Steiner Tree

CS4054D - PARAMETERIZED ALGORITHM

Instructor: Mrs. Subhashini R
Compiled by: Simran Gangwani (M190359CA)

Content

1. Introduction

The definition, NP Completeness and the parameterized algorithm for Steiner Tree problem will be introduced.

1.1 The Steiner Tree[2]

After more than 26 years of war, emperor Qinshi Huangdi finally conquered the other six Chinese countries and built the first powerful centralised state in 221 BC. In order to consolidate and strengthen the empire, the emperor wanted to build a road-network which would connect some main cities of the seven countries. From the economic view, the newly established road-network should be as short as possible and be built on the basis of existing roads. However, which roads should be selected then? It was a big problem for Qingshi Huangdi in his time.

Actually, Qinshi Huangdi's problem can be transformed to a problem on an undirected graph G = (V, E, w), where V is the vertex set, E is the edge set, and w vector of edge weight. In this graph, each city is represented by a vertex v, the road between these cities are edges e, and the length of each road is the weight of the corresponding edge. The main cities that need to be connected from a subset T of V . The set T is called terminal set and the points in this set are called terminal points/terminals. Then Qinshi Huangdi's problem can be treated as a kind of Steiner tree problem (STP) which is defined as follows.

Definition . *Given an undirected graph G = (V, E, w) and a subset T of V , the Steiner tree problem is to find the shortest subtree (the shortest sum weight of edges in this tree) of G which connects all the points in T .*

1.2 Applications

Steiner trees arise in network design problems, because the mini- mum Steiner tree describes how to connect a given set of sites using the smallest amount of wire. Analogous problems occur when designing networks of water pipes, heating ducts, or communication devices. Typical Steiner tree problems in electronic circuit design require connecting a set of sites to (say) ground under constraints such as material cost and signal propagation delay.

1.3 Special Cases

Some special cases can be solved efficiently, which includes the following:

- If $|T| = 2$, STP is a shortest path problem between the two given terminals. The solution can be obtained in polynomial time in the input size.

- If $|T| = n$, STP is a minimal spanning tree problem (MSTP) of G. Here too it is still a problem to find a shortest tree that connects all the vertices. The solution can be obtained in time O(m + nlogn) as well.

1.4 Versions of Steiner Tree Problem

Following includes the three versions of Steiner Tree problem:

Optimisation version Instance : *Given an undirected graph G = (V,E) with non negative edge weights and a subset of vertices T.* Question : *Find the smallest tree connecting all the vertices of T(minimum weight Steiner Tree)* Classical Complexity: *NP - Hard*
Decision Version Instance: *Given an undirected graph G = (V,E) with non negative edge weights, a subset of vertices T, and a natural number K.* Question: *Does there exist a Steiner Tree spanning T that contains at most K edges.* Classical Complexity: *NP - Complete*
Parameterised Version Instance : *Given an undirected graph G(V,E), subset of vertices T.* Parameter : $K = \|T\|$ Question : *A tree, subgraph of G, spanning T, which minimizes the sum of its edge-weights.* Classical Complexity : $O(3^k \cdot n^{O(1)})$

2. NP Completeness

2.1 The Class NP Complete[2]

A problem $\Pi_1 = (I_1, I_1^+)$ is polynomially reducible to another problem $\Pi_2 = (I_2, I_2^+)$ if there exists a polynomial deterministic algorithm which can transform an instance $I_1 \in I_1$ into an instance $I_2 \in I_2$, such that $I_2 \in I_2^+$ if and only if $I_1 \in I_1^+$. If a problem Π_1 is polynomially reducible to a problem Π_2, the algorithm for solving Π_2 can be used to solve Π_1 as well through transforming the instance of Π_1 into the instance of Π_2.

The definition of the class N P − complete can be described as follows. A problem Π is NP − complete if $\Pi \in$ NP and all other NP problems are polynomially reducible to Π. Since the polynomial reduction is transitive, an NP problem Π would be in NP − complete if there exists an NP − complete problem which is polynomially reducible to Π.

The decision version of the Steiner tree problem is one of the Karp's original 21 NP −complete problems.This document shows the proof of NP Completeness of Steiner Tree via candidate problem *Exact cover by 3 sets.*

2.2 Proof of NP Completeness[1]

To show that decision version of Steiner Tree(ST) ∈ NP-Complete, we need to show that:

- ST ∈ NP
- ST ∈ NP-Hard

To show that ST ∈ NP, it is enough to show that the YES-instance of ST are verifiable in polynomial time; and to show that ST ∈ NP-Hard, it is enough to show that there exists an NP Complete problem Π such that $\Pi \leq_p$ ST.

To show that $\Pi \leq_p$ ST, we need to design a polynomial-time algorithm say R, given an instance I_1 of Π.

- R(I_1) Transform instance of P into instance of ST.
- I_1 is an YES instance of P iff R(I_1) is an YES instance of ST.

The above two point highlights that it is not enough to convert the instance of Π into an instance of ST, but also be able to convert the solution of ST back to a solution for Π.

2.2.1 Steiner Tree in NP

So firstly we want to be sure that the ST problem is actually in NP. Given a graph G = (V,E) ,T subset of V, and K.

Assume ⟨G, T, k⟩ ∈ ST ,define a certificate H to denote a subgraph H = (V′, E′), we can check in polynomial time that:

1. H is a subgraph of G(V' is contained in V and E' is contained in E)
2. H is really a tree: it contains no cycles and it is connected.
3. The tree H touches all the terminals specified by the set T; T is a subset of V'.
4. the number of edges used by the tree is no more than k i.e |E'| <= k.

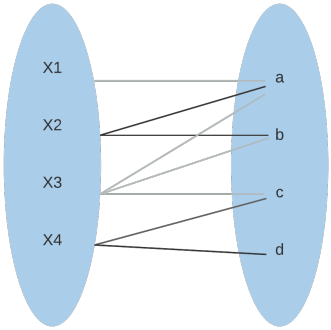
2.2.2 Reduction

As mentioned earlier our candidate problem is Exact Cover by 3 set. Before defining Exact Cover by 3 set, let's understand what is Exact Cover.

Exact Cover Problem Statement[4]

Input: *a family S, subset of a set X.*
Question: *Does there exist a sub family $S^* \subset S$,such that each element in X is* ***contained*** *in exactly one subset in S*.*

For example:
Let X = {a, b, c, d} and S = {X_1, X_2, X_3, X_4} where,
X_1 = {a}
X_2 = {a, b}
X_3 = {a, b, c}
X_4 = {c, d}

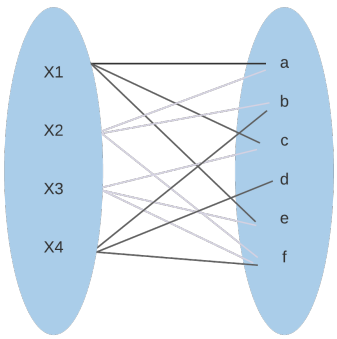


Because X1,X_2 and X3 are the only subsets which contains 'a', exact cover must contain any one of them. X2 and X3 are the only subset that contain 'b', so exact cover may contain any one among X2 or X3; if exact cover contain X2 then it cannot contain X3 or X1 as 'a' is common. 'C' is common in X3 and X4 but since 'd' is present only in X4 we have to include that in exact cover.
Hence the solution set S* = {X_2, X_4}

Exact Cover by 3 Set (X3C) Problem Statement

Input: Given a set X, with |X| = 3q and S of 3-element subsets of X.
Question: Does there exist, a sub family S* of S such that each element in X is **contained** in exactly one subset in S*.

For example:
Let X = {a, b, c, d, e, f}, |X| = 3x2
and S = {X_1, X_2, X_3, X_4} where,
X_1 = {a, c, e}
X_2 = {a, b, f}
X_3 = {c, e, f}
X_4 = {b, d, f}
Then S* = {X_1, X_4}



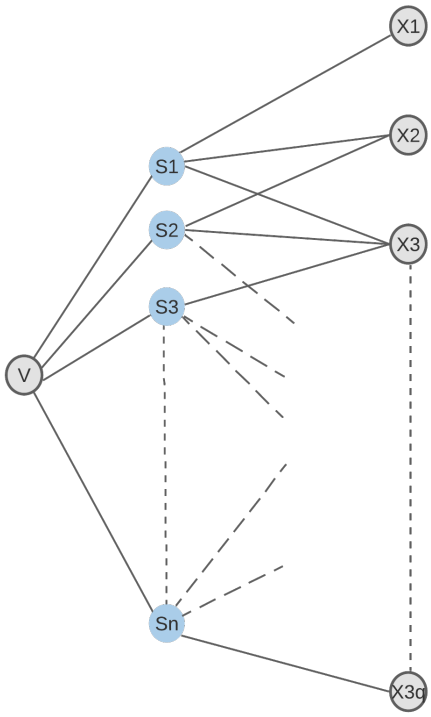
Note that, where S* is a solution which certificate that $\langle X, S\rangle \in$ X3C then:
1. The members of the solution S* form a partition of the set X
2. $|S^*| = q$.

Transform X3C to ST
Given an instance of X3C, defined by the set $X = \{x_1,\cdots,x_{3q}\}$ and a collection of 3-element sets $S = \{s_1,\cdots,s_n\}$, we have to build the ST instance specifying the graph G = (V,E), the set of terminals T, and the upper-bound on the spanning tree size k.

- Define the set of vertices v as:
  $V(G) = \{v\} \cup \{s_1,\cdots,s_n\} \cup \{x_1,\cdots,x_{3q}\}$.
  We add a new vertex v, so as to make graph connected, as without it the graph is bipartite; vertices of S are not connected with each other.

- Define the set of edges E as:
  $E(G)=\{vs_1,\cdots,vs_n\}\cup\{\cup s_i x_j \text{ for all } x_j \in s_i\}$.
  There is an edge from v to each node c_i, and an edge $c_i x_j$ if the element x_j appears into the set C_i of the X3C instance.
- Define the set of Terminal Vertices T as:
  $T = \{v,x_1,\cdots,x_{3q}\}$
- $k = |4q|$

It's easy to see that the reduction from X3C to ST can be done in polynomial time.The graph constructed according to these rules is shown in the figure below.



We can see that vertex 'v' serves it purpose of connecting the graph. The grey nodes represents the terminal vertices.

⟨X, C⟩ belongs to X3C if and only if ⟨G, R, k⟩ belongs to ST

Now we are going to prove that there exists a Steiner tree with no more than k edges if and only if there is an exact cover for the X3C instance of the problem.
For that we have the following lemma:
Lemma: ⟨X, C⟩ belongs to X3C if and only if ⟨G, R, k⟩ belongs to ST.
Proof: We split the proof into two parts, one for each implication.

- X3C⇒ST

  Suppose there is an exact 3-cover S* for the X3C problem. Clearly, S* uses exactly q subsets. Without loss of generality suppose they are $s_1,\cdots,s_q$ (if it's not the case, we just have to relabel them). Then the tree consisting of edges:

– $vc_1,\cdots,vc_q$
– s_ix_j, if $x_j \in C_i$ and $1 \le i \le q$

is a Steiner tree solving the problem with $q + 3q = 4q = k$ edges. So, if there is an exact 3-cover, then there is a Steiner tree using no more than k edges.

- X3C⇐ST

  Suppose now there exists a Steiner tree H with at most 4q edges. Since H is a tree, it has at most 4q + 1 nodes. According to the definition of Steiner tree, ST must also touch the terminal nodes $x_1, \cdots, x_{3q}$ and v, so the number of s-nodes in H, nc, satisfies:

  nc <= (4q+1) - (3q+1) (Total number of nodes, minus |T| = |X|+|v|)
  = q.

But the degree of s-nodes (considering only the edges toward x-nodes) is 3, so it is impossible to hit all the 3q x-nodes if the tree contains less than q s-nodes. Since the tree must also contain X and v, we have that the tree must contain at least $3q + q + 1 = 4q+1$ nodes.Any tree with 4q+1 nodes has 4q edges. Without loss of generality suppose these nodes are $s_1, \cdots, s_q$ then the solution S* of the X3C problem is given by the set

$S^* = \{s_1,\cdots,s_q\}$.

This concludes the proof.

3. Parameterized Algorithm

As a kind of exact algorithm for a general NP – hard problem, the parameterized algorithm was firstly investigated by Downey and Fellows in 1990. This algorithm is to find a solution in polynomial time with respect to the input size and in exponential time with respect to a (small) parameter k. It is also called "fixed parameter tractable" (FPT) algorithm, and a problem allowing such an algorithm is called FPT problem.

Downey and Fellows show that STP is fixed parameter tractable in the sense that it can be solved in time O(f(k)g(n)), where f is an exponential function, g is a polynomial function , n is the number of vertices (or edges), and k is a smaller value than n (number of the terminals or other parameter), n and k are the number of vertices and number of terminal vertices respectively. Then, the algorithms for STP is in time f(k) times a polynomial in n. This complexity is always written as $O^*(f(k))$, since the algorithm only takes into account the exponential growth of f(k). For example, the complexity $O(n^9 \cdot 2.684^k)$ is written as $O^*(2.684^k)$, since the polynomial time part can be suppressed.

3.1 The Dreyfus - Wagner algorithm for STP

A well known exact algorithm (parameterized algorithm) for the Steiner tree problem is the Dreyfus-Wagner Algorithm in time $O^*(3^k)$.

The Dreyfus-Wagner algorithm is a dynamic programming algorithm. It computes the length of a minimum Steiner subtree for a given terminal set T ($\subset$ V) from its minimum subtrees.

3.3.1 Problem Statement and notations

Let G be an undirected graph on n vertices and $T \subseteq V(G)$ be a set of Terminal vertices and a weight function $w: E(G) \rightarrow R_{>0}$.

Goal is to find a Steiner Tree for T in G forming a connected subgraph H of G containing T, i.e $T \subseteq V(H)$, whose weight w(H) is minimised.

Parameter = k, where $k = |T|$.

For a pair of vertices $u, v \in V(G)$, by dist(u, v) we denote the cost of a shortest path between u and v in G. for any two vertices u, v, the value dist(u, v) is computable in polynomial time, say by making use of Floyd Warshall Algorithm.

3.3.2 Assumptions

As pre processing steps we assume the following:

1. **K > 1**, otherwise solution is trivial.

2. **G is connected**: a Steiner tree for T exists in G only if all terminals of K belong to the same connected component of G and, if this is the case, then we may focus only on this particular connected component.
3. **Terminal node in T is of degree exactly 1 and no two terminal vertices are adjacent:** To achieve this property, for every terminal $t \in T$, we attach a new neighbour t' of degree 1, that is, we create a new vertex t' and an edge tt' of some fixed weight, say 1.

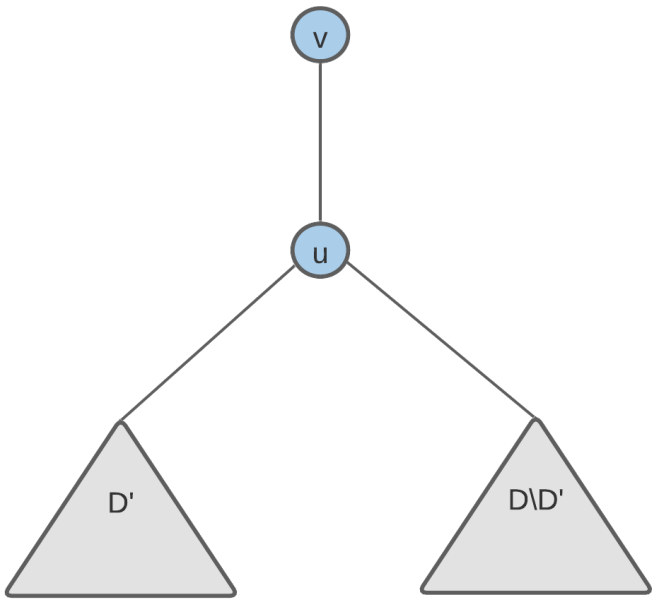
3.3.3 Algorithm

Ideology behind the algorithm

The crucial observation is as follows. The optimal Steiner Tree ST $(D \cup v)$, where $D \subseteq T$. Then v is joined in ST(D∪v) to some node u of ST(D∪v) along a shortest path P_{vu}, such that u is a steiner node in ST(D∪v). In both cases we have ST(D∪v) = P_{vu}∪ST(D∪u). In case u is a Steiner node, it splits ST(D∪u), i.e., we can decompose

$ST(D \cup u) = ST(D' \cup u) \cup ST(D \setminus D' \cup w)$.

$ST(D \cup v) = \min P_{vu} \cup ST(D \cup u) \cup ST(D \setminus D' \cup u)$. (1)



The diagram shows the visualisation of the equation (1)

Where Steiner tree $ST(D \cup u)$ and Steiner Tree $ST(D \setminus D' \cup u)$ represents the subtrees of Steiner Tree rooted at v $ST(D \cup v)$.

The algorithm follows as

Step 1: compute shortest distance for all u,v ∈V.
// by using Dijkstra Algorithm or Floyd Warshall.

// Base Case (|D| = 1)

Step 2: If D = {x} for some x ∈ T then for every v ∈ V we set ST({x},v) = dist(x,v).

// Recursive Case (|D| > 1)
Step 3: Each vertex in T can be either in D′ or D\D′, compute the minimum Steiner Tree formed with terminal vertices in D′ and D\D′ using the following recurrence relation.
T [D, v] = min {T [D′, u] + T [D \ D′, u] + dist(v, u)} .
Where u∈V (G)\T and D' is a non empty subset of D

3.3.4 Correctness of Recurrence Relation[3]

For every D ⊆ T of size at least 2, and every v ∈ V (G) \ T, the following holds

T [D, v] = min {T [D', u] + T [D \ D', u] + dist(v, u)}

We prove the equation by showing inequalities in both directions.

==>

For all u ∈V(G) and for all non empty subset D′ of D. Let H1 be a Steiner Tree for $D' \cup \{u\}$ in G of minimum possible weight, similarly we define H2 for the value T [D \ D', u]. So these two components H1 and H2 are connected by u, and let P be the shortest path between u and v.

We observe that components H1, H2 and P are connected and forms a subgraph D∪{v}.

Weight of D∪{v} = $w(H_1 \cup H_2 \cup P) \leq w(H_1)+w(H_2)+w(P) = T[D', u]+T[D\D', u]$ +dist(v, u).

≤ min {T [D', u] + T [D \ D', u] + dist(v, u)}, Where u∈V (G)\K =D' is a non empty subset of D.

<==

Let H be a Minimum Steiner Tree rooted at v as D ∪ {v} in G.

Let u be the vertex in G which has children >=2, and is closest to root, we can say this because |D| >= 2, so we have a subset of terminal vertices which has more than two terminal vertices and u can be ancestor of the vertices in D.

As we have assumed that degree of terminal vertices = 1. Therefore, u doesn't belongs to T.

We can decompose the H into three subgraphs:

1. P, a path between u and v.
2. H1, subtree rooted at u.
3. H2, subtree rooted at u(i.e it contains edges excluding edges in H1)

Let D′ = V(H1) ∩ T be the terminals in H1 and D\D′ = V(H2) ∩ T. Since every terminal node is of degree exactly one, there exist no edge between D′ and D\D′.

Also, D′ is a non empty subset of D, otherwise H\H2 isaSteiner tree for D ∪ {v} in G.

Since we assumed that H is an optimal Steiner Tree we can conclude that w(H1) = T [D′, u0], w(H2) = T [D \ D′, u0] and, moreover, P is a shortest path between u0 and v.

Therefore, T [D, v] = w(H) = T [D′, u0] + T [D \ D′, u0] + dist(v, u0)

$$\geq \min \{T [D', u] + T [D \setminus D', u] + dist(v, u)\} .$$

Where u∈V (G)\K =D′ is a non empty subset of D.

3.3.5 Complexity Analysis[3]

Analysing the algorithm, we can compute shortest path in polynomial time.

Base case can be implemented in linear time with respect to K.

Recursive case can be computed in time $2^{|D|}n^{O(1)}$.

Consequently all values of ST[D,v] are computed in time.

$$\sum_{v \in V(G) \setminus K} \sum_{D \subseteq K} 2^{|D|} n^{\mathcal{O}(1)} \leq n \sum_{j=2}^{|K|} \binom{|K|}{j} 2^j n^{\mathcal{O}(1)} = 3^{|K|} n^{\mathcal{O}(1)}$$

3.2 Unweighted Steiner Tree using Inclusion Exclusion Principle

We will use the intersection version of Inclusion exclusion principle which says as follow,Let A1,A2,...,Ak be subsets of a universe U, and let Bi = U \ Ai. Then

$|Ai| = (-1)^{|X|} |\cap j \in X Bj|$ where $i \in [k]$ $X \subseteq [k]$

3.2.1 Problem Statement

Instance: *Given an unweighted graph G = (V, E), and a subset of vertices T ⊆ V (G), called terminals, and a number l ∈ N.*
Parameter: $k = |T|$
Problem: *determine whether there is a tree H ⊆ G (called the Steiner tree) with at most l edges that connects all the terminals.*

Time Complexity: can be solved in $2^k \cdot poly(n)$ time.

3.2.2 Ideology

First we try to think of a *counting problem* of Steiner Tree which follows as:

If the number of l -edge subtrees of G containing K is non-zero, then a Steiner Tree on edges exists.

Since counting trees is hard, so we count an easier object called Branching walks. We count branching walks using Inclusion- Exclusion principle.

3.2.3 Terminologies[5]

Ordered Rooted Tree: A tree H where vertices have been labeled by {0, 1, 2, 3 . . . , $|V(H)| - 1$} via a DFS. Alternatively, every internal node of H has an ordering among it's children.

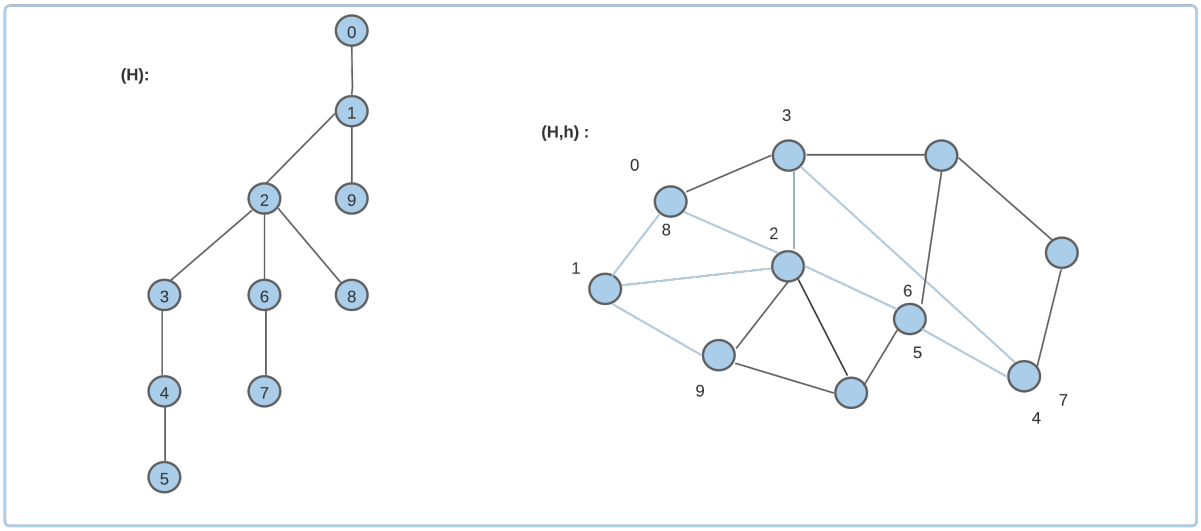
Let $r \in V(H)$ denote the root of H.

Branching Walk: A Homomorphic Image of an ordered rooted tree in G. It is a pair B = (H,h) where H is an ordered rooted tree, and $h : V(H) \rightarrow V(G)$ is a map such that if $(x, y) \in E(H)$ then

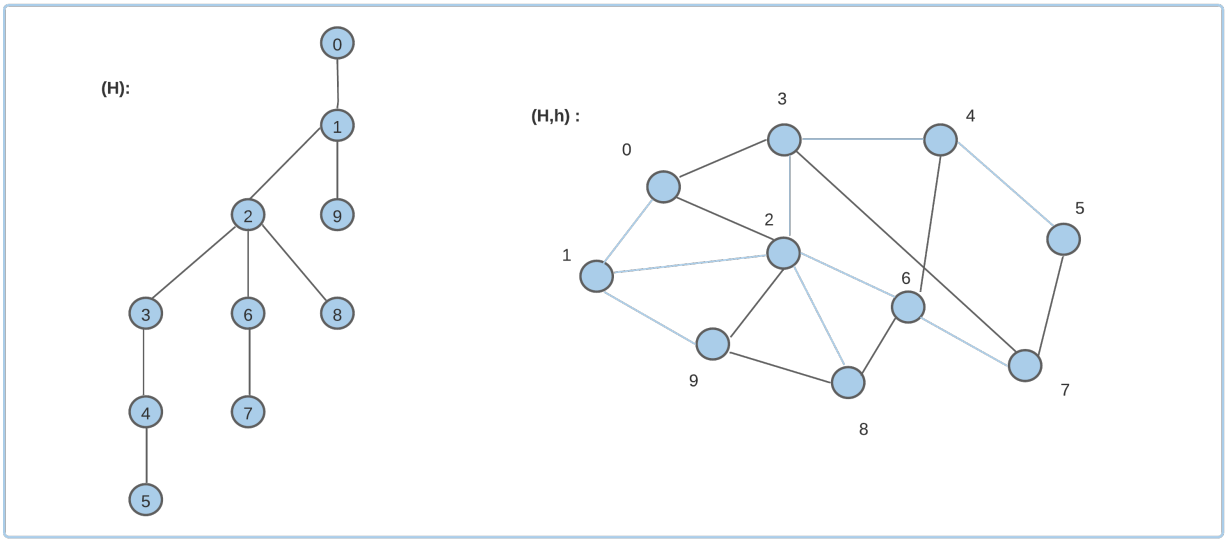
$(h(x), h(y) \in E(G)$.

Let $V(B) = \{h(x) \mid x \in V(H)\}$, and s = h(r) be the root of B.

Following example portrays the difference between non injective and injective homomorphism.



Non - injective Homomorphism



Injective Homomorphism

3.2.4 Observations and lemmas[3]

Observation: *Fix a terminal $s \in T$ as the root. G contains a Steiner Tree on <= l edges if and only if there is a Branching Walk B = (H, h) from s such that $T \subseteq V(B)$, and $|E(H)| = l$.*

Counting walks from s

- Universe U = all branching walks of length l from s.
- For each $v \in T$, $A_v = \{B \in U \mid v \in V(B)\}$
- Clearly $|\cap_{v \in K} A_v| != 0$ iff there is a Steiner Tree.
- Sufficient: Given $X \subseteq T$ compute $|\cap_{v \in X} B_v|$ where $B_v = U \setminus A_v$. $\cap_{v \in X} B_v$ is the set of all branching walks that avoid X.

- They lie in the graph G - X, so enough to count all Branching walks from s in the graph G - X of length l.

Lemma: *for every* $X \subseteq T$, $|\cap_{v \in X} B_v|$ *this can be computed in polynomial time using* $O(ml^2)$ *arithmetic operations.*

- Let G′ = G - X. It contains all Branching walk avoiding X.
- For a ∈ V(G′) and j <= l, let $b_j(a)$ denote the number of branching walks from a of length j in G′.
- if s/∈ V (G′) we should return 0, and otherwise our goal is to compute $b_l(s)$.
- compute $b_j(a)$ for all a ∈ V(G′) and j ∈ {0,...,l} using dynamic programming with the following recursive formula:

$$b_j(a) = \begin{cases} 1 & \text{if } j = 0, \\ \sum_{t \in N_{G'}(a)} \sum_{j_1 + j_2 = j-1} b_{j_1}(a) b_{j_2}(t) & \text{otherwise.} \end{cases}$$

- Once we have numbers $|\cap_{v \in X} B_v|$ for every X ⊆ T, we can compute the number of Steiner Tree via the inclusion - Exclusion formula, specified in section 3.2

The unweighted Steiner Tree problem can be solved in $2^k(nk)^{O(1)}$ time and polynomial space.

References

[1] Steiner Tree NP-completeness Proof by Alessandro Santuari May 7, 2003.
[2] EXACT ALGORITHMS FOR THE STEINER TREE PROBLEM, DISSERTATION, to obtain, the degree of doctor at the University of Twente, on the authority of the rector magnificus, prof. dr. W. H. M. Zijm, on account of the decision of the graduation committee, to be publicly defended, on Wednesday 25th of June 2008 at 15.00 hrs by Xinhui Wang.
[3] Marek Cygan, Fedor V. Fomin Łukasz Kowalik, Daniel Lokshtanov Dániel Marx Marcin Pilipczuk Michał Pilipczuk, Saket Saurabh, Parameterized Algorithms.
[4] https://en.wikipedia.org/wiki/Exact_cover
[5] Łukasz Kowalik University of Warsaw FPT School, Będlewo, August 2014