Project Proposal: Sudoku Solver

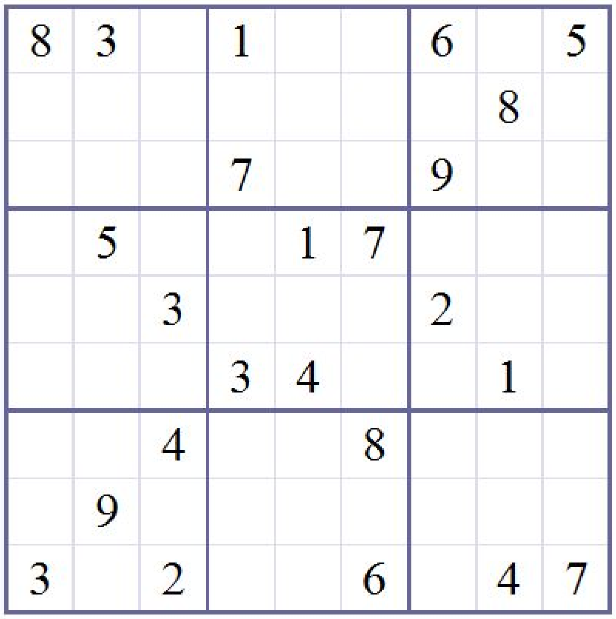
Rhett Crismon and Dylan Ahearn
CSCI4830/ECEN4313

Introduction

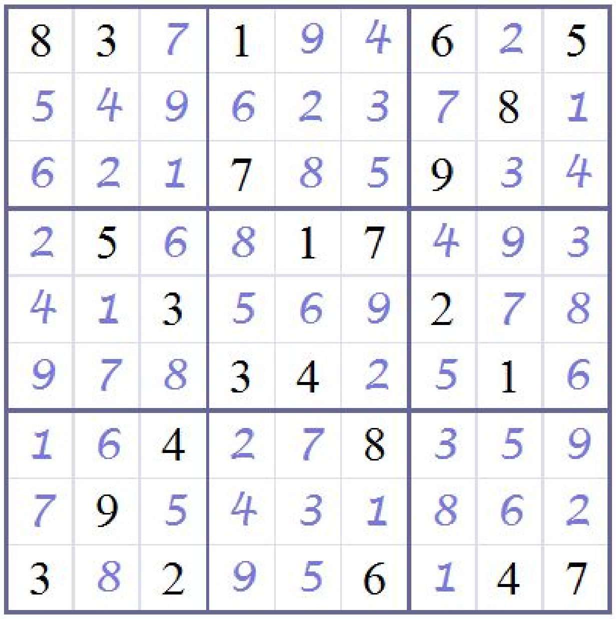
The sudoku puzzle, a partially completed constraint problem, can be solved algorithmically using several well known solutions. As a final project in Concurrent Programming, we will approach these algorithms with the main objective of adding parallelization and concurrency. The secondary objective is to analyze the performance of the modified algorithms with different computational constraints and puzzle difficulties. As a final objective, we will look for a hybrid solution that incorporates features and methods of the other concurrent algorithms as a higher performance solution.

Example Problem

The Sudoku puzzle is a mental game that consists of a nine by nine square grid (81 unique spaces) divided into nine three by three quadrants. Some spaces are filled in at the beginning of the puzzle, building the constraint (figure left) and the unique solution (figure right). The goal of solving the puzzle is to fill in the entire grid with the numbers one through nine, such that within each column, each row, and each three by three quadrant there must be exactly one of each number 1-9.



8	3		1			6		5
							8	
			7			9		
	5			1	7			
		3				2		
			3	4			1	
		4			8			
	9							
3		2			6		4	7



8	3	7	1	9	4	6	2	5
5	4	9	6	2	3	7	8	1
6	2	1	7	8	5	9	3	4
2	5	6	8	1	7	4	9	3
4	1	3	5	6	9	2	7	8
9	7	8	3	4	2	5	1	6
1	6	4	2	7	8	3	5	9
7	9	5	4	3	1	8	6	2
3	8	2	9	5	6	1	4	7

Proposed Solution

The three algorithms we will parallelize and analyze are the backtracking algorithm, the constraint programming algorithm, and the simulated annealing algorithm.

- Simulated annealing is an algorithm that starts with a randomly generated solution, in our case a randomly filled in Sudoku board. The algorithm makes a small change to this solution and based on how good this new solution is decides to either accept or reject the new solution. Then it repeats the process. A way of parallelizing this could be to have multiple threads generate and measure the fitness of a new solution. Then a lock would be placed on the current favored solution while it is compared and switched with a new solution.
- Backtracking is an algorithm that selects a cell and places an uninformed guess in it. If there is a conflict in the puzzle then a different number is selected. Then the next cell is filled in and the same check is done. If all possible values cause a conflict then the algorithm goes to the previous cell and tryes another value. This process is continued until all the cells are filled in and no conflicts are found. This process could then be parallelized to have each thread run the same algorithm and maintain a master list of solutions that do not work. There would only need to be locks on the object that keeps track of failed solutions.
- A constraint algorithm would use logic as its primary means of filling in squares. A benefit of this algorithm would be that cells are never filling in with guess; i.e. if a value is placed in a cell we know it is correct. Therefore we do not need to lock in this parallelized method. That is because the only read error that would occur is to read a blank cell when really it is about to be filled in. The only write error would then occur when a thread thinks that a cell is blank when it is not. Because the thread only writes correct values it will be overwriting a value with the same value. These are tolerable errors.
- Finally there will be a hybrid solution that takes advantage of parallelizing at least two of the three algorithms to try to get an even faster solution. A possible example is using the constraint algorithm to take care of easily placed squares while a guessing algorithm works on the more obscure cases.

Proposed Quantitative Evaluation

The quantitative methods and tools we will use to evaluate the properties of our project includes timing the processing time and number of operations of the algorithm with varying difficulties of puzzles and varying degrees of concurrent constraints. We will compare the timing of sequential iterations of the algorithm with the concurrent iteration to determine a speedup factor and correlation between varying degrees of concurrent constraints and hybridization. It will be useful to run these tests both on a laptop and on a server to see if there is a difference in speedup depending on the hardware’s ability to parallelize. The average value and variance of the results would be graphed on the same plot to show the results in a clear, concrete way.