1 P2P Network Design

- N - nodes in network
- L - layers in model
- $l_i = (x, y) : x, y \in [0, L); y > x$ - range of layers node i has
- ℓ_{ij} - latency from node i to node j
- c_i - computational cost of node i
- p_i - preload cost of node i
- e_i - embedding cost of node i

2 Optimal Path Finding

The P2P network is constructed as a directed graph. Edges are compatiable nodes with the next layers of the model. We know that this is a DAG becasue the layers must be preformed sequentially. The weights of the edges are the sum of latency and computational cost. The optimal path can be found using a topological sort. The start node is the users computer, and the terminal nodes are all nodes that have layer N. The optimal path connects the start node to a terminal node.

2.1 Topological Sort

Algorithm 1 Topological Sort of a Directed Acyclic Graph (DAG)

Require: A directed acyclic graph $G = (V, E)$
Ensure: A linear ordering of vertices such that for every directed edge (u, v), vertex u comes before vertex v

```
result[]                                   ▷ An empty list to store the sorted vertices
visited[false; V]                          ▷ A map to track visited vertices
temp[false; V]                             ▷ A map to track vertices in the current recursion stack
for each vertex v ∈ V do
    if visited[v] = false then
        DFS-Visit(G, v, visited, temp, result)
    end if
end for
return Reverse(result)
function DFS-Visit(G, u, visited, temp, result)
    temp[u] ← true                         ▷ Mark current vertex as being processed
    for each vertex v such that (u, v) ∈ E do
        DFS-Visit(G, v, visited, temp, result)
    end for
    temp[u] ← false                        ▷ Mark u as processed
    visited[u] ← true                      ▷ Mark u as visited
    Append u to result
end function
```

2.2 Dynamic Programming