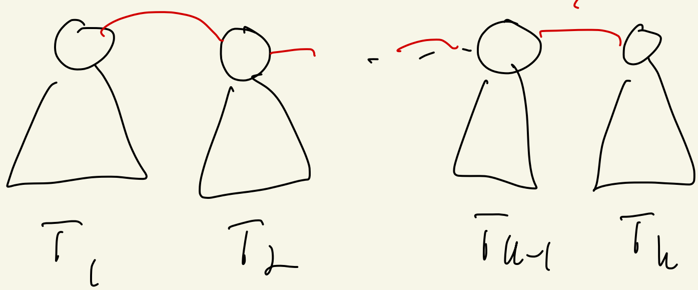
Proposition 10.5 If X is a separator of G and $tw(G-X) \le t$ then $tw(G) \le t + |X|$

proof:

Let $(\{X_i \mid i \in I_1\}, T_1) \dots (\{X_i \mid i \in I_k\}, T_k)$ be tree-decompositions of the connected components $G_1, G_2, \dots, G_k$ of $G-X$

- add the vertices of X to all bags in the k tree-decompositions and join $T_1, T_2 \dots T_k$:

new edges in T



The largest bag in $\{X_i \mid i \in I_1 \cup \dots \cup I_k\}$ has size $|X| + t$ where t is the maximum width of the k t.d. above.

Constructing good tree-decomposition

Theorem Deciding for given G, k whether $tw(G) \le k$ is NP-complete

We know that we can find $tw(G)$ in pol. time when G is chordal as $tw(G) = \omega(G) - 1$ and we can construct an optimal tree-decomp in pol time:

This follows from the proof of (i) ⇒ (iii) in Theorem 4.8 of Golumbic chapter 4

Theorem (Bodlaender 1996) There is an algorithm running in time $O(f(k)(n+m))$ for construction an optimal t-d of given G when k is fixed

$f(k)$ grows very fast ☹

In practice we just need a heuristic to find a t-d. of G with width close to $tw(G)$ or at least width at most $c\,tw(G)$ for some constant c

∃ 4-approximation algorithm for tree-width which runs in time $O(c^{tw(G)})$ for some constant c. So it is exponential in $tw(G)$

General approach for finding a good tree-decomp:

1. If G is chordal, use the algorithm that follows from the proof of Thm 4.8 in Golumbic

2. If $G=(V,E)$ is <u>not</u> chordal we try to make it chordal by adding a small set of new edges E' so that $G'=(V,E\cup E')$ <u>is</u> chordal and then proceed as in 1.

Here we use the important property that $tw(H) \geq tw(H')$ for every subgraph (induced or not) H' of H:

if $(\{X_i \mid i\in I\}, T)$ is a tree-decomp for H

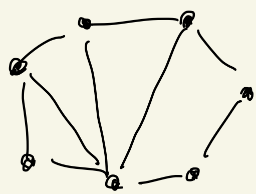
then $(\{X_i' \mid i\in I\}, T)$ is a tree-decomp for H'

when $X_i' = X_i \cap V(H')$

Important: We don't need an optimal t.d. of G in order for the algorithms we shall see to work.
It is only the running time which gets worse if the width of our decomposition is larger (running time grows exponentially with the width)

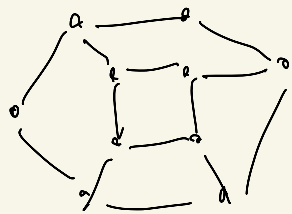
Planar Graphs (just a few remarks)

a graph G is outerplanar if it has a planar embedding in which every vertex is adjacent to the outer face



all outerplanar G are hamiltonian

- r-outerplanar:
  - 1-outerplanar = outerplanar
  - For $r > 1$ G minus vertices on outer cycle is $(r-1)$-outerplanar

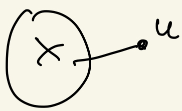


2-outerplanar

Known fact:

For a planar graph G, the size of a minimum vertex cover, the size of a minimum dominating set and $tw(G)$ are linearly related

[$X \subseteq V(G)$ is a dominating set if each vertex $u \in V - X$ is adjacent to X



]

<u>Theorem 10.13</u> Let G be a planar graph

If G has a vertex cover or a dominating set of size k then $tw(G) \in O(\sqrt{k})$

This is best possible in terms of the exponent:

For the $k \times k$-grid we saw that the tree-width is k while we clearly have that min VC and min dom. set are both of size $O(k^2)$

10.4 Dynamic programming for vertex cover

The algorithm below finds an optimal vertex cover of G no matter which tree-decomposition of G we use. It is only the running time which depends on the width of the tree-decomposition we use.

The running time of the algorithm depends exponentially on the width of the decomposition but if this is small the algorithm works fine even for large graphs!

Theorem 10.14 Given $G=(V,E)$ and a tree-decomposition $(\{X_i \mid i \in I\}, T)$ of G which has width w, we can find an optimal vertex cover in time $O(2^w \cdot w \cdot |I|)$

Proof:

central fact: There are at most $2^{|X_i|}$ possible vertex covers of $G[X_i]$ for each bag X_i

goal: Combine them vertex covers (efficiently!) via T

For each X_i we associate a table A_i

A_i has 2^{n_i} rows when $n_i = |X_i|$

Then rows corresponds to all the different subsets of X_i

$X_i = \{x_{i_1}, x_{i_2}, \dots x_{in_i}\}$

The j'th row corresponds to the j'th subset X_i^j of X_i

For each row j we also associate a number $m_i(X_i^j)$ when

row	x_{i_1}	x_{i_2}	x_{i_3}	$\cdots$	x_{in_i}
1	0	0	0	$\cdots$	1
2	0	0	0	$\cdots$ 1	0
	0	0	0	$\cdots$ 1	1
j	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
2^{n_i}	1	1	1	$\cdots$ 1	1

$$\boxed{m_i(X_i^j) = \min \{ |V'| \mid V' \subseteq V \text{ is a vertex cover of } G \text{ and } V' \cap X_i = X_i^j \}}$$

So $m_i(X_i^j)$ is the size of a smallest VC of G whose intersection with X_i is precisely X_i^j.

Clearly if X_i^j is not a vertex cover of $G[X_i]$, then there is no such V' above and hence we set $m_i(X_i^j) = \infty$

Step 1 table initialization (setting values of $m_i(C_i)$'s)

For every $i \in I$ and every row j of A_i:

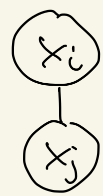
$$m_i(X_i^j) := \begin{cases} |X_i^j| & \text{if } X_i^j \text{ is a VC of } G[X_i] \\ \infty & \text{otherwise} \end{cases}$$

Note that once an '∞' appears this subset can never be used

Step 2 Dynamic programming

method: process decomposition tree $(\{X_i \mid i \in I\}, T)$ from leaves toward the root (root T arbitrarily)

with updating data for X_i via a child X_j:



Rename such that $X_i = \{z_1, z_2, \ldots, z_s, v_1, v_2, \ldots v_{t_i}\}$

$$X_j = \{z_1, z_2, \ldots z_s, u_1, u_2, \ldots u_{t_j}\}$$

where $X_i \cap X_j = \{z_1, z_2 \ldots z_s\}$

Consider a mapping $C : Z \to \{0,1\}$ and say that a mapping C_p (from X_i or X_j to $\{0,1\}$) agrees with C on Z if $C_p(w) = C(w)$ $\forall w \in Z$

In terms of subsets: the subset X_i^p of X_i agrees with the subset Z' of Z if and only if $X_i^p \cap Z = Z'$

$\forall$ subset $Z' \subseteq Z$:

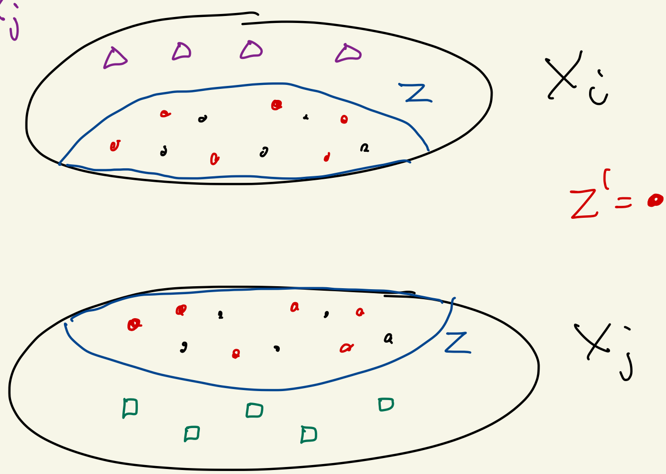
$\forall$ subset $X_i^p \subseteq X_i$ s.t $X_i \cap Z = Z'$

$$m_i(X_i^p) := m_i(X_i^p) + \min\{ m_j(X_j^q) \mid X_j^q \cap Z = Z' \} - |Z'|$$

$\triangle$ = vertices of $X_i^p - X_j$

$\odot = X_i^p \cap Z$
$= Z'$
$= X_j^q \cap Z$

$\square$ = vertices of $X_j^q - X_i$

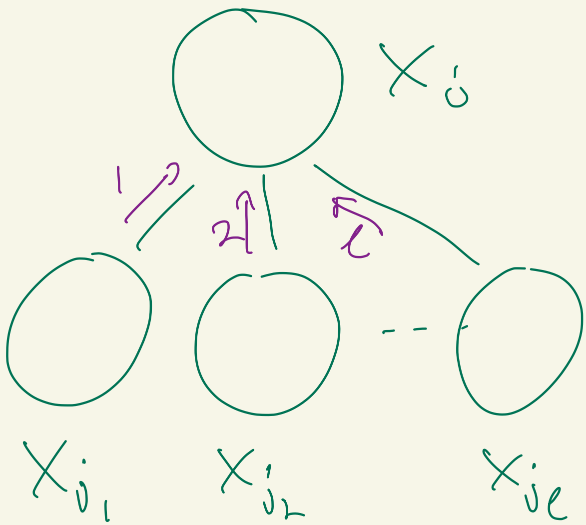


We count red vertices twice in $m_i(X_i^p) + m_j(X_j^q)$ so we substract $|Z'|$

Notes:

- We can use bitvector to keep track of subsets of X_i, X_j and Z
- We can save a pointer to the row q of A_j which give the minimum above

When X_i has children $X_{j_1}, X_{j_2}, \ldots X_{j_\ell}$ we update A_i against each of $A_{j_1}, A_{j_2} \ldots A_{j_\ell}$ successively:



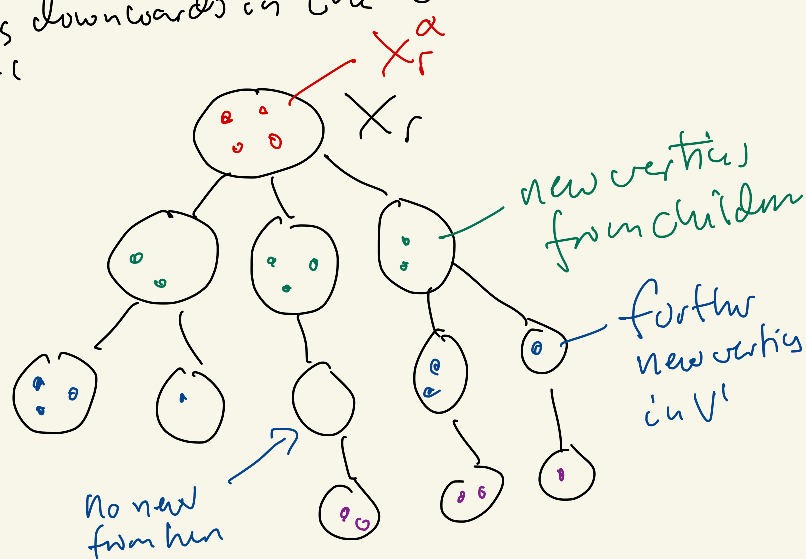
Step 2 continues until the root has been completely updated

Step 3 Constructing a minimum vertex cover

The size of a minimum vertex cover can be found from $\min\{m_r(X_r^\alpha) \mid \alpha \text{ row of } A_r\}$ where X_r is the root bag of T

We can construct a minimum VC V' as follows

- Start be setting $V' := X_r^\alpha$ where row α of A_r gave the minimum above
- Using the pointers which show which cover for each child caused an update of $m_r(X_r^\alpha)$ we can collect all vertices of V' that are in the children of X_r
- Continuing downwards in the tree we collect all of V'

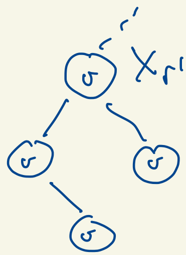


Correctness:

a) As $V = \bigcup_{i \in I} X_i$ every vertex of G has been considered for inclusion in V'

b) $\forall e \in E$ we have $e \in X_i$ for some i so every edge is covered as we only deal with subsets X_i^p of X_i for which $m_i(X_i^p) < \infty$

c) By the consistency property, the bags of T which contain a vertex v form a subtree T_v of T and via the rooting of T in r we get a unique rooting of T_v



Hence v is in V' precisely when $v \in X_{r'}^q$ where this was the row of $A_{r'}$ used to update the parent of $X_{r'}$

By step 2 the value of $m_{r'}(X_{r'}^q)$ is updated precisely from those subsets of the children of $X_{r'}$ which contain v. So v ends up in V' only if it is in all the updating subsets of T_v

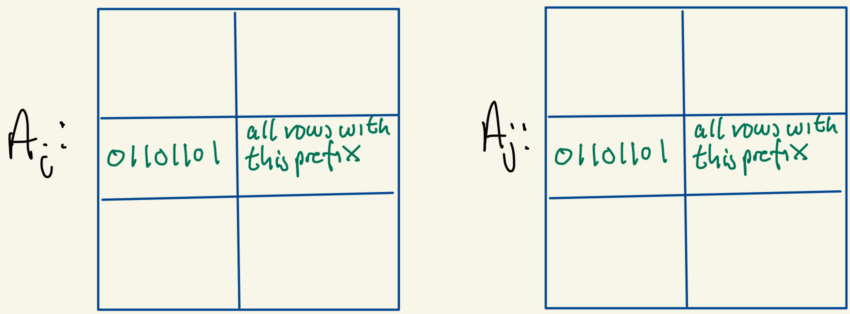
Running time

Updating A_i via A_j:

First find $Z = X_i \cap X_j$ and sort element of X_i, X_j such that $X_i = \{z_1, z_2 .. z_s, v_1, v_2 .. v_{t_i}\}$ $Z = \{z_1, .., z_s\}$

$$X_j = \{z_1, z_2 .. z_s, u_1, u_2, ..., u_{t_j}\}$$

By lexicographic sorting we can order A_i and A_j such that



Go through A_i from row 1 to row $2^{|X_i|}$

For each set of rows whose prefix is the same subset Z' of Z

Go through the rows of A_j whose Z-prefix correspond to Z' and find best subset (row) X_j^q (the one with the lowest $m_j(X_j^q)$)

For every subset (row) X_i^p of X_i with prefix Z' set $m_i(X_i^p) = m_i(X_i^p) + m_j(X_j^q) - |Z'|$

This takes time $O(2^w w)$:

- We can use sorting to arrange A_i and A_j as we wanted them in time $O(2^w w)$ as $|X_i|, |X_j| \leq w+1$ and $|A_i| = 2^{|X_i|}$ $|A_j| = 2^{|X_j|}$
- After sorting we just visit each row of A_i, A_j once to collect first X_j^q and then update $m_i(X_i^p)$ for all p such that $X_i^p \cap Z = X_j^q \cap Z$.

T has $|I|-1$ edges so the total time is $O(2^w w |I|)$

$\square$.