FEDOR V. FOMIN

Part I. Introduction to treewidth

SCHOOL ON PARAMETERIZED ALGORITHMS AND COMPLEXITY

17-22 August 2014

Będlewo, Poland

Why treewidth?

Very general idea in science: large structures can be understood by breaking them into small pieces

Why treewidth?

Very general idea in science: large structures can be understood by breaking them into small pieces

In Computer Science: divide and conquer; dynamic programming

Why treewidth?

Very general idea in science: large structures can be understood by breaking them into small pieces

In Computer Science: divide and conquer; dynamic programming

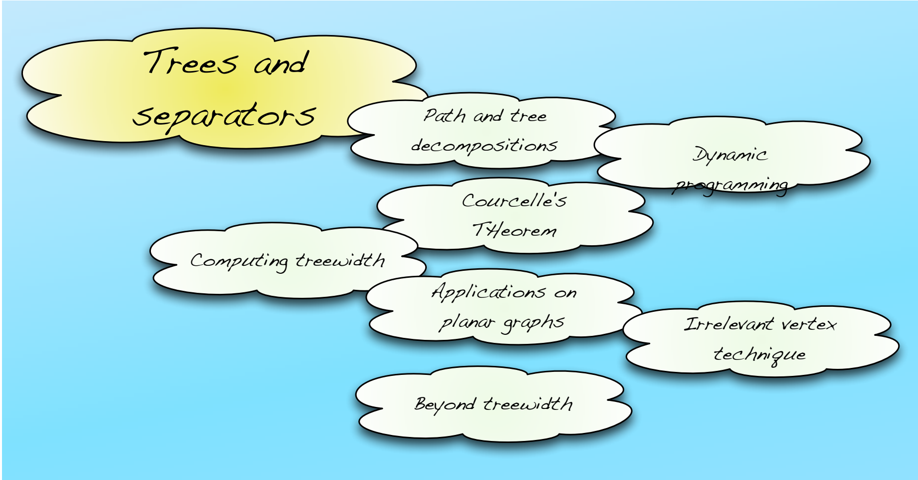
In Graph Algorithms: Exploiting small separators

Why treewidth?

Very convenient to decompose a graph via small separations + Obstacles for decompositions = Powerful tool

Plan

- Trees and separators
- Path and tree decompositions
- Dynamic programming
- Courcelle's THeorem
- Computing treewidth
- Applications on planar graphs
- Irrelevant vertex technique
- Beyond treewidth


Trees and separators
Path and tree decompositions
Dynamic programming
Courcelle's Theorem
Computing treewidth
Applications on planar graphs
Irrelevant vertex technique
Beyond treewidth

The Party Problem

PARTY PROBLEM

Problem:	Invite some colleagues for a party.
Maximize:	The total fun factor of the invited people.
Constraint:	Everyone should be having fun.

The Party Problem

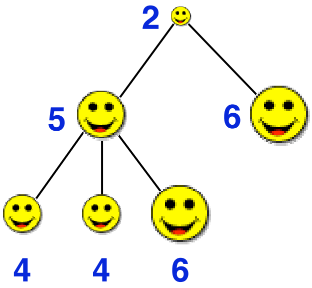
Party Problem

Problem:	Invite some colleagues for a party.
Maximize:	The total fun factor of the invited people.
Constraint:	Everyone should be having fun.
	No fun with your direct boss!

The Party Problem

Party Problem

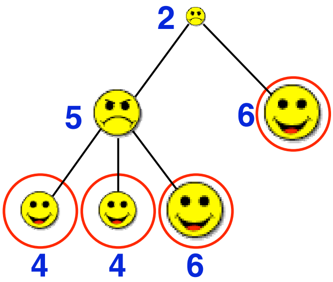
Problem:	Invite some colleagues for a party.
Maximize:	The total fun factor of the invited people.
Constraint:	Everyone should be having fun.
	No fun with your direct boss!



- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

The Party Problem

Party Problem	
Problem:	Invite some colleagues for a party.
Maximize:	The total fun factor of the invited people.
Constraint:	Everyone should be having fun.
	No fun with your direct boss!



- **Input:** A tree with weights on the vertices.
- **Task:** Find an independent set of maximum weight.

Solving the Party Problem

Dynamic programming paradigm: We solve a large number of subproblems that depend on each other. The answer is a single subproblem.

T_v: the subtree rooted at v.

$A[v]$: max. weight of an independent set in T_v

$B[v]$: max. weight of an independent set in T_v that does not contain v

Goal: determine $A[r]$ for the root r.

Solving the Party Problem

Dynamic programming paradigm: We solve a large number of subproblems that depend on each other. The answer is a single subproblem.

T_v: the subtree rooted at v.
$A[v]$: max. weight of an independent set in T_v
$B[v]$: max. weight of an independent set in T_v that does not contain v

Goal: determine $A[r]$ for the root r.

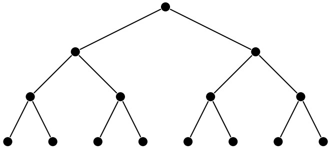
Method:

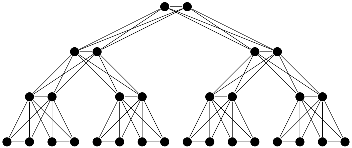
Assume $v_1, \ldots, v_k$ are the children of v. Use the recurrence relations

$$B[v] = \sum_{i=1}^{k} A[v_i]$$
$$A[v] = \max\{B[v]\ ,\ w(v) + \sum_{i=1}^{k} B[v_i]\}$$

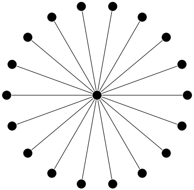
The values $A[v]$ and $B[v]$ can be calculated in a bottom-up order (the leaves are trivial).

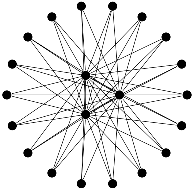
What is a tree-like graph?


tree


treelike

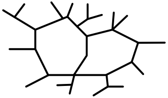
What is a tree-like graph?

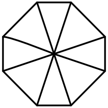

tree

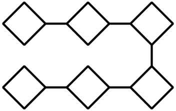

treelike

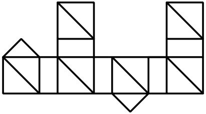
What is a tree-like graph?

1. Number of cycles is bounded.

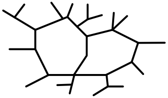

good

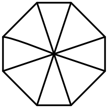

bad

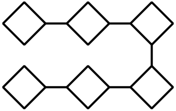

bad

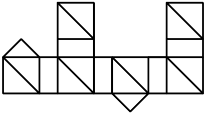

bad

2. Removing a bounded number of vertices makes it acyclic.

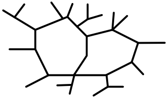

good

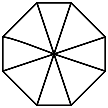

good

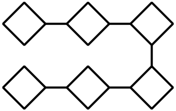

bad

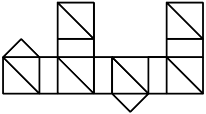

bad

3. Bounded-size parts connected in a tree-like way.

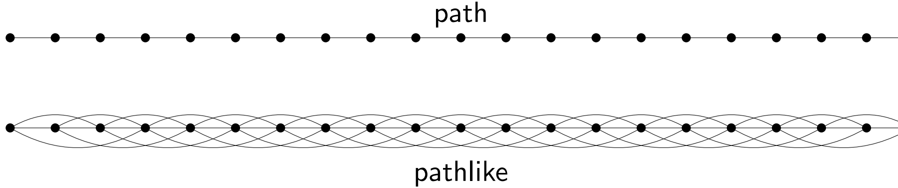

bad


bad


good

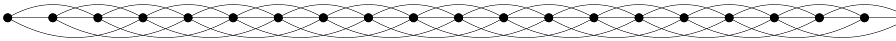

good

Less ambitious question: What is a path-like graph?



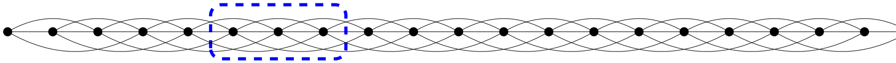
Introduction

Crucial property of pathlike treelike graphs: separators.



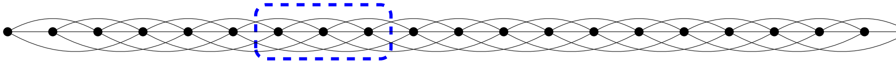
Introduction

Crucial property of pathlike treelike graphs: separators.



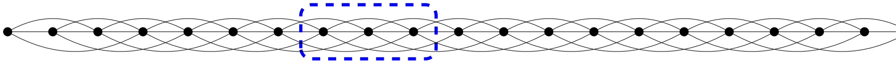
Introduction

Crucial property of pathlike treelike graphs: separators.



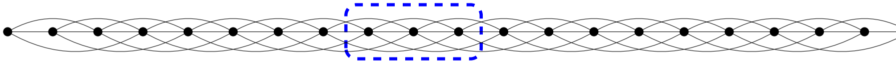
Introduction

Crucial property of pathlike treelike graphs: separators.



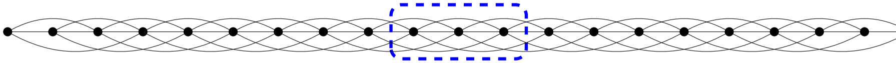
Introduction

Crucial property of pathlike treelike graphs: separators.



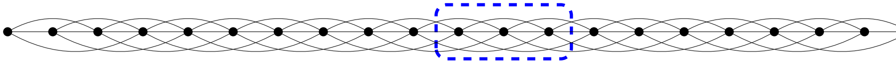
Introduction

Crucial property of pathlike treelike graphs: separators.



Introduction

Crucial property of pathlike treelike graphs: separators.



Useful point of view: generating sequence

For an integer k, we generate graph G (if we can) using the following operations:

Useful point of view: generating sequence

For an integer k, we generate graph G (if we can) using the following operations:

1. $init$: $V := \emptyset, E := \emptyset, X := \emptyset$

Useful point of view: generating sequence

For an integer k, we generate graph G (if we can) using the following operations:

1. $init$: $V := \emptyset, E := \emptyset, X := \emptyset$
2. $introduce - vertex(v)$ for $v \notin V$:
   $V := V \cup \{v\}$
   $X := X \cup \{v\}$

Useful point of view: generating sequence

For an integer k, we generate graph G (if we can) using the following operations:

1. $init$: $V := \emptyset, E := \emptyset, X := \emptyset$
2. $introduce - vertex(v)$ for $v \notin V$:
   $V := V \cup \{v\}$
   $X := X \cup \{v\}$
3. $forget(v)$ for $v \in X$:
   $X := X \setminus \{v\}$

Useful point of view: generating sequence

For an integer k, we generate graph G (if we can) using the following operations:

1. $init$: $V := \emptyset, E := \emptyset, X := \emptyset$
2. $introduce - vertex(v)$ for $v \notin V$:
   $V := V \cup \{v\}$
   $X := X \cup \{v\}$
3. $forget(v)$ for $v \in X$:
   $X := X \setminus \{v\}$
4. $introduce - edge(uv)$ for $u, v \in X$:
   $E := E \cup \{uv\}$

Useful point of view: generating sequence

For an integer k, we generate graph G (if we can) using the following operations:

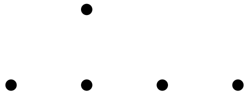
1. $init$: $V := \emptyset, E := \emptyset, X := \emptyset$
2. $introduce - vertex(v)$ for $v \notin V$:
   $V := V \cup \{v\}$
   $X := X \cup \{v\}$
3. $forget(v)$ for $v \in X$:
   $X := X \setminus \{v\}$
4. $introduce - edge(uv)$ for $u, v \in X$:
   $E := E \cup \{uv\}$

A sequence of operations must always satisfy $|X| \leq k$.

Generating sequence

Example:

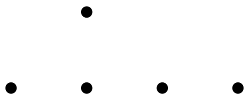
init



Generating sequence

Example:

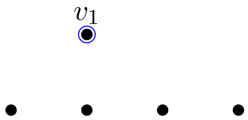
introduce-vertex(v_1)



Generating sequence

Example:

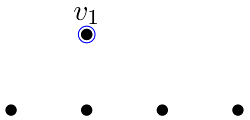
introduce-vertex(v_1)



Generating sequence

Example:

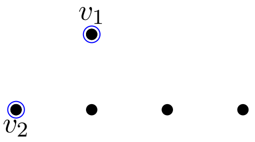
introduce-vertex(v_2)



Generating sequence

Example:

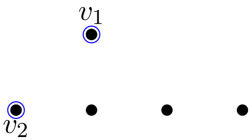
introduce-vertex(v_2)



Generating sequence

Example:

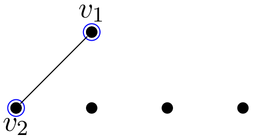
introduce-edge$(v_1 v_2)$



Generating sequence

Example:

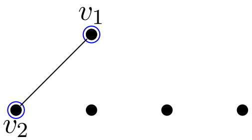
introduce-edge($v_1 v_2$)



Generating sequence

Example:

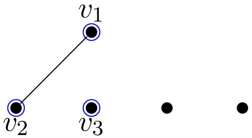
introduce-vertex(v_3)



Generating sequence

Example:

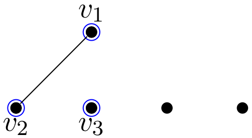
introduce-vertex(v_3)



Generating sequence

Example:

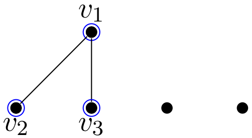
introduce-edge$(v_1 v_3)$



Generating sequence

Example:

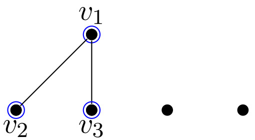
introduce-edge$(v_1 v_3)$



Generating sequence

Example:

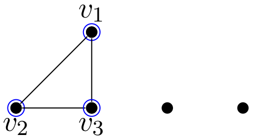
introduce-edge$(v_2 v_3)$



Generating sequence

Example:

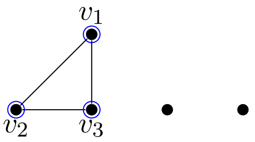
introduce-edge$(v_2 v_3)$



Generating sequence

Example:

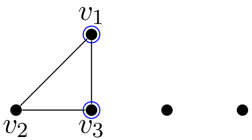
forget(v_2)



Generating sequence

Example:

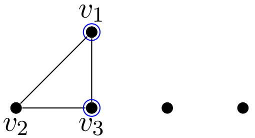
forget(v_2)



Generating sequence

Example:

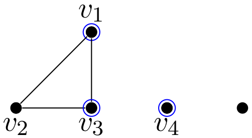
introduce-vertex(v_4)



Generating sequence

Example:

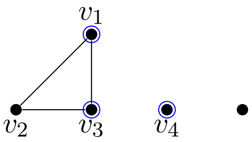
introduce-vertex(v_4)



Generating sequence

Example:

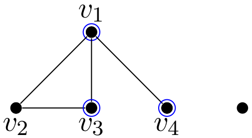
introduce-edge($v_1 v_4$)



Generating sequence

Example:

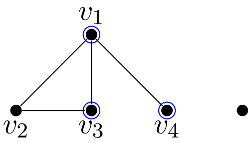
introduce-edge($v_1 v_4$)



Generating sequence

Example:

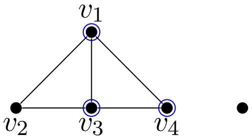
introduce-edge(v_3v_4)



Generating sequence

Example:

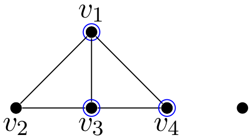
introduce-edge(v_3v_4)



Generating sequence

Example:

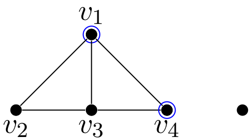
forget(v_3)



Generating sequence

Example:

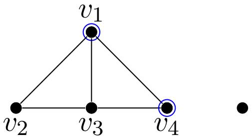
forget(v_3)



Generating sequence

Example:

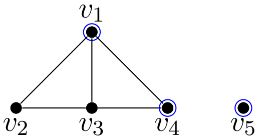
introduce-vertex(v_5)



Generating sequence

Example:

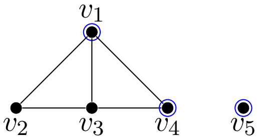
introduce-vertex(v_5)



Generating sequence

Example:

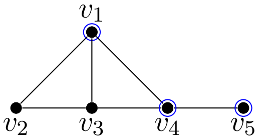
introduce-edge$(v_4 v_5)$



Generating sequence

Example:

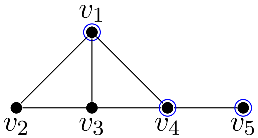
introduce-edge($v_4 v_5$)



Generating sequence

Example:

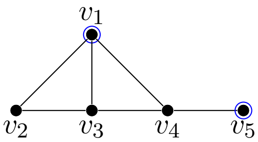
forget(v_4)



Generating sequence

Example:

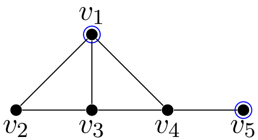
forget(v_4)



Generating sequence

Example:

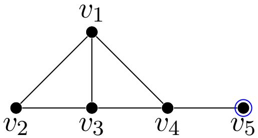
forget(v_1)



Generating sequence

Example:

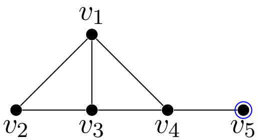
forget(v_1)



Generating sequence

Example:

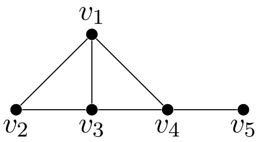
forget(v_5)



Generating sequence

Example:

forget(v_5)



Pathwidth definition (first attempt)

- Since a path can be generated with k equal to

Pathwidth definition (first attempt)

- Since a path can be generated with k equal to 2
- Call the pathwidth of a graph G the minimum $k+1$ such that G can be generated

Running example

Independent Set
Input: *A graph G and an integer k.*
Question: *Is there a subset S of $V(G)$ of size k such that there are no edges between vertices in S?*

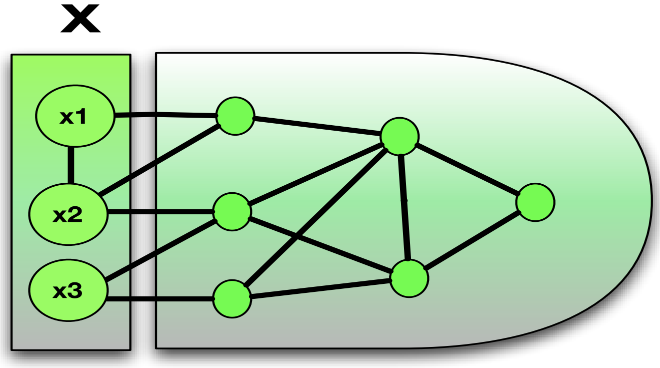
Or find the size of a maximum independent set of G.

Idea

- Follow a generating sequence the graph was constructed
- Exploit the fact that the set of special vertices X is small to compute MIS.

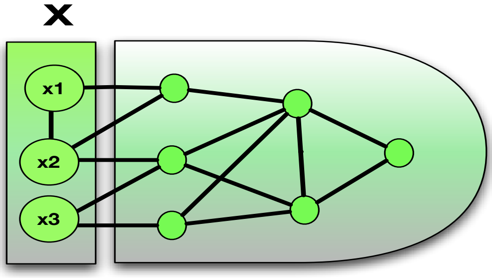
t-boundaried graphs

A k-boundaried graph is a graph with n vertices and at most k special vertices $X \subseteq \{x_1, \ldots, x_k\}$. X is called the boundary of G. Special vertices are $\partial(V_j)$.



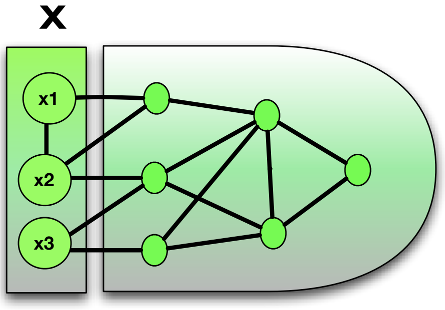
Dynamic table: Generalization of Party Argument

For every subset S of the boundary X, $T[S]$ is the size of the largest independent set I such that $I \cap X = S$, or $-\infty$ if no such



Dynamic table

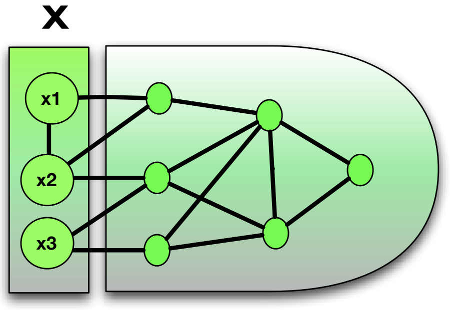
The size of the largest independent set I such that $I \cap X = S$, or $-\infty$ if no such set exists.



$T[\emptyset]$	
$T[x_1]$	
$T[x_2]$	
$T[x_3]$	
$T[x_1, x_2]$	
$T[x_1, x_3]$	
$T[x_2, x_3]$	
$T[x_1, x_2, x_3]$	

Dynamic table

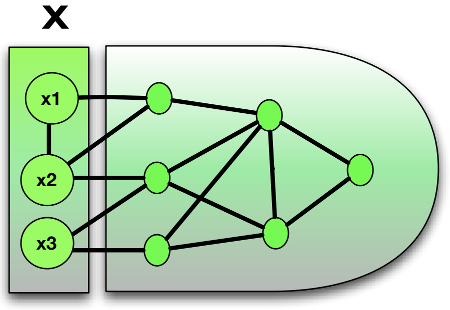
The size of the largest independent set I such that $I \cap X = S$, or $-\infty$ if no such set exists.



$T[\emptyset]$	4
$T[x_1]$	
$T[x_2]$	
$T[x_3]$	
$T[x_1, x_2]$	
$T[x_1, x_3]$	
$T[x_2, x_3]$	
$T[x_1, x_2, x_3]$	

Dynamic table

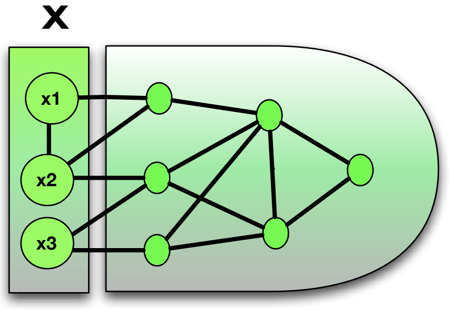
The size of the largest independent set I such that $I \cap X = S$, or $-\infty$ if no such set exists.



$T[\emptyset]$	4
$T[x_1]$	
$T[x_2]$	
$T[x_3]$	
$T[x_1, x_2]$	
$T[x_1, x_3]$	
$T[x_2, x_3]$	
$T[x_1, x_2, x_3]$	$-\infty$

Dynamic table

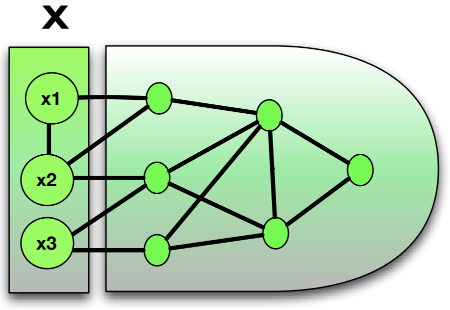
The size of the largest independent set I such that $I \cap X = S$, or $-\infty$ if no such set exists.



$T[\emptyset]$	4
$T[x_1]$	
$T[x_2]$	
$T[x_3]$	
$T[x_1, x_2]$	
$T[x_1, x_3]$	
$T[x_2, x_3]$	3
$T[x_1, x_2, x_3]$	$-\infty$

Dynamic table

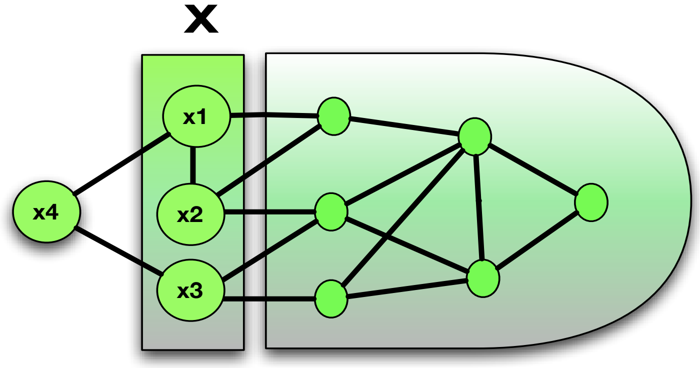
The size of the largest independent set I such that $I \cap X = S$, or $-\infty$ if no such set exists.



$T[\emptyset]$	4
$T[x_1]$	4
$T[x_2]$	3
$T[x_3]$	3
$T[x_1, x_2]$	$-\infty$
$T[x_1, x_3]$	3
$T[x_2, x_3]$	3
$T[x_1, x_2, x_3]$	$-\infty$

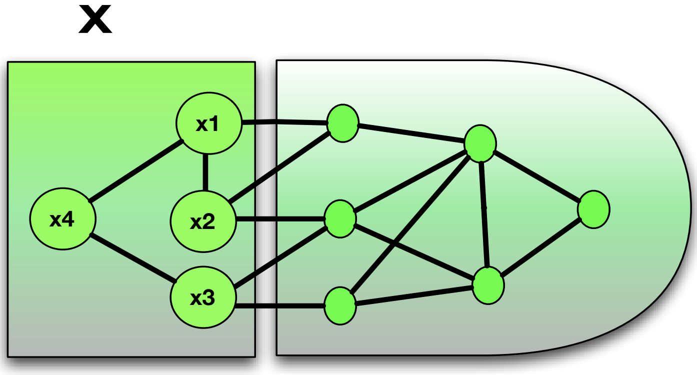
Introduce

Add a vertex $x_i \notin X$ to X. The vertex x_i can have arbitrary neighbours in X but no other neighbours.



Introduce

Add a vertex $x_i \notin X$ to X. The vertex x_i can have arbitrary neighbours in X but no other neighbours.



Introduce: Updating table T

Suppose x_i (here x_4) was introduced into X, with closed neighbourhood $N[x_i]$. We update the table T.

Introduce: Updating table T

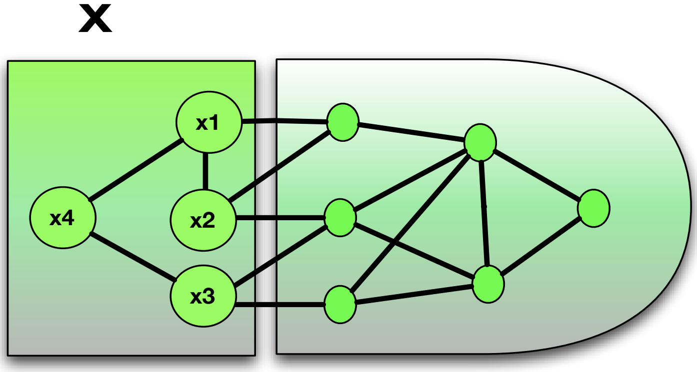
Suppose x_i (here x_4) was introduced into X, with closed neighbourhood $N[x_i]$. We update the table T.

$$T[S] = \begin{cases} T[S] & \text{if } x_i \notin S, \\ -\infty & \text{if } x_i \in S \text{ and } S \cap N(x_i) \neq \emptyset, \\ 1 + T[S \setminus x_i] & \text{if } x_i \in S \text{ and } S \cap N(x_i) = \emptyset. \end{cases}$$

Update time: $2^k \cdot n^{\mathcal{O}(1)}$ [There are tricks to turn it into $2^k \cdot k^{\mathcal{O}(1)}$]

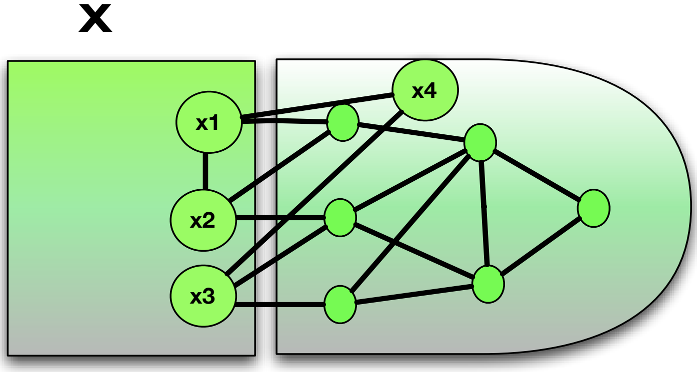
Forget operation

Pick a vertex $x_i \in X$ and forget that it is special (it loses the name x_i and becomes "nameless").



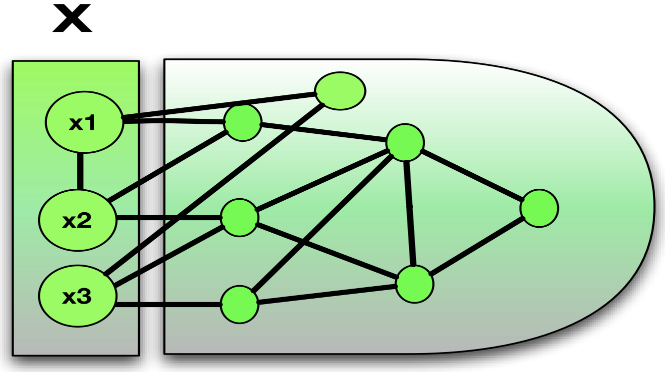
Forget operation

Pick a vertex $x_i \in X$ and forget that it is special (it loses the name x_i and becomes "nameless").



Forget operation

Pick a vertex $x_i \in X$ and forget that it is special (it loses the name x_i and becomes "nameless").



Forget: Updating table T

Forgetting x_i (here x_4).

$$T[S] = \max \left\{ T[S], T[S \cup x_i] \right\}$$

Update time: $2^k k^{\mathcal{O}(1)}$

Two questions:

Two important questions are not answered so far

- How to find a good generating sequence?

Two questions:

Two important questions are not answered so far

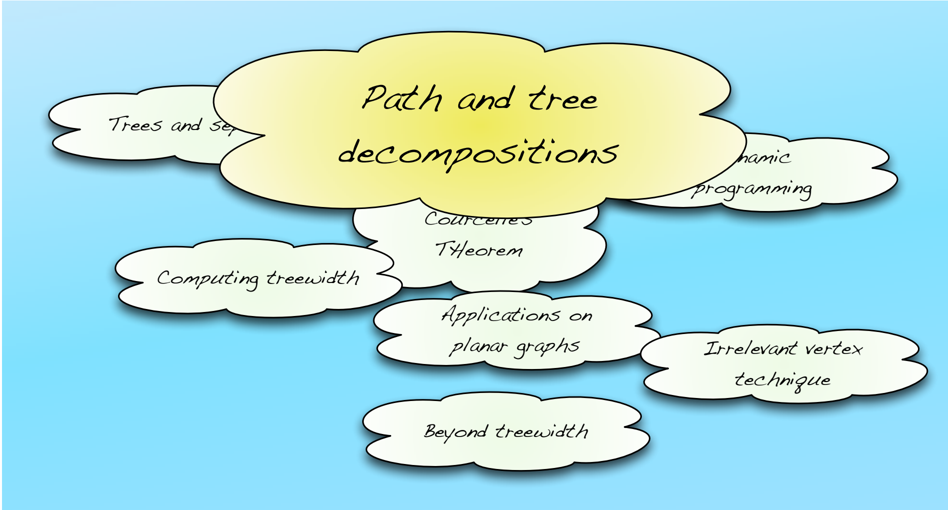
- How to find a good generating sequence?
- While the pathwidth of a tree can be arbitrarily large, the dynamic programs we used on trees and on graphs with small pathwidth are quite similar. Is it possible to combine both approaches?

Two questions:

Two important questions are not answered so far

- How to find a good generating sequence?
- While the pathwidth of a tree can be arbitrarily large, the dynamic programs we used on trees and on graphs with small pathwidth are quite similar. Is it possible to combine both approaches?

In what follow we provide answers to both questions. The answer to the questions will be given by making use of *tree decompositions* and treewidth.


Path and tree decompositions
Trees and sep
namic programming
Courcelle's Theorem
Computing treewidth
Applications on planar graphs
Irrelevant vertex technique
Beyond treewidth

Pathwidth (canonical definition)

A *path decomposition* of graph G is a sequence of *bags* $X_i \subseteq V(G)$, $i \in \{1, \ldots, , r\}$,

$$(X_1, X_2, \ldots, X_r)$$

such that

Pathwidth (canonical definition)

A *path decomposition* of graph G is a sequence of *bags* $X_i \subseteq V(G)$, $i \in \{1, \ldots, , r\}$,

$$(X_1, X_2, \ldots, X_r)$$

such that

(P1) $\bigcup_{1 \leq i \leq r} X_i = V(G)$.

(P2) For every $vw \in E(G)$, there exists $i \in \{1, \ldots, , r\}$ such that bag X_i contains both v and w.

(P3) For every $v \in V(G)$, let i be the minimum and j be the maximum indices of the bags containing v. Then for every k, $i \leq k \leq j$, we have $v \in X_k$. In other words, the indices of the bags containing v form an interval.

The *width* of a path decomposition $(X_1, X_2, \ldots, X_r)$ is $\max_{1 \leq i \leq r} |X_i| - 1$. The *pathwidth* of a graph G is the minimum width of a path decomposition of G.

Example

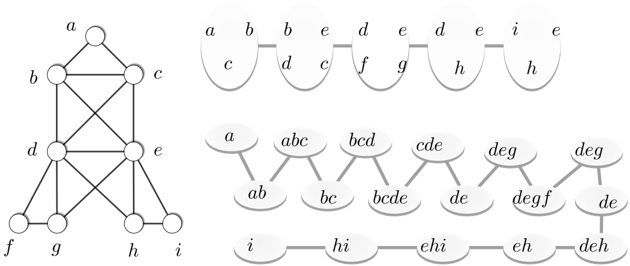


Figure : A graph and its path-decompositions.

Nice Decompositions

It is more convenient to work with nice decompositions.

A path decomposition $(X_1, X_2, \ldots, X_r)$ of a graph G is *nice* if

- $|X_1| = |X_r| = 1$, and
- for every $i \in \{1, 2, \ldots, r-1\}$ there is a vertex v of G such that either $X_{i+1} = X_i \cup \{v\}$, or $X_{i+1} = X_i \setminus \{v\}$.

Nice Decompositions

It is more convenient to work with nice decompositions.
A path decomposition $(X_1, X_2, \ldots, X_r)$ of a graph G is *nice* if

- $|X_1| = |X_r| = 1$, and
- for every $i \in \{1, 2, \ldots, r-1\}$ there is a vertex v of G such that either $X_{i+1} = X_i \cup \{v\}$, or $X_{i+1} = X_i \setminus \{v\}$.

Thus bags of a nice path decomposition are of the two types. Bags of the first type are of the form $X_{i+1} = X_i \cup \{v\}$ and are *introduce nodes*. Bags of the form $X_{i+1} = X_i \setminus \{v\}$ are *forget nodes*.

An Example

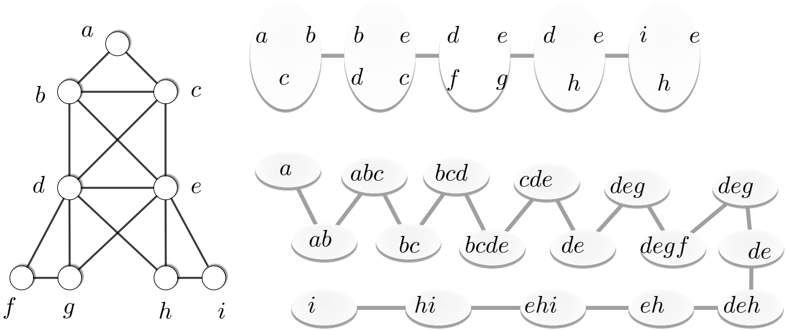


Figure : A graph, its path and nice path decompositions.

An Example

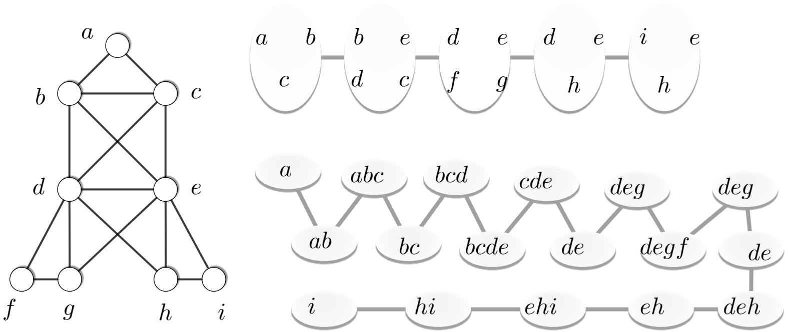


Figure : A graph, its path and nice path decompositions.

Exercise: Construct an algorithm that for a given path decomposition of width k constructs a nice path decomposition of width k in time $\mathcal{O}(k^2 n)$.

Equivalence of definitions

What about separators?

Lemma

Let $(X_1, X_2, \ldots, X_r)$ *be a path decomposition. Then for every* $j \in \{1, \ldots, r-1\}$*,* $\partial(X_1 \cup X_2 \cdots \cup X_j) \subseteq X_j \cap X_{j+1}$*. In other words,* $X_j \cap X_{j+1}$ *separates* $X_1 \cup X_2 \cdots \cup X_j$ *from the other vertices of* G*.*

Proof.

□

DP on graphs of small pathwidth

- The pathwidth($\mathbf{pw}(G)$) of G is the minimum boundary size needed to construct G from the empty graph using introduce and forget operations... -1
- Have seen: Maximum Independent Set can be solved in $2^k k^{\mathcal{O}(1)} n$ time if a path decomposition of width k is given as input.

Tractable problems on graphs of pathwidth p

Independent Set	$O(2^p pn)$
Dominating Set	$O(3^p pn)$
q-Coloring	$O(q^p pn)$
Max Cut	$O(2^p pn)$
Odd Cycle Transversal	$O(3^p pn)$
Hamiltonian Cycle	$O(p^p pn)$
Partition into Triangles	$O(2^p pn)$

Tightness

We will see later that up to SETH these bounds are tight

Independent Set	$\mathcal{O}(2^k kn)$
Dominating Set	$\mathcal{O}(3^k kn)$
q-Coloring	$\mathcal{O}(q^k kn)$
Max Cut	$\mathcal{O}(2^k kn)$
Odd Cycle Transversal	$\mathcal{O}(3^k kn)$
Partition into Triangles	$\mathcal{O}(2^k kn)$

Pathwidth

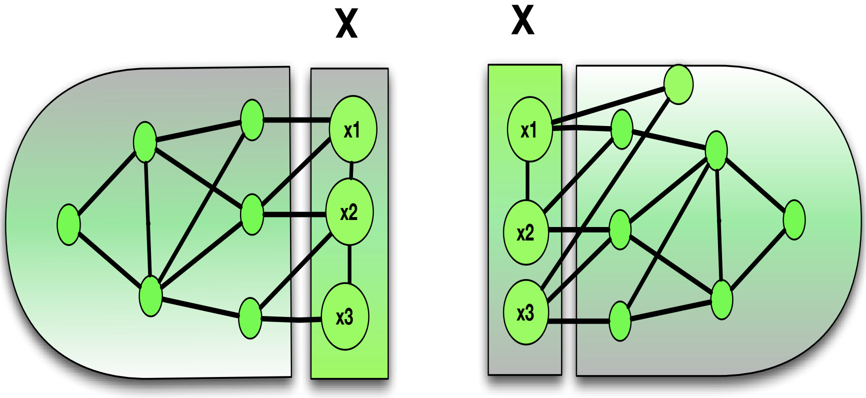
- Introduced in the 80's as a part of Robertson and Seymour's Graph Minors project.
- (Bodlaender and Kloks 96) Graphs of pathwidth k can be recognized in $f(k)n$ time — FPT algorithm.

Another Operation: Join Operation

Given two t-boundaried graphs G_1 and G_2, the join operation glues them together at the boundaries.

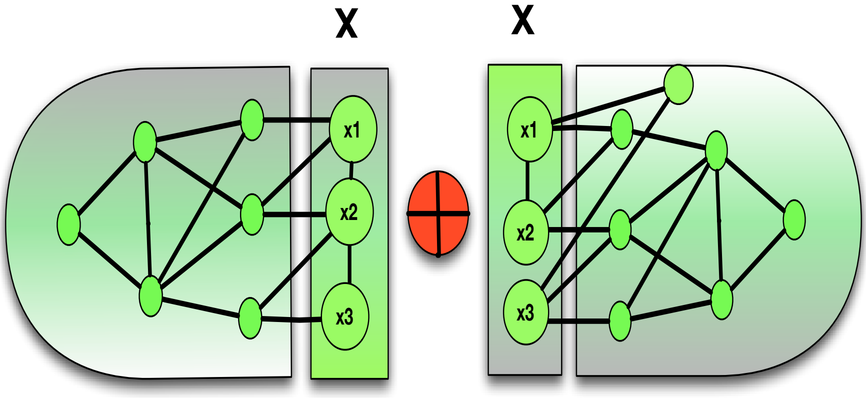
Another Operation: Join Operation

Given two t-boundaried graphs G_1 and G_2, the join operation glues them together at the boundaries.



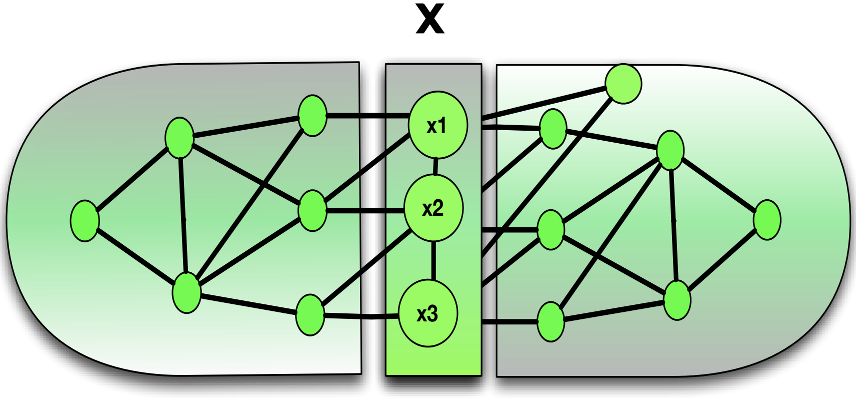
Another Operation: Join Operation

Given two t-boundaried graphs G_1 and G_2, the join operation glues them together at the boundaries.



Another Operation: Join Operation

Given two t-boundaried graphs G_1 and G_2, the join operation glues them together at the boundaries.



Joining G_1 and G_2: Updating the Table T for Maximum Independent Set

Have a table T_1 for G_1 and T_2 for G_2, want to compute the table T for their join.

$$T[S] = T_1[S] + T_2[S] - |S|$$

Update time: $\mathcal{O}(2^k$

Treewidth

- The treewidth($\mathbf{tw}(G)$) of G is the minimum boundary size needed to construct G from the empty graph using introduce, forget and join operations... -1
- Have seen: INDEPENDENT SET can be solved in $2^k k^{\mathcal{O}(1)} n$ time if a construction of G with k labels is given as input.

Tree Decomposition: canonical definition

A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \chi)$, where T is a tree and mapping χ assigns to every node t of T a vertex subset X_t (called a bag) such that

Tree Decomposition: canonical definition

A *tree decomposition* of a graph G is a pair $\mathcal{T} = (T, \chi)$, where T is a tree and mapping χ assigns to every node t of T a vertex subset X_t (called a bag) such that

(T1) $\bigcup_{t \in V(T)} X_t = V(G)$.

(T2) For every $vw \in E(G)$, there exists a node t of T such that bag $\chi(t) = X_t$ contains both v and w.

(T3) For every $v \in V(G)$, the set $\chi^{-1}(v)$, i.e. the set of nodes $T_v = \{t \in V(T) \mid v \in X_t\}$ forms a connected subgraph (subtree) of T.

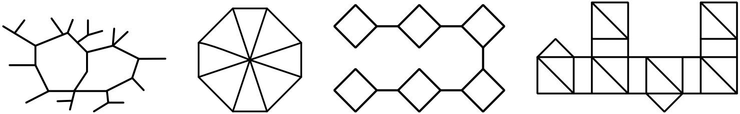
The *width* of tree decomposition $\mathcal{T} = (T, \chi)$ equals $\max_{t \in V(T)} |X_t| - 1$, i.e the maximum size of it s bag minus one. The *treewidth* of a graph G is the minimum width of a tree decomposition of G.

Treewidth Applications

- Graph Minors
- Parameterized Algorithms
- Exact Algorithms
- Approximation Schemes
- Kernelization
- Databases
- CSP's
- Bayesian Networks
- AI
- ...

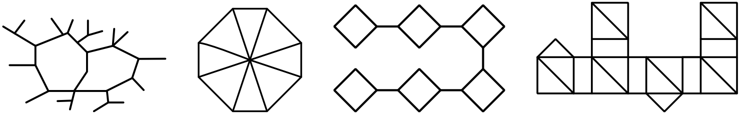
Exercise: What are the widths of these graphs?

1. Number of cycles is bounded.



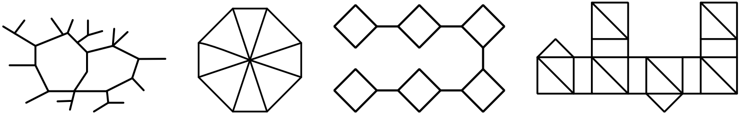
good bad bad bad

2. Removing a bounded number of vertices makes it acyclic.



good good bad bad

3. Bounded-size parts connected in a tree-like way.



bad bad good good

Treewidth

- Discovered and rediscovered many times: Halin 1976, Bertelé and Brioschi, 1972
- In the 80's as a part of Robertson and Seymour's Graph Minors project.
- Arnborg and Proskurowski: algorithms

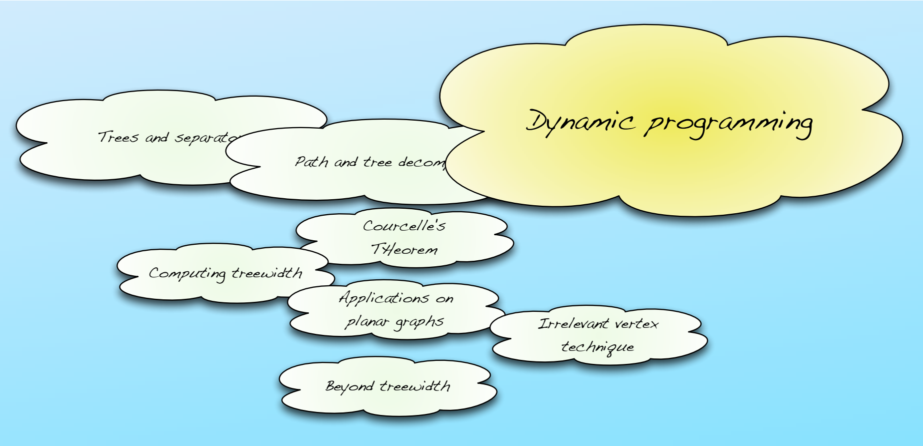
Separation Property

For every pair of adjacent nodes of the path of a path decomposition, the intersection of the corresponding bags is a separator.

Separation Property

For every pair of adjacent nodes of the path of a path decomposition, the intersection of the corresponding bags is a separator.

Treewidth also has similar properties—every bag is a separator.


Dynamic programming
Trees and separato
Path and tree decom
Courcelle's Theorem
Computing treewidth
Applications on planar graphs
Irrelevant vertex technique
Beyond treewidth

Reminder: Solving the Party Problem on trees

T_v: the subtree rooted at v.
$A[v]$: max. weight of an independent set in T_v
$B[v]$: max. weight of an independent set in T_v that does not contain v

Goal: determine $A[r]$ for the root r.

Method:

Assume $v_1, \ldots, v_k$ are the children of v. Use the recurrence relations

$$B[v] = \sum_{i=1}^{k} A[v_i]$$
$$A[v] = \max\{B[v]\ ,\ w(v) + \sum_{i=1}^{k} B[v_i]\}$$

The values $A[v]$ and $B[v]$ can be calculated in a bottom-up order (the leaves are trivial).

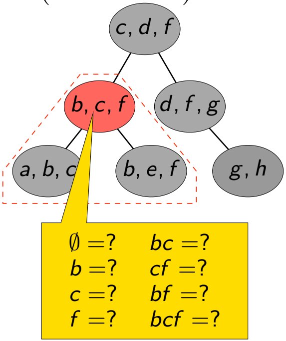
Weighted Max Independent Set
and tree decompositions

Fact: Given a tree decomposition of width k, Weighted Max Independent Set can be solved in time $\mathcal{O}(2^k k^{\mathcal{O}(1)} \cdot n)$.

X_t: vertices appearing in node t.
V_t: vertices appearing in the subtree rooted at t.

Generalizing our solution for trees:

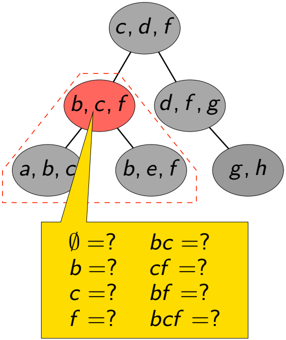
Instead of computing two values $A[v]$, $B[v]$ for each **vertex** of the graph, we compute $2^{|X_t|} \leq 2^{k+1}$ values for each bag X_t.



Weighted Max Independent Set
and tree decompositions

X_t: vertices appearing in node t.
V_t: vertices appearing in the subtree rooted at t.

$c[t, S]$: the maximum weight of an independent set $I \subseteq V_t$ with $I \cap X_t = S$.

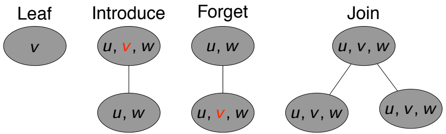


How to determine $c[t, S]$ if all the values are known for the children of t?

Nice tree decompositions

Definition: A rooted tree decomposition is **nice** if every node t is one of the following 4 types:

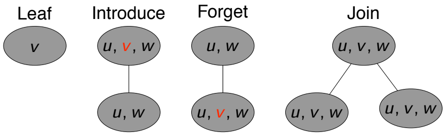
- **Leaf:** no children, $|X_t| = 1$
- **Introduce:** one child q, $X_t = X_q \cup \{v\}$ for some vertex v
- **Forget:** one child q, $X_t = X_q \setminus \{v\}$ for some vertex v
- **Join:** two children t_1, t_2 with $X_t = X_{t_1} = X_{t_2}$



Nice tree decompositions

Definition: A rooted tree decomposition is **nice** if every node t is one of the following 4 types:

- **Leaf:** no children, $|X_t| = 1$
- **Introduce:** one child q, $X_t = X_q \cup \{v\}$ for some vertex v
- **Forget:** one child q, $X_t = X_q \setminus \{v\}$ for some vertex v
- **Join:** two children t_1, t_2 with $X_t = X_{t_1} = X_{t_2}$

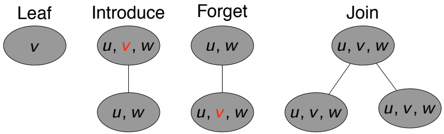


Fact: A tree decomposition of width k and n nodes can be turned into a nice tree decomposition of width k and $O(kn)$ nodes in time $O(k^2n)$.

Weighted Max Independent Set
and nice tree decompositions

- **Leaf:** no children, $|X_t| = 1$
  Trivial!
- **Introduce:** one child q, $X_t = X_q \cup \{v\}$ for some vertex v

$$c[t,S] = \begin{cases} c[q,S] & \text{if } v \notin S, \\ c[q, S \setminus \{v\}] + w(v) & \text{if } v \in S \text{ but } v \text{ has no neighbor in } S, \\ -\infty & \text{if } S \text{ contains } v \text{ and its neighbor.} \end{cases}$$



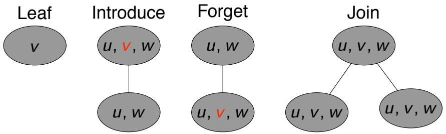
Weighted Max Independent Set
and nice tree decompositions

- **Forget:** one child y, $X_t = X_q \setminus \{v\}$ for some vertex v

$$c[t, S] = \max\{c[q, S], c[q, S \cup \{v\}]\}$$

- **Join:** two children t_1, t_2 with $X_t = X_{t_1} = X_{t_2}$

$$c[t, S] = c[t_1, S] + c[t_2, S] - w(S)$$



Weighted Max Independent Set
and nice tree decompositions

- **Forget:** one child y, $X_t = X_q \setminus \{v\}$ for some vertex v

$$c[t, S] = \max\{c[q, S], c[q, S \cup \{v\}]\}$$

- **Join:** two children t_1, t_2 with $X_t = X_{t_1} = X_{t_2}$

$$c[t, S] = c[t_1, S] + c[t_2, S] - w(S)$$

There are at most $2^{k+1} \cdot n$ subproblems $c[t, S]$ and each subproblem can be solved in $\mathcal{O}(n)$ time (assuming the children are already solved). There is a trick [exercise] to reduce it to $\mathcal{O}(k)$. $\Rightarrow$ Running time is $\mathcal{O}(2^k \cdot k^{\mathcal{O}(1)} n)$.

Dominating Set

Exercise

Show how to solve the dominating set problem in $5^k k^{\mathcal{O}(1)} n$ time on graphs of treewidth k.

Dominating Set

Exercise

Show how to solve the dominating set problem in $5^k k^{\mathcal{O}(1)} n$ time on graphs of treewidth k.

Each vertex can be in one of three states:

- chosen to the solution,
- not chosen, not yet dominated,
- not chosen, dominated.

But join operation is expensive.

Dominating Set

Exercise

Show how to solve the dominating set problem in $5^k k^{\mathcal{O}(1)} n$ time on graphs of treewidth k.

Each vertex can be in one of three states:

- chosen to the solution,
- not chosen, not yet dominated,
- not chosen, dominated.

But join operation is expensive. It is possible to improve to $3^k k^{\mathcal{O}(1)} n$ by making use of subset convolution (later...)

Steiner tree

We are given an undirected graph G and a set of vertices $K \subseteq V(G)$, called *terminals*. The goal is to find a subtree H of G of the minimum possible size (that is, with the minimum possible number of edges) that connects all the terminals.

Fact: Given a tree decomposition of width k, Steiner tree can be solved in time $k^{\mathcal{O}(k)} \cdot n$.

Treewidth DP for Steiner tree

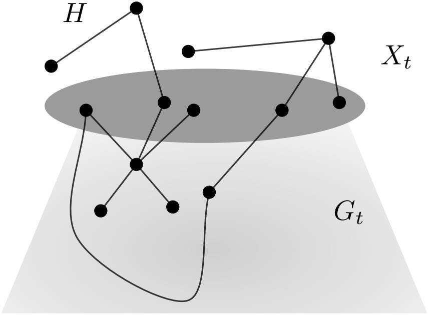


Figure : Steiner tree H intersecting bag X_t and graph G_t.

Treewidth DP for Steiner tree

Idea: Construct forest F in G_t such that

Every terminal from $K \cap V_t$ should belong to some connected component of F.
Encode this information by keeping, for each subset $X \subseteq X_t$ and each partition $\mathcal{P}$ of X, the minimum size of a forest F in G_t such that

(a) $K \cap V_t \subseteq V(F)$, i.e., F spans all terminals from V_t,

(b) $V(F) \cap X_t = X$, and

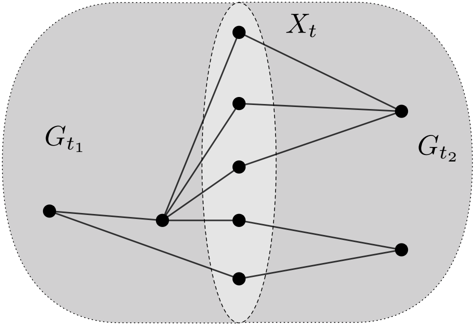
(c) the intersections of X_t with vertex sets of connected components of F form exactly the partition $\mathcal{P}$ of X.

Treewidth DP for Steiner tree

- When we introduce a new vertex or join partial solution (at join nodes), the connected components of partial solutions could merge and thus we need to keep track of the updated partition into connected components.

Treewidth DP for Steiner tree

- When we introduce a new vertex or join partial solution (at join nodes), the connected components of partial solutions could merge and thus we need to keep track of the updated partition into connected components.
- How to avoid cycles in join operations?



Treewidth DP for Steiner tree

- At the end, everything boils down to going to all possible partitions of all bags, which is, roughly $k^k \cdot n$.

Treewidth DP for Steiner tree

- At the end, everything boils down to going to all possible partitions of all bags, which is, roughly $k^k \cdot n$.
- We will see how single-exponential $2^{\mathcal{O}(k)}$ on treewidth can be obtained later.

Treewidth DP

Conclusion

The main challenge for most of the problems is to understand what information to store at nodes of the tree decomposition. Obtaining formulas for forget, introduce and join nodes can be a tedious task, but is usually straightforward once a precise definition of a state is established.

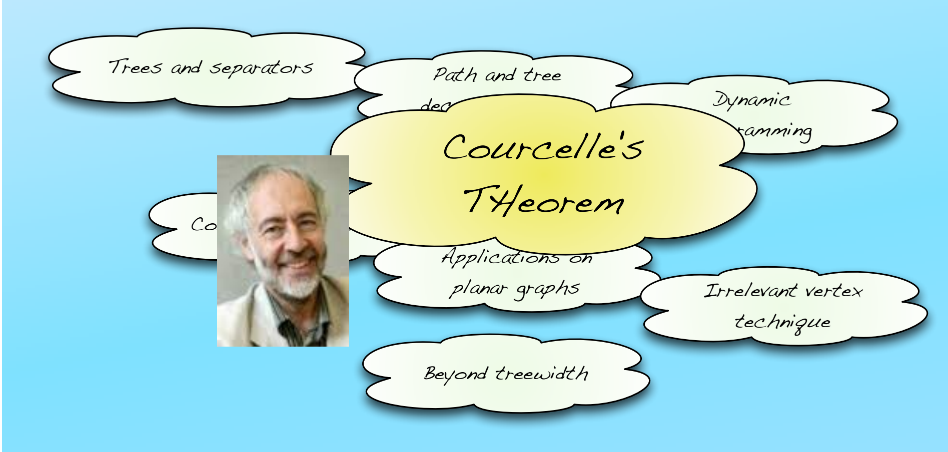
Fact

Independent Set, Dominating Set, q-Coloring, Max-Cut, Odd Cycle Transversal, Hamiltonian Cycle, Partition into Triangles, Feedback Vertex Set, Vertex Disjoint Cycle Packing and million other problems are FPT parameterized by the treewidth.

Meta-theorem for treewidth DP

While arguments for each of the problems are different, there are a lot of things in common...

Coming soon...


Trees and separators
Path and tree
Dynamic
Courcelle's Theorem
Applications on planar graphs
Irrelevant vertex technique
Beyond treewidth