AI report

1. Introduction
2. Method
3. Implementation
4. Results
5. Introduction

Sudoku, in Japanese, means Number place, and it is one of the most popular number games since 2005. To solve Sudoku, one needs to use a logic-based combination and trial and error. As is shown in Figure 1 (a), a Sudoku board contains 81 squares, and some of the squares are initially filled with digits from 1 to 9. Given this partially filled 9 × 9 board, one needs to fill all empty grids. The rules is that no digit appears twice in any row, column, or 3 × 3 box. The result of this Sudoku is shown in Figure 1(b). The basic idea to solve the Sudoku is using the depth-first search (DFS) with backtracking. Although Sudoku is a completed game and it is just involved 81 squares, there are 6,670,903,752,021,072,936,960 valid Sudoku grids. Hence, for some difficult Sudoku problem, it is too slow to use the basic DFS algorithm. In this report, two algorithms are mainly discussed to solve the Sudoku problem: one is DFS with the optimization, such as constrain propagation and heuristics, the other is dancing links algorithm. In the second part, these methods will be introduced in detail. The implementation and results will be discussed in the third part.

1. Method
   1. Constrain satisfaction problem (CSP)

A Sudoku puzzle could be considered as a CSP with 81 variables and one square is one variable. Obviously, the domains of variables are the digits from 1 to 9. As is shown in Figure 2, the names of 81 variables could be A1 through A9 for the top row (left to right), down to I1 through I9 for the bottom row. A row, column, or box is called a unit. What’s more, there are 27 different constrains: 9 for row, 9 for column and 9 for 3 × 3 box (Figure 3).

* 1. Backtracking

As mentioned above, the direct method to solve the Sudoku problem is backtracking search which is a kind of DFS: one picks the first empty square and assign one and then check whether there is any conflict. If there is not, do the same procedure for the second empty square. If there is conflict, then assign two to the first square. So the basic idea of backtracking is that if there is no conflict, then choose the next variable and assign a value for it. If there is no possible value for the square, then return to previous square and re-assign another value. It is known that backtracking search is a complete algorithm and it is guaranteed to find a solution if there exist a solution, since, every possible states are taken into consideration during backtracking search. For Sudoku, there are roughly 9^(81-n) states to be search (n is the filled squares in the given board) and we could reduced the number of the states to be searched by some strategy. Constrain propagation, minimum-remaining-value and least-constraining-valueheuristic are adopted in our project.

* + 1. Constrain propagation

In the project, two kinds of constrain propagation are implemented. First, if the square M has only one value, then remove this value from the domain of other 20 squares in the same row, column and 3 × 3 box with M. Second, if a unit (one row, column or 3 × 3 box) has only one place for the value, then put the value in that place. Taken the second row as an example: in the second row, there are B1, B2,

* + 1. Minimum-remaining-value heuristic
    2. Least-constraining-value heuristic
  1. gfs
  2. gfes

1. Implementation:
2. Results: