SEARCH METHODS

1. Why do we need Search Methods in AI?

Search Methods are popping up everywhere in AI, but used mostly for problem solving. Search means also an exploration of possibilities. We need to search “intelligently” to find to best way or the shortest way.

2. The Search Methods are:

- Nets and Basic Search
  - Blind Methods
    - Depth-first Search
    - Breadth-first Search
    - Non-deterministic Search
  - Heuristically Informed Methods
    - Hill Climbing
    - Beam Search
- Nets and Optimal Search
  - The Best Path
    - The British Museum Procedure
    - Branch-and-Bound Search
  - Redundant Paths
    - Discrete Dynamic Programming
    - A* Procedure
- Trees and Adversarial Search
  - Algorithmic Methods
    - Minimax Procedure
    - Alpha-Beta Pruning
  - Heuristic Method
    - Progressive Deepening

3. Evaluating a Search:

Evaluating the performance of a search technique can be very complicated. In fact, the evaluation takes an important place in AI. There are two important measurements;

- How quickly the search finds the path
- How good the solution is

The importance of the measurement depends on the type of the problem. As a result, finding an optimal solution and finding a quick solution are different from each other.

For example, you want to find a path from a city to another. If you often travel between these cities, it is meaningful to find the optimal way and to spend worth a lot of search time. Otherwise, if you use this way only once, the quickest path is more useful.
4. Nets and Basic Search:

Blind Searches are methods where you know nothing about how well you are doing if you reach the goal. On the other hand, search efficiency may improve spectacularly is there is a way to order the choices so that the most promising are explored earlier. In many situations, you can make measurements to determine a reasonable ordering. Search Methods that take advantage of such measurements are called heuristically informed searches.

4.1. Blind Methods:

The purpose of these blind methods is finding one path. Suppose we want to find some path through a net of cities. The path begins at city S and ends at city G.

The most obvious way to find a solution is to look all possible paths. But we should discard paths that revisit any particular city to prevent from getting stuck in a loop. So it is arranged all possible paths from the start node in a search tree.

4.1.1. Depth-first Search:

The first thing in depth-first search to do is to go headlong to the bottom of the tree along the leftmost branches. If it is reached to the end node and it is not the goal node, back up the tree until to the nonleaf node and go down again to the other child node.

4.1.2. Breadth-first Search:

Breadth-first search checks all nodes on the founded level, before going next level.

4.1.3. Choosing the Right Method:

Depth-search is a good method, when all partial parts either reach dead ends or become complete paths after reasonable number of steps. However, if there are long paths, like infinitely long paths, this search method is not a good idea.

Breadth-first search works even in trees those are infinitely deep. On the other hand, breadth-first search is wasteful when all paths to lead the goal node at more or less the same depth.

4.1.4. Nondeterministic Search:

This search method moves randomly into the search tree. If there is no info about the problem or any calculated branching factor or long useless paths, the nondeterministic search is a middle way between breadth-first search and depth-first search.

4.2. Heuristically Informed Methods:

Search efficiency may improve spectacularly if there is a way to order the choices so that the most promising are explored earliest.

4.2.1. Hill Climbing Method:

To move through a tree of paths using hill climbing, you proceed as you would in depth-first search, except that you order your choices according to some heuristic measure of the remaining distance the goal.
The better heuristic measure is the better hill climbing will be relative to ordinary depth-first search. The hill climbing method chooses at its next step the node that appears to place it closest to the goal (that is, farthest away from the current position). More knowledge generally leads to reduced search time.

4.2.2. Beam Search:

Beam search is like breadth-first search in that it progresses level by level. Unlike breadth-first search, however, beam search moves downward only through the best \( w \) nodes at each level; the other nodes are ignored. Consequently, the number of nodes explored remains manageable, even if there is a great deal of branching and the search is deep. Whenever beam search is used, there are only \( w \) nodes under consideration at any depth, rather than the exponentially explosive number of nodes which you must cope whenever you use breadth-first search.

4.2.3. Search Alternatives Form a Procedure Family:

- Depth-first search is good when unproductive partial paths are never too long.
- Breadth-first search is good when the branching factor is never too large.
- Nondeterministic search is good when you are not sure whether depth-first search or breadth-first search would be better.
- Hill climbing is good when there is a natural measure of distance from each place to the goal and a good path is likely to be among the partial paths that appear to be good at each choice point.
- Beam search is good when there is a natural measure of goal distance and a good path is likely to be among the partial paths that appear to be good at all levels.

5. Nets and Optimal Search:

Nets and Optimal Search methods deal with search situations in which the cost of traversing a path is of primary importance.

5.1. The Best Path:

5.1.1. The British Museum Procedure:

One procedure for finding the shortest path through a net is to find all possible paths and to select the best one from them. This plodding procedure is known as the British Museum Procedure.

To find all possible paths, either a depth-first search or a breadth-first search will work with one modification: search continues until every solution is found. If the breadth and the depth of the tree are small as in the map-traversal example, then there is no problem.

Unfortunately, the size of search trees is often large, so making any procedure for finding all possible paths extremely unpalatable. Suppose that, the branching is completely uniform and the number of alternative branches at each node is \( b \) and the depth of the search tree is \( d \). Continuing this analysis leads to the conclusion, the number of paths can be \( b^d \). So we can say simply, with British Museum procedure we can find the optimal path, but the calculation for finding the all possible paths results expensively.

5.1.2. Branch-and-Bound Search:

One way to find optimal paths with less work is to use branch and bound search.
This method always keeps track of all partial paths contending for further consideration. The shortest one is extended one level, creating as many new partial paths as there are branches. Next, these new paths are considered, along with the remaining old ones: again the shortest is extended. This process repeats until the goal is reached along some path. Because the shortest path was always the one chosen for extension, the path first reaching the goal is likely to be the optimal path.

In some cases, branch-and-bound search is improved by estimates of remaining distance as well as distances accumulated.

\[ e(\text{total path}) = d(\text{already traveled}) + e(\text{distance remaining}) \]

5.2. Redundant Paths:

Let’s look at the traversal problem with a view toward weeding out redundant paths that destroy search efficiency.

5.2.1. Discrete Dynamic Programming:

The dynamic-programming principle: The best way through a particular, intermediate place is the best way to it from the starting place, followed by the best way from it to the goal. There is no need to look at any other paths to or from the intermediate place.

5.2.2. A* Procedure:

The A* procedure is branch and bound search, with an estimate of remaining distance, combined with the dynamic-programming principle. If the estimate of remaining distance is a lower-bound on the actual distance, then A* produces optimal solutions.

5.3. Several Search Procedures Find the Optimal Path:

- The British Museum procedure is only good when the search tree is small.
- Branch-and-bound search is good when the tree is big and bad paths turn distinctly bad quickly.
- Branch-and-bound search with a guess is good when there is a good lower-bound estimate of the distance remaining to the goal.
- Dynamic-programming is good when many paths converge on the same place.
- The A* procedure is good when both branch-and-bound search with a guess and dynamic-programming are good.

6. Trees and Adversarial Search:

Tree search algorithms are used in situations where one wants to choose the best path on a situation or game. Games like chess use these algorithms to determine the opponent’s movement and play to the best situation until a side has no possible movement.

6.1. Algorithmic Methods:

Game situations can be represented in game trees and those trees can be searched so as to make the most promising move. In the game tree, the nodes denote board configurations, and the branches indicate how one board configuration can be transformed into another by a single move. There is special twist in that the decisions are made by two adversaries who take turns making decisions.
The ply of a game tree, \( p \), is the number of levels of the tree, including the root level. If the depth of a tree is \( d \), then \( p = d + 1 \). In a game, each choice refers to a move.

6.1.1. **Minimax Procedure:**

The procedure by which the scoring information passes up the game tree is called the Minimax Procedure, because the score at each node is either the minimum or the maximum of the scores at the nodes immediately below.

The whole idea of minimaxing rests on the translation of board quality into a single, summarizing number, the static value. Also note that minimaxing can be expensive, because either the generation of paths or static value can require a lot of computation. Which costs more depends on how the move generator and static evaluator have been implemented.

6.1.2. **Alpha-Beta Pruning:**

The alpha-beta procedure prunes game trees. The alpha-beta principle is; if you have an idea that is surely bad, do not take time to see how truly awful it is. This procedure uses two parameters, traditionally called alpha and beta, to keep track of expectations.

In the special context of games, the alpha-beta principle dictates that, whenever you discover a fact about a given node, you should check what you know about ancestor nodes. It may be that no further work is sensible below the parent node. Also, it may be that the best that you can hope for at the parent node can be revised or determined exactly.

6.2. **Heuristic Methods:**

These methods are used to search game trees under time pressure so we can make reasonable move within allowed time limits.

6.2.1. **Progressive Deepening:**

In tournaments, players are required to make a certain number of moves within time limits enforced by a relentless clock.

This method analyzes each situation to depth 1, then to depth 2, then to depth 3, and so on until the amount of time set aside for the move is used up. This way, there is always a move choice ready to go. The choice is determined by the analysis at one level less deep then the analysis in progress when the time runs out.

**REFERENCES**

