CE 5290 Course Project: Report

Bi-directional Dijkstra's Algorithm

Hrishikesh Gadekar (CE17B116)

A. Algorithm Description:

Dijkstra's Algorithm, a Label-setting algorithm, is an iterative approach to determine the shortest path between a pair of nodes, such as a source node *s* to a sink node *t* in a directed Network *G*. These are also known as 'Single source shortest path problem'. The algorithm maintains temporary distance label *d(i)* with every node *i* at each iteration which represent an upper bound on the shortest path to that node. At any intermediate sstep, the algorithm divides the nodes into two sets: those which it designates as permanently labeled in set *S* and those it designates as temporarily labeled in set *S'*. In the beginning, source node *s* is added to the set *S* with $d(s) = 0$. All other nodes which are in set *S'* are assigned a distance label equal to ∞. The algorithm selects a node *i* with the minimum temporary label, makes it permanent, adds it to set *S* and scans arcs in its Adjacency list *A(i)* to update the distance labels of adjacent nodes. The algorithm terminates when it has designated all nodes as permanent. When Dijkstra's algorithm is used to find the shortest path from the sink node *t* to the source node *s*, it is called as Reverse Dijkstra's algorithm.

The Assumptions involved are as follows:

i. All arc lengths or Costs are non-negative integers.
ii. The network contains a directed path from node *s* to every other node in the network.
iii. The network does not contain a negative cycle.
iv. The network is directed.

The two important operations performed by the algorithm which attribute to its Correctness are:

i. *Node Selection*: Picking node *i* with smallest temporary label ensures optimality. This operation is performed *n* times. Total node selection time is $O(n^2)$.
ii. *Arc Processing / Distance update*: For each outgoing are (i, j) from node i,

$$\text{if } d(j) > d(i) + c_{ij} \text{ then}$$
$$\text{set } d(j) = d(i) + c_{ij}$$
$$\text{pred } [j] = i$$

The Shortest path is determined from the list of Predecessors.

Similarly, for Reverse Dijkstra's Algorithm, the list of Successors is maintained.

Arc Processing operation is done: if $d(i) > d(j) + c_{ij}$ then

$$\text{set } d(i) = d(j) + c_{ij}$$
$$\text{succ } [i] = j$$

Bidirectional Dijkstra's Algorithm is the simultaneous implementation of the Forward Dijkstra's algorithm from the source node *s* and the Reverse Dijkstra's algorithm from sink node *t*. This algorithm is effective and faster that the unidirectional Dijkstra's algorithm when it is required to find the shortest path distance from the source *s* to any specified node in the Network. It maintains sets *S, S1* and sets *T, T1* for forward and reverse algorithm respectively. The algorithm alternatively designates a node in *S1* and a node in *T1* as permanent until both the forward and reverse algorithms have permanently labeled the same node, say node k.
At this point, Let *P(i)* denote the shortest path from source node *s* to node $i \subseteq S$ found by the forward Dijkstra's algorithm, and let *P'(j)* denote the shortest path from node $j \subseteq T$ to sink node *t* found by the reverse Dijkstra's algorithm. The optimality check is done by comparing the current optimal path with $P(i) + P'(j)$ with the path $P(i) \cup \{(i, j)\} \cup P'(j)$ for some arc *(i, j)* for $i \subseteq S$ and $j \subseteq T$.

The Complexity Analysis is as follows:

1. *Node selections:* The Algorithm performs this operation n times and each such operation requires that it scans each temporarily labeled node. Therefore, the total node selection time is $n + (n-1) + (n-2) + \ldots + 1 = O(n^2)$.
2. *Distance updates:* The algorithm performs this operation $|A(i)|$ times for node i. Overall, the algorithm performs this operation m times (total number of edges). Since each distance update operation requires $O(1)$ time, the algorithm requires $O(m)$ total time to update all distance labels.
3. *Common node:* The Algorithm checks for the common node after each iteration, and considering at each iteration there exist n nodes in each list S1 and T1, It requires $O(n*n) = O(n^2)$ of total time.

   Total time complexity $= O(n^2 + m + n^2) = O(n^2)$

B. Pseudocode for Bi-directional Dijkstra's Algorithm:

```
Algorithm Bi-directional Dijkstra's (G, u, v)
Input: Directed Graph G = (N, A) with non-negative arc lengths c_ij, Source node s,
Sink node t
Output: Shortest path distance from the Source node s to the Sink node t


begin:
      S : = Φ; T : = Φ; S1 : = N; T1 : = N
      d(i) : = ∞ for each node i ϵ N
      d1(s) : = 0 and pred[s] : = 0
      d2(t) : = 0 and succ[t] : = 0

      while S1 ∩ T1 : = {} do
      begin
            For Forward Dijkstra's from source s:
            let i ϵ S1 be a node for which d1(i) = min{d1(j): j ϵ S};
            S : = S U {j};
            S1 : = S1 - {j};
            for each (i, j) ϵ A(i) do
                  if d(j) > d(i) + c_ij then d1(j) : = d1(i) + c_ij
                  and pred(j) : = i;
                  end if;
            end for;
            For Reverse Dijkstra's from sink t:
            let j ϵ T1 be a node for which d2(j) = min{d2(i): i ϵ T};
            T : = T U {i};
            T1 : = T1 - {i};
            for (u, v) in A do
                  if v = j then
                        for each (i, j) ϵ A(i) do
                              if d(j) > d(i) + c_ij then d(j) : = d(i) + c_ij
                              and succ(i) : = j;
                              end if;
                        end for;
                  end if;
            end for;
      end while if S1 ∩ T1 : = k;

      set current optimal distance : = d1(k) + d2(k)
      do
      for every i ϵ S and j ϵ T do
            compute op : = d1(i) + c_ij + d2(j)
            if op < current optimal distance then current optimal distance = op
            else current optimal distance : = d1(k) + d2(k)
            end if;
      end for;

      return current optimal distance;
end
```