Heuristic Search

Heuristic search is an AI search technique that employs heuristic for its moves. Heuristic is a rule of thumb that probably leads to a solution. Heuristics play a major role in search strategies because of exponential nature of the most problems. Heuristics help to reduce the number of alternatives from an exponential number to a polynomial number. In Artificial Intelligence, heuristic search has a general meaning, and a more specialized technical meaning. In a general sense, the term heuristic is used for any advice that is often effective, but is not guaranteed to work in every case. Within the heuristic search architecture, however, the term heuristic usually refers to the special case of a heuristic evaluation function.

Generate and Test search algorithm

Generate-and-test search algorithm is a very simple algorithm that guarantees to find a solution if done systematically and there exists a solution.

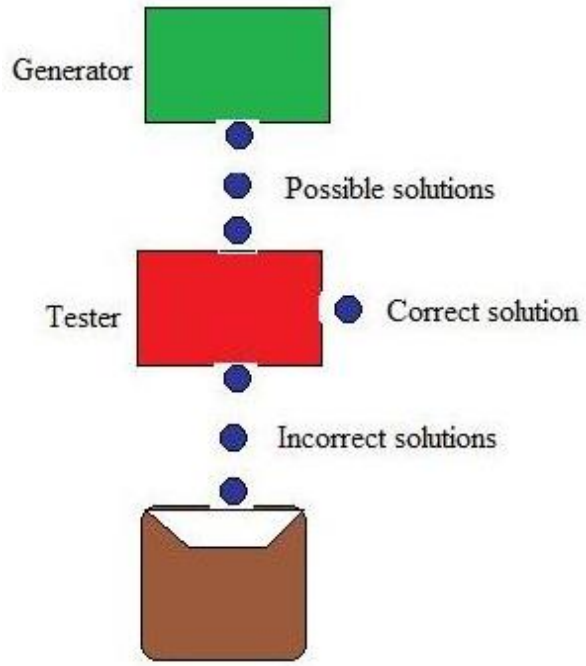
Algorithm: Generate-And-Test

1. Generate a possible solution.

2. Test to see if this is the expected solution.

3. If the solution has been found quit else go to step 1.

Potential solutions that need to be generated vary depending on the kinds of problems. For some problems the possible solutions may be particular points in the problem space and for some problems, paths from the start state.



Generate-and-test, like depth-first search, requires that complete solutions be generated for testing. In its most systematic form, it is only an exhaustive search of the problem space. Solutions can also be generated randomly but solution is not guaranteed. This approach is what is known as British Museum algorithm: finding an object in the British Museum by wandering randomly.

Hill Climbing

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value. It stops when it reaches a “peak” where no neighbour has higher value. This algorithm is considered to be one of the simplest procedures for implementing heuristic search. The hill climbing comes from that idea if you are trying to find the top of the hill and you go up direction from where ever you are. This heuristic combines the advantages of both depth first and breadth first searches into a single method.

The name hill climbing is derived from simulating the situation of a person climbing the hill. The person will try to move forward in the direction of at the top of the hill. His movement stops when it reaches at the peak of hill and no peak has higher value of heuristic function than this. Hill climbing uses knowledge about the local terrain, providing a very useful and effective heuristic for eliminating much of the unproductive search space. It is a branch by a local evaluation function.

The hill climbing is a variant of generate and test in which direction the search should proceed. At each point in the search path, a successor node that appears to reach for exploration.

Algorithm:
Step 1: Evaluate the starting state. If it is a goal state then stop and return success.
Step 2: Else, continue with the starting state as considering it as a current state.
Step 3: Continue step-4 until a solution is found i.e. until there are no new states left to be applied in the current state.

Step 4:
a. Select a state that has not been yet applied to the current state and apply it to produce a new state.

b. Procedure to evaluate a new state.

i. If the current state is a goal state, then stop and return success.

ii. If it is better than the current state, then make it current state and proceed further.

iii. If it is not better than the current state, then continue in the loop until a solution is found.

Step 5: Exit.
Advantages:

- Hill climbing technique is useful in job shop scheduling, automatic programming, designing, and vehicle routing and portfolio management.
- It is also helpful to solve pure optimization problems where the objective is to find the best according to the objective function.
- It requires much less conditions than other search techniques.

Disadvantages:
The question that remains on hill climbing search is whether this hill is the highest hill possible. Unfortunately without further extensive exploration, this question cannot be answered. This technique works but as it uses local information that's why it can be fooled. The algorithm doesn't maintain a search tree, so the current node data structure need only record the state and its objective function value. It assumes that local improvement will lead to global improvement.

Best First Search

Best first search is an instance of graph search algorithm in which a node is selected for expansion based o evaluation function f (n). Traditionally, the node which is the lowest evaluation is selected for the explanation because the evaluation measures distance to the goal. Best first search can be implemented within general search frame work via a priority queue, a data structure that will maintain the fringe in ascending order off values. This search algorithm serves as combination of depth first and breadth first search algorithm. Best first search algorithm is often referred greedy algorithm this is because they quickly attack the most desirable path as soon as its heuristic weight becomes the most desirable.

Concept:

Step 1: Traverse the root node

Step 2: Traverse any neighbour of the root node, that is maintaining a least distance from the root node and insert them in ascending order into the queue.

Step 3: Traverse any neighbour of neighbour of the root node, that is maintaining a least distance from the root node and insert them in ascending order into the queue

Step 4: This process will continue until we are getting the goal node

Algorithm:

Step 1: Place the starting node or root node into the queue.

Step 2: If the queue is empty, then stop and return failure.

Step 3: If the first element of the queue is our goal node, then stop and return success.

Step 4: Else, remove the first element from the queue. Expand it and compute the estimated goal distance for each child. Place the children in the queue in ascending order to the goal distance.

Step 5: Go to step-3

Step 6: Exit.

Graph Searching and the Generic Search Algorithm

Many AI problems can be cast as the problem of finding a path in a graph. A graph is made up of nodes and arcs. Arcs are ordered pairs of nodes that can have associated costs.

Suppose we have a set of nodes that we call "start nodes" and a set of nodes that we call "goal nodes", a solution is a path from a start node to a goal node.

Consider the following simple graph (this is a tree as there is at most one arc going into each node). The start nodes are colored grey, the goal nodes as are colored yellow, and the other nodes are not coloured.

AIspace Applet failed to load. Is Java enabled in your browser?

To find a solution, we need to search for a path. We use the generic searching algorithm. The frontier is a set of paths from a start node (we often identify the path with the node at the end of the path). The nodes at the end of the frontier are outlined in green or blue. Initially the frontier is the set of empty paths from start nodes. Intuitively the generic graph searching algorithm is:

- Repeat
    - select a path on the frontier. Let's call the path selected P.
    - if P is a path to a goal node, stop and return P,
    - remove P from the frontier
    - for each neighbor of the node at the end of P, extend P to that neighbour and add the extended path to the frontier
- Until the frontier is empty. When it is empty there are no more solutions.

To see how this works you can carry out the generic search algorithm selecting the nodes manually. The frontier is initially all coloured in green. You can click on a node on the frontier to select it. The node and the path to it turn red, and its neighbors (given in blue) are added to the frontier. The new frontier is then the nodes outlined in blue and green; the blue outlined nodes are the newly added nodes, and the green outlined nodes are the other node on the frontier. You can keep clicking on nodes till you find a solution. Then you can reset the search to try a different node ordering.

There are a number of features that should be noticed about this:

- For a finite graph without cycles, it will eventually find a solution no matter which order you select paths on the frontier.
- Some strategies for selecting paths from the frontier expand fewer nodes that other strategies.
- As part of the definition of the algorithm a solution is only found when a goal node is selected from the frontier, not when it is added.

Backtracking

The order in which variables are instantiated can have a large effect on the size of the search tree. The idea of variable ordering is to order the variables form most constrained to least constrained. For example, if a variable has only a single value remaining that is consistent with the previously instantiated variable, it should be assigned that value immediately. In general, the variables should be instantiated in increasing order of the size of their remaining domains. This can either be done statically at the beginning of the search or dynamically, reordering the remaining variables each time a variable is assigned a new value.

LISP (list processing)

LISP, an acronym for list processing, is a programming language that was designed for easy manipulation of data strings. Developed in 1959 by John McCarthy, it is a commonly used language for artificial intelligence (AI) programming. It is one of the oldest programming languages still in relatively wide use.

In LISP, all computation is expressed as a function of at least one object. Objects can be other functions, data items (such as constants or variables), or data structures. LISP's ability to compute with symbolic expressions rather than numbers makes it convenient for AI applications.

Natural Language Processing

NLP is a way for computers to analyze, understand, and derive meaning from human language in a smart and useful way. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation.

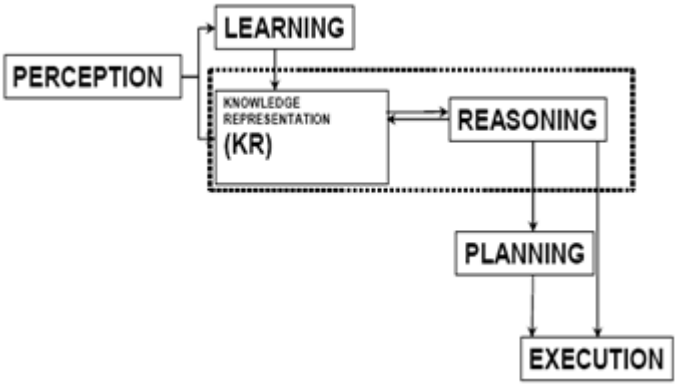
Knowledge Representation

Knowledge representation is probably, the most important ingredient for developing an AI. A representation is a layer between information accessible from outside world and high level thinking processes. Without knowledge representation it is impossible to identify what thinking processes are, mainly because representation itself is a substratum for a thought.

The subject of knowledge representation has been messaged for a couple of decades already. For many applications, specific domain knowledge is required. Instead of coding such knowledge into a system in a way that it can never be changed (hidden in the overall implementation), more flexible ways of representing knowledge and reasoning about it have been developed in the last 10 years.

The need of knowledge representation was felt as early as the idea to develop intelligent systems. With the hope that readers are well conversant with the fact by now, that intelligent requires possession of knowledge and that knowledge is acquired by us by various means and stored in the memory using some representation techniques. Putting in another way, knowledge representation is one of the many critical

aspects, which are required for making a computer behave intelligently. Knowledge representation refers to the data structures techniques and organizing notations that are used in AI. These include semantic networks, frames, logic, production rules and conceptual graphs.



Properties for knowledge Representation

The following properties should be possessed by a knowledge representation system.

a. Representational Adequacy: It is the ability to represent the required knowledge.

b. Inferential Adequacy: It is the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original.

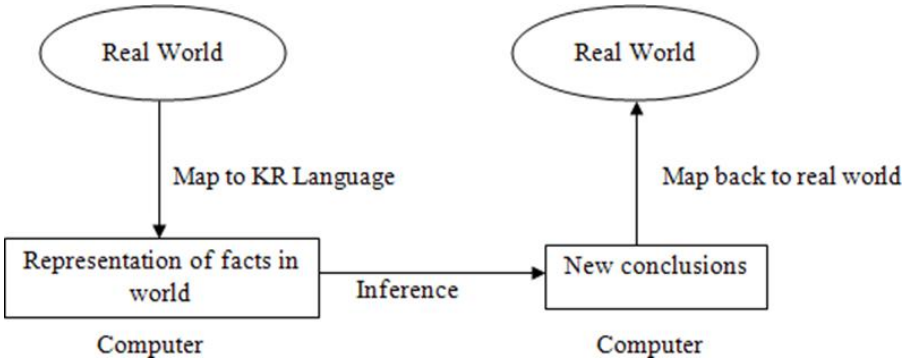
c. Inferential Efficiency: The ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides.

d. Acquisitional Efficiency: The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

Syntax and semantics for Knowledge Representation

Knowledge representation languages should have precise syntax and semantics. You must know exactly what an expression means in terms of objects in the real world. Suppose we have decided that “red 1” refers to a dark red colour, “car1” is my car, car2 is another. Syntax of language will tell you which of the following is legal: red1 (car1), red1 car1, car1 (red1), red1 (car1 & car2)?

Semantics of language tell you exactly what an expression means: for example, Pred (Arg) means that the property referred to by Pred applies to the object referred to by Arg. E.g., properly “dark red” applies to my car.



Types of Knowledge Representation

Knowledge can be represented in different ways. The structuring of knowledge and how designers might view it, as well as the type of structures used internally are considered. Different knowledge representation techniques are

a. Logic

b. Semantic Network

c. Frame

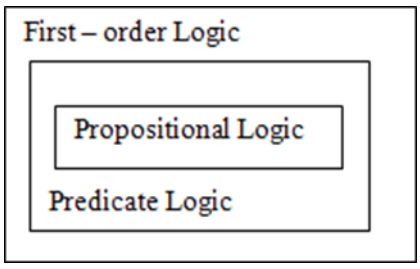
d. Conceptual Graphs

e. Conceptual Dependency

f. Script

Logic

A logic is a formal language, with precisely defined syntax and semantics, which supports sound inference. Different logics exist, which allow you to represent different kinds of things, and which allow more or less efficient inference. The logic may be different types like propositional logic, predicate logic, temporal logic, description logic etc. But representing something in logic may not be very natural and inferences may not be efficient.



Prepositional Logics

In propositional logic a complete sentence can be presented s an atomic proposition. and complex sentences can be created using AND, OR, and other operators.....these propositions has only true of false values and we can use truth tables to define them... like book is on the table....this is a single proposition... In predicate logic there are objects, properties, functions (relations) are involved.

In propositional logic, we use letters to symbolize entire propositions. Propositions are statements of the form "x is y" where x is a subject and y is a predicate. For example, "Socrates is a Man" is a proposition and might be represented in propositional logic as "S".

Predicate Logics

In predicate logic, we symbolize subject and predicate separately. Logicians often use lowercase letters to symbolize subjects (or objects) and uppercase letter to symbolize predicates. For example, Socrates is a subject and might be represented in predicate logic as "s" while "man" is a predicate and might be represented as "M". If so, "Socrates is a man" would be represented "Ms".

The important difference is that you can use predicate logic to say something about a set of objects. By introducing the universal quantifier ("∀"), the existential quantifier ("∃") and variables ("x", "y" or "z"), we can use predicate logic to represent thing like "Everything is green" as "∀Gx" or "Something is blue" as "∃Bx.

Resolution

Robinson in 1965 introduced the resolution principle, which can be directly applied to any set of clauses. The principal is "Given any two clauses A and B, if there is a literal P1 in A which has a complementary literal P2 in B, delete P1 & P2 from A and B and construct a disjunction of the remaining clauses. The clause so constructed is called resolvent of A and B."

For example, consider the following clauses A: P V Q V R

B: `p V Q V R C: `Q V R

Clause A has the literal P which is complementary to `P in B. Hence both of them deleted and a resolvent (disjunction of A and B after the complementary clauses are removed) is generated. That resolvent has again a literal Q whose negation is available in C. Hence resolving those two, one has the final resolvent.

A: P V Q V R (given in the problem) B: `p V Q V (given in the problem) D: Q V R (resolvent of A and B)

C: `Q V R (given in the problem) E: R (resolvent of C and D)