A Sudoku Solver

CSC 242: Artificial Intelligence

Mike Schermerhorn

Introduction

The standard Sudoku puzzle consists of a nine by nine grid, broken into nine three by three boxes. Each of the eighty-one squares must be filled in with a number between one and nine. There are only three rules for filling in these numbers: each row, column and box must contain all of the numbers one through nine. Enough of the numbers are initially filled in to assure that the puzzle has exactly one solution.

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

Despite these simple rules there are 6,670,903,752,021,072,936,960 valid Sudoku puzzles (Felgenhauer and Jarvis). The size of the state space makes this an interesting and challenging constraint satisfaction problem. Clearly the search algorithm has to be more intelligent than just searching the entire space until it gets to a puzzle that matches the initial values. There are several search strategies that are effective for this problem. Simulated annealing has been shown to be effective (Lewis 394). Also, Sudoku has been modeled as a SAT problem (Lynce and Ouaknine). This paper will, however focus on the application of backtracking search and hill climbing methods to the Sudoku problem.

Variations and Terminology

Although the standard Sudoku puzzle is nine by nine, any n^2 by n^2 grid can be made into a Sudoku like puzzle. For example a sixteen by sixteen puzzle can be created with sixteen four by four boxes. In a sixteen by sixteen puzzle the rows, columns and boxes are filled with the numbers one to sixteen. Such a puzzle will be referred to as size four in reference to the value of n. The standard Sudoku will be referred to as a size three puzzle. Similarly a four by four puzzle will be called a size two puzzle. Although these puzzles are not really challenging, they do offer a simple case of the problem that is useful for developing the basic algorithms.

1			4
	3	2	

Throughout this paper size three puzzles will be referred to with a level number. The level numbers are from one to nine and refer to a series of Sudoku puzzle books by Frank Longo. Level one puzzles are the easiest and level nine are the hardest.

Backtracking Search

Perhaps the most obvious way to solve a Sudoku puzzle is to just pick the first empty square and assign one to that square. If the one conflicts with another value, then change it to a two. Keep doing this until a value that does not conflict is found. Once a value that does not conflict has been found, pick another square and repeat this process. If a square has no possible values, then return to the previously assigned square and change its value. This method of search is known as a backtracking search (Russell and Norvig 142). It is guaranteed to find a solution if there is one,

simply because it will eventually try every possible number in every possible location. This algorithm is very effective for size two puzzles. Unfortunately for size three puzzles there are nine possibilities for each square. This means that there are roughly 9^{81-n} possible states that might need to be searched, where n is number of given values. Obviously this version of backtracking search is not going to work for size 3 puzzles. Fortunately there are several means by which this algorithm can be improved: constraint propagation, forward checking and choosing most constrained value first.

Forward Checking

The first improvement on backtracking search is forward checking. Notice that the old version of backtracking search had to place a value and then check for conflicts. Instead it is easier to just maintain a list of which possible values each square can possibly have given the other numbers that have been assigned. Then when the values are being assigned to that square, only consider the ones that do not directly conflict with the other already placed numbers. For a size three puzzle forward checks can be stored in a nine by nine by nine boolean array. Basically each square has its own array of nine boolean values corresponding to each of the numbers that could possibly go in that square. If the third value in the array is set to false, then that square cannot contain a three. Maintaining these lists is simple. Whenever a new value x is assigned is assigned, go to every other square in the same row, column and box and mark false in its array for value x. Storing this information can be used in two ways. First, it can be used to verify that no value is ever assigned that directly conflicts with another assigned value. Second, if the array for any square contains all false values, then there is no possible value for that square and the most recently assigned value is wrong. Using forward checking the backtracking search can now solve size three puzzles.

Constraint Propagation

Forward checking can only catch conflicts right before they cause a certain branch to fail. It is possible to detect errors even earlier and prune off entire branches. Consider the following size two puzzle:

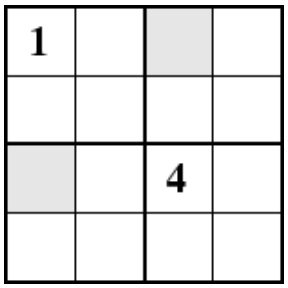
1			4
4			
2	3		
			3

It may seem like a good idea to place a four in the shaded box. It doesn't immediately conflict with any other locations and after placing it all of the squares still have possible values. Next look at the square right below the shaded one, it must be a two. Filling in a two there, however, leaves the lower left square with no possible value. This may not seem like a big deal, it only takes an extra two assignments to realize that the four was wrong, but what if the two isn't the next assignment. If the search is moving from left to right across the rows, it will assign the three empty squares to the right of the shaded one first. Depending on the possible values that these squares have, there may be up to three layers of branching before reaching the conflict. Each branch must now fail separately before the search realizes that the four was a bad choice. The result is an exponential increase in the time needed for the search to realize that the four was a bad choice. The solution is to assign values that only have one possible choice immediately. This is known as constraint propagation (Russell and Norvig 145). After each value is assigned by the search algorithm, constraint propagation iterates through the squares assigning values to squares with only one possible value. If a square has no possible values, the algorithm fails and returns to the search

which reselects the last value. If multiple values are assigned by constraint propagation, then they are all repealed at once upon a fail. In the example above after the four is assigned, constraint propagation realizes that the space below the four must be a two. It then notices that the lower left corner has no possible value and fails, returning to the search, which chooses another value for the shaded square.

Minimum Remaining Values

Another method for improving the backtracking search is the minimum remaining values heuristic (Russell and Norvig 143). The minimum remaining values heuristic is used to alter the order in which squares are guessed in order to reduce the number of branches at each level. Basically instead of choosing the first empty square, the square with the least number of possible values is chosen. For example, in the puzzle given below one of the two shaded squares would be chosen next. This is because the two shaded squares have two possible values and the other squares have three possible values.



By choosing the square with only two possible values instead of three the search tree only branches in two directions instead of three. Basically the search tree is reduced in size by a factor of two thirds.

Hill Climbing

Given the size of the state space it is logical to use a search method with a heuristic function to avoid searching less promising sections of the state space. One type algorithm that meets this

criteria is hill climbing. Hill climbing algorithms work by generating a list of successors of the current state, then choosing the one with the lowest heuristic value. In order to apply hill climbing to Sudoku three things must be defined: the start state, the successor function and the heuristic function. One way to go about this is to fill in each box so that it contains the numbers one to n^2 and allow successors to be generated by switching values within the same box (Lewis 391). Instead of filling in the boxes with the numbers one to n^2 the rows will be filled, but the idea is the same. Let the start state be defined as the initially puzzle with all of the empty spaces filled in such that each row contains the numbers one to n^2. Using this as a start state, the successor function can be defined as swapping any two non-fixed values in the same row. The heuristic can simply be the sum of the number of conflicts that appear within each column and each box. Since each row has exactly the numbers one to n^2 there are no conflicts within the rows. Hill climbing can successfully solve size two puzzles occasionally.

Random Restart

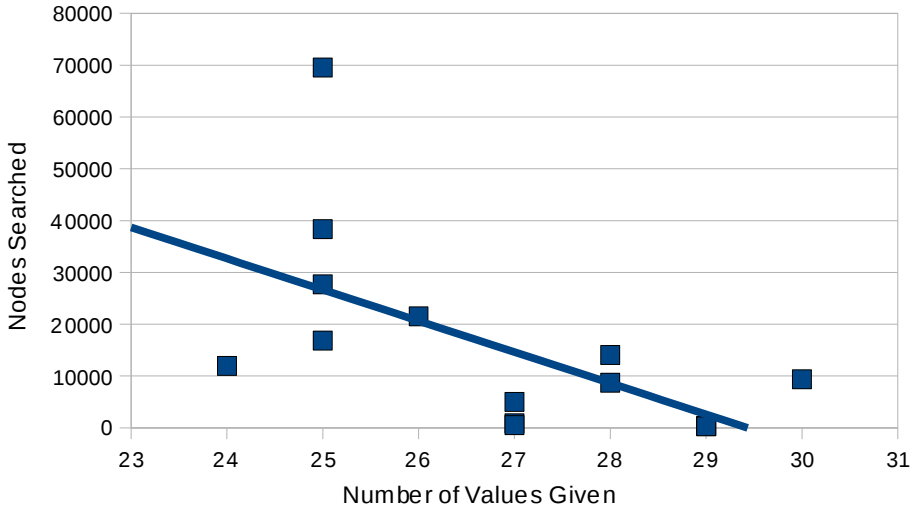
The general hill climbing algorithm described above is incomplete. This is because it can get stuck in a local minimum. One simple way to fix this is to randomly restart the algorithm whenever it goes a while without improving the heuristic value. This is known as random restart hill climbing (Russell and Norvig 114). This version of hill climbing does not quite suffice to solve the puzzle. The problem comes from the completely random way in which the variables are filled in. One technique for dealing with this is to switch certain values after the initial random placement. Known as post-swapping this technique attempts to minimize the number of values that conflict with the values given in the original puzzle (Jones et al., 45). Another possible solution is to use the constraint propagation algorithm. Applying constraint propagation before filling in any random values prevents some values from being randomly guessed incorrectly. Also, constraint

propagation can be applied in between the assignments of each row to make sure that the future rows agree better with the rows that have already been assigned. This method allows most puzzles of size three to be solved.

Puzzle Difficulties

Puzzle level	Average Number of Givens	Average Number of Nodes Searched			
		BTS	BTS+CP	BTS+CP +MRV	Random Restart Hill Climbing
1	28	4866	3	2	5579.6
4	27	20257	192	137	240340.5
8	25	19963	89	71	-

The chart above indicates the average performance of the various algorithms described on size three puzzles of varying difficulties. Clearly the backtracking search method has an advantage over the hill climbing method. It should be noted that the level one puzzles could be almost completely solved by the constraint propagation algorithm and required little actual search. The search algorithm is expensive and avoiding it entirely is very valuable in those cases where it is possible. It should also be noted that the logic used by the constraint propagation algorithm is very simple and its ability to solve the easy puzzles means that they do not require advanced logic. The graph below shows the number of nodes searched by a backtracking search without constraint propagation or minimum remaining values as a function of the number of givens.



The graph definitely indicates a correlation between the number values given in the puzzle and the time required to solve it. Puzzles with more values given can be solved after a shorter search.

Larger Puzzles

Expanding to larger Sudoku puzzles poses even more of a problem for the search algorithms in question. The increase in the size of the state space is exponential. For a sample size four puzzle the backtracking search using constraint propagation needed to expand 133797 nodes to find the solution. Using the minimum remaining values heuristic, the solution was found after expanding only 93094 nodes. The random restart hill climbing algorithm was unable to solve size four puzzles and none of the algorithms could solve puzzles of size five**.**

Conclusion

Although the Sudoku problem is a difficult constraint satisfaction problem, it is not completely invulnerable to search methods. Puzzles of size three or less can be solved very quickly and consistently. The backtracking search can consistently solve size three Sudoku puzzles after considering fewer than 200 states. Considering that there are 6,670,903,752,021,072,936,960 valid Sudoku puzzles, searching only 200 of them to find a solution is excellent. Random restart hill climbing is also successful on easy puzzles of size three, but is unable to solve more difficult puzzles.

References

Felgenhauer, Bertram and Frazer Jarvis. “Enumerating possible Sudoku grids.” 20 June 2005. 06 April 2009. <http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf>.

Jones, S.K., P.A. Roach, and S. Perkins. “Construction of Heuristics for a Search-Based Approach to Solving Sudoku.” In the Proceedings of the 27th SGAI International Conference on

Innovative Techniques and Applications of Artificial Intelligence. pp. 37-49. 2007.

Lewis, Rhyd. "Metaheuristics can solve sudoku puzzles." Journal of Heuristics. Vol 13, (2007): 387-401. 06 April 2009.

Longo, Frank. Green Belt Sudoku: Martial Arts Sudoku, Level 4: Not-So-Easy. New York: Sterling Publishing Company, 2006.

Longo, Frank. Red Belt Sudoku: Martial Arts Sudoku, Level 8: Super Tough. New York: Sterling Publishing Company, 2006.

Longo, Frank. White Belt Sudoku: Martial Arts Sudoku, Level 1: Easy. New York: Sterling Publishing Company, 2006.

Lynce, I., J. Ouaknine. "Sudoku as a SAT problem." In: Proceedings of the 9th Symposium on Artificial Intelligence and Mathematics, 2006.

Russell, Stuart and Peter Norvig. Artificial Intelligence: A Modern Approach. New Jersey: Prentice Hall, 2003.