Assignment 1 – Part 2 Search

COMP3411 20T1, 20th of March

Chris Joy, z5113243

Question 1 - Search Algorithms for the 15 Puzzle

a) Number of states expanded from various start nodes for each search algorithm.

Algorithm	start10	start12	start20	start30	start40
UCS	2565	Mem	Mem	Mem	Mem
IDS	2407	13812	5297410	Time	Time
A*	33	26	915	Mem	Mem
IDA*	29	21	952	17297	112571

b) Efficiency of each algorithm, in terms of time and memory usage.

Algorithm	Discussion
UCS	Uniform cost search uses a **priority** queue, associating nodes with their cumulative cost from the root node. Its time complexity is $O(b^{1+(C/e)})$, where b is its branching factor, C is the destination cost and e denotes the steps taken to reach the destination. Hence $1 + C/e$ is the total number of layers expanded to reach the destination state. The space complexity of UCS is the same as its time complexity. In terms of memory usage, UCS performed the worst, running out of memory from start12 onward, probably due to the algorithm's exponential memory requirements.
IDS	Iterative-deepening search basically iterates Depth-first search (DFS) on different breadths (like Breadth-first search - BFS), hence having the space-efficiency of DFS and being able to find nodes closer to the root (like BFS). Its worst-case time complexity is $O(b^d)$, where b is its branching factor and d is its depth. In the table above, we see that IDS is timing out from start30 onwards, probably due to its exponential time complexity. The space required is only $O(d)$ which is that of DFS, hence we didn't see the algorithm running out of memory during our testing.
A*	A-star search is smarter than UCS and IDS as its cost function utilises a heuristic, which is an underestimate of the distance from a node to the destination. Its worst-case time complexity is $O(b^d)$, where b is its branching factor and d is its depth. This is the same for its space complexity, as it needs to keep track of all nodes that it has expanded, in memory. Hence why we see it running out of memory from start30 onwards.
IDA*	Iterative-deepening a-star search is the most efficient algorithm out of the four. IDA* may visit the same node many times as it doesn't keep track of nodes it's already visited (like A*). This means that its average time-complexity is worse than that of A*. However, it has a better space-complexity (i.e. $O(bd)$ - where b is its branching factor and d is its depth), because it doesn't leverage dynamic programming. This is perhaps why IDA* was able to complete start30 and start40 (unlike A* - running out of memory).

Question 2 - Heuristic Path Search for the 15 Puzzle

a & c) Length of path (G) and number of states (N) expanded for each heuristic search algorithm.

	start50		start60		start64	
IDA*	50	14642512	60	321252368	64	1209086782
1.2	52	191438	62	230861	66	431033
1.4	66	116342	82	4432	94	190278
1.6	100	33504	148	55626	162	235848
Greedy	164	5447	166	1617	184	2174

b) heuristic.pl

```
34  depthlim(Path, Node, G, F_limit, Sol, G2)  :-
35      nb_getval(counter, N),
36      N1 is N + 1,
37      nb_setval(counter, N1),
38      % write(Node),nl,   % print nodes as they are expanded
39      s(Node, Node1, C),
40      not(member(Node1, Path)),      % Prevent a cycle
41      G1 is G + C,
42      h(Node1, H1),
43      F1 is (0.8 * G1) + (1.2 * H1),
44      F1 =< F_limit,
45      depthlim([Node|Path], Node1, G1, F_limit, Sol, G2).
```

The cost function on line 43 was updated to take into account an arbitrary weight (w). The cost function used for the heuristic path search algorithm is the following:

$$f(n) = (2 - w) \cdot g(n) + w * h(n)$$

We substitute $w = 1.2$ in order to get our new cost function:

$$f(n) = 0.8 \cdot g(n) + 1.2 \cdot h(n)$$

d) Trade-offs for each algorithm (in terms of speed & quality of the solution).

Algorithm	Discussion
IDA*	Iterative deepening a* search expanded the most nodes compared to the other algorithms, hence being the slowest. Despite this, it has produced the most optimal solution (i.e. generates the shortest path solution).
Heuristic (w= 1.2, 1.4 & 1.6)	Heuristic Path Search's speed and quality of the solution tend to vary for different values of w. As w increases, we observe the algorithm expanded fewer nodes (i.e. becoming faster), but the quality of the solution gets worse (i.e. the path length increases).
Greedy	Greedy search seems to be producing the least optimal solution (i.e. generates the longest path solution). However, in terms of speed, it has expanded the least number nodes compared to the other algorithms, hence being the fastest.