Assignment No 01: Heuristic Function

CSE-0408 Summer 2021

Tahsin Shovon
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
ID:UG02-44-17-023

***Abstract*—A heuristic function h(n) , takes a node n and returns a non-negative real number that is an estimate of the cost of the least-cost path from node n to a goal node. The function h(n) is an admissible heuristic if h(n) is always less than or equal to the actual cost of a lowest-cost path from node n to a goal.**

***Index Terms*—About Heuristic Function and the 8-puzzle problem solving in Python.**

I. Introduction

Definition: A heuristic function is a shortcut to solving a problem when there are no exact solutions for it or the time to obtain the solution is too long.A good heuristic function is determined by its efficiency. More is the information about the problem, more is the processing time.

objective: The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand. This solution may not be the best of all the solutions to this problem, or it may simply approximate the exact solution.

II. Literature Review

The development and user interface analysis to an object tends to follow technological developments, as it has become an extension of the human being. In recent years, with the development of digital interfaces, a set of procedures, rules and usability standards have been developed to facilitate projecting processes and evaluation of interfaces to ensure a better user interaction. There are plenty of heuristics sets, created by authors, which serve as the basis for other researchers and professionals in the improvement and expansion of the joint heuristics. Thus, this research presents a study on the heuristics used for the assessment of human-computer interaction process (IHC), the contact of users with interfaces. The objective of the research is to identify the similiarities, singularities and equivalences between the concepts from the heuristics set available for evalution of interface.

III. Proposed Methodology

Heuristics are methods for solving problems in a quick way that delivers a result that is sufficient enough to be useful given time constraints. Investors and financial professionals use a heuristic approach to speed up analysis and investment decisions. Current state [[8,1,2],[3,6,4],[0,7,5]] and the goal state is [[1,2,3],[8,0,4],[7,6,5]

IV. RULES OF SOLVING PROBLEM

It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive. Admissibility of the heuristic function is given as: h(n) ¡= h*(n) 1.UP 2.Down 3.Right 4.Left

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, Reading (1984)
[2] Nilsson, N.J.: Principles of Artificial Intelligence. Tioga Publishing Company (1980)
[3] Stentz, A.: Optimal and Efficient Path Planning for Partially-Known Environments. In: Proceedings IEEE International Conference on Robotics and Automation, pp. 3310–3317 (1994)
[4] Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. Journal of the ACM 32(3), 505–536 (1985)
[5] Chakrabarti, P., Ghosh, S., Acharya, A., DeSarkar, S.: Heuristic search in restricted memory. Artificial Intelligence 47, 197–221 (1989)
[6] Gaschnig, J.: Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Computer Science Department, Carnegie Mellon University (1979)
[7] Kanal, L.N., Kumar, V.: Search in Artificial Intelligence. Springer, Berlin (1988)

Assignment -01 Huristic Function Code

```
#include<bits/stdc++.h>

using namespace std;

#define D(x) cerr<<__LINE__<<" : "<<#x<<" -> "<<x<<endl

#define rep(i,j) for(inti = 0; i< 3; i++) for(int j = 0; j < 3; j++)

#define PII pair <int, int>

typedef vector<vector<int>>vec2D;



constint MAX = 1e5+7;

int t=1, n, m, l, k, tc;



intdx[4] = {0, 0, 1, -1};

intdy[4] = {1, -1, 0, 0};



vec2D init{

{8, 1, 2},

{3, 6, 4},

{0, 7, 5}
```

```
};

vec2D goal{

{1, 3, 2},

{8, 0, 4},

{7, 6, 5}

};

//vec2D init{

// {1, 2, 3},

// {8, 6, 0},

// {7, 5, 4}

//};

//vec2D goal{

// {1, 2, 3},

// {8, 0, 4},

// {7, 6, 5}

//};

//vec2D init{

// {1, 3, 2},
// {4, 0, 7},

// {6, 5, 8}

//};

//vec2D goal{

// {0, 2, 4},

// {1, 3, 8},

// {6, 5, 7}

//};



struct Box {

vec2D mat{ { 0,0,0 },{ 0,0,0},{ 0,0,0} };

int diff, level;

int x, y;

intlastx, lasty;

Box(vec2D a,int b = 0, int c = 0, PII p = {0,0}, PII q = {0,0}) {

rep(i,j) mat[i][j] = a[i][j];

diff = b;

level = c;

x = p.first;
y = p.second;

lastx = q.first;

lasty = q.second;

}

};



bool operator < (Box A, Box B) {

if(A.diff == B.diff) return A.level<B.level;

return A.diff<B.diff;

}



intisEqual(vec2D a, vec2D b) {

intret(0);

rep(i,j) if (a[i][j] != b[i][j]) ret--;

return ret;

}



bool check(inti, int j) {
```

```
return i>=0 and i<3 and j>=0 and j<3;

}



void print(Box a) {

rep(i,j)

cout<<a.mat[i][j] << (j == 2 ? "\n" : " ");

D(-a.diff);

D(-a.level);

cout<< "(" <<a.x<< "," <<a.y<<")\n\n";

}





void dijkstra(int x, int y) {

map < vec2D, bool>mp;

priority_queue< Box > PQ;

intnD = isEqual(init, goal);

Box src = {init, nD, 0, {x,y}, {-1,-1}};

PQ.push(src);
```

```
int state = 0;

while(!PQ.empty()) {

state++;

Box now = PQ.top();

PQ.pop();

print(now);

if(!now.diff) {

puts("Goal state has been discovered");

cout<< "level : " << -now.level<< "\n";

D(state);

break;

}

if(mp[now.mat]) continue;

mp[now.mat] = true;

for(inti = 0; i< 4; i++) {

int xx = now.x + dx[i];

intyy = now.y + dy[i];

if(check(xx, yy)) {
```

```
if(now.lastx == xx and now.lasty == yy) continue;

Box temp = now;

swap(temp.mat[temp.x][temp.y], temp.mat[xx][yy]);

temp.diff = isEqual(temp.mat, goal);

temp.level = now.level - 1;

temp.x = xx;

temp.y = yy;

temp.lastx = now.x;

temp.lasty = now.y;

PQ.push(temp);

}

}

}

}



signed main() {

puts("Current State:");
```

```
rep(i,j) cout<<init[i][j] << (j == 2 ? "\n" : " ");

puts("");

puts("Goal State:");

rep(i,j) cout<< goal[i][j] << (j == 2 ? "\n" : " ");

puts("\n............Search Started...............\n");

rep(i,j) if(!init[i][j]) dijkstra(i,j);

return 0;

}
```

Output of the code given below:

```
Current state :

8   1   2

3   6   4

0   7   5


Goal State:

1   3   2

8   0   4

7   6   5
```

```
-------- Search started to --------

8    1    2

6    6    4

0    7    5


79  :   -a.diff   ->  6

80  :   -a.level   ->  0


(2  ,   0  )
```

Assignment No 02: Breadth-First Search(BFS)

CSE-0408 Summer 2021

Tahsin Shovon
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
ID:UG02-44-17-023

***Abstract*—This paper introduced for Breadth-First Search(BFS) problem solved using C++ language.**

***Index Terms*—Here I mostly used in My report C++ language and Code-block code editor.**

I. Introduction

BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

II. Literature Review

BFS and its application in finding connected components of graphs were invented in 1945 by Konrad Zuse, in his (rejected) Ph.D. thesis on the Plankalk ul programming language, but this was not published until 1972. It was reinvented in 1959 by Edward F. Moore, who used it to find the shortest path out of a maze,[5][6] and later developed by C. Y. Lee into a wire routing algorithm (published 1961). In 2012 Farhad S. et. al. [4] proposed new resolution for solving N-queens by using combination of DFS (Depth First Search) and BFS (Breadth First Search) techniques.

III. Proposed Methodology

Here i Discuss BFS Algorithm: 1. for each u in V s 2. do color[u] ←WHITE 3. d[u] ←infinity 4. [u] ←NIL 5. color[s] ←GRAY 6. d[s] ←0 7. [s] ←NIL 8. Q ← 9. ENQUEUE(Q, s) 10 while Q is non-empty 11. do u ←DEQUEUE(Q) 12. for each v adjacent to u 13. do if color[v] ←WHITE 14. then color[v] ←GRAY 15. d[v] ←d[u] + 1 16. [v] ←u 17. ENQUEUE(Q, v) 18. DEQUEUE(Q) 19. color[u] ←BLACK

IV. RULES OF SOLVING PROBLEM

Breadth-first search starts at a given vertex s, which is at level 0. In the first stage, we visit all the vertices that are at the distance of one edge away. When we visit there, we paint as "visited," the vertices adjacent to the start vertex s - these vertices are placed into level 1. In the second stage, we visit all the new vertices we can reach at the distance of two edges away from the source vertex s. These new vertices, which are adjacent to level 1 vertices and not previously assigned to a level, are placed into level 2, and so on. The BFS traversal terminates when every vertex has been visited.

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] Borges, Paulo HR, et al. "Carbonation of CH and C–S–H in composite cement pastes containing high amounts of BFS." Cement and concrete research 40.2 (2010): 284-292 . CRC press, 1984.
[2] Coppin, B. (2004). Artificial intelligence illuminated. Jones Bartlett Learning. pp. 79–80.
[3] Cormen, Thomas H. "22.2 Breadth-first search". Introduction to algorithms. ISBN 978-81-203-4007-7. OCLC 1006880283. International Conference on Robotics and Automation, pp. 3310–3317 (1994)
[4] Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. Journal of the ACM 32(3), 505–536 (1985)
[5] Chakrabarti, P., Ghosh, S., Acharya, A., DeSarkar, S.: Heuristic search in restricted memory. Artificial Intelligence 47, 197–221 (1989)
[6] Gaschnig, J.: Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Computer Science Department, Carnegie Mellon University (1979)
[7] Boston: Addison-Wesley, ISBN 978-0-201-89683-1

Assignment-02

Breadth-First Search(BFS) code

```
#include<bits/stdc++.h>

using namespace std;



#define MX 110



vector < int > graph[MX];

bool vis[MX];

int dist[MX];

int parent[MX];



void bfs(int source){

queue < int > Q;

// initialization

vis[source] = 1;

dist[source] = 0;

Q.push(source);
while(!Q.empty()){

int node = Q.front();

Q.pop();



for (int i = 0; i < graph[node].size(); i++){

int next = graph[node][i];

if (vis[next] == 0){

vis[next] = 1; // visit

dist[next] = dist[node] + 1; // update

Q.push(next); // push to queue



// set parent

parent[next] = node;

}

}

}

}



/*
```

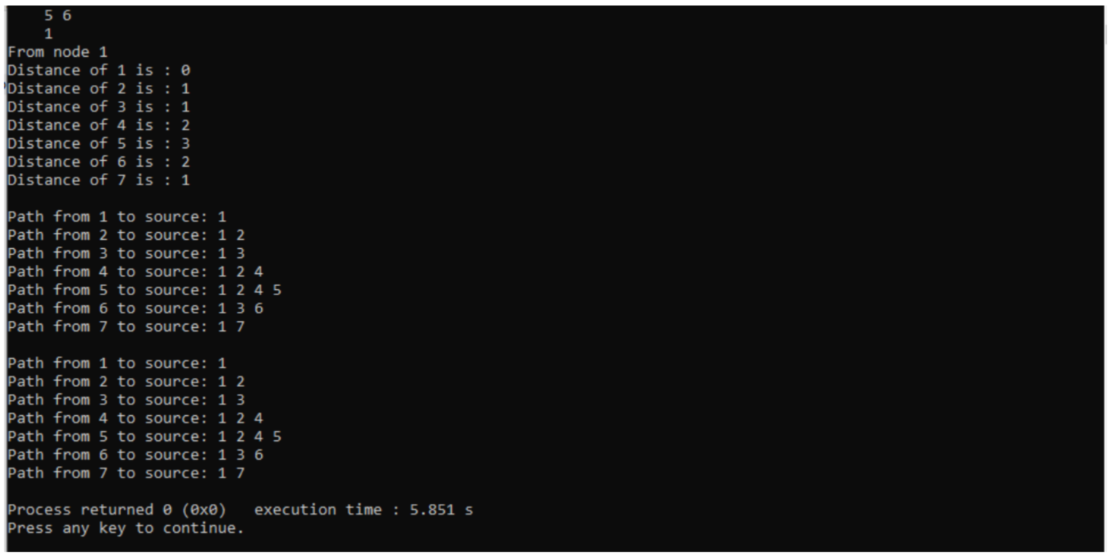
```
input:

7 9

1 2

1 3

1 7

2 3

3 7

2 4

4 5

3 6

5 6

1

*/



// path printing functions



// recursive function

void printPathRecursive(int source, int node){
```

```
if (node == source){

cout << node << " "; // print from source

return;

}

printPathRecursive(source, parent[node]);

cout << node << " ";

}



// iterative function

void printPathIterative(int source, int node){

vector<int> path_vector;



while(node != source){

path_vector.push_back(node);

node = parent[node];

}

path_vector.push_back(source); // inserting source



for (int i = path_vector.size() - 1; i >= 0; i--){
```

```
cout << path_vector[i] << " ";

}

}



int main()

{

int nodes, edges;

cin >> nodes >> edges;



for (int i = 1; i <= edges; i++){

int u, v;

cin >> u >> v;

graph[u].push_back(v);

graph[v].push_back(u);

}



int source;

cin >> source;
```

```
bfs(source);


cout << "From node " << source << endl;

for (int i = 1; i <= nodes; i++){

cout << "Distance of " << i << " is : " << dist[i] << endl;

}

cout << endl;


// path printing example


// recursive version

for (int i = 1; i <= nodes; i++){

cout << "Path from " << i << " to source: ";

printPathRecursive(source, i);

cout << endl;

}


cout << endl;
```

```
// iterative version

for (int i = 1; i <= nodes; i++){

cout << "Path from " << i << " to source: ";

printPathIterative(source, i);

cout << endl;

}



return 0;

}
```



```
    5 6
    1
From node 1
Distance of 1 is : 0
Distance of 2 is : 1
Distance of 3 is : 1
Distance of 4 is : 2
Distance of 5 is : 3
Distance of 6 is : 2
Distance of 7 is : 1

Path from 1 to source: 1
Path from 2 to source: 1 2
Path from 3 to source: 1 3
Path from 4 to source: 1 2 4
Path from 5 to source: 1 2 4 5
Path from 6 to source: 1 3 6
Path from 7 to source: 1 7

Path from 1 to source: 1
Path from 2 to source: 1 2
Path from 3 to source: 1 3
Path from 4 to source: 1 2 4
Path from 5 to source: 1 2 4 5
Path from 6 to source: 1 3 6
Path from 7 to source: 1 7

Process returned 0 (0x0)   execution time : 5.851 s
Press any key to continue.
```