

程式已加到我的網頁(<https://homepage.ntu.edu.tw/~twlin/>)，程式如附，茲簡介如下：

我愛玩電腦，但反應慢贏不了電腦，所以很少玩電腦遊戲。但聽說玩 Sudoku 可以預防老人失智，於是寫了一個可以玩任意階的數獨網路程式。難易可自由選擇。

目前提供最多階數為 5×5 。再高的話可能就不是人玩的了。就好像要解 4 元以上的聯立線性方程式只能求助於電腦了。

當 $R \geq 3$ 或 $C \geq 3$ 時，除 3×3 與 4×3 外，並非所有的 Seed 都能快速排出題目，故決定選用少數能於 1 秒內快速排出題目的 Seed 做為基本並與其他慢速 Seed 共用，再用 `permuteBoard()` 做宮格或行或列或數字之間的 4 種隨機重新排列即可得到幾乎無窮多的不同題目。

求解方法

一般會用遞回(Recursive)方式。為考慮效率而採用回圈(Loop)方式。但回圈層數非固定而且很大(一個空格一層回圈)：例如 4×4 階時就超過一百層回圈。

本程式採用拙作介紹的方法(詳 1.11 節)，即可僅用一個 **While-Loop**(取代許多 `goto`)內含一個動態 **For-Loop** 回圈(須用 `if` 取代)而完成任意多層回圈。如有興趣可參考 `SudokuSolver2()`。由該程式從全部空格解題可將所有格子填滿數字即得基本排列，若解題時填入空格的數字的順序為隨機(註)，就沒必要用 `permuteBoard()`。最後用 `deleteBoard()` 去除任意數量隨機選定的格子的數字即可當作題目。

求解複雜度

要確定答案是正確的，需要比對之次數與 $R \times C \times R \times C$ 成正比。這個數量是 R 或 C 的 4 次方(解 N 元聯立式為 N 的 3 次方)。

每一空格(第 i 空格)的可能填入數字之數目 $m[i]$ 雖然與 $R \times C$ 成正比，但可能都很小，但至少為 2。因為可以先用 `SudokuSolver1()` 逐步把唯一的可填數字填入(這是簡易的部分)。如果沒有空格留下，則完成解答，且為唯一解。

若剩下的空格為 N ，則組合總數為 N 個 $m[i]$ 之乘積(這是較難的部分。但若空格成對出現在二個同行或同列之宮格內時，應該很容易做人為判斷)。最後再檢查各種組合之正確性。如果每次填入前先檢查確定可填入再填入，則因考慮先前空格的填入情形， $m[i]$ 會減少很多，甚至減為 0，此時即可不必執行後繼空格的回圈，可大量減少組合總數。詳 `SudokuSolver2()`。

找到題目的時間取決於正確數字在其空格內之順位(隨機或由小到大)，愈多順位小者時間愈短，因此差異極大。以致於有很多($R \times C$ 愈大愈多)的 Seed 無法快速排出題目。

註：

前面提過解題時填入空格的數字的順序為隨機，製造題目時須從全部空白解題，則回圈有 $R \times C \times R \times C$ 層，每層的隨機順序有 $R \times C$ 個須固定並予保存。則須儲存 $R \times C \times R \times C \times R \times C$ 個整數，數量略為大些。折衷作法是只用 $R \times C \times R \times C$ 個整數，隨機產生 $R \times C$ 組 1 到 $R \times C$ 的隨機數 `pc[1:R*C][1:R*C]`。

前 $R \times C$ 個回圈用第一組的 $R \times C$ 個隨機數做為選用隨機數的組號。如第 3 個回圈，`pc[1][3]=5`，則用隨機順序 `pc[5][1:R*C]`。

次 $R \times C$ 個回圈用第二組的 $R \times C$ 個隨機數。如第 $R \times C + 4$ 個回圈，`pc[2][4]=7`，則用隨機順序 `pc[7][1:R*C]`。餘類推。如此即可保存 $R \times C \times R \times C$ 組的隨機順序。

為何大費周章做隨機排序？因為如果不用隨機順序，則只會產生一個規則變化的題目，

即使再用 `permuteBoard()` 重新排列，仍然可以看出來簡單的規則性：即宮格內的列(行)的 C(R) 個數的相異組合不多。以 3x3 為例共有 $9 \times 8 \times 7 / (3 \times 2 \times 1) = 84$ 個組合，但列只出現(1,2,3)(4,5,6)(7,8,9)(1,2,4)(1,3,5)(2,4,6)(3,5,6)這 7 種組合。而行只出現(1,4,7)(2,5,8)(3,5,9)(1,6,9)(2,3,8)(4,5,7)(1,2,8)(3,4,7)(5,6,9)這 9 種組合。這些數據可由 N=81 出一個全空白的 3x3 題目，再 `showAns` 即可得到。因解題時每個空格填入順序由小至大，取代出題時用隨機順序。

Sudoku Web Program: Sudoku09.html

The program has been added to my webpage (<https://homepage.ntu.edu.tw/~twlin/>). Below is a brief introduction:

I love playing with computers, but since my reaction speed is slow, I can't win against the computer, so I rarely play computer games. However, I heard that playing Sudoku can help prevent dementia in the elderly, so I wrote a web-based Sudoku program that allows playing at any grid size. The difficulty level is adjustable.

Currently, the maximum grid size provided is 5x5. Any larger, and it might not be suitable for human play—just like how solving systems of linear equations with more than four variables requires the help of a computer.

When $R \geq 3$ or $C \geq 3$, except for 3x3 and 4x3 grids, not all random seeds can generate puzzles quickly. Therefore, I selected a few random seeds that can generate puzzles within one second as the base cases and shared them with other slow seeds. Then, by using the `permuteBoard()` function, which randomly rearranges the regions, rows, columns, and numbers, an almost infinite number of different puzzles can be generated.

Solving Method

The typical approach uses recursion, but for efficiency, I used a loop-based method instead. However, the number of loop levels is not fixed and can be very large (one loop per empty cell). For example, in a 4x4 grid, there are already over 100 nested loops.

This program adopts a method I introduced in my book (see Section 1.11), which achieves any number of loop levels using just a single **while loop** (replacing multiple **goto** statements) that contains a dynamic **for loop** (which requires **if conditions** for handling). If interested, check out the `SudokuSolver2()` function.

The program first solves a fully blank board, filling all cells to create a base grid. If the number placement order during solving is random (note), then `permuteBoard()` is unnecessary. Finally, the `deleteBoard()` function removes digits in a selected number of random selected cells to generate a puzzle.

Computational Complexity

To ensure the solution is correct, the number of verification steps is proportional to $R \times C \times R \times C$, which is the fourth power of R or C (compared to solving N-variables linear equations, which requires a cubic complexity).

Although the number of possible values for each empty cell (`m[i]`) is proportional to $R \times C$, in practice, it is often quite small but at least 2. The program first uses `SudokuSolver1()` to fill in cells that have only one possible value (the simple part). If all cells are filled, a unique solution is found.

If N empty cells remain, the total number of possible solutions is the product of the $m[i]$ values (this is the harder part). However, if empty cells appear in matching pairs within the same row or column regions, they are easier to resolve manually.

Each time a number is placed, the program checks its validity before proceeding. Since each placement affects future possible numbers, $m[i]$ is greatly reduced, sometimes even to 0. In such cases, unnecessary loops are skipped, significantly reducing the number of combinations to check. See **SudokuSolver2()** for details.

The time required to generate a puzzle depends on the order in which the correct numbers are placed. The lower the order, the shorter the time, resulting in significant variation. This is why many seeds (especially for larger $R \times C$ values) cannot generate puzzles quickly.

Note on Randomized Number Placement

As mentioned earlier, while solving, numbers are placed in empty cells randomly. When generating puzzles, the program starts from a fully blank board and solves it. This involves $R \times C \times R \times C$ loop levels, with each level requiring a fixed and stored $R \times C$ random order. This means storing $R \times C \times R \times C \times R \times C$ integers—quite a large amount.

A compromise is to use only $R \times C \times R \times C$ integers and generate $R \times C$ sets of random numbers (1 to $R \times C$), denoted as `pc[1:R*C][1:R*C]`.

- The first $R \times C$ loops use the first set of $R \times C$ random numbers as the set selector. For example, in the third loop, if `pc[1][3] = 5`, the program uses the random sequence `pc[5][1:R*C]`.
- The next $R \times C$ loops use the second set of $R \times C$ random numbers. For example, in the $(R \times C + 4)$ th loop, if `pc[2][4] = 7`, the program uses `pc[7][1:R*C]`, and so on.

This method preserves $R \times C \times R \times C$ sets of random orders.

Why go through so much trouble for randomization? Without it, the generated puzzles would follow a simple, repetitive pattern. Even with **permuteBoard()**, the pattern remains noticeable: the unique row (or column) combinations within a region would be limited.

For example, in a 3×3 grid, there are $9 \times 8 \times 7 / (3 \times 2 \times 1) = 84$ possible unique row (or column) combinations. However, only a limited number of distinct combinations appear:

- Rows: (1,2,3), (4,5,6), (7,8,9), (1,2,4), (1,3,5), (2,4,6), (3,5,6) \rightarrow 7 combinations
- Columns: (1,4,7), (2,5,8), (3,5,9), (1,6,9), (2,3,8), (4,5,7), (1,2,8), (3,4,7), (5,6,9) \rightarrow 9 combinations

These combinations can be verified by generating a completely blank 3×3 Sudoku puzzle ($N=81$) and displaying the answer (**showAns**), since the solution follows a non-randomized, sequential order instead of a randomized placement.

This translation aims to preserve the original meaning while improving readability. Let me know if you need further refinements! 😊

40

You say :

Very good! Thank you very much!

ChatGPT say :

You're very welcome! 😊 I'm glad you liked it. Let me know if you ever need more translations or any other help. Happy coding and Sudoku solving! 🐛 🎮

40