and "9" do not need to backtracking.

Responsibility distribution:
Zeyu Ni will be in charge of Magic Squares solution design;
Binglin Xie will be responsible for n-queens solution design;
Yingbin Zheng will take sudoku solution design;
Together: We are all engaged in the testing and evaluation for the three algorithms about Magic Squares, n-queens, and Sudoku.