treewidth and applications

1 an insight into graph theory

classical NP-hard problems

We consider always undirected simple graphs $G = (V, E)$

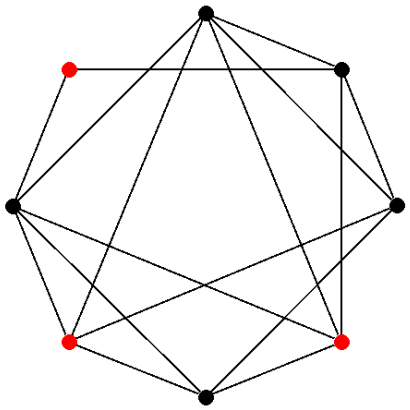
- $\omega(G)$: size of the maximum clique
- $\chi(G)$: chromatic number = smallest k such as G has a k-coloring [NP-hard for $(k \geq 3)$]
- longest (weighted) path (+ special case: Hamiltonian path), maximum cut, ...
- $\alpha(G)$: size of the maximum independent set

1 an insight into graph theory

classical NP-hard problems

We consider always undirected simple graphs $G = (V, E)$

- $\omega(G)$: size of the maximum clique
- $\chi(G)$: chromatic number = smallest k such as G has a k-coloring [NP-hard for $(k \geq 3)$]
- longest (weighted) path (+ special case: Hamiltonian path), maximum cut, ...
- $\alpha(G)$: size of the maximum independent set $\triangleleft \triangleleft \triangleleft$



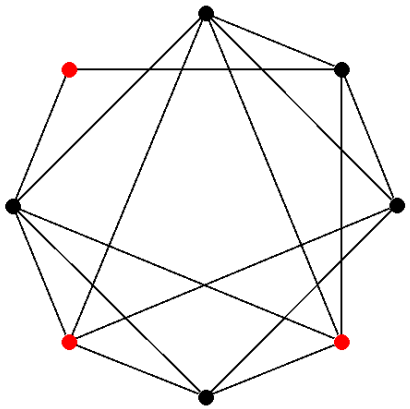
⟵ maximum independant set

1 an insight into graph theory

classical NP-hard problems

We consider always undirected simple graphs $G = (V, E)$

- $\omega(G)$: size of the maximum clique
- $\chi(G)$: chromatic number = smallest k such as G has a k-coloring [NP-hard for $(k \geq 3)$]
- longest (weighted) path (+ special case: Hamiltonian path), maximum cut, ...
- $\alpha(G)$: size of the maximum independent set $\triangleleft \triangleleft \triangleleft$



⟵ maximum independant set

- $\alpha(G) = \omega(\overline{G})$
- $\chi(G) \geq \omega(G)$
- ...

some classes of graphs

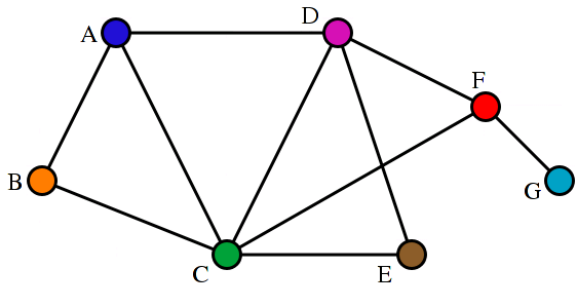
IDEA: problems become easier when graphs have some extra properties

- **trees** $\longleftarrow$ acyclic
- **bipartite graphs** $\longleftarrow \chi(G) = 2$
- **perfect graphs** $\longleftarrow \omega(H) = \chi(H)$ for every induced subgraph (incl. G itself)
- **planar graphs** $\longleftarrow$ drawable in a plane with no crossing edge
- **outerplanar graphs** $\longleftarrow$ planar graphs with all the vertices on the outer face
- **interval graphs** $\longleftarrow$ intersection graphs of intervals
- **chordal graphs** $\longleftarrow$ every cycle has a chord
- **SP-graphs** $\longleftarrow$ recursively obtained by combinations in series and parallel

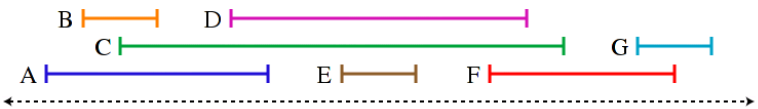
some classes of graphs

IDEA: problems become easier when graphs have some extra properties

- **trees** ⟵ acyclic
- **bipartite graphs** ⟵ $\chi(G) = 2$
- **perfect graphs** ⟵ $\omega(H) = \chi(H)$ for every induced subgraph (incl. G itself)
- **planar graphs** ⟵ drawable in a plane with no crossing edge
- **outerplanar graphs** ⟵ planar graphs with all the vertices on the outer face
- **interval graphs** ⟵ intersection graphs of intervals ◁ ◁ ◁
- **chordal graphs** ⟵ every cycle has a chord
- **SP-graphs** ⟵ recursively obtained by combinations in series and parallel



⟵ interval graph



some classes of graphs

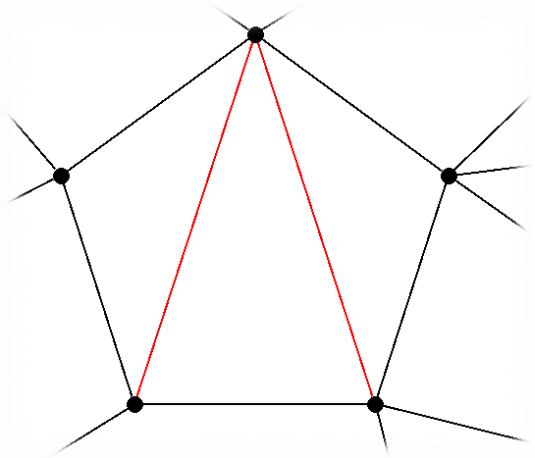
IDEA: problems become easier when graphs have some extra properties

- **trees** $\longleftarrow$ acyclic
- **bipartite graphs** $\longleftarrow \chi(G) = 2$
- **perfect graphs** $\longleftarrow \omega(H) = \chi(H)$ for every induced subgraph (incl. G itself)
- **planar graphs** $\longleftarrow$ drawable in a plane with no crossing edge
- **outerplanar graphs** $\longleftarrow$ planar graphs with all the vertices on the outer face
- **interval graphs** $\longleftarrow$ intersection graphs of intervals
- **chordal graphs** $\longleftarrow$ every cycle has a chord
- **SP-graphs** $\longleftarrow$ recursively obtained by combinations in series and parallel

some classes of graphs

IDEA: problems become easier when graphs have some extra properties

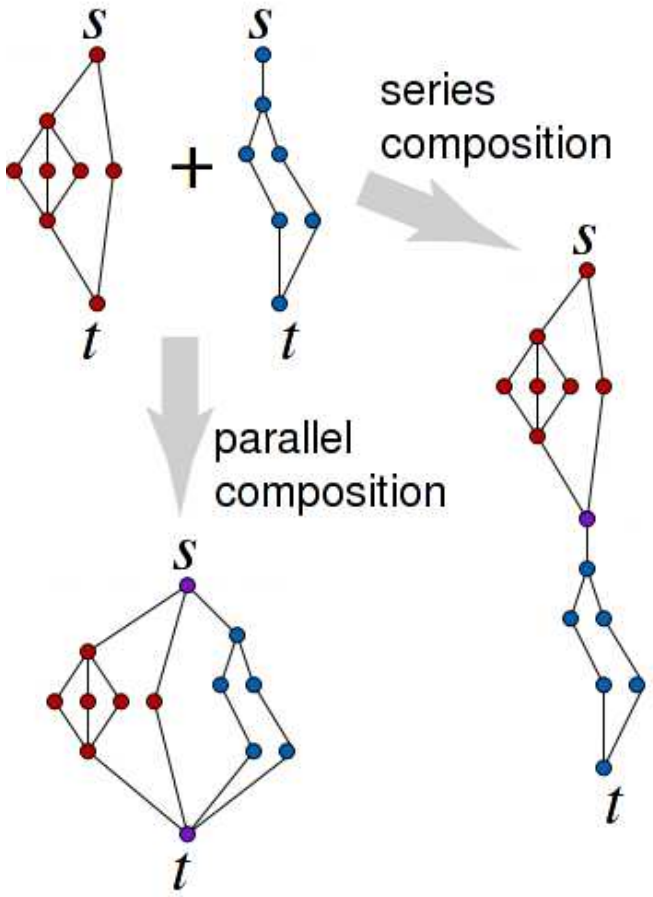
- **trees** $\longleftarrow$ acyclic
- **bipartite graphs** $\longleftarrow$ $\chi(G) = 2$
- **perfect graphs** $\longleftarrow$ $\omega(H) = \chi(H)$ for every induced subgraph (incl. G itself)
- **planar graphs** $\longleftarrow$ drawable in a plane with no crossing edge
- **outerplanar graphs** $\longleftarrow$ planar graphs with all the vertices on the outer face
- **interval graphs** $\longleftarrow$ intersection graphs of intervals
- **chordal graphs** $\longleftarrow$ every cycle has a chord $\triangleleft\triangleleft\triangleleft$
- **SP-graphs** $\longleftarrow$ recursively obtained by combinations in series and parallel



$\longleftarrow$ chordal graph

some classes of graphs

- **SP-graphs** ⟵ recursively obtained by combinations in series and parallel ◁ ◁ ◁



⟵ series-parallel graph (SP-graph)

some classes of graphs

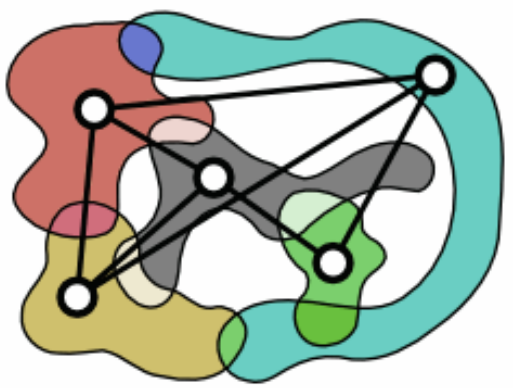
- **trees** $\longleftarrow$ acyclic
- **bipartite graphs** $\longleftarrow \chi(G) = 2$
- **perfect graphs** $\longleftarrow \omega(H) = \chi(H)$ for every induced subgraph (incl. G itself)
- **planar graphs** $\longleftarrow$ drawable in a plane with no crossing edge
- **outerplanar graphs** $\longleftarrow$ planar graphs with all the vertices on the outer face
- **interval graphs** $\longleftarrow$ intersection graphs of intervals
- **chordal graphs** $\longleftarrow$ every cycle has a chord
- **SP-graphs** $\longleftarrow$ recursively obtained by combinations in series and parallel

straigtforward relations

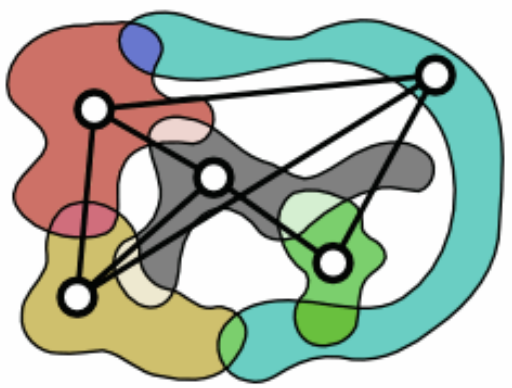
- trees are bipartite
- interval graphs are chordal
- outerplanar graphs are SP-graphs
- SP-graphs are planar
- chordal graphs are perfect
- ...

graphs as intersection graphs

DEF: A graph is an intersection graph if there exists a universe Λ and for every vertex v a set $S_v \subset \Lambda$ of objects of the universe such that edge (v, w) exists if and only if S_v and S_w intersect.



graphs as intersection graphs

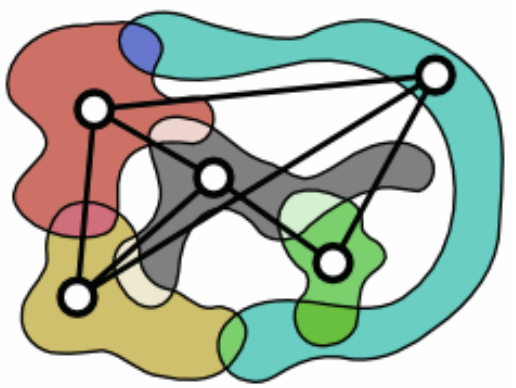


DEF: A graph is an intersection graph if there exists a universe Λ and for every vertex v a set $S_v \subset \Lambda$ of objects of the universe such that edge (v, w) exists if and only if S_v and S_w intersect.

TH: Every graph is an intersection graph.

- restrictions on the universe Λ leads to new classes of graphs
- interval graphs = intersection graphs of intervals
- chordal graphs are intersection graphs of subtrees of a tree

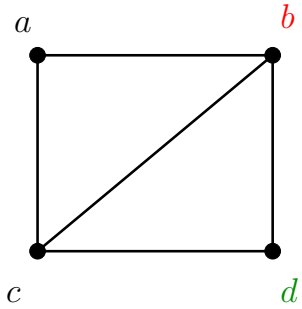
graphs as intersection graphs

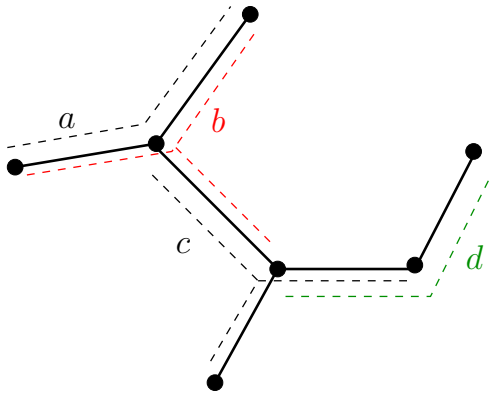


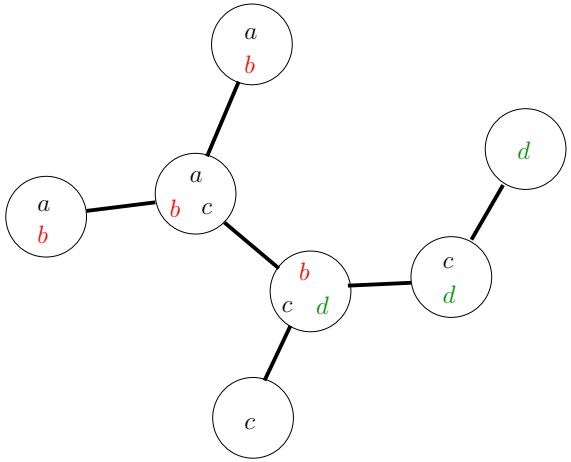
DEF: A graph is an intersection graph if there exists a universe Λ and for every vertex v a set $S_v \subset \Lambda$ of objects of the universe such that edge (v, w) exists if and only if S_v and S_w intersect.

TH: Every graph is an intersection graph.

- restrictions on the universe Λ leads to new classes of graphs
- interval graphs = intersection graphs of intervals
- chordal graphs are intersection graphs of subtrees of a tree

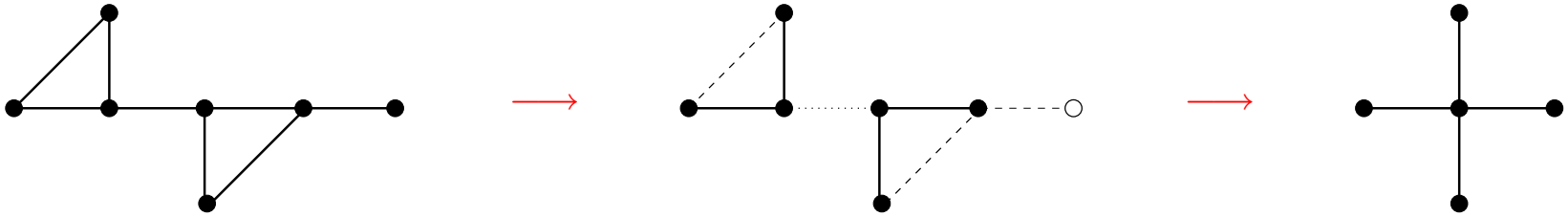






the Graph Minor Theorem

DEF: graph minor: delete vertices and edges, contract edges



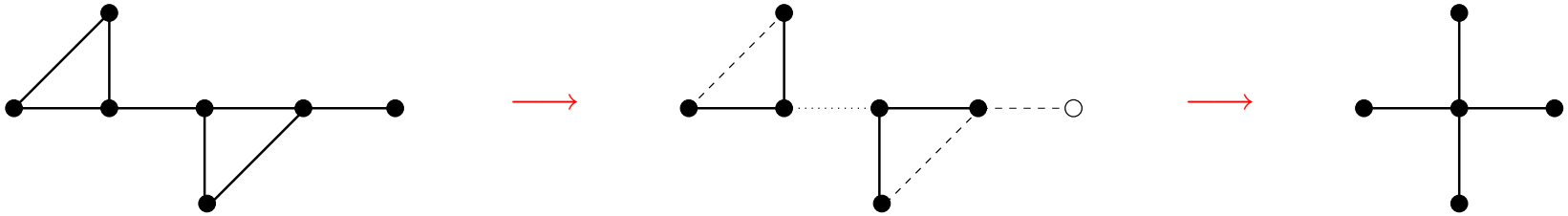
TH: [Robertson-Seymour-83 $\longrightarrow$ 04]

Every minor-hereditary graph class has a finite set of forbidden minors.

Trees = $< C_3 >$, Planars = $< K_{3,3}, K_5 >$, SP-Graphs = $< K_4 >$

the Graph Minor Theorem

DEF: graph minor: delete vertices and edges, contract edges



TH: [Robertson-Seymour-83 $\longrightarrow$ 04]
Every minor-hereditary graph class has a finite set of forbidden minors.
Trees = $< C_3 >$, Planars = $< K_{3,3}, K_5 >$, SP-Graphs = $< K_4 >$

treewidth and tree decomposition

IDEA: problems are usually much simpler when the graph is acyclic
try to measure "how far" is a graph from being a tree

- $\omega(G)$, $\alpha(G)$, $\chi(G)$...
- $tw(G)$: treewidth of G [NP-hard if the expected treewidth is part of the input] ◁ ◁ ◁

2 treewidth: definition + properties

dynamic programming on trees

EX: Maximum independent set on $T = (V, E)$ rooted at $r \in V$

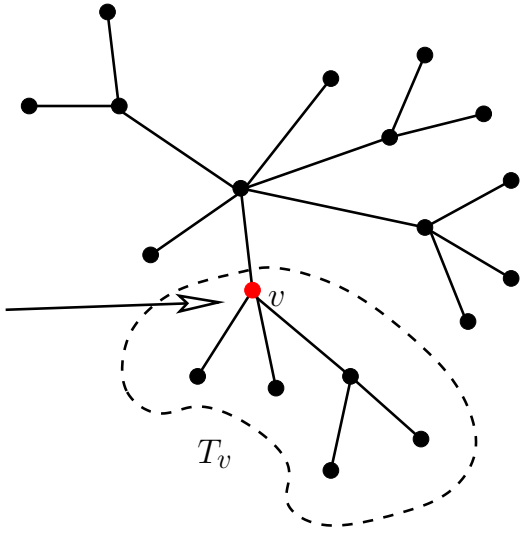
For $v \in V$, let T_v denote the subtree rooted at v.
Let $f^+(v)$ be the size of a maximum independent set for T_v that contains v.
Similarly, $f^-(v)$ is the size of a maximum independent set for T_v that does not contain v.

The following algorithm computes a maximum independent set for T in $0(|V|)$ time.

Traverse T starting from r in post order.
Let v be the current vertex.

- if v is a leaf, let $f^+(v) = 1$ and $f^-(v) = 0$.
- else let $x_1, ..., x_k$ be the children of v.
  Set $f^+(v) = 1 + \sum_{i=1}^{k} f^-(x_i)$
  and $f^-(v) = \sum_{i=1}^{k} max\{f^+(x_i), f^-(x_i)\}$

Return $max\{f^+(r), f^-(r)\}$



tree decomposition: definition

DEF: A tree decomposition for a graph $G = (V, E)$ is a pair

$$(\{X_i | i \in I\}, \quad T = (I, F))$$

bags tree

such that

- $\cup_{i \in I} X_i = V$ (bags cover vertices)
- for each $\{u, v\} \in E$ there is some $i \in I$ so that $\{u, v\} \subset X_i$ (bags cover edges)
- for all $v \in V$ the set $I_v = \{i \in I | v \in X_i\}$ is connected in T (tree property)

tree decomposition: definition

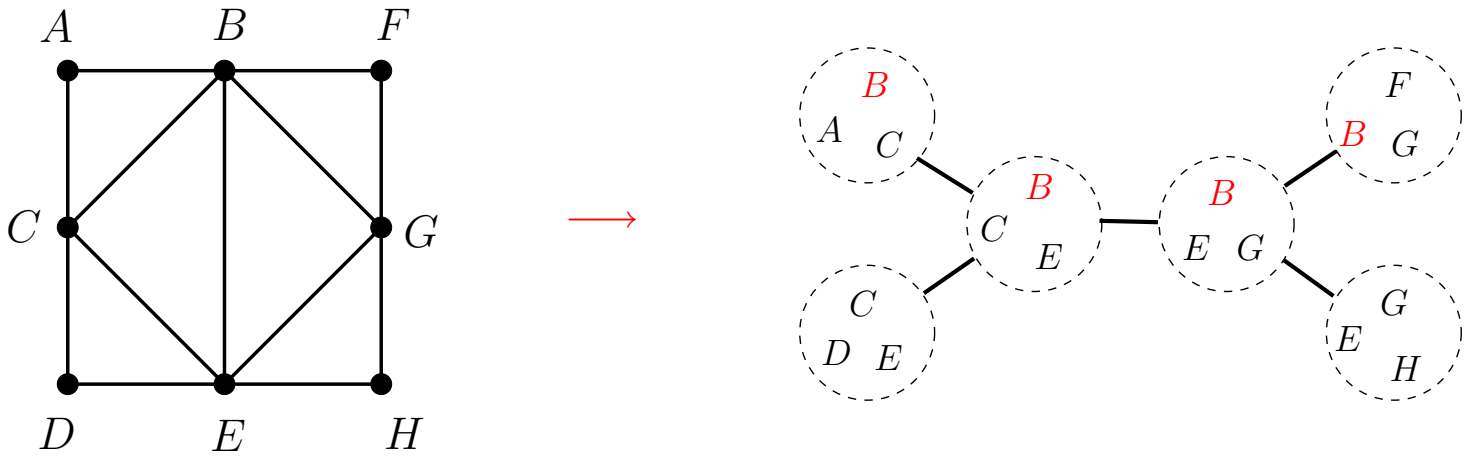
DEF: A tree decomposition for a graph $G = (V, E)$ is a pair

$$(\{X_i | i \in I\}, \quad T = (I, F))$$

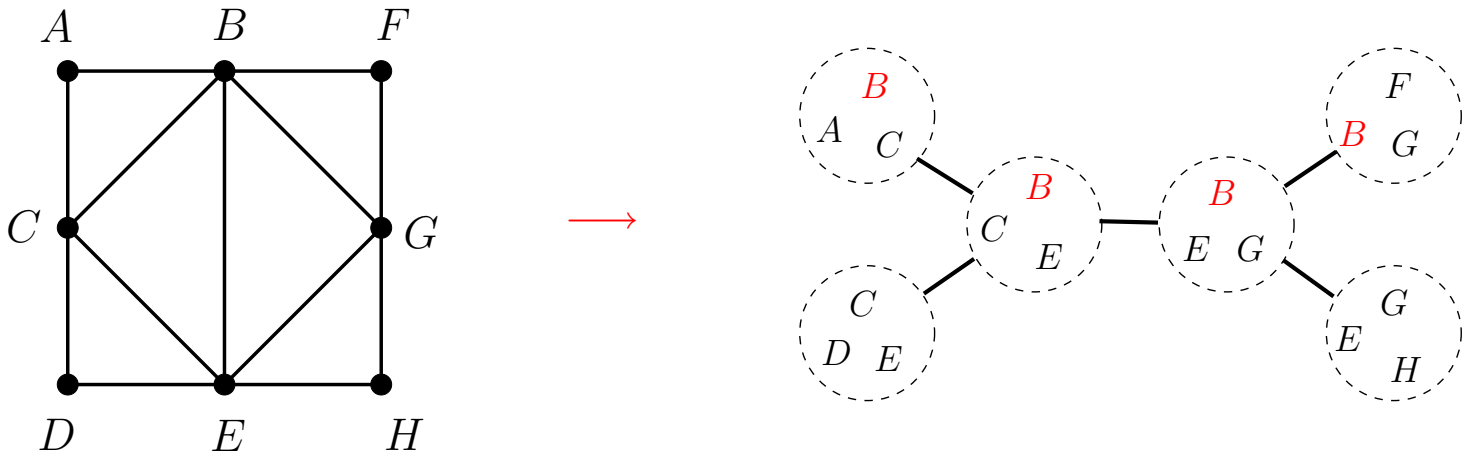
bags tree

such that

- $\cup_{i \in I} X_i = V$ (bags cover vertices)
- for each $\{u, v\} \in E$ there is some $i \in I$ so that $\{u, v\} \subset X_i$ (bags cover edges)
- for all $v \in V$ the set $I_v = \{i \in I | v \in X_i\}$ is connected in T (tree property)



tree decomposition: definition



Intuitively, a tree decomposition represents the vertices of the given graph as subtrees of a tree, in such a way that vertices are adjacent only when the corresponding subtrees intersect.

DEF: the **width** of a tree decomposition is $max_{i \in I} |X_i| - 1$.
DEF: the **treewidth** of a graph G, noted $tw(G)$, is the min width of all its tree decompositions
DEF: a graphs of treewidth at most k is called a **partial k-tree**

treewidth: examples and properties

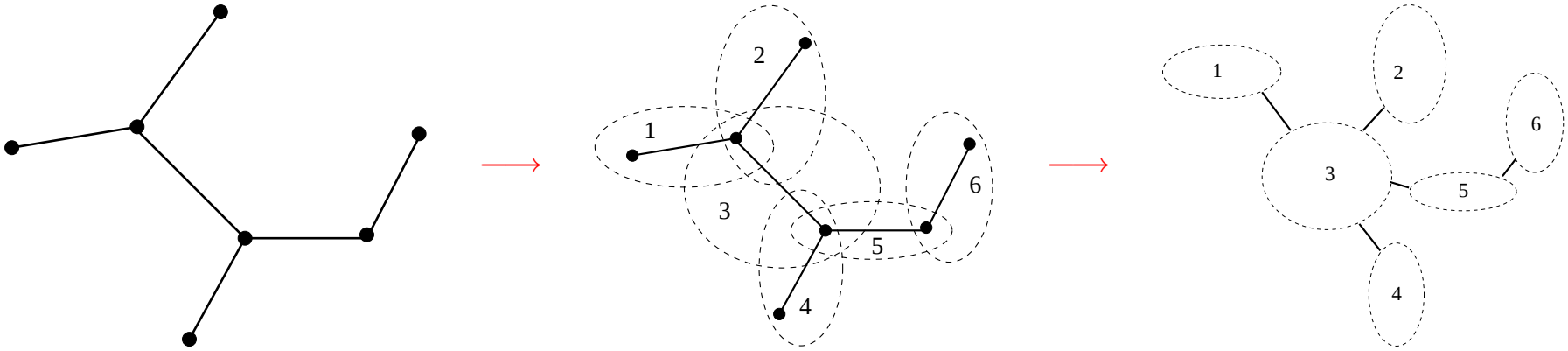
We are interested in tree decompositions of small treewidth, which certify that the graph is in some way “tree-like”.

EX: trees have treewidth 1

treewidth: examples and properties

We are interested in tree decompositions of small treewidth, which certify that the graph is in some way “tree-like”.

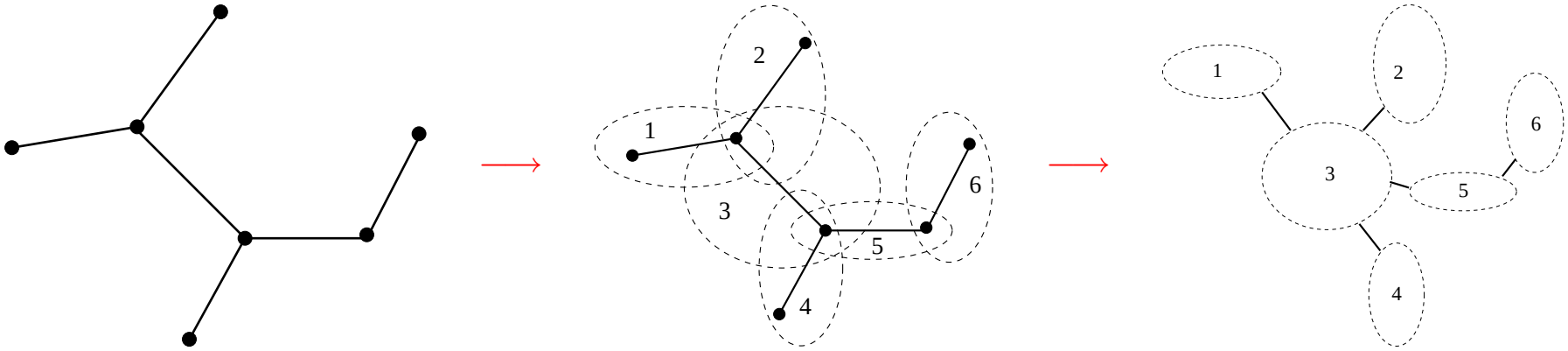
EX: trees have treewidth 1



treewidth: examples and properties

We are interested in tree decompositions of small treewidth, which certify that the graph is in some way “tree-like”.

EX: trees have treewidth 1

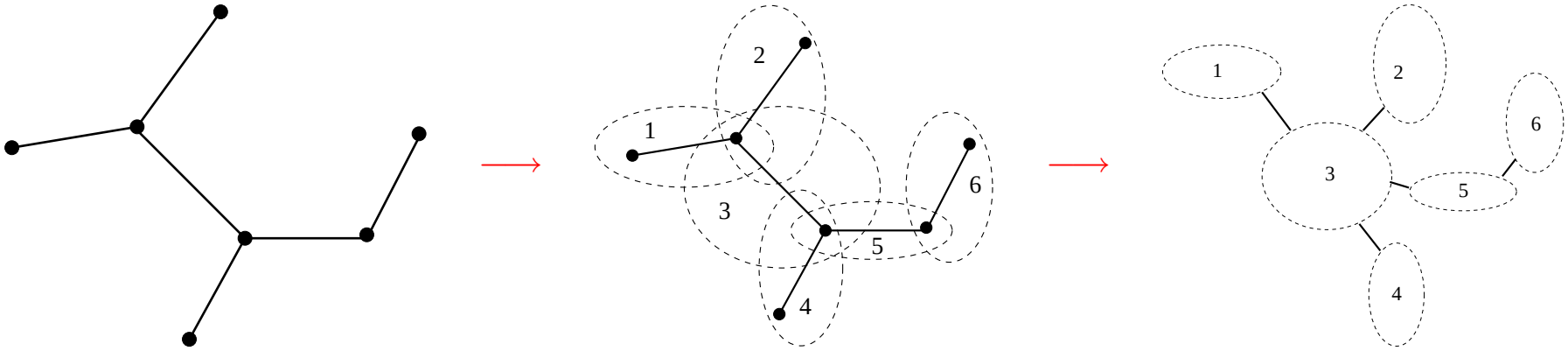


EX: For any $G = (V, E)$ a single bag containing V forms a tree decomposition of width $n - 1$.

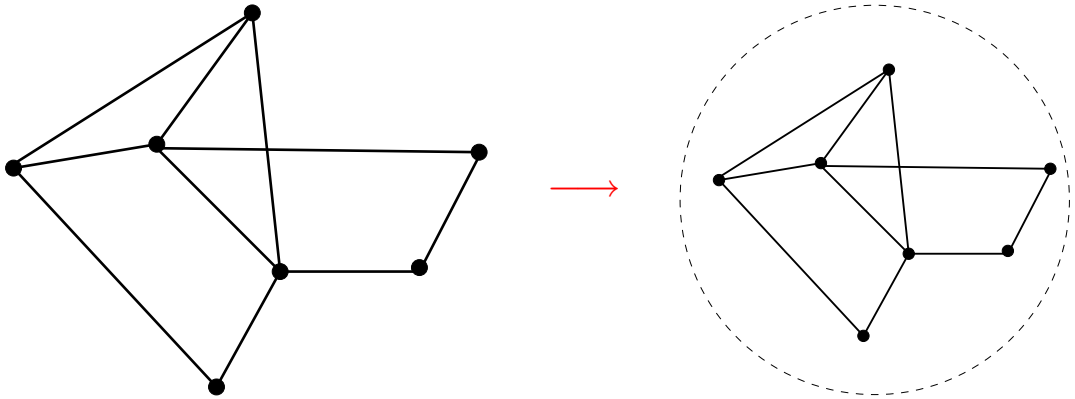
treewidth: examples and properties

We are interested in tree decompositions of small treewidth, which certify that the graph is in some way “tree-like”.

EX: trees have treewidth 1



EX: For any $G = (V, E)$ a single bag containing V forms a tree decomposition of width $n - 1$.



PROP: $tw(H) \leq tw(G)$ for any subgraph H of a graph G.

PROP: if a graph $G = (V, E)$ has two components A and B, with $A \cup B = V$ then $tw(G) = max\{tw(A), tw(B)\}$.

PROP: Let $(\{X_i | i \in I\}, T = (I, F))$ be a tree decomposition for $G = (V, E)$. For any clique $G[W], W \subset V$, there is an $i \in I$ such that $W \subset X_i$.

PROP: $tw(H) \leq tw(G)$ for any subgraph H of a graph G.

PROP: if a graph $G = (V, E)$ has two components A and B, with $A \cup B = V$ then $tw(G) = max\{tw(A), tw(B)\}$.

PROP: Let $(\{X_i | i \in I\}, T = (I, F))$ be a tree decomposition for $G = (V, E)$. For any clique $G[W], W \subset V$, there is an $i \in I$ such that $W \subset X_i$.

EX: the treewidth of K_n is $n - 1$

EX: SP-graphs have treewidth 2

EX: the $n \times n$-grid has treewidth n

PROP: $tw(H) \leq tw(G)$ for any subgraph H of a graph G.

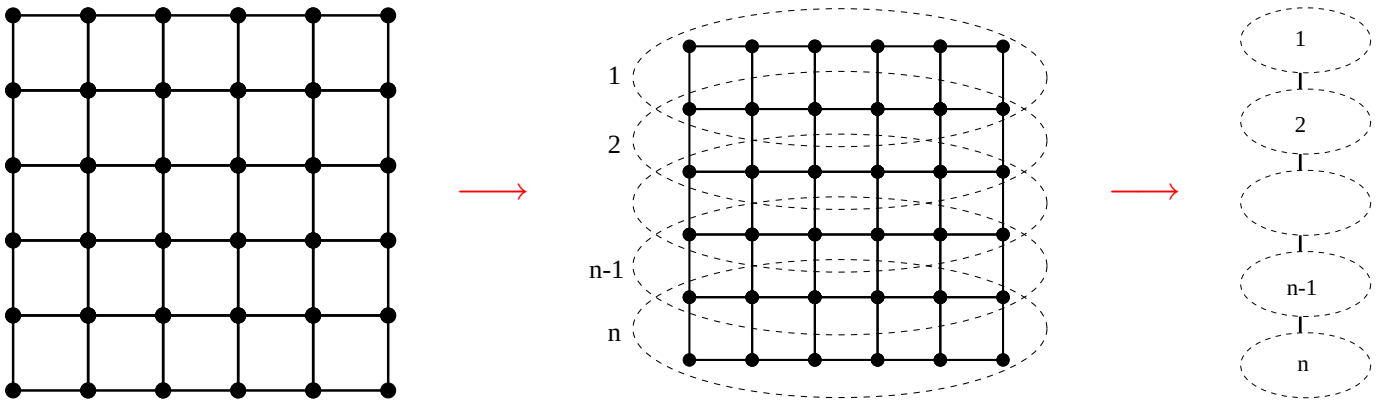
PROP: if a graph $G = (V, E)$ has two components A and B, with $A \cup B = V$ then $tw(G) = max\{tw(A), tw(B)\}$.

PROP: Let $(\{X_i | i \in I\}, T = (I, F))$ be a tree decomposition for $G = (V, E)$. For any clique $G[W], W \subset V$, there is an $i \in I$ such that $W \subset X_i$.

EX: the treewidth of K_n is $n - 1$

EX: SP-graphs have treewidth 2

EX: the $n \times n$-grid has treewidth n



A tree decomposition of width $2n - 1$

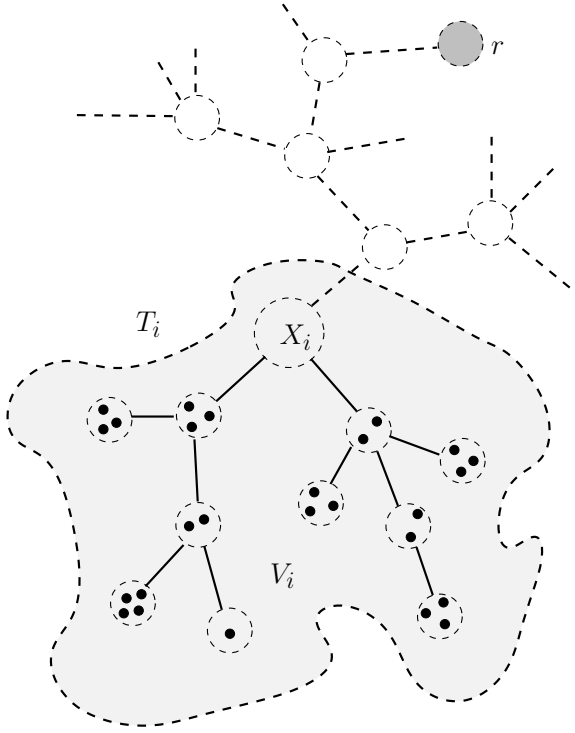
dynamic programming on partial k-trees

EX: Again, maximum independent set for a graph $G = (V, E)$,
if we are provided a tree decomposition $(\{X_i | i \in I\}, T = (I, F))$ of width $\leq k$.

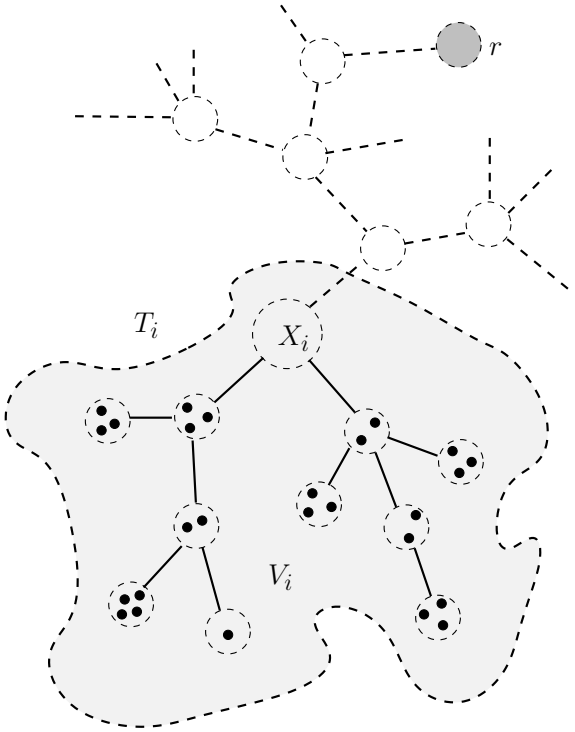
The algorithm below computes a maximum independent set for G in $O(k^2 4^k |V|)$ time.

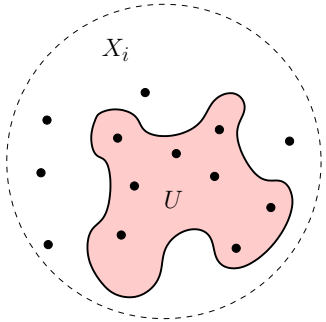
Pick an arbitrary root $r \in I$ and for $i \in I$,
let $V_i = \cup_{j \in T_i} X_j$ where T_i denotes the subtree rooted at i.
For $U \subset X_i$ let $f^U(i)$ be the size of a maximum independent subset of V_i whose intersection with X_i is exaclty U.

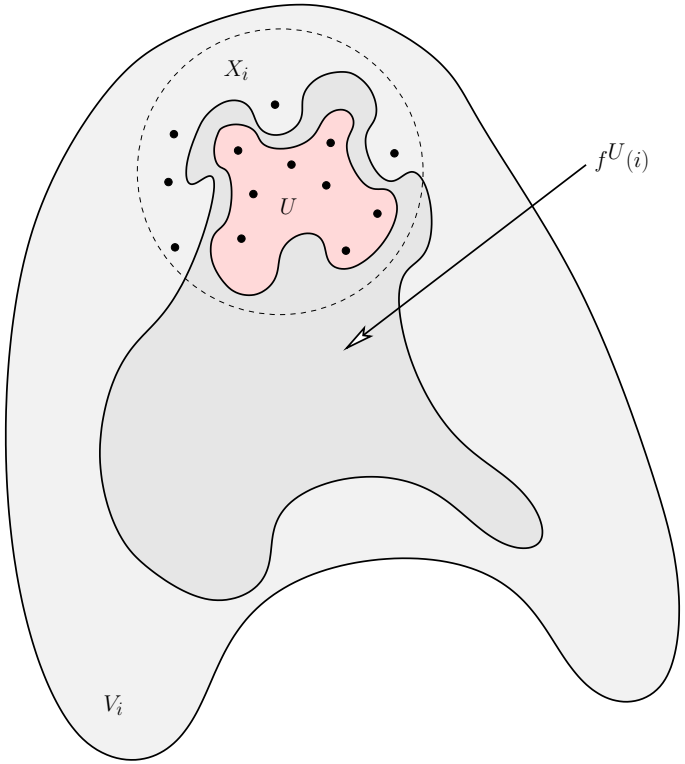
- traverse T starting from r in post order.
  Let i be the current index.
- if i is a leaf, for every $U \subset X_i$
  let $f^U(i) = |U|$ if U is independent in G
  and $f^U(i) = -\infty$, otherwise.
- else let $c_1, ..., c_l$ be the children of i.
  Set $f^U(i) = |U| + \sum_{j=1}^{l} max\{ f^W(c_j) | W \subset X_{c_j} - U$
  and $W \cup U$ is independent $\}$



Pick an arbitrary root $r \in I$ and for $i \in I$, let $V_i = \cup_{j \in T_i} X_j$ where T_i denotes the subtree rooted at i.
For $U \subset X_i$ let $f^U(i)$ be the size of a maximum independent subset of V_i whose intersection with X_i is exaclty U.







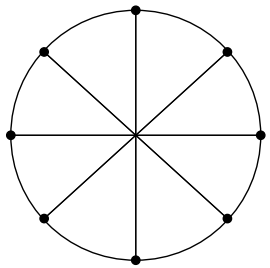
- traverse T starting from r in post order. Let i be the current index.
- if i is a leaf, for every $U \subset X_i$ let $f^U(i) = |U|$ if U is independent in G and $f^U(i) = -\infty$, otherwise.
- else let $c_1, ..., c_l$ be the children of i. Set $f^U(i) = |U| + \sum_{j=1}^{l} max\{f^W(c_j) | W \subset X_{c_j} - U$ and $W \cup U$ is indep. $\}$

characterization

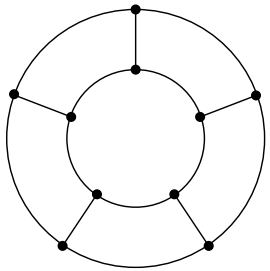
PROP: Partial k-trees are closed under taking minors.

By [Robertson-Seymour] there is hence a finite set of forbidden minors.
But they are not known, except for small k.

- $k = 1$ (trees) $\longrightarrow < K_3 >$
- $k = 2$ (SP-graphs) $\longrightarrow < K_4 >$
- $k = 3 \longrightarrow < K_5, K_{2,2,2},$ GR-1, GR-2 $>$
- $k = 4 \longrightarrow$ more than 75...



GR-1



GR-2

TH: [Arnborg-Corneil-Proskurowski-87]
Deciding whether the treewidth of a given graph is at most k is NP-complete.

TH: [Bodlaender-96]
For any $k \in \mathbb{N}$ there exists a linear time algorithm to test whether a given graph has treewidth at most k and – if so – output a corresponding tree decomposition.
RK: The runtime is exponential in k^3.

Open: Can the treewidth be computed in polynomial time for planar graphs ?

3 further discussions

propositions for further discussion

- perfect elimination ordering $\longrightarrow$ variable elimination in a CSP
- RNA secondary structure, pseudoknots, outerplanar graphs, relations with treewidth
- further discussions on dynamic programming (serial vs non-serial DP)
  + applications to bioinfo
- recursive treewidth ?

the dynamic programming paradigm (DP)

- the initial stage contains trivial solutions to sub-problems,
- each partial solution in a later stage can be calculated by recurring on only a fixed number of partial solutions in an earlier stage,
- the final stage contains the overall solution.

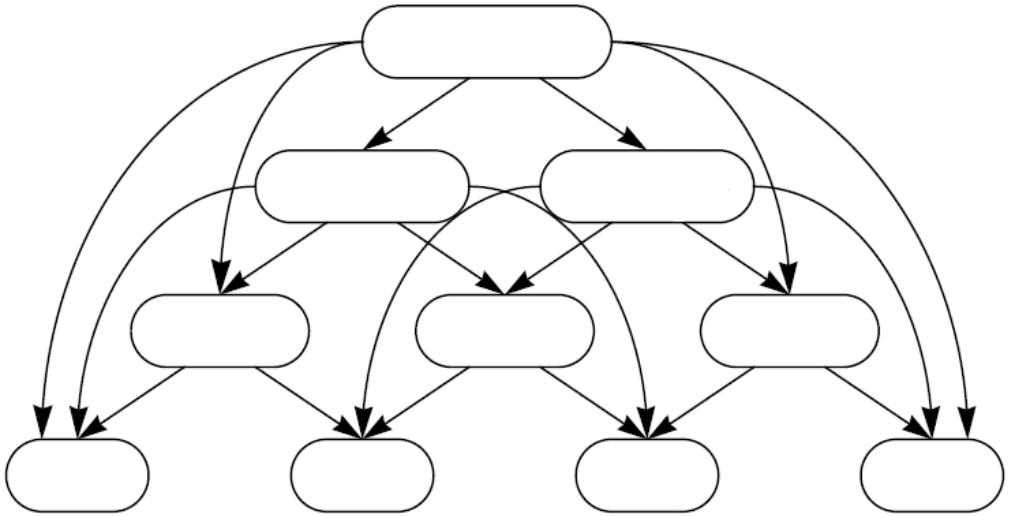


Illustration of DP on two examples coming from bioinformatics

In both examples we are dealing with substrings, and thus using the natural order on them

string edit distance (aka Levenshtein distance)

input: 2 strings of length $0(n)$ over a given alphabet Σ (eg. $\Sigma = \{a, t, c, g\}$)
operations: insertion, deletion, substitution
output: an alignment of the strings (or equivalently, an editing script)
optimization: each operation has a non negative cost, we seek to minimize the editing cost

(1) LEAD $\xrightarrow{del}$ LED $\xrightarrow{del}$ LD $\xrightarrow{ins}$ OLD $\xrightarrow{ins}$ GOLD
(2) LEAD $\xrightarrow{subs}$ GEAD $\xrightarrow{subs}$ GELD $\xrightarrow{subs}$ GOLD

(1)
- - L E A D
G O L - - D

(2)
L E A D
G O L D

For any string $m \in \Sigma^*$ we note $m[i..j]$ its substring starting at i and ending at j.

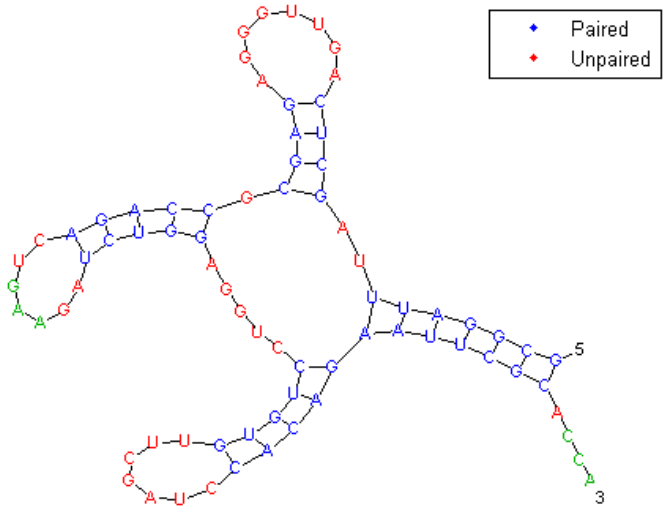
Given two strings a and b, we define $d(i,j) = mincost(a[1..i] \longrightarrow b[1..j])$
to be the minimal cost to edit the substring $a[1..i]$ into the substring $b[1..j]$.

$$d(i,j) = min \begin{cases} d(i-1,j) + del\_cost & \text{delete the last character of } a[1..i] \\ d(i,j-1) + ins\_cost & \text{delete the last character of } b[1..j] \\ d(i-1,j-1) + subs\_cost & \text{substitute } b[j] \text{ to } a[i] \end{cases}$$

- complexity of the algorithm: space $0(n)$ / time $0(n^2)$
- complexity of the problem (in the general case): time $0(n^2)$ [+ ref]
- refinements when imposing some restrictions: affine convex gap function + four russians speed-up [+ ref]

RNA secondary structure

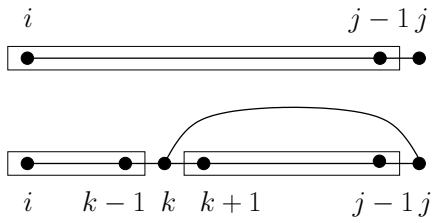
input: a string s over the alphabet $\{a, u, c, g\}$
operations: base pairings $a = u$, $c = g$ (watson-crick), and $g = u$ (wooble)
output: a list of pairings that forms a well parenthesisized expression
optimization: we seek the structure that gives the maximum number of pairings




```
5′  GCGGAUUUAGCUCAGUUGGGAGAGCGCCAGACUGAAGAUCUGGAGGUCCUGUGUUCGAUCCACAGAAUUCGCACCA   3′
    ((((((((..((((........)))).(((((.......)))))).....(((((.......)))))))))))....
```

$M(i,j)$ is defined as the number of pairings of the optimal secondary structure on the substring $s[i..j]$. The structure is optimal when it maximizes the number of pairings.

$$M(i,j) = max \begin{cases} M(i,j-1) \\ M(i,k-1) + M(k+1,j-1) + 1 \end{cases} \qquad (i \leq k < j)$$



- complexity of the algorithm: space $O(n^2)$ / time $O(n^3)$ [ref: nussinov]
- complexity of the problem: open

RK: this biologically unrealistic algorithm is actually the basis for a more relevant one, that includes a fully precise thermodynamic model of the molecule [zuker, vienna], leaving the complexities unchanged.

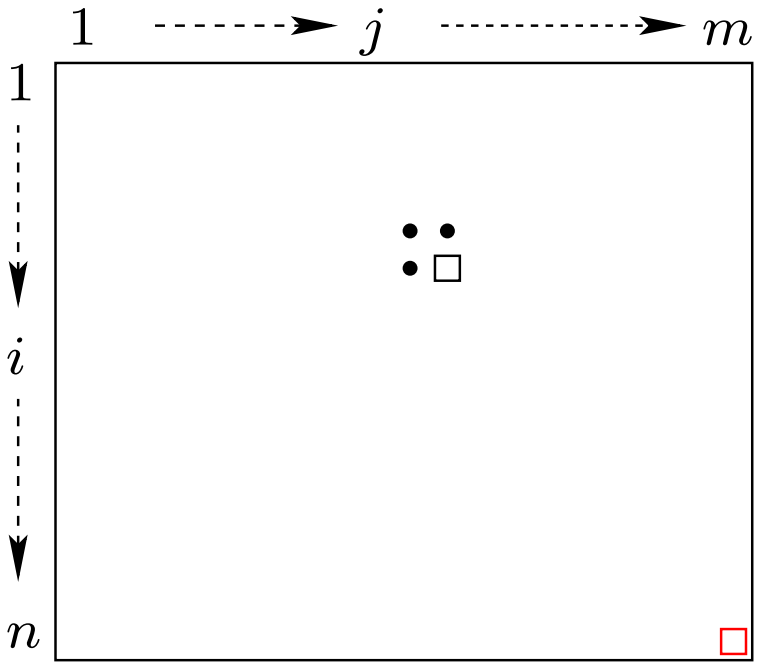
$1 \leq i \leq n$

$1 \leq j \leq m \qquad (m \in O(n))$

$space\ O(n)$

$time\ O(n^2)$

$$d(i,j) = min \begin{cases} d(i-1,j) + del\_cost \\ d(i,j-1) + ins\_cost \\ d(i-1,j-1) + subs\_cost \end{cases}$$



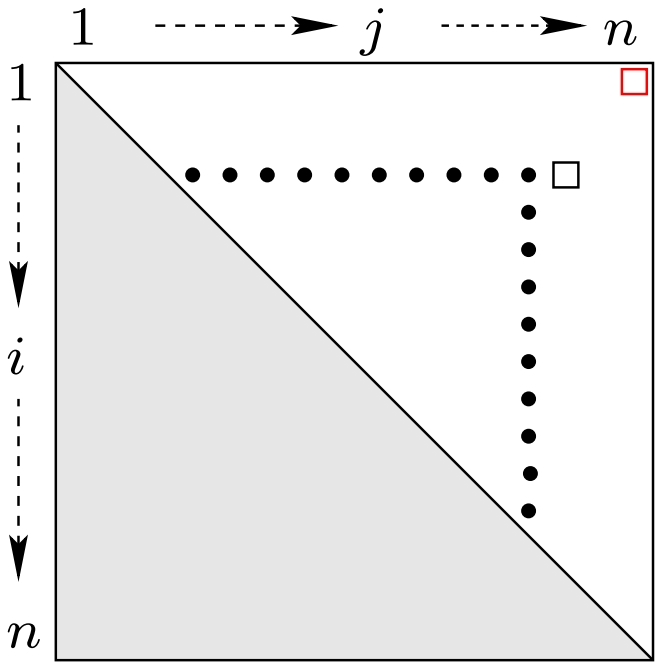
$[Levenshtein]$

$1 \leq i \leq j \leq n$

$space\ O(n^2)$

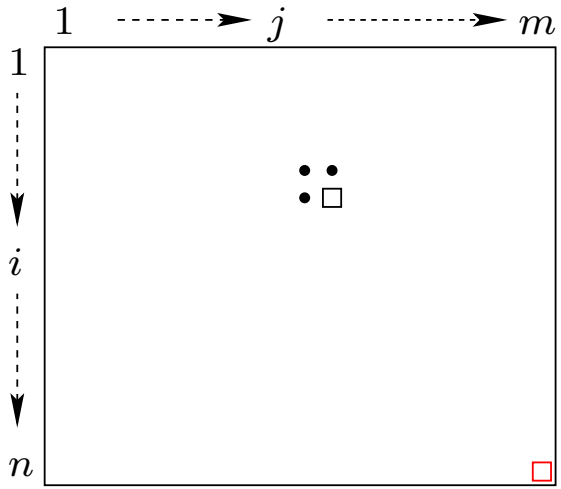
$time\ O(n^3)$

$$M(i,j) = max \begin{cases} M(i,j-1) \\ M(i,k-1) + M(k+1,j-1) + 1 \end{cases}$$

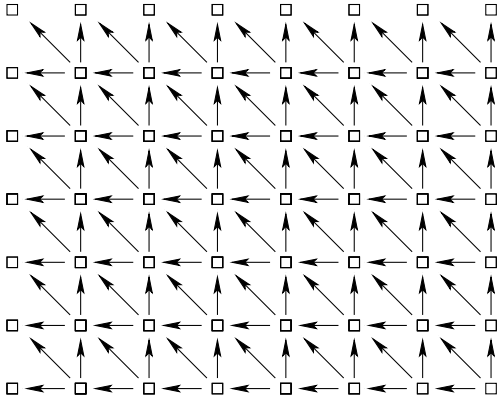


$[Nussinov]$

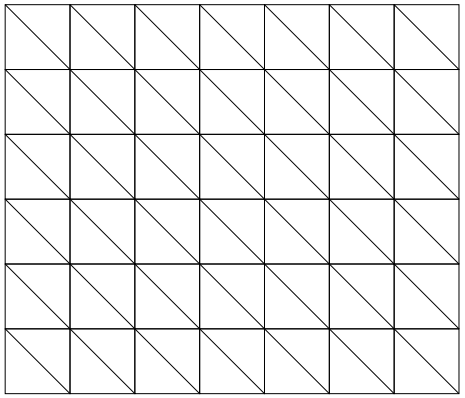
serial vs. non-serial DP



matrix



associated digraph



associated graph