**Deerwalk Institute of Technology  
*Siphal, Kathmandu***

Assignment: Design an AI System

Our Case: UNILOC (UNIversity LOCation Finder)

**Submitted By**

Ruby Shrestha | 0341

Shreha Regmi | 0335

Section A

**Submitted To**

Birodh Risal

Artificial Intelligence

DWIT

**UNILOC – THE UNIversity LOCation Finder**

**ABSTRACT**

AI systems at present play with large amount of data and efficient searching in the available search space developed using those data. There are different real world problems which yearn to be solved via such AI mechanism of searching in a proper search space. UNILOC (University Location Finder) is one of such real world problem. The searching in UNILOC can be performed using A\* Search Algorithm to search location to particular location within the university.

**PROBLEM STATEMENT**

Universities usually cover a large land area. In a university, there are many different departments and every department again has many rooms. Different university authorities also have dedicated rooms for themselves. What these prior statements are trying to indicate is that when a student (especially, freshmen) need to visit certain location/room within the university, the student is going to have a difficult time. He might have to visit certain location within certain time constraint and hence, having to keep searching for a way to get to that location might be infuriating for him. In such a situation, he would want to discover a shortest path to get to that particular location from his current location within the university.

**SOLUTION**

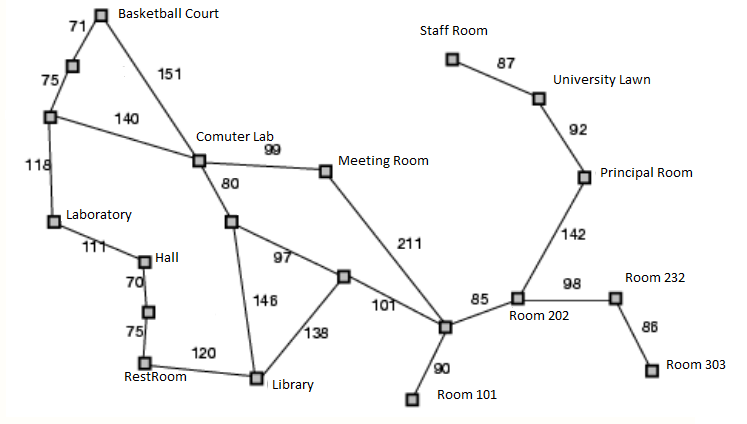
Finding an effective path to our destination is usually a problem to many of us. An application that helps us to find a short path to our destination in our domain of interest, in this case it’s the University of Choice, can be a boon to us. Here, we are implementing the AI system considering our domain of interest as a particular university. However, it can also be implemented in places such as a large departmental store to find the shortest path to the location where the product we are trying to find is available.

Let us consider our scenario- the University Location Finder. Each room and location within the university can be considered / added as nodes of the search space and the edges joining those nodes can be considered as path between those nodes. These edges have weights. Those weights indicate the distance of the path joining two nodes.

The place we are currently present in will be the START NODE. Our destination will be the GOAL NODE. From that START NODE, the shortest path to our destination will be computed and will be shown via internal location map service of the University. The parameters: START NODE, GOAL NODE and WEIGHTS are basis for our knowledge base. With the help of these parameters, we obtain the shorted path.

We can make use of A\* Search Algorithm for searching the space.

**ALGORITHM**



Here, node stands for each destination in our university as the given figure.

struct node {

node \*parent;

int x, y;

float f, g, h;

}

In order to calculate the distance, we’ll be doing A\* search. We initialize a queue1 where we will store f, the cost function. The cost in this case is the total distance. Now, we pop the location with the least **f** off the queue1, and we generate its successors. If one of its successors is the goal, we found our destination and we now can stop the search. For each successor, we also calculate its distance and we add it to the queue1. At the end of this iteration, we put the destination we popped off from the queue1 to a new queue2.

|  |  |
| --- | --- |
| 1:  2:  3:  -  4:  5:  6:  7:  8:  9:  10:  11:  12:  -  13:  -  14:  -  15:  16:  17:  18: | // A\*  initialize the queue1  initialize the queue2  put the starting node on queue1 (f is left zero)  while queue1 is not empty  find the node with the least **f** on the open list, call it "q"  pop q off the queue1  generate q's successors and set their parents to q  for each successor  if successor is the goal, stop the search  successor.g = q.g + distance between successor and q  successor.h = distance from goal to successor  successor.f = successor.g + successor.h  if a node with the same position as successor is in queue1  which has a lower **f** than successor, skip this successor  if a node with the same position as successor is in queue2  which has a lower **f** than successor, skip this successor  otherwise, add the node to the queue1  end  push q on the queue2  end |

**CONCLUSION**

An overview of how the problem of finding the shortest path to a location of choice in a large university can be solved has been provided by this paper via the use of A\* Search Algorithm. Similar approach can be used to find solutions to similar problems of short path finding.