Advanced Graph Algorithms Jan-Apr 2014

Lecture 31 — May 02, 2014

Lecturer: Saket Saurabh *Scribe: Roohani Sharma*

1 Overview

The aim is to solve the **Minimum Steiner Tree** problem parameterized by treewidth of the graph. In the first section an algorithm using dynamic programming over a nice tree decomposition running in time $O^*(t^{O(t)})$ is presented. In the next section a way to reduce the number of states stored for each bag using Gaussian elimination is presented yielding an algorithm running in time $O^*(2^{O(t)})$.

2 Minimum Steiner Tree problem using dynamic programming

The **Minimum Steiner Tree** problem is defined as follows.

Given an undirected graph $G = (V, E)$, a nice tree decomposition (T_r, f) of G and ($K \subseteq V$ called the terminals), the algorithm should find the minimum sized (in terms of number of edges) tree $T = (V', E')$ such that $K \subseteq V'$.

We will use dynamic programming over nice tree decomposition. Let us represent $f(i)$ as X_i. The idea is that at any bag store possible forests that could be extended to get the optimal Steiner tree. Thus, for any bag X_i guess the intersection of the optimal Steiner tree with X_i. For each such guessed subset, guess a partition of this subset such that each block in the partition is a part of a unique component of the forest in $G[V_i]$ that spans all terminals from V_i. Thus, more formally, for every bag X_i let us store for all $X \subseteq X_i$ and for all parititions $\mathcal{P} = \{P_1, \ldots, P_l\}$ of X, let us store $c[X, \mathcal{P}, i]$ = minimum number of edges in a forest F satisfying the following conditions.

(For any partition $\mathcal{P} = \{P_1, \ldots, P_l\}$ we call $P_1, P_2, \ldots, P_l$ the blocks of $\mathcal{P}$.)

1. F has exactly l components $C_1, \ldots, C_l$ and for all $s \in \{1, \ldots, l\}$, $P_s \subseteq C_s$.
2. $X_i \cap V(F) = X$ (i.e. vertices of $X_i \setminus X$ are untouched by the Steiner tree extending F).
3. $(K \cap V_i) \in V(F)$ (i.e. all terminals seen so far are in F).

Note that having computed these states for all bags of the nice tree decomposition the size of the minimum Steiner tree is given by the following expression.

$$min\{c[\emptyset, \emptyset, r], \quad min_{X \subseteq X_r \quad X \neq \emptyset} \quad c[X, \{X\}, r]\}$$

Note that the partition corresponding to the solution must be a single block partition because we are looking for a tree.

We can see how to compute $c[X, \mathcal{P}, i]$ for each bag X_i starting from the leaf node and given rules to compute it at each kind of node.

Leaf Node Since the leaf nodes are empty, the trivial forest below it has length 0.

$$c[\emptyset, \emptyset, l] = 0$$

Introduce Vertex Node Let j be the introduce vertex bag and let i be its child. Then $X_j = X_i \cup \{v\}$. If $v \in K$ then for all $X \subseteq X_i$ not containing v, $c[X, \mathcal{P}, i] = \perp$ (where $\perp$ represents an invalid entry). Also, if $v \in X$ then the valid partitions of X are the ones where v is in a separate block because when v is introduced for the first time no edges incident on it are introduced. Hence all $c[X, \mathcal{P}, i] = \perp$, where $\mathcal{P}$ is not a valid partition of X. Otherwise we have the following.

$$c[X, \mathcal{P}, i] = \begin{cases} c[X, \mathcal{P}, j] & \text{if } v \notin X \\ c[X \setminus \{v\}, \mathcal{P} \setminus \{\{v\}\}, j] & \text{otherwise} \end{cases}$$

Introduce Edge Node Let i be the node to which edge $\{u, v\}$ is introduced and let j be the child of i.If either u or v is not in X then that means we have for that guessed subset X, we have decided not to take it in the final Steiner tree. Hence if either u or v is not in X then the edge $\{u, v\}$ cannot be taken. Otherwise, if u and v both are in X then we can either pick the edge $\{u, v\}$ or not. Hence we have the following cases.

$$c[X, \mathcal{P}, i] = \begin{cases} c[X, \mathcal{P}, j] & \text{if } u \notin X \text{ or } v \notin X \\ min\{min_{\mathcal{P}'}\{c[X, \mathcal{P}', j] + 1\}, c[X, \mathcal{P}, j]\} & \text{otherwise} \end{cases}$$

Here $\mathcal{P}'$ are all those partitions where u and v are in separate blocks (so that adding the $\{u, v\}$ does not create a cycle) and after merging the blocks containing u and v we obtain the partition $\mathcal{P}$.

Forget Node

Let i be the forget node that forgets v and let j be its child. Then $X_i = X_j \setminus \{v\}$. If $v \in K$, then v must belong the forest that X sees in V_i.Otherwise, v might belong the forest that X sees in V_i or not. Depending upon these cases we have the following expression.

$$c[X, \mathcal{P}, i] = \begin{cases} min_{\mathcal{P}'}\{c[X \cup \{v\}, \mathcal{P}', j]\} & \text{if } v \in K \\ min\{min_{\mathcal{P}'}\{c[X \cup \{v\}, \mathcal{P}', j]\}, c[X, \mathcal{P}, j]\} & \text{otherwise} \end{cases}$$

Join Node Let i be the join node such that j and k are its children. Then $X_i = X_j = X_k$. Let us define a binary operation $\sqcup$ on partitions. Let $\mathcal{P}'$ and $\mathcal{P}''$ be two partitions of X. Then $\mathcal{P} = \mathcal{P}' \sqcup \mathcal{P}''$ is the join of $\mathcal{P}'$ and $\mathcal{P}''$ in the partition lattice language. Less formally, if we imagine two graphs $G_{\mathcal{P}'}$ and $G_{\mathcal{P}''}$ on the same vertex set X, where the family of connected components of $G_{\mathcal{P}'}$ is $\mathcal{P}'$ and the family of connected components of $G_{\mathcal{P}''}$ is $\mathcal{P}''$ then, $\mathcal{P}' \sqcup \mathcal{P}''$ corresponds to the set of connected components in the union of $G_{\mathcal{P}'}$ and $G_{\mathcal{P}''}$.

We call $\mathcal{P}$ to be the acyclic merge of two partitions $\mathcal{P}'$ and $\mathcal{P}'$ if $\mathcal{P} = \mathcal{P}' \sqcup \mathcal{P}''$ and $G_{\mathcal{P}'} \cup G_{\mathcal{P}''}$ is a forest.

$$c[X, \mathcal{P}, i] = min_{\mathcal{P}', \mathcal{P}'}\{c[X, \mathcal{P}', j], c[X, \mathcal{P}'', k]\}$$

Here $\mathcal{P}^{'}$ and $\mathcal{P}^{''}$ are such that $\mathcal{P}$ is the acyclic merge of $\mathcal{P}^{'}$ and $\mathcal{P}^{''}$.

Running Time Analysis Since for each bag of the nice tree decomposition, we store $2^{t+1}t^t$ number of states and each state can be computed in atmost $\binom{t}{\frac{t}{2}}$ time, the total running time is $O(t^O(t)n)$

3 Minimum Steiner Tree using Gaussian Elimination in $2^{O(t)}$ time

3.1 The idea behind it

In the previous section we saw an algorithm which used dynamic programming and computed the size of the minimum Steiner tree in time $O^*(t^{O(t)})$. If we look at the running time analysis of the algorithm we observe that it is the number of partitions that we are storing is causing the runnning to be doubly exponential. If we could store,say, $2^{O(t)}$ number of partitions we could get an algorithm that runs in time $O^*2^{O(t)}$. Thus the question now is, is it possible to discard some partitions and do all the necessary work with the remaining. It turns out yes!

Before we dwell further into it, let us tweak the algorithm of previous section to store at each state the value $c[X, \mathcal{P}, z, i]$ instead of $c[X, \mathcal{P}, i]$, where $c[X, \mathcal{P}, z, i] = 1$ if there exists a forest of size z with properties mentioned in the previous section and $c[X, \mathcal{P}, z, i] = 0$ if no such forest exists. The previous algorithm can be easily modified to store such states.

Now let us come back to the question of storing fewer partitions. Let us analyse that is the kind of property that we want our smaller subfamily (which is the subfamily of partitions that we want to store) to have such that we do not loose any useful information while discarding the other partitions.

Let $\mathcal{A}$ be the family of partitions which was being kept by the previous algorithm (or the family of partitions that are required to be kept) and let $\mathcal{A}^{'} \subseteq \mathcal{A}$ be the subfamily that we actually keep.

Consider any partition $p \in \mathcal{A}$ that could be extended to an optimum solution, say OPT. Let F be the forest obtained by intersecting OPT with the graph (V_i, E_i) Let r be the partition of X corresponding to the connected components of $OPT \setminus F$. Clearly $p \sqcup r = \{\{X\}\}$ as OPT is connected. If $p \in \mathcal{A}^{'}$ then we are done, otherwise there should exist a $q \in \mathcal{A}^{'}$ such that $q \sqcup r = \{\{X\}\}$ because if such a q exists then there would exist another optimum solution $OPT^{'} = (OPT \setminus F_p) \cup F_q$, where F_q is the forest corresponding to the reachable state (X, q, z, i) if F_p is the forest corresponding to the reachable state (X, p, z, i). (We call a state (X, p, z, i) reachable if $c[X, p, z, i] = 1$).

More formally what we aim for is an algorithm that behaves as said in the following theorem.

(We use $\pi(X)$ to denote the collection of all partitions of X.)

Theorem 1. *There is an algorithm that given a set of partitions $\mathcal{A} \subseteq \pi(X)$ in time $|\mathcal{A}|2^{O(|U|)}$ finds a set $\mathcal{A}^{'} \subseteq \mathcal{A}$ such that for any $p \in \mathcal{A}$ and $r \in \pi(X)$ with $p \sqcup r = \{\{X\}\}$ there exists a partition $q \in \mathcal{A}^{'}$ such that $q \sqcup r = \{\{X\}\}$.*

If we prove thie theorem and show an upper bound of $2^{O(t)}$ on the size of $\mathcal{A}^{'}$, then we are done. So lets get started towards it!

3.2 The trick

Let us construct a matrix $M \in \{0,1\}^{\pi(X)\times\pi(X)}$ as follows:

$$M[p,q] = \begin{cases} 1 & \text{if } p \sqcup q = \{\{X\}\} \\ 0 & \text{otherwise} \end{cases}$$

The trick is as follows. Choose $\mathcal{A}^{'}$, the smaller (or representative) family to be the basis of the row space of this matrix. Now let us see why $\mathcal{A}^{'}$ has all the desired properties. (Note that we will work in $\mathbb{Z}_2$.)

For any $p \notin \mathcal{A}^{'}$, since $\mathcal{A}^{'}$ is the basis of the row space of M, hence

$$M[p,.] = \sum_{q\in X} M[q,.]$$

where $Y \subseteq \mathcal{A}^{'}$.

Note that $M[p,r] = 1$ if and only if $M[s,r] = 1$ for some $s \in Y$ because we are working in $\mathbb{Z}_2$. Since $s \in Y \in \mathcal{A}^{'}$. Hence, $\mathcal{A}^{'}$ is indeed a representative family for any given $\mathcal{A} \subset \pi(X)$.

The algorithm which Theorem 1 talks about is nothing but Gaussian Elimination on the matrix M.It if we would notice, there is a trap. The matrix M is huge. The rank of M and hence the size of $\mathcal{A}^{'}$ is at most $min\{$number of rows of M, number of columns of $M\} \leq |\pi(X)| = (t+1)^{t=1}$ which is not what we were aimimg for.

So the next step is to somehow encode the information contained in M in a matrix that has fewer number of columns. For this let us define cuts and subsequently a new matrix $\mathcal{C}$.

Definition 2. *$cuts(X) = \{(V_1,V_2)|V_1 \cup V_2 = X, V_1 \cap V_2 = \emptyset, 1 \in V_1\}$ where 1 is an arbitrary but fixed element of X.*

We will realize the importance of taking $1 \in V_1$ when we do some counting arguments modulo 2.

Definition 3. *Let us define a matrix $\mathcal{C} \in \{0,1\}^{\pi(X)\times cuts(X)}$ such that*

$$\mathcal{C}[p,(V_1,v_2)] = \begin{cases} 1 & (V_1,V_2) \sqsubseteq p \\ 0 & otherwise \end{cases}$$

Here $p \sqsubseteq q$ represents that q is a refinement of p i.e. every block of p is contained entirely in some block of p.

Having defines these we proved the following lemma.

Lemma 4. $M \equiv_2 \mathcal{C}\mathcal{C}^T$.

Proof. For every $p,q \in \pi(X)$,

$$\mathcal{C}\mathcal{C}^T[p,q] = \sum_{(V_1,V_2)\in cuts(X)} [(V_1,V_2) \sqsubseteq p][(V_1,V_2) \sqsubseteq q]$$

$$= \sum_{(V_1,V_2)\in cuts(X)} [(V_1, V_2) \sqsubseteq p \sqcup q]$$

$$= 2^{\#blocks(p\sqcup q)-1}$$

$$\equiv_2 1 \quad if \quad and \quad only \quad if \quad [p \sqcup q = \{\{X\}\}]$$

□

Thus we have been able to write the big matrix M in terms of relatively small matrix $\mathcal{C}$. Let us now formally see how exactly does the information carried by M gets encoded by $\mathcal{C}$.

Lemma 5. *Let $Y \subseteq \pi(X)$ and $p, r \in \pi(X)$ such that $\mathcal{C}[p, .] \equiv_2 \sum_{q\in Y} C[q, .]$ and $r \sqcup p = \{X\}$. then there exists $q \in Y$ such that $r \sqcup q = \{\{X\}\}$.*

Proof. From Lemma 4 we have,

$$M[p, .] \equiv_2 \mathcal{C}[p, .]\mathcal{C}^T$$

By assumption,

$$\equiv_2 (\sum_{q\in Y} \mathcal{C}[q, .])\mathcal{C}^T$$

By linearity of matrix multipliction,

$$\equiv_2 \sum_{q\in Y} M[q, .]$$

In particular $M[p, r] \equiv_2 \sum_{q\in Y} M[q, r]$. Therefore, if $M[p, r] = 1$ then $M[q, r] = 1$ for some $q \in Y$. □

Having proved these lemmas the proof of Theorem 1 follows from the following argument. For any given family $\mathcal{A} \subseteq \pi(X)$, let $\mathcal{C}' = \mathcal{C}[\mathcal{A}, .]$. Let $\mathcal{A}'$ be the basis of the row space of $\mathcal{C}'$. Then for any $p \in \mathcal{A}$ if there exists an r such that $p \sqcup r = \{\{X\}\}$ and $p \notin \mathcal{A}'$ then $\mathcal{C}[p, .] \equiv_2 \sum_{q\in Y} C[q, .]$ where $Y \subseteq \mathcal{A}'$. Hence by Lemma there exist a $q \in \mathcal{A}'$ such that $q \sqcup r = \{\{X\}\}$. Hence we are done!

3.3 Running time analysis

Since we perform Gaussian elimination on $\mathcal{C}'$, the algorithm runs in time $|\mathcal{A}^{O(1)}|2^{O(|X|)}$ (which is polynomial in the number of rows of $\mathcal{C}'$ and polynomial in its number of columns). Also since the number of columns in $\mathcal{C}$ is 2^{t+1}, the size of $\mathcal{A}'$ is bounded by 2^{t+1}.

Therefore, now at each node of the nice tree decomposition we store $2^{t+1}2^{t+1}n$ states (the first 2^{t+1} for all subsets of the vertices in the node, the second 2^{t+1} for all partitions required to be stored and n for all z, $1 \leq z \leq n$). To calculate the state of each node we require $\binom{\frac{2}{t}}{2} \leq 2^{2t}$. Then for each node we run Gaussian elimination too. Hence the overall running time of this algorithm is $O(2^{O(t)}n^2)$.