An Experimental Comparison of Memory Bounded & Iterative Deepening A* Algorithm

Abhinandan Dubey
Stony Brook University, New York - USA

October 09, 2016

1 Heuristics

We have used the following as heurstics analyze the problem:

1. **Manhattan Distance** : This is basically the sum of horizontal and vertical distance from the current position of the tiles to their goal positions. More formally, for a k-puzzle problem, the value of this heuristic $h(s_c)$ on a current state s_c and a goal state s_g is given by ;

$$h(s_c) = \sum_{n=1}^{k} (\mid x_n(s_c) - x_n(s_g) \mid + \mid y_n(s_c) - y_n(s_g) \mid)$$

   where x_n and y_n denote the coordinates of the element (point) in the matrix.

   The heuristic is *admissible* because it is the sum of the distances from the actual positions of all tiles to their goal positions, the term $h(s_c)$ in the above expression always underestimates the actual solution path length. The reason is never over-estimates is that it is a distance metric, and measures the shortest path length between the two points.

2. **Number of misplaced tiles (NMT)** : It is the count of the number of tiles which are not on their actual positions

   The heuristic is *admissible* because it is simply a count of tiles that are incorrectly placed. The count can never be an over-estimate as it is basically showing the difference between the current state and the goal state.

2 Memory Issue with A*

The A* algorithm requires to maintain a list of already visited nodes and a queue of nodes which have to be visited according to the priority (open-list). This results in a problem when the test case for the problem is complex enough for A* to keep looking for the goal state while increasing the number of visited nodes.

If we use a heuristic which is just a constant, A* will become a uniform cost search, and the space it requires would hence increase exponentially with the input size.

To solve the 15-puzzle problem, lets consider the worst case space requirement by A*. The algorithm keeps all visited nodes in the memory, which is exponential. In the worst case, where every step ever taken is wrong, there will be an exponential number of nodes in the queue before reaching the goal node.

3 Memory Bounded Search Algorithm - IDA*

We have implemented Iterative Deepening A* as my memory bounded search algorithm. The algorithm has a memory bound on the $f - values$ of the nodes. The procedure starts with a relatively low value of f, which is declared as *maxf* in the code. It then performs A* over the start state, and checks at every step if the $f - value$ of the state is below the maximum value of f allowed in that stage. If it is not, then the loop breaks, and the algorithm starts with a higher bound which is sum of the current bound and the minimum value of f seen till now.

It is important to note that the $f - values$ we're talking about here are the values of

$$f(s) = g(s) + h(s)$$

Completeness: The method is complete because it always finds a solution if it exists. This is obvious from the fact that if we call this with a very high value of $maxf$, the algorithm is same as A*
Optimality: Since the algorithm starts with an upper bound to reach the goal, and proceeds with an A*, since this now

reduces to an A* with a bound, it is necessarily finite and thus gives an output in finite time.

Complexity Analysis:

Space : Since IDA* does not keep a list of all nodes except the ones on the current path, it requires linear amount of space in terms of the length of the path it finds to the goal node.

Time : The time complexity of IDA* will also depend on the maximum value of f that it is initiated with. It expands all the nodes in the frontier of the initial state with f-values less than the value of $maxf$ supplied. Since this implementation of A* uses a priority queue, it expands all the nodes in the queue, which is same as A* (in the last step, where it finds the solution). Thus, the final stage of IDA*, where it finds the goal state, expands the same set of nodes as A*. Thus asymptotically, IDA* expands the same number of nodes as A*.

4 Performance of A* Implementation

We tested our algorithms on randomly generated set of 10 8-Puzzle problems using a random test case generator, and a set of 10 15-Puzzle problems using the same. We have provided the test cases in `randomtestcases.zip`

Table 1: Performance of A*

Test Case Number	Number of States explored		Time (in millisec)		Depth	
	Manhattan Dist.	NMT	Manhattan Dist.	NMT	Manhattan Dist.	NMT
1	185	147	19.210100174	5.8810710907	64	22
2	407	1011	40.9898757935	93.6398506165	50	70
3	154	462	17.548084259	33.0278873444	40	34
4	165	852	14.2951011658	72.9451179504	40	44
5	73	377	9.43112373352	20.9939479828	24	28
6	58	197	4.5690536499	9.41395759583	22	36
7	188	1499	19.2420482635	172.061920166	50	48
8	489	266	59.0319633484	15.3639316559	50	26
9	66	401	5.42402267456	24.6870517731	22	50
10	7	8	0.679016113281	0.540018081665	6	6
11	155	47	23.453950882	1.7409324646	16	16
12	3209	376	1049.30496216	30.6560993195	25	25
13	12652	7242	13851.8650532	4678.20501328	136	200
14	11	12	1.72090530396	0.550031661987	10	10
15	17	100	2.37607955933	4.76503372192	11	11
16	1942	11040	497.585058212	13233.9830399	122	214
17	6942	15688	3970.51310539	26248.0208874	224	232
18	3	3	0.827789306641	1.36113166809	2	2
19	85	1643	11.2509727478	282.732009888	27	47
20	4507	3106	1858.24799538	824.815988541	70	132

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx