Comparing Uninfromed Search Algorithms

Abstract:

Uninformed search is the type of search where the agent does not have any extra information about the task environment except what is provided in the problem definition. Uninformed search can check if a particular node is the goal state and expand a node to its successors. Uninformed searches are further categorized depending on how the nodes are expanded[1]. In this paper, we analyze how 5 different types of uninformed search: breadth first search, depth first search, depth limited search, uniform cost search and iterative deepening search, perform in different problems with varying level of difficulty.

Problem Description:

a) 8-puzzle problem

8-puzzle is played in a 3×3 grid with 8 squares numbered from 1 till 8. The goal of the game is to arrange the blocks so that square blocks are in order, with the blank square occupying the first spot (top-left) in the grid. The blank square can move 1 square adjacent to its current position in any direction (up, down, left, right) given that its new position is still in the 3×3 grid.

It is important to notice that not every starting position for the 8-puzzle problem has a valid solution. The starting state for the 'Easy 8-puzzle problem' was generated by starting from the goal state and making 4 random moves where as the starting state for the 'Hard 8-puzzle problem' was generated by starting from the goal state and making 22 random moves. By construction, the problem has a solution within 4 moves and the second problem has a solution within 22 moves.

b) Jumping problem

Two starting states with different course length were selected for the jumping problem[2].The 'Easy jumping problem' was on a course length of 5 where as the 'Hard jumping problem' was on a course length of 30. The number of possible solution sequence for the jumping problem increases exponentially with the course length. The same search algorithm can perform with very different relative efficiency for the jumping problems in different staring states.

Data Collection:

Each of the 4 problems described above was searched using 5 different uninformed search algorithms. For each search, the length of the solution sequence and the total number of nodes expanded in the process of finding the solution sequence was recorded. The data is recorded in two different table labelled as the "8-puzzle problem" and the "Jumping problem". The length of the sequence for the solution is interpreted as the "Total number of actions" needed to reach the goal state from the start-state.

8-Puzzle Problem

Search Algorithm	Total number of actions	Number of nodes expanded
Easy 8-puzzle problem		
(breadth-first-search)	4	23
(depth-first-search)	26	26
(depth-limited-search)	4	15
(uniform-cost-search)	4	22
(iterative-deepening search)	4	41

[1]The description of the 8-problem was paraphrased from: https://www.cs.princeton.edu/courses/archive/spr10/cos226/assignments/8puzzle.html

[2]The description of the 'jumping problem' was provided in the Lab Assignment. The complete description of the 'jumping problem' is available here: https://www.cs.grinnell.edu/~weinman/courses/CSC261/2018S/assignments/search.pdf

Search Algorithm	Total number of actions	Number of nodes expanded
Hard 8-puzzle problem		
(breadth-first-search)	6	109
(depth-first-search)	428	434
(depth-limited-search)	20	38,940
(uniform-cost-search)	6	92
(iterative-deepening search)	6	196

Jump Problem

Search Algorithm	Total number of actions	Number of nodes expanded
Easy jumping problem		
(breadth-first-search)	4	10
(depth-first-search)	4	6
(depth-limited-search)	5	5
(uniform-cost-search)	4	12
(iterative-deepening search)	4	25
Hard jumping problem		
(breadth-first-search)	9	8,616
(depth-first-search)	9	41
(depth-limited-search)	30	30
(uniform-cost-search)	9	23,192
(iterative-deepening search)	9	35,803

Data Analysis

Optimality of Solution

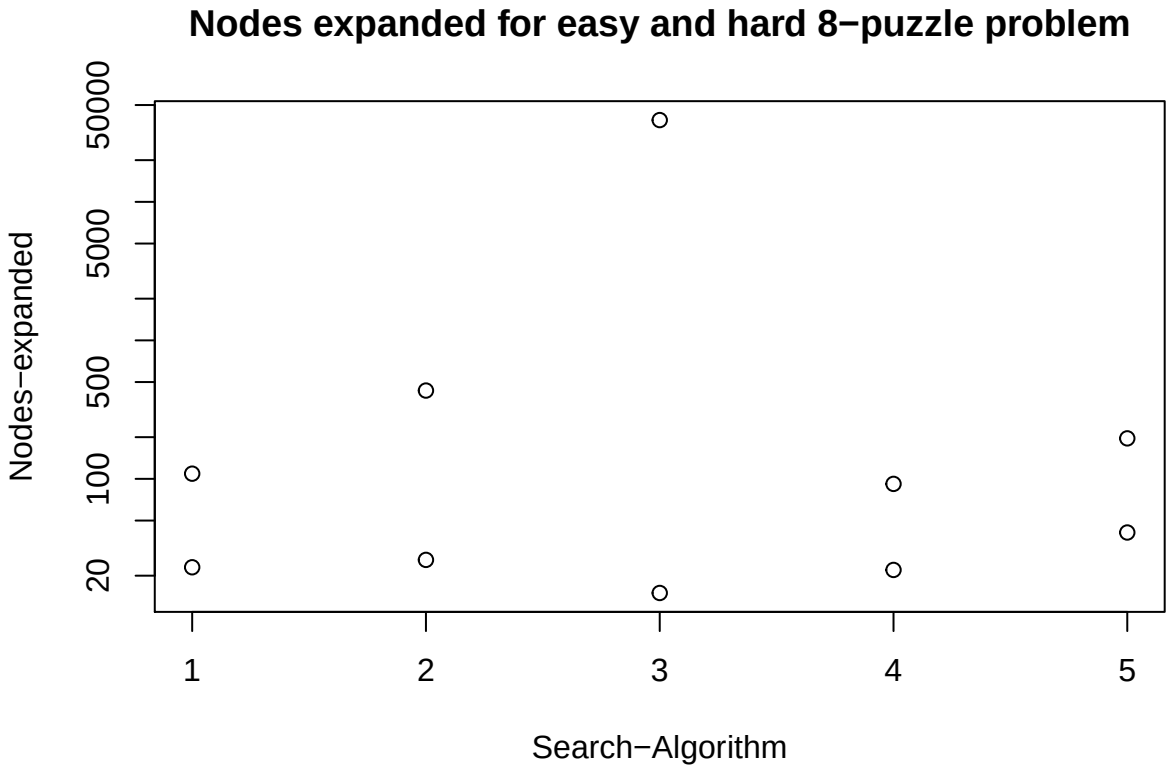
For any given problem, breadth-fist-search, uniform-cost-search and iterative-deepening-search algorithms are optimal. These three logarithms guarantee that no solution with fewer number of actions exist. Analyzing the solution length from the data, we see that all three algorithms found a solution of length 4 for the *Easy 8-puzzle problem*, solution of length 6 for the *Hard 8-puzzle problem*, solution of length 4 for the *Easy jumping problem* and a solution of length 9 for the *Hard jumping problem*. The data is consistent with our understanding of how these three algorithms work as they all returned the minimum solution cost among all the search algorithms.

Depth-first-search and depth-limited-search are not optimal search algorithms. For the *Easy 8-puzzle problem*, depth-first search found a solution of length 26 when all the other algorithms found the optimal solution of length 4. Unsurprisingly, depth-first-search performed the worst relatively in the *Hard 8-puzzle problem*. It found a solution at depth of 428 where as the depth-limited-search with its limit capped at 22 found a solution at depth of 20. Since both depth-limited-search and depth-first-search are not optimal, it found sub-optimal solutions. Moreover, depth-first-search has not limit in how deep it can go and hence, found a solution at an unimpressive depth of 428.

For jump problems as well, depth-first-search performed the worst in finding optimal solutions. Depth-limited-search found an optimal solution for both the jumping problems. This is very interesting as depth-limited-search is not optimal and yet, found optimal solutions for both *Easy jumping problem* and *Hard jumping problem*.

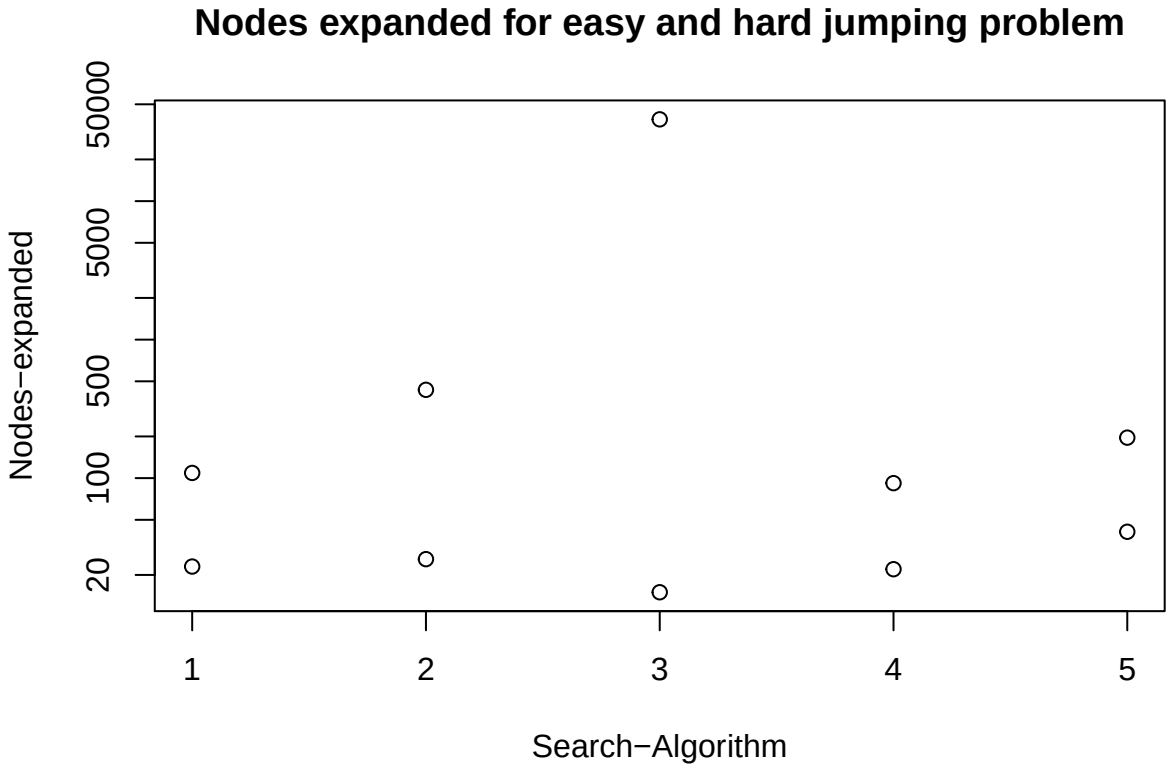
Number of nodes expanded

For the easy problems, iterative-deepening-search performed the worst among all other algorithms. This is understandable as iterative-deepening-search works by increasing its limit and repeatedly performing depth-limited-search. When the solution is near-by, iterative-deepening-search falls behind all the other algorithms. Comparing the number of nodes expanded for the other algorithms for the easy problems, there aren't any drastic differences. Depending on the star-state, the algorithms could perform relatively better or poorer; we cannot make any bold assertions just based on the results from the easy problems.



Legend for Algorithm numbering:
1 breadth-first-search
2 depth-first-search
3 depth-limited-search
4 uniform-cost-search
5 iterative-deepening-search

As expected, more number of nodes are expanded by the algorithms for the *Hard 8-puzzle* compared to the *Easy 8-puzzle*. Notice that the increase in the numbers of nodes expanded by depth-limited-search increases dramatically, it expanded 15 nodes for the easy problem where as 38940 nodes for the hard problem. This is quite a remarkable increase in the number of nodes.



Legend

for Algorithm numbering:
1 breadth-first-search
2 depth-first-search
3 depth-limited-search
4 uniform-cost-search
5 iterative-deepening-search

For the jumping problems as well, more number of nodes are expanded by all algorithms for the hard problems compared to the easy ones.Breadth-first-search performs relatively very poorly in the hard problem compared to its performance in the easy problem; it expands only 10 nodes in the easy problem where as it expands 8616 nodes in the hard problem. Iterative-deepening-search performs worse that breadth-first-search with this metric, it expands 25 nodes for the easy problem where as it expands 35803 nodes in the hard problem. Likewise, uniform-cost-search also performs very poorly in the hard problem compared to the its performance in the easy problem. Very surprisingly, the two algorithms that are both incomplete and sub-optimal perform the best in the jumping problem. Depth-first-search expands only 6 nodes for the easy problem and 41 nodes for the hard problem. On the other hand, depth-limited-search expands 5 nodes for the easy problem and 30 nodes for the hard problem. These numbers are very impressive compared to the results given by the other three algorithms.

#####Conclusions

Even though uninformed search algorithms may seem limited by the range of actions available to the agent, the different ordering of the nodes in the frontier can cause drastically different performances. In terms of finding optimal solutions, breadth-first-search, iterative-deepening-search and uniform-cost-search will always return the optimal solution if there exits one.

Depth-first-search and depth-deepening-search may seem inferior to the other algorithms as both of these algorithms are not complete and optimal. However, there may be certain problems where these depth based algorithms excel. Especially when it is not important to get the optimal solution, depth-first-search and depth-deepening-search can generate solution by expanding minimal number of nodes. Depending on the difficultly of the problem and the starting state, each of the 5 uninformed search algorithms can show dramatically different relative efficiency. For this reason, it is recommended to run all the search algorithms

on the same problem as it can't always be told beforehand which one will perform better.