Exact Algorithms for NP-hard problems

Advanced Algorithms: Part 2, Lecture 4

Today

- Discuss Dynamic Programming homework assignment
- Dynamic Programming over a tree decomposition
  - Some practice with & properties of tree decompositions
  - Vertex cover over a tree decomposition
  - Nice tree decompositions (vertex cover)
  - Constructing a tree decomposition & clique tree

Mathijs de Weerdt

Woeginger, exercise 33: Scheduling with precedence constraints **and release times**

Given

- 1-machine, set J of n jobs, each with a length p_j *and a release time r_j*
- precedence constraints (partial order), i.e. i precedes j iff $i \rightarrow j$

Find

- non-preemptive schedule with completion times C_j for each job j
- obeying precedence constraints and release times, and with
- minimum sum of completion times $\Sigma_j^n C_j$

Q. Recursive formulation for optimal value for jobs S *without* release times?

$OPT[S] = \min_{j \in LAST(S)} \{ OPT[S-\{j\}] + p(S) \}$

where LAST(S) is set of jobs in S without successor in S and $p(S)=\Sigma_{i \in S} p_i$.

Q. How to additionally deal with the release times?

TU Delft

Woeginger, exercise 33: Scheduling with precedence constraints **and release times**

without release times we had:
$OPT[S] = \min_{j \in LAST(S)} \{ OPT[S-\{j\}] + p(S) \}$
where $LAST(S)$ is set of jobs in S without successor in S and $p(S)=\Sigma_{i \in S} p_i$

with release times a job can be scheduled with a *gap* (wait until release):

- So completion time is not just sum of earlier processing times.
- Let T[S] denote the completion time of optimally scheduling all jobs in S.

Q. Then OPT[S] = …?
$OPT[S] = \min_{j \in LAST(S)} \{ OPT[S-\{j\}] + T[S] \}$

Q. How to express T[S] recursively?
Hint: it's the completion time of a job… which job? How to compute?

TU Delft

Woeginger, exercise 33: Scheduling with precedence constraints **and release times**

without release times we had:
$OPT[S] = \min_{j \in LAST(S)} \{ OPT[S-\{j\}] + p(S) \}$
where $LAST(S)$ is set of jobs in S without successor in S and $p(S)=\Sigma_{i \in S} p_i$

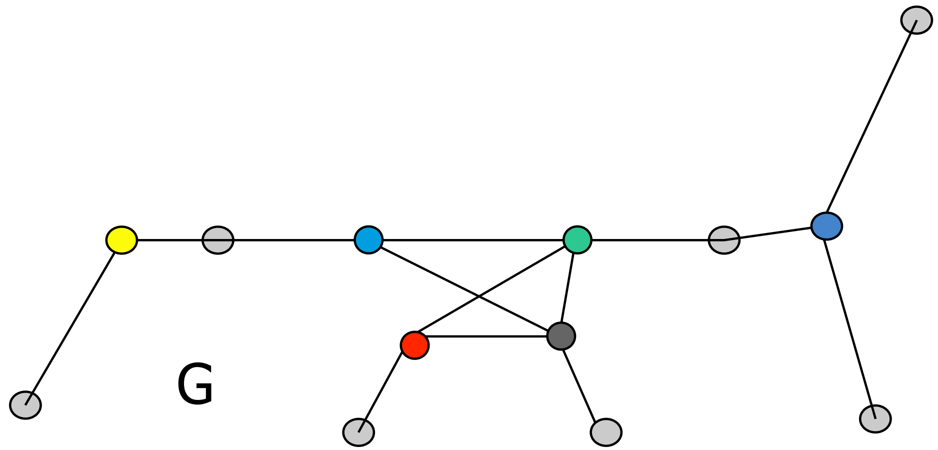
with release times a job can be scheduled with a *gap* (wait until release):
- So completion time is not just sum of earlier processing times.
- Let $T[S]$ denote the completion time of optimally scheduling all jobs in S.

$OPT[S] = \min_{j \in LAST(S)} \{ OPT[S-\{j\}] + T[S] \}$

A. $T[S] = f(j^*)$
where $j^* = \arg\min_{j \in LAST(S)} \{ OPT[S-\{j\}] + f(j) \}$
where $f(j) = \max(T[S-\{j\}], r_j) + p_j$

Recall: Tree decomposition

Q. Given a graph G, what is (the definition of) a tree decomposition?

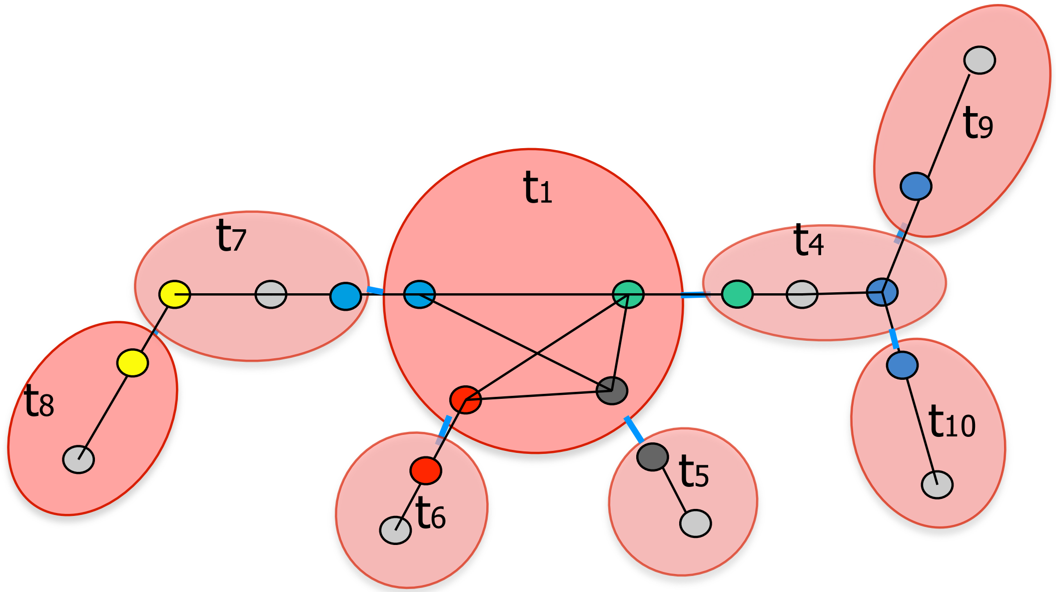


Recall: Tree decomposition

Definition

A tree decomposition of a graph $G = (V, E)$ is
- a pair $(Tr = (T,F), \{V_t : t \in T\})$
- where Tr is a tree such that
  - $\bigcup_{t \in T} V_t = V$ (vertex coverage)
  - $\{u,v\} \in E \Rightarrow \{u,v\} \subseteq V_t$ for some $t \in T$ (edge coverage)
  - $\forall v \in V : T_v = \{t \in T : v \in V_t\}$ is connected in Tr (coherence)

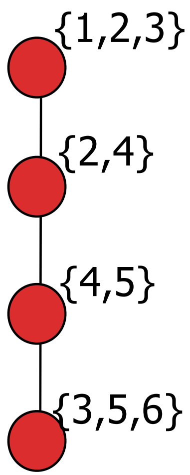
Main properties:
When removing tree node/edge, subgraphs:
- Share no vertices
- Share no edges



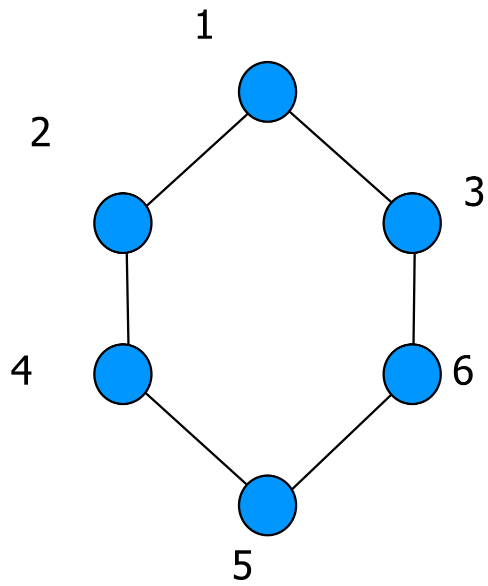
Q. Why is the pair (Tr = (T,F), $\{V_t : t \in T\}$) not a tree decomposition of G?

A. It contradicts coherence: 3 occurs in bags of top and bottom node of the tree, but not in bags of all other nodes on (tree) path between these.

tree (T,F) with bags V_t:

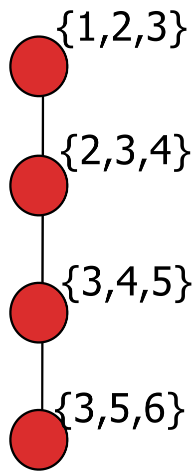


graph G:

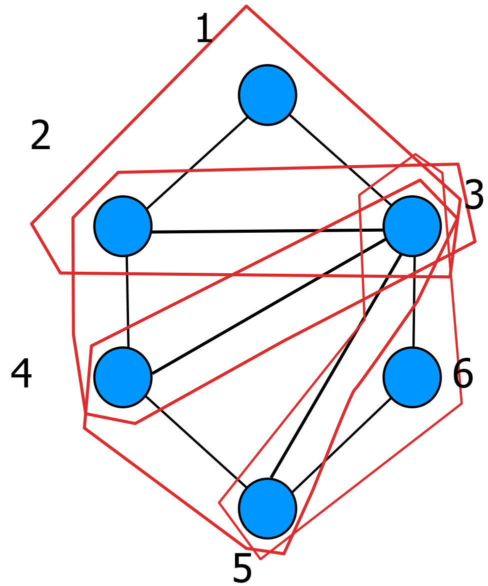


Q. Why is the pair ($Tr = (T,F)$, $\{V_t : t \in T\}$) not a tree decomposition of G?

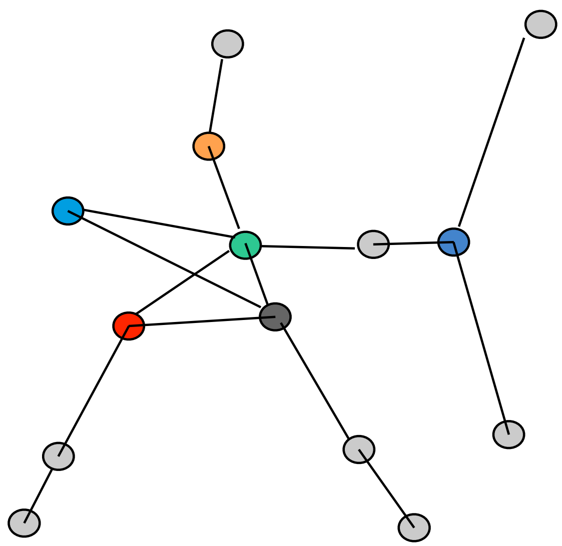
tree (T,F) with bags V_t:



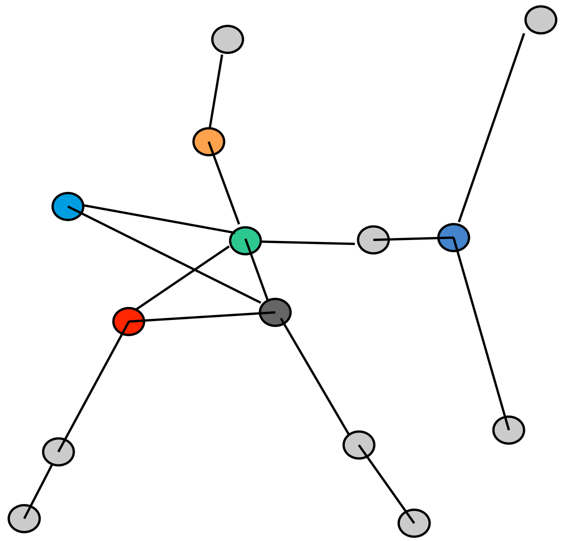
graph G:

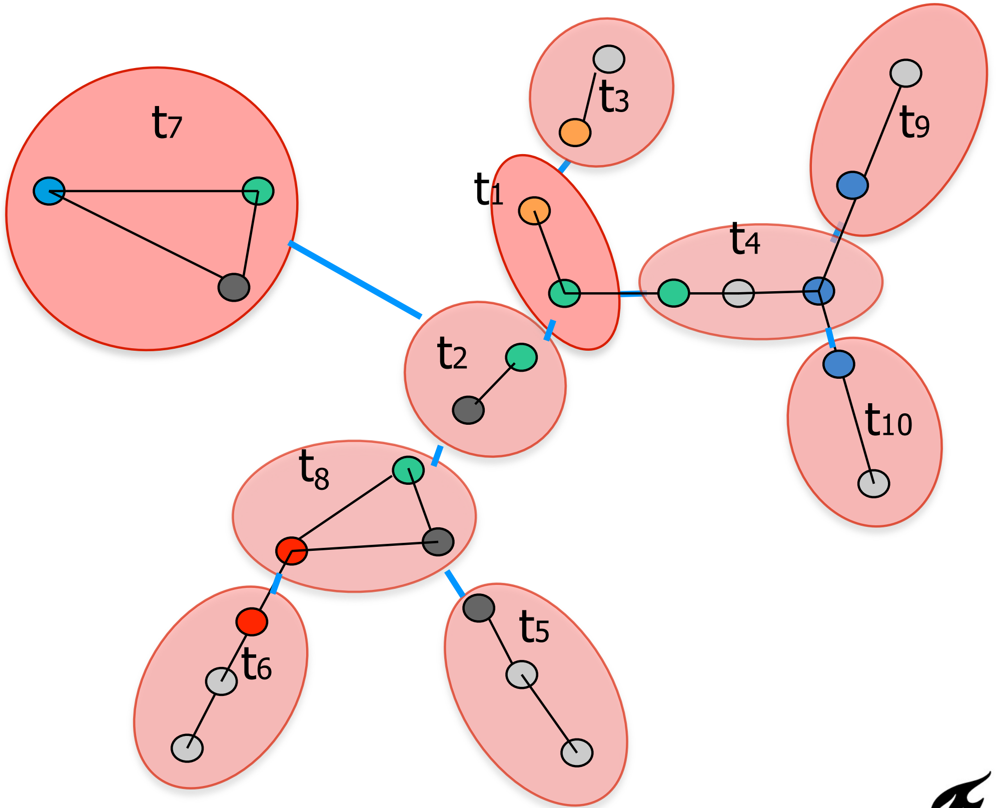


Q. What is the treewidth of the following graph?

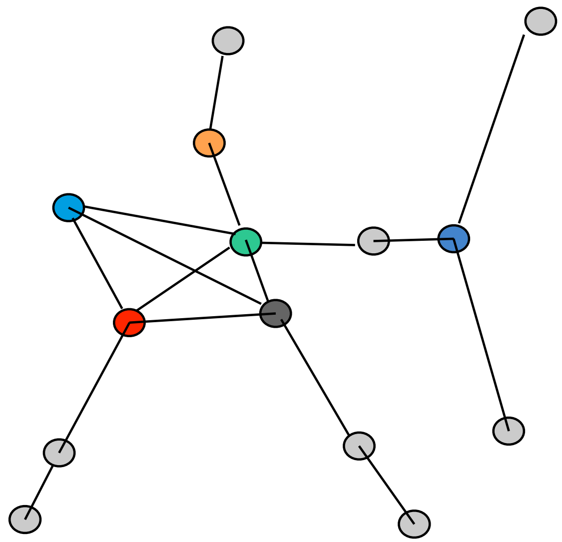


Q. What is the treewidth of the following graph?





Q. What is the treewidth of the following graph?



Treewidth: some easy observations

to support reasoning about tree decompositions and treewidth

Q. If H is a subgraph of G what do we know about the relation between their treewidths?

Treewidth: some easy observations

Observation 1
If H is a subgraph of G then $tw(H) \leq tw(G)$.

Take the tree decomposition of G with minimum width and consider the subset $\{V_t \cap H : t \in T\}$. This generates a tree decomposition T′ of H. Observe: $width(T') \leq tw(G)$ and thus $tw(H) \leq tw(G)$.

Q. If $G = (V, E)$ has two unconnected components A and B such that $A \cup B = V$ what do we know about $tw(G)$?

Treewidth: some more easy observations

Observation 1

If H is a subgraph of G then tw(H) ≤ tw(G).

Take the tree decomposition of G with minimum width and consider the bags $\{V_t \cap H: t \in T\}$.
This generates a tree decomposition of H with at most the same width.

Observation 2

If G =(V, E) has two unconnected components A and B such that $A \cup B = V$ then tw(G) = max{tw(A), tw(B)}

Take tree decompositions of A and of B with minimum width,
Take a (root) node V_a from the tree decomposition of A and a root node V_b from B.
Connect V_a and V_b. This is a tree decomposition of G with width max{tw(A),tw(B)}.
No smaller tree decomposition exists, because then one would for either A or B (using observation 1).

Maximum Treewidth

Q. For any graph $G = (V, E)$, wat is the maximum treewidth?

Hint: Use *constructive* proof: provide "algorithm" to produce tree decomposition with this width.

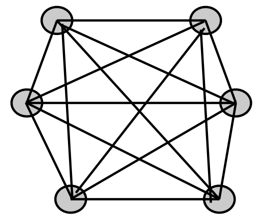
Observation 3

For every graph $G = (V, E)$ it holds that $tw(G) \leq |V|-1$.

For every $G = (V, E)$ a single bag $V_t = V$ forms a tree $Tr = (\{t\}, \emptyset)$ of width $|V| - 1$

Q. For which graph $G = (V, E)$ do you think this is the smallest treewidth possible?

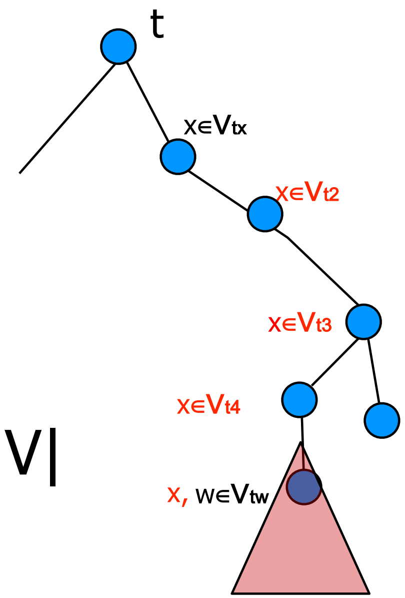
Maximum Treewidth



Observation 4: Let G = (V, E) be a clique. Show that tw(G) = |V|-1

Proof *(idea: point out which tree node contains all vertices)*

1. Take an arbitrary tree decomposition T for G with root node t.
2. For every $v \in V$, find the node t_v closest to t such that $v \in V_{tv}$.
3. Then take the $w \in V$ associated with the tree node t_w with maximum distance from t.

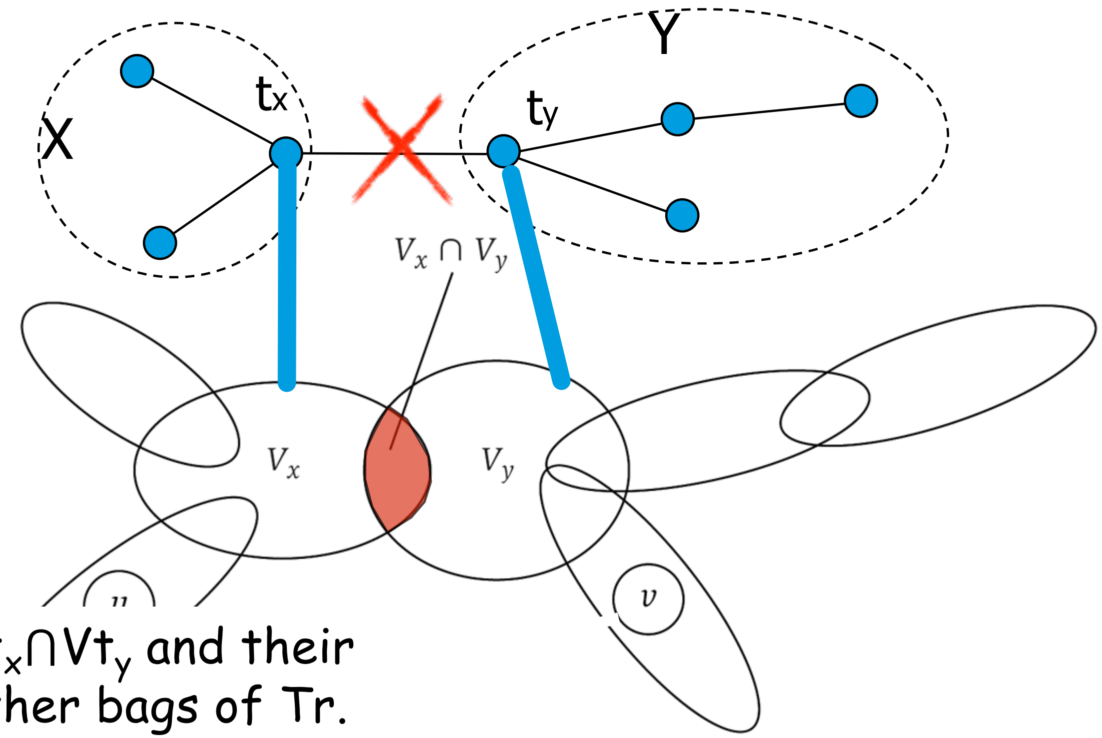


4. For every x the following holds:
   since G is a clique, edge (x,w) needs to be covered,
   and thus x should be somewhere in the subtree of V_{tw}.

5. So every other vertex x should be in V_{tw} (by coherence): $|V_{tw}| = |V|$
   Together with Observation 1, this implies tw(G) = |V|-1.

Tree decomposition properties: separating tree edge

Observation 5. Let a tree decomposition ($Tr=(T,F)$, $\{V_t : t \in T\}$) of $G=(V,E)$ be given. **Remove a tree-edge (t_x,t_y) from T*.** Let resulting components of T be X and Y. Remove $V_x \cap V_y$ from G.
Are the components $G_1 = G_X - (V_x \cap V_y)$ and $G_2 = G_Y - (V_x \cap V_y)$ separated?



*) Remove respective vertices $Vt_x \cap Vt_y$ and their incident edges from G and the other bags of Tr.

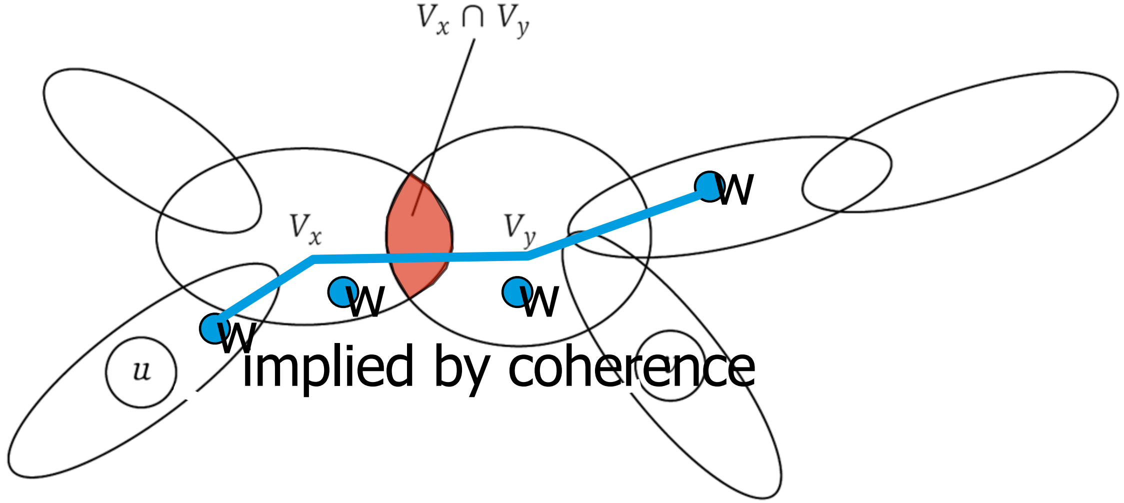
Tree decomposition properties: separating tree edge

Observation. Let a tree decomposition ($Tr=(T,F)$, $\{V_t : t \in T\}$) of $G=(V,E)$ be given. Remove a tree-edge (t_x,t_y) from T. Let resulting components of T be X and Y. Remove $V_x \cap V_y$ from G.

Are the components $G_1 = G_X - (V_x \cap V_y)$ and $G_2 = G_Y - (V_x \cap V_y)$ separated?

Q. Why don't G_1 and G_2 share a vertex?

Follows from coherence: Suppose $w \in V_1 \cap V_2$. Let t_1 and t_2 be the respective tree nodes in X and Y containing w. Then every tree node t on the path in T from t_1 to t_2 contains w. But then $w \in V_x$ and $w \in V_y$. So $w \notin V_1 \cap V_2$.



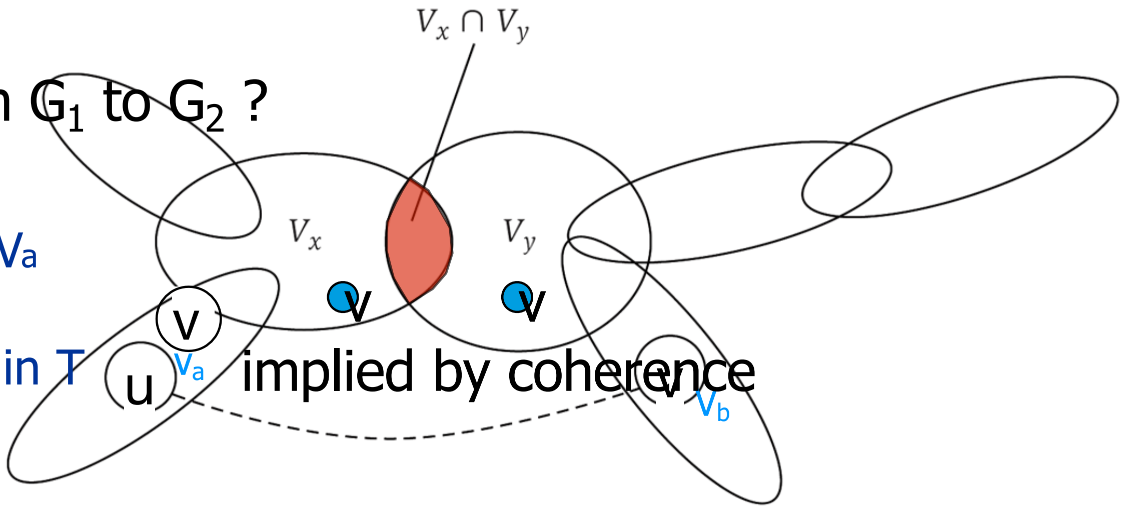
Tree decomposition properties: separating tree edge

Observation. Let a tree decomposition ($Tr=(T,F)$, $\{V_t : t\in T\}$) of $G=(V,E)$ be given. Remove a tree-edge (t_x,t_y) from T. Let resulting components of T be X and Y. Remove $V_x \cap V_y$ from G.
Are the components $G_1 = G_X - (V_x \cap V_y)$ and $G_2 = G_Y - (V_x \cap V_y)$ separated?

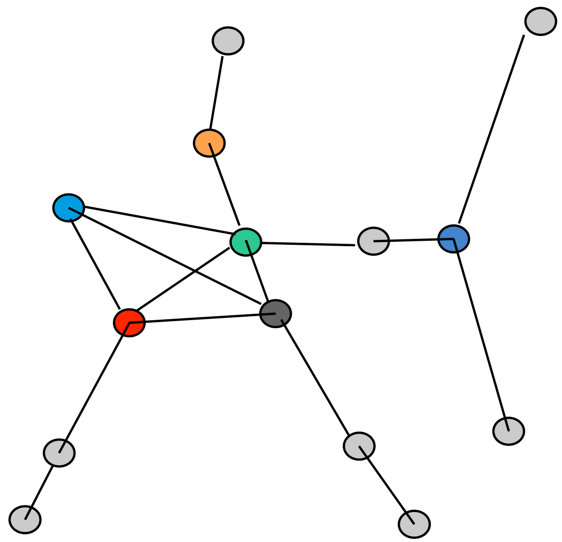
Pf. G_1 and G_2 don't share any vertex:
Follows from coherence: Suppose $w \in V_1 \cap V_2$. Let t_1 and t_2 be the respective tree nodes in X and Y containing w. Then every tree node t on the path in T from t_1 to t_2 contains w. But then $w \in V_x$ and $w \in V_y$. So $w \notin V_1 \cap V_2$.

Q. Why is there no edge (u,v) from G_1 to G_2 ?
Follows from edge coverage:

1. $\{u,v\}$ implies a node $a\in T$ with $u,v \in V_a$
2. w.l.o.g. let V_a be in X
3. hence $v\in V_z$ for every z on path a - b in T
4. so $v \in V_x \cap V_y$; contradiction



Q. Add edge (red,blue). Which of the following statements can we use to prove that the treewidth of this graph is at least 3? Give all that apply.



A. For every graph G = (V, E) it holds that $tw(G) \leq |V|-1$.
B. If H is a subgraph of G then $tw(H) \leq tw(G)$.
C. If G =(V, E) has two unconnected components A and B such that $A \cup B = V$ then $tw(G) = \max\{tw(A), tw(B)\}$.
D. Let G = (V, E) be a clique. Then $tw(G) = |V|-1$.

TU Delft

Dynamic programming over tree decomposition

First, similar to last week's lecture on max. weight independent set:

- min. weight vertex cover over a tree
- min. weight vertex cover over a nice tree decomposition

Weighted Vertex cover

Given
- an undirected graph G=(V,E)
- weights w_u for every u in V
- a nonnegative integer k

Decide
- is there a subset of vertices $C \subseteq V$ with sum of weights less than k such that each edge in E has one endpoint in C?

Note: The following discusses the same idea of DP using a tree decomposition, but for a minimum weighted vertex cover (=of all edges) instead of maximum weighted independent set (=no neighbors).

Start again simple: what if G is a tree itself?

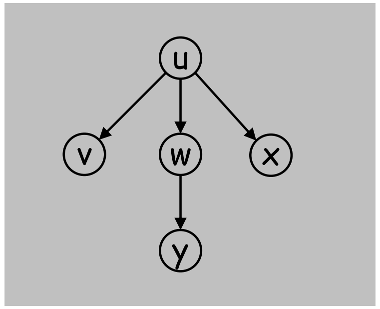
TU Delft

Minimum Weighted Vertex Cover on Trees

Q. What are possible subproblems for a node u?

The two subproblems are:

1. include u and *possibly include* children of u, or
2. don't include u and *include* all children of u.



children(u) = { v, w, x }

Idea. Use different notation for OPT with and without a node u.

- OPT_{in} (u) = min weight of vertex cover subtree rooted at u, containing u.
- OPT_{out}(u) = min weight of vertex cover subtree rooted at u, not containing u.

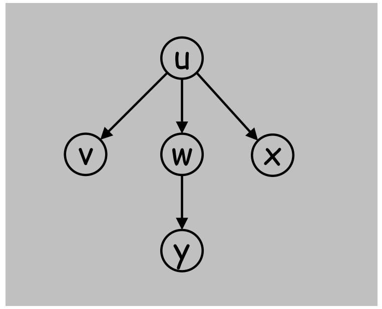
Q. How to express these formally (recursively)?

Weighted Vertex Cover on Trees

determine value for
each decision (in/out)
on the root node

Idea. Use different notation for OPT with and without u.

- $OPT_{in}(u)$ = min weight of vertex cover subtree rooted at u, containing u.
- $OPT_{out}(u)$ = min weight of vertex cover subtree rooted at u, not containing u.



Q. How to express these formally (recursively)?

$$OPT_{in}(u) = w_u + \sum_{v \in \text{children}(u)} \min\left\{OPT_{in}(v), OPT_{out}(v)\right\}$$

$$OPT_{out}(u) = \sum_{v \in \text{children}(u)} OPT_{in}(v)$$

NB: Σ… = 0 if u has no children

Q. How to compute optimum of whole problem?

$$OPT(u) = \min\left\{OPT_{in}(u), OPT_{out}(u)\right\}$$

Weighted Vertex Cover on Trees: DP Algorithm

Claim. The following dynamic programming algorithm efficiently finds a minimum weighted vertex cover in trees.

```
Weighted-Vertex-Cover-In-A-Tree(T) {
   Root the tree at a vertex r
   foreach (vertex u of T in postorder) {
      if (u is a leaf) {
         M_in [u] = w_u
         M_out[u] = 0
      }
      else {
         M_in [u] = Σ_{v∈children(u)} min(M_out[v], M_in[v]) +  w_u
         M_out[u] = Σ_{v∈children(u)} M_in[v]
      }
   }
   return min(M_in[r], M_out[r])
}
```

start from bottom:
ensures a vertex is visited after
all its children

Q. What is the space and runtime of this algorithm?
A. Takes O(n) space and time: O(1) time amortized per vertex since we visit vertices in postorder and examine each edge exactly once. ▪

Weighted Vertex Cover: dynamic programming over a tree decomposition

Vertex cover. Given a graph G=(V,E), find a set S⊆V of minimum weight that covers all edges.

Idea. Similar to Maximum Weight Independent Set:

1. Use a tree decomposition T of G to construct a search tree.
2. Do dynamic programming over T to find minimum weight vertex covers of subgraph of G (induced by all pieces/bags in tree with root V_t).
3. Brute force over all vertex covers in every piece/bag of T.

Instead of $OPT_{in}(t)$ and $OPT_{out}(t)$ we now define OPT(t,U) where U contains the vertices from V_t that are selected in the cover.

Weighted Vertex Cover: dynamic programming over a tree decomposition

Express weight of minimum vertex cover recursively using children:

OPT(t,U) = w(U) + $\Sigma_{i=1,..,d}$ *minimum weights of vertex covers of graphs induced by subtrees with roots at V_{ti}, consistent with U*

Minimize weight over *all* vertex covers $U_i \subseteq V_{ti}$ in G *consistent with* U.
(Nodes selected by U in $V_t \cap V_{ti}$ should be the same as nodes selected by U_i in $V_t \cap V_{ti}$, so U_i should be a vertex cover for which $U_i \cap V_t = U \cap V_{ti}$.)

OPT(t,U) = w(U) +

$$\sum_{i=1}^{d} min_{U_i \subseteq V_{t_i}} \left\{ \begin{array}{c} OPT(t_i, U_i) - w(U_i \cap U) \\ \text{such that } U_i \cap Vt = U \cap Vti \text{ and for every } \{u, v\} \in E \cap Vti \;\; u \text{ or } v \text{ is in } U_i \end{array} \right\}$$

the second condition ensures U_i is a vertex cover of V_{ti}

Dynamic programming over a tree decomposition

```
To find a ~~maximum weight independent set~~ minimum weight vertex cover of G,
given a tree decomposition (T, {V_t}) of G:

  Root T at a node r
  For each node t of T in post-order
    If t is a leaf then
      For each independent set U of V_t
        f_t(U) = w(U)
    Else
      For each independent set U of V_t
        f_t(U) is determined by the recurrence
    Endif
  Endfor
  Return max {f_r(U) : U ⊆ V_r is independent}.
```

NB: $f_t(U) = OPT(t, U)$

(with table look-ups)

Q. Given a graph with n nodes, and a tree decomposition of width w. What is the *space* required by this algorithm?

A. For a given tree node t, we store a value for each vertex cover U: $O(2^{w+1})$ with at most n tree nodes this is thus $O(n2^{w+1})$.

TU Delft

Dynamic programming over a tree decomposition

```
        minimum weight vertex cover
To find a ~~maximum weight independent set~~ of G,
given a tree decomposition (T, {V_t}) of G:

  Root T at a node r
  For each node t of T in post-order
      If t is a leaf then
          For each independent set U of V_t
              f_t(U) = w(U)
      Else
          For each independent set U of V_t
              f_t(U) is determined by the recurrence
      Endif
  Endfor
  Return max {f_r(U) : U ⊆ V_r is independent}.
```

NB: $f_t(U) = OPT(t, U)$

(with table look-ups)

Q. Given a graph with n nodes, and a tree decomposition of width w. What is the *runtime* of this algorithm?

A. One calculation of $OPT_t(U)$ takes $O(2^{w+1}wd)$, where d is #children.
Needs to be done for each vertex cover U: $O(2^{w+1})$ times.
So $O(4^{w+1}wn)$, because |T| is at most O(n) children in total.

Weighted Vertex Cover: dynamic programming over a tree decomposition

Express weight of minimum vertex cover recursively using children:

OPT(t,U) = w(U) + $\Sigma_{i=1,..,d}$ *minimum weights of vertex covers of graphs induced by subtrees with roots at V_{ti}, consistent with U*

Minimize weight over *all* vertex covers $U_i \subseteq V_{ti}$ in G *consistent with* U.

(Nodes selected by U in $V_t \cap V_{ti}$ should be the same as nodes selected by U_i in $V_t \cap V_{ti}$, so U_i should be a vertex cover for which $U_i \cap V_t = U \cap V_{ti}$.)

OPT(t,U) = w(U) +

$$\sum_{i=1}^{d} min_{U_i \subseteq V_{t_i}} \left\{ \begin{array}{c} OPT(t_i, U_i) - w(U_i \cap U) \\ \text{such that } U_i \cap Vt = U \cap Vti \text{ and for every } \{u, v\} \in E \cap Vti \;\; u \text{ or } v \text{ is in } U_i \end{array} \right\}$$

the second condition ensures U_i is a vertex cover of V_{ti}

complex; to define & to guarantee correctness…

TU Delft

Dynamic programming over a *nice* tree decomposition

Reasoning about (constructing an algorithm for) subsets and multiple children can become quite complex…

Suppose the tree decomposition is *nice*:

- just one child with a small change in the bag or
- two children which are equivalent

To give the required recursive function $OPT_t(U)$ representing the minimum weight of a vertex cover *consistent* with U would be much easier.

Nice tree decomposition

For complicated dynamic programming algorithms (and their proofs), *nice* tree decompositions make life easier. See (Bodlaender, 1997).

Definition

A rooted tree decomposition $(Tr = (T,F), \{V_t : t \in T\})$ of G is nice if for every $t \in T$:

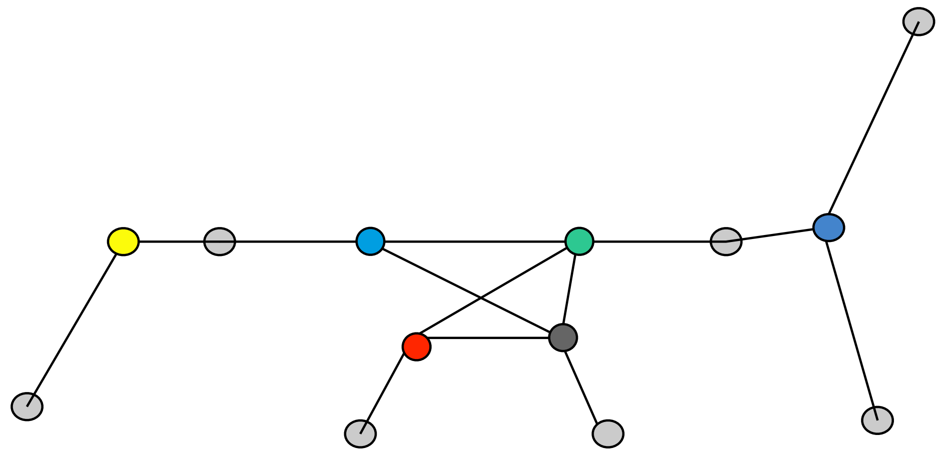
Note: terms reason from bottom to top

- $|V_t| = 1$ (leaf), or
- t has one child t′ with $V_t \subset V_{t'}$ and $|V_t| = |V_{t'}| - 1$ (*forget*), or
- t has one child t′ with $V_{t'} \subset V_t$ and $|V_t| = |V_{t'}| + 1$ (*introduce*), or
- t has two children t_1 and t_2 with $V_t = V_{t1} = V_{t2}$ (*join*).

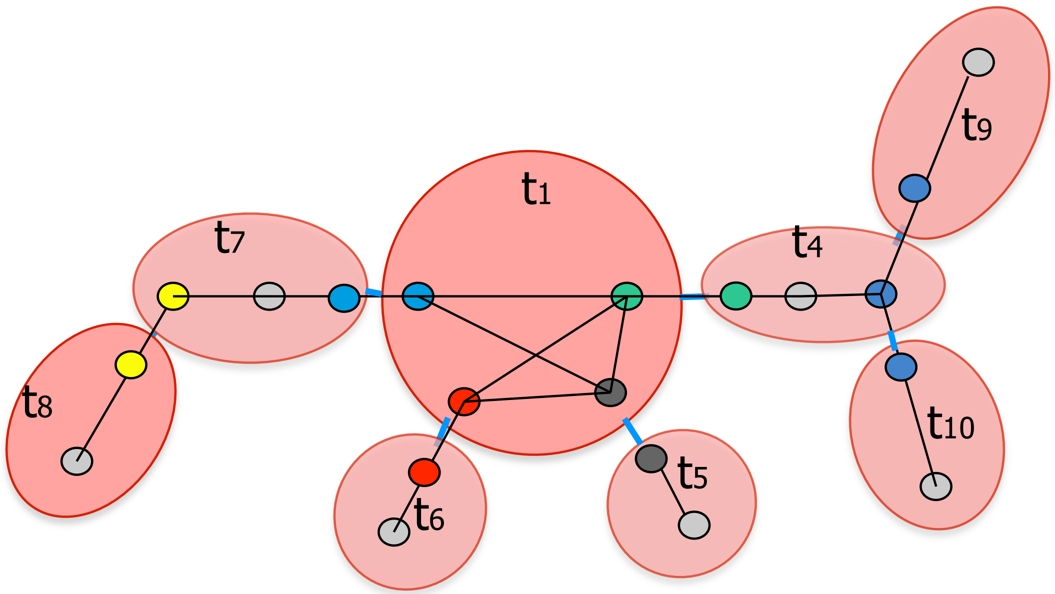
Given a tree decomposition of width w of G, in polynomial time we can construct a nice tree decomposition $(Tr = (T,F), \{V_t : t \in T\})$ of G of width w, with $|T|$ in $O(wn)$, where $n = |V(G)|$.

Bodlaender, H. Treewidth: Algorithmic Techniques and Results. In *Mathematical Foundations of Computer Science*, pages 19-36, 1997.

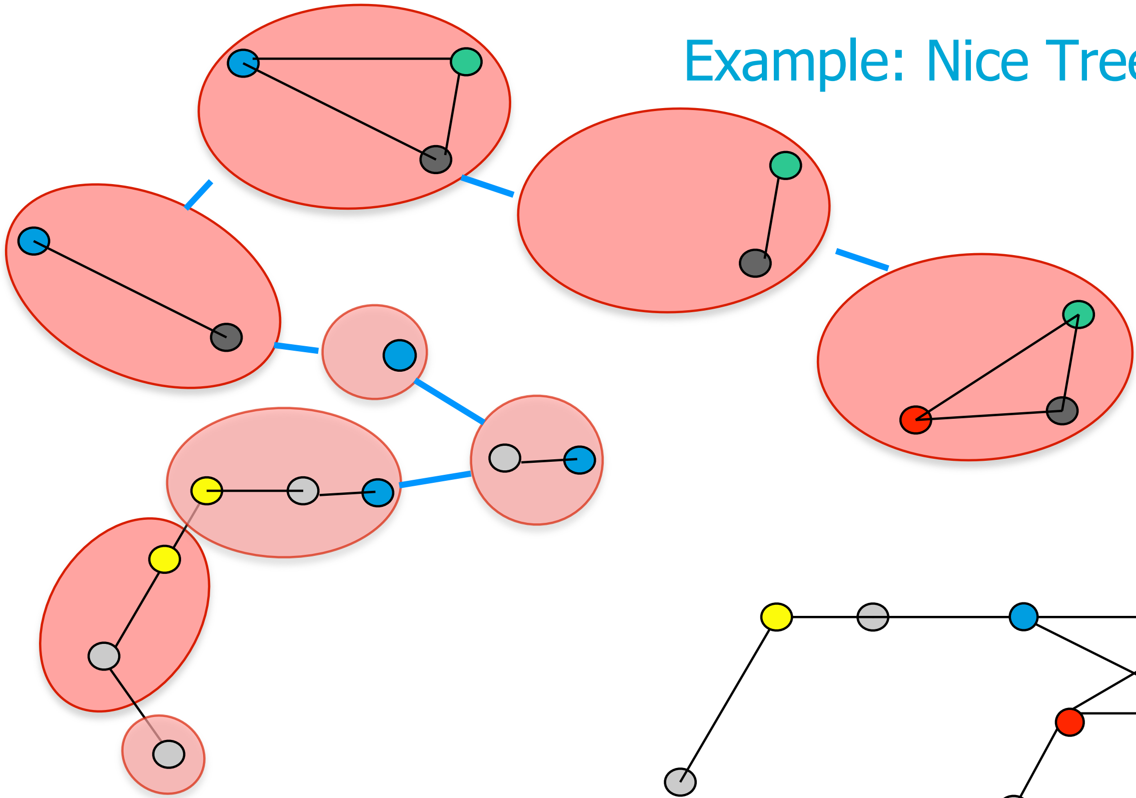
Example: Nice Tree decomposition

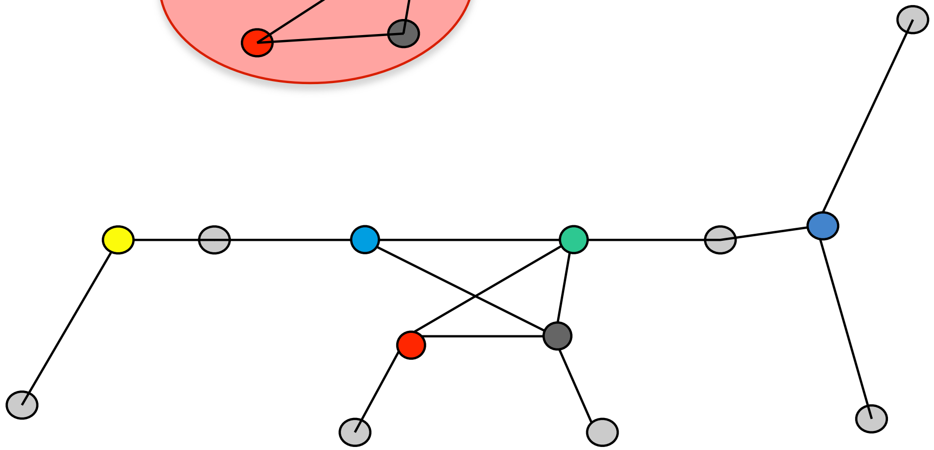


Recall: Tree decomposition

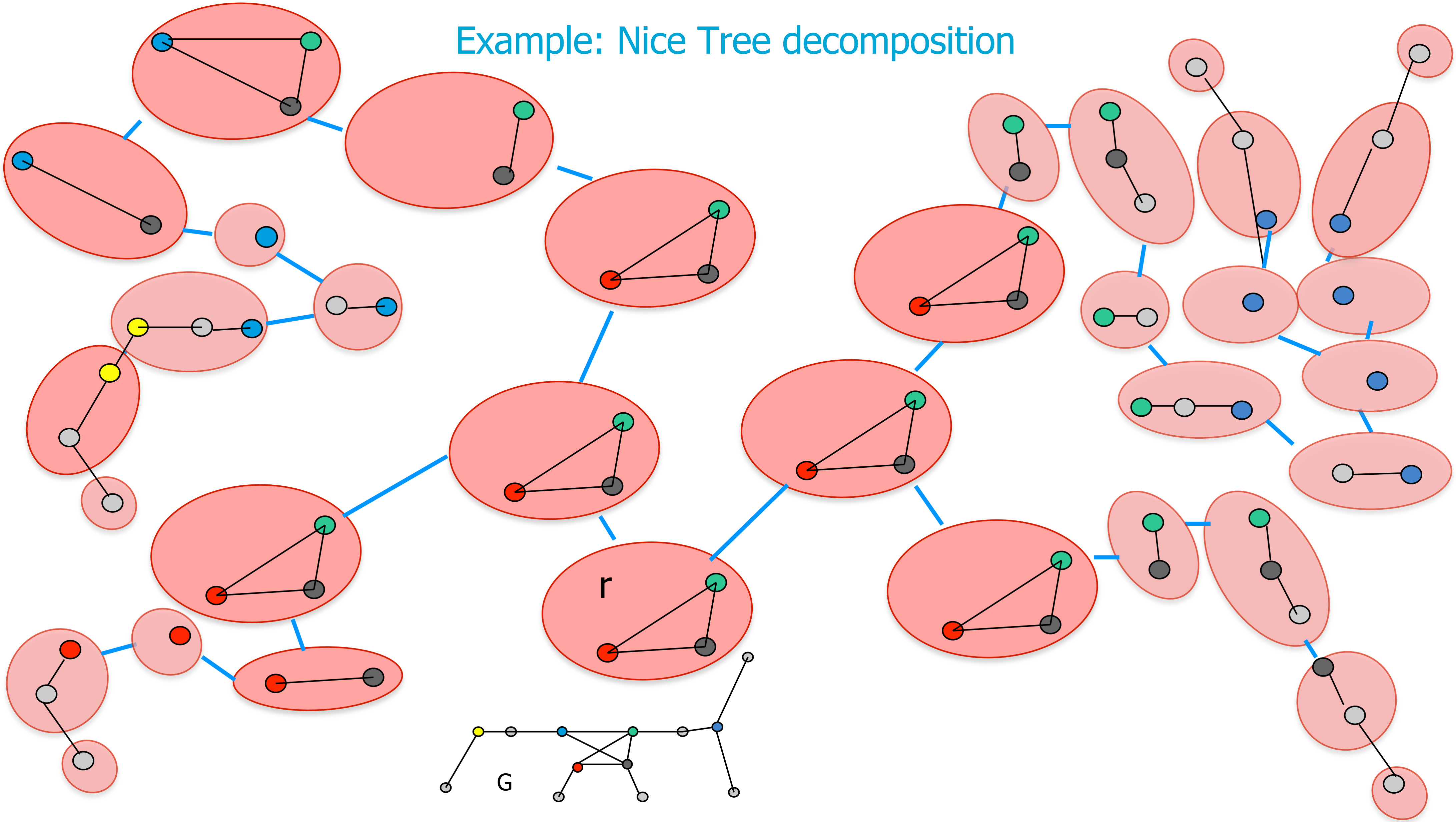


Example: Nice Tree decomposition





Example: Nice Tree decomposition



Weighted vertex cover over nice tree decomposition

Before we had $OPT_{in}(u)$ and $OPT_{out}(u)$ to represent the optimal values for the decision on u. Now we need the optimal values for **all subsets** U of V_t.

Leaf: $|V_t| = 1$. Q. Define OPT(t,U) for every U.

$OPT(t,\{v\}) = w(\{v\})$ for $V_t=\{v\}$

$OPT(t,\emptyset) = 0$

Pf. Vt contains one vertex, so ∅ and {v} are both covers and cost are determined.

Forget: $V_t = V_{t'} \setminus \{v\}$ (So parent V_t does not have v)

$OPT(t,U) = \min\{ OPT(t',U), OPT(t',U\cup\{v\}) \}$

Pf. two options consistent with U; consider both and choose the best

Introduce: $V_t = V_{t'}\cup\{v\}$ (So parent V_t includes v)

$$OPT(t,U) = \begin{cases} OPT_{t'}(U) & \text{if } v \notin U \text{ and all neighbors in } V_t \text{ are in } U \\ OPT(t',U\setminus\{v\}) + w(v) & \text{if } v \in U \\ \infty & \text{if } v \notin U \text{ and not all neighbors in } V_t \text{ are in } U \end{cases}$$

Pf. if v not in U: need to make sure U is a cover (1st and 3rd line); otherwise, add weight of v and consistent subproblem

Join: two children t_1 and t_2 with $V_t = V_{t1} = V_{t2}$

$OPT(t,U) = OPT(t_1,U) + OPT(t_2,U) – w(U)$

Pf. add weights of two consistent subproblems and remove overlap

Q. How to obtain minimal weight from root node r?

A. $\min_{U\subseteq Vr} \{ OPT(r,U) \}$

Constructing a tree decomposition (Ch.10.5)

Bad news. Constructing a tree decomposition of width less than k is an NP-complete problem.

Good news.

- There are efficient algorithms for special cases (e.g. chordal graphs).
- There is a FPT algorithm that is linear in n (but exponential in k) and produces a tree decomposition of *linear size* (Bodlaender, 1996)
- There is a ratio 4 approximation of $O^*(2^w)$ (in Ch.10.5).
- There is a polynomial time O(log n) approximation.
- There are good and very fast heuristics (e.g. minimum degree).

Reproducing algorithm from Ch.10.5 not required at exam.

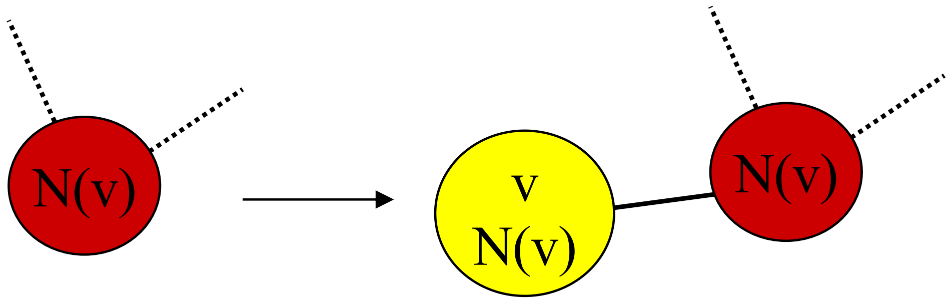
Bodlaender, Hans L. "A Linear-time Algorithm for Finding Tree-decompositions of Small Treewidth." *SIAM Journal on computing* 25, no. 6 (1996): 1305-1317.

The minimum degree heuristic

min-degree(G):

- If G is a clique, create node t with all vertices and return T=({t},{})
- Otherwise:
  - Take vertex *v* of minimum degree
  - Make neighbors of *v* a clique
  - Remove *v*, and repeat on rest of G: T' = min-degree(G – {v})
  - Create node t_v with bag {v}∪N(v), connect to node of tree decomposition T' containing neighbors
  - return T'∪t_v

It's a heuristic, but often works well! (Try it on the cycle graph.)
NB: Also reason about a lower bound as we did earlier!

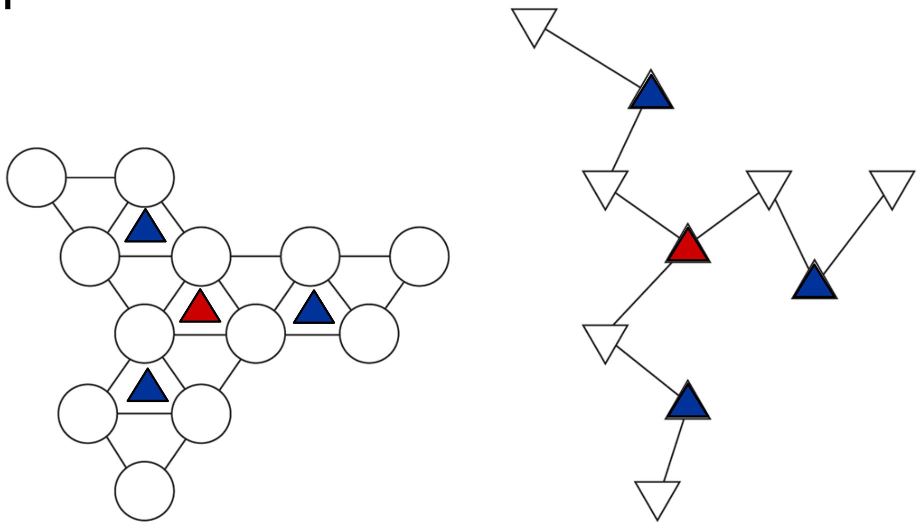


TU Delft

Special case: Clique tree

Definition

A *clique tree* of G is a tree decomposition (Tr = (T,F), $\{V_t : t \in T\}$) where every bag V_t is a clique in G.



(b) (c)

Q. Any tree decomposition of G can be made into a clique tree of some graph G′ with V′⊇V and E′⊇E. How?

A. connect all vertices in every bag

Optional: All-Pairs-Shortest-Paths (own research, time permitting)

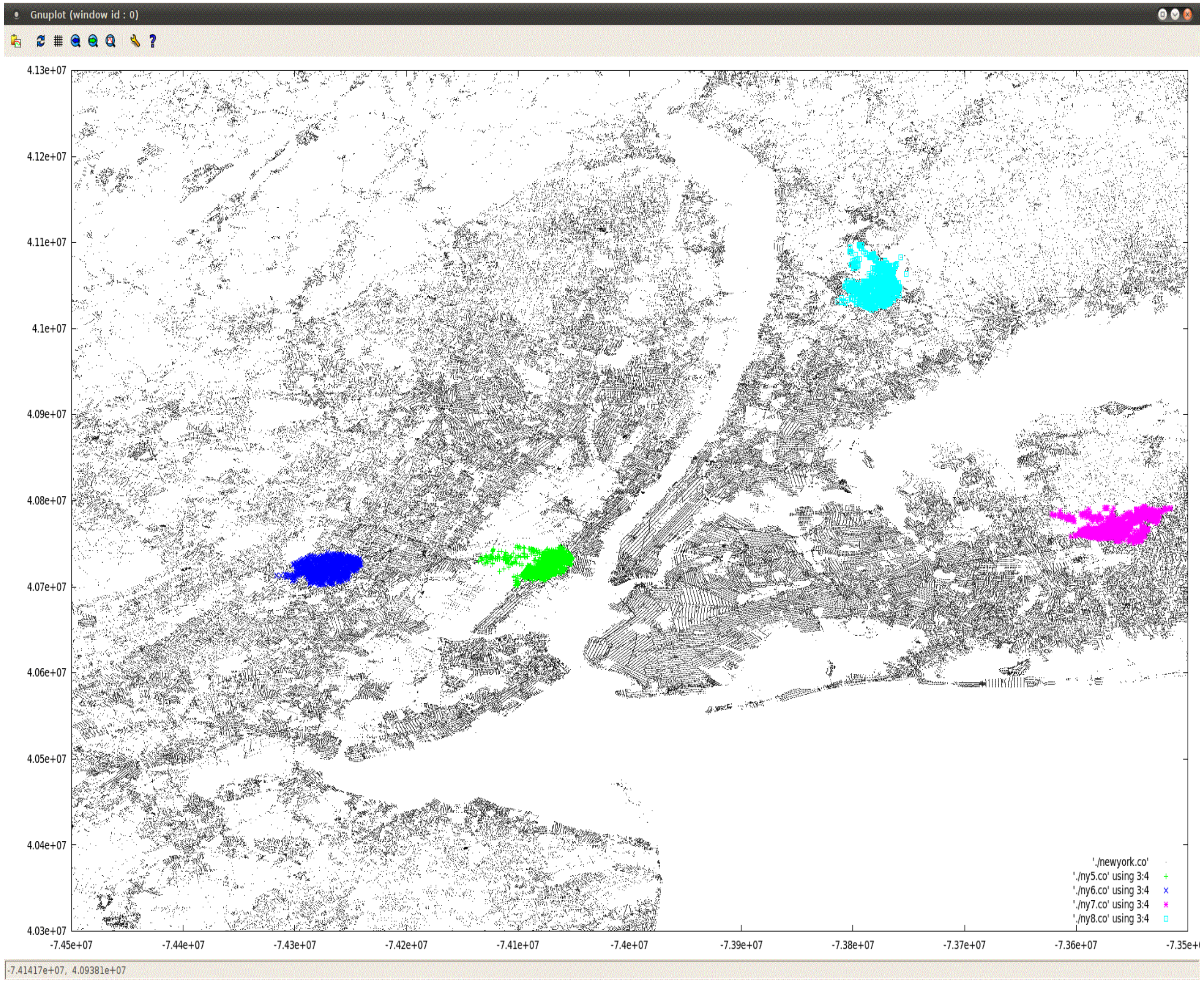
Tree decomposition (clique trees) can be used for solving problems in P !

- Floyd-Warshall (1962, dynamic programming) $O(n^3)$
- Johnson (1977, Bellman-Ford, Dijkstra) $O(nm + n^2 \log n)$
- Chleq / Snowball by Planken, de Weerdt (2011) $O(n^2 w_d)$

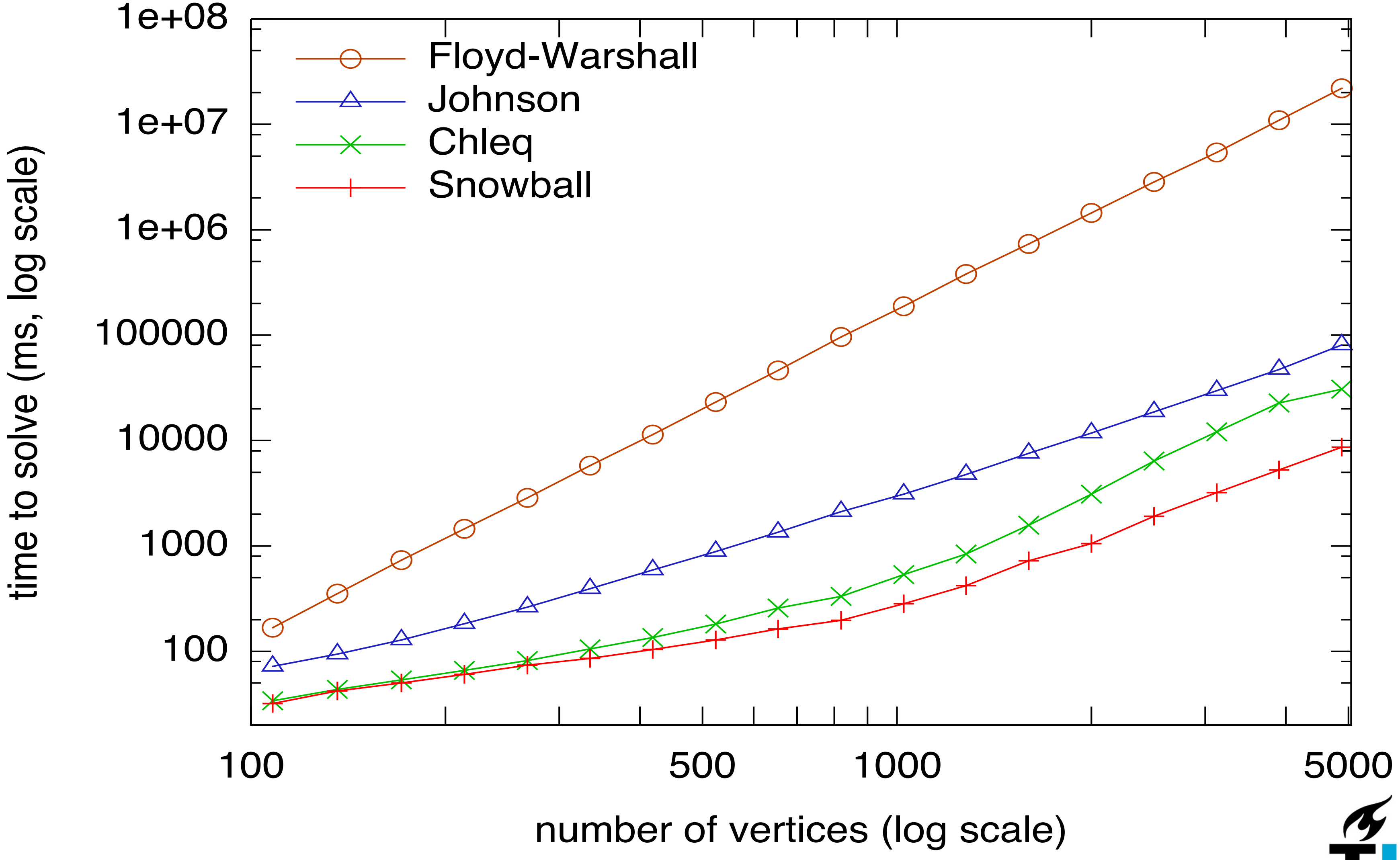
where w_d is the width of the clique tree.

Snowball is theoretically better than Johnson if w_d is $o(\log n)$.

All-Pairs-Shortest-Paths (own research)



All-Pairs-Shortest-Paths (own research)



TU Delft

1-Slide Summary on Tree Decomposition

Tree decomposition

- *(nice) tree decomposition* is a tree of *bags* defined on top of a graph
  - meeting *vertex coverage*, *edge coverage* and *coherence* conditions
- children represent subproblems that are *independent* apart from parent
- runtime is exponential in size of the bags (the *width*)
- *treewidth* tw(G) of a graph G is smallest width of any tree decomposition

Properties (with proofs):

- For every graph G = (V, E) it holds that tw(G) ≤ |V|-1.
- If H is a subgraph of G then tw(H) ≤ tw(G).
- If G =(V, E) has two unconnected components A and B such that A∪B = V then tw(G) = max{tw(A), tw(B)}
- Let G = (V, E) be a clique. Then tw(G) = |V|-1.
- "Removing" a tree-node or tree-edge separates the graph.

Examples

- weighted independent set using a (nice) tree decomposition
- weighted vertex cover using a (nice) tree decomposition

Study Advice

Please read remaining parts of two papers and chapter 10 (about 10 pages are new:)

1. Section 10.3 from Jon Kleinberg and Eva Tardos, *Algorithm Design*, 2006.
2. Gerhard Woeginger, Exact algorithms for NP-hard problems: A survey, *Combinatorial Optimization*, LNCS 3570, pp 187-207, 2003: Section 4 for DP, section 5 for Preprocessing [section 6 for local search is optional]

Homework

- Independent set over a nice tree decomposition
- Weighted max cut