KAIST, School of Computing, Spring 2024
Algorithms for NP-hard problems (CS492)
Lecture: Eunjung KIM

Scribed By: Eunjung KIM
Lecture #10-11
28 March, 2 April 2024

Contents

1 Tree decomposition: definition and basic properties

Definition 1. *Let G be a graph. A* tree decomposition *of G is a pair (T, χ), where T is a tree and $\chi : V(T) \to 2^{V(G)}$ is a mapping associating each tree node t to a vertex subset $\chi(t) \subseteq V(G)$ called a* bag, *which satisfies the following conditions.*

- **Vertex Coverage.** $\bigcup_{t \in V(T)} \chi(t) = V(G)$.
- **Edge Coverage.** *For every edge $e = uv$ of G, there is a tree node t of T such that $u, v \in \chi(t)$.*
- **Connectivity.** *For every vertex v of G, the nodes of T whose bags contain v is connected in T. In other words, the set $\{t \in V(T) : v \in \chi(t)\}$ is a subtree of T for each v of G.*

The width *of a tree decomposition (T, χ), written as* $\mathsf{width}(T, \chi)$, *equals* $\max_{t \in V(T)} |\chi(t)| - 1$. *The* treewidth *of G, denoted as* $\mathsf{tw}(G)$ *is*

$$\mathsf{tw}(G) := \min_{(T,\chi)} \mathsf{width}(T, \chi),$$

where (T, χ) is taken over all tree decompositions of G.

Some graph terminology. We set up some terminology to be used later. We follow the standard graph terminology, e.g. as in Diestel [1].

A tree decomposition (T, χ) of G is said to be *redundant* if there are two adjacent nodes t, t' of T such that $\chi(t) \subseteq \chi(t')$ or $\chi(t') \subseteq \chi(t)$ holds, non-redundant otherwise. A *rooted* tree decomposition is a tree decomposition (T, χ) where T is a rooted tree. When T is rooted, one can talk about a child/parent/ancestor/descendant of a tree node.

Let $A, B \subseteq V(G)$ be vertex subsets of G; note that they may intersect. An (A, B)*-path* in G is a path with one endpoint in A and another endpoint in B. If there is a vertex v in the intersection $A \cap B \neq \emptyset$, then v itself forms a trivial (A, B)-path. We say that a vertex set $X \subseteq V(G)$ *separates* A and B, or say that X *is an* (A, B)*-separator in* G if every (A, B)-path contains a vertex of X. Note that any (A, B)-separator must contain $A \cap B$ fully. In the special case when $A = \{a\}$ and $B = \{b\}$ for some $a, b \in V(G)$, we write (a, b)-separator instead of (rather cumbersome) $(\{a\}, \{b\})$-separator. A vertex set X is called a *separator* of G if there $a, b \in V(G) \setminus X$ such that X is an (a, b)-separator.

A *contraction* of a graph G by an edge $e = uv$ of G is an operation on G which (i) deletes u and v from G, (ii) adding a new vertex z_e, and (iii) connecting z_e to each vertex of $N_G(u) \cup N_G(v)$ (by an edge). The graph obtained from G by contracting an edge e (of G) is written as G/e.

Here are some basic properties of a tree decomposition and treewidth of a graph.

Lemma 1. *Let (T, χ) be a tree decomposition of G. The following holds.*

1. *One can turn (T, χ) into a non-redundant tree decomposition in linear time without increasing the width.*
2. *For any clique K of G, there is a tree node t of T such that $K \subseteq \chi(t)$.*
3. *Let xy be an edge of T and let T_x (respectively T_y) be the subtree of $T - xy$ containing x (respectively, with y). Then $\chi(x) \cap \chi(y)$ is a separator of $\bigcup_{t \in V(T_x)} \chi(t)$ and $\bigcup_{t \in V(T_y)} \chi(t)$. In particular, there is no edge between $\bigcup_{t \in V(T_x)} \chi(t) \setminus (\chi(x) \cap \chi(y))$ and $\bigcup_{t \in V(T_y)} \chi(t) \setminus (\chi(x) \cap \chi(y))$.*

4. *If (T,χ) is non-redundant, T has at most n nodes.*

Proof: We prove the last property[1]. Consider a mapping $\mathsf{top} : V(G) \to V(T)$ defined as

$$\mathsf{top}(v) := \text{the topmost node of } T \text{ containing } v.$$

In other words, top chooses the node of T whose distance to root is the minimum among all nodes whose bags contain v.

We first argue that top is well-defined, i.e. there is a unique such node achieving the minimum distance to root. Suppose the contrary, i.e. there are two distinct nodes x, y of T with $v \in \chi(x)$ and $v \in \chi(y)$ neither of which is an ancestor of the other. By the connectivity property of tree decomposition, all the nodes on the (x,y)-path P of T contains v in their bags. In particular, the least common ancestor of x and y, which must be distinct from both x and y, lies on P and contains v. This contradicts the choice of x and y.

Next, we claim that top is surjective, i.e. for every tree node t there is a vertex w of G such that $\mathsf{top}(w) = t$. Indeed, the non-redundancy of (T,χ) subsumes $\chi(\mathsf{root}) \neq \emptyset$ and for every vertex $w \in \chi(\mathsf{root})$, $\mathsf{top}(w) = \mathsf{root}$. If t is an internal node with a parent node $p(t)$, the set $\chi(t) \setminus \chi(p(t)) \neq \emptyset$ due to the non-redundancy of the given tree decomposition. For each $w \in \chi(t) \setminus \chi(p(t))$, the mapping top maps w to t. This proves that top is surjective, implying $|V(T)| \leq |V(G)|$.

$\square$

Lemma 2. *For any graph G, he following holds.*

1. $\mathsf{tw}(G) - 1 \leq \mathsf{tw}(G - v) \leq \mathsf{tw}(G)$ *for any vertex v of G.*
2. $\mathsf{tw}(G) - 1 \leq \mathsf{tw}(G - e) \leq \mathsf{tw}(G)$ *for any edge e of G.*
3. $\mathsf{tw}(G) - 1 \leq \mathsf{tw}(G/e) \leq \mathsf{tw}(G)$ *for any edge e of G.*
4. *The number of edges in G is at most* $\mathsf{tw}(G) \cdot n$*, where* $n := |V(G)|$.

Lemma 3. *A graph G is a forest if and only if* $\mathsf{tw}(G) \leq 1$.

Nice tree decomposition. A nice tree decomposition is a tree decomposition which is tailored to ease the design and description of a dynamic programming algorithm over a tree decomposition.

Definition 2. *Let G be a graph. A* nice tree decomposition (T,χ) *of G is a rooted tree decomposition such that each tree node t of T falls into one of the next four types and satisfies the corresponding property.*

- **Leaf node.** *t is a leaf of T.*
- **Introduce node.** *t has exactly one child, say t', in T and it holds that $\chi(t) = \chi(t') \cup \{v\}$ for some vertex v of G.*
- **Forget node.** *t has exactly one child, say t', in T and it holds that $\chi(t) = \chi(t') \setminus \{v\}$ for some vertex v of G.*
- **Join node.** *t has exactly two children, say t_1 and t_2, in T and it holds that $\chi(t) = \chi(t_1) = \chi(t_2)$.*

[1]Other properties were proved in the class.

Lemma 4. *Let (T,χ) be a (non-redundant) tree decomposition of G of width w. In linear time, one can obtain a nice tree decomposition (T',χ') of G of width w. Moreover,*

- *one can further impose the condition that all leaf nodes and the root of T' has empty bags, and*
- *the number of nodes in T' is at most $4wn$, where $n := |V(G)|$.*

Let us fix some notations for a rooted tree decomposition (T,χ) of G. The root of T is written as roote. For a tree node t of T, T_t denotes the subtree of T rooted at t; that is, the set of nodes in T_t is precisely the set of all tree nodes x such that the (unique) path between x and root in T intersects t. We define

$$V_t := \bigcup_{b \in V(T_t)} \chi(b), \qquad\qquad G_t := G[V_t].$$

A useful property of a nice tree decomposition for designing DP algorithm is that the vertex v introduced in an introduce node has all its neighbors in G_t in the bag containing v. This property is a simple corollary of (iii) in Lemma 1 applied for the specific setting of a nice tree decomposition.

Lemma 5. *Let (T,χ) be a nice tree decomposition of G. Let t be an introduce node of T with a child t' and v be the vertex of G such that $\chi(t)=\chi(t')\cup\{v\}$. Then $N_{G_t}(v)\subseteq\chi(t)$.*

A similar consequence of Lemma 1 to join node is stated in the next lemma.

Lemma 6. *Let (T,χ) be a nice tree decomposition of G. Let t be a join node of T with two children t_1 and t_2. Then $V_{t_1}\cap V_{t_2}=\chi(t)$.*

2 Dynamic programming over tree decomposition

Recall that $I \subseteq V(G)$ is an independent set of G if no two vertices of I are adjacent.

MAXIMUM INDEPENDENT SET
Instance: a graph $G=(V,E)$.
Goal: Find a maximum size independent set of G.

MAXIMUM INDEPENDENT SET is one of the exemplary NP-hard problems. On the other hand, it can be solved in polynomial time (in fact, $O(n)$-time) via a bottom-up dynamic programming algorithm when the input graphs are restricted to trees. Even more generally, when a graph is ‘close to being a tree’, a dynamic programming algorithm can be designed in a similar fashion. ‘Close to being a tree’ can be defined in various ways. Arguably the most prominent way is by means of treewidth. We often talk about ‘graphs of small (constant, bounded) treewidth’. This is an informal way of referring to a graph class $\mathcal{C}$, i.e. a collection of graphs, with a universal constant w such that $\mathsf{tw}(G)\leq w$ for all graph $G\in\mathcal{C}$. Recall that trees have tree width at most 1. Graphs of small treewidth, a.k.a. a class of graphs all of which have treewidth at most w for some fixed w, has many powerful properties that trees have. Admitting a linear[2] time algorithm for well-known NP-hard problems such as MAXIMUM INDEPENDENT SET is one of them.

We show that MAXIMUM INDEPENDENT SET can be solved in time $O(2^w\cdot n)$ when the input is additionally given with a tree decomposition (T,χ) of G of width at most w.

[2]The running time of a typical DP algorithms is $f(w)\cdot n$. When w is deemed as a fixed constant, such an algorithm runs in linear time (albeit there is quite a large hidden constant in the runtime function).

Theorem 1. *There is an algorithm which, given as input G and a (non-redundant) tree decomposition (T, χ) of width at most w, finds a maximum size independent set of G in time $O(w2^w \cdot n)$.*

To begin with, we assume that the given tree decomposition (T, χ) is a nice tree decomposition of width at most w and all leaf bags and the root bag are $\emptyset$. This assumption can be achieved in time $O(n)$ by applying the algorithm of Lemma 4.

We define $\mathsf{IND}_t[S]$, for each tree node t and each subset $S \subseteq \chi(t)$, as the following value:

$$(\star) \quad \mathsf{IND}_t[S] = \begin{cases} \text{the size of a maximum independent set } I \text{ of } G_t \text{ such that } I \cap \chi(t) = S, \text{ if one exists} \\ \perp \qquad \text{if no such } I \text{ exists.} \end{cases}$$

Note that $\mathsf{IND}_{\mathsf{root}}[\emptyset]$ is precisely the size of a maximum independent set of $G = G_{\mathsf{root}}$. Therefore, if we can compute the values of the table IND_t (over all subsets $S \subseteq \chi(t)$) for each tree node t, we have solved MAXIMUM INDEPENDENT SET. The plan is the following.

A. (Initialization and base case of induction) At each leaf t, compute a DP table IND_t.

B. (Recursive formula) For a forget/introduce/join node t, define a recursive formula to compute IND_t from the tables of its children

C. (Runtime analysis) Argue that computing the value of $\mathsf{IND}_t(S)$ takes constant time for each $S \subseteq \chi(t)$.

Notice that the current DP algorithm only computes the size of a maximum independent set, and does not return an actual set which achieves the maximum size. However, once the value of $\mathsf{IND}_{\mathsf{root}}(\emptyset)$ has been computed, one can backtrack the entries which leads to the size of a maximum independent set along the tree decomposition in a top-to-bottom manner. This will be discussed at the end of the section.

Initialization. Let t be a leaf node of T. For every $S \subseteq \chi(t)$, we claim

$$\mathsf{IND}_t(S) = \begin{cases} |S| & \text{if } S \text{ is an independent set,} \\ \perp & \text{otherwise.} \end{cases}$$

Since $V_t = \chi(t)$, a max independent I of G_t such that $I \cap \chi(t) = S$ is precisely S itself, if S is independent, and no such I exists if S is not independent. Therefore the above equation holds.

Forget node t. Let t' be a child of t and $\chi(t) = \chi(t') \setminus \{v\}$. We claim that the next recursive formula holds for each $S \subseteq \chi(t)$.

$$\mathsf{IND}_t(S) = \max\{\mathsf{IND}_{t'}(S), \mathsf{IND}_{t'}(S \cup \{v\})\},$$

Here, we define $\perp < 0$.

To see $\mathsf{IND}_t(S) \leq \max\{\mathsf{IND}_{t'}(S), \mathsf{IND}_{t'}(S \cup \{v\})\}$ observe that for any (maximum) independent set I of G_t with $I \cap \chi(t) = S$, either $I \cap \chi(t') = S$ or $I \cap \chi(t') = S \cup \{v\}$ holds. In particular, this means at least one of the inequalities $\mathsf{IND}(S) \leq \mathsf{IND}_{t'}(S)$ and $\mathsf{IND}(S) \leq \mathsf{IND}_{t'}(S \cup \{v\}$ holds. This proves the claimed inequality.

To establish the inequality $\mathsf{IND}_t(S) \geq \max\{\mathsf{IND}_{t'}(S), \mathsf{IND}_{t'}(S \cup \{v\})\}$, suppose that there is an independent set I in G_t with $I \cap \chi(t') = S$ and take I as such set of maximum size. Note that $I \cap \chi(t) = S$, and hence $\mathsf{IND}_t(S) \geq |I| = \mathsf{IND}_{t'}(S)$. Similarly, if there is an independent set J in G_t with $J \cap \chi(t') = S \cup \{v\}$, take J as such set of maximum size. Note that $J \cap \chi(t) = J \cap (\chi(t') \setminus \{v\}) = J \cap \chi(t') \setminus \{v\} = S$, which implies $\mathsf{IND}_t(S) \geq |J| = \mathsf{IND}_{t'}(S \cup \{v\})$. Therefore, we have $\mathsf{IND}_t(S) \geq \max\{\mathsf{IND}_{t'}(S), \mathsf{IND}_{t'}(S \cup \{v\})\}$.

Introduce node t**.** Let t' be a child of t and $\chi(t) = \chi(t') \cup \{v\}$. We claim that the next recursive formula holds for each $S \subseteq \chi(t)$.

$$\mathsf{IND}_t(S) = \begin{cases} \mathsf{IND}_{t'}(S) & \text{if } v \notin S, \\ \bot & \text{if } v \in S \text{ and } S \text{ is not an independent set,} \\ \mathsf{IND}_{t'}(S \setminus \{v\}) + 1 & \text{if } v \in S \text{ and } S \text{ is an independent set.} \end{cases}$$

We prove the equation for each case of S. Note that $G_t = G_{t'}$.

If S does not contain the newly introduced vertex v, then we have $S \subseteq \chi(t')$. From $G_t = G_{t'}$, the equation immediately holds. If S is not an independent set, then it is clear that there is no independent set I satisfying $I \cap \chi(t) = S$ and the equation correctly decides the value of $\mathsf{IND}_t(S)$.

Assume $v \in S$ and S is an independent set and we want to settle the equation $\mathsf{IND}_t(S) = \mathsf{IND}_{t'}(S \setminus \{v\}) + 1$. Note that $\mathsf{IND}_t(S) \neq \bot$ in this case as S itself is an independent set. For $\mathsf{IND}_t(S) \leq \mathsf{IND}_{t'}(S \setminus \{v\}) + 1$, for any maximum independent set I of G_t satisfying $I \cap \chi(t) = S$, $I \setminus \{v\}$ is an independent set of $G_{t'}$ satisfying $I \cap \chi(t') = S \setminus \{v\}$. Hence, we deduce

$$\mathsf{IND}_t(S) = |I| = |I \setminus \{v\}| + |\{v\}| \leq \mathsf{IND}_{t'}(S \setminus \{v\}) + 1.$$

Conversely, consider a maximum independent set J of $G_{t'}$ satisfying $J \cap \chi(t') = S \setminus \{v\}$. The key observation here is that $J \cup \{v\}$ is an independent set of G_t. If this is not the case, i.e. $J \cup \{v\}$ has an edge (in G_t), then it must be incident with v because J is an independent set of $G_{t'}$, thus of G_t. Due to Lemma 5, it follows that $N_{G_t}(v) \subseteq \chi(t)$, and $N_{G_t}(v) \cap J \subseteq \chi(t) \cap J = S \setminus \{v\}$. This contradicts the assumption that S is an independent set, and we conclude that $J \cup \{v\}$ is an independent set of G_t. Now,

$$\mathsf{IND}_t(S) \geq |J \cup \{v\}| = |J| + 1 \geq \mathsf{IND}_{t'}(S \setminus \{v\}) + 1,$$

where the first inequality is due to $J \cup \{v\}$ is an independent set of G_t satisfying $(J \cup \{v\}) \cap \chi(t) = S$.

Join node t**.** Let t_1, t_2 be the two children of t with $\chi(t) = \chi(t_1) = \chi(t_2)$. We claim that the next recursive formula holds for each $S \subseteq \chi(t)$.

$$\mathsf{IND}_t(S) = \mathsf{IND}_{t_1}(S) + \mathsf{IND}_{t_2}(S) - |S|,$$

where $\bot + z$ is defined as $\bot$ for any $z \in \{\bot\} \cup \mathbb{N}$.

Let I be a maximum independent set of G_t satisfying $I \cap \chi(t) = S$. Note that $I_i := I \cap V_{t_i}$ is an independent set of G_{t_i} satisfying $I \cap \chi(t_i) = S$ for both $i = 1, 2$. Because $V_{t_1} \cup V_{t_2} = V_t$, it holds that $I = I \cap V_t = I \cap (V_{t_1} \cup V_{t_2}) = (I \cap V_{t_1}) \cup (I \cap V_{t_2})$[3]. Therefore we can rewrite

$$\mathsf{IND}_t(S) = |I| = |I_1 \cup I_2| = |I_1| + |I_2| - |S| \leq \mathsf{IND}_{t_1}(S) + \mathsf{IND}_{t_2}(S) - |S|.$$

[3]Here we use the fact that set intersection is distributive over set union.

Here, the third equation holds because $I_1 \cap I_2 = (I \cap V_{t_1}) \cap (I \cap V_{t_2}) = I \cap (V_{t_1} \cap V_{t_2}) = I \cap \chi(t) = S$ by Lemma 6.

To see the opposite direction of the inequality, consider a maximum independent set J_i of G_{t_i} satisfying $J_i \cap \chi(t_i) = S$ for $i = 1, 2$. We claim that $J_1 \cup J_2$ is an independent set of G_t. Indeed, if this is not the case, then there is a vertex $v_i \in J_i \setminus S$ for $i = 1, 2$ such that $v_1 v_2$ is an edge of G_t. However, the existence of such an edge contradicts the property 3 of Lemma 1. This means that $J_1 \cup J_2$ is an independent set of G_t which furthermore satisfies $(J_1 \cup J_2) \cap \chi(t) = S$. Therefore,

$$\mathsf{IND}_t(S) \geq |J_1 \cup J_2| = |J_1| + |J_2| - |J_1 \cap J_2| = \mathsf{IND}_{t_1}(S) + \mathsf{IND}_{t_2}(S) - |S|.$$

Here, the last equation is justified because $J_1 \cap J_2 \subseteq V_{t_1} \cap V_{t_2} = \chi(t)$ due to Lemma 6 and thus $J_1 \cap J_2 = (J_1 \cap J_2) \cap \chi(t) = (J_1 \cap \chi(t)) \cap (J_2 \cap \chi(t)) = S$.

Recap of the algorithm, correctness and runtime. Using the recursive formula described above, the algorithm simply computes the tables IND_t in a bottom-to-top manner. Computing the table IND_t for each leaf is straightforward from the equation as one can enumerate all subsets of $\chi(t)$ and check if there is an edge in the set under consideration. This takes $2^{w+1} \cdot poly(w)$-time. The correctness of this step, i.e. the computed table IND_t matches the definition of IND_t in $(\star)$, is trivial.

The algorithm then selects a lowermost internal (i.e. non-leaf) node t for which the table IND_t is not computed yet, and compute IND_t using the recursive formula depending on the type of t. The computed table IND_t is correct, i.e. is as defined in $(\star)$, is shown above for each node type. The running time of computing IND_t for each t is $O(2^{\chi(t)} \cdot poly(w))$. As the number of nodes in T is $4w \cdot n$ by Lemma 4, the total running time of the algorithm is $O^*(2^w \cdot n)$, where the $O^*()$ notation hides the poly-logarithmic factor.

References

[1] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.