Lecture notes for Apr 19, 2023
Tree-width and dynamic programming

Chun-Hung Liu

April 12, 2023

Let G be a graph. A *tree-decomposition* of G is a pair $(T, \mathcal{X})$, where T is a tree and $\mathcal{X} = \{X_t : t \in V(T)\}$ is a collection of subsets of $V(G)$ indexed by the vertices of T such that

(TD1) $\bigcup_{t \in V(T)} X_t = V(G)$,

(TD2) for every edge $uv \in E(G)$, there exists $t \in V(T)$ such that $\{u, v\} \subseteq X_t$, and

(TD3) for every vertex $v \in V(G)$, the set $\{t \in V(T) : v \in X_t\}$ induces a connected subtree of T

(that is, if $t_1, t_2 \in V(T)$ with $v \in X_{t_1} \cap X_{t_2}$, then $v \in X_t$ for every vertex t in the path in T between t_1 and t_2).

Each set X_t is called the *bag* at t. The *width* of the tree-decomposition $(T, \mathcal{X})$ is defined to be $\max_{t \in V(T)} |X_t| - 1$. The *tree-width* of G is the minimum width of a tree-decomposition of G.

Recall that every edge of the tree in a tree-cut decomposition of a graph G gives an edge-cut of G. Tree-decomposition has a similar property, by replacing "edge-cut" by "separation", as shown in the following.

Proposition 1 *Let G be a graph. Let $(T, \mathcal{X})$ be a tree-decomposition of G. Let $t_1t_2 \in E(T)$. For $i \in [2]$, let T_i be the component of $T - t_1t_2$ containing t_i. Then $(\bigcup_{t \in V(T_1)} X_t, \bigcup_{t \in V(T_2)} X_t)$ is a separation of G with order $|X_{t_1} \cap X_{t_2}|$.*

Proof. We first show that $(\bigcup_{t \in V(T_1)} X_t, \bigcup_{t \in V(T_2)} X_t)$ is a separation of G. For simplicity, for each $i \in [2]$, let $A_i = \bigcup_{t \in V(T_i)} X_t$. By (TD1), $A_1 \cup A_2 = V(G)$.

So it suffices to show that there exists no edge of G between $A_1 - A_2$ and $A_2 - A_1$ and show that $A_1 \cap A_2 = X_{t_1} \cap X_{t_2}$.

Suppose to the contrary that there exists an edge of G between $v_1 \in A_1 - A_2$ and $v_2 \in A_2 - A_1$. By (TD2), there exists $t \in V(T)$ such that $\{v_1, v_2\} \subseteq X_t$. Note that t is in T_1 or T_2. That is, $\{v_1, v_2\} \subseteq X_t$ is either contained in A_1 or contained in A_2. If $X_t \subseteq A_1$, then $v_2 \in X_t \subseteq A_1$, contradicting $v_2 \in A_2 - A_1$. If $X_t \subseteq A_2$, then $v_1 \in X_t \subseteq A_2$, contradicting $v_1 \in A_1 - A_2$.

So (A_1, A_2) is a separation. Now we study the order of (A_1, A_2). Clearly, $X_{t_1} \cap X_{t_2} \subseteq A_1 \cap A_2$. If $v \in A_1 \cap A_2$, then there exist $s_1 \in V(T_1)$ and $s_2 \in V(T_2)$ such that $v \in X_{s_1} \cap X_{s_2}$, so (TD3) implies that $v \in X_s$ for every vertex s of T contained in the path in T between s_1 and s_2, and hence $v \in X_{t_1} \cap X_{t_2}$. So $A_1 \cap A_2 \subseteq X_{t_1} \cap X_{t_2}$. Hence $A_1 \cap A_2 = X_{t_1} \cap X_{t_2}$. That is, the order of (A_1, A_2) is $|A_1 \cap A_2| = |X_{t_1} \cap X_{t_2}|$. ■

Proposition 1 serves a main motivation for tree-decomposition and explains why we want the conditions (TD2) and (TD3) when we define a tree-decomposition. Proposition 1 also implies that every bag gives a vertex-cut, as shown in the following.

Proposition 2 *Let G be a graph. Let $(T, \mathcal{X})$ be a tree-decomposition of G. Let $t_0 \in V(T)$. Let C_1 and C_2 be distinct components of $T - t_0$. Then $\bigcup_{t \in V(C_1)} X_t - X_{t_0}$ and $\bigcup_{t \in V(C_2)} X_t - X_{t_0}$ are disjoint, and there exists no edge of G between $\bigcup_{t \in V(C_1)} X_t - X_{t_0}$ and $\bigcup_{t \in V(C_2)} X_t - X_{t_0}$.*

Proof. Let t_1 be the neighbor of t_0 contained in C_1. Let (A, B) be the separation of G given by the edge $t_0 t_1$ as shown in Proposition 1. Since $A - B$ and $B - A$ are disjoint, $\bigcup_{t \in V(C_1)} X_t - X_{t_0}$ and $\bigcup_{t \in V(C_2)} X_t - X_{t_0}$ are disjoint. And every edge of G between $\bigcup_{t \in V(C_1)} X_t - X_{t_0}$ and $\bigcup_{t \in V(C_2)} X_t - X_{t_0}$ is an edge of G between $A - B$ and $B - A$ by (TD3). So there exists no edge of G between $\bigcup_{t \in V(C_1)} X_t - X_{t_0}$ and $\bigcup_{t \in V(C_2)} X_t - X_{t_0}$. ■

1 Dynamic programming

Now we show how to use a tree-decomposition to obtain FPT algorithms. We first show how to find a maximum stable set.

================================

A dynamic programming for finding a maximum stable set with given a tree-decomposition
Input: A graph G, a tree-decomposition $(T, \mathcal{X})$, a node r of T, and a stable set S of $G[X_r]$.
Output: A stable set I of G with $I \cap X_r = S$ such that $|I|$ is maximum among all stable sets I' of G with $I' \cap X_r = S$.
Procedure:

Step 1: If $|V(T)| = 1$, then output $I = S$ and stop.

Step 2: For every neighbor c of r in T,

- let T_c be the component of $T - r$ containing c,
- let $G_c = G[\bigcup_{t \in V(T_c)} X_t]$,
- let $\mathcal{X}_c = \{X_t : t \in V(T_c)\}$,
  (so $(T_c, \mathcal{X}_c)$ is a tree-decomposition of G_c)
- let $S_c = S \cap X_c$,
- for every stable set D of $G[X_c]$ with $D \cap X_c \cap X_r = S_c$, run this algorithm with input $(G, (T, \mathcal{X}), r, S) = (G_c, (T_c, \mathcal{X}_c), c, D)$ to obtain a stable set $I_{c,S,D}$,
- let $I_{c,S}$ be the set $I_{c,S,D}$ that gives maximum $|I_{c,S,D}|$ among all stable sets D of $G[X_c]$ with $D \cap X_c \cap X_r = S_c$.

Step 3: Let $I = S \cup \bigcup_{c \in N_T(r)} I_{c,S}$. Output I.

==============================

Theorem 3 *The above algorithm runs correctly.*

Proof. We show the correctness by induction on $|V(T)|$. When $|V(T)| = 1$, $G = G[X_r]$, so it is obviously true. So we may assume $|V(T)| \geq 2$ and the algorithms outputs the correct answer when $|V(T)|$ is smaller.

Hence for every neighbor c of T and stable set D of $G[X_c]$ with $D \cap X_c \cap X_r = S_c$, $I_{c,S,D}$ is a stable set of G_c with $I_{c,S,D} \cap X_c = D$ such that $|I_{c,S,D}|$ is as large as possible. So for every c of T, $I_{c,S}$ is a stable set of G_c with $I_{c,S} \cap X_c \cap X_r = S_c = S \cap X_c$ such that $|I_{c,S}|$ is as large as possible.

Let I^* be a stable set of G with $I^* \cap X_r = S$ such that $|I^*|$ is as large as possible. It suffices to show $|I^*| \leq |I|$.

Let d be an arbitrary neighbor of r. Then $I^* \cap V(G_d)$ is a stable set of G_d with $I^* \cap X_r \cap X_d = S \cap X_d$. Hence $|I^* \cap V(G_d)| \leq |I_{d,S}|$. Moreover, $V(G_d) \cap X_r = X_d \cap X_r$ by (TD3), so $I^* \cap V(G_d) \cap X_r = I^* \cap X_d \cap X_r = S \cap X_d$. Similarly, $I_{d,S} \cap X_r = I_{d,S} \cap V(G_d) \cap X_r = I_{d,S} \cap X_d \cap X_r = S \cap X_d$. Therefore, $|I^* \cap V(G_d) - X_r| = |I^* \cap V(G_d)| - |I^* \cap V(G_d) \cap X_r| = |I^* \cap V(G_d)| - |S \cap X_d| \leq |I_{d,S}| - |S \cap X_d| = |I_{d,S}| - |I_{d,S} \cap X_r| = |I_{d,S} - X_r|$.

By (TD3), if c_1 and c_2 are distinct neighbors of r in T, then $V(G_{c_1}) - X_r$ and $V(G_{c_2}) - X_r$ are disjoint. Hence, $|I^* - X_r| = \sum_{c \in N_T(r)} |I^* \cap V(G_c) - X_r| \leq \sum_{c \in N_T(r)} |I_{c,S} - X_r| = |I - X_r|$. So $|I^*| = |I^* - X_r| + |I^* \cap X_r| = |I^* - X_r| + |S| \leq |I - X_r| + |S| = |I - X_r| + |I \cap S| = |I|$. This shows that I is desired. ■

Theorem 4 *Let w be a positive integer. Given a graph G and a tree-decomposition $(T, \mathcal{X})$ of G with width at most w, we can find a maximum stable set of G in time $O(w4^w|V(T)|)$.*

Proof. Let r be a vertex of T. For each subset S of $G[X_r]$, we can test whether S is a stable set in $G[X_r]$ by brute force in time $2^{|X_r|}|X_r| = O(w2^w)$, and if so, we can run the previous algorithm to obtain a stable set I_S of G with $I_S \cap X_r = S$ such that $|I_S|$ is maximum. We can obtain I_S correctly by Theorem 3. Let I^* be the stable set I_S such that $|I_S|$ is maximum among all stable sets S of $G[X_r]$ with $I_S \cap X_r = S$. Then I^* is a maximum stable set of G.

So it suffices to show the time complexity. Note that the algorithm produce a search tree T', where the root instance is $(V(G), r, *)$, and its children are the instances $(V(G), r, S)$ among all stable sets S of $G[X_r]$, and for every such $(V(G), r, S)$, its children are $(V(G_c), c, D)$, where $c \in N_T(r)$ and D is a stable set of $G[X_c]$ with $D \cap X_r \cap X_c = S \cap X_c$, and so on. And to construct this search tree T', it takes 2^{w+1} time to construct the children of each node. So it takes $(w+1)2^{w+1}|V(T')|$ time to construct T'.

For each node of T' that has no child, it takes $O(1)$ time to obtain the answer. For each node v of T' that has a child, it takes $O(\deg_{T'}(v))$ time to obtain the answer for v from its children. Hence once T' is constructed, it takes $O(|V(T')| + |E(T')|) = O(|V(T')|)$ time to obtain the answer for the root. Therefore, the total running time is $O((w+1)2^{w+1}|V(T')|) = O(w2^w|V(T')|)$.

Note that each non-root node of T' is marked as (G_z, z, D) for some $z \in V(T)$ and $D \subseteq X_z$, and we know G_z is completely determined by z. So there are at most $|V(T)| \cdot 2^{w+1}$ non-root nodes. Hence $|V(T')| \leq 1 + 2^{w+1}|V(T)|$. Therefore, the total running time is $O(w2^w|V(T')|) = O(w4^w|V(T)|)$. ■

Note that the time complexity in Theorem 4 might look strange because it does not involve $|V(G)|$ at the first glance. But notice that since every bag has size at most $w + 1$, we have $(w + 1)|V(T)| \geq |V(G)|$, so $|V(T)| \geq |V(G)|/(w + 1)$.