Intelligent Search Methods and Strategies

Search is inherent to the problem and methods of Artificial Intelligence (AI). This is because AI problems are intrinsically complex. Efforts to solve problems with computers which human can routinely innate cognitive abilities, pattern recognition, perception and experience, invariably must turn to considerations of search. All search methods essentially fall into one of two categories, exhaustive (blind) methods and heuristic or informed methods.

State Space Search

The state space search is a collection of several states with appropriate connections (links) between them. Any problem can be represented as such space search to be solved by applying some rules with technical strategy according to suitable intelligent search algorithm.

What we have just said, in order to provide a formal description of a problem, we must do the following:

1. Define a state space that contains all the possible configurations of the relevant objects (and perhaps some impossible ones). It is, of course, possible to define this space without explicitly enumerating all of the states it contains.
2. Specify one or more states within that space that describe possible situations from which the problem solving process may start. These states are called the initial states.
3. Specify one or more states that would be acceptable as solutions to the problem. These states are called goal states.
4. Specify a set of rules that describe the actions (operators) available. Doing this will require giving thought to the following issues:
    - What unstated assumptions are present in the informal problem description?
    - How general should the rules be?
    - How much of the work required to be solved the problem should be pre-computed and represented in the rules?

The problem can then be solved by using rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found. Thus the process of search is fundamental to the problem-solving process. The fact that search provides the basis for the process of problem solving does not, however, mean that other, more direct approaches cannot also be exploited. Whenever possible, they can be included as steps in the search by encoding those rules. Of course, for complex problems, more sophisticated computations will be needed. Search is a general mechanism that can be used when no more direct methods is known. At the same time, it provide the framework into which more direct methods for solving subparts of a problem can be embedded. For successfully design and implement search algorithms, a programmer must be able to analyze and predict their behavior. Questions that need to be answered include:

- Is the problem solver guaranteed to find a solution?
- Will the problem solver always terminate, or can it become caught in an infinite loop?
- When a solution is found, is it guaranteed to be optimal?
- What is the complexity of the search process in terms of time usage or memory usage, etc...?
- How can the interpreter most effectively reduce search complexity?
- How can an interpreter be designed to most effectively utilize a representation language?

To get a suitable answer for these questions search can be structured into three parts. A first part presents a set of definitions and concepts that lay the foundations for the search procedure into which induction is mapped. The second part presents an alternative approaches that have been taken to induction as a search procedure and finally the third part present the version space as a general methodology to implement induction as a search procedure. If the search procedure contains the principles of the above three requirement parts, then the search algorithm can give a guarantee to get an optimal solution for the current problem.

Search Techniques

Having formulated some problems, we now need to solve them. This is done by a search through the state space. The root of the search tree is a search node corresponding to the initial state. The first step is to test whether this is a goal state. Because this is not a goal state, we need to consider some other states. This is done by expanding the current state; that is, applying the successor function to the current state, thereby generating a new set of states. Now we must choose which of these possibilities to consider further. We continue choosing, testing and expanding either a solution is found or there are no more states to be expanded. The choice of which state to expand is determined by the search strategy. It is important to distinguish between the state space and the search tree. For the route finding problem, there are only ***N*** states in the state space, one for each city. But there are an infinite number of nodes. There are many ways to represent nodes, but we will assume that a node is a data structure with five components:

- **STATE:** the state in the state space to which the node corresponds**.**
- **PARENT-NODE:** the node in the search tree that generated this node.
- **ACTION**: the action that was applied to the parent to generate the node.
- **PATH-COST**: the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers.
- **DEPTH:** the number of steps along the path from the initial state.

As usual, we differentiate between two main families of search strategies: systematic search and local search. Systematic search visits each state that could be a solution, or skips only states that are shown to be dominated by others, so it is always able to find an optimal solution. Local search does not guarantee this behavior. When it terminates, after having exhausted resources (such as time available or a limit number of iterations), it reports the best solution found so far, but there is no guarantee that it is an optimal solution. To prove optimality, systematic algorithms are required, at the extra cost of longer running times with respect to local search. Systematic search algorithms often scale worse with problem size than local search algorithms.

Search Technique Types

Usually types of intelligent search are classified into two classes; **blind and heuristic search**. Blind search is a technique to find the goal without any additional information that help to infer the goal, with this type there is no any consideration with process time or memory capacity.

In the other side the heuristic search always has an evaluating function called heuristic function which guides and controls the behavior of the search algorithm to reach the goal with minimum cost, time and memory space. The following sections have details of each type of search with simple examples.

A. Blind Search

There many search strategies that come under the heading of blind search (also called uniformed search). The term means that they have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state. Thus blind search strategies have not any previous information about the goal nor do the simple paths lead to it. However blind search is not bad, since more problems or applications need it to be solved; in other words there are some problems give good solutions if they are solved by using depth or breadth first search. So, there are two types:

1. **Depth first search**

It always expands the deepest node in the current fringe of the search tree. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the fringe, so then the search "backs up" to the next shallowest node that still has unexplored successors.

```
1. Form a one element queue Q consisting of the root node.
2. Until the Q is empty or the goal has been reached, determine if the first element in the Q is
   the goal.
     a. If it is, do nothing.
     b. If it is not, remove the first element from the Q and add the first element's children, if any,
        to the FRONT of the Q.
3. If the goal is reached, success; else failure.
```

The depth-first algorithm is:

2. Breadth first search

It is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. In general all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

The Breadth -first algorithm is:

```
1. Form a one element queue Q consisting of the root node.
2. Until the Q is empty or the goal has been reached, determine if the first element in the Q is
   the goal.
     a. If it is, do nothing.
     b. If it is not, remove the first element from the Q and add the first element's children, if any,
        to the Back of the Q.
3. If the goal is reached, success; else failure.
```

B. <u>heuristic search</u>

In heuristic search, we generally use one or more heuristic functions to determine the better candidate states among a set of legal states that could be generated from a known state. The heuristic function, in other words, measures the fitness of the candidate states. The better the selection of the states, the fewer will be the number of intermediate states for reaching the goal. However; the most difficult task in heuristic search problems is the selection of the heuristic functions. One has to select them intuitively, so that in most cases hopefully it would be able to prune the search space correctly.

The heuristic search methods use an evaluation function to guide the search towards goal. Information about the state space will be used to guide the search. Knowledge is used in the queuing function to determine which node to expand next. This process uses an evaluation function:

Evaluation function f(n)

Cost: g(n)=C

Heuristic: h(n)=E (estimate of distance to goal)

where: g(n):- Measures the actual length of path from any state (n) to the start state. In other words, it is the actual cost from the initial node (start node) to node n (i.e. the cost finding of optimal path)

h(n):- It is a heuristic estimate of the distance from state (n) to the goal. In other words, it is the estimated cost of the optimal path from node n to the target node (destination node).

The evaluation function returns a number describing the desirability of expanding the node and uses these numbers to guide the search towards goal.

1. Hill Climbing search

The idea here is that, you don't keep the big list of states around you just keep track of the one state you are considering, and the path that got you there from the initial state. At every state you choose the state leads you closer to the goal according to the heuristic estimate cost of status g(n), and continue from there.

The name "Hill Climbing" comes from the idea that you are trying to find the top of a hill and you go in the direction that is up from wherever you are.

The Hill-climbing algorithm is:

```
1. Form a one element queue Q consisting of the root node.
2. Until the Q is empty or the goal has been reached, determine if the first element in the Q is
   the goal.
    a) If it is, do nothing.
    b) If it is not, remove the first element from the Q, sort it's children, if any, according to the
       estimating remaining distance value g(n), and add this sorted list to the FRONT of the Q.
3. If the goal is reached, success; else failure.
```

2. Best-first search

In best-first search expansion of nodes is resumed from the best open node visited so far, no matter where it is in the partially explored tree. It also you closes the goal according to the heuristic estimate cost of status g(n).

The best-first algorithm is:

```
1. Form a one element queue Q consisting of the root node.
2. Until the Q is empty or the goal has been reached, determine if the first element in the Q is the goal.
    a) If it is, do nothing.
    b) If it is not, remove the first element from the Q, add it's children to the Q, if any, and sort
       all the Q according to the estimating remaining distance value g(n).
3. If the goal is reached, success; else failure.
```

3. Branch and Bound search

This method considers the best path ``so far'' instead of the best successor. It stops only when it has discovered at least one goal and there are no unexplored paths of shorter length than the shortest root-goal path found. At every state you choose the state leads you closer to the goal according to the heuristic estimate (Estimate of Distance to goal) h(n), and continue from there.

```
1. Form a one element queue Q consisting of the root node.
2. Until the Q is empty or the goal has been reached, determine if the first element in the Q is the goal.
    a) If it is, do nothing.
    b) If it is not, remove the first element from the Q, add it's children to the Q, if any, and sort
       all the Q according to the estimating remaining distance to goal value h(n).
3. If the goal is reached, success; else failure.
```

The branch and bound algorithm is:

4. A* search

A-star search algorithm is a widely used graphic searching algorithm. It is also a highly efficient heuristic algorithm used in finding a variable or low cost path. It is considered as one of the best intelligent search algorithms that combines the merits of both best-first algorithm and branch and bound algorithm. A-star path searching algorithm uses the evaluation function (usually denoted f (n)) to guide and determine the order in which the search visits nodes in the tree. The evaluation function is given as: $f(n) = g(n) + h(n)$

where: **g(n)** is the actual cost from the initial node (start node) to node n (i.e. the cost finding of optimal path), h(n) is the estimated cost of the optimal path from node n to the target node (destination node), which depends on the heuristic information of the problem area.

The A* search algorithm is:

```
1. Form a queue Q of partial paths. Let the initial Q consist of the zero-cost, zero-step path
from the root to nowhere.
2. Until the Q is empty or the goal has been reached, determine if the first path in the Q
reaches the goal.
    2a. If it does, do nothing.
    2b. if it does not:
        2b1. Remove the first path from the Q.
        2b2. Form new paths from the removed one by extending one step.
        2b3. Add the new paths to the Q.
        2b4. Sort the Q according to the sum of actual cost so far + the lower-bound estimate of
                the cost remaining.
3. If the goal is reached, success; else failure.
```