COP290 Task 2 Subtask 2

Network Simulations

Gaurav Jain & T Abishek
2019CS10349 & 2019CS10407

1158 Words

28th July, 2021

1 Introduction

Network connectivity problems are abundant in computational biology research, where graphs represent a wide range of phenomena: from physical interactions between molecules to more abstract relationships such as gene co-expression. One common challenge in studying biological networks is extracting meaningful, small subgraphs from large databases of potential interactions.

In protein-protein interaction (PPI) networks, edges represent physical contact between proteins, either within stable multi-sub-unit complexes or through transient causal interaction. In other words, an edge (x, y) means that protein x can cause a change to the molecular structure of protein y and thereby alter its activity.

One of the problems in this domain is to find a possible subgraph of the PPI network that simultaneously satisfies all the protein nodes present in the set of active protein nodes, thereby explaining the overall biological activity.

2 Problem Formulation

The edges in the PPI network can be assigned a probability value (reflecting the credibility of their experimental evidence) by taking the negative log of these values as edge weights. The problem is formulated as:

Given a PPI network $G = (V, E, L)$ and a set of active proteins (S), find a subgraph $G' = (V', E')$ such that:

- $S \subseteq V' \subseteq V$.
- There exists a path between every pair of active proteins in G'.
- $\sum |L(E')|$ is minimized.

This problem can be mapped to the *Prize Collecting Steiner tree (PCST) problem*, where the task is to find a Steiner tree with maximal profit at a minimal cost. In our problem, the edge weight is the cost function. This cost function is to be minimized. There is no profit function involved in this problem. PCST is an NP-complete problem.

3 Hardness Analysis of the Problem

We have shown that this problem can be mapped to the Prize Collecting Steiner Tree problem. We will now lay the foundations for the proof of NP-completeness of the PCST problem.

- The first step is to show that PCST, Π is in NP.
- Choose a well-known and basic NP-complete problem Π'.
- Construct a transformation f from Π to Π'.
- Show that f is a polynomial transformation.

The complete proof is not presented in this report. Interested readers may read the *Steiner Tree NP-completeness Proof*.

4 Algorithms and Relevant Data Structures

The graph $G = (V, E, L)$ is stored in an **adjacency list** and as discussed above the edge weights are assigned according to their experimental probabilities. Thus, all edge weights are non-negative values.

4.1 A Greedy Algorithm based approach

$S \subset V$ is the set of terminal nodes to be visited. We can use an adaptation of a greedy approach of the Travelling Salesman Problem (TSP), which builds the solution greedily by choosing at each iteration the closest required node to the last node added to the walk.
In this approach, we start the walk with an initial vertex $v_1 \in S$. We maintain a **unordered-set** of visited nodes N and stores the current path P in a **linked-list**. At each iteration, the node $next$ to be visited is set to the closest among all yet unvisited required nodes.
The shortest path P' from $current$ to $next$ is computed using *Dijkstra's algorithm*. The partially built walk P is updated by appending the shortest path P' to it. The set of already visited required nodes N and the $current$ node are updated. Finally, after completing the loop, the shortest path from current to the initial node v_1 is appended to the walk.

In this problem, the greedy criterion is the choice of the nearest terminal node to be visited. This is an example of randomized greedy or semi-greedy problems. Semi-greedy algorithms act by replacing the deterministic greedy choice of the next element to be incorporated into the solution under construction by the random selection of an element from a restricted set of best candidate elements, called the restricted candidate list.
A simple quality-based scheme is used to define the restricted candidate list in this approach by finding the minimum and maximum sum of shortest path's edge weights between all the remaining unvisited terminal nodes.

Algorithm 1: Adapted greedy algorithm for PCST.

Select initial required node $v_1 \in S$;
$N \leftarrow v_1$;
$P \leftarrow v_1$;
$current \leftarrow v_1$;
while $N \neq S$ **do**
  $next \leftarrow$ closest node to $current$ among all those in $S \setminus N$ (Dijkstra's Algorithm);
  $P' \leftarrow$ shortest path from $current$ to $next$;
  $P \leftarrow P \oplus P'$;
  $N \leftarrow N \cup \{next\}$;
  $current \leftarrow next$;
end
$P' \leftarrow$ shortest path from $current$ to initial node v_1;
$P \leftarrow P \oplus P'$;
return P;

4.1.1 Dijkstra's Algorithm for Shortest Path

Dijkstra's Algorithm is used to find the shortest path between two vertices. We can implement this algorithm using a **Priority Queue** by storing all vertices and their priorities. The priority queue is initialized by setting source vertex's priority as 0 and other vertices' priority as ∞.
We also need to store the parent pointer in the priority queue to backtrack the path computed by the algorithm. The parent pointer for all vertices is set to NULL.
We will continue to run this algorithm until all terminal nodes are not processed. The processed nodes are dequeued from the priority queue and marked as visited. In each iteration, the vertex with minimum priority is processed.
All of the adjacent vertices of the dequeued vertex is checked and if they are not visited already their priorities are recalculated as the minimum of the current priority and the sum of the edge weight and dequeued vertex's priority. If the sum's value is less than the old priority, we update the parent of the adjacent vertex to the dequeued vertex.
The updated priorities after this step of computation will denote the minimum cost (edge weights) needed to go to a terminal node. We take a vertex $v \in S \setminus N$ which has the minimum cost and return the path by backtracking to the source node using the parent pointers.
The time complexity of this algorithm can be improved using a **Fibonacci Heap** as the priority queue instead of min-heap as both insertion and decreasing priorities takes $O(1)$ time which take place in every iteration.

4.1.2 Complexity Analysis

The deletion operation in Fibonacci Heap takes $O(logn)$ time and decrease priority function takes $O(1)$ time. As the Dijkstra's algorithm dequeues every vertex from the priority queue, the running time is $O(|V|log(|V|)$. We also decrease the priorities $2|E|$ times (sum of degree of all vertices). Thus, the overall time complexity for Dijkstra's algorithm is:

$$O(|V|log(|V|) + |E|)$$

The greedy approach to the PCST problem repeats the Dijkstra's algorithm $|S|$ times. So the overall time complexity of the implementation is:

$$O(|S|(|V|log(|V|) + |E|))$$

If we use the semi-greedy approach to this problem then the time complexity would increase as in each step we select a set of possible next vertex instead of selecting a single vertex. So the overall time complexity in this case could go upto:

$$O(|S|^2(|V|log(|V|) + |E|))$$

We can improve the above implementation by using dynamic programming to store the results of the Dijkstra's algorithm for each vertex $v \in S$. However in this case the space complexity would increase.

5 Simulation

We can simulate the PPI network using the available database compiled by various experiments. The PPI network of yeast is a famous and easily visualizable network with over 4000 vertices and over 20,000 edges. We can simulate different biological activities by selecting the right set of active proteins or terminal nodes.

We have illustrated a simple undirected PPI network with 20 nodes, and the size of the terminal nodes is 5. The edge weights and node labels are hidden for illustrative purposes.

The simulation can contain animations of the path taken by the simulator to go from one terminal node to another. The terminal nodes can be shown in different colors. The current node and the next node are shown in different colors.

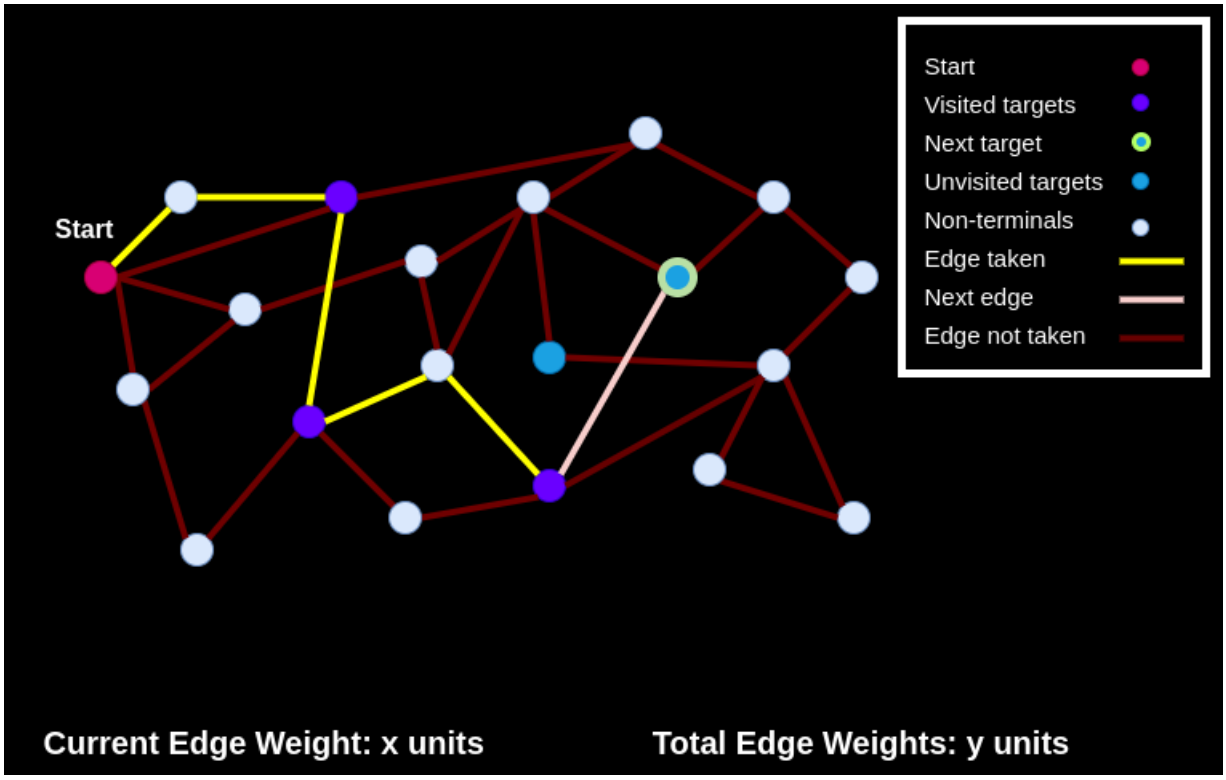

Fig. 1. Illustration of a Simulation.