COL351: Assignment 4

Vishwas Kalani (2020CS10411)
Viraj Agashe (2020CS10567)

September 2022

Contents

1 Hitting Set

Given a set $U = \{u_1, u_2 \cdots u_n\}$ and a collection of subsets $A_1, A_2 \cdots A_m$ of subsets of U. A set $S \subseteq U$ is a hitting set for the collection $A_1, A_2 \cdots A_m$ if $S \cap A_i \neq \phi$ for all i. The Hitting-Set Problem (HS) for the input $(U, A_1, A_2 \cdots A_m)$ is to decide if there exists a hitting set $S \subseteq U$ of size at most k.

1.1 Proving the problem in NP Class

For proving the problem is in NP class, we need to prove that there is a polynomial time verifier of the problem. The problem can be verified in polynomial time as follows :

Algorithm 1 Hitting set of size at most k

Input: The set S, and $(U, A_1, A_2 \cdots A_m)$
Output: If the set S is a valid hitting set of size at most k

```
if |S| > k then
    return False
end if
for each A_i in A_1, A_2 ... A_m do
    flag ← False
    for each x in A_i do
        if x ∈ S then
            flag ← True
            break
        end if
    end for
    if flag = False then
        return False
    end if
end for
return True
```

1.1.1 Time complexity of verification

- There are two nested for loops, with the first for loop contributing m iterations, the inner for loop contributing $|A_i|$ iterations which can be n in the worst case. Further checking if $x \in S$ takes $|S|$ time in the worst case that is k.
- Therefore the total time complexity for the verification is $O(mnk)$ and thus it is polynomial time verification.

1.2 NP completeness

For proving the NP completeness of this problem, we can convert the vertex cover problem to the hitting set problem as follows :

1. Consider a graph $G = (V, E)$ with a vertex cover C.
2. Let the set of vertices of the graph V be the elements of U that is $\{u_1, u_2 \cdots u_n\} = \{v_1, v_2 \cdots v_n\} = V$.
3. Let the set of edges of the graph E be the collection of the subsets that is $\{A_1, A_2 \cdots A_m\} = \{e_1, e_2 \cdots e_m\} = E$. Since each edge is of form (v_i, v_j) thus each $A_k = (u_i, u_j)$ for the converted problem, thus we have each $A_k \subseteq U$ for all k.

Theorem 1. *If there exists a vertex cover of size atmost k in the graph G then there exists a hitting set S of size at most k for the converted problem instance*

Proof. Consider a vertex cover C of size at most k in the graph and the set S for the hitting set problem to be C. If C is a vertex cover then by definition of vertex cover, it contains a vertex belonging to each edge of the graph therefore $C \cap e_i \neq \phi$, thus for the converted hitting set instance we have $S \cap A_i \neq \phi$ for all $1 \leq i \leq m$ and thus the set S is a hitting set of size at most k.

Theorem 2. *If there exists a hitting set of size atmost k for the converted problem instance then there exists a vertex cover of size atmost k in the graph G*

Proof. Consider a hitting set of size at most k in for the hitting set problem. That is we have a set S such that $S \cap A_i \neq \phi$ for all $1 \leq i \leq m$, that is the set S either contains u_i or both u_i and u_j from any set A_i. If we consider the same set to be the set C will have a vertex from each edge of the graph and thus C satisfies the condition of being a vertex cover of size atmost k.

From the above theorems, we see that Vertex Cover can be reduced to the Hitting Set Problem. The graph transformations performed take polynomial time. Since we have found a polynomial time reduction from an NP-Complete problem to Hitting Set Problem which is in NP, we have proven that the Hitting Set Problem is NP-Complete. ■

2 Tracking Shortest Paths

2.1 Part I: Proving Problem is in NP

Objective : To show the problem belongs to NP class, we need to have a polynomial time verifier for the given problem statement. Given a graph $G = (V, E)$, a source s, a sink t and a tracking set T, we need to verify whether the given tracking set is a valid tracking set in polynomial time.

2.1.1 Algorithm

1. Return false if the size of the given set is greater than k
2. Calculate the shortest distance between s and t in graph G using simple BFS algorithm. Let us call the shortest distance d_0
3. Calculate the shortest distance between all the pair of vertices using *Floyd Warshall algorithm* .
4. For each edge $e = (u, v) \in E$ check if : $distance(s, u) + 1 + distance(t, v) = d_0$ or $distance(s, v) + 1 + distance(t, u) = d_0$
   If the above condition is not satified, remove the edge e. Remove all the vertices with degree = 0 from the graph. The resulting graph contains all the edges and vertices which participate in any shortest path from s to t. Let us call the graph after the above reduction to be G'
5. For each vertex in the reduced graph G', label the vertex with its distance from s in a table L. For instance $L[s] = 0$ and $L[t] = d_0$.
6. **Conversion to a DAG :** For each edge in G' direct the edge from a lower label to a higher label i.e for an edge (u, v), direct the edge from u to v if $L[u] < L[v]$ otherwise direct it from v to u.
7. **Condition for the tracking set in a DAG :** Consider a set $T' = T \cup \{s, t\}$. For any two vertices (u, v) such that $(u, v) \in T'$, check if there exists atmost 1 path between u and v in the graph G' without using any vertex from the set $T \backslash \{u, v\}$. If T satisfies the above condition then it is a valid tracking set otherwise not.

2.1.2 Time complexity analysis

1. Calculating the shortest distance between s and t takes $O(m + n)$ time.
2. Application of *floyd warshall algorithm* takes $O(n^3)$ time.
3. Removal of the edges and then vertices which don't participate in any s to t shortest path takes $O(m)$ time.
4. Labelling each vertex with its distance from s takes $O(n)$ time.
5. Conversion to DAG means iterating through all the edges which takes $O(m)$ time.
6. For a tracking set T of size l, checking if it satisfies the condition mentioned in the algorithm takes $O(l^2(m + n))$ time.
7. Therefore we have an overall **polynomial time** complexity.

2.1.3 Proof

Theorem 3. *The conversion steps* $2, 3, 4$ *and* 5 *of the algorithm convert the undirected graph to a directed acyclic graph and each vertex and edge in the graph participates in a* $s - t$ *path.*

Proof. There is no edge between vertices such that $L[v_1] = L[v_2]$ (equal levels means equal distance of v_1 and v_2 from both s and t) in the reduced undirected graph after step 4 of the algorithm. Since we will have $d_0 = distance(s, v_1)+distance(t, v_1)$ therefore the edge can't satisfy either $distance(s, v_1)+1+distance(t, v_2) = d_0$ or $distance(s, v_2) + 1 + distance(t, v_1) = d_0$. Thus for any edge $\{v_1, v_2\}$ after step 4, either $L[v_1] > L[v_2]$ or $L[v_2] > L[v_1]$ and the difference between them will also be exactly 1.

After step 4, we have no edge or vertex which doesn't participate in a shortest $s - t$ path. We are then converting the undirected graph to a directed one using step 5. All the shortest $s - t$ paths are now $s - t$ paths in the new directed graph.

Let us assume there is a cycle in the graph obtained after the above mentioned steps of reduction and conversion thus there exists a path of form $v_0, v_1, v_2 \cdots v_0$. Since every edge as per the conversion step is between vertices from a lower level to a higher level therefore we have $L[v_0] < L[v_1] < \cdots L[v_0]$, which is a contradiction, therefore we can't have a cycle in the graph.

Theorem 4. *Let* $G = (V, E)$ *be a DAG with a source* s *and a sink* t *such that each vertex and edge participates in* $s - t$ *paths, then a given set* T *is a tracking set if and only if it satisfies the tracking set condition mentioned in step* 6 *of the algorithm*

Proof. Let T be a tracking set of the graph then first we prove that it satisfies the tracking set conditions. Let us assume that it doesn't satisfies the **condition** 7. Then there exists two vertices $v_1, v_2 \in T \cup \{s, t\}$ such that there are two paths P_1 and P_2 which don't have any vertex from the set $T\backslash\{v_1, v_2\}$. From the above **theorem** 3, we have that all the vertices participate in $s - t$ paths therefore we have a path from s to v_1, let us say P_1^s and similarly we have a path from v_2 to t let us say P_2^t. So the two shortest paths from s to t, $P_1^s.P_1.P_2^t$ and $P_1^s.P_2.P_2^t$ have the same intersection with the set T which is a contradiction to the fact that T is a tracking set.

Now let us prove the converse that is let a set T satisfies the **condition** 7, then we need to prove that it is a tracking set. Let us assume that it is not a tracking set. Thus we have two shortest $s-t$ paths P_1 and P_2 such that $T \cap P_1 = T \cap P_2$. Consider $T' = T \cup \{s, t\}$, and v_1 be the first vertex on path P_1 such that v_1 belongs to $P_1 \cap T'$ and similarly v_2 be the next vertex after v_1 on path P_2 which belongs to $P_1 \cap T'$. Since we have $T \cap P_1 = T \cap P_2$ therefore both v_1 and v_2 lie on path P_2 as well. Let P_1' be the path between v_1 and v_2 on path P_1 and similarly P_2' be a path between v_1 and v_2 on path P_2. Thus we have pair of vertices $v_1, v_2 \in T'$ which have more than one path between them which doesn't contain any vertex from the set $T\backslash\{v_1, v_2\}$ which is a contradiction hence T is a tracking set.Hence proved ■

2.2 Part II: Proving NP Completeness

To prove the NP Completeness of this problem, we reduce the vertex cover problem to the tracking shortest paths problem as follows:

1. Consider a graph $G = (V, E)$ which is an instance of the vertex cover problem.
2. From G, we construct a two layered graph $H = (V', E')$ where the vertex set contains the source and destination vertices s, t as well as a vertex corresponding to each vertex as well as each edge in the graph G. Further, we add some auxiliary vertices v^*, v^{**} and e^*.
3. For all vertices except p, q and v^*, e^*, connect v_i to e_j if the edge e_j is incident on v_i in the original graph G. Connect s to all vertices v_i $\forall i$ as well as v^*, v^{**} and e^*, and t to all vertices e_j as well as e^*. Further, connect v^* to e^* as well as all edge vertices e_j, v^{**} to e^* and connect e^* to t.

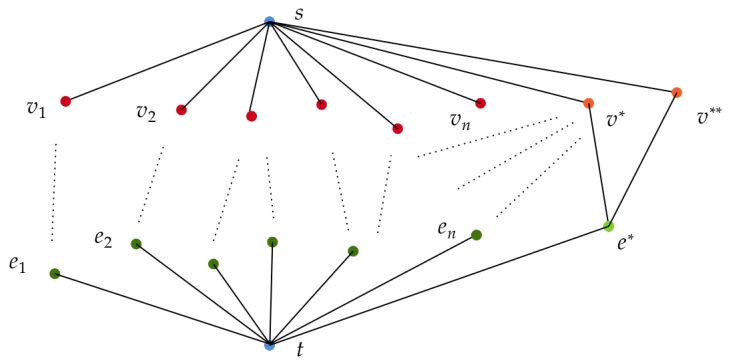

Figure 1: Transformed Graph $H = (V', E')$

Theorem 5. *If there exists a vertex cover of size k in the graph G then there exists a tracking set of size $k + |E| + 1$ in the converted problem instance H.*

Proof. Let the vertex cover of size k in the original graph be V^*. We claim that the set of vertices $T = V^* \cup E \cup \{v^*\}$ is a tracking set of the transformed graph H. Suppose this is not true. Then there must exist two different paths P_1 and P_2 such that $V(P_1) \cap T = V(P_2) \cap T$. Note that any shortest path in the graph H is of the form (s, v_i, e_j, t), therefore let the paths be $P_1 = (s, v_1, e_1, t)$ and $P_2 = (s, v_2, e_2, t)$.

Note that $e_1 = e_2$, otherwise the intersection of the paths with T would be different (since $E \subseteq T$ by construction). Then it must be true that $v_1 \neq v_2$. Call $e_1 = e_2 = e$, It must also be true that $v_1 \notin V^*$ and $v_2 \notin V^*$ otherwise the intersection of P_1 and P_2 with T would be different. However, since both v_1 and v_2 are connected to e, the edge e must be incident on both v_1 and v_2. Since V^* is a vertex cover of G, at least one of $v_1, v_2 \in V^*$, which is a contradiction. Thus, $V^* \cup E$ is a tracking set for H. ■

Theorem 6. *If there exists a tracking set of size $k + |E| + 1$ in the converted problem instance, then there exists a vertex cover of size k in the original graph G.*

Proof. We are given that the converted graph H has a tracking set T of size $k+|E|+1$. Note that so that the intersection between the paths of the form (s, a, e, t) to be different for all pairs, the vertices e corresponding to the edges and e^* must all be in the tracking set T, with possibly the exception of a single vertex, say e_k (which can have an empty intersection with T). If $e_k \neq e^*$, we can replace e^* with e_k in the tracking set, since there is only one $s-t$ path passing through the vertex e^* (all others have some intersection with T).

Further, so that the intersection of the paths (s, v^*, e^*, t) and (s, v^{**}, e^*, t) with T is different, either $v^* \in T$

or $v^{**} \in T$. Without loss of generality, suppose $v^* \in T$. We now claim that $T \setminus (E \cup \{v^*\})$ is a vertex cover of G. If this is not true, consider two uncovered vertices v_1, v_2 and e be the edge between them. Now since all edges are in T the intersection of the shortest paths (s, v_1, e, t) and (s, v_2, e, t) with T can only be different if at least one of v_1 or v_2 is in T and hence the vertex cover, which is a contradiction. Therefore, there exists a vertex cover of G of size $|T \setminus (E \cup \{v^*\})| = k$. ■

From the above theorems, we see that Vertex Cover can be reduced to the Tracking Shortest Paths Problem (TSPP). The graph transformations take polynomial time. Since we have found a polynomial time reduction from an NP-Complete problem to TSPP, which is in NP, we have proven that TSPP is NP-Complete. ■

3 Flows and Cuts

3.1 Flow Network Setup

To approach the problem, we first set up the flow network we will consider to solve this problem. Suppose the original graph is $G = (V, E)$. We construct the network graph G^* from G as follows (assuming $|E| = m$):

- Create two source and sink vertices s and t, and connect them to all $v \in V$.
- Corresponding to each undirected edge e between $v_i, v_j \in V$, create two directed edges e_{ij} and e_{ji}.
- Assign capacities to the edges as follows:
  - For all edges e_{ij} with $i, j \in V$, set the capacities to 1.
  - For all edges e_{si} with $i \in V$, set the capacities to m.
  - For all edges e_{it} with $i \in V$, set the capacities to $m + 2\alpha - d_i$, where d_i is the degree of the vertex v_i in the graph G.

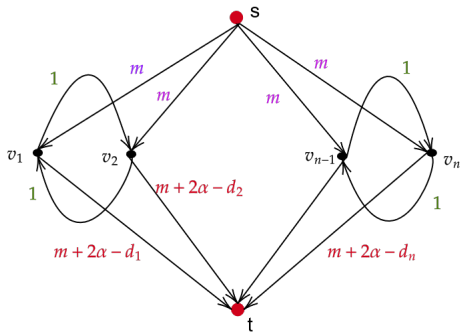

Figure 2: Flow network for the problem

3.2 Analysis of Network

We will now analyse the constructed network to solve the problem of finding a subset of density at least α. Consider any (s, t)-cut of the graph G^*. Note that the capacity of this cut is,

$$C(A, \bar{A}) = \sum_{j \in \bar{A}} c_{sj} + \sum_{i \in A, j \in \bar{A}} c_{ij} + \sum_{i \in A} c_{it}$$

Note that $c_{sj} = 2\alpha$, $c_{ij} = 1$ and $c_{it} = d_i$. Let us consider the vertices in the cut A other than s to be the subgraph S. Then,

$$C(A, \bar{A}) = m(|V| - |S|) + \sum_{i \in A, j \in \bar{A}} 1 + (m + 2\alpha)|S| - \sum_{i \in A} d_i$$

Now, note that $\sum_{i \in A} d_i = 2|E(S)| + |E(\bar{S})|$, where $|E(S)|$ is the number of edges with both endpoints in S, and $|E(\bar{S})|$ is the number of edges with exactly one endpoint in S, which is exactly equal to $\sum_{i \in A, j \in \bar{A}} 1$. Therefore, we get that,

$$C(A, \bar{A}) = m|V| + |E(\bar{S})| + 2\alpha|S| - 2|E(S)| - |E(S')| = m|V| + 2\left(\alpha|S| - E(S)\right)$$

We know that the maximum flow through the network is equal to the capacity of the minimum cut. Therefore, if there exists a cut with capacity less than or equal to $m|V|$, we can conclude that,

$$\alpha|S| - E(S) \leq 0 \implies \frac{E(S)}{|S|} \geq \alpha$$

i.e. there exists a subset of the graph G with density greater than or than α.

3.3 Part 1

3.3.1 Algorithm

Based on the above analysis, we therefore have the following algorithm for Part 1:

1. Construct the flow network G^* given the parameter α as described above.
2. Run the Edmond-Karp algorithm on the network G^* and calculate the maximum flow in the network.
3. If the maximum flow $f_{\max} \geq m|V|$, then there exists a cut $c(A, \bar{A})$ with capacity $\geq m|V|$. Remove s from the cut A. Then this is the required subset of G with density $\geq \alpha$, by the above analysis.

3.3.2 Time Complexity Analysis

We claim that the algorithm has a polynomial running time, more specifically, $O\left(mn(m+n)\right)$, where $|V| = n$ and $|E| = m$.

- The creation of the network flow graph G^* from G requires $O(m+n)$ time.
- Running the Edmond-Karp algorithm requires $O(mn(m+n))$ time.
- Comparing the value of the max-flow is an $O(1)$ operation.

Therefore, the overall running time of the algorithm is $O(mn(m+n))$.

3.4 Part 2

3.4.1 Algorithm

Furthermore, we observe that the density of all subgraphs of G can only take some finite values. By definition, the density ρ_S of a subgraph S is given by,

$$\rho_S = \frac{E(S)}{|S|}$$

Notice that $E(S)$ is an integer satisfying $0 \leq E(S) \leq |E| = m$, and further $|S|$ is also an integer satisfying $1 \leq |S| \leq n$. Therefore for any graph there are atmost $O(mn)$ values possible for ρ_S. Therefore, we can perform a search over these mn values and find the maximum value for which the max-flow in the graph G^* $f_{max} \geq m|V|$. Thus, the algorithm becomes the following:

1. Enumerate the possible values for ρ_S in the graph G and store them in a list, say L_ρ.
2. Perform a binary search over the values in L_ρ. For each value α examined, run the algorithm in Part 1 to analyse if there exists a subgraph with density of at least α or not.
3. Find the maximum value $\alpha_{\max}$ in the list L_ρ which satisfies the conditions from Part 1. Then, $\alpha_{\max}$ is the required solution.

3.4.2 Time Complexity Analysis

We claim that this algorithm is $O(mn(m+n)\log n)$. We argue this as follows:

- We perform binary search over a list of size mn. While examining each element, we take time $O(mn(m+n))$. We will examine at most $\log(mn)$ elements.
- Note that $mn \leq n^3$, since m is at most $\frac{n(n-1)}{2}$. Therefore, $\log(mn) = \log\left(n^3\right) = 3\log n$.

Therefore, the total time taken is $O(3\log n \times mn(m+n)) = O(mn(m+n)\log n)$