A n Alternate Way Of Implementing Heuristic Searching technique

Mr. Girish P Potdar
Associate Professor, Computer Engineering Department,
P.I.C.T, Pune, India
gppotdar@pict.edu, gppotdar@hotmail.com

Dr. R C Thool
Professor, Department of Information Technology,
SGGS IE&T, Nanded, India
rcthool@sggs.ac.in, rcthool@yahoo.com

ABSTRACT: This work was carried out to explore the use of different data structure for heuristic search algorithm which can result in better performance with respect to time. The commonly used data structure Generalized Link List (GLL) and proposed use of data structure Multi Leveled Link List (MLL) schemes are presented in this paper. This scheme is more intended for better utilization of system resources. Almost all the heuristic algorithms have to maintain the list, which are to be added, updated and processed. Here different lists are maintained and operated upon simultaneously. Maintaining these lists simultaneously is not only tedious but complex too. The use of MLL will reduce this complexity. The paper aims at presenting the method for effective use of system resources without compromising on the optimality of the algorithm.

Index Terms— GLL, MLL, Heuristic Search.

Introduction

Knowledge plays a very important role in almost all the human activities. Let it be the interaction, information gathering, or implementing the tasks. Hence the need for applying AI techniques in day-to-day life spreads across the global, so the need for study of these techniques, and extensive analysis. The different approaches need more attention. So the advanced implementation details to be given more importance classical problems like shortest path finding, graph partitioning algorithms make use of heuristic search. The A* algorithms are a heuristic best fit algorithm that cost estimates to improve efficiency. For all such applications the traditional method for given a problem space, start state, and the goal state, applied searching technique will generate the state space at the every level. This contributes to the search tree.

Heuristic Search

Heuristic is a strategy or rule of thumb which directly limits for solutions in relatively large problem spaces. All the methods begin from the start state generate the search tree; select the available optimal path till we get the goal. Heuristic means to discover. One can define heuristic search, as “the searching process which tries to locate the predefined state by applying heuristic.” [1] The program statements, which refer to the heuristic, are called “heuristic function”. One may identify the following conditions where in we may make use of heuristic such as:

- When an uncertainty in the state space is foresighted.
- When the problem have vast solution spaces.
- When the problem have additional requirements.

There are various heuristic searching techniques like: BEST FIRST SEARCH, A*, AO*, hill climbing, simulated annealing. [2], [7]

The AI applications that use the heuristic techniques build their own heuristic function within the application constraints. The existence of heuristic function is basically to evaluate the two costs, g and h’. Where in g refers to the cost from start state to the current state, and h’ refers to the evaluated cost from current state to the goal state. So the total evaluated cost from the heuristic function is the summation of the two costs g and h’. That is f’ = g + h’. This total evaluated cost refers to, how far the goal state is from the current state. The heuristic searching techniques like A* evaluate the heuristic cost using heuristic function [5],[ 7].

Existing Approach

Given the start state and the goal state, begin from start state. Make the start state the best state. Expand it, pick up the best state from the generated states so far, and continue to create the search tree till goal state is found, or till no states to expand. To manage this, various lists are created and updated. As the state is expanded, generated states are entered in the OPEN list. This is continuously updated and sorted. So the OPEN list is the ordered list, which maintains the states (nodes) according to their priority. The priority of the state is decided according to their heuristic costs. Expanded states are placed in the CLOSE list [4], [5].

The Method

The procedure followed in traditional approach initially keeps the OPEN list as empty. The start state and goal state are given input. Pick up the state from OPEN list and is checked whether the state is already generated or goal state and update the path if necessary. Then it generates the intermediate states by making update in OPEN list and CLOSE list. The entire process is
terminated once we reach to goal state or when no state is available for expansion in OPEN list.

Let us consider the simple example. As shown in the figure 1, is simple graph structure, consist of ‘n’ nodes (A-E). In case of any AI application every node represents a state, which is evaluated for a value. This evaluated value is referred as the “Heuristic cost”. Decision of the state to be expanded next depends upon the heuristic cost of the state (i.e., how nearer is the heuristic cost of the current state from the goal state is) [2],[6].

![Simple Graph Structure](image)

Fig. 1. Example of a simple graph structure

The states in the OPEN list are sorted according to their heuristic costs. Here a state with the higher Heuristic cost will be considered as the best node for expansion. Initially OPEN list is empty. ‘A’ is being the start state, initialize OPEN list to ‘A’, ‘A’ being the only node for expansion. As we expand A, B, and C are its generated nodes (generated states) or successors. Enter B and C in the OPEN list, place ‘A’ in the CLOSE list and sort the OPEN list. Again pickup B for expansion, repeat the same procedure as shown below till the goal state is found or the OPEN list is empty. Applying the method mentioned above, we get:

<table>
<thead>
<tr>
<th>OPEN</th>
<th>CLOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>empty</td>
</tr>
<tr>
<td>A</td>
<td>empty</td>
</tr>
<tr>
<td>B C</td>
<td>A</td>
</tr>
<tr>
<td>E C D</td>
<td>AB</td>
</tr>
<tr>
<td>F C D</td>
<td></td>
</tr>
</tbody>
</table>

CLOSE : A B E
F = goal state,

We can see the two lists (OPEN and CLOSE) are updated continuously. The parent successor list has to be maintained along with these lists. It is a tedious job to maintain these lists for the applications that generate the large search tree. One can easily avoid creation of CLOSE list by maintaining the multilinked list or multiple linked lists.

**Multi Linked List Approach** [8]

I. Multilinked List is one of the dynamic memory allocation methods. MLL maintains the parent list and successor list. Every parent node maintains its list of successors. Again all these parent nodes are linked together. As it is clear from the figure 2 parent list binds all the expanded nodes together. Parent list can be referred as the CLOSE list and the successor list can be considered as the generated nodes (states). A typical Multiple Linked List organization is as given in the figure 2.

![Multi Linked List Organization](image)

Fig. 2. Multi linked List Organization

Here P1 and P2 are the parent nodes. It is clear from the above figure that they are the expanded nodes, as they linked with their respective successors, S1, S2 and S3, S4. At present S1, S2, S3 and S4 are the generated nodes. Now let us apply the MLL approach on the same algorithm. We maintain the single list ‘gen’ which is like the OPEN list. One can conclude that the parent link list is like the CLOSE list. So to refer to the expanded nodes one needs to refer to the parent link list.

**Explanation**

Gen is an ordered list that maintains the generated nodes. The successors are generated and arranged in ordered list gen. A best node is chosen for the expansion from gen. Update MLL to hold the proper present parent successor list structure.

**Proposed Method**

Let us discuss the same procedure as that of previous one, but using the MLL and the gen list. Apply it on the same sample example. Here node refers to the state as per the AI application.
This method initializes MLL to start state, where in start state and goal state are given as input. Create an ordered list which will be initialized to start state. Expand the start state, update MLL for the parent successors link and now update and sort the gen list. Pickup the next node for expansion from gen list after ensuring that the node is not expanded so far. Update MLL for proper parent successor linked list structure.

Let us apply the same algorithm on the sample example given in the Figure 1.

**gen**: A  
**MLL**: Start  
Null  

**gen**: B C  
**MLL**: A  
B  
C  

**gen**: E D C  
**MLL**: A  
B  
C  

**gen**: F D C  
**MLL**: A  
B  
C  
D  
E  

Initialize MLL to NULL and gen to A, as A being the start state. Pickup ‘A’ for expansion. Initialize MLL to A and generate its successors B and C. Update and sort the gen list, now B is the next possible best node. Expand B, D and E are its successors. Update the gen list and the MLL. Attach B in the parent D list, D and E in the successor list. Sort gen for the next best node that is the node ‘E’. Pick up E or expansion generating the goal node F. A path traversed to reach to the goal state can be recovered from MLL. The path here is A->B->E->F.>

In the existing approach OPEN and CLOSE are referred, along with parent successor list. In the current approach only one list gen has to be referred along with MLL, since MLL itself maintains the parent successor list. To refer the expanded nodes use the parent list. As MLL maintains the parent successor list one can easily update for better path using MLL. Gen list can also be used as the reference list.

**Conclusion**

The basic organization of MLL will eliminate the use of two separate lists being maintained and referred. This will definitely result in reduced time and space complexity. The problems such traveling sales person, graph partitioning and others where heuristic search is used, may use the MLL.

**REFERENCES**