Run-Time Analysis of DP-Based Algorithm on Graphs of Bounded Treewidth - Weighted Independent Set

Ishrat Jahan Eliza - 1605089

Bangladesh University of Engineering and Technology

January 26, 2022

Treewidth Revisited

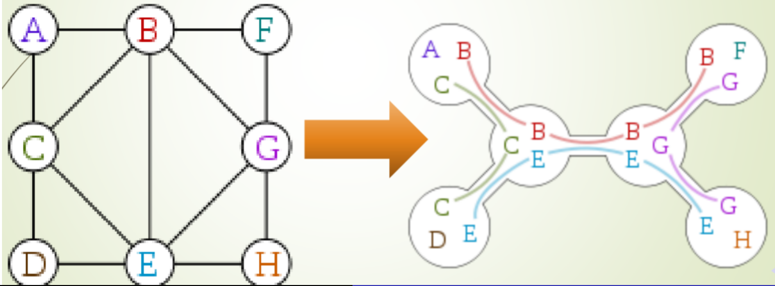
- What is treewidth of a graph?
- The treewidth of an undirected graph is a number associated with the graph that captures how similar a graph is to a tree.

Treewidth Revisited

- What is treewidth of a graph?
- The treewidth of an undirected graph is a number associated with the graph that captures how similar a graph is to a tree.

Tree Decomposition Revisited

- A tree decomposition is a mapping of a graph into a tree that can be used to define the treewidth of the graph.
  - Definition: a tree decomposition of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, where T is a tree whose every node t is assigned a vertex subset $X_t \subseteq V(G)$.
  - The following three conditions hold:
    - (T1) $\bigcup_{t \in V(T)} X_t = V(G)$
    - (T2) For every $uv \in E(G)$, there exists a node t of T such that bag X_t contains both u and v.
    - (T3) For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$, induces a connected subtree of T.
-
-



Treewidth by Tree Decomposition

- After we defined what a tree composition is, we can define the treewidth of a graph.
- The width of tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ equals $\max_{t \in V(T)} |X_t| - 1$,
- The treewidth of a graph G, denoted by $tw(G)$, is the minimum possible width of a tree decomposition of G.

In our given Example, $max|X_t| = 3$. So, treewidth will be

$$max|X_t| - 1 = 3 - 1 = 2$$

Nice Tree Decompostion

We will think of a nice tree decomposition as rooted trees. A (rooted) tree decomposition (T, $\{X_t\}_{t \in V(T)}$) is nice if the following conditions are satisfied:

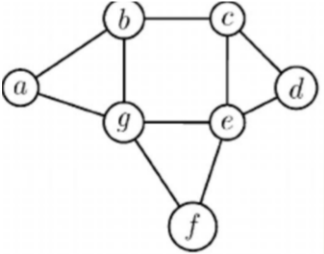
- $X_r = \emptyset$ for r the root of T and $X_l = \emptyset$ for every leaf l of T.
- Every non-leaf node of T is of one of the following three types:
  - Introduce node: a node t with exactly one child t' such that X_t $= X_{t'}$ U {v} for some vertex $v \notin X_{t'}$ (we say that v is introduced at t).
  - Forget node: a node t with exactly one child t' such that $X_t = X_{t'} - \{w\}$ for some vertex w (we say that w is introduced at t).
  - Join node: a node t with two children t1, t2 such that $X_t = X_{t1} = X_{t2}$

Nice Tree Decomposition (Lemma)

Lemma 7.4. *If a graph G admits a tree decomposition of width at most k, then it also admits a nice tree decomposition of width at most k. Moreover, given a tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ of G of width at most k, one can in time $\mathcal{O}(k^2 \cdot \max(|V(T)|, |V(G)|))$ compute a nice tree decomposition of G of width at most k that has at most $\mathcal{O}(k|V(G)|)$ nodes.*

Weighted Independent Set Dynamic Programming Algorithm

Let G=(V,E) be a graph of n-vertex with width of at most k.

- Let T1 = (T, $X_{t\,t \in V(T)}$) be a tree decomposition of G.
- By applying Lemma 7.4 we can assume that T1 is a nice tree decomposition.
- 

G

Weighted Independent Set Dynamic Programming Algorithm (Contd.)

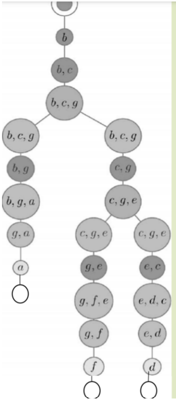


Figure 1: Nice tree decomposed tree: T1

Weighted Independent Set Dynamic Programming Algorithm (Contd.)

- Among independent sets I satisfying $I \cap X_t = S$ for some fixed S, all the maximum-weight solutions have exactly the same weight of the part contained in V_t.
- For every node t and every $S \subseteq X_t$, define the following value: c[t,S] = Maximum possible weight of a set S' such that $S \subseteq S' \subseteq V_t$, $S' \cap X_t$, where S' is independent.
- If no such set S' exists, then we put $c[t, S] = \infty$ (iff S is not independent).
- Final solution is c[r,$\emptyset$].
- We can solve this recursively using bottom-up DP approach.

Weighted Independent Set Dynamic Programming Algorithm (Contd.)

Leaf node. If t is a leaf node, then we have only one value $c[t, \emptyset] = 0$.

Weighted Independent Set Dynamic Programming Algorithm (Contd.)

Introduce node. Suppose t is an introduce node with child t' such that $X_t = X_t' \cup \{v\}$ for some $v \notin X_t'$.
Let S be any subset of X_t. If S is not independent, then we can immediately put $c[t,S] = -\infty$; hence assume otherwise.
Then we claim that the following formula holds:

$$c[t,S] = \begin{cases} c[t',S] & \text{if } v \notin S; \\ c[t', S \setminus \{v\}] + \mathbf{w}(v) & \text{otherwise.} \end{cases}$$

Weighted Independent Set Dynamic Programming Algorithm (Contd.)

Forget node. Suppose t is a forget node with child t' such that $X_t = X_{t'} \setminus \{w\}$ for some $w \in X_t$. Let S be any subset of X_t; again we assume that S is independent, since otherwise we put c[t, S] = $-\infty$.
We claim that the following formula holds:

$$c[t,S] = \max\left\{c[t',S], c[t',S \cup \{w\}]\right\}.$$

Weighted Independent Set Dynamic Programming Algorithm (Contd.)

Join node. Finally, suppose that t is a join node with children t_1, t_2 such that $X_t = X_{t1} = X_{t2}$. Let S be any subset of X_t;
as before, we can assume that S is independent.
The recursive formula is as follows:

$$c[t,S] = c[t_1,S] + c[t_2,S] - \mathbf{w}(S).$$

Run-Time Analysis of the Algorithm

- We have a graph G with n vertices and treewidth of at most k, which means $|X_t| \leq k+1$ for every node t
- Thus for every node t, we compute $2^{|X_t|} \leq 2^{k+1}$ values of c[t,S].
- In naive solution we will say that each c[t,S] computed in $n^{O(1}$ time. It is possible to construct a data structure that allows performing adjacency queries in time O(k), so computing each c[t,S] will take only $k^{O(1)}$ time.
- We assumed that the number of nodes of the given tree decomposition is O(kn) (lemma 7.4).
- The total running time of the algorithm is $2^k.k^{O(1)}.n$.

Run-Time Analysis of the Algorithm

- We have a graph G with n vertices and treewidth of at most k, which means $|X_t| \leq k + 1$ for every node t
- Thus for every node t, we compute $2^{|X_t|} \leq 2^{k+1}$ values of c[t,S].
- In naive solution we will say that each c[t,S] computed in $n^{O(1}$ time. It is possible to construct a data structure that allows performing adjacency queries in time O(k), so computing each c[t,S] will take only $k^{O(1)}$ time.
- We assumed that the number of nodes of the given tree decomposition is O(kn) (lemma 7.4).
- The total running time of the algorithm is $2^k.k^{O(1)}.n$.

Run-Time Analysis of the Algorithm

- We have a graph G with n vertices and treewidth of at most k, which means $|X_t| \leq k+1$ for every node t
- Thus for every node t, we compute $2^{|X_t|} \leq 2^{k+1}$ values of c[t,S].
- In naive solution we will say that each c[t,S] computed in $n^{O(1}$ time. It is possible to construct a data structure that allows performing adjacency queries in time O(k), so computing each c[t,S] will take only $k^{O(1)}$ time.
- We assumed that the number of nodes of the given tree decomposition is O(kn) (lemma 7.4).
- The total running time of the algorithm is $2^k.k^{O(1)}.n$.

Run-Time Analysis of the Algorithm

- We have a graph G with n vertices and treewidth of at most k, which means $|X_t| \leq k+1$ for every node t
- Thus for every node t, we compute $2^{|X_t|} \leq 2^{k+1}$ values of c[t,S].
- In naive solution we will say that each c[t,S] computed in $n^{O(1}$ time. It is possible to construct a data structure that allows performing adjacency queries in time O(k), so computing each c[t,S] will take only $k^{O(1)}$ time.
- We assumed that the number of nodes of the given tree decomposition is O(kn) (lemma 7.4).
- The total running time of the algorithm is $2^k.k^{O(1)}.n$.

Run-Time Analysis of the Algorithm

- We have a graph G with n vertices and treewidth of at most k, which means $|X_t| \leq k + 1$ for every node t
- Thus for every node t, we compute $2^{|X_t|} \leq 2^{k+1}$ values of c[t,S].
- In naive solution we will say that each c[t,S] computed in $n^{O(1}$ time. It is possible to construct a data structure that allows performing adjacency queries in time O(k), so computing each c[t,S] will take only $k^{O(1)}$ time.
- We assumed that the number of nodes of the given tree decomposition is O(kn) (lemma 7.4).
- The total running time of the algorithm is $2^k.k^{O(1)}.n$.

Observations (Obtained Theorem)

Let G be an n-vertex graph with weights on vertices given together with its tree decomposition of width at most k. Then the Weighted Independent Set problem in G is solvable in time $2^k.k^{O(1)}.n$.

Observations (Corollary)

Let G be an n-vertex graph given together with its tree decomposition of width at most k. Then one can solve the Vertex Cover problem in G in time $2^k.k^{O(1)}.n$.

Improvement of the Running Time

It depends on the following optimizations:

- How fast we can implement all the low-level details of computing formulas for c[t,S], e.g., iteration through subsets of vertices of a bag X_t. This in particular depends on how we organize the structure of G and T1 in the memory.
- How fast we can access the exponential-size memory needed for storing the dynamic-programming table c[·,·].

Thank You!