CSC 831: Artificial Intelligence

Uninformed Search Algorithms and Hopfield Neural Network

EDAWARE PATRICIA EWOMA-ZINO
179074012

Uninformed Search Method

Uninformed search method means that the strategies 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. Uninformed search methods include breadth first search, depth first search, depth limited search and iterative deepening search.

Breadth First Search

It requires the root node being expanded first, then all the successors of the root node are expanded next, then their successors, and so on. Therefore, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

It is implemented using a FIFO queue. The breadth first search algorithm is both optimal and complete.

BFS Algorithm

```
string breadthFirstSearch(Node root, Node goal)
{
    std::queue<Node> Q;
    std::vector<Node> children;
    string path = "";
    Q.push(root);
    while(!Q.empty())
    {
        Node t = Q.front();
        path += t.getValue();
        cout << "The path: ";
        cout << path << endl;
        Q.pop();
        if(t == goal){
            return path;
        }
        children = t.getChildren();
        for (int i = 0; i < children.size(); ++i)
        {
            Q.push(children[i]);
```

```
        }
    }
    return path;
}
```

Depth First Search

The Depth-First Search algorithm is a technique for searching a graph that begins at the root node, and exhaustively searches each branch to its greatest depth before backtracking to previously unexplored branches. It is implemented using a FIFO queue (stack). The tree search for DFS is not complete and optimal.

DFS Algorithm

```
string depthFirstSearch(Node root, Node goal)
{
    std::stack<Node> Q;
    std::vector<Node> children;
    string path = "";
    Q.push(root);
    while(!Q.empty())
    {
        Node t = Q.top();
        path += t.getValue();
        cout << "The path: " << path;
        Q.pop();
        if(t == goal){
            return path;
        }
        else{
            cout << " Failure " << endl;
        }
        children = t.getChildren();
        std::reverse(children.begin(),children.end());
        for (int i = 0; i < children.size(); ++i){
            Q.push(children[i]);
        }
```

```
    }
    return path;
}
```

Depth Limited Search

Depth limited solves the problem of infinite state spaces in depth first search, by providing a limit to which the depth must not go beyond. However, it can be incomplete in the sense that the shallowest goal may be beyond the depth limit.

DLS Algorithm

```
string depthLimitedSearch(Node root, Node goal)
{
    std::stack<Node> Q;
    std::vector<Node> children;
    std::stack<int> stackDepth;
    string path = "";
    int depth = 0;
    int limit = 2;
    Q.push(root);
    stackDepth.push(depth);
    while(!Q.empty())
    {
        Node t = Q.top();
        path+= t.getValue();
        Q.pop();
        depth = stackDepth.top();
        stackDepth.pop();
        if(t==goal){
        return path;
        }

        if(depth<limit){
            children = t.getChildren();
            std::reverse(children.begin(), children.end());
            for(int i=0;i<children.size();i++){
```

```
                Q.push(children[i]);
                stackDepth.push(depth+1);
            }
        }
        else{
        cout<< "Goal node is not found within the depth"<<endl;
        break;
        }
    }
}
```

Hopfield

Hopfield is a recurrent neural network that has feedback loops from its outputs to its inputs. It is required to store a set of M fundamental memories, Y1;Y2;...;YM. After which it is tested to know whether it is capable of recalling the fundamental memories. Lastly, a probe vector (Incomplete or corrupted version of the fundamental memory) is presented to the network, to retrieve a stable state.

.