Michael Nugent
CSCI 164 – Artificial Intelligence
Professor David Ruby
04/08/22

State Space Search

Github repository: https://github.com/nugentmichael1/StateSpaceSearch

Please email me if you require access: NugentMichael@mail.fresnostate.edu. I added what looked like your account.

Code Overview

Initially an attempt was made to use JavaScript to implement the search algorithms since it would enable a visualization of the search space via a virtual tile slider. However, the browser began to hang up on the easiest problems due to memory and processing speed constraints. Therefore, a switch was made to C++, and the previous BFS, DFS, and IDDFS JavaScript functions were translated.

The code starts with verification of the start and goal states so that no time is wasted on incorrectly formatted states. It checks three things: 1) both states must have sequential characters (i.e. 0-8; not 0,3,5,6,7,9,A,M,P); 2) Every character per string must be unique; and 3) Both strings must be of the same length.

All five search algorithms were implemented: BFS, DFS, IDDFS, A*, and IDA*; but each required a different implementation of the frontier or an expanded vector as in A*'s case. To handle this more elegantly, a frontier class was created that provided push, pop, and size functions of whichever structure was initially asked of it. This helped keep the code compact since two major functions could be reused regardless of algorithm (except A*): expand(), and checkReachedAddToFrontier().

A* was an exception because the reached vector is irrelevant to its implementation. Instead, it required an "expanded" vector to be checked. A* also required a priority queue, which was further

complicated by a need to remove repeat states once one was expanded. To accomplish this a customized priority queue was created that inherited methods from the standard priority queue.

A state class was created that held: the permutation, which represented the particular puzzle state; the parent of the state, the move take to go from the parent state to this state, the level or number of moves from the start state, and two heuristic attributes (g(n) and f(n)).

Once the respective algorithms found the goal state from the start state, traceBack() was initiated to follow the path from final state to origin state via pointers to parent states. Each movement character was appended to a string, and then at completion, the string was reversed.

The solution tester took a solution string, and a start state. Then it calculated what state would result from the application of the movement sequence. It did this with the help of two private functions: legalMove() to check if the next move was allowable, and makeMove() which made the move.

Main has three different methods to run the algorithms: command line arguments of start, goal, and desired algorithm; command line argument of a batch file of start states; and no command line arguments for hard coded start, goal, and algorithm parameters.

The heuristics used were out-of-place (OOP), and Manhattan distance. The former is weaker and just counts how many tiles are in the wrong position. The latter is stronger, and calculates how far away a tile is from its correct position, and sums up all.

The hard problems were unable to be solved. IDA* was ran for 12 hours and didn't get past the 57^{th} Manhattan distance heuristic distance, whereas the easy problems were solved in seconds. An attempt was made to use just A* because memory was not issue, but it, too, failed to produce anything. Another attempt was made to implement a "pattern database" heuristic that would enable many states

to be recognized and provide a more accurate estimate distance to the goal state. However, it required a lot of data capital in that 57 million sub problems needed to be solved, and just the first started to stall out at around the same place as IDA*.

Problem Results:

8-Puzzle

Start State	Solution String	Algorithm Used
"160273485"	DDLUURDLLDRRUULDRD	A* w/ Out-Of-Place
"462301587"	RDLURULLDRDLURULDRRULDDR	A* w/ Out-Of-Place
"821574360"	LURULLDRULDDRURULDLURRDLURDD	A* w/ Out-Of-Place
"840156372"	DLLURDLDRRULLURDDRUULDDR	A* w/ Out-Of-Place
"530478126"	DLDLURDRUULDLURDDRULLDRR	A* w/ Out-Of-Place
"068314257"	DRRULLDRDLURDRUULDLDRR	A* w/ Out-Of-Place
"453207186"	RDLUULDDRUULDRDR	A* w/ Out-Of-Place
"128307645"	LURDRULLDRDLURRDLURD	A* w/ Out-Of-Place
"035684712"	RDLDRURDLULDRUULDDRURULDRD	A* w/ Out-Of-Place
"684317025"	RURULLDDRURULLDDRRUULLDRDR	A* w/ Out-Of-Place
"028514637"	DRRDLURULLDRDLURRDLUURDD	A* w/ Out-Of-Place
"618273540"	LUULDRRULDDRUULLDDRULDRR	A* w/ Out-Of-Place
"042385671"	DDRRULLURDDRUULDLURDRULDDR	A* w/ Out-Of-Place
"420385716"	DLLURDDLURRULLDDRURD	A* w/ Out-Of-Place
"054672813"	DRDLURURDLULDRRDLUURDLDR	A* w/ Out-Of-Place
"314572680"	LULDRUULDRURDLULDRURDLDR	A* w/ Out-Of-Place
"637218045"	RRUULDLURDRDLULDRRULDR	A* w/ Out-Of-Place
"430621875"	DLURDLLDRULURRDDLURD	A* w/ Out-Of-Place
"158274036"	URDRUULDDRULDLURDRUULDRD	A* w/ Out-Of-Place
"130458726"	DLDRUULDDRULDR	A* w/ Out-Of-Place

16-Puzzle: Easy

Start State	Solution String	Algorithm Used
"16235A749C08DEBF"	LUURRDDLDR	A* w/ Out-Of-Place
"0634217859ABDEFC"	DRULDDRRRD	A* w/ Out-Of-Place
"012456379BC8DAEF"	RRDRDLLDRR	A* w/ Out-Of-Place
"51246A38097BDEFC"	RULURRDDRD	A* w/ Out-Of-Place
"12345678D9CFEBA0"	ULDLLURRDR	A* w/ Out-Of-Place

Michael Nugent
CSCI 164 – Artificial Intelligence
Professor David Ruby
04/08/22

16-Puzzle: Hard

Start State	Solution String	Algorithms Tried
"71A92CE03DB4658F"	?	IDA* w/ Manhattan, A* w/ Manhattan
"02348697DF5A1EBC"	?	IDA* w/ Manhattan, A* w/ Manhattan
"39A1D0EC7BF86452"	?	IDA* w/ Manhattan, A* w/ Manhattan
"EAB480FC19D56237"	?	IDA* w/ Manhattan, A* w/ Manhattan
"7DB13C52F46E80A9"	?	IDA* w/ Manhattan, A* w/ Manhattan