A Statistical Analysis of Backtracking for Sudoku

DEPARTMENT OF
MATHEMATICS

Jacob Scherzer[1] Tyler Beauregard [1] Cory Shields [1]

[1]The Ohio State University

Abstract

Sudoku is a number puzzle game where you are given a 9×9 board, blocked into 9 3×3 blocks, where your goal is to fill the board, such that each block, row, and column has each number 1 through 9 appearing exactly once. Usually, some cells are filled ahead of time. Given that there is a finite number of possibilities for each cell, and a finite number of cells to fill in, this puzzle is solvable via backtracking. Since backtracking itself can be rather slow, we investigate how it can be sped up using logic.

Definitions

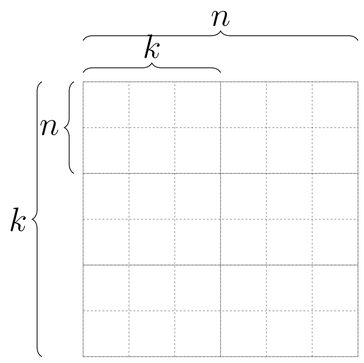
- A *fixed cell* is a cell that has been given a certain number beforehand and cannot be changed.
- An *open cell* is a cell which currently has no number in it. The number of open cells a board has is denoted with m.
- An (n, k) *board* is an $k \times n$ grid of $n \times k$ blocks, where each block, row and column must contain each integer between 1 and nk exactly once.
- *Backtracking* is a problem-solving algorithm which finds a solution by guessing different options and undoing them (backtracking) if they do not lead to a solution.
- A *naked single* is an open cell which has only one possibility which keeps the board valid.

References

[1] Frazer Jarvis Bertram Felgenhauer. Mathematics of sudoku i, 2006.

Example

A $(2, 3)$ board with all open cells



A simple backtracking algorithm

procedure BACKTRACKING($board$)
  $cell \leftarrow$ `chooseOpenCell`($board$) ▷ Returns NULL if no open cell
  if cell = `NULL` **then** ▷ The board is already filled
    return 1
  end if
  $options \leftarrow$ `getOptions`($board$, $cell$)
  $sols \leftarrow 0$
  for $option$ in $options$ **do**
    $board[cell] \leftarrow option$
    $sols \leftarrow sols +$ `backtracking`($board$)
  end for
  $board[cell] \leftarrow 0$ ▷ 0 represents an empty cell
  return $sols$
end procedure

Future Work

- Actually implement efficient Sudoku model into Code to confirm expected running time analysis via simulation
- Investigate expected running time when adding logic such as naked singles to algorithm
- Use number of solutions return to help investigate the distribution of solutions given an (n, k) board with m open cells

Results

Based upon the given algorithm, we can model this algorithm as a recursion relationship, assuming the most efficient implemtation:

$$b(k, n, m) = cnk + G(k, n, m)b(k, n, m-1)$$
$$b(k, n, 0) = c$$

where b is the time for the backtracking algorithm and G is a discrete random variable with paramaters k, n, m that represent the number of options returned by `getOptions`. Solving exactly for $\mathbb{E}[G]$ is greatly beyond the scope of this poster, as indicated by [1], which needs some brute force counting for just getting the number of solutions for an empty $(3, 3)$ board. We simplify this problem by assuming that for any open cell, its block, row, and column are completely independent of each other, as well as assuming that the probability any number appears is independent by whether other numbers appears. This leads to

$$\mathbb{E}[G] = nk \left(\frac{nk + m}{n^2k^2 + nk} \right)^3$$

where m is the number of open cells. Thus the recurrence relation for the expected running time of our backtracking is

$$b(k, n, m) = cnk + nk \left(\frac{nk + m}{n^2k^2 + nk} \right)^3 b(k, n, m-1)$$

which solves to

$$b(n, k, c_1) \in \Theta(nk)$$
$$b(n, k, pn^2k^2) \in \Theta\left(\left(\frac{1}{nk} \right)^{5pn^2k^2} \left(\frac{(nk + pn^2k^2)!}{(nk)!} \right)^3 \right)$$
$$b(c_1, c_2, m) \in \Theta\left(\left(\frac{c_1c_2}{(c_1^2c_2^2 + c_1c_2)^3} \right)^m ((c_1c_2 + m)!)^3 \right)$$