# Advanced Killer Sudoku Solver: Description of Approach Report

## How was the Killer Sudoku Grid represented?

The Killer Sudoku Grid was represented in nearly the same way as the standard sudoku grid, except for the inclusion of a list of cages to keep track of the cells covered by each cage in a grid and the sum of the values in each cell required to successfully fill the cage.

Each cage object in the list of cages for a Killer Sudoku Grid holds a cageTotal and a list of cell coordinates, stored as strings in the format "<row>,<column>". There are also two more lists, cageCombos and cageAssortments which are utilised by the advanced solver only for tracking all valid combinations of cell values and permutations of those combinations respectively.

## Description of approach and rationale behind it

The Advanced Killer Sudoku Solver begins by finding all possible combinations of cell values for each given cage. These combinations must follow the provided cage constraints in that they do not hold duplicate values and that the sum of all values matches the expected total for the cage. The combinations do not account for which value occupies which cell, this is to be dealt with in the following step concerning permutations of the combination.

As an example, consider a killer sudoku grid with 4 rows and 4 columns using 4 symbols 1, 2, 3 and 4. A cage consisting of two cells with an expected total of 5 may be solved by two combinations: 1 and 4 or 2 and 3. The number of values in a combination must match the amount of cells in the cage.

Once all possible combinations for a cage have been found, permutations of all these combinations may be identified. Permutations of these combinations are necessary so that each value in the combination may be sequentially allocated a cell in the cage. These will be referred to as assortments.

Continuing with our previous example, the permutations of the combinations 1 and 4, 2 and 3 include {1,4}, {4,1}, {2,3} and {3,2}. With our cage of two cells a and b, the first symbol in the set will be allocated to cell a and the second to b. Using these permutations we may consider all possible valid assortments of values in the cells of the cage.

After all assortments have identified, we may begin to recursively solve for a solution. Rather than iterating through each cell and then each symbol, we are able to instead iterate through each cage and then each assortment. This approach drastically reduces the amount of comparisons required to solve a sudoku grid.

For example, For a cage of three cells in a 4x4 grid, we would be required to try up to 64 different assortments before being certain we have found a valid assortment. The advanced algorithm, however, may only be required to try as few as 6 different assortments as the valid assortments for the given cage have already been identified.

Finally, solving for a solution follows the backtracking algorithm. A cage is set to one of its valid assortments. If the cage is valid, we move on to the next cage and set that to one of its assortments. Should an assortment be invalid to be set, we try the next assortment for the same cage. Once all assortments for a particular cage have been attempted we must backtrack and try the next assortment on the previous cage and so on until a solution is found or the starting grid is found to be invalid.

**Comparison with Backtracking Algorithm and Empricial Evidence**

As mentioned previously, the advanced solver improves upon the backtracking solver alone by preventing the trial of values that would not be valid for a given cage, reducing the amount of comparisons to be made overall.

While the regular backtracking algorithm must solve cell by cell for each and every symbol, the advanced algorithm begins solving with an advantage in the form of knowledge of which values may legally appear in which cells of each cage.

The results in the table below show the run time for each algorithm for a range of puzzles averaged over 10 tests.

| Puzzle | Killer Backtracking Solver | Killer Advanced Solver |
|---|---|---|
| 4x4 | 0.0388 seconds | 0.0372 seconds |
| 9x9 | 3.7271 seconds | 0.5484 seconds |

As can be seen in the table above, the advanced solver is only slightly faster than the backtracking solver for the 4x4 grid, however when using a 9x9 grid the advanced solver is significantly faster but a much larger margin on a consistent basis.