# Searching Algorithms.

# We are going to examine 3 search algorithms. Breath First Search, Depth First Search and A\*.

# In a breadth-first algorithm, the search is concentrated at the high level and not until a solution is found at this level does the algorithm go deeper into the lower levels. This algorithm is queue-based, and almost the entire search space needs to be searched before an answer is found. On the average, this algorithm is O(N) in complexity where N is the number of nodes in the tree representing the search space. Best case performance for a breadth-first algorithm is and worst case performance are O(N).

# In a depth-first algorithm, the search is concentrated at the lower levels. This algorithm is stack-based, and a potential solution may be found early in the search. The worst case performance is no better than a breadth-first algorithm, but on average a depth-first algorithm will find a feasible solution quicker. On the average, this algorithm is O(N) in complexity where N is the number of nodes in the tree representing the search space. Best case performance for a depth-first algorithm is O(1), while worst case performance is O(N)

 A\* where the priority functions are split into two components. One represents the known cost to get to a node in the search space while the other represents the estimated cost of continuing towards the goal. The estimated cost must be positive and must be an underestimation of the actual cost. It can be shown that A\* is the best of all best-first search algorithms. The A\* algorithm is more complex because it requires the definition of the priority function. On the average, this algorithm is an order-of-magnitude less than N in complexity where N is the number of nodes in the tree. Best case performance for a best-first algorithm is O(1), while worst case performance is O(N).

There is an interesting tradeoff between the cost of visiting a node in the search space and the cost of calculating the priority function. If the search space is small, inexpensive to traverse, and the cost of calculating the priority function is expensive, then the depth-first and breadth-first algorithms may have better total performance over the best-first algorithm. The cost of calculating the priority function can be controlled by varying the quality of the answers returned by the priority function. If the search space is complex and large, then the cost of calculating a precise priority function is negligible. On the other hand, some situations call for a cheap priority function. In the limiting case, the priority function could be simply that all next steps have the same priority and the algorithm becomes a breadth-first algorithm. Alternatively, the priority function could reflect the depth of the search space and the best-first algorithm would behave like a depth-first algorithm.

The results in this thesis on methodology performance are analogous to the associated results in search theory. All three search algorithms are complete for a static search space. The differences appear when the search space is dynamic. A breadth-first search algorithm may miss a solution. Both a depth-first search algorithm and a best-first search algorithm will find a solution in a dynamic search space. In some cases, the best-first search algorithm will find a solution in less than or equal time to the other two methodologies. The worst case performance of all three search algorithms are the same. On average, the breath-first search algorithm will find a solution in O(N). On average, the depth-first search algorithm will find a solution in O(N). On average, the best-first search algorithm will find a solution is an order-of-magnitude less than N.

References:

<http://infolab.stanford.edu/~burback/watersluice/node84.html>

Intorduction to Java Programming 2010 Liang