Searching and Optimization Techniques in Artificial Intelligence: A Comparative Study & Complexity Analysis

Ashwani Chandel, Manu Sood

Abstract— **Solving a problem mean looking for a solution, which is best among others. Finding a solution to a problem in Computer Science and Artificial Intelligence is often thought as a process of search through the space of possible solutions. On the other hand in Engineering and Mathematics it is thought as a process of optimization i.e. to find a best solution or an optimal solution for a problem. We categorize the different AI search and optimization techniques in a tabular form on the basis of their merits and demerits to make it easy to choose a technique for a particular problem.**

***Index Terms*—artificial intelligence, complexity, optimization, searching**

I. Introduction

Search algorithms are used for a multitude of AI tasks, one of them being the path finding. The area of search in AI is very much connected to real life problem solving. AI has investigated search methods that allow one to solve path planning problems in large domains. Having formulated problems, we need to solve them and it is done by searching through the state space during this process. Most of the researches on search methods have studied how to solve one-shot path-planning problems. Search is mostly a repetitive process, therefore, many AI systems re-plan from scratch to solve the path planning problem independently.
Optimization is one of the most important tasks the engineers have to carry out. The engineers are required to design new, better, more efficient, less complex and less expensive systems as well as to devise plans and procedures for the improved operation of existing systems in both industrial and the scientific world.

II. Problem Statement

There are many search and optimization algorithms in Artificial Intelligence, the popular ones being Uninformed Search, Heuristic Search and Evolutionary algorithms etc. Although a lot of research work is done on individual algorithm but not enough research is done on the comparison of these algorithms under different problems. This is essential considering the fact that these algorithms behave differently or perform differently for different problems. By analyzing how an algorithm performs under a certain problem, the shortcomings of the algorithm can be found out and more research could be done on removing those shortcomings. Further, this research work also helps in choosing an algorithm best suited to a particular problem by finding out the pros and cons of the tested algorithm.

Ashwani Chandel, *Department of Computer Science, Himachal Pradesh University, Shimla, India*
Manu Sood, *Professor, Department of Computer Science, Himachal Pradesh University, Shimla, India*

III. AI Solution Search Techniques & Algorithms

Search problems can be classified by the amount of information that is available to the search process. Such information might relate to the problem space as a whole or to only some states. It may be available a priori or only after a node has been expanded. On such basis we can categorize different search techniques as shown in Fig 1. [3].

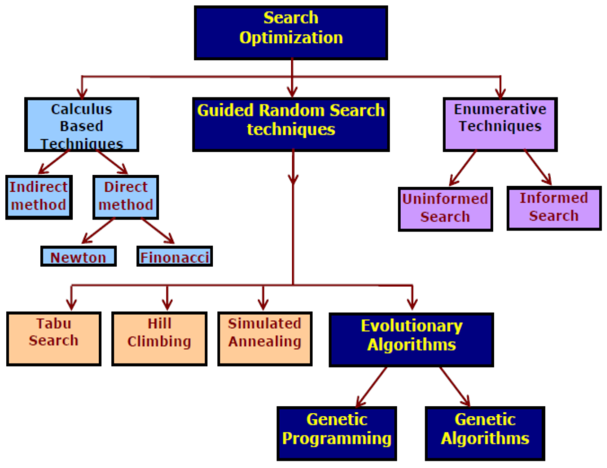

Fig.1 Classification of Different Search Techniques [4]

Uninformed search algorithms for problem solving are a central topic of classical computer science by Horowitz and Sahni 1978, Operations research by Drefus by 1969.Uninformed search strategies use only that information which is available in the problem definition. Followings are important types of uniformed search strategies:-

A. Brute force or Blind search methods

Brute force or blind search is a uninformed exploration of the search space and it does not explicitly take into account either planning efficiency or execution efficiency. Blind search is also called Brute Force Search. It is the search which has no information about its domain [3]. The only thing that a blind search can do is to differentiate between a non goal state and a goal state. These methods do not need domain knowledge but they are less efficient in result. All brute force search algorithms must take O(bd) time and use O(d) space [4]. The most important brute force techniques are breadth first, depth first, uniform cost, depth first iterative deepening and bidirectional search. Uninformed strategies don't use any information about how close a node might be to a goal. They differ in the order that the nodes are expanded.

B. Breadth First Search (BFS)

Breadth first search is a general technique of traversing a graph [4]. Breath first search may use more memory but will always find the shortest path first [3]. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution, if one exists [4]. The solution which is found is always the optimal solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level. Thus all expanded nodes are retained till the search is completed. It can be implemented most easily by maintaining the queue of nodes. The number of node at level d is (bd), the total number of nodes produced in the worst case is b+b2+b3+……. + bd which is O(bd) the asymptotic time complexity of breadth first search. [2]. Breadth first search is a complete algorithm with exponential time and space complexity.

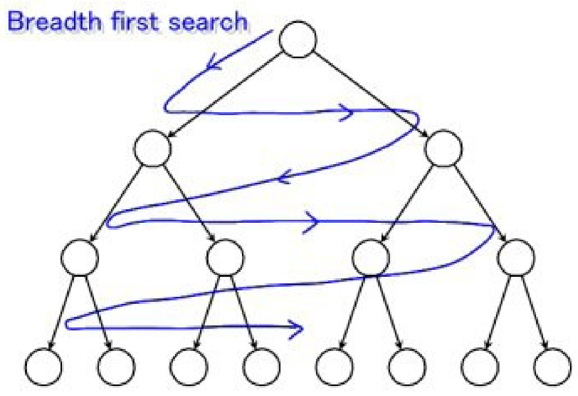

Fig.2 Breadth First Search Technique for a Tree Structure [5]

C. Depth First Search (DFS)

Depth first search is also important type of uniform or blind search. DFS visits all the vertices in the graph, this type of algorithm always chooses to go deeper into the graph [5]. After DFS visited all the reachable vertices from a particular source vertex it chooses one of the remaining undiscovered vertices and continues the search. DFS reminds the space limitation of breadth first search by always generating next a child of the deepest unexpanded nodded One interesting property of depth first search is that, the discover and finish time of each vertex form a parenthesis structure. If we use one open parenthesis when a vertex is finished then the result is properly nested set of parenthesis [6].

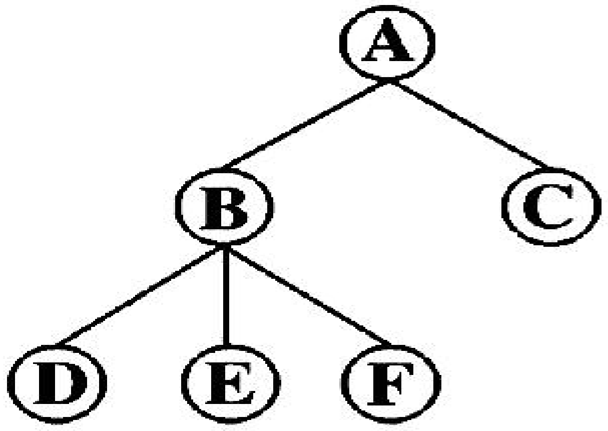

Fig.3 Depth First Search Technique for a Tree Structure [6]

The disadvantage of depth first search is that, it may not terminate on an infinite tree, but simply go down the left most paths forever. Even a finite graph can generate an infinite tree [4].

D. DFS Iterative Deepening

This kind of search performs depth first search to bounded depth d, starting d=1, and on each iteration it increases by 1 [2]. Depth First Iterative Deepening (DFID) is asymptotically optimal in terms of time and space among all brute force search algorithms that finds optimal solution on a tree. It was created as an attempt to combine the ability of BFS to always find an optimal solution. With the lower memory overhead of the DFS, we can say it combines the best features of breadth first and depth first search [5]. It performs the DFS search to depth one, then starts over, executing a complete DFS to depth two, and continues to run depth first searches to successfully greater depths until a solution is found. This algorithm is equivalent to common backtracking algorithm in that a path is expanded until a solution is found. DFID does better because other nodes at depth d are not expanded [5]. It never generates a node until all shallower nodes have been generated. The first solution found by DFID is quite guaranteed to be along the shortest path and depth is increased one by one. Its main function is that it returns a solution or failure. This search is faster than BFS because latter also generates nodes at depth d+1 even if the solution is at depth d. It is liked often because it is effective compromise between two other methods of search [4]. It terminates if there is a solution. It can produce the same solution as produced by depth first search produces but it does not use the same memory. Its main properties are that it is memory efficient and always find best solution if one exists.

E. Greedy Search

This algorithm uses an approach which is quite similar to the best first search algorithm. It is a simple best first search which reduces the estimated cost to reach the goal. Basically it takes the closest node the goal state and continues its searching from there. It expands the node that appears to be closest to the goal [3]. This search starts with the initial vertex and makes very single possible change then looks at the change it made to the score. This search then applies the change till the greatest improvement. The search continues until no further improvement can be made. The Greedy Search never makes a lateral or uphill move. It uses minimal estimated cost h(n) to the goal state as measure which decreases the search time but the algorithm is neither complete nor optimal. The main advantage of this search is that it is simple and finds solution quickly and as far as its disadvantages are concerned it is not optimal, susceptible to false start and the time complexity $O(b^m)$ is same for space complexity [3].

F. Bidirectional Search

The idea behind bidirectional search is to run two searches same time, one forward from the initial state and other backward from the goal state, stopping when the two searches meet in the middle [4]. Bidirectional search is implemented by having one or both of the searches check each node before it is expanded to see if it is in the fringe of other search tree, if it is so, a solution has been found. For example if a problem has solution depth d=6, and each direction runs breadth-first search one node at a time then in the worst case the two searches meet when each has expanded all but one of the nodes at depth 3. For b=10, it means a total of 22,200 node generation, compared with 11,111,100 for a standard breadth-first search. This algorithm is complete and optimal, if both searches are breadth first, other combinations may sacrifice completeness, optimally or both [4]. The most difficult case for bidirectional search is when the goal test gives only an implicit description of some possibly large set of goal states.

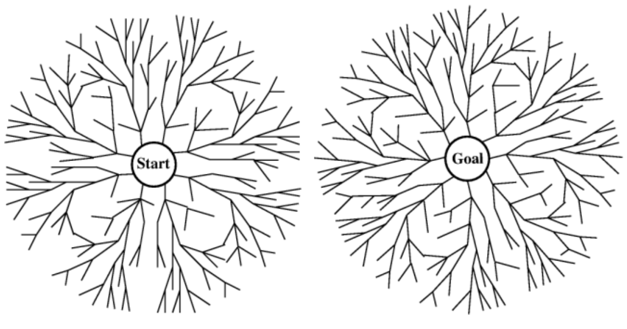

Fig.4 Bidirectional Search [5]

Other than Uninformed search techniques, Heuristic is a problem specific knowledge that decreases expected search efforts. It is a technique which works sometimes but not always. Heuristic search algorithms use information about the problem to help directing the path through the search space. These searches use some functions that estimate the cost from the current state to the goal presuming that such function is efficient. Generally heuristic incorporates domain knowledge to improve efficiency over blind search [6]. In AI heuristic has a general meaning and also a more specialized technical meaning. Generally a term heuristic is used for any advice that is effective but is not guaranteed to work in every case [5].

G. A Search*

A* is a cornerstone name of many AI systems and has been used since it was developed in 1968[1] by Peter Hart, Nils Nilsson and Betram Rapahel. It is combination of Dijkstra's algorithm and best first search. It can be used to solve many kinds of problems. A* search finds the shortest path through a search space to goal state using heuristic function. This technique finds minimal cost solutions and is also directed to a goal state called A* search. The A* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A*requires a heuristic function to evaluate the cost path that passes through the particular state [2]. It is very good search method but with complexity problems. This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It is defined by the following formula:-

$$f(n)= g(n)+h(n) \ [2]$$

Where g(n) is the cost of the path from the start state to node n and h(n) is the cost of path from node n to the goal state. The speed of execution of A* search is highly dependent on the accuracy of the heuristic algorithm that is used to compute h(n). A* search is both complete and optimal. Thus if we are trying to find the cheapest solution a reasonable thing to try first is the node with the lowest value of g(n)+h(n). It turns out that this strategy is more than just reasonable which provides that the heuristic function h(n).

H. Hill Climbing Search

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value, which is uphill. It stops when it reaches a "peak" where no neighbour has a higher value. The hill climbing comes from that idea that if you trying to find the top of the hill and you go up direction from where ever you are. The question that remains is whether this hill is indeed the highest hill possible. Unfortunately, without further extensive exploration, that question cannot be answered [7]. This technique works but as it uses local information so it can be fooled. The algorithm does not maintain a search tree, so the current node data structure need only record the state and its objective function value.

In this algorithm only a local state is considered when making a decision of which node is to expand next? When a node is entered all of its successor nodes have a heuristic function applied to them. The successor node with the most desirable result is chosen for traversal. Hill climbing sometimes called greedy local search because it catches a good neighbour state without thinking ahead about where to go next. Hill climbing often makes very rapid progress towards a solution because it is usually quite easy to improve a bad state. Hill climbing is best suited to the problems, where the heuristic gradually improve the closer it gets to the solution. It works badly, where there are sharp drop-offs. It assumes that local improvement will lead to global improvement. There are some reasons by which hill climbing often gets stuck which are stated below.

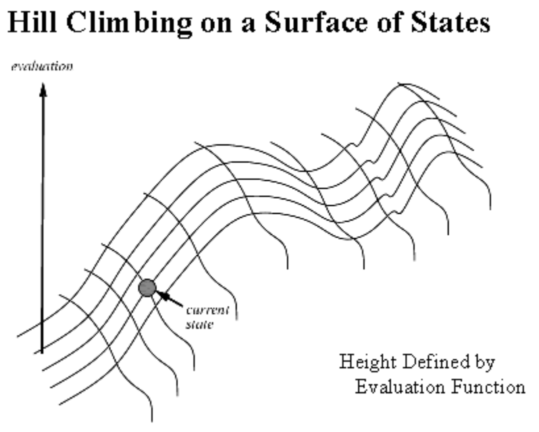
Hill Climbing on a Surface of States
evaluation
current state
Height Defined by Evaluation Function

Fig.5 Hill climbing search [8]

Local Maxima: A local maximum is the peak that is higher than each of its neighbouring states, but lower than the global maximum. Hill climbing algorithms that reach the vicinity of local maximum will be drawn upwards towards the peak, but then will be stuck with nowhere else to go [4].

Ridges: Steps of East, North, South and West may go down but the step to North West may go up. Ridges result in a sequence of local maxima that is very difficult for greedy algorithm to navigate [4].

Plateaus: The space has a broad flat area that gives the search algorithm no direction (random walk).

Many variants of hill climbing have also been invented which are described below [4]:

Stochastic Hill climbing: This variant chooses at random from among the uphill moves and the probability of selection can vary with the steepness of the uphill move. This usually converges more slowly than steepest ascent but in some state landscapes it finds better solution.

First choice Hill climbing : First choice Hill climbing variant implements stochastic hill climbing by generating successors randomly until one is generated that is better than current state. This is a good strategy when a state has thousands of successors.

Random Restart Hill Climbing: This Variant adopts the well known adage, if at first you don't succeed try again and again. It conducts a series of hill climbing searches from randomly generated initial states, stopping when a goal is found.

I. Simulated Annealing

In the early 1980's, Kirkpatrick, Gelatt & Vecchi (1982, 1983) and independently Cerny,
In 1985 introduced the concept of the physical annealing process in combinatorial optimization problem. The reason originates from the analogy between the solid annealing process and the problem of solving large scale combinatorial optimization problem [7]. Simulated annealing exploits an analogy between the way in which metal cools and freezes into a minimum energy, crystalline structure and the search for a minimum in a more general system. Simulated annealing is a probabilistic search algorithm and can avoid becoming trapped at local minima. Simulated annealing uses a control parameter T, which by analogy with the original application is known as the system temperature. It escapes local maxima by allowing some bad moves but gradually decrease their frequency.
Properties
If T decreases slowly enough, then simulated annealing search will find a local optimum with probability approaching. It is also widely used in VLSI layout, airline scheduling etc.

J. Generate and Test search

This is the simplified form which contains the following steps:

- It generates a possible solution
- Compares the possible solution to the goal state.
- If the solution is found it returns the success otherwise it again goes to first stage.

These are brute force algorithms that simply generate a possible solution and test to see if it is
Correct, if the solution is not correct then they repeat.

Advantages:- The main benefit of this algorithm is that it is easy to implement but its time complexity is higher than other search algorithms [7]. It takes very long time before the solution is found. This algorithm is improved to hill climbing algorithm. In such algorithms heuristic function is used to estimate the distance from the goal state. Thus only that solution is generated that will minimize the distance.

Disadvantages:- Generate and test approach is not very efficient because it also generates many wrong assignments of values of variables which are rejected in the testing phase. Furthermore the generator leaves out the conflicting instantiations and it generates other assignments independently of the conflict. Visibly one can get far better efficiency if the validity of the constraint is tested as soon as its respective variables are instantiated [7].

K. Back Tracking (BT)

A variant of Depth First Search is called Back Tracking search, which uses still less memory. In this search only one successor is generated at a time rather than all successors. Each partially expanded node remembers which successor to generate next. In this way only O(m) memory is needed rather than O(b^m). It is the most common algorithm for solving constraint satisfaction problem (CSP).

There is a major disadvantage of the standard backtracking scheme which is Thrashing. It occurs because the standard BT algorithm does not identify the main reason of the conflict or problem i.e. conflicting variables. That is why search in different parts of the space keeps failing for the same reason. Intelligent back tracking can reduce or finish thrashing. It is done by the scheme on which backtracking is done directly to the variable that cause the failure [8].

L. Best First Search

Best first search is an instance of the general tree or graph search algorithms in which a node is selected for expansion based on evaluation function f (n) [8]. Traditionally the node with the lowest evaluation is selected for the expansion because the evaluation measures distance to the goal. Best first search can be implemented with in general search framework via a priority queue, a data structure that will maintain the fringe in ascending order of f values. This search algorithm serves as combination of Depth First Search and Breadth First Search algorithms. BFS algorithm is often referred to as greedy algorithm because this algorithm quickly attacks the most, desirable path as soon as its heuristic weight becomes the most desirable [3]. There is a whole family of BFS algorithms with different evaluation functions. A key component of these algorithms is a heuristic function denoted h (n):
h(n) = estimated cost of the cheapest path from node n to a goal node [8].
The main steps of this search algorithm are: first add the initial node (starting point) to the queue and secondly it compares the front node to the goal state, if they match then the solution is found. If they don't match then expand the front node by adding all the nodes from the links. If all the nodes in the queue are expanded then the goal state is not found i.e. there is no solution and it stops. Apply the heuristic function to evaluate and reorder the nodes in the queue [9].

M. Branch and Bound

The branch and bound method was first used for parsimony by Hendy and Penny [8]. In 1975 La Blanc presented a branch and bound algorithm solution methodology for the discrete equilibrium transportation network design problem [9]. These search methods basically rely on that idea that we can divide our choice into sets using same domain knowledge and ignore a set when we can determine that the optimal element can't be in it. In 1991 Chen proposed a branch and bound with a stochastic incremental traffic assignment approach for the single class network design problem. It is an algorithmic technique which finds the optimal solution by keeping the best solution found so far. If partial solution can't be improved to its best, it is abandoned. By this method the number of nodes which are explored can also be reduced. It also deals with the optimization problems over a search that can be presented as the leaves of search tree. The usual technique for eliminating the sub trees from the search tree is called pruning. The load balancing aspects for branch and bound algorithms make it parallelization difficult. The primary difficulty being that usual assumption requires no priori information about the likely location of the search target.

N. Means End Analysis

Mean End Analysis is also an important kind of search algorithm and it is used in AI applications when a complex search is needed to be done. It is a different approach to find the solution and they are a common form of heuristic algorithm. Early implementations included the general problem solver (GPS). Now a day Means-End analysis is still used to create effective searches in the field of distributed computed Artificial Intelligence. It also focuses the search on actions which decrease the distance between current and target.
There are three main kind of goals used in mean end analysis search algorithm which are
1- Transform a state into a set of states
2- Decrease a distance possessed by a state
3-Apply an operator to the state to reduce the difference.

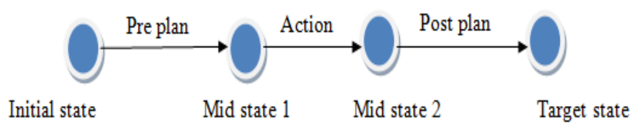

Fig.6 Means End Analysis States [9]

IV. Comparison of Different Search Algorithms

The output of problem solving algorithm is either failure or solution. Some algorithms might get stuck in an infinite loop and never return an output. When determining which search algorithm is appropriate for a problem space, it is necessary to derive and compare general attributes of each algorithm [3] .We will evaluate algorithm's performance in four ways

A. Completeness: - Is that algorithm guaranteed to find a solution when there is one? This column is a Boolean indicator of whether or not the search algorithm is exhaustive.
B. Optimality: - Does that strategy find the optimal solution? This column indicates that whether or not the solution found will always be the optimal solution.
C. Time Complexity: - How long does it take to find a solution? It is the order of complexity search time used by algorithm expressed as a function.
D. Space Complexity: - How much memory is needed to perform the search? This column is the order of complexity memory requirements of algorithm also expressed as a function.

The comparison among different search algorithms by these factors is shown by the table 1:

TABLE 1: Comparison of different search algorithms

Algorithm	**Time**	**Memory**	**Complete**	**Optimal**
Breadth First	$O(b^d)$	$O(b^d)$	Yes	Yes
Depth First	$O(b^d)$	$O(d)$	No	No
DF iterative deepening	$O(bd)$	$O(d)$	Yes	Yes
Bidirectional	$O(b^{d/2})$	$O(b^{d/2})$	Yes	Yes
Hill climbing	$O(b^d)$	$O(1)$- $O(b^d)$	No	No
Best First	$O(b^d)$	$O(b^d)$	Yes	No
A*	$O(b^d)$	$O(b^d)$	Yes	Yes
Beam Search	$O(n^d)$	$O(n^d)$	No	No
Means End	$O(b^d)$	$O(b^d)$	No	No

Where
d =depth of solution with in search tree
b = branching factor of search tree
n =subset of b for which algorithm will actually process.

In table 1, the attributes create a basis for decision making. Each of the algorithms discussed contains weak and strong attributes.

V. CONCLUSION

It is the function of the problem space to weight the trade-offs between the algorithms and determines which algorithm provides the best solution [9]. It can be seen from the table that the time estimate from all the searches are similar. The three exceptions are the Bidirectional, Beam and Generate and Test searches. The main reason that the Bidirectional search has a lesser time estimate is because it is simultaneously working from both ends of the problem looking for a common intermediate node. The Beam search has a time estimate of $O(n^d)$ as opposed to the more common $O(b^d)$. It is because the Beam Search is modified A* Search that examines on the best n branches at any node. It speeds up processing, but at the cost of assuming that a suboptimal node will never need to be travelled to reach the goal state. If that is the case the solution to the search will never be found. The memory requirement of the search algorithms are more distributed than the time estimates. In many cases a search algorithm will approach a problem Breadth First or Depth First.

REFERENCES

[1]: R. Saunders, "Lecture Notes on Introduction to Artificial Intelligence for Games",2006, http://www.soi.city.ac.uk/ ~rob/Lecture09-8up.pdf.
[2]: S.J. Kelly, "Article on Applying Artificial Intelligence Search Algorithms and Neural Networks for Games", http://www.generation5.org/content/2003/KellyMiniPaper.asp, 2003.
[3]: D.W. Patterson, " Introduction to Artificial Intelligence and Expert Systems", PHI Learning Private Limited, 2009.
[4]: S. Russel and P.Norvig," Artificial Intelligence a Modern Approach, A book on Artificial Intelligence and Algorithms", 2006.
[5]: R.E. Korf," Scientific Paper on Artificial Intelligence Search Algorithms", University of California Los Angeles, June 1999.
[6]: P.O. Doyle," Definition on AI Search Algorithms and Techniques", http://www.cs.dartmouth.edu/7Ebrd/Teaching/AI/ Lectures/Summaries/Search.html, May 2006.
[7]: A. Hertz," Lecture Notes on AI and Search Algorithms",http://www.csalbpc3.massey.ac.nz/notes/ 59302/103.html, May 2006.
[8]: J. Betali," Lecture Notes on Cognitive Science and Search Algorithms, University of California Sandiago, Nov 16, 1999.
[9]: T.A. Assaleh," Article on Intelligent Search Algorithm", http://www.cosc.brocku.ca/~cspress/Helloworld/1999/02-feb/search_algorithms.html, Brooke University Canada, 1999.

Dr. Manu Sood is a Professor in the Department of Computer Science, Himachal Pradesh University Shimla. He holds a Bachelor's degree in engineering, M.Tech. (with a gold medal) and has done his PhD from Delhi University. He has been working in the Department of Computer Science, HPU, since 1993.He has a keen Research interest in the field of Software Engineering and related fields.

Er. Ashwani Chandel is a M.Tech student in the Department of Computer Science, Himachal Pradesh University, Shimla. He holds his B.Tech. in IT from the University Institute of Information Technology, HPU Shimla. His research area of interest is Artificial Intelligence.