Exact Algorithms for NP-hard problems

Advanced Algorithms: Part 2, Lecture 3

Today

- Discuss Dynamic Programming homework assignment
- Tree decompositions
  - Definitions (tree decomposition, treewidth)
  - Properties
- Dynamic programming over a tree decomposition
  - Maximum Weighted independent set

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?

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,j] recursively?
Hint: it's the completion time of a job… which job? How to compute?

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 $f(j) = \max(T[S-\{j\}], r_j) + p_j$

General idea from last lecture on dynamic programming

A bit like a search tree:

but additionally reusing solutions to same subproblems

- root represents the complete problem
- children are smaller subproblems: alternatives for single decision (mutually exclusive, all need to be investigated)
- expressed as recursive algorithm
- store (and re-use) values of optimal solutions for subproblems
- first analyze space, then runtime (often: space * work for data entry)

1-Slide Summary on Dynamic Programming

Traveling Salesperson

$OPT[\{i\};i] = d(1,i)$ for every i

$OPT[S;i] = \min_{j\in S-\{i\}} \{ OPT[S-\{i\};j] + d(j,i) \}$

$\min_{i\in \{2,\ldots,n\}} \{ OPT[\{2,\ldots,n\};i] + d(i,1) \}$

Scheduling with precedences

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

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

Circular Arc Coloring

Enumerate all k-colorings F_i of the intervals through v_i that are consistent with the colorings F_{i-1} of the intervals through v_{i-1}.

Is F_n consistent with the coloring in F_0?

TU Delft

Tree decomposition and tree width

- Definition of a tree decomposition
- Definition of treewidth
- Properties of a tree decomposition

Tree decomposition

General idea

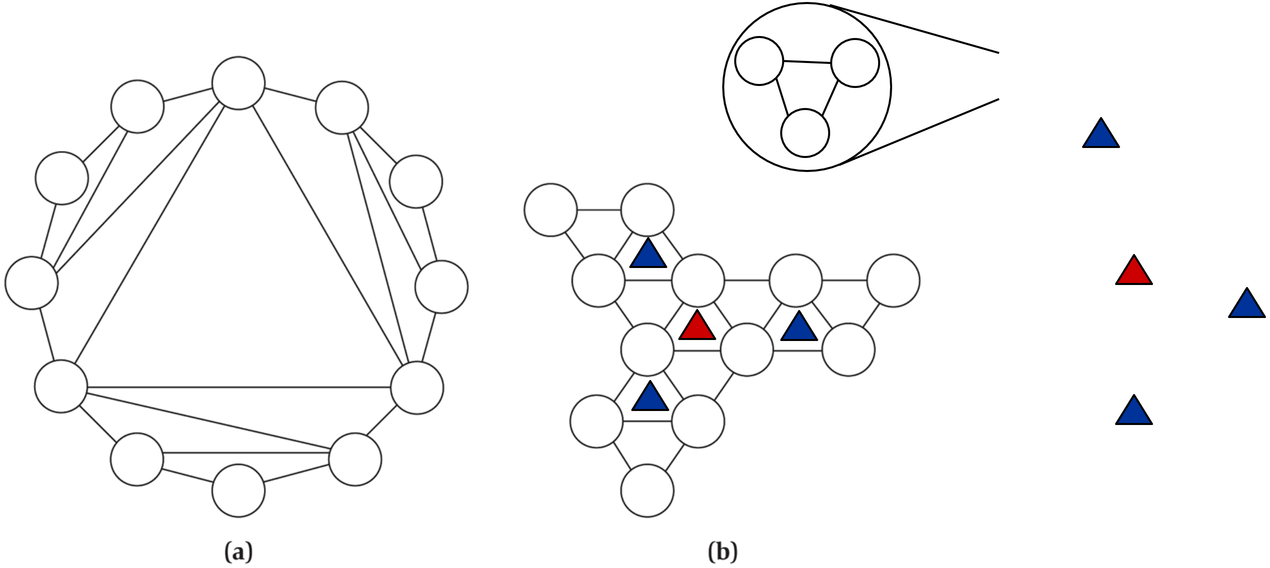
- often algorithms efficient on trees, but problem hard on general graphs (e.g. maximum weighted independent set: $O(n)$ vs $O^*(1.3803^n)$)
- graphs in practice are often "almost" trees, so
  - define measure for "tree"-likeness: tree width
  - run efficient algorithm for trees somehow on these graphs

Applications

- computer/electricity/water networks: tree-like, but redundant links for robustness
- compiler optimization (dependencies are tree-like)
- natural language processing (item relations are tree-like)
- expert systems (inference rules are tree-like)

Tree decompositions play central role in algorithmic graph theory.

Tree decomposition



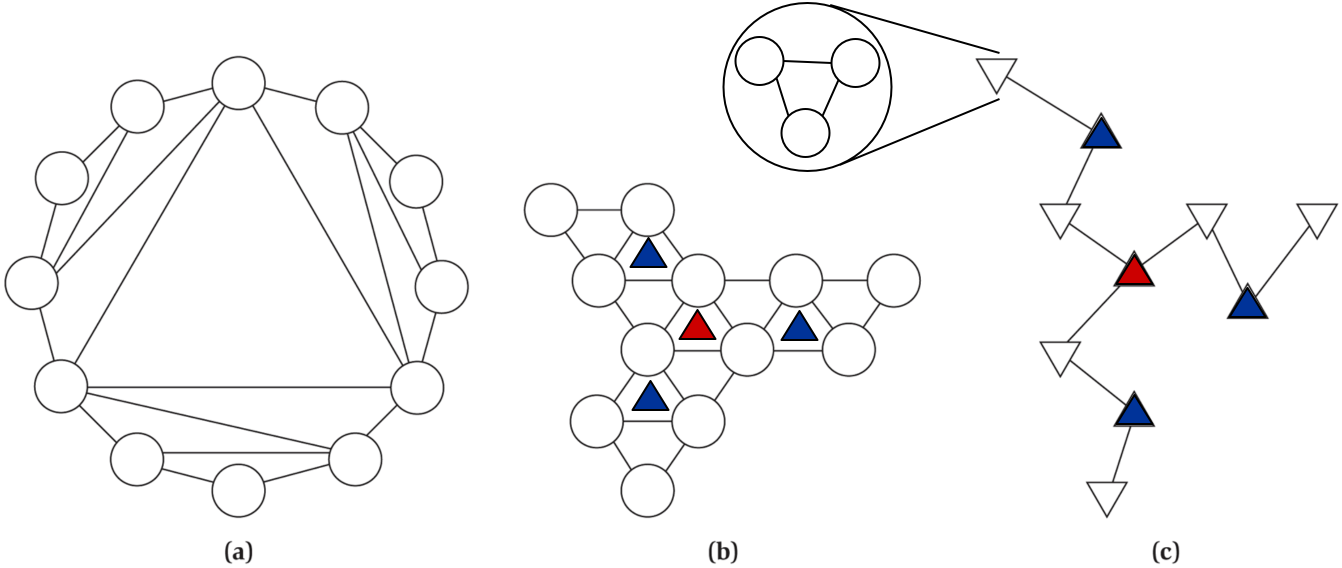
One graph, more representations

Q. Do you “see” the tree through the vertices in (b) ?

A. A tree (T, F) has tree nodes T (triangles), tree edges F, and each tree node represents vertices in original graph

TU Delft

Tree decomposition



One graph, more representations

Q. Do you "see" the tree through the vertices in (b) ?

A. A tree (T, F) has tree nodes T (triangles), tree edges F, and each tree node represents vertices in original graph

TU Delft

Tree decomposition

tree

bags

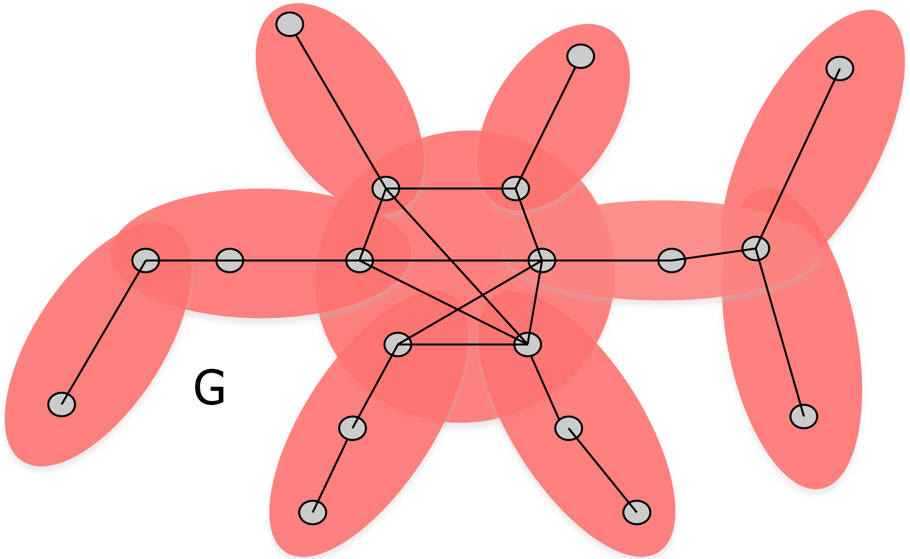
Definition

A tree decomposition of a graph $G = (V, E)$ is a pair $(Tr = (T,F), \{V_t \subseteq V : t \in T\})$ 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)

Q. Give a tree decomposition of $G=(V,E)$.

A. Create 10 tree nodes.
Largest contains 6 vertices.
(Many other answers possible.)



Tree decomposition

tree

bags

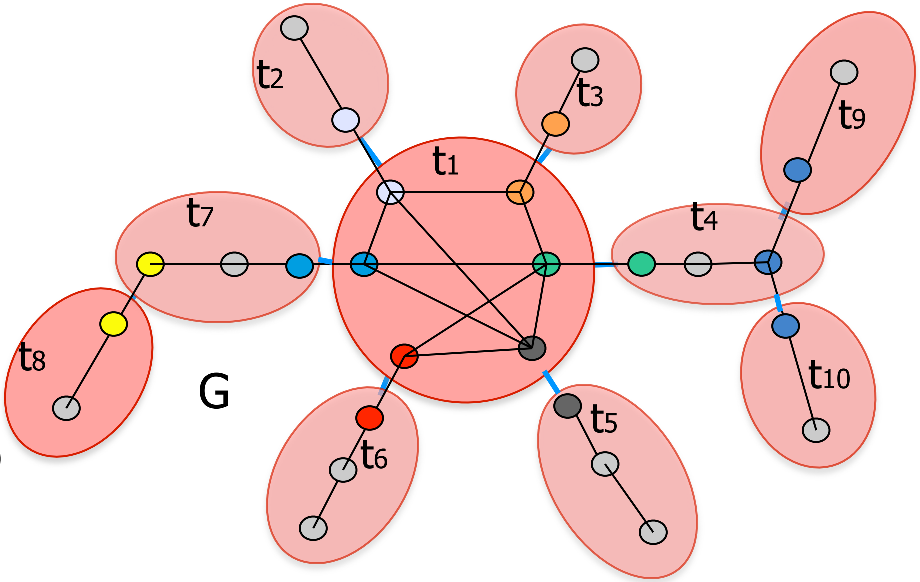
Definition

A tree decomposition of a graph $G = (V, E)$ is a pair $(Tr = (T,F), \{V_t \subseteq V: t \in T\})$ 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)

Q. Give a tree decomposition of $G=(V,E)$.

A. Create 10 tree nodes.
Largest contains 6 vertices.
(Many other answers possible.)



Tree decomposition

tree

bags

Definition

A tree decomposition of a graph $G = (V, E)$ is a pair $(Tr = (T,F), \{V_t \subseteq V : t \in T\})$ 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)

Definition

The width of a tree decomposition $(\{V_t : t \in T\}, Tr = (T,F))$ of a graph G is $\max_{t \in T} \{|V_t| - 1\}$.

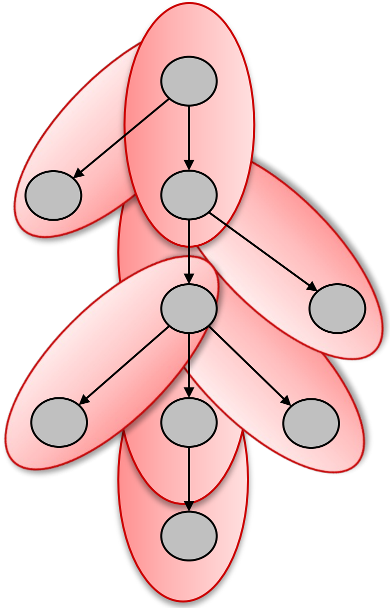
Definition

The treewidth $tw(G)$ of a graph G is the smallest possible width of any tree decomposition of G.

Tree decomposition: examples

Example. A tree decomposition of a tree G=(V,E).

1. create a node t_e in T with bag V_{te} = {u,v} for each edge e={u,v} in E
2. create a connected subtree for tree-nodes for which bags overlap
3. coherence satisfied because each vertex only part of connected tree-nodes (i.e., its parent and children)



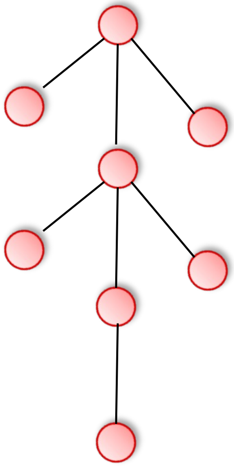
Q. What is the width of this tree decomposition?

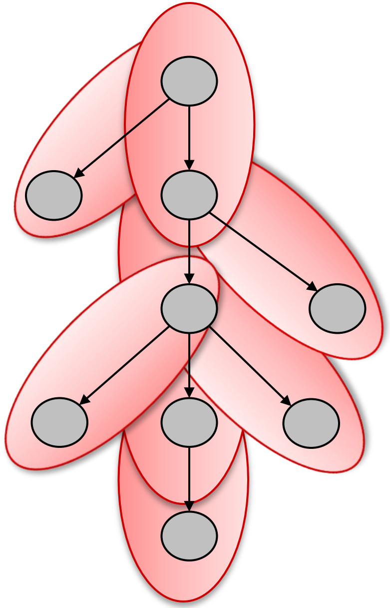
A. 1. This is the treewidth of all trees.

Tree decomposition: examples

(tree nodes, tree edges)

(T,F) then looks like this:



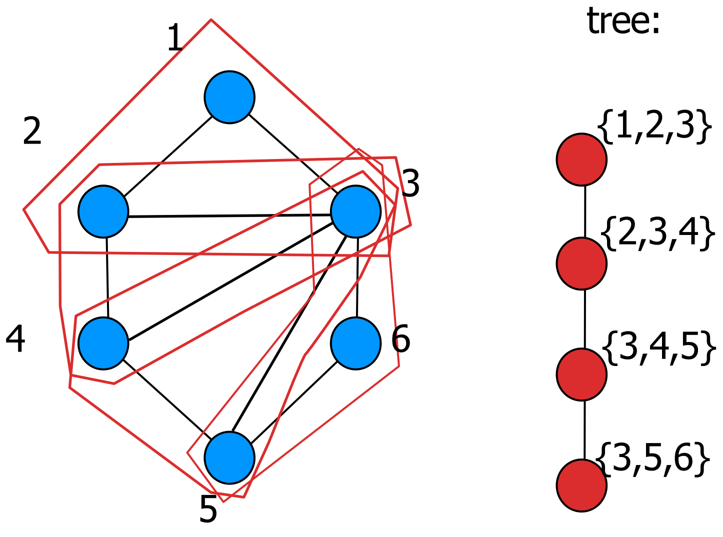


If the intersection of two bags is non-empty, they should be
a) directly connected, or
b) connected via other nodes with bags including this intersection, because of coherence.

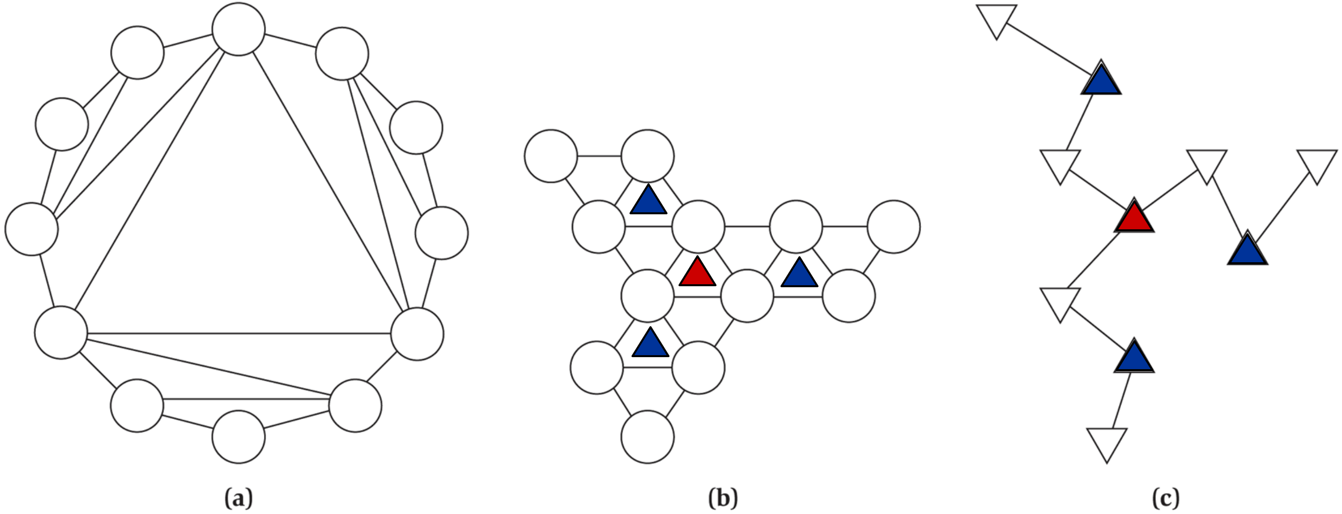
Treewidth: other results

Q: Find the treewidth of a graph G consisting of a single simple cycle of n vertices.

A: The treewidth tw(G) is equal to 3-1=2



Tree decomposition

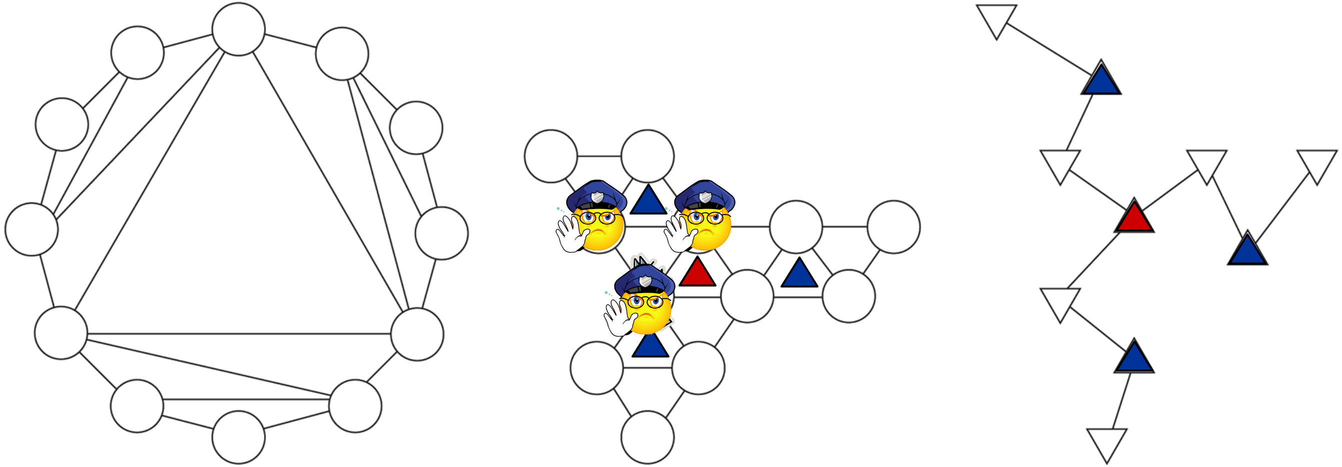


(c) is tree decomposition using triangles in (b) as bags, because:

- vertex coverage
- edge coverage
- coherence: each vertex belongs to a subtree

Q. Is this the tree decomposition with the smallest width? (treewidth)

Tree decomposition



(a) (b) (c)

(because police uses helicopter, robber can quickly go to location police started from)

another way to think about treewidth… (catching a robber)

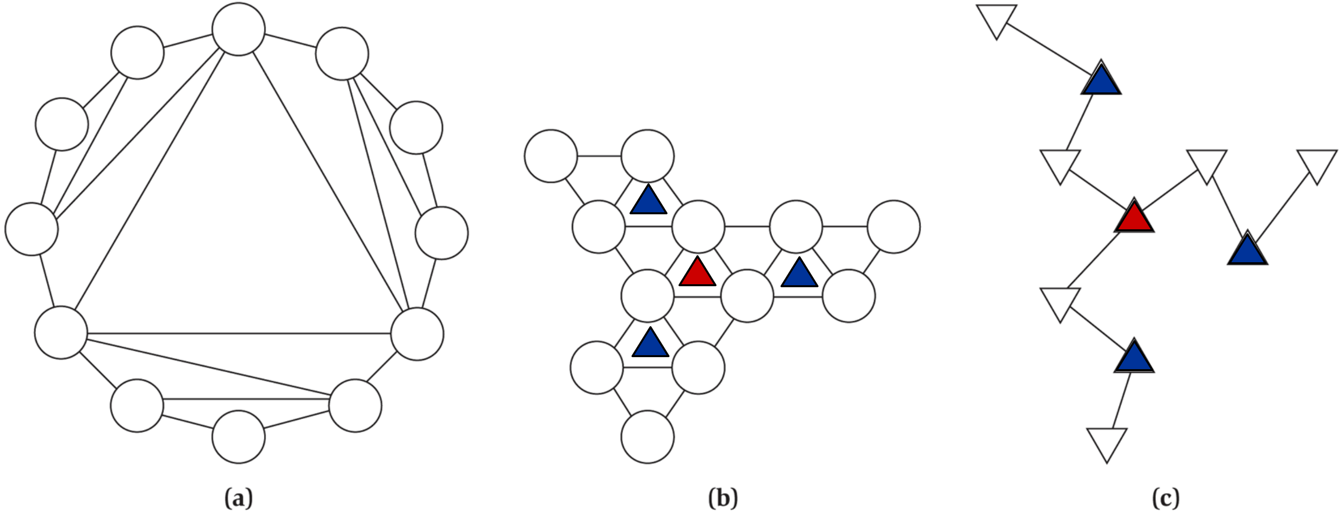
- a robber can see police coming and quickly run to neighbor vertices
- police go to locations with helicopters

how many to lock-in the robber (=set vertices of graph = one tree-node)?

treewidth = min. number of policemen - 1

Seymour & Thomas '93

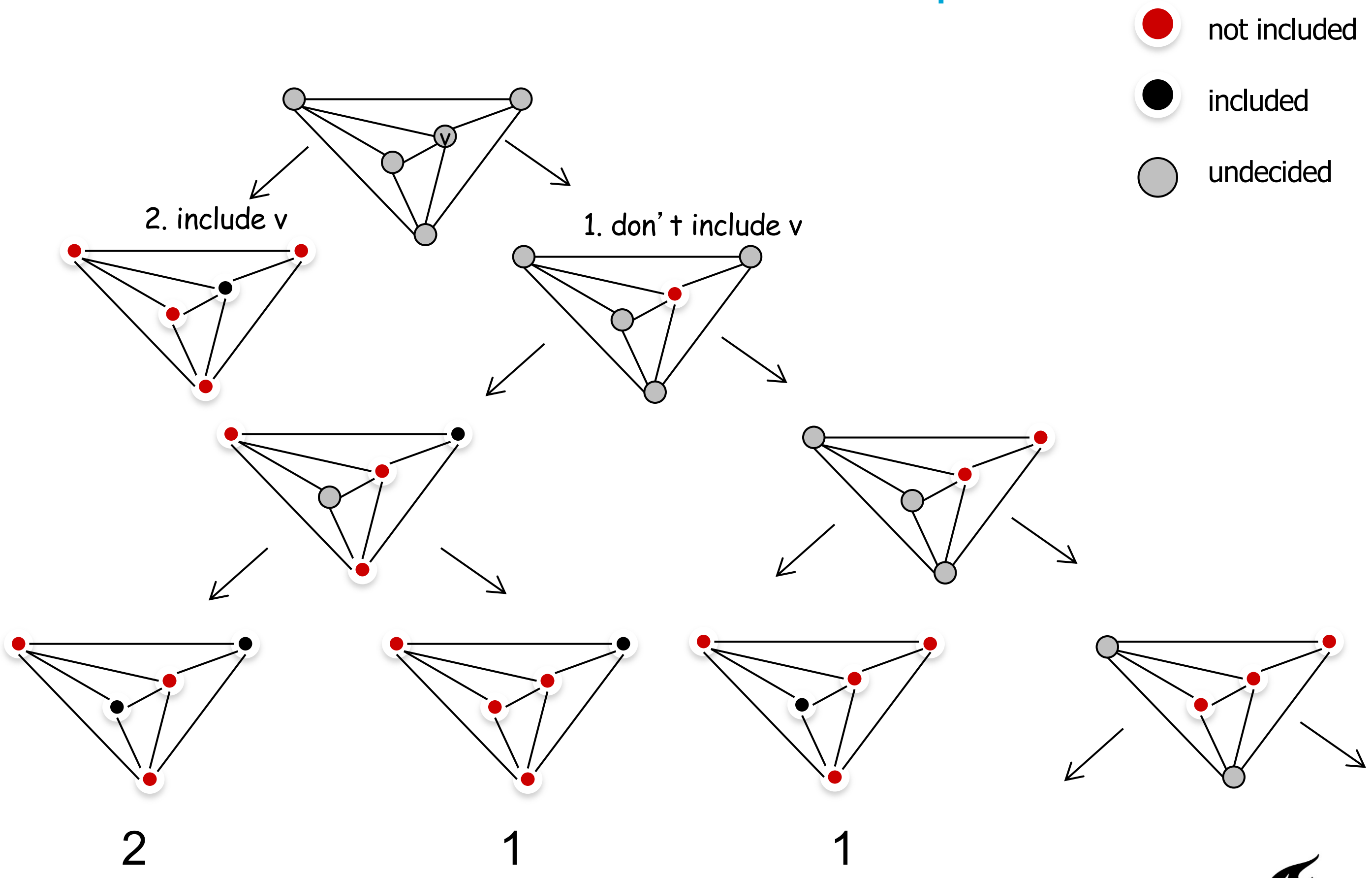
Tree decomposition



Why tree decompositions?

- A fixed decision on the vertices in (e.g.) the red triangle *decouples* the subproblems of the three branches (in c) – so subproblems the same!
- Runtime "only" exponential in possible decisions for the red triangle (no combinations of possibilities from different branches)

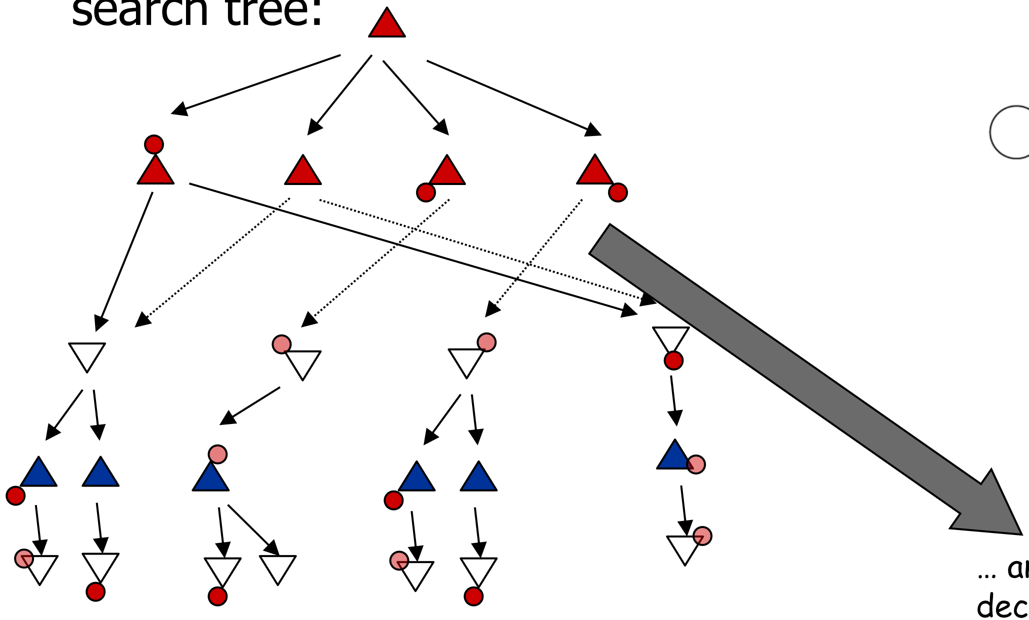
Revisit of the search tree for Independent set



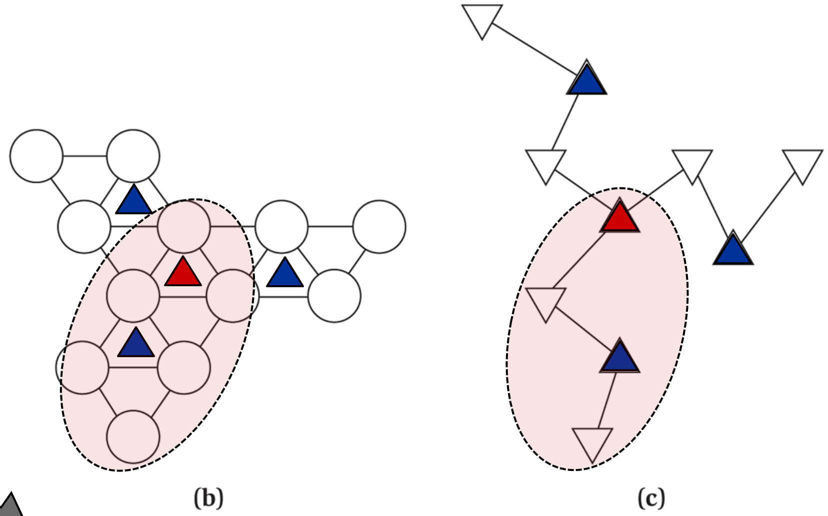
T(n) is $O^*(2^n)$ or even $O^*(1.3803^n)$

Sketch of main idea: Tree decomposition for independent set

Tree decomposition leads to a different search tree:



(subtree for deciding on bottom branch of tree decomposition does not contain any decisions on top-left and right branches)



… and two similar sets of subtrees for deciding top-left and right branches of tree decomposition

Q. How can we do this for the second branch (of red triangle)?

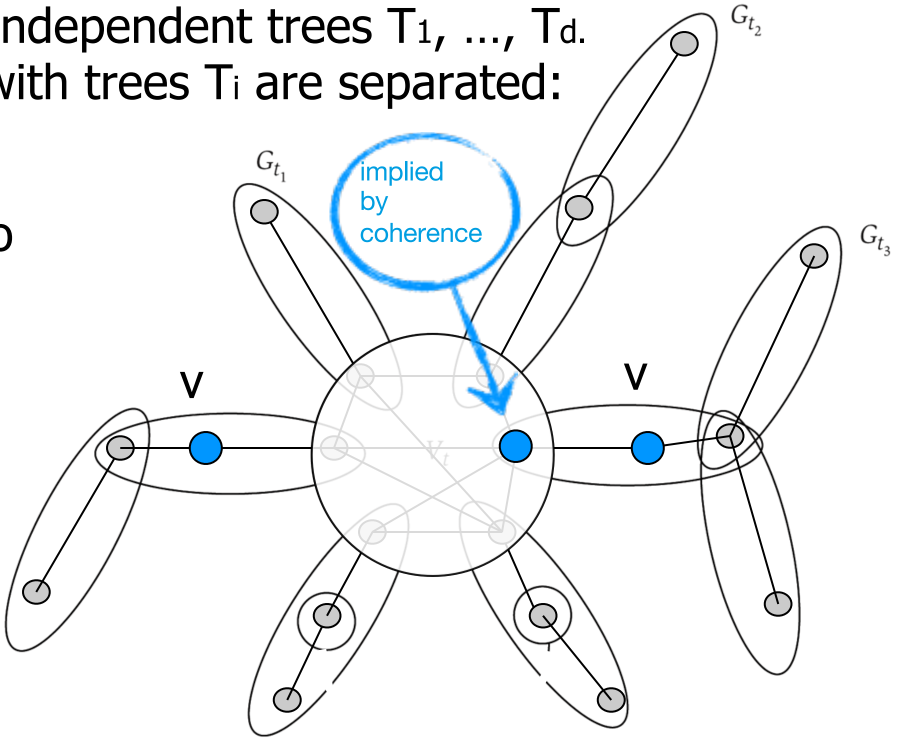
Q. How many subcases for a tree node in worst case? How many such nodes?

T(n) is something like $O^*(|T|\ 2^w)$

Tree decomposition properties: separating tree node

Observation. *(a separating tree node)*
Let (Tr=(T,F), {V_t : t∈T}) be a tree decomposition of G= (V, E) and let t∈T.
Remove tree node t from T.*
Remove V_t from G. The result is a set of independent trees T_1, ..., T_d.
Then resulting subgraphs G_{T_i} associated with trees T_i are separated:

1. They share no vertices
   (from coherence: every vertex v in two or more components should be in V_t and is thus removed)



*) Remove respective vertices Vt and their incident edges from G and the other bags of Tr.

Tree decomposition properties: separating tree node

Observation. *(a separating tree node)*

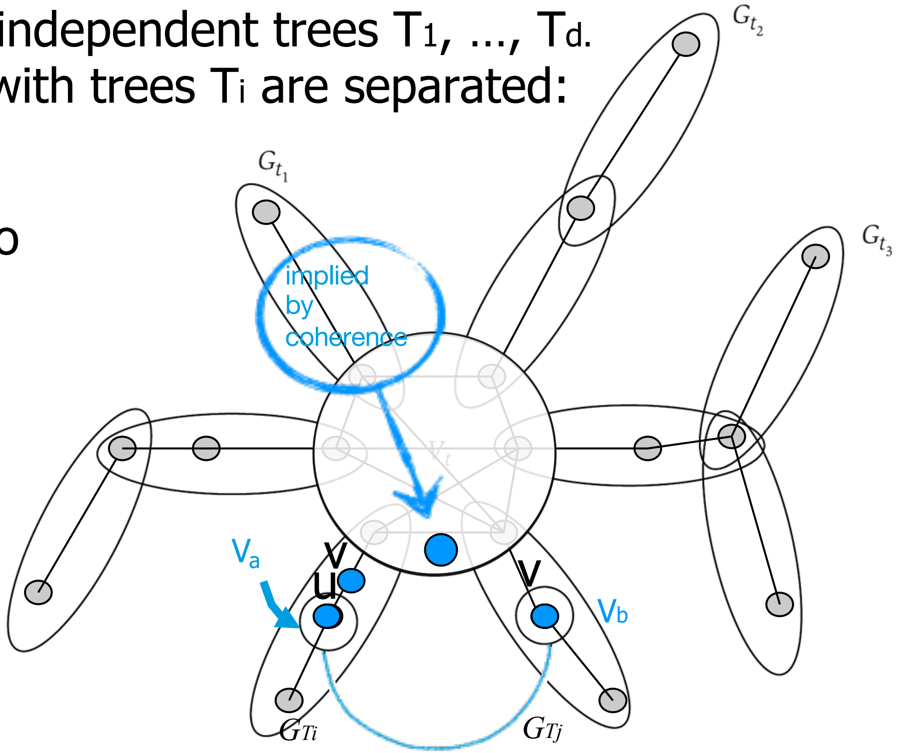
Let $(Tr=(T,F), \{V_t : t \in T\})$ be a tree decomposition of $G= (V, E)$ and let $t \in T$.

Remove tree node t from T.

Remove V_t from G. The result is a set of independent trees $T_1, \ldots, T_d$.

Then resulting subgraphs G_{T_i} associated with trees T_i are separated:

1. They share no vertices
   (from coherence: every vertex v in two or more components should be in V_t and is thus removed)
2. No edges {u,v} between them
   (follows from edge coverage)
   1. {u,v} implies a node $a \in T$ with $u,v \in V_a$ and $v \in V_b$
   2. w.l.o.g. let V_a be in T_i
   3. hence $v \in V_z$ for every z on path a - b in T
   4. so $v \in V_t$; contradiction



Overview of today

- Tree decompositions
  - Definitions (tree decomposition, treewidth)
  - Properties

- Dynamic programming over a tree decomposition
  - Weighted independent set on trees
  - Weighted independent set on tree decompositions

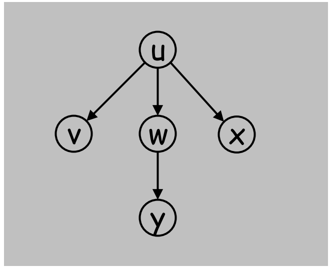
Weighted Independent Set on Trees

neighbors are not allowed

Weighted independent set on trees. Given a tree and vertex weights $w_v > 0$, find an independent set S that maximizes $\Sigma_{v \in S}\, w_v$.

Brute Force. $O(2^n)$

With dynamic programming… efficiently solvable!



(problem instance)

Weighted Independent Set on Trees

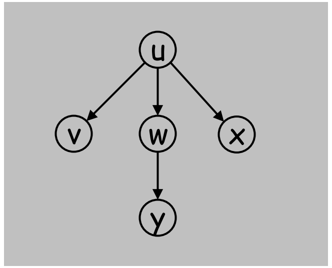
Weighted independent set on trees. Given a tree and vertex weights $w_v > 0$, find an independent set S that maximizes $\Sigma_{v \in S}\, w_v$.

Start with defining a search tree:

Q. Starting at root u, what are the options?

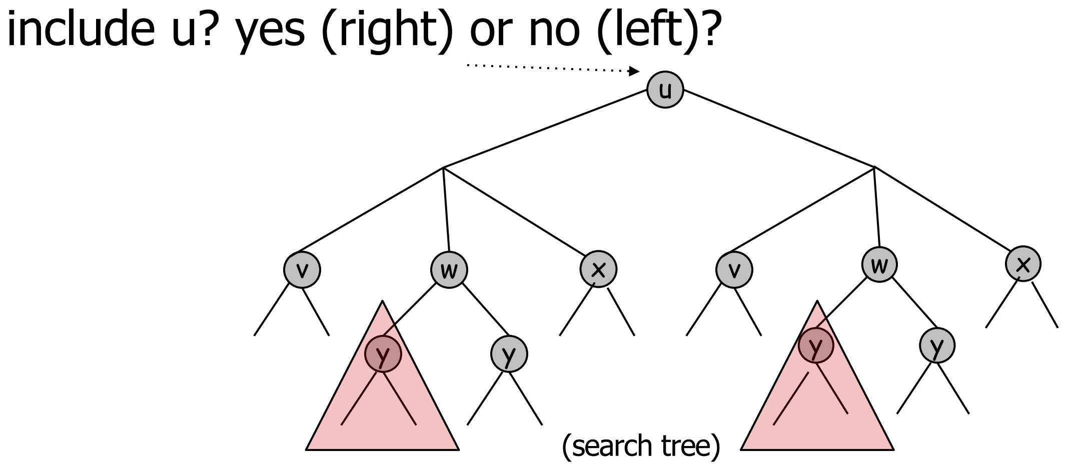
A.

1. include u, or
2. don't include u

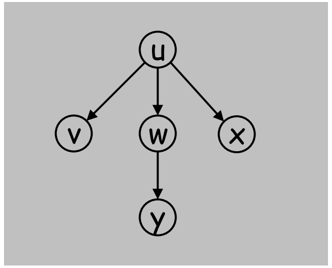


(problem instance)

Independent Set: Brute Force Search Tree



(search tree)



(problem instance)

Observation. Number of nodes grows exponentially with problem size.
Observation. Search tree may contain redundant sub-problems (e.g. y).
Two types of subproblems y: where y may be chosen, or not.

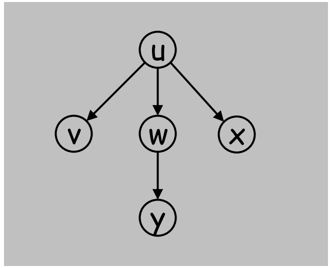
Idea. Dynamic programming:
1. Store and reuse solutions to subproblems.
2. Compute these bottom-up.

(Contrastingly, in search trees we aim for only unique subproblems)

Weighted Independent Set on Trees

Weighted independent set on trees. Given a tree and vertex weights $w_v > 0$, find an independent set S that maximizes $\Sigma_{v \in S}\, w_v$.

Idea. Use dynamic programming to optimize sum of weights for a tree with root u.



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

Q. Starting at root u, what are the options?
A.
1. include u (and thus don't include children of u), or
2. don't include u (and possibly include all children of u).

Q. How to express the value of an optimal solution in these cases?
A.
1. = w_u + sum over optimal solution of children, excluding children
2. = sum over optimal solution of children

Idea. Use different notation for optimal solution with and without u.

Weighted Independent Set on Trees

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

- OPT_{in} (u) = max weight independent set rooted at u, containing u.
- OPT_{out}(u) = max weight independent set rooted at u, not containing u.

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

The two subcases are:

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

Give recursive formulas for OPT_{in} and OPT_{out}.

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

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

Q. For a DP, in what order should we calculate the subproblems?

Independent Set on Trees: DP Algorithm

Claim. The following dynamic programming algorithm efficiently finds a maximum weighted independent set in trees.

```
Weighted-Independent-Set-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) M_out[v]  +  w_u
         M_out[u] = Σ_v∈children(u) max(M_out[v], M_in[v])
      }
   }
   return max(M_in[r], M_out[r])
}
```

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

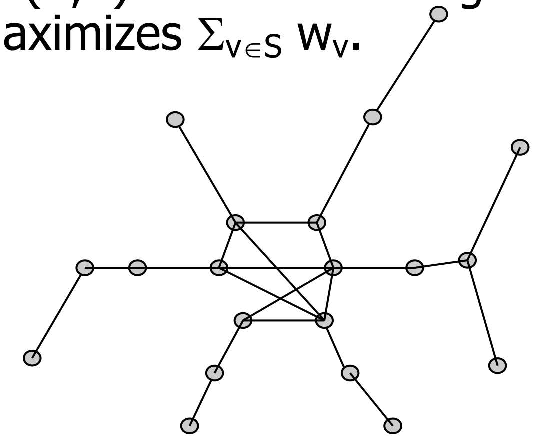
Q. What is the space and runtime of this algorithm?

A. Takes O(n) space and O(n) time since we visit vertices in post-order and examine each edge exactly once. ▪

Dynamic programming over a Tree *Decomposition*

Dynamic programming over a tree decomposition

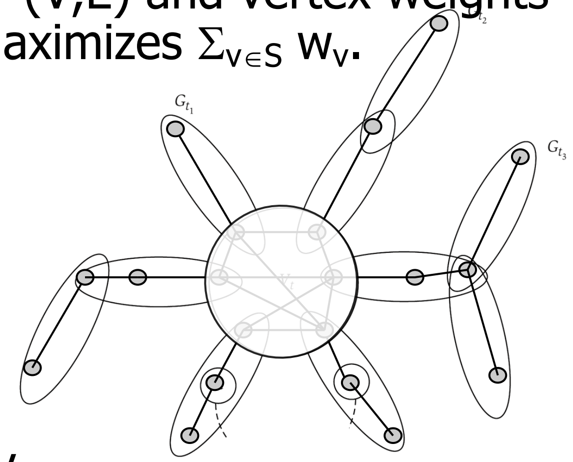
Weighted independent set. Given a graph $G=(V,E)$ and vertex weights $w_v>0$, find an independent set $S \subseteq V$ that maximizes $\Sigma_{v \in S}\, w_v$.



TU Delft

Dynamic programming over a tree decomposition

Weighted independent set. Given a graph $G=(V,E)$ and vertex weights $w_v>0$, find an independent set $S \subseteq V$ that maximizes $\Sigma_{v \in S}\, w_v$.



Idea. Use

- tree decomposition ($Tr=(T,F)$, $\{V_t : t \in T\}$)
- dynamic programming over Tr to optimize sum of weights OPT_t for a tree with root V_t
- brute force over all possible independent sets in every bag $t \in T$.

That's why we'd like the width as small as possible.

Dynamic programming over a tree decomposition

Refining the idea (i)

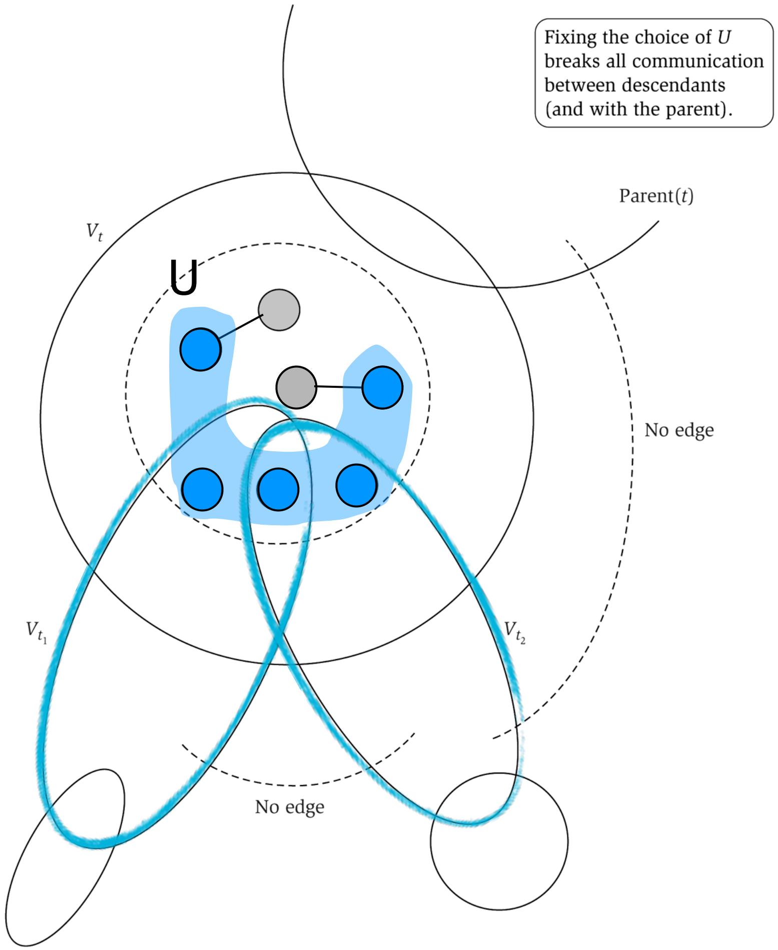
Given a tree decomposition $(Tr=(T,F), \{V_t : t \in T\})$ with root V_t, branch on (sub-cases are):

(all combinations of in/out of $v \in V_t$, so)

all possible independent sets $U \subseteq V_t$ in G (with $w(U) = \Sigma_{u \in U} w_u$)

So let us compute the value $OPT_t(U)$ of each such a subset U.

But choosing such a U has consequences for the choice of independent sets in the children of V_t !



Dynamic programming over a tree decomposition

Refining the idea (ii)

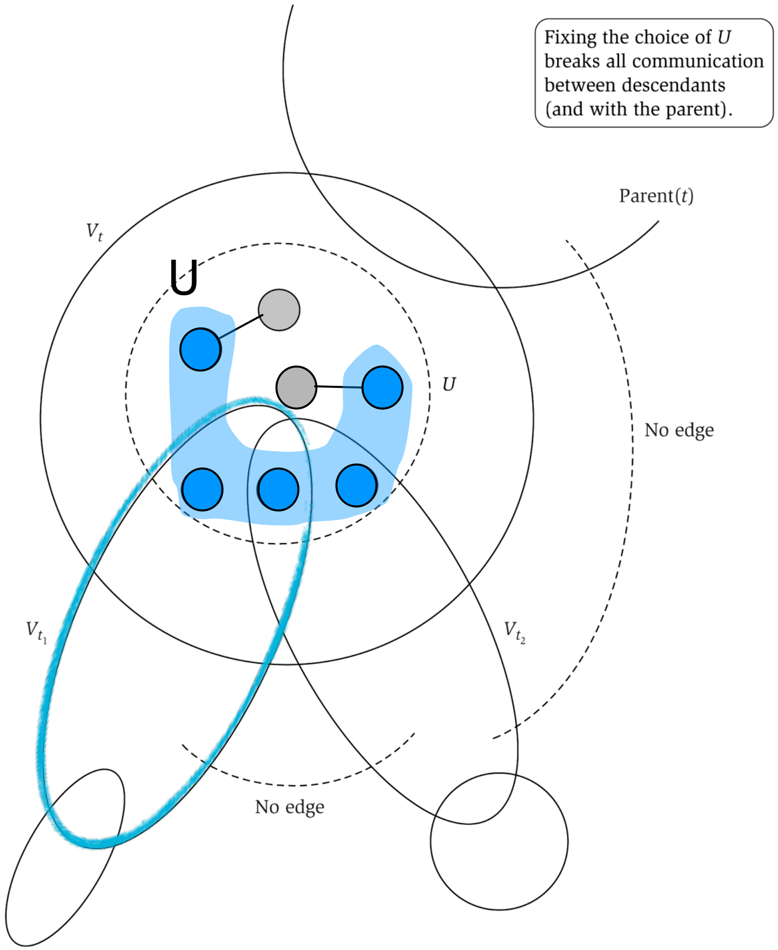
Express maximum value $OPT_t(U)$ of an independent set U recursively, using V_t's children in T (i.e. $V_{t1}, \ldots, V_{td}$):

$OPT_t(U) = w(U) + \Sigma_{i=1,..,d}$ maximum of choices of independent sets for subtrees with root at V_{ti}, consistent with U

Dynamic programming over a tree decomposition

Consistent choices:

independent vertices selected by U in $V_t \cap V_{ti}$ should be the same as independent vertices selected by U_i in $V_t \cap V_{ti}$, so $U_i \cap V_t = U \cap V_{ti}$.



Dynamic programming over a tree decomposition

Refining the idea (ii)

Express maximum value $OPT_t(U)$ of an independent set U recursively, using V_t's children in T (i.e. $V_{t1}, \ldots, V_{td}$):

$OPT_t(U) = w(U) + \Sigma_{i=1,..,d}$ maximum of choices of independent sets for subtrees with root at V_{ti}, consistent with U

Consistent choices:

independent vertices selected by U in $V_t \cap V_{ti}$ should be the same as independent vertices selected by U_i in $V_t \cap V_{ti}$, so $U_i \cap V_t = U \cap V_{ti}$.

Improving the equation

$$OPT_t(U) = w(U) + \sum_{i=1}^{d} \max_{U_i \subseteq V_{t_i}} \left\{ OPT_{t_i}(U_i) - w(U_i \cap U) : \begin{array}{r} U_i \cap V_t = U \cap V_{t_i} \text{ and} \\ U_i \subseteq V_{t_i} \text{ is independent} \end{array} \right\}$$

Q. How to use this recursive equation to implement a dynamic programming solution?

Dynamic programming over a tree decomposition

$$OPT_t(U) = w(U) + \sum_{i=1}^{d} \max_{U_i \subseteq V_{t_i}} \left\{ OPT_{t_i}(U_i) - w(U_i \cap U) : \begin{array}{r} U_i \cap V_t = U \cap V_{t_i} \text{ and} \\ U_i \subseteq V_{t_i} \text{ is independent} \end{array} \right\}$$

Base.
Q. What is $OPT_t(U)$ if t is a leaf in the tree T?
A. Just compute w(U) (for every $U \subseteq V_t$ that is an independent set)

We have $OPT_t(U)$ for every tree-node t and independent subset $U \subseteq V_t$.
Q. What is the size of the maximum independent set of the whole graph?
A. max{ $OPT_r(U)$: $U \subseteq V_r$ is independent }

Q. Given a graph G, a tree decomposition with root V_r, give a dynamic programming algorithm to calculate the optimal value OPT_r. In what order should we calculate the subproblems?
A. Post-order: leaves first.

TU Delft

Dynamic programming over a tree decomposition

```
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: $\mathrm{OPT}_t = f_t$

(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 independent set U: $O(2^{w+1})$ with at most n tree nodes this is thus $O(n2^{w+1})$.

Dynamic programming over a tree decomposition

```
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: $\mathrm{OPT}_t = f_t$

(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 $\mathrm{OPT}_t(U)$ takes $O(2^{w+1}wd)$, where d is #children.
Needs to be done for each independent set U: $O(2^{w+1})$ times.
So $O(4^{w+1}wn)$, because |T| is at most O(n) children in total.

Study Advice

Please read (about 20 pages)

1. Section 10.2 and Section 10.4 (and 10.5) from Jon Kleinberg and Eva Tardos, *Algorithm Design*, 2006.

2. Falk Hueffner, Rold Niedermeier and Sebastian Wernicke, Techniques for Practical Fixed-Parameter Algorithms, *The Computer Journal*, 51(1):7–25, 2008: Section 1 background, Section 5 conclusions

Homework (on BrightSpace)

- Give tree decomposition
- Exercise 10.4 from Kleinberg (Tree decomposition of triangulated cycle graphs)

TU Delft