Assignment: Best-First search in Graph representation problem solving

CSE-0408 Summer 2021

Afifa Nowreen Akhter
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
afifa.nowreen@gmail.com

***Abstract*—The Best first search uses the concept of a Priority queue and heuristic search. To search the graph space, the BFS method uses two lists for tracking the traversal. An 'Open' list which keeps track of the current 'immediate' nodes available for traversal and 'CLOSED' list that keeps track of the nodes already traversed.**

***Index Terms*—BFS, Search**

I. Introduction

If we consider searching as a form of traversal in a graph, an uninformed search algorithm would blindly traverse to the next node in a given manner without considering the cost associated with that step. An informed search, like Best first search, on the other hand would use an evaluation function to decide which among the various available nodes is the most promising (or 'BEST') before traversing to that node.

II. Literature Review

Best First Search is a merger of Breadth First Search and Depth First Search. Best First Search is implemented using the priority queue. The advantage of Depth First Search is that it gives a solution without calculating all node. While Breadth First Search arrives at a solution without search guaranteed that the procedure does not get caught. Best First Search, being a mixer of these two, licenses exchanging between paths. At each stage the nodes among the created ones, the best appropriate node is chosen for facilitating expansion, might be this node have a place to a similar level or different, hence can flip between Depth First and Breadth First Search. It is also known as greedy search. Time complexity is O(bd) and space complexity is O(bd), where b is branching factor and d is solution depth.

III. Conclusion

Best-first search algorithms are very well suitable when goal node cannot be reached from all nodes. The BFS algorithm is useful for analyzing the nodes in a graph and constructing the shortest path of traversing through these.

Acknowledgment

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

References

[1] Daniel Carlos Guimaraes Pedronette received a B.Sc. in computer science (2005) from the State University of Sao Paulo (Brazil) and ˜ the M.Sc. degree in computer science (2008) from the University of Campinas (Brazil). He got his doctorate in computer science at the same university in 2012.

[2] Arboleya, P., Mohamed, B., Gonzalez-Mor ´ an, C., El-Sayed, I. (2015). BFS algorithm for voltage-constrained meshed DC traction networks with nonsmooth voltage-dependent loads and generators. IEEE Transactions on Power Systems, 31(2), 1526-1536.

[3] Awerbuch, B., Gallager, R. G. (1985, October). Distributed BFS algorithms. In 26th Annual Symposium on Foundations of Computer Science (sfcs 1985) (pp. 250-256). IEEE.

[4]] Gharehchopogh, F. S., Seyyedi, B., Feyzipour, G. (2012). a new solution for n-queens problem using blind approaches: DFS and BFS algorithms. International Journal of Computer Applications, 53(1).

[5] Chikkerur, S., Cartwright, A. N., Govindaraju, V. (2006, January). Kplet and coupled BFS: a graph based fingerprint representation and matching algorithm. In International Conference on Biometrics (pp. 309- 315). Springer, Berlin, Heidelberg.

[6] Chakraborty, S., Satti, S. R. (2017, August). Space-efficient algorithms for maximum cardinality search, stack BFS, queue BFS and applications. In International Computing and Combinatorics Conference (pp. 87-98). Springer, Cham.

```
#include <bits/stdc++.h>
using namespace std;
int a,b;
int parent[100];
int cost [1000][10000];
int find(int i)
{
    while (parent[i] != i)
        i = parent[i];
    return i;
}

void unionl(int i, int j)
{
    int s = find(i);
    int r = find(j);
    parent[s] = r;
}

void BFsMST()
{
    int mincost = 0;
    int edge_count = 0;
    while (edge_count < a - 1)
    {
        int min = INT_MAX, s = -1, r = -1;
        for (int i = 0; i < a; i++) {
            for (int j = 0; j < a; j++)
            {
                if (find(i) != find(j) && cost[i][j] < min)
                {
```

Fig. 1. Code

```
                {
                    min = cost[i][j];
                    s = i;
                    r = j;
                }
            }
        }

        unionl(s, r);
        cout<<"Edge "<<edge_count++<<":("<<s<<" "<<r<<") cost:"<<min<<endl;
        mincost += min;
    }
    cout<<endl<<"Minimum cost= "<<mincost;
}

int main()
{
    a=8;
    b=9;
    for(int i=0;i<a;i++)
    {
        for(int j=0; j<a; j++)
        {
            cost[i][j]= INT_MAX;
        }
    }

cost[0][1]=1;
cost[0][2]=2;
cost[2][3]=3;
cost[3][4]=4;
```

Fig. 2. Code

```
cost[0][1]=1;
cost[0][2]=2;
cost[2][3]=3;
cost[3][4]=4;
cost[3][5]=5;
cost[5][4]=100;
cost[5][6]=7;
cost[6][7]=101;
cost[4][7]=8;
    for (int i = 0; i < a; i++)
        parent[i] = i;

    BFsMST();
    return 0;
}
```

Fig. 3. Code