COMPARISON OF VARIOUS HEURISTIC SEARCH TECHNIQUES FOR FINDING SHORTEST PATH

Mr. Girish P Potdar¹, Dr. R C Thool².

¹Associate Professor, Computer Engineering Department, P.I.C.T, Pune,  
²Professor, Department of Information Technology, SGGS IE&T, Nanded,

ABSTRACT

Couple of decades back, there was a tremendous development in the field of algorithms, which were aimed at finding efficient solutions for widespread applications. The benefits of these algorithms were observed in their optimality and simplicity with speed. Many of the algorithms were readdressed to solve the problem of finding shortest path. Heuristic search techniques make use of problem specific knowledge to find efficient solutions. Most of these techniques determine the next best possible state leading towards the goal state by using evaluation function. This paper shows the practical performance of the following algorithms, to find the shortest path: Hill Climbing, Steepest-ascent, and Best-First and A*. While implementing these algorithms, we used the data structures which were indicated in the original papers. In this paper we present an alternative data structure multi-level link list and apply the heuristic technique to solve shortest path problem. This was tested for class of heuristic search family-- A* and Best First Search approaches. The results indicate that use of this type of data structure helps in improving the performance of algorithms drastically.

Keywords:  
Multilevel link list, Informed search techniques, Heuristic function, Shortest path algorithm.

1. INTRODUCTION

Heuristic search algorithms have exponential time and space complexities as they store complete information of the path including the explored intermediate nodes. Hence many applications involving heuristic search techniques to find optimal solutions tend to be expensive. Despite of these, the researchers have strived to find optimal solution in best possible time. In this paper we have considered major algorithms which are applied to find the shortest path: hill – climbing, steepest – ascent, best first and A* [1, 2, 4].

Hill climbing algorithms expand the most promising descendant of the most recently expanded node until they encounter the solution. Steepest – ascent hill climbing differs from hill climbing algorithm only the way in which the next node is selected. In this method it selects best successor node for expansion, unlike the first successor node for expansion, as done in hill climbing. Though this method tries to choose best possible path, but this method, like hill climbing method may fail to find a solution by reaching to a node from were no improvements can be done [5, 8]. Best first search method selects the “best” node for further expansion by applying a
heuristic function. It then generates the successor node in similar fashion till the goal node is reached. This technique tries to explore the advantages of breadth first and depth first search technique and provides better time bound solution. Best first algorithm involves OR graph, it avoids the node duplication and also works on the assumption that each node has parent link to give the best node from the node where it is derived and link to successors. A* algorithm is a slight modified version of best search algorithm. The difference is that in A* the estimate to the goal state is given by heuristic function and also it makes use of the cost of the path developed [2,3,6].

We will now discuss each of these methods for finding the shortest path.

2. HILL CLIMBING METHOD FOR SHORTEST PATH FINDING

Hill climbing algorithm expands one node at a time beginning with the initial node. Each time it expands only the best node reachable from current node. Thus this method does not involve complex computation and due to this reason cannot ensure the completeness of the solution. Hill climbing method does not give a solution as may terminate without reaching the goal state [12]. Now let us look at algorithm of hill climbing for finding shortest path:

Procedure for hill climbing algorithm to find the shortest path:

```plaintext
hill_climb_sp (s, g, Q)
{
// s & g are start and goal nodes respectively.
// Q is queue which stores the successor
// nodes.
// let curr_node indicate current working
// node.
// path_cost gives the cost of the path.

initialiseQ;
curr_node = s;
path_cost=0;
while (1)
{
if (curr_node is goal node) then
terminate the process with SUCCESS;
else
{
find successor node of curr_node;
add this node in Q ;
}

if(Q is empty) then
terminate the process with FAILURE;
else
{
temp_node = first node of Q ;
}
}
}
```
One may notice that there can be failure state when algorithm may fail to reach the goal node. This will happen especially when the processing has reached to a node from where no new best nodes are available for further expansion. This will happen especially when the processing has reached to a node from where no new best nodes are available for further expansion.

3. STEEPEST ASCENT HILL CLIMBING METHOD FOR SHORTEST PATH FINDING

This method is a result of variation in hill climbing. Here, instead of moving the immediate best node, all the reachable nodes from current node are considered and among these the best one is chosen. In case of simple hill climbing, the first successor node which is better, is selected, due to this we may omit the best one. On the contrary steepest ascent hill climbing method not only reaches to the better state but also climbs up the steepest slope.

The variation in algorithm will be only in finding the best successors node from all the possible successor nodes from all possible successor, and not just the first best node [2,12,15].

The algorithm is given below:

```plaintext
steep_asc_hll(s, g, Q)
{
    initialise Q;
    curr_node = s;
    path_cost = 0;
    while (1)
    {
        if (curr_node is goal node) then
            terminate the process with SUCCESS;
        else
            find all the reachable node from curr_node;
            determine the cost of reaching to these nodes from curr_node;
            according their cost add them in Q.
    }
    if (Q is empty) then
```

terminated the process with FAILURE;
else
{    temp_node = first node of Q ;
    path_cost = path_cost +
                edge_cost [curr_node][temp_node];
    curr_node = temp_node;
    delete first node from Q ;
}
}

One can notice that hill climbing and steepest – hill climbing may fail to find a solution. Either
algorithm may not reach goal node as it may reach to a node where we may not find better nodes.
In such cases we may need to back-track as use more rules before choosing the next node.
However this process will be time consuming.

Both the methods discussed, may terminate not by finding a goal node but may reach node from
where no better nodes can be generated.

This will happen if the processing has reached to one of the following situations:

i) A node might have been selected which may be better that its neighbors, however
there may be few better nodes available which are step away. This situation is termed
as local maxima.

ii) A node might have been selected, whose neighbors may have the same value and
hence choosing next best node is difficult. This is known as plateau.

iii) A ridge is a special kind of local maximum, though the path selected so far may be
the best, yet making further moves difficult.
The next algorithms described here try to overcome these problems.

4. BEST FIRST METHOD FOR SHORTEST PATH FINDING

Best first search is a type of graph search algorithm. Here the nodes are expanded one at time by
choosing lowest evaluation value. This evaluation value is a result of heuristic function giving a
measure of distance to the goal node. For typical applications such as shortest path problems, the
evaluation function will be accurate as it accounts for distance or an absolute value [14,19].

Best first search is a combination of breadth and depth first search. Depth first search has an
advantage of arriving at solution without computing all nodes, whereas breadth first arriving at
solution without search ensured that the process does not get trapped. Best-first search, being
combination of these two, permits switching between paths. At every stage the nodes among the
generated ones, the best suitable node is selected for further expansion, may be this node belong
to the same level or different, thus can toggle between depth-first and breadth-first. This method
involves OR graph, avoids node duplication, and also requires two separate lists for processing.
OPEN list keeps the nodes whose heuristic values are determined, but yet to be expanded.
CLOSE list have the nodes which have been already checked, further these nodes are kept in this
list to ensure no duplications. It implies that the OPEN list has the nodes which need to be
considered for further processing and the entries in CLOSE list indicate the nodes which may not be re-required in further steps [6,7].

Let us look at the best first search algorithm for finding shortest path:

\[
bfs\_sp (s, g)
\]

\[
{ // s is start node \& g is goal node
  // let OPEN and CLOSE be the two lists.
  // let current\_w indicates current working //node.
  // path\_cost indicates cost of reaching to a // node x.

  path\_cost=0;
  OPEN=NULL;
  CLOSE=NULL;

  do
  {
    add\_node (OPEN, s); //add S to //OPEN list;
    current\_w= first element of OPEN;
    determine f(n) for successor nodes of current\_w;
    add these new nodes to OPEN based on their f (n) values;
    move current\_w to CLOSE;
    current\_w= first node of OPEN;
    path\_cost=path\_cost + f(n) of current\_w;
  }
  while (current\_w is not g and OPEN is not empty);
  If (current\_w=g) then
  print path\_cost;
  else
  print failure;
}
\]

In this case f (n) a heuristic function is an actual edge cost function.

5. A* ALGORITHM FOR SHORTEST PATH FINDING

We know that the various search techniques are designed, tested and are being used for various purposes whatever it is for system software or application software. But the base for this is however mainly because of the problems in planning domain. Classical approaches to heuristic search algorithm work on assumption of the existence of deterministic model of sequential decision making leading to the solution. The research work focused on solving planning problems under uncertainty [1]. Heuristic algorithms have given a new looked into the problems belonging to this domain [6,10].
The shortest path problem can be solved by A* algorithm. The heuristic function needs to evaluate two costs, g and h. Let \( g(n) \), in shortest path problem, represent cost of choosing the path from starting node to node \( n \); and \( h(n) \) represents optimal cost of node \( n \) to the goal node. Now the cost of node \( n \) is given by: \( f^*(n) = g(n) + h^*(n) \). However the value of \( h^*(n) \) will be unknown in most of the situations, which results in unknown value of \( f^*(n) \). A* algorithm, however makes a best approximation for \( h^*(n) \)[16,17].

The A* algorithm to solve the shortest path problem can be written as: [10]

Step 1: Start from the start node; place it in OPEN list. This will be current working node.

Step 2: Explore all the nodes adjacent to the one in OPEN list.

Step 3: Determine the cost function for all the nodes obtained in step 2; and place them in OPEN list in increasing order of cost function values.

Step 4: Move current working node, from OPEN list to CLOSE list.

Step 5: Now the first node in OPEN List will be the current working node (which is having least cost function due to insertion criteria in step 3).

Step 6: If this current working node is not the goal state (final node), then repeat step 2 to step 5.

Step 7: The CLOSE list gives the shortest path and the value of last cost function obtained gives the optimal cost.

6. EXPERIMENTAL RESULTS

All the algorithms discussed in previous sections were implemented in C++ and run on 2.4 GHz Intel C2D system with 2GB RAM. The random data sets were created for varying number of input nodes and saved in separate files. While testing these algorithms stored data was given as input data and processed. The algorithms were tested for the number nodes and edges explored/visited were compared. The Number of nodes and edges considered during the process for various algorithms are given in Table 1 and Table 2 respectively.

Note: HC –Hill Climbing, ST_AC --Steepest Ascent Hill Climbing, BFS—Best First Search and A*.
Table 1: Number of nodes considered.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>HC</th>
<th>ST_AC</th>
<th>BFS</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1439</td>
<td>915</td>
<td>351</td>
<td>183</td>
</tr>
<tr>
<td>1100</td>
<td>2171</td>
<td>1006</td>
<td>782</td>
<td>426</td>
</tr>
<tr>
<td>1200</td>
<td>2394</td>
<td>1106</td>
<td>1806</td>
<td>1029</td>
</tr>
<tr>
<td>1300</td>
<td>2333</td>
<td>1214</td>
<td>762</td>
<td>401</td>
</tr>
<tr>
<td>1400</td>
<td>2787</td>
<td>1310</td>
<td>1871</td>
<td>1057</td>
</tr>
<tr>
<td>1500</td>
<td>2923</td>
<td>1394</td>
<td>2360</td>
<td>1346</td>
</tr>
<tr>
<td>1600</td>
<td>3023</td>
<td>1510</td>
<td>363</td>
<td>188</td>
</tr>
<tr>
<td>1700</td>
<td>3339</td>
<td>1598</td>
<td>2696</td>
<td>1536</td>
</tr>
<tr>
<td>1800</td>
<td>3373</td>
<td>1722</td>
<td>2876</td>
<td>1634</td>
</tr>
<tr>
<td>1900</td>
<td>3749</td>
<td>1820</td>
<td>219</td>
<td>111</td>
</tr>
<tr>
<td>2000</td>
<td>3875</td>
<td>1921</td>
<td>2767</td>
<td>1567</td>
</tr>
<tr>
<td>2100</td>
<td>4063</td>
<td>2015</td>
<td>1952</td>
<td>1065</td>
</tr>
<tr>
<td>2200</td>
<td>4277</td>
<td>2106</td>
<td>3683</td>
<td>2137</td>
</tr>
<tr>
<td>2300</td>
<td>3557</td>
<td>2198</td>
<td>3713</td>
<td>2132</td>
</tr>
<tr>
<td>2400</td>
<td>4751</td>
<td>2301</td>
<td>1850</td>
<td>1017</td>
</tr>
<tr>
<td>2500</td>
<td>4959</td>
<td>2388</td>
<td>2603</td>
<td>1479</td>
</tr>
<tr>
<td>2600</td>
<td>4995</td>
<td>2502</td>
<td>3641</td>
<td>2074</td>
</tr>
<tr>
<td>2700</td>
<td>5275</td>
<td>2608</td>
<td>1885</td>
<td>1099</td>
</tr>
<tr>
<td>2800</td>
<td>5257</td>
<td>2704</td>
<td>2163</td>
<td>1190</td>
</tr>
<tr>
<td>2900</td>
<td>5705</td>
<td>2805</td>
<td>1712</td>
<td>922</td>
</tr>
<tr>
<td>3000</td>
<td>5945</td>
<td>2900</td>
<td>1657</td>
<td>876</td>
</tr>
</tbody>
</table>

Table-1 shows that there is significant amount of improvement on number of nodes being considered in A* algorithm compared to the rest of the methods.

Table 2: Number of edges considered

<table>
<thead>
<tr>
<th>Nodes</th>
<th>HC</th>
<th>ST_AC</th>
<th>BFS</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>919881</td>
<td>492103</td>
<td>318260</td>
<td>165347</td>
</tr>
<tr>
<td>1100</td>
<td>1207619</td>
<td>595531</td>
<td>695807</td>
<td>376975</td>
</tr>
<tr>
<td>1200</td>
<td>1437589</td>
<td>709383</td>
<td>1273917</td>
<td>702929</td>
</tr>
<tr>
<td>1300</td>
<td>1686345</td>
<td>835800</td>
<td>839059</td>
<td>439800</td>
</tr>
<tr>
<td>1400</td>
<td>1957171</td>
<td>970824</td>
<td>1663677</td>
<td>920304</td>
</tr>
<tr>
<td>1500</td>
<td>2255995</td>
<td>1112690</td>
<td>2003573</td>
<td>1112315</td>
</tr>
<tr>
<td>1600</td>
<td>2549145</td>
<td>1267835</td>
<td>545470</td>
<td>281622</td>
</tr>
<tr>
<td>1700</td>
<td>2885731</td>
<td>1432627</td>
<td>2585858</td>
<td>1430620</td>
</tr>
<tr>
<td>1800</td>
<td>3223745</td>
<td>1611151</td>
<td>2905999</td>
<td>1605239</td>
</tr>
<tr>
<td>1900</td>
<td>3605601</td>
<td>1795683</td>
<td>402089</td>
<td>202895</td>
</tr>
<tr>
<td>2000</td>
<td>3992219</td>
<td>1990342</td>
<td>3443484</td>
<td>1905039</td>
</tr>
<tr>
<td>2100</td>
<td>4401245</td>
<td>2193714</td>
<td>3072621</td>
<td>1667820</td>
</tr>
<tr>
<td>2200</td>
<td>4831941</td>
<td>2408930</td>
<td>4399274</td>
<td>2416884</td>
</tr>
<tr>
<td>2300</td>
<td>5014481</td>
<td>2633403</td>
<td>4721867</td>
<td>2629654</td>
</tr>
<tr>
<td>2400</td>
<td>5754649</td>
<td>2867561</td>
<td>3522853</td>
<td>1921764</td>
</tr>
<tr>
<td>2500</td>
<td>6244621</td>
<td>3112370</td>
<td>4758366</td>
<td>2602019</td>
</tr>
<tr>
<td>2600</td>
<td>6744499</td>
<td>3368084</td>
<td>5818135</td>
<td>3240099</td>
</tr>
<tr>
<td>2700</td>
<td>7280819</td>
<td>3631545</td>
<td>4146417</td>
<td>2263359</td>
</tr>
<tr>
<td>2800</td>
<td>7805331</td>
<td>3907490</td>
<td>4805723</td>
<td>2621745</td>
</tr>
<tr>
<td>2900</td>
<td>8402039</td>
<td>4191736</td>
<td>4195443</td>
<td>2246319</td>
</tr>
<tr>
<td>3000</td>
<td>8993299</td>
<td>4487278</td>
<td>4208595</td>
<td>2241750</td>
</tr>
</tbody>
</table>
Table-2, as a consequence of nodes expanded or considered results in varying number of edges. In BFS and A* the edge count reduces, as we make proper heuristic estimation. Where as in Hill climbing or in Steepest Ascent, we try to choose the immediate best node, which will ultimately result in exploring more number of edges.

The resulting graphs of the two algorithms are given in Fig 1 and Fig 2.

![Figure 1: Comparison of number nodes considered against total nodes in graph.](image1)

![Figure 2: Comparison of number edges considered against total nodes in graph.](image2)

One may also observe here that certain unexpected variations in the values. This is mainly due to the fact that these algorithms were executed till they find the solution and were not run for fixed number of iterations.

**7. BEST FIRST SEARCH USING MLL AS DATA STRUCTURE**

Now let us look at the variations to the algorithms presented in section 4 and 5. Here let us make use of multi-level linked list as data structure for implementation [8,18,20].
In section 4, we discussed the conventional approach of Best first search for finding shortest path. In this section we present slightly modified approach for solving the problem of shortest path finding. We store the current working node in parent list of MLL and all the adjacent nodes in its successor list. The best node as determined by f(n), will be chosen for further expansion.

\[ f(n) = \min(\text{cost}(n,i)), \forall i, \text{ where } i \text{ is an adjacent node of } n. \]

The skeleton of the algorithm is given below:

**Best First method with multi-level linked list:**

```java
bfs_mll_sp (s, g)
{
    // s is start node & g is goal node
    // begin the process from s; this will be the
    // first node in MLL, let it be current // working node call it current_w
    // path_cost indicates cost of reaching to a
    // node x.
    path_cost=0;
    do
    {
        determine f(n) for successor nodes of current_w; add these new nodes to
        successor link S based on their f(n) values for the current parent node;
        current_w= first node of S;
        path_cost=path_cost + f(n) of current_w;
    }while (current_w is not g and S
    of current parent node is not empty);
    If (current_w=g) then
    print path_cost;
    else
    print failure;
}
```

In this case f(n) a heuristic function is an actual edge cost function.

**8. A* ALGORITHM USING MLL AS DATA STRUCTURE**

As stated earlier, shortest path problem aims at finding minimum cost path between pair of nodes cumulatively and then find the final path between start and goal nodes[9,21]. A* algorithm with MLL, result in pruning the search space [8,17]. The approach which has been followed in our work makes use of an exact accurate function. The evolution function f(n) is given as:

\[ f(n)=g(n)+h(n); \]

where g(n) is the cost of an edge between the currently explored node or current working node and the node n being examined, h(n) is the best edge cost value from the set of edge costs going out from the node n to the all possible adjacent nodes.

Let us look at the algorithm.

Step 1: Start from start state; this will become the first node in MLL, call it as current working node. Since this is the first node this will be the first node of parent list.
Step 2: Explore all the nodes adjacent to the one in the current parent node list.
Step 3: Determine g value of the current working node.
Step 4: Obtain the h values for all the nodes obtained in step 2.
Step 5: Find the f values for the expanded nodes; keep them according to their values in successor list for the current parent node (this will result in the list maintained in increasing order of the f values, which will be the cost function).
Step 6: Pick up the first node from the successor list obtained in step 5 which will be the next working node.
Step 7: If this current working node is not the goal state, then attach this node to parent list and repeat step 2 and step 6.
Step 8: The set of nodes belonging to parent list gives the shortest path and the cost function determined in the last step will be the optimal cost.

9. EXPERIMENTAL RESULTS OF BFS AND A* WITH MLL:

The BFS and A* algorithms discussed in previous sections were implemented in C++ and run on 2.4 GHz Intel C2D system with 2GB RAM. The random data sets were created for varying number of input nodes and saved in separate files. While testing these algorithms stored data was given as input data and processed.

The algorithms were tested for the number nodes and edges explored/visited were compared. The Number of nodes and edges considered during the process for various algorithms are given in Table 3 and Table 4 respectively.

Table 3: Number of nodes considered.

[Existing approach indicates the conventional approach that is being implemented and in MLL we used new method]

<table>
<thead>
<tr>
<th>Nodes Considered</th>
<th>BFS</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Existing Approach</td>
<td>Using MLL</td>
</tr>
<tr>
<td>1000</td>
<td>351</td>
<td>20</td>
</tr>
<tr>
<td>1100</td>
<td>782</td>
<td>21</td>
</tr>
<tr>
<td>1200</td>
<td>1806</td>
<td>30</td>
</tr>
<tr>
<td>1300</td>
<td>762</td>
<td>28</td>
</tr>
<tr>
<td>1400</td>
<td>1871</td>
<td>33</td>
</tr>
<tr>
<td>1500</td>
<td>2360</td>
<td>37</td>
</tr>
<tr>
<td>1600</td>
<td>362</td>
<td>47</td>
</tr>
<tr>
<td>1700</td>
<td>2696</td>
<td>64</td>
</tr>
<tr>
<td>1800</td>
<td>2876</td>
<td>32</td>
</tr>
<tr>
<td>1900</td>
<td>219</td>
<td>33</td>
</tr>
<tr>
<td>2000</td>
<td>2767</td>
<td>36</td>
</tr>
<tr>
<td>2100</td>
<td>1952</td>
<td>82</td>
</tr>
<tr>
<td>2200</td>
<td>3683</td>
<td>27</td>
</tr>
<tr>
<td>2300</td>
<td>3713</td>
<td>52</td>
</tr>
<tr>
<td>2400</td>
<td>1850</td>
<td>33</td>
</tr>
<tr>
<td>2500</td>
<td>2682</td>
<td>51</td>
</tr>
<tr>
<td>2600</td>
<td>3641</td>
<td>77</td>
</tr>
<tr>
<td>2700</td>
<td>1886</td>
<td>33</td>
</tr>
<tr>
<td>2800</td>
<td>2163</td>
<td>56</td>
</tr>
<tr>
<td>2900</td>
<td>1712</td>
<td>40</td>
</tr>
<tr>
<td>3000</td>
<td>1637</td>
<td>40</td>
</tr>
</tbody>
</table>
Table 4: Number of edges considered.

[Existing approach indicates the conventional approach that is being implemented and in MLL we used new method]

<table>
<thead>
<tr>
<th>Nodes Considered</th>
<th>BFS</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Existing method</td>
<td>Using MLL</td>
</tr>
<tr>
<td>1000</td>
<td>318260</td>
<td>8795</td>
</tr>
<tr>
<td>1100</td>
<td>698307</td>
<td>10394</td>
</tr>
<tr>
<td>1200</td>
<td>1273917</td>
<td>12667</td>
</tr>
<tr>
<td>1300</td>
<td>839039</td>
<td>14001</td>
</tr>
<tr>
<td>1400</td>
<td>1663677</td>
<td>17652</td>
</tr>
<tr>
<td>1500</td>
<td>2603375</td>
<td>18267</td>
</tr>
<tr>
<td>1600</td>
<td>545470</td>
<td>22183</td>
</tr>
<tr>
<td>1700</td>
<td>2585858</td>
<td>27306</td>
</tr>
<tr>
<td>1800</td>
<td>2905999</td>
<td>24276</td>
</tr>
<tr>
<td>1900</td>
<td>402089</td>
<td>26541</td>
</tr>
<tr>
<td>2000</td>
<td>3443484</td>
<td>30619</td>
</tr>
<tr>
<td>2100</td>
<td>3072621</td>
<td>37144</td>
</tr>
<tr>
<td>2200</td>
<td>4359274</td>
<td>33394</td>
</tr>
<tr>
<td>2300</td>
<td>4721867</td>
<td>37251</td>
</tr>
<tr>
<td>2400</td>
<td>3322853</td>
<td>38437</td>
</tr>
<tr>
<td>2500</td>
<td>4758366</td>
<td>42381</td>
</tr>
<tr>
<td>2600</td>
<td>5818135</td>
<td>48610</td>
</tr>
<tr>
<td>2700</td>
<td>4146417</td>
<td>46383</td>
</tr>
<tr>
<td>2800</td>
<td>4605723</td>
<td>50453</td>
</tr>
<tr>
<td>2900</td>
<td>4195443</td>
<td>52588</td>
</tr>
<tr>
<td>3000</td>
<td>4208595</td>
<td>55831</td>
</tr>
</tbody>
</table>

10. CONCLUSION

We have presented major class of heuristic algorithms. The comparison shows that though all these algorithms can be applied to find the shortest path, but should not be used unless there is a real-time, event driven actions are anticipated. The comparison gives us clear idea that best-first search and A* algorithms are very well suitable when goal node cannot be reached from all nodes. However there may be interesting scenarios that may come out when these algorithms are applied with different data structures.

The results clearly indicate that hill climbing or steepest ascent hill climbing algorithms are not suitable for problems such as shortest path finding. This is due to the fact that there is no assurance of getting final optimal solution for all the cases. Best first and A* algorithms on the other hand ensure optimal solution for limited graph size. For larger number of nodes these algorithms not only tend to take more time but the optimality factor may be of concern.
There are number of factors for using different data structure approach, in heuristic algorithm – as special case study we implemented Best First Search and A* algorithm with multilevel linked list as data structure is the main reason of the increased speed in determining the shortest path. One can easily figure out the fact that both these algorithms with multilevel lined list results in reduced area that is to be searched, which eventually gives us the better way of handling the nodes at runtime. Since the number of nodes or edges considered in the process is less, the time taken to find the optimal solution will also be less which are shown in the results section.

One may even work on eliminating the already explored nodes in subsequent levels, which may further reduce the space requirement.

REFERENCES