Advanced Graph Algorithms Jan-Apr 2014

Presentation 3 — 25 APRIL, 2014

Lecturer: Aditya Potukuchi *Scribe: Aditya Potukuchi*

1 Overview

In this lecture, we shall see an FPT algorithm for counting the number of Steiner Trees and Hamiltonian paths parameterized by treewidth. We start off with the definitions of the decision variants of these problems.

Definition 1 (Steiner Tree Problem)**.** *Given a graph G, a set $T \subseteq V(G)$ (called the Terminals), and an integer k, is there a tree $H \subseteq G$, such that $T \subseteq V(H)$ and $|E(H)| \leq k$?*

Definition 2 (Hamiltonian Cycle Problem)**.** *Given a graph G, is there a cycle that contains all the vertices of G?*

The counting variants of these problems simply ask to return the number of Steiner Trees/Hamiltonian Cycles in a graph G. We shall arrive at $2^{\mathcal{O}(t)}n^{\mathcal{O}(1)}$ algorithms, which shows us that the counting variants of these problems are not much more difficult than their decision variants for graphs of bounded treewidth (in fact, we shall see a deterministic algorithm for deciding steiner tree in the next class that has a worse running time!).

The method presented in this lecture is fairly general and can be applied to counting those edge sets in graphs which have a structure close to that of trees

2 Counting Connected edge sets

2.1 A convenient formulation for trees

Consider an arbitrary orientation of a graph G, i.e. give directions arbitratily to all it's edges. Let A be the incidence matrix of G, i.e. A has n rows ($n = |V(G)|$), m columns ($m = |E(G)|$), the entries are as follows:

$$a_{v,e} = \begin{cases} 0 & \text{if edge } e \text{ is not incident on vertex } v \\ 1 & \text{if } (u,v) \in E(G) \\ -1 & \text{if } (v,u) \in E(G) \end{cases}$$

Here, we state and prove a lemma which is used in the proof of the Matrix Tree Theorem (for undirected graphs).

Lemma 3. *For any $v' \in V(G)$, let F be the matrix A with the row corresponding to v' removed, and for any $S \subseteq E(G)$, let F_S be the matrix restricted to columns in S. If a set of edges $S \subseteq E(G)$ form a tree, then $det(F_S) = \pm 1$.*

Proof. If the egdes in S contain a cycle, the columns corresponding to the cycle in F are linearly dependent, therefore it is only required that we prove the other direction, i.e. If the edges in S form a tree, then $det(F_S) = \pm 1$. To prove that, we shall prove that there is exactly one mapping from vertices to edges in S that results in a non-zero term.

Recall that for a matrix $M_{m \times m}$, it's determinant is given by

$$det(M) = \sum_{\sigma \in S_n} (-1)^{sgn(\sigma)} \prod_{i \in \lfloor m \rfloor} m_{i,\sigma(i)}$$

For any mapping σ that maps a vertex v to an edge that is not incident on it, $\sigma(v) = 0$. Hence, in the tree rooted at v', every vertex is either mapped to the edge connecting it to it's parent, or one of the edges connecting it to its children. Let v_1 be a vertex mapped to edge v_1v_2 such hat v_2 is a child of v_1. If v_2 is a leaf of the tree, then there is no edge it can be mapped to to give a non zero term. Otherwise, v_2 is mapped to some edgev_2v_3 where v_3 is a child of v_2. Let v_i be the last vertex in this path. If v_i is a leaf, then again, there is no edge is can be mapped to for a non-zero term, otherwise, v_i must be mapped to an edge v_iv_{i+1} where v_{i+1} is some child of v_i, thus contradicting our assumption that v_i is the last vertex in the path. Thus, the only mapping which produces a non-zero term, produces ± 1, since entries of the matrix are ± 1. □

We shall use this lemma to express the number of Striner Trees/ Hamiltonian Cycles as a sum, and build up the sum using Dynamic Programming on the Tree Decomposition. But before we start, we enrich our notion of a tree decomposition given in *Lecture 4* with another kind node called *introduce edge* node. The idea is that in an *introdice vertex* node, we do not introduce any of the edges that are incident upon the vertex, we have a separate bag for each edge. However, when a vertex is forgotten, all the edges incident upon it are forgotten, we do not require a special bag. Also, each edge is introduced exactly once.

Definition 4 (Nice Tree Decomposition)**.** *A nice tree decomposition of a graph is a tree decomposition (T, χ) (*Lecture 3*) of a graph G with 5 kinds of nodes:*

Leaf Node: *a leaf i of the tree T with $B_i = \emptyset$.*

Introduce Vertex Node: *a node i having one child j, with $B_i = B_j \cup \{v\}$, for some vertex $v \in V(G)$. We say i* introduces *v.*

Introduce Edge Node: *a node i with one child j, $B_i = B_j$, and further, i is labeled with an edge $uv \in E(G)$. We say i* introduces *edge uv.*

Forget Node: *a node i having one child j, with $B_j = B_i \cup \{v\}$, for some vertex $v \in V(G)$. We say i* forgets *v.*

Join Node: *a node i with two children j and k such that $B_i = B_j = B_k$*

Every edge is introduced exactly once.

Claim 5. *Given a Tree Decomposition as in* Lecture 5*, i.e. with only the first* 4 *kinds of nodes in the above definition, there is an algorithm that, in time* $\mathcal{O}(n^2)$ *returns a Nice Tree Decomposition as defined above.*

Proof of the above left as exercise.

Before we start the algorithm, we create one more tool related to inversions in a mapping. For any two sets $A, B \subseteq U$, we define $inv(A, B) = |\{(a, b) \mid a \in A, b \in B, a > b\}|$. For ordered sets A, B and a function $f : A \xrightarrow{1-1} B$, we have the signature of f, given by $sgn(f) = -1^{|\{(a_1,a_2)|a_1>a_2, f(a_1)<f(a_2),\}|}$, i.e. signature of mapping is the number of inversions in the mapping.

Lemma 6. *Let* $X_l, X_r \subseteq V$ *and* $Y_l, Y_r \subseteq E$*, such that* $X_l \cap X_r = \emptyset$*,* $Y_l \cap Y_r = \emptyset$ *and for every* $e_1 \in Y_l$ *and* $e_2 \in Y_r$*,* $e_l < e_r$*. Further, let* $f_1 : Y_l \xrightarrow{1-1} X_l$ *and* $f_2 : Y_r \xrightarrow{1-1} X_r$*. Let* $f = f_1 \cup f_2$*, i.e.* $f(v) = f_1(v)$ *is* $v \in Y_l$ *and* $f(v) = f_2(v)$ *if* $v \in Y_r$*. Then* $sgn(f) = sgn(f_1)sgn(f_2)(-1)^{X_l, X_r}$

Proof. The number of inversions in f is given by the number of inversions in f_1, the number of inversions in f_2, and the number of such pairs (a, b) such that $a \in Y_l, b \in Y_r$ and $f(a) > f(b)$. This is nothing but $inv(X_l, X_r)$. Therefore, $sgn(f) = sgn(f_1)sgn(f_2)(-1)^{inv(X_l,X_r)}$ □

To use the above lemma, we need some kind of ordering on the edges and vertices of the graph (we will be looking at mappings of edges onto vertices), preferably, with respect to the Nice Tree Decomposition. So, we shall consider *postorder* numbering on vertices in the order they are forgotten in the tree decomposition (every vertex is forgotten exactly once), and on edges, in the order they are introduced in the tree decomposition (every edge is introduced exactly once). The reason for this shall become clear eventually.

Now, we are ready to start counting.

3 Counting Hamiltonian Paths

3.1 Constructing Partial Solutions

The idea is to use *Lemma 3* used above to express the number of Hamiltonian Cycles as a sum that can be computed by DP over the tree decomposition. Though Hamiltonian cycles are not exactly trees, we observe if any of the edges is removed, it is a spanning tree of a graph where the degree of each vertex is at most 2. Let B be the number of Hamiltonian Cycles

$$B = \sum_X [\text{X is a Hamiltonian Cycle}]$$

$$= \frac{1}{n} \sum_{\substack{X \subseteq E(G) \\ \forall v, deg_X(v)=2}} \sum_{\substack{S \subseteq X \\ |S|=|X|-1}} [(V, S) \text{ is a tree}]$$

$$= \frac{1}{n} \sum_{\substack{X \subseteq E(G) \\ \forall v, deg_X(v)=2}} \sum_{\substack{S \subseteq X \\ |S|=|X|-1}} (det(F_S))^2$$

$$= \frac{1}{n} \sum_{\substack{X \subseteq E(G) \\ \forall v, deg_X(v)=2}} \sum_{\substack{S \subseteq X \\ |S|=|X|-1}} \Big(\sum_{f: S \xrightarrow{1-1} V(G)} sgn(f) \prod_{e \in S} a_{f(e),e} \Big)^2$$

$$= \frac{1}{n} \sum_{\substack{X \subseteq E(G) \\ \forall v, deg_X(v)=2}} \sum_{\substack{S \subseteq X \\ |S|=|X|-1}} \sum_{f_1, f_2: S \xrightarrow{1-1} V(G)} sgn(f_1) sgn(f_2) \prod_{e \in S} a_{f_1(e),e} a_{f_2(e),e}$$

Now, unlike the treewidth DP that we have seen so far, there is no obvious partial solution that can be built up. So, the trick here, lies in constructing one that can be efficiently computed over a tree decomposition.

For each bag i in the tree decomposition, $s_1, s_2 \in \{0,1\}^{B_i}$, and $s_{deg} \in \{0,1,2\}^{B_i}$ we store the partial sum S_i as follows:

$$S_i[s_{deg}, s_1, s_2] = \frac{1}{n} \sum_{\substack{X \subseteq E_i \\ \forall v \in V_i \setminus B_i, deg_X(v)=2 \\ \forall v \in B_i, deg_X(v)=s_{deg}(v)}} \sum_{S \subseteq X} \sum_{\substack{f_1: S \xrightarrow{1-1} V_x \setminus \{v_1\} \setminus s^{-1}(0) \\ f_1: S \xrightarrow{1-1} V_x \setminus \{v_1\} \setminus s^{-1}(0)}} sgn(f_1) sgn(f_2) \prod_{e \in S} a_{f_1(e),e} a_{f_2(e),e}$$

$s_{deg}(v)$ gives the degree of the vertex v from our chosen edges so far, $s_1(v)$ says if some edge is mapped to v by f_1, and $s_2(v)$ says if some edge is mapped to v by f_2. At this point, it is intutively easier to think of our partial solution as a sum of terms where each term is the product of matrix entries corresponding to two partial permutations.

3.2 Dynamic Programming

Leaf Node:

For any leaf node i,

$$S_i[\emptyset, \emptyset, \emptyset] = 1$$

i.e. the empty tree

Introduce Vertex Node:

For the introduction of a vertex v from node j to node i:

$$S_i[s_{deg}, s_1, s_2] = \begin{cases} S_j[s_{deg}|_{B_j}, s_1|_{B_j}, s_2|_{B_j}] & \text{if } s_{deg}[v] = s_1[v] = s_2[v] = 0 \\ 0 & \text{otherwise} \end{cases}$$

Since we are not introducing any edge incident on v yet, any mapping that maps any edge e' to v makes it zero (Since $F_{e',v} = 0$ if e' is not incident on v)

Forget Node:

For forgetting vertex v from node j to node i:

$$S_i[s_{deg}, s_1, s_2] = \begin{cases} S_j[s_{deg}[v \leftarrow 2], s_1[v \leftarrow 1], s_2[v \leftarrow 1]] & \text{if } v \neq v_1 \\ S_j[s_{deg}[v \leftarrow 2], s_1[v \leftarrow 0], s_2[v \leftarrow 0]] & \text{if } v = v_1 \end{cases}$$

If we are forgetting a vertex $v \neq v_1$, then it must have degree 2 and must be mapped to by both functions f_1 and f_2. If $v = v_1$, then it's degree must still be 2 though it must not be mapped to by f_1 or f_2.

Introduce Edge Node:

For introduction of edge $e = uv$ from bag j to bag i:

$$S_i[s_{deg}, s_1, s_2] = S_j[s_{deg}, s_1, s_2] + S_j[s'_{deg}, s_1, s_2] + \sum_{\substack{u' \in s_1^{-1}(1) \setminus \{v_1\} \setminus \mathrm{e} \\ v' \in s_2^{-1}(1) \setminus \{v_1\} \setminus \mathrm{e}}} S_j[s'_{deg}, s_1[u' \leftarrow 0], s_2[v' \leftarrow 0]] a_{u',e} a_{v',e} .(-1)^{inv(s_1^{-1}(1),u') + inv(s_2^{-1}(1),v')}$$

Where s'_{deg} is s_{deg} with the values for u and v reduced by one. The first term in the sum is the caseses where e is not in X. The second term sorrespond to the cases where e is in X but not in S. The third term corresponds to all the cases where e is part of S. Hence e is being mapped in all places where it is possible. The updation of the signature of the mapping is calculated using *Lemma 6*, and the fact that we have a post order numbering on edges and vertices, vertices from $V_i \setminus B_i$ cannot participate in new inversions with e.

Join Node:

For the case of joining of two nodes j and k to give i:

$$S_i[s_{deg}, s_1, s_2] = \sum_{\substack{s_{deg,j}+s_{deg,k}=s_{deg} \\ s_{1,j}+s_{1,k}=s_1 \\ s_{2,j}+s_{2,k}=s_2}} S_j[s_{deg,j}, s_{1,j}, s_{2,j}].S_j[s_{deg,k}, s_{1,k}, s_{2,k}]$$

$$.(-1)^{inv(s_{1,j}^{-1}(1),s_{1,k}^{-1}(1))+inv(s_{2,j}^{-1}(1),s_{2,k}^{-1}(1))}$$

Since the edges of j are disjoint from the edges of k, we only need to make sure no vertex is mapped in both nodes, and that their degree adds up in i. This also gives us an opportunity to optimize the algorithm, since this is the bottle neck, we need not consider all $12 \times 12 = 144$ possible states for every vertex (i.e. we don't need to consider cases when a vertex is mapped in both children, degree is 2 is both children, etc..). The following shows the table for what pairs are to be considered. This also gives us another chance for optimization. For every vertex not equal to v_1, we need not even consider all 12 states. Just 6 will do.

Table 1: States required to be considerd for Hamiltonian Cycles

	(0,0,0)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)	(2,1,1)
(0,0,0)	(0,0,0)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)	(2,1,1)
(1,0,0)	(1,0,0)	X	X	X	(2,1,1)	X
(1,0,1)	(1,0,1)	X	X	(2,1,1)	X	X
(1,1,0)	(1,1,0)	X	(2,1,1)	X	X	X
(1,1,1)	(1,1,1)	(2,1,1)	X	X	X	X
(2,1,1)	(2,1,1)	X	X	X	X	X

Thus only 15 such pairs are to be considered, taking at most 15^t steps at each join node (t is the width). This gives us the running time for out algorithm.

Theorem 7. *There is an algorithm that takes a tree decomposition of a Graph G of width t, and solves* Counting Hamiltonian Cycles *problem in time $15^t t^{\mathcal{O}(1)} n^2$ and in time $6^p p^{\mathcal{O}(1)} n^2$ time for a path decomposition of width p.*

For the root node r, $S_r[\emptyset, \emptyset, \emptyset]$ gices the number of Hamiltonian Cycles in the graph.

The path decomposition algorithm takes faster because of another observation: For every vertex not equal to v_1, we need not even consider all 12 states. Just 6 will do (the ones corresponding the the rows/columns of the table).

4 Counting Steiner Trees

4.1 Constructing Partial Solutions

We shall approach this in the exact same way that we did, the *Counting Hamiltonian Cycles* problem. The set of terminals is given by T. Let B be the number of Steiner Trees of a graph G.

$$B = \sum_{\substack{X \subseteq E \\ |X|=k}} [\text{X is a Steiner tree}]$$

$$= \sum_{\substack{Y \subseteq V \\ |Y|=k+1 \\ T \subseteq Y}} \sum_{\substack{X \subseteq E(Y,Y) \\ |X|=|Y|-1}} [(X,Y) \text{ is a tree}]$$

$$= \sum_{\substack{Y \subseteq V \\ |Y|=k+1 \\ T \subseteq Y}} \sum_{\substack{X \subseteq E(Y,Y) \\ |X|=|Y|-1}} (det(F_{Y,X}))^2$$

$$= \sum_{\substack{Y \subseteq V \\ |Y|=k+1 \\ T \subseteq Y}} \sum_{\substack{X \subseteq E(Y,Y) \\ |X|=|Y|-1}} \sum_{f_1, f_2 : X \xrightarrow{1-1} Y \setminus \{v_1\}} sgn(f_1) sgn(f_2) \prod_{e \in X} a_{f_1(e),e} a_{f_2 e, e}$$

And similar to the previous problem, we write our partial soltion solution as follows:

$$S_i[q, s_Y, s_1, s_2] = \sum_{\substack{Y \subseteq V_i \\ |Y|=q \\ T \cup V_i \subseteq Y \\ Y \cap B_i = s_Y^{-1}(1)}} \sum_{X \subseteq E(Y,Y) \cap E_i} \sum_{\substack{f_1 : X \xrightarrow{1-1} Y \setminus \{v_1\} \setminus s_1^{-1}(0) \\ f_2 : X \xrightarrow{1-1} Y \setminus \{v_1\} \setminus s_2^{-1}(0)}} sgn(f_1) sgn(f_2) \prod_{e \in X} a_{f_1(e),e} a_{f_2 e, e}$$

Here, q is the number of vertices in the steiner tree so far, s_Y tells what vertices in the bag are part of the partial trees, s_1 and s_2 mean exactly the same thing as before.

4.2 Dynamic Programming

Leaf Node:

$$S_i[0, \emptyset, \emptyset, \emptyset] = 1$$

Introduce Vertex Node:

For a node i, with child j that introduces a vertex v:

$$S_i[q, s_Y, s_1, s_2] = \begin{cases} S_j[q, s_Y|_{B_i}, s_1|_{B_i}, s_2|_{B_i}] & \text{if } s_Y[v] - s_1[v] = s_2[v] = 0 \\ S_j[q-1, s_Y|_{B_i}, s_1|_{B_i}, s_2|_{B_i}] & \text{if } v = v_1 \\ 0 & \text{otherwise} \end{cases}$$

Basically, we are counting al the partial solutions where v is part of the Steiner Tree and all those where v isn't. At this point, there are no non-zero terms which contain v mapped in either f_1 or f_2, as no edges incident on it have been introduced yet. If $v \in T$, then we have to include it in Y.

Forget Node:

For a node i, with child j that forgets a vertex v:

$$S_i[q, s_Y, s_1, s_2] = \begin{cases} S_j[q, s_Y[v \leftarrow 1], s_1[v\ gets1], s_2[v \leftarrow 1]] & \text{if } v \in T \text{ and } v \neq v_1 \\ S_j[q, s_Y[v \leftarrow 1], s_1[v\ gets0], s_2[v \leftarrow 0]] & \text{if } v = v_1 \\ S_j[q, s_Y[v \leftarrow 1], s_1[v\ gets1], s_2[v \leftarrow 1]] & \\ +S_j[q, s_Y[v \leftarrow 0], s_1[v\ gets0], s_2[v \leftarrow 0]] & \text{otherwise} \end{cases}$$

We are counting the cases when v is part of our Steiner Trees, in which case it has to be mapped to by both functions unless it is v_1, and those cases when it is not part of the Steiner trees. If $v \in T$, the v has to belong to the Trees.

Introduce Edge Node:

For introduction of edge $e = uv$ from bag j to bag i:

$$S_i[q, s_Y, s_1, s_2] = S_j[q, s_Y, s_1, s_2] + \sum_{\substack{u' \in s_1^{-1}(1) \setminus \{v_1\} \setminus \mathrm{e} \\ v' \in s_2^{-1}(1) \setminus \{v_1\} \setminus \mathrm{e}}} S_j[q, s_Y, s_1[u' \leftarrow 0], s_2[v' \leftarrow 0]] a_{u',e} a_{v',e}$$

$$.(-1)^{inv(s_1^{-1}(1), u') + inv(s_2^{-1}(1), v')}$$

Here also, we count the cases where e is not an edge in the steiner tree, and those cases where e is part of the steiner tree, and to do that, we need to map e and update the number of inversions.

Join Node:

For the case of joining of two nodes j and k to give i:

$$S_i[q, s_Y, s_1, s_2] = \sum_{\substack{q_j+q_k-|s_Y^{-1}(1)|=q \\ s_{1,j}+s_{1,k}=s_1 \\ s_{2,j}+s_{2,k}=s_2}} S_j[q_j, s_Y, s_{1,j}, s_{2,j}].S_j[q_k, s_Y, s_{1,k}, s_{2,k}]$$

$$.(-1)^{inv(s_{1,j}^{-1}(1),s_{1,k}^{-1}(1))+inv(s_{2,j}^{-1}(1),s_{2,k}^{-1}(1))}$$

Similar to the previous case, we only need to maintain that a vertex is not mapped to by both functions and that the numbwe vertices counted in our steiner trees on both sides so far add up.

Here also, we notice that we don't need to maintain all $2 \times 2 \times 2 = 8$ states foe every vertex because some of them are trivially zero. The following table shows the states we need to maintain for each node and the number of pairs we need to consider for the join node. Note that we need to maintain one such pair for each q

Table 2: States required to be considerd for Steiner Trees

	(0,0,0)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
(0,0,0)	(0,0,0)	X	X	X	X
(1,0,0)	X	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
(1,0,1)	X	(1,0,1)	X	(1,1,1)	X
(1,1,0)	X	(1,1,0)	(1,1,1)	X	X
(1,1,1)	X	(1,1,1)	X	X	X

for the root node r, $S_r[k+1, \emptyset, \emptyset, \emptyset]$ gives the number of Steiner Trees of size k.

Theorem 8. *There is an algorithm that takes a tree decomposition of a Graph G of width t, and solves* Counting Steiner Trees *of size k problem in time $10^t t^{\mathcal{O}(1)} n^3$ and in time $5^p p^{\mathcal{O}(1)} n^3$ time for a path decomposition of width p.*

For the root node r, $S_r[k+1, \emptyset, \emptyset, \emptyset]$ gives the number of Steiner Trees of size k.

References

[1] Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, Jesper Nederlof: Solving weighted and counting variants of connectivity problems parameterized by treewidth deterministically in single exponential time. CoRR abs/1211.1505 (2012)