Algorithm Description

This project aims to solve a Rubik's cube using an Iterative Deepening A* algorithm.

While a 2x2 cube would have a total of approximately 3.6 million of possible states, this number rapidly explodes as we increase the size. A 3x3 cube, for perspective, has 43 quintillion (43,252,003,274,489,856,000) possible configurations. This significant problem size provides the opportunity to see substantial improvements when applying various optimization techniques such as parallelism.

Clearly, this magnitude of states becomes rapidly unmanageable with the traditional A* algorithm, as too many unnecessary states would be stored in memory. Hence, we plan to use the variation of A* known as IDA* which iteratively increases a depth threshold of the visited states in order to control their memory footprint. Below is a basic description of how this algorithm works, assuming basic knowledge of A*:

1) Initialize the threshold to the heuristic value of the start state.

2) Perform a depth-first search around the start node, but pruning any path where the f-score $f(n) = g(n)+h(n)$ exceeds the threshold. This essentially blocks the graph search, keeping state expansion under control. The smallest $f(n)$ that exceeds the current threshold will be returned, and will potentially become the new threshold for the next iteration.
3) If the goal is found within the current threshold, terminate the search and return the solution. If the goal is not found within the threshold, update the threshold to the smallest exceeded $f(n)$ determined in the previous step. The search then starts anew from the root, applying again DFS but with an increased threshold.

In this algorithm, it is important to note that each iteration is fully independent from the one preceding it. Therefore, in subsequent iterations the same nodes may be visited over and over again. While this algorithm clearly offers no performance advantage over A*, the main objective of the algorithm is to minimize memory usage, freeing completely all memory after each iteration.

The goal of the project will thus be to utilize IDA* for memory purposes, whilst parallelizing it and making other optimizations to improve its performance and make it run within a reasonable time. This will provide a greater opportunity for performance optimization.

Plan for Parallelization

The main aspect that we plan to parallelize for this project is the DFS execution within each iteration. Within each iteration, we will have to perform DFS from the root, and this can be easily parallelized since each subtree is fully independent from the others. We can assign to each of the (8) cores a subtree to explore, and wait for them to return before collecting their results and resuming execution. Given that the graph for the Rubik's cube will not be acyclic, we may need to coordinate the threads to avoid re-exploring the same paths, which leaves room for optimization. Moreover, we can also compute heuristic values for the cube, either at a preprocessing stage, or dynamically, using parallel techniques.

Other optimization ideas

Store in some database all the precomputed optimal heuristic values for the states, as these are likely to remain identical independently of the instance of the problem. This could potentially make each run more efficient as the heuristics would not have to be calculated from scratch each time.

Input data

We aim to write this project in a modular fashion that will permit switching with minimal overhead from the 2x2 to potentially the 3x3 Rubik's cube or anything more complicated. Another complexity knob we will implement is the parallelization of the process, so that we can switch from 1 to 8 cores (and anything in between) to tangibly measure the effectiveness of parallelism.

The input data will mainly consist of random permutations from the goal state of gradually increasing depth. We will test the algorithm by starting at shuffled states at a closer predefined depth from the goal, and gradually increasing this distance.