Jeffrey Martin
CS 4341 Introduction To Artificial Intelligence
Professor Neil Heffernan
October 25, 2016

Solving A Rubik's Cube with Goal Decomposition and IDA* Search

1 Abstract

This report compares the efficiency of solving a Rubik's Cube using various sub goal decompositions. The IDA* algorithm is applied to find solutions from the current cube state to the next subgoal state; this process is repeated for each subgoal until a path (sequence of moves) has been discovered from the initial state to the solved state. Using this methodology, a decomposition was discovered that solves a rubik's cube more efficiently than myself, a solver of average rubik's cube solving capabilities.

2 Introduction

The iconic Rubik's Cube puzzle was invented in 1974 by Ernő Rubik. It is a cube composed of 6 colored faces. Each face contains 9 colored pieces. When the faces are turned, the distribution of colors varies throughout the cube. The goal of the puzzle is to turn the faces in such a way that the cube returns to its original state (each face is composed of a single color: red, green, orange, blue, white, or yellow).

Although it can be quite a challenge to solve the puzzle in itself, many have increased the challenge by considering what the most efficient method of solving the cube is. In this context, efficiency relates to the number of moves used to solve the Rubik's Cube. In July of 2010, using years of CPU time donated by Google, it was proven that any scramble of a Rubik's cube can be solved in 20 moves or fewer. Unfortunately, this is just a bound, and not an algorithm to find the most efficient solution to a scramble.

The problem is quite complex because the Rubik's Cube is capable of being in 43,252,003,274,489,856,000 unique states; therefore an exhaustive search for the optimal solution isn't a practical method. In 1985, Richard Korf was the first to describe the IDA* (Iterative Deepening A*) algorithm and in 2006 he applied this algorithm to optimally solve a few random scrambles of the rubik's cube. This was done by generating immense pattern databases to improve his heuristic.

It is a daunting task to search this state space in its entirety, and thus human solvers never do. Instead, the cube is viewed as a series of steps, or subgoals, that the solver must accomplish before moving onto the next step. This has two advantages: first, it generalizes cases; second, it decreases the longest path between current state and goal state. This first advantage is extremely important to human solvers, as they aim to memorize and apply generalized algorithms. The second is extremely important from a computer search perspective, because it significantly limits the state space, and thus the problem size.

In this project Rubik's cube solving using goal decomposition and IDA* search is explored. This method will apply advantage two discussed above to increase the utility of IDA* search. However, this approach eliminates the optimality of IDA*. This is because the optimal path from A to B combined with the optimal path from B to C does not guarantee an optimal path from A to C. The subgoals simply guide IDA* through the state space so it can avoid exploring regions that will likely be fruitless.

In this project various subgoal decompositions are compared to each other and to other rubik's cube solving methods. It is expected that the subgoal decomposition will solve a Rubik's Cube more efficiently than average human solvers (I use myself as a reference), and less efficient than an optimal solver.

3 Implementation

In order to solve a rubik's cube with a computer, a virtual model must be generated. This was done with an array of 54 characters. Each character represents 1 of the piece faces on the cube. Although this implementation is not necessarily the most efficient (with $4.3252 * 10^{19}$ different cube states, each cube can theoretically be represented with 66 bits or 8.25 bytes), It is computationally simpler. This is because the only operations needed to perform moves or check for goal consistency are memory reads and writes. A more compressed storage method would require more complicated compression mathematics and would therefore be more computationally expensive.

A goal decomposition is simply a list of goals. Each goal must check the cube to determine if it is consistent with the desired result. All of the decompositions explored featured augmentative goals. This means the current goal is satisfied if all previous goals are satisfied and some new constraints are met. Three different decompositions were implemented and each is explained below.

The beginners method decomposition follows the strategy used by most first time rubik's cube solvers.

1. Solve all 4 white edge pieces (white cross)

2. At least 1 white corner is solved
3. At least 2 white corners are solved
4. At least 3 white corners are solved
5. All of the white corner pieces are solved (entire white layer)
6. At least 1 edge in the second layer is solved.
7. At least 2 edges in the second layer are solved.
8. At least 3 edges in the third layer are solved.
9. all of the edges in the second layer are solved. (entire second layer)
10. All yellow edges have the yellow piece face on the yellow cube face.
11. All yellow edges are solved. (yellow cross)
12. At least 1 corner on the yellow face is in the right location (not necessarily the right orientation)
13. All corners on the yellow face are in the right location.
14. All pieces are solved. (Completely solved cube)

The random decomposition solves the pieces 1 by 1 until all of the pieces are solved. It does this in a random order.

The parallel piece decomposition solves the pieces 1 by 1, but it solves whichever piece can be solved in the fewest moves. However, do to problems encountered with the random decomposition (discussed in results) the ending of this decomposition was modified to Included the ‘all edges solved’ and ‘all corners in the right location’ goals.

1. At least 1 piece is solved
2. At least 2 pieces are solved
3. …
4. At least 17 pieces are solved
5. All edges are solved
6. All corners are in the right location.
7. All pieces are solved

The IDA* search algorithm relies on a heuristic to approximate the cost of solving a state. This allows the algorithm to avoid unpromising branches in the state tree.The heuristic used for the search counts the number of unsolved pieces. If an admissible heuristic is desired, this value is divided by 8. This is because each turn adjusts 8 pieces and it is possible for a single move to solve 8 pieces. Thus it would be unreasonable to conclude admissibly that x pieces require x turns to be solved. However, when the heuristic is made admissible it tends to do a worse job at approximating the cost (it is extremely rare that a single turn solves 8 pieces). When admissible, the algorithm could explore to about 8 moves deep before reaching the 1 billion state cap; when not admissible it could explore to about 12 moves deep. Thus, admissibility comes at a cost.

The solving algorithm starts with a scrambled cube and iterates over the list of goals. For each goal, it performs IDA* on the current cube state until the goal state is reached. This means it explores to a path cost depth. This depth is iteratively increased such that the algorithm explores deeper and deeper, but it explores the moves that seem most promising base on the current number of moves used and the heuristic approximation for the state. Once the goal state is reached, the algorithm records the moves used to get there and proceeds to the next goal. If at any time the program explore more than one billion states, the program aborts to avoid indefinite searching. When exploring moves, there are 18 moves to consider (all 6 faces can be rotated 90 degrees, 180 degrees, or 270 degrees). However, after the first move, there are really only 15 moves to consider, as it would be pointless to rotate the same face twice in a row. Thus the branching factor of this search is 15. Unfortunately, during initial testing it was discovered that the last goal of each decomposition was not solvable by the algorithm in a reasonable amount of time. This is because when all pieces are solved except two corner pieces it can in some cases take a minimum of 16 moves. This is far deeper than this implementation is capable of exploring To overcome this, the IDA* search supports additional ‘moves’ or handy end-cube algorithms to branch on when solving the last subgoal state. These additional moves increased the branching factor to 24 but greatly decreased the depth of the search.

Each of these decompositions was used to solve 10 randomly scrambled cubes. The results were compared to one another as well as to 10 solves done by myself on randomly scrambled cubes.

4 Results

All of the Raw Data is shown the the Appendix; The important aspects will be summarized here.

The Random Decomposition did not solve a the cube for any of its trials. It would consistently solve at least 17 of the pieces and then fail to solve this last three. As previously stated, solving the last 2 pieces can take a minimum of 16 moves and this IDA* implementation could not search that deep into the state space. From this it becomes evident that a one by one strategy is not ideal for a rubik's cube; an ideal algorithm would work on solving all the pieces at once.

Because of the results seen in the random decomposition, modifications were made to the Parallel Piece Placement and Beginners Method Decompositions. These modifications, as previously discussed, allowed useful end-cube algorithms to be considered as a single move and included in the IDA* branching.

The baseline for this experimentation is the number of moves need by an average human solver to solve a Rubik’s Cube. I solved a cube 10 times and used 107 moves on average.

Without the admissible version of the heuristic, the Parallel Piece Placing decomposition took an average of 111 moves while the Beginners Method decomposition took an average of 117 moves. Clearly neither method outperformed the baseline, but this is likely because of the lack of admissibility.

With the admissible version of the heuristic the Parallel Piece Placing decomposition took an average of 77.8 moves while the Beginners Method decomposition took an average of 70 moves. Clearly adding admissibility to the IDA* heuristic greatly improves the solution length; however, 4 of the trials for the Parallel Piece Placing decomposition failed to find a solution in the 1 billion search limit and 7 trials for the beginners method failed to. Thus admissibility in this case provides better solutions, but cannot reliable provide them.

5 Improvements

These results indicate that the Beginners Method Decomposition with an admissible heuristic is a promising decomposition, but it is to unreliable. When using this decomposition with admissibility, there were four goals in particular that tripped up the IDA* search. The first problem occurs when solving for the 4th edge piece in the second layer, in particular when the edge is in the correct location but the improper orientation. The second problem occurs when building an inconsistent cross and none of the yellow edges are have their yellow face on the cube's yellow face. The third problem occurs when establishing consistency within the yellow cross, and there is initial consistency within opposite edges rather than adjacent edges. The last problem occurs when solving for the last four corners if all 4 corners are unsolved.

A fourth decomposition, the Improved Decomposition was created to account for these problems with the hopes that it would work well with the admissible heuristic. Problem 1 was solved by completely restructuring the goals for the first to layers. Instead of solving one layer after the other, they are solved simultaneously. This is an advanced rubik's cube solving technique called the Fridrich Method. It works by solving an adjacent edge corner pair at once. All four pairs are solved 1 by 1. The second problem was fixed by adding an intermediate goal when building the inconsistent yellow cross. the intermediate goal checks that there are at least 2 edge pieces with their yellow face on the cube's yellow face, before the next goal checks for all four. The third problem was solved by once again adding an intermediate goal. This goal checks that the yellow cross has two adjacent solved edges before making all 4 edges solved. The fourth problem was solved by adding a new goal before the solved goal. This new goal checks that at least 1 corner is solved. This way the program does not attempt to solve all four corners at once.

The Improved decomposition used on average 72 moves to solve the scrambled cube. Additionally it successfully solved all 10 trials of the scrambled cube without exceeding the 1 billion state space limit. Clearly this implementation is more efficient than the previous decompositions as well as an average human solver.

6 Conclusions

The hypothesis for this experiment were correct, an optimized goal oriented IDA* search could out perform a human solver, however it did not perform optimally. In order to approach optimality, two things must be considered. First, goals should be selected as far apart as allowed for by the IDA* implementation, because IDA* is guaranteed to find an optimal path between goals when the heuristic is admissible. Second, Goals should be picked in ways that they do not deviate from the overall optimal solution. The goals guide the IDA* search, thus the guide should be as close to the ideal path as possible.

The first aspect described above is limited by computational resources; the computational demands could be improved if the Rubik's Cube model were made more efficient, or the heuristic was made more accurate while still maintaining admissibility. If this project were to be extended in the future, these would be improvements that could be made.

The second aspect described above is less promising. Although I have no backing aside from intuition, I do not believe there are common ideal goals shared by the optimal solve for every rubik's cube. An ideal subgoal decomposition could very likely depend on the scramble given. Generalizing these goals to fit all scrambles would likely not do well to achieve optimality.

7 Data

Parallel Piece Placement Decomposition without admissibility

Trial Number	Number of States Searched	Number Of Moves Used
1	79,946,286	121
2	98,975,690	119
3	76,131,728	121
4	470,177,560	119
5	57,543,871	137
6	316,286,623	119

7	39,508,456	126
8	501,840,064	71
9	33,346,038	59
10	293,720,925	121
Average	196,747,724	111
Standard Deviation	182,601,794	25.16

Parallel Piece Placement Decomposition with admissibility

Trial Number	Number of States Searched	Number Of Moves Used
1	345,425,322	53
2	1,000,000,000	N/A
3	1,000,000,000	N/A
4	742,764,848	87
5	392,724,993	83
6	297,516,394	60
7	737,895,180	88
8	354,891,197	96
9	1,000,000,000	N/A
10	1,000,000,000	N/A
Average (valid only)	478,536,322	77.83
Standard Deviation (valid only)	205,047,401	17.20

Beginners Method Decomposition without admissibility

Trial Number	Number of States Searched	Number Of Moves Used
1	102,252,611	121
2	26,580,158	121

3	171,668,839	120
4	23,602,612	121
5	68,078,576	124
6	24,594,280	97
7	71,607,805	100
8	11,047,335	137
9	19,052,671	113
10	97,555,742	125
Average	61,604,062	117.9
Standard Deviation	51,216,521	11.86

Beginners Method Decomposition with admissibility

Trial Number	Number of States Searched	Number Of Moves Used
1	1,000,000,000	N/A
2	228,533,887	72
3	1,000,000,000	N/A
4	1,000,000,000	N/A
5	1,000,000,000	N/A
6	285,178,407	78
7	1,000,000,000	N/A
8	1,000,000,000	N/A
9	1,000,000,000	N/A
10	175,483,879	60
Average (valid only)	229732057	70
Standard Deviation (valid only)	54857078.65	6.67

Human Solver (Myself)

Trial Number	Number Of Moves Used
1	90
2	119
3	121
4	118
5	126
6	111
7	40
8	120
9	102
10	142
Average (valid only)	108.90
Standard Deviation (valid only)	27.89

Improved Decomposition with admissibility

Trial Number	Number of States Searched	Number Of Moves Used
1	350667524	58
2	217293880	61
3	508845101	63
4	125633656	94
5	276928650	79
6	701968223	84
7	258303110	71
8	825773055	72
9	812354088	75

10	195893401	68
Average (valid only)	427366068	72.5
Standard Deviation (valid only)	265589033	11.04