CS575: Final Project Report

**Project Title: Sudoku Solver**

**Team Member(s): Aditya Parandekar, Basavaraj Beeraladinni, Rosh Thaivalappil**

**Github link :- https://github.com/adi2896/DAA-Project.git**

1. PROBLEM

A standard Sudoku contains 81 cells, in a 9×9 grid, and has 9 boxes, each box being the intersection of the first, middle, or last 3 rows, and the first, middle, or last 3 columns. Each cell may contain a number from one to nine, and each number can only occur once in each row, column, and box.

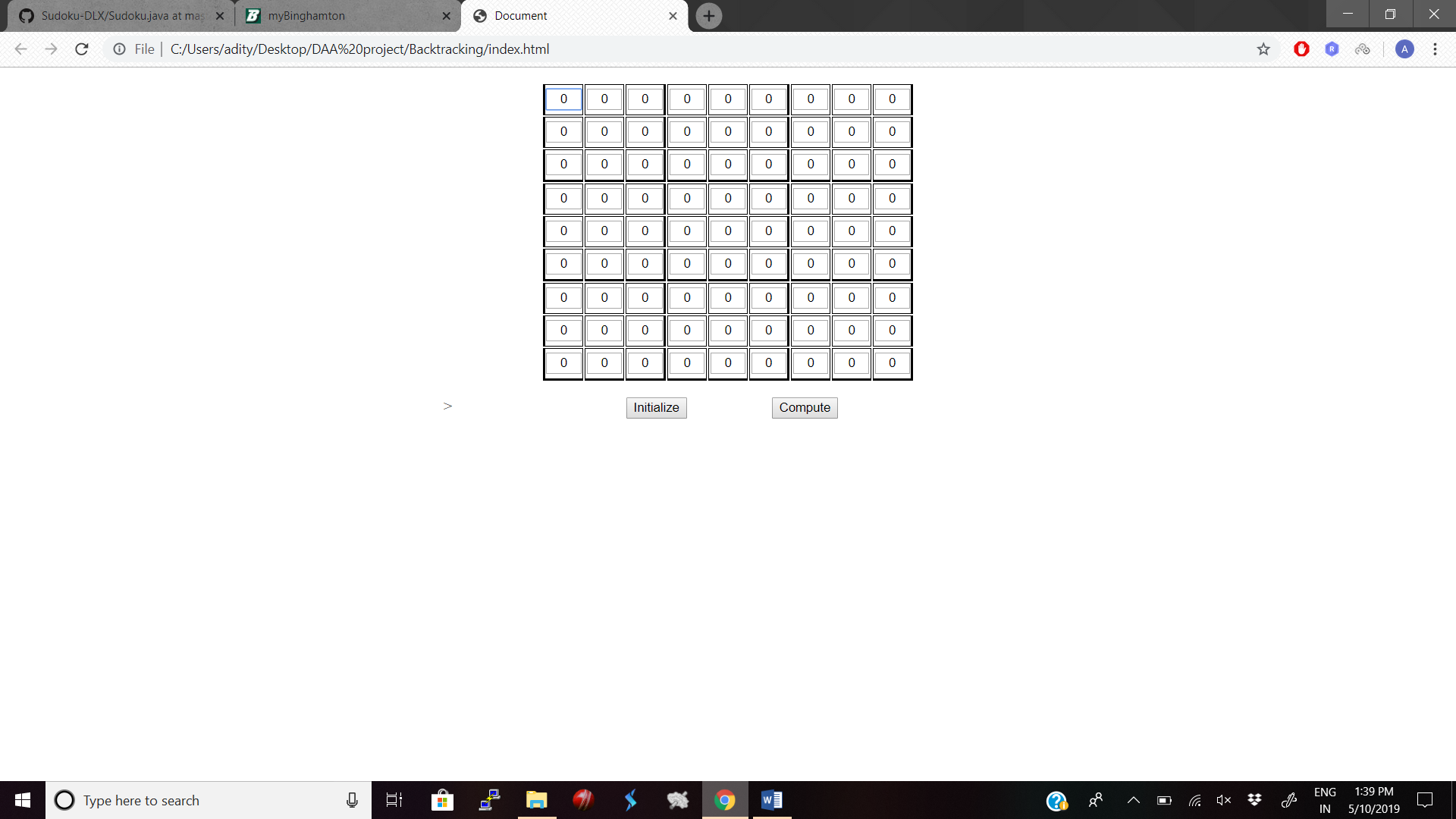
There are several computer algorithms that will solve most 9×9 puzzles (n=9) in fractions of a second, but combinatorial explosion occurs as n increases, creating limits to the properties of Sudokus that can be constructed, analyzed, and solved as n increases.

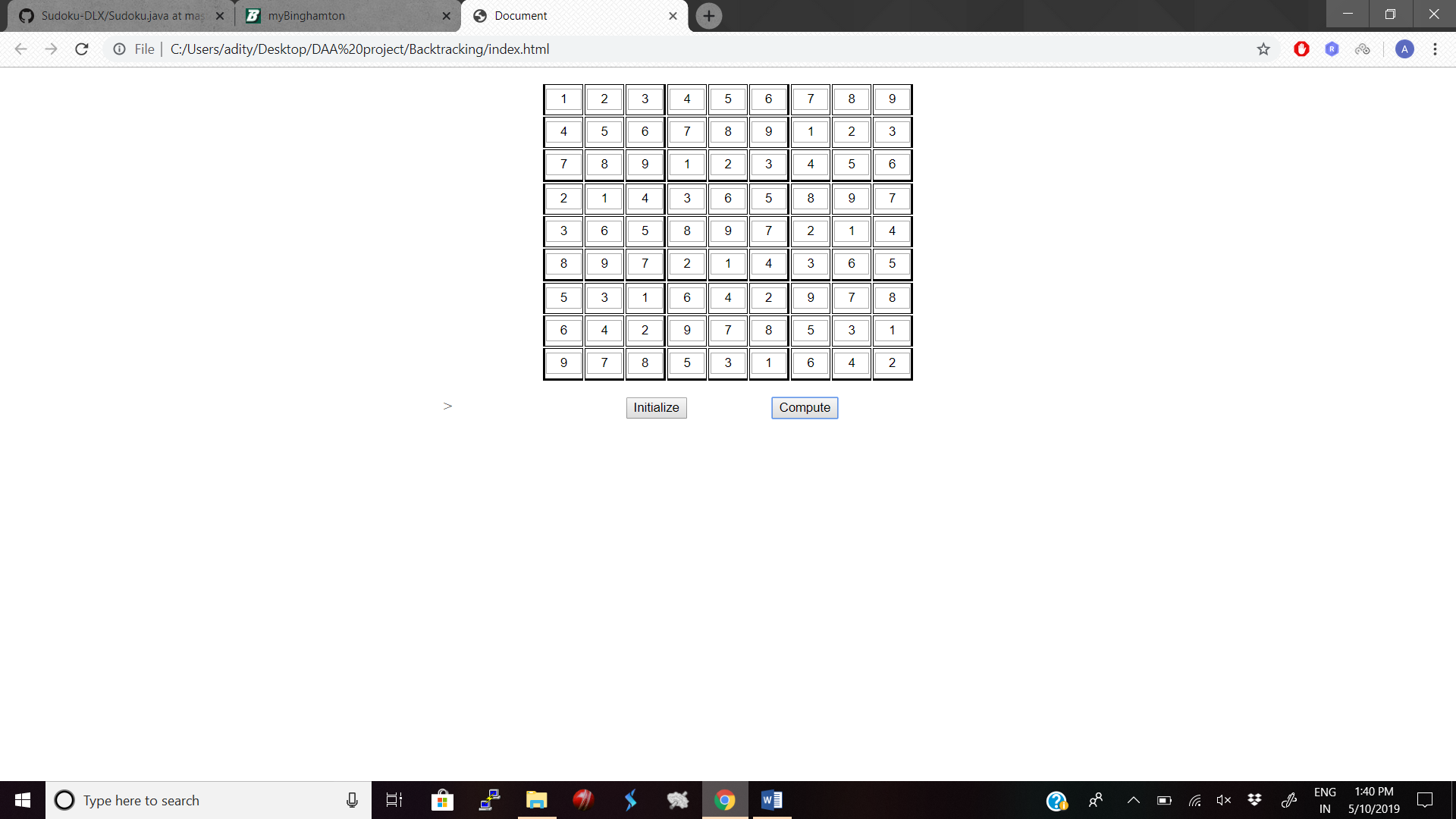
1. ALGORITHMS

*A. Backtracking*

This algorithm solves computational problems, in a incremental approach, one piece at a time. If at any point if a complete solution cannot be built with a “piece” (partial solution), it backtracks and makes a different choice.

Basically, it’s a smart way of brute forcing a problem.





*B. Constraint Satisfaction Problem*

Constraint satisfaction problems (CSPs) are mathematical questions defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods.

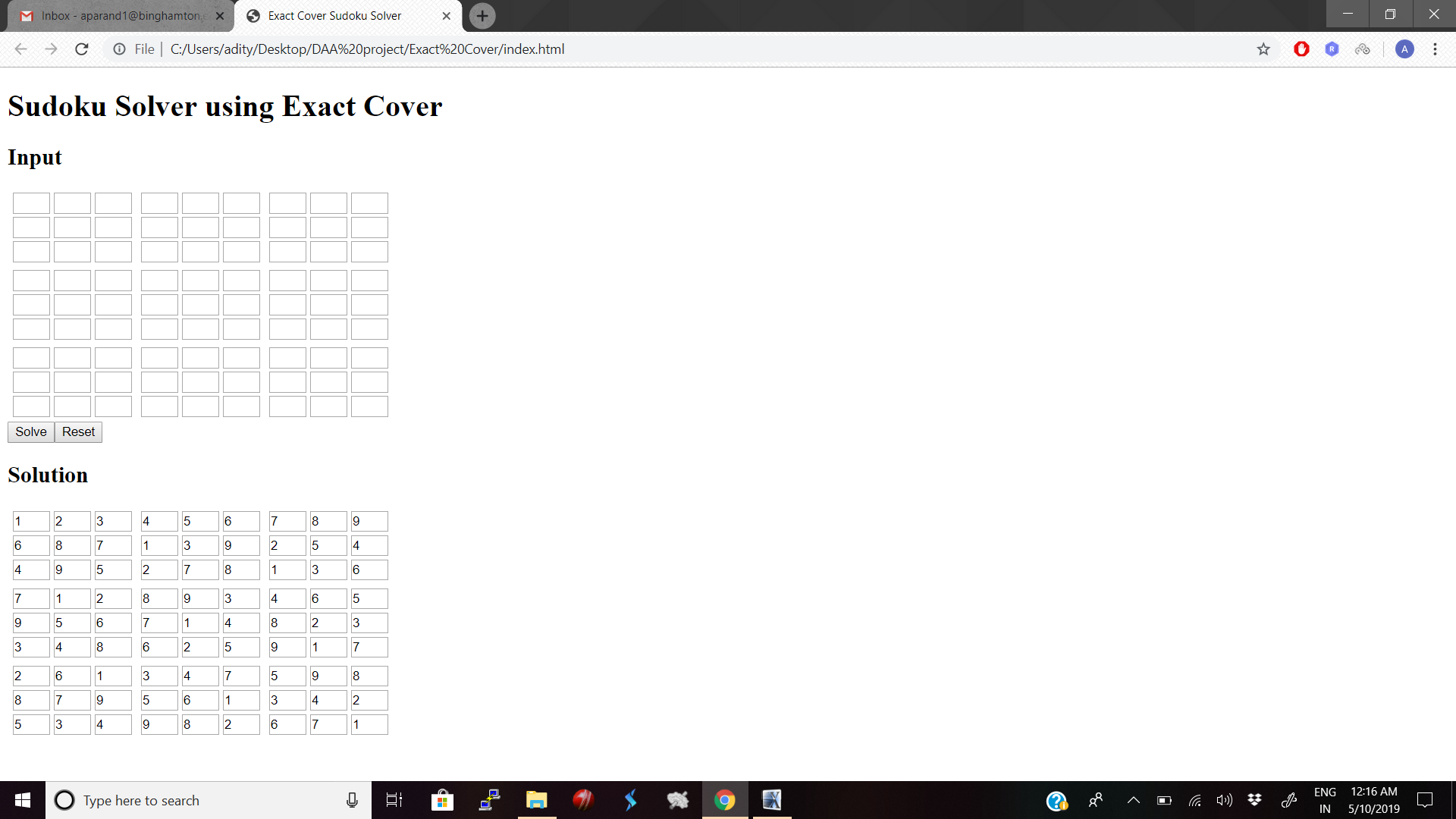
*C. Stochastic Search Algorithm*

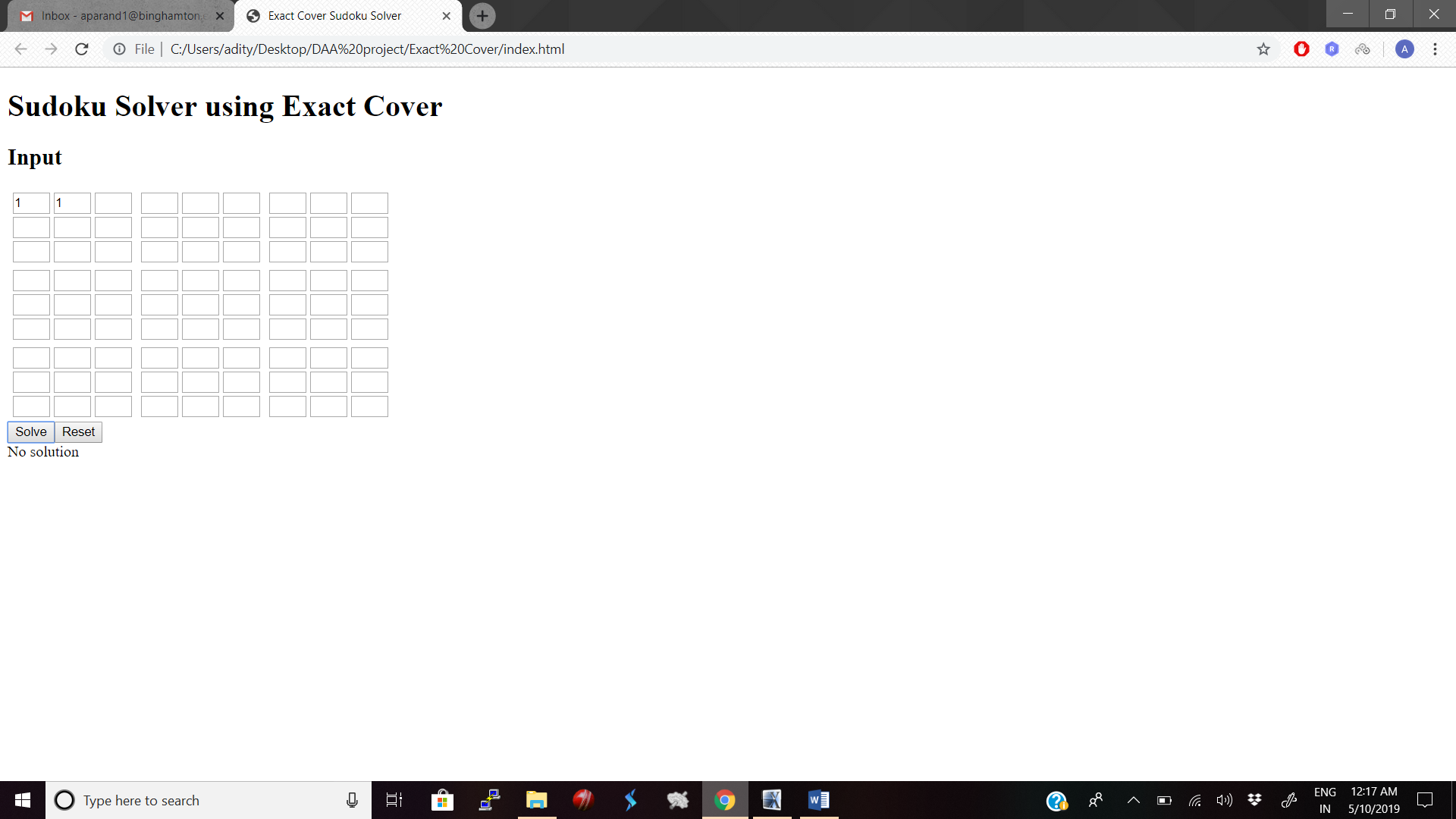
Stochastic search methods are optimization methods that generate and use random variables. For stochastic problems, the random variables appear in the formulation of the optimization problem itself, which involves random objective functions or random constraints. Stochastic optimization methods also include methods with random iterates. Some stochastic optimization methods use random iterates to solve stochastic problems, combining both meanings of stochastic optimization.

*D. Exact Cover Algorithm*

In mathematics, given a collection S of subsets of a set X, an exact cover is a sub collection S\* of S such that each element in X is contained in exactly one subset in S\*. One says that each element in X is covered by exactly one subset in S\*. In computer science, the exact cover problem is a decision problem to determine if exact cover exists.

An Exact Cover problem can be represented as a sparse matrix where the rows represent possibilities, and the columns represent constraints. Every row will have a 1 in every column (constraint) that it satisfies, and a 0 otherwise. A set of rows that together have exactly one 1 for each column can be said to be the solution set of the Exact Cover problem. Now, Dancing Links is an efficient way of solving such a problem. The idea is to take the Exact Cover matrix and put it into a toroidal circular doubly-linked list. Thus, every node in such a list will be connected to 4 other nodes and the list will be circular i.e. the last element will point to the first one. In the case of Dancing Links, for every column of the linked list, there is a special ColumnNode (which extends the normal Node) that contains identifying information about that particular column as well as the size of the column i.e. the number of nodes in it. Each Node points to four other nodes as mentioned, as well as its ColumnNode.





III. SOFTWARE DESIGN AND IMPLEMENTATION

# A. Software Design

The different algorithms listed above are used to solve a 9x9 Sudoku puzzle.

# B. Implementation and Tools Used

The algorithms are implemented in JavaScript. The tool used is Sublime Text editor. The output is visualized on the browser.

# C. Performance Evaluation

1. Backtracking Algorithm:

* Time to solve a Easy puzzle: 17.764ms  
  Number of backtracks: 215
* Time to solve a Medium puzzle: 32.55ms  
  Number of backtracks: 295
* Time to solve a Hard puzzle: 42.37ms  
  Number of backtracks: 1045

2. Exact cover Algorithm:

* Time to solve a Easy puzzle: 13.450ms

Number of backtracks: 123

* Time to solve a Medium puzzle: 22.30ms

Number of backtracks: 215

* Time to solve a Hard puzzle: 35.134ms

Number of backtracks: 780

*References:*

[1] https://en.wikipedia.org/wiki/Sudoku\_solving\_algorithms [2]http://www.ocf.berkeley.edu/~jchu/publicportal/sudoku/0011047.pdf