[[1]](#footnote-1)

**SELECTION OF SEARCH ALGORITHMS AT RUNTTIME**

Pramod Kulkarni, Prasad Hegde, Prasann K Patil *,* M Gaurav

*Dept. of CSE, Ramaiah Institute of Technology*

*ABSTRACT:* There are many search algorithms available in the current scenario mainly classified into two categories :- informed and uninformed algorithms. Different algorithms maybe be suitable for various types of databases depending on the complexity, size of the database and the data structures used. Data structures can include linked lists, arrays, search trees, hash tables, or various other storage methods. The appropriate search algorithm often depends on the data structure being searched. Search functions are also evaluated on the basis of their complexity, or maximum theoretical run time. We have chosen Data sets with over 10000 data items in it. The main aim should be to reduce this cost by efficiently choosing an algorithm best suitable for the given dataset at runtime. There are various parameters that can be taken into consideration, like time taken to retrieve the search item, memory used, user reactions

**Keywords:** Informed algorithms, Uninformed algorithms, Data sets

.

1. INTRODUCTION

Comparison of search algorithms can be done by putting them

in context with one another. Search problems can be represented

as a search graph with associated states, operators and cost of

traversal. These search algorithms can be classified into two

categories:

* Informed search
* Uninformed search

An uninformed search algorithm generates the search tree without

using any domain specific knowledge.

An informed search algorithm on the other hand uses a predefined heuristic function to decide upon the next course of action.

A few of the uninformed search strategies are given below:

* Breadth First Search
* Depth Limited Search
* Depth first Search
* Uniform-cost search
* Iterative deepening depth-first search
* Bidirectional search

Some of the informed search strategies include:

* Greedy best first search
* A\* search

The following properties can be used to decide the suitability of the

search algorithms:

* Completeness
* Time complexity
* Space complexity
* Optimality

1. **LITERATURE REVIEW**

Kamlesh Kumar Pandey[1] in his paper “A Comparison and Selection on Basic Type of Searching Algorithm in Data

Structure “ explains about the different types of search algorithms which include linear search. This research talks about the working of various search algorithms, selecting algorithm on the basis of parameters like total number of comparison, type of data Structure, time complexity and space complexity.

Yujun Zheng[2] in his paper “A category theoretic approach to search algorithms” explains how backtracking is used for search and optimization problems, but varies with data set. At first specification of various search algorithms is specified, followed by derivation of the given algorithm, and finally coming up with an efficient algorithm which can solve various given problems. The approach achieves a high level of abstraction and automation without losing performance.

Feng Hao [3] proposes fast search algorithm for a large fuzzy database of codes or data with a similar binary structure. The fuzzy nature of the codes and their high dimensionality provide many modern search algorithms, mainly relying on sorting. The algorithms being used are based on a brute force exhaustive search through a database, looking for a match that is close enough showing a substantial improvement in search speed with a negligible loss of accuracy. In addition, the author demonstrates that the empirical results match theoretical predictions.

Raymond Chiong[4] in his paper has done a comparative study on informed and uninformed search for intelligent travelling planning. Examination of different search algorithms from artificial intelligence for solving different tasks such as finding the shortest path. The author has investigated about various informed and uninformed search algorithms based on major cities and towns. An improved version of Dijkstras is presented for this task. It is shown that best first search and A\* are productive to find short useful paths while algorithms such as hill climbing and various other uninformed search algorithms are less useful. Finally it is concluded that the improved version of Dijkstras algorithm is the best in accordance to parameters like accuracy and optimality.

Pratyaksha osca[5] in her paper “Optimised A\* algorithm in hexagon based environment using parallel bidirectional search” talks about how shortest path algorithms can be implemented in real time scenarios, mainly in IT sector especially in gaming industry. She also tries to stress that A\* algorithm is one of the most popular shortest path algorithm. The paper explains the optimization of A\* algorithm using PBS which results in a PBS A\*which accelerates the naïve A\* algorithm by early 70% faster in the given environment. One of the fascinating games include creating a Non Playable Character who tends to move only in the available path and occasionally finds the player’s position too

1. **Problem Statement**

There are various search algorithms in the current scenario where each algorithm can be utilized for different problems in different situations. For example, if the user knows the depth of the result node in the search tree, user might utilize Depth First Search. Every algorithm has its own merits and demerits. We provide users with various options such as heuristic values, depth, breadth etc. Our aim is to choose the best suitable algorithm for the real time scenarios with various parameters such as time complexity, space complexity, completeness and optimality will be provided at runtime.

1. **Methodology**

In this section, we describe the methodology used in our application and we also explain the theory behind the search algorithms in artificial intelligence.

* 1. **Uninformed search algorithms**

In uninformed searches, there is no idea about whether one non goal state is better than any other. The search is considered to be blind.

* + 1. **Breadth First Search**

Bfs is a strategy where in the root node is expanded first then all the successors of the root node are expanded in the next step and so on.

Algorithm:

Insert the root into the expanding queue

While expanding queue is not empty

      Copy contents of expanding queue to temporary queue

      Empty the expanding queue

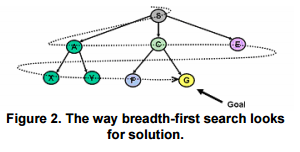
      For each node in the temporary queue

            Dequeue one element from the temporary queue

            If the path is ending in the goal state, print the path and exit

            Else

                  Insert all the children of the dequeued element into the expanding queue



Properties of Breadth First Search

* Complete: Yes ,as it always reaches the goal
* Time: O(b^d+1) (equal to the number of nodes generated)
* Space: O(b^d+1) (keeps every node in memory either in fringe or a path to fringe)
* Optimality: Yes (if it is guaranteed that deeper solutions are less optimal)

Ex:step cost=1

* Space is a bigger problem than time in Bfs
  + 1. **Uniform Cost Search**

It is an extension of Breadth First Search with an extra parameter of step cost. In UCS instead of expanding the shallowest node it expands the node with the lowest path cost. It does not consider about the number of steps but only the total cost

Algorithm:

Insert the root into the queue

While the queue is not empty

      Dequeue the maximum priority element from the queue

      (If priorities are same, alphabetically smaller path is chosen)

      If the path is ending in the goal state, print the path and exit

      Else

            Insert all the children of the dequeued element, with the cumulative costs as priority

Properties of Uniform Cost Search

* Complete: Yes ,only if step cost is greater than 0. In case it is 0, it gets stuck in an infinite loop.
* Time: O(b^d+1) (equal to the number of nodes generated)
* Space: O(b^d+1) (keeps every node in memory either in fringe or a path to fringe)
* Optimality: Yes, for any step cost
  + 1. **Depth First Search**

Dfs expands along the deepest node of the search tree, the search progresses along the next deepest level of th search tree where the nodes have no successor. This strategy is implemented is implemented using LIFO queue.

Algorithm:

Insert the root node into the priority queue

While the queue is not empty

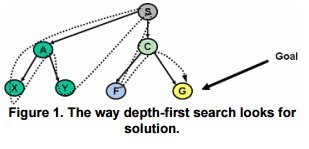
      Dequeue the element with highest priority

      (In case the priorities are same, the alphabetically smaller element is chosen)

      If the path is ending in the goal state, print the path and exit

      Else

  Insert all the children of the dequeued element, into the queue



Properties of Depth First Search

* Complete: No, tends to fail in case of infinite depth spaces
* Time: O(b^m)where m is the maximum depth, it is optimal if m is much larger than the depth d
* Space: O(bm) which is linear space(in this search algorithm the user only needs to know a single path and expanded unexplored nodes)
* Optimality: Not optimal, as it may find a non optimal goal first
  + 1. **Iterative Deepening Search**

To avoid the infinite depth problem of DFS, it can be searched until depth L, i.e. we don’t expand beyond depth L. If the solution is deeper than L, it can be increased iteratively. It this inherits the memory advantage of DFS.

Algorithms:

function ITERATIVE-DEEPENING- SEARCH(problem)returns a solution or failure

Inputs: problem, a prolem

for depth  0 to oo do

result  DEPTH-LIMITED- SEARCH(problem,depth)

if result != cutoff then return result

Properties of Iterative Deepening Search

* Complete: Yes
* Time: O(b^d)
* Space: O(bd)
* Optimality: optimal. If step cost=1 or increasing function of depth
  1. **Informed Search algortihms** 
     1. **Greedy Best First Search**

This expands the node which is nearer to the goal on the basis that it might lead to the solution quickly. It evaluates the nodes using the heuristic function: f(n)=h(n)

Properties of Greedy Best First Search

* Complete: No, as it can get stuck in loops
* Time: O(b^m), but a good heuristic can give dramatic improvement
* Space: O(b^m), as it keeps all nodes in memory
* Optimality: not optimal.
  + 1. **A\*algorithm**

A\* is the most renowned form of best first search. The evaluation is done by combining g(n) i.e. the cost to reach the node and h(n) i.e. the cost to reach the goal from the node:

F(n)=h(n)+g(n)

Since g(n) gives the path cost from start node to node n, h(n) is considered as the estimated cost of the cheapest path from node to goal.

Algorithm:

Insert the root node into the queue

While the queue is not empty

      Dequeue the element with the highest priority

f(n)=g(n)+h(n)

     (If priorities are same, alphabetically smaller path is chosen)

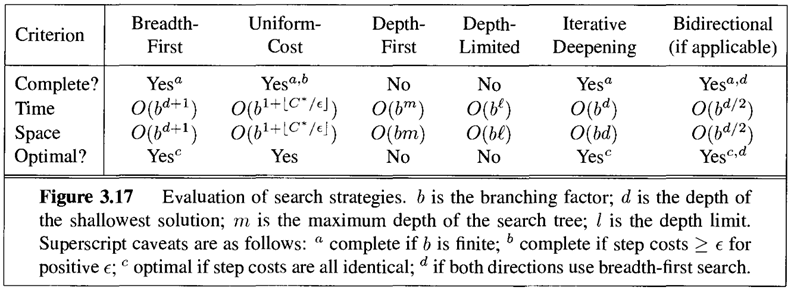
      If the path is ending in the goal state, print the path and exit

      Else

            Insert all the children of the dequeued element, with f(n) as the priority

Properties of A\* algorithm

* Complete: Yes, unless there are infinitely many nodes with f<=f(G)
* Time: O(log(h\*n))
* Space: O(b^m), as it keeps all nodes in memory
* Optimality: optimal.

****

1. **Proposed Framework**

The project includes an application which has been developed on Python platform and tkinter has been used for GUI. The algorithms which are being dealt with are depth first search, breadth first search, uniform cost search, depth limited search, iterative deepening depth first search, greedy best first search and A\* algorithms. There are two types of implementation accessible to the user, dynamic and static. In the dynamic implementation, the user is provided with various keywords such as depth, breadth, cost, heuristic values so that the user can describe his requirements regarding the real time scenario which the user is currently facing. For the demonstration purpose, a static implementation is provided with the graphical structures and the tracing of the algorithm with the time taken to execute.

Based on the input given by the user, the application developed links to the corresponding search algorithm. Upon the selection of the algorithm the application processes the problem and provides the user with the optimal solution which includes the various complexities.

1. **Conclusion**

The application helps the user effectively choose an algorithm which is appropriate and efficient according to the requirements. It also helps differentiate between the present informed and uninformed algorithms and provides the algorithm which is most efficient.

1. **References**

1)B. Liu, “Intelligent route finding: Combining knowledge, case

And an efficient search algorithm,” in Proc. 12thEuropean,Conference

on Artificial Intelligence (ECAI 1996)

2)A. K. Goel, K. S. Ali, M. W. Donnellan, A. G. de Silva

Garza, and T. J. Callantine, “Multistrategy adaptive path

planning,

3)R. V. Helgason, J. L. Kennington, and B. D. Stewart,

“The one-to-one shortest-path problems: an empirical

analysis with the two-tree Dijkstra algorithm”

4)Artificial Intelligence , A Modern Approach 2nd Edition : Stuart Russell, Peter Norvig

5)G. Khachaturov, “An approach to trip- and route-

planning problems,” Cybernetics and Systems

1. [↑](#footnote-ref-1)