Backtracking Sudoku Solver

Matthew Eads

March 12, 2015

1 Introduction

Many features of the sudoku solver used in this assignment were given by Prof. Anselm Blumer; the framework for reading in and printing out the sudoku puzzle, as well as the framework for solving were written for us. Thus the focus of this assignment was writing and experimenting with different heuristics to optimize the backtracking algorithm. Algorithms were outlined by Prof. Blumer, and Russel & Norvig's *Artificial Intelligence: A Modern Approach*.

The different heuristics were measured in time taken to solve diffenent puzzles (although they are all close to 0 for 9x9 puzzles), maximum depth reached, and total recursive calls. Maximum depth signifies the deepest level of recursion before reaching the solution. Solutions are unique and at a constant depth, which is included here for reference.

Difficulty	Solution Depth
Easy	44
Medium	52
Hard	58
Tough	60
Sixteen	168

2 Different Backtracking Heuristics

2.1 Naive Backtracking Algorithm

The simplest search uses no heuristics, and isn't particularly fast or efficient. The backtracking search picks the next unassigned position in the puzzle, and tries assigning values to that position in numerical order. For each assignment it removes legal values from the same row, column, and box, and then recurses with this new state. The recursive call either fails, in which case a new value is tried, or it succeeds.

This naive algorithm does not solve the 16x16 puzzle in a reasonable amount of time, but still solves 9x9 puzzles very quickly.

Difficulty	Time (s)	Max Depth	Total Recursive Calls
Easy	0.00	38	66
Medium	0.00	39	701
Hard	0.01	51	8,578
Tough	0.02	50	31,927
Sixteen	n/a	n/a	n/a

2.2 Arc3 Consistency

This is less a modification to the backtracking algorithm, and more an extra step to check for failure. This algorigthm checks that every position affected by the position in the puzzle just modified has remaining legal values. The more general algorithm checks that each position has a value consistent with every value in the given position, but due to the nature of the sudoku puzzle, this is as simple as ensuring no position has a domain size of 0 (no legal values).

Adding the arc consistency check to the naive algorithm improves the efficiency by a noticable amount, as it finds states with errors earlier and avoids some unnecessary recursion. The improvement is not significant enough however to allow completion of the 16x16 puzzle in a reasonable amount of time.

Difficulty	Time (s)	Max Depth	Total Recursive Calls
Easy	0.00	17	54
Medium	0.00	37	257
Hard	0.00	44	1,523
Tough	0.01	44	12,828
Sixteen	n/a	n/a	n/a

2.3 Minimum Remaining Values

The Minimum Remaining Values (MRV) heuristic chooses a position to test based on the fewest legal values available to that position. By first choosing positions with 0 legal values, it finds grids with errors quickly and fails early. Similarly it prioritizes positions with 1 legal value, which is naturally either correct or incorrect, but allows for much less guesswork.

This algorithm shows marked improvement over the naive heurisitic. For easy puzzles, the algorithm can pick a cell with one legal value on each recursive step, thus never failing and backtracking (note the max failure depth reached for the easier puzzles is 0, and the total recursive calls is equal to the depth of the solution). This heuristic also allows for completion of the 16x16 puzzle.

Difficulty	Time (s)	Max Depth	Total Recursive Calls
Easy	0.00	0	44
Medium	0.00	0	52
Hard	0.00	49	243
Tough	0.00	49	942
Sixteen	54.80	146	17,465,086

2.4 Maximum Remaining Values

The max. remaining values heuristic is functionally identical to the min. remaining values heurisitc, except it chooses a position with the largest number of legal values. Used alone, this heuristic is significantly worse than even the naive search, so much so that only the easy puzzle completes in a reasonable amount of time. Pairing this heuristic with other heuristics (arc consistency and most/least constraining value) improves efficiency somewhat (reduces time to 0s), although it is still only able to solve the easiest puzzle.

Easy	
Time (s)	4.85
Max Depth	42
Total Recursive Calls	5,700,037

2.5 Most Constraining Value

After a position is picked, the order of values to be tried is then chosen. Using the MCV heuristic, values are chosen based on how many other values they rule out. This is calculated by counting the number of positions in the same row/col with the value as a legal value. This number is maximized for MCV, or minimized for LCV (least constraining value).

When MCV is the only heuristic used, it is not particularly effective at all; although more constraining values are picked first, they aren't any more likely to be correct, or even fail sooner, as the results indicate. However it does offer some improvement when used in combination with the MRV heuristic, as is discussed in more detail later.

Difficulty	Time (s)	Max Depth	Total Recursive Calls
Easy	0.00	38	81
Medium	0.01	43	7,820
Hard	0.03	50	19,279
Tough	0.06	50	34,610
Sixteen	n/a	n/a	n/a

2.6 Least Constraining Value

This heuristic is the opposite of the MCV heuristic; choosing a value which constrains the fewest other positions in the puzzle.

Like MCV, when used alone, this heuristic does not offer any improvement from the naive heuristic, and in fact actually gives worse results for the harder puzzles than the MCV heuristic.

Difficulty	Time (s)	Max Depth	Total Recursive Calls
Easy	0.00	19	68
Medium	0.01	42	6651
Hard	0.05	51	30,102
Tough	0.39	51	219,377
Sixteen	n/a	n/a	n/a

3 Combining Heuristics

Although some heuristics are fairly powerful on their own, the most efficient sudoku solver is created by combining different heuristics. The arc-consistency checker offers improvement in all situations, as it simply finds errors more quickly, and does not change the algorithm significantly. The minimum remaining values heuristic is clearly the superior method for choosing the variable, but the difference between the value heuristics (naive, least/most constraining) is less pronounced, and more interesting to examine.

For conciseness, only the 16x16 puzzle is used to measure performance, as it has the most noticable difference in time. The Maximum Remaining Values heuristic is not being tested here due, so I will refer to Minimum Remaining Values as MRV. ARC3 refers to the arc-consistency algorithm, LCV to the least constraining value heuristic, and MCV to the most constraining value heuristic.

Heuristics Used	Time (s)	Max Depth	Total Recursive Calls
MRV	54.44	146	17,465,086
MRV & ARC3	59.19	145	15,312,511
MRV & LCV	55.21	146	17,465,086
MRV & MCV	55.47	154	17,956,284
MRV & ARC3 & LCV	60.93	150	15,817,945
MRV & ARC3 & MCV	60.98	153	15,758,792

4 Conclusion and Analysis

We see from these results that the simple Minimum Remaining Values heuristic is the fastest combination. However the MRV and ARC3 combination is the most efficient by measure of total recursive calls. It is likely that although checking for arc-consistency finds failure states sooner and thus prune the tree more effectively, the additional computational cost associated with the algorithm ends up not being worth it, at least in the 16x16 puzzle. Furthermore the LCV and MCV heuristics do not add a significant improvement over the MRV algorithm alone. The MRV algorithm is fairly efficient, and quite powerful, especially with relatively easy puzzles; if values can be narrowed down sufficiently, the algorithm will be able to always find positions with one or zero legal values. Thus there is little room for optimization by LCV and MCV (there are no or few values to pick from), and the additional computation is a net negative.

Much of the performance of different algorithms is very much an artefact of the structure of the sudoku puzzle; every position has the same number of legal values initially, each position has exactly one correct value, and the only way to easily constrain other positions is to pick a value for another posisition. Thus we find the simple MRV algorithm performs the best, while the other algorithms add more computational complexity than they return in efficiency.

4.1 Additional Tests

Here is a full list of tests, there are 18 possible combinations of these heurisitics, all are shown either in this table or previously in this report.

Heurisitics Used	Difficulty	Time (s)	Max Depth	Total Recursive Calls
Arc3 & LCV	Easy	0.00	17	63
	Medium	0.00	37	1,909
	Hard	0.01	47	4,343
	Tough	0.04	47	54,195
Arc3 & MCV	Easy	0.00	19	56
	Medium	0.00	39	2,508
	Hard	0.00	46	3,523
	Tough	0.01	44	13,802
Arc3 & Min. Remaining Values	Easy	0.00	0	44
	Medium	0.00	0	52
	Hard	0.00	48	232
	Tough	0.00	48	860
	Sixteen	59.19	145	15,312,51
Arc3 & Min. Remaining Values & LCV	Easy	0.00	0	44
	Medium	0.00	0	52
	Hard	0.00	34	88
	Tough	0.00	48	2,516
	Sixteen	60.93	150	15,817,945
Arc3 & Min. Remaining Values & MCV	Easy	0.00	0	44
	Medium	0.00	0	52
	Hard	0.00	48	224
	Tough	0.00	48	905
	Sixteen	60.98	153	15,758,79
Arc3 & Max. Remaining Values	Easy	0.00	25	1,229
Arc3 & Max. Remaining Values & LCV	Easy	0.00	26	1,097
Arc3 & Max. Remaining Values & MCV	Easy	0.00	25	708
Min. Remaining Values & LCV	Easy	0.00	0	44
	Medium	0.00	0	52
	Hard	0.00	35	91
	Tough	0.00	49	2,754
	Sixteen	55.21	146	17,465,08
Min. Remaining Values & MCV	Easy	0.00	0	44
	Medium	0.00	0	52
	Hard	0.00	49	234
	Tough	0.00	49	993
	Sixteen	55.47	154	17,956,284
Max. Remaining Values & LCV	Easy	4.08	42	4,833,133
Max. Remaining Values & MCV	Easy	1.92	42	2,258,248