Enumeration algorithms in graphs

Aurélie Lagoutte - GT Graphes
G-SCOP, Grenoble INP / Université Grenoble Alpes

Journées du GDR-IM – April 5, 2023, Paris

Enumeration : principle

Some problems need as an answer a **list** of solutions, instead of a **single** solution. For example :

Enumeration : principle

Some problems need as an answer a **list** of solutions, instead of a **single** solution. For example :

- Answer to a database query

```
$ select appellation, vignoble, type from AOC
Côte-Rôtie                 |  Vallée du Rhône  |  Rouge
Saint-Emilion              |  Bordeaux         |  Rouge
Saint-Nicolas-de-Bourgueil |  Val de Loire     |  Rouge
```

Enumeration : principle

Some problems need as an answer a **list** of solutions, instead of a **single** solution. For example :

- Answer to a database query

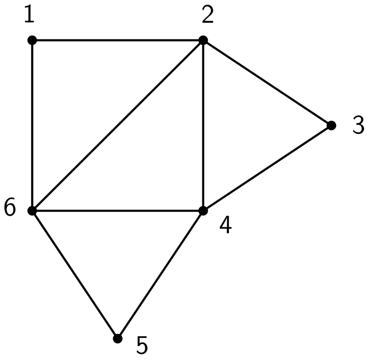
```
$ select appellation, vignoble, type from AOC
Côte-Rôtie                   |  Vallée du Rhône   |  Rouge
Saint-Emilion                |  Bordeaux          |  Rouge
Saint-Nicolas-de-Bourgueil   |  Val de Loire      |  Rouge
```

- Truth table : list all input combinations

e_1	e_2	e_3	S
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Enumeration : a typical example

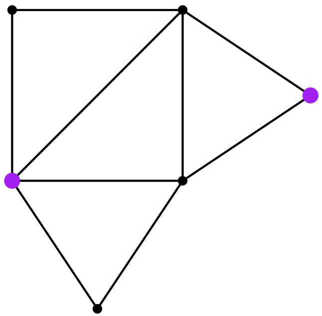
Input: Graph G
Output : The list of all **inclusion-wise maximal** stable sets of G



$\{1,3,5\},\{1,4\},\{2,5\},\{3,6\}$

Focus on easy problems

Input: Graph G
Output : one **inclusion-wise maximal** stable set. $\in$ P (greedy)



Not to be confused with :
Input: Graph G
Output : a stable set of maximum size

NP-complete

Enumerating in graphs : useful cases

- Graph databases : answer to a query
- Graph model is not exact : some solutions are *best* based on qualitative criteria, we have to examine them one by one
- Identify all problematic (or interesting !) patterns in a network

Application fields: bioinformatics (phylogenetic trees), chemistry (molecule structure), complex system modeling, databases...

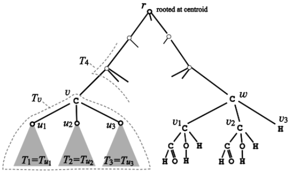


Diagram of a stereoisomer[1]

[1] Comparison and Enumeration of Chemical Graphs, T. Akutsu, H. Nagamochi, *Comp. and Struct. Biotechnology Journal*

Complexity for enumeration problems

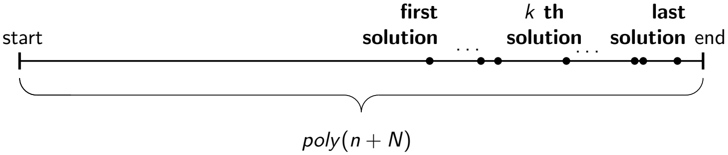
In most cases : exponential number of solutions to output
(ex: $3^{n/3}$ max. stable sets)
$\Rightarrow$ *Good complexity measure?*

Complexity for enumeration problems

In most cases : exponential number of solutions to output
(ex: $3^{n/3}$ max. stable sets)
$\Rightarrow$ *Good complexity measure?*

1. Output-polynomial

Input of size n, N solutions to output.

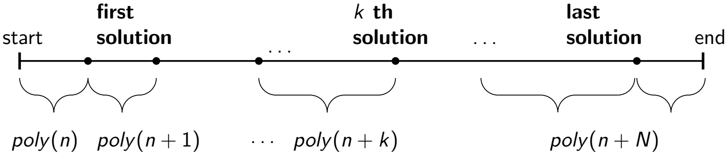


Complexity for enumeration problems

In most cases : exponential number of solutions to output
(ex: $3^{n/3}$ max. stable sets)
$\Rightarrow$ *Good complexity measure?*

1. Output-polynomial
2. Incremental polynomial

Input of size n, N solutions to output.

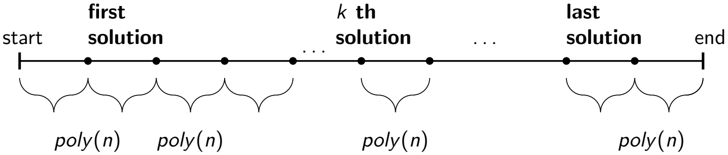


Complexity for enumeration problems

In most cases : exponential number of solutions to output
(ex: $3^{n/3}$ max. stable sets)
$\Rightarrow$ *Good complexity measure?*

1. Output-polynomial
2. Incremental polynomial
3. Polynomial delay

Input of size n, N solutions to output.



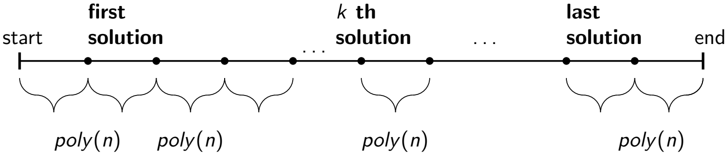
Complexity for enumeration problems

In most cases : exponential number of solutions to output
(ex: $3^{n/3}$ max. stable sets)
$\Rightarrow$ *Good complexity measure?*

1. Output-polynomial
2. Incremental polynomial
3. Polynomial delay

poly space vs.
exponential space

Input of size n, N solutions to output.



Interesting objects to enumerate

1. Inclusion-wise minimal transversal of a hypergraph

Interesting objects to enumerate

1. Inclusion-wise minimal transversal of a hypergraph
2. Inclusion-wise minimal dominating sets

Interesting objects to enumerate

1. Inclusion-wise minimal transversal of a hypergraph
2. Inclusion-wise minimal dominating sets
3. Spanning trees

Interesting objects to enumerate

1. Inclusion-wise minimal transversal of a hypergraph
2. Inclusion-wise minimal dominating sets
3. Spanning trees
4. "Structured patterns" : inclusion-wise max. stable sets or cliques ...

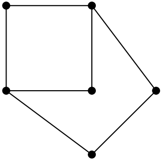
Interesting objects to enumerate

1. Inclusion-wise minimal transversal of a hypergraph
2. Inclusion-wise minimal dominating sets
3. Spanning trees
4. "Structured patterns" : inclusion-wise max. stable sets or cliques ...
5. Inclusion-wise minimal "Π-fixings" of a graph
   $\rightarrow$ Completions, deletion, induced subgraphs of a graph ...
   ... satisfying a given property Π

Minimal fixings

3 variants

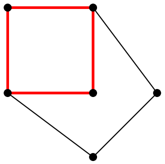
We want to satisfy a given property Π
Example : $\Pi = C_4$-free
(contains no induced C_4)



Minimal fixings

3 variants

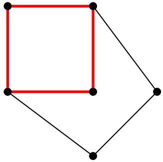
We want to satisfy a given property Π
Example : $\Pi = C_4$-free
(contains no induced C_4)



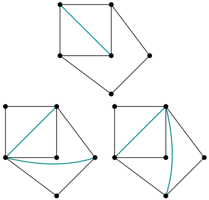
Minimal fixings

3 variants

We want to satisfy a given property Π
Example : $\Pi = C_4$-free
(contains no induced C_4)



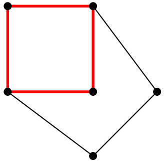
Fixing by
adding edges
Min. Π-completion



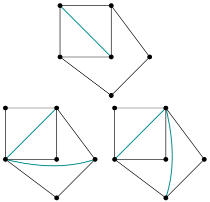
Minimal fixings

3 variants

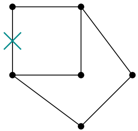
We want to satisfy a given property Π
Example : $\Pi = C_4$-free
(contains no induced C_4)



Fixing by adding edges
Min. Π-completion



Fixing by deleting edges
Min. Π-deletion

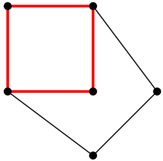


$+$ 3 others

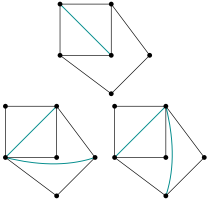
Minimal fixings

3 variants

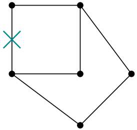
We want to satisfy a given property Π
Example : $\Pi = C_4$-free
(contains no induced C_4)



Fixing by
adding edges
Min. Π-completion

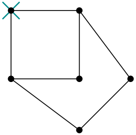


Fixing by
deleting edges
Min. Π-deletion



+ 3 others

Fixing by
deleting vertices
Max. Π-induced subgraph

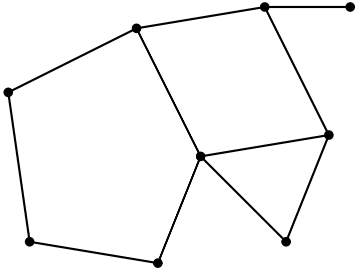


+ 3 others

Chordal completion

Chordal completion of a graph G : a completion of G that is chordal (no chordless cycle of length ≥ 4).

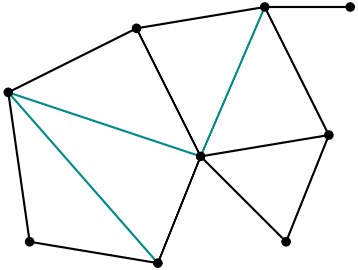
A chordal completion of G is also called a *triangulation* of G or sometimes a *fill-in*.



Chordal completion

Chordal completion of a graph G : a completion of G that is chordal (no chordless cycle of length ≥ 4).

A chordal completion of G is also called a *triangulation* of G or sometimes a *fill-in*.

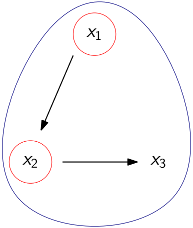


Link with database query evaluation

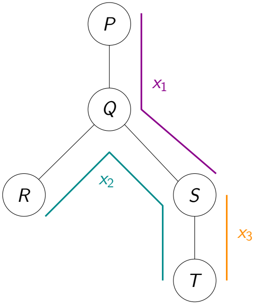
φ : a conjunctive query seen as a First-Order formula with only existential quantifyer and $\wedge$ operator

φ-EVAL
Input: A database D
Output : All tuples from D satisfying φ

Example : $\varphi = \exists x_2 \; P(x_1) \wedge R(x_2) \wedge Q(x_1, x_2) \wedge T(x_2, x_3) \wedge S(x_1, x_2, x_3)$



Hypergraph repr. of φ



Join tree of φ

Algorithmic methods to enumerate

Not-to-be-missed results

Theorem (Courcelle, 2009)

For every monadic second-order formula $\varphi(X1, \ldots, Xp)$, there exists an algorithm that takes as input a graph G of treewidth at most k and that enumerates the set of p-tuples satisfying φ in G:

- *after a preprocessing using time $\mathcal{O}(n \log n)$, where $n = |V(G)|$,*
- *with linear delay*

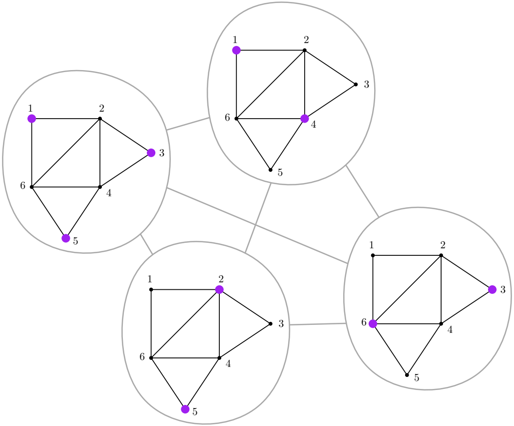
Theorem (Eiter, Gottlob, 1995)

There exists an incremental-polynomial algorithm that, given in input a hypergraph H with bounded-size hyperedges, enumerates all minimal transversals of H.

General principle of an enumeration algorithm

- Metagraph of solutions : traversal of this metagraph

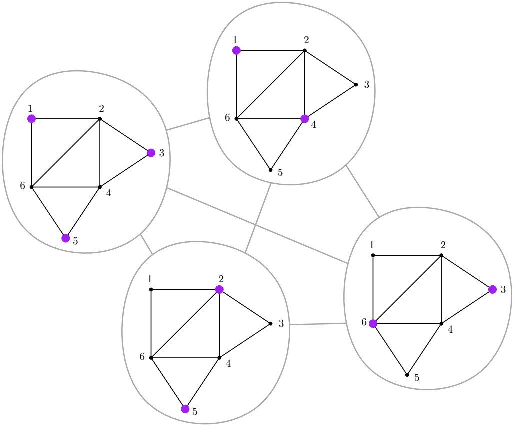
Example with
maximal stable sets
Solution metagraph



General principle of an enumeration algorithm

- Metagraph of solutions : traversal of this metagraph
- outputting each solution once

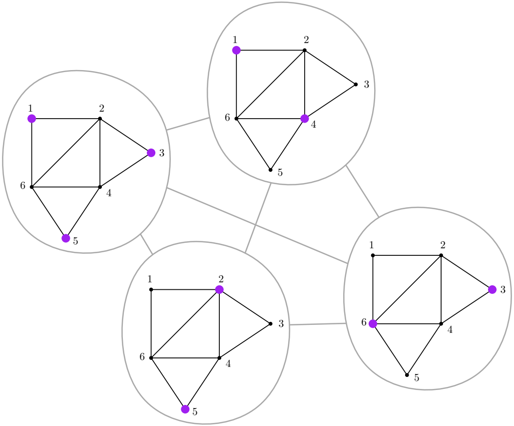
Example with
maximal stable sets
Solution metagraph



General principle of an enumeration algorithm

- Metagraph of solutions : traversal of this metagraph
- outputting each solution once
- and **only** once

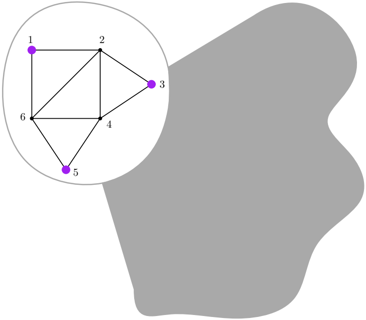
Example with
maximal stable sets
Solution metagraph



General principle of an enumeration algorithm

- Metagraph of solutions : traversal of this metagraph
- outputting each solution once
- and **only** once

Example with
maximal stable sets
Solution metagraph



Three classical methods :

Goal : get polynomial delay
+ poly space for 1 et 2 (and sometimes 3)

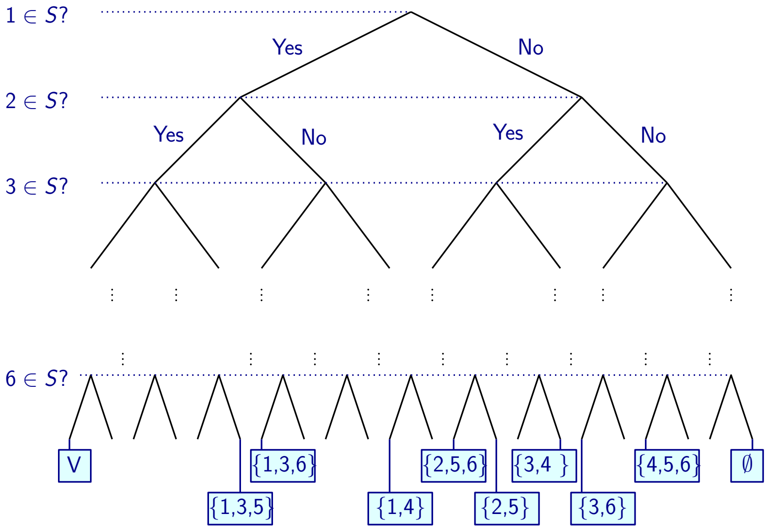
1. *Flashlight search* or *Binary partition*
   [Read, Tarjan '75]
2. *Reverse search*
   [Avis, Fukuda '96]
3. *Proximity Search*
   [Conte, Uno '19]
   [Conte, Grossi, Marino, Uno, Versari, '21]

Three classical methods :

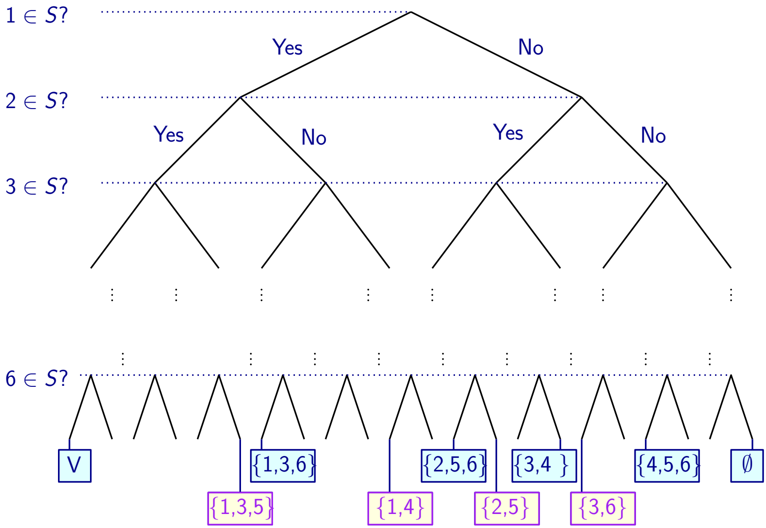
Goal : get polynomial delay
+ poly space for 1 et 2 (and sometimes 3)

1. ***Flashlight search*** or *Binary partition*
   [Read, Tarjan '75]
2. *Reverse search*
   [Avis, Fukuda '96]
3. *Proximity Search*
   [Conte, Uno '19]
   [Conte, Grossi, Marino, Uno, Versari, '21]

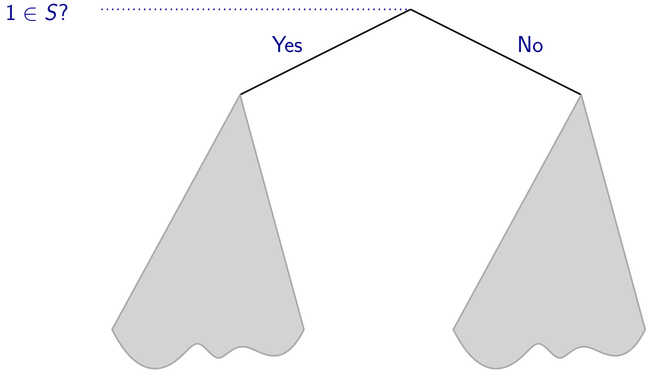
Flashlight search or *Binary partition*



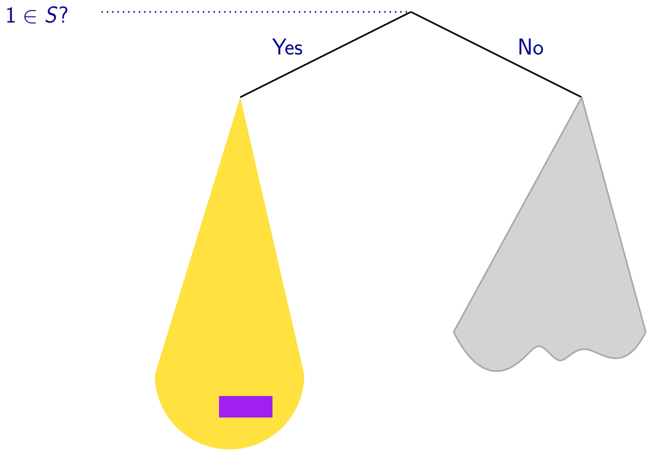
Flashlight search or *Binary partition*



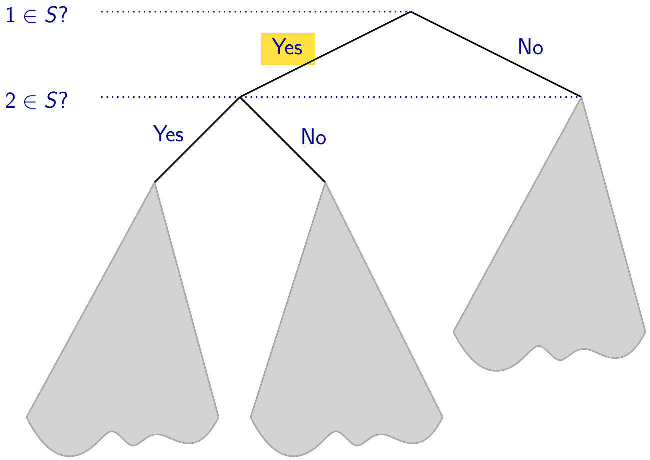
Flashlight search or *Binary partition*



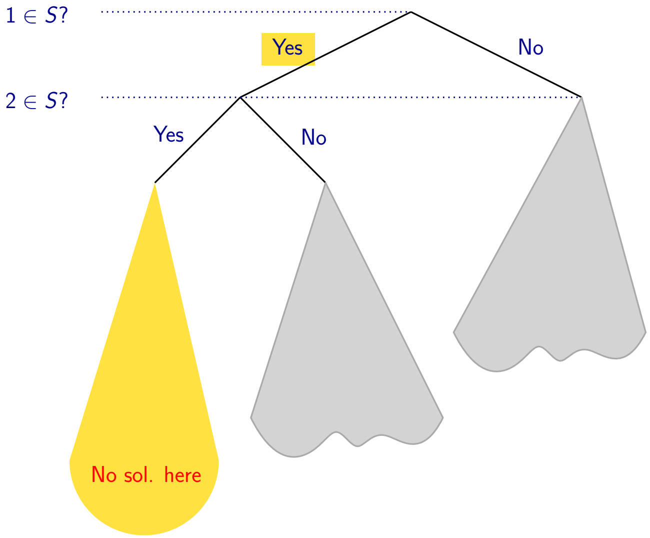
Flashlight search or *Binary partition*



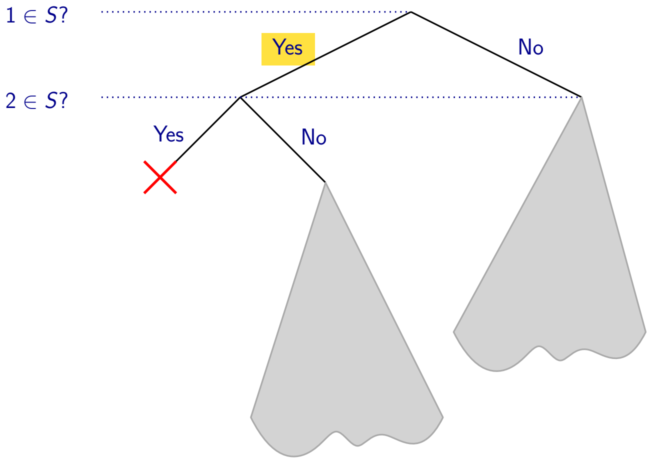
Flashlight search or *Binary partition*



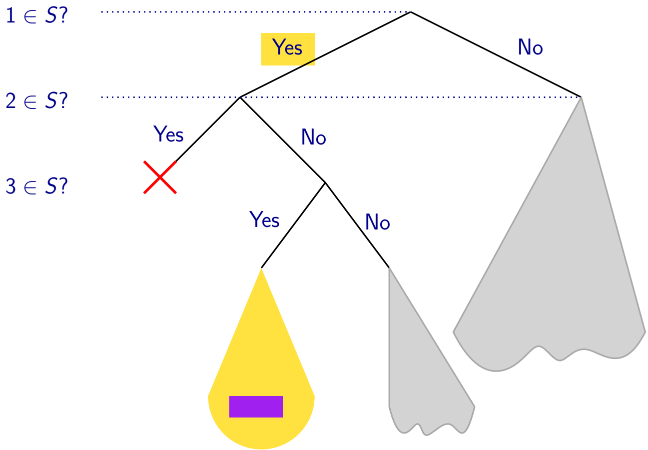
Flashlight search or *Binary partition*



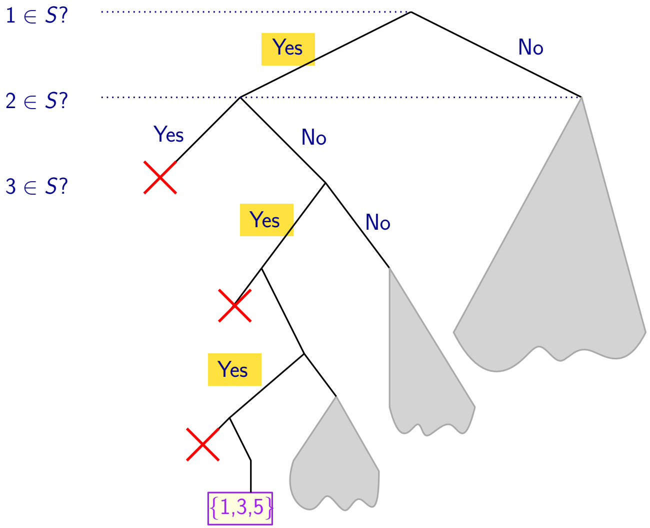
Flashlight search or *Binary partition*



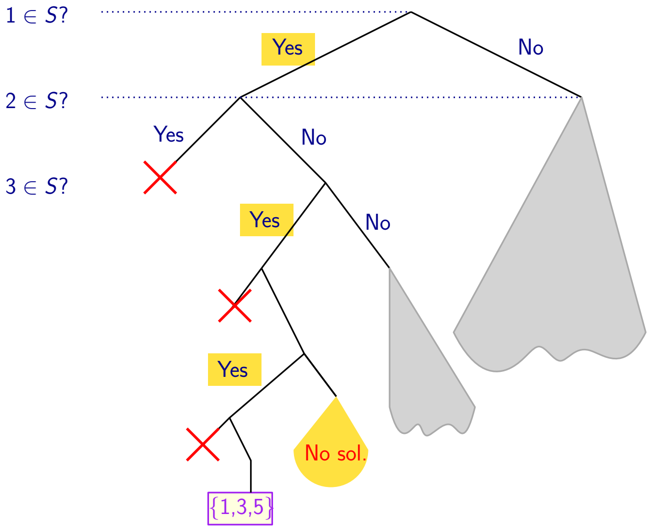
Flashlight search or *Binary partition*



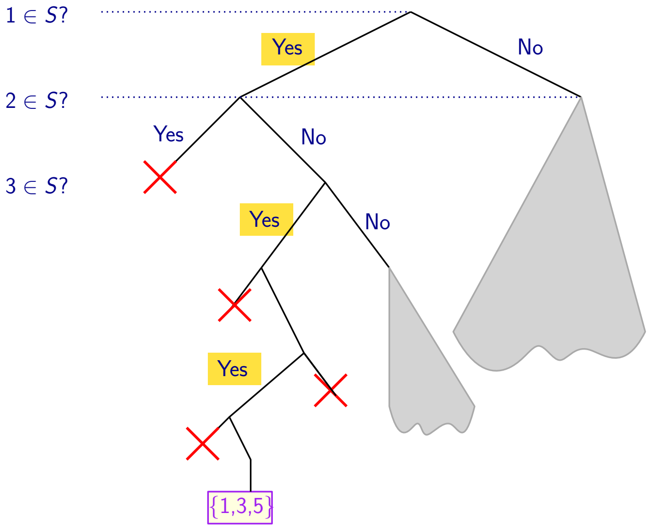
Flashlight search or *Binary partition*



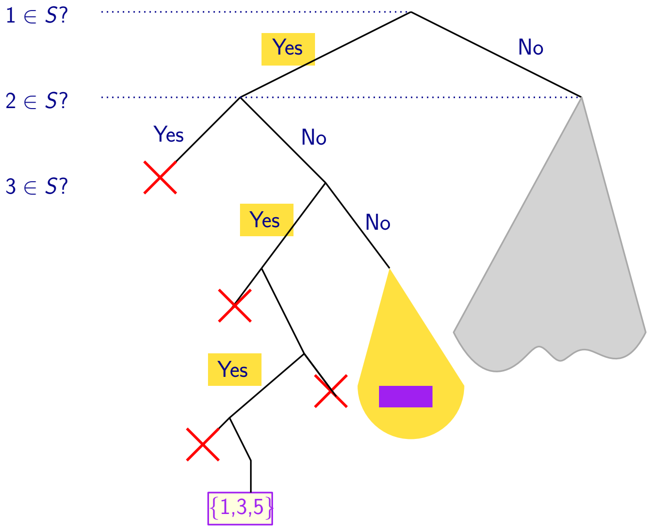
Flashlight search or *Binary partition*



Flashlight search or *Binary partition*



Flashlight search or *Binary partition*

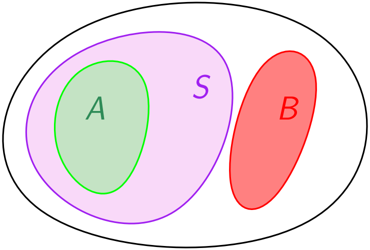


Flashlight search or *Binary partition*

For *Flashlight search* to run in poly delay (and space) :

Answer in poly time to the *extension problem* of $\mathcal{P}$:

Let A and B be two disjoint sets
is there a solution S of $\mathcal{P}$ such that $A \subseteq S$ and $S \cap B = \emptyset$?



All of A must be in solution
and all of B is forbidden in the solution.

Extension problem

A typical example : enumerate all models of a formula F in DNF.

Ex: $F = (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_3 \wedge x_4)$

Solution = variable assignment satisfying F.

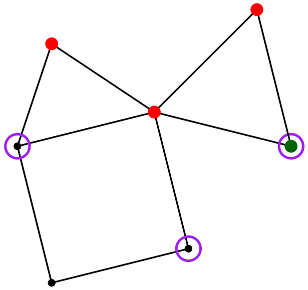
$A = \{x_1, x_3\} \Rightarrow x_1 = x_3 = \mathit{True}$
$B = \{x_2\} \Rightarrow x_2 = \mathit{False}$
Does there exist a model F containing A and disjoint from B ?
$F_{A,B} = (\mathit{False}) \ \vee \ (\mathit{False}) \ \vee (x_4)$

Extension problem : negative result

Assumptions : hereditary and "non-trivial" property Π
Pb: Π-fixings by deleting vertices

Input: Graph G, disjoint A, $B \subseteq V$
Output : Does there exist an induced subgraph of G :

- containing A,
- satisfying property Π under interest,
- inclusion-wise maximal,
- and avoiding B ?

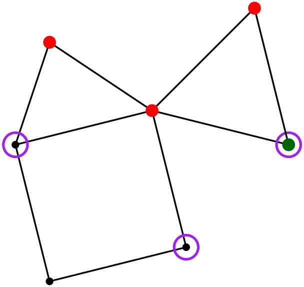


Extension problem : negative result

Assumptions : hereditary and "non-trivial" property Π
Pb: Π-fixings by deleting vertices

Input: Graph G, disjoint A, $B \subseteq V$
Output : Does there exist an induced subgraph of G :

- containing A,
- satisfying property Π under interest,
- inclusion-wise maximal,
- and avoiding B ?



Theorem [Brosse, L., Limouzy, Mary, Pastor – 2020+]

The extension problem for Π-induced subgraphs, for any hereditary non-trivial property Π, is NP-hard.

Three classical methods :

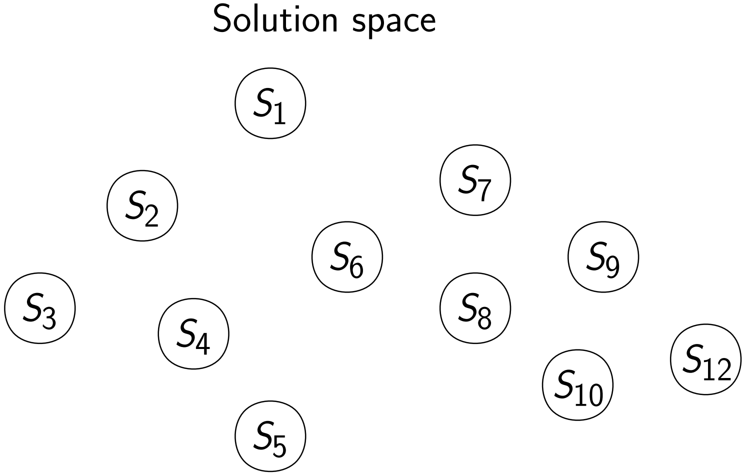
Goal : get polynomial delay
+ poly space for 1 et 2 (and sometimes 3)

1. *Flashlight search* or *Binary partition*
   [Read, Tarjan '75]

2. ***Reverse search***
   [Avis, Fukuda '96]

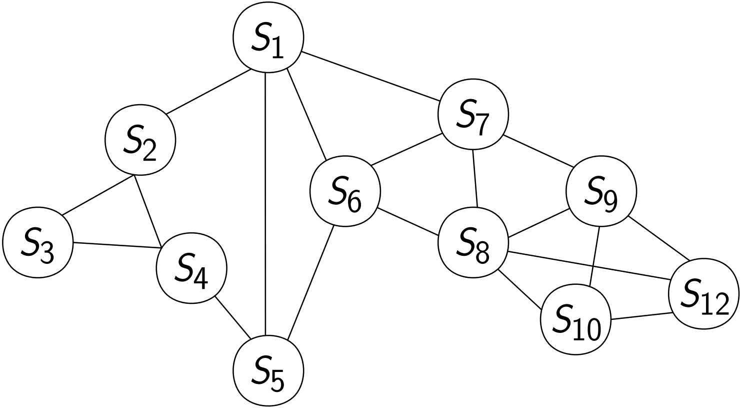
3. *Proximity Search*
   [Conte, Uno '19]
   [Conte, Grossi, Marino, Uno, Versari, '21]

Reverse search

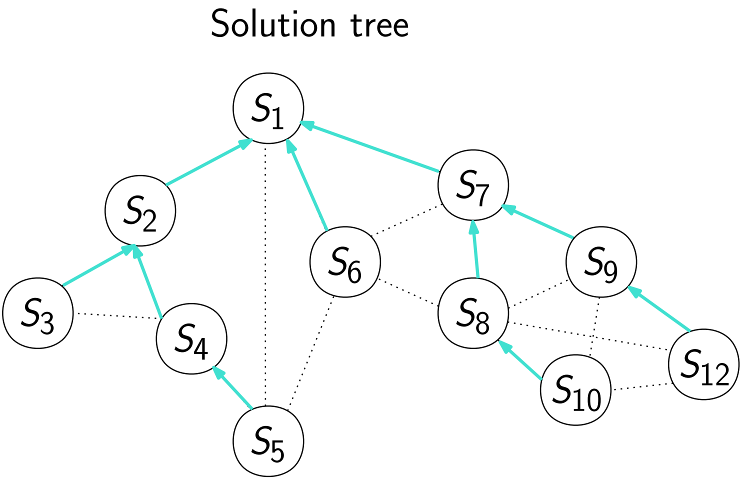


Reverse search

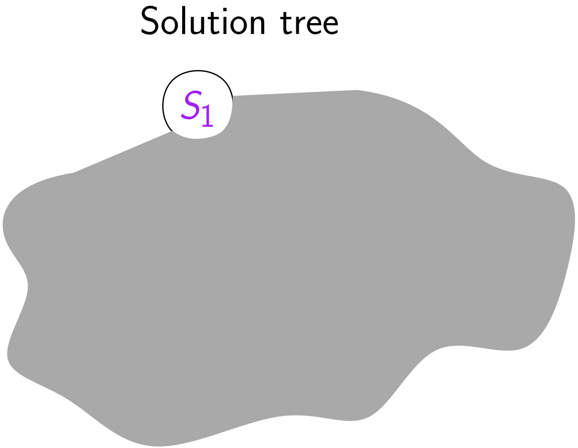
Solution metagraph



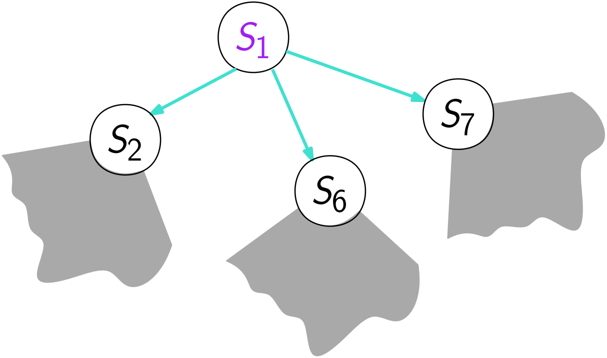
Reverse search



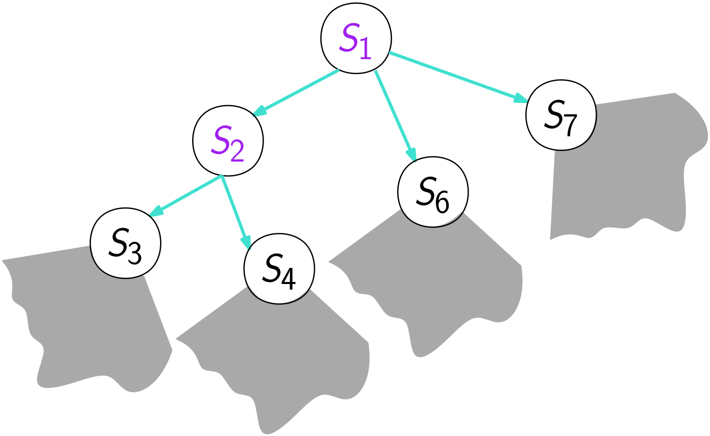
Reverse search



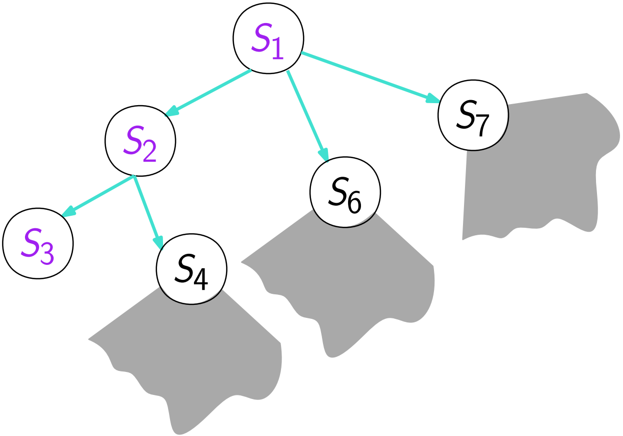
Reverse search



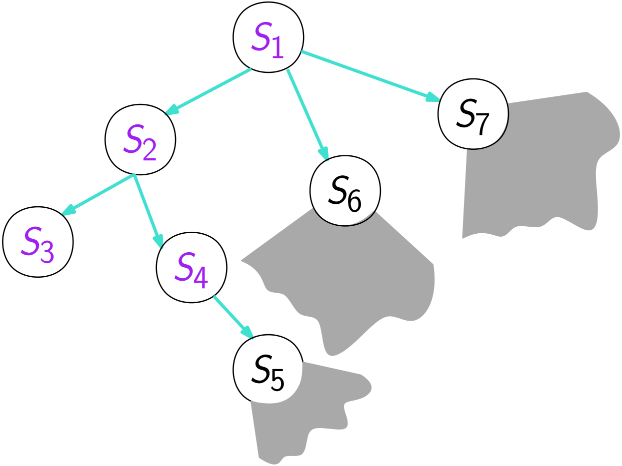
Reverse search



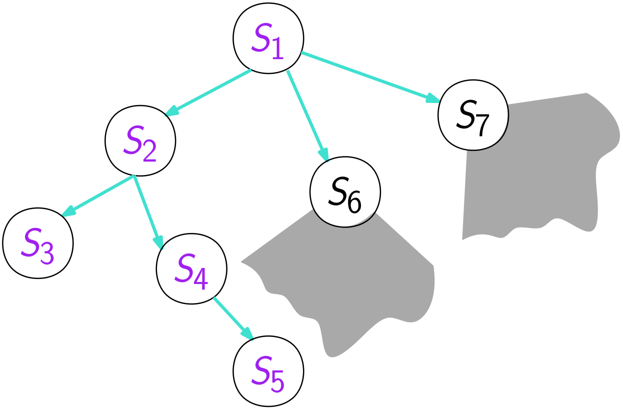
Reverse search



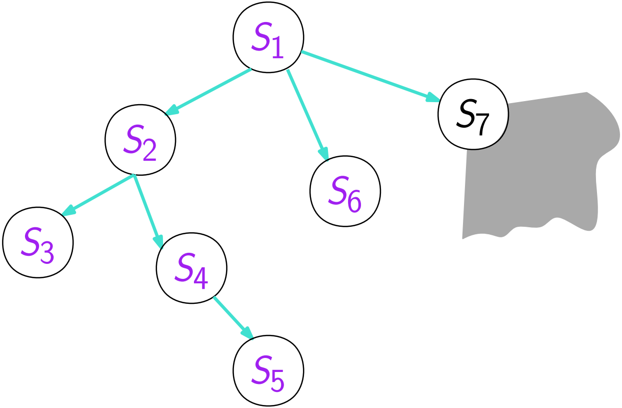
Reverse search



Reverse search



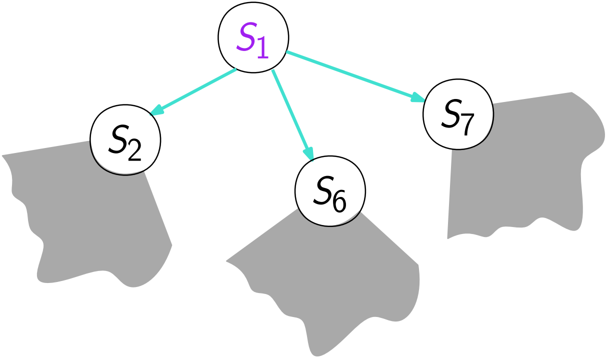
Reverse search



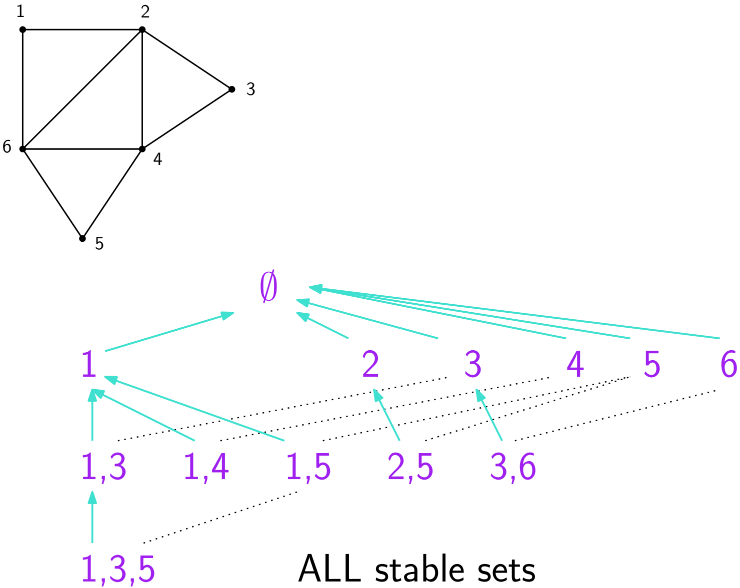
Reverse search

To have *Reverse search* run in poly delay and space :

Generate in poly time and space the *children* of a solution (each solution must have ***a single*** father)

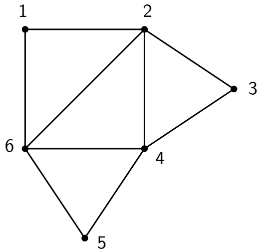


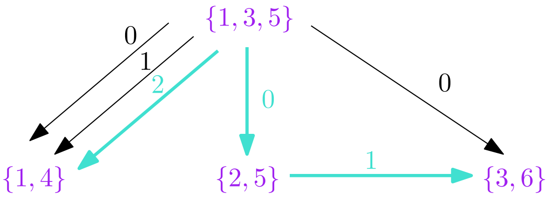
Reverse Search



ALL stable sets

Reverse Search for Maximal Stable Sets





$S \xrightarrow{i} S'$ if $S \cap \{v_1, \ldots, v_i\} = S' \cap \{v_1, \ldots, v_i\}$ and S is the lexicographically smallest among all solutions containing $S' \cap \{v_1, \ldots, v_i\}$

P is the father of S if $P \xrightarrow{i} S$ and there is no arc to S indexed $> i$

Three classical methods :

Goal : get polynomial delay
+ poly space for 1 et 2 (and sometimes 3)

1. *Flashlight search* or *Binary partition*
   [Read, Tarjan '75]
2. *Reverse search*
   [Avis, Fukuda '96]
3. ***Proximity Search***
   [Conte, Uno '19]
   [Conte, Grossi, Marino, Uno, Versari, '21]

Proximity Search

Designed for enumerating *inclusion-wise maximal* (or min.) solutions to a problem.

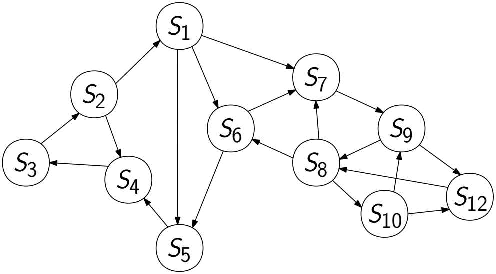
Proximity Search

Designed for enumerating *inclusion-wise maximal* (or min.) solutions to a problem.

Recursive depth-first search (rather classical) with conditions :

- generate **neighbors** of a vertex in poly time
  $\rightarrow$ *poly degree*

Solution metagraph



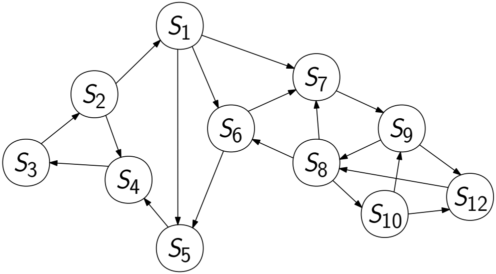
Proximity Search

Designed for enumerating *inclusion-wise maximal* (or min.) solutions to a problem.

Recursive depth-first search (rather classical) with conditions :

- generate **neighbors** of a vertex in poly time
  → *poly degree*
- check if a vertex has already been visited
  → *exponential space :-(*

Solution metagraph



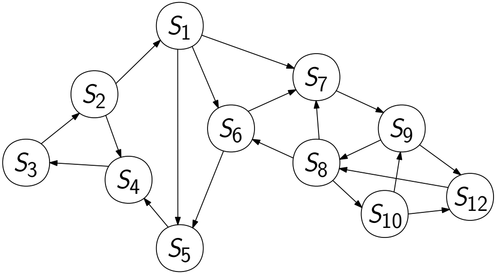
Proximity Search

Designed for enumerating *inclusion-wise maximal* (or min.) solutions to a problem.

Recursive depth-first search (rather classical) with conditions :

- generate **neighbors** of a vertex in poly time
  $\rightarrow$ *poly degree*
- check if a vertex has already been visited
  $\rightarrow$ *exponential space :-(*
- strong connectedness of the graph to be proven thanks to a notion of **proximity**

Solution metagraph



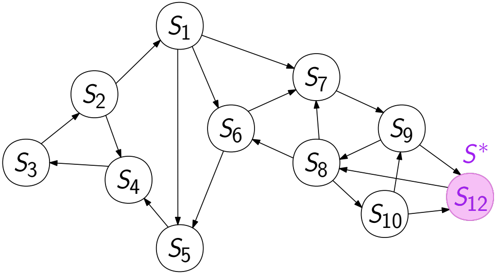
Proximity Search

Designed for enumerating *inclusion-wise maximal* (or min.) solutions to a problem.

Recursive depth-first search (rather classical) with conditions :

- generate **neighbors** of a vertex in poly time
  $\rightarrow$ *poly degree*
- check if a vertex has already been visited
  $\rightarrow$ *exponential space :-(*
- strong connectedness of the graph to be proven thanks to a notion of **proximity**

Solution metagraph



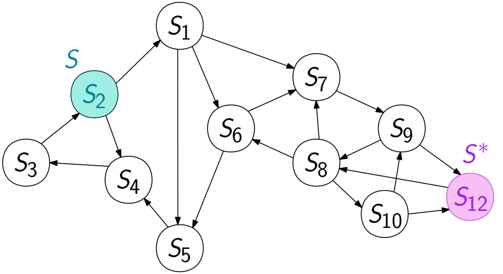
Proximity Search

Designed for enumerating *inclusion-wise maximal* (or min.) solutions to a problem.

Recursive depth-first search (rather classical) with conditions :

- generate **neighbors** of a vertex in poly time
  $\rightarrow$ *poly degree*
- check if a vertex has already been visited
  $\rightarrow$ *exponential space :-(*
- strong connectedness of the graph to be proven thanks to a notion of **proximity**

Solution metagraph



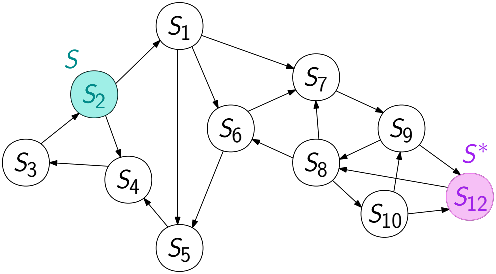
Proximity Search

Designed for enumerating *inclusion-wise maximal* (or min.) solutions to a problem.

Recursive depth-first search (rather classical) with conditions :

- generate **neighbors** of a vertex in poly time
  $\rightarrow$ *poly degree*
- check if a vertex has already been visited
  $\rightarrow$ *exponential space :-(*
- strong connectedness of the graph to be proven thanks to a notion of **proximity**

Solution metagraph



$|S \tilde{\cap} S^*| = 7$

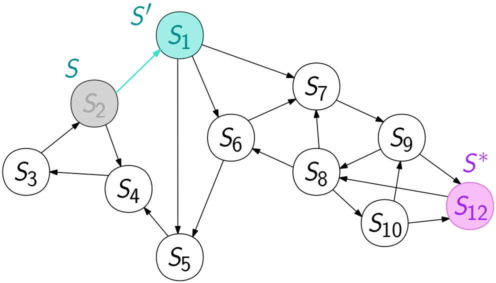
Proximity Search

Designed for enumerating *inclusion-wise maximal* (or min.) solutions to a problem.

Recursive depth-first search (rather classical) with conditions :

- generate **neighbors** of a vertex in poly time
  → *poly degree*
- check if a vertex has already been visited
  → *exponential space :-(*
- strong connectedness of the graph to be proven thanks to a notion of **proximity**

Solution metagraph



$|S' \tilde{\cap} S^*| = 10$

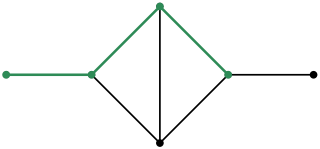
Three classical methods :

Goal : get polynomial delay
+ poly space for 1 et 2 (and sometimes 3)

1. *Flashlight search* or *Binary partition*
   [Read, Tarjan '75]
2. *Reverse search*
   [Avis, Fukuda '96]
3. *Proximity Search*
   [Conte, Uno '19]
   [Conte, Grossi, Marino, Uno, Versari, '21]

Cographs = P_4-free graphs

P_4



$G \notin Cographs$

The class of cographs is hereditary and auto-complementary.

Theorem

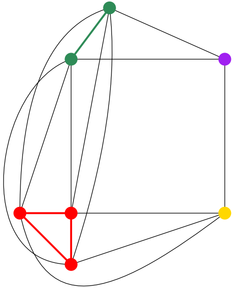
Via *Proximity Search*, there exists an algorithm for :
Input: a graph G
Output : all inclusion-wise maximal induced subgraphs of G that are cographs (cograph fixing by min. deletion of vertices) ;
running in complexity :

- polynomial delay ;
- exponential space.

Cographes & twins

Cographs admit a "twin ordering" on $V(G)$: G can always be obtained from a single vertex by adding a true or a false twin to a existing vertex.

True twins : $N[x] = N[y]$ False twins $N(x) = N(y)$



Cographes & twins

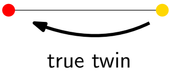
Cographs admit a "twin ordering" on $V(G)$: G can always be obtained from a single vertex by adding a true or a false twin to a existing vertex.

True twins : $N[x] = N[y]$ False twins $N(x) = N(y)$

Cographes & twins

Cographs admit a "twin ordering" on $V(G)$: G can always be obtained from a single vertex by adding a true or a false twin to a existing vertex.

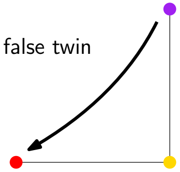
True twins : $N[x] = N[y]$ False twins $N(x) = N(y)$



Cographes & twins

Cographs admit a "twin ordering" on $V(G)$: G can always be obtained from a single vertex by adding a true or a false twin to a existing vertex.

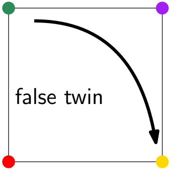
True twins : $N[x] = N[y]$ False twins $N(x) = N(y)$



Cographes & twins

Cographs admit a "twin ordering" on $V(G)$: G can always be obtained from a single vertex by adding a true or a false twin to a existing vertex.

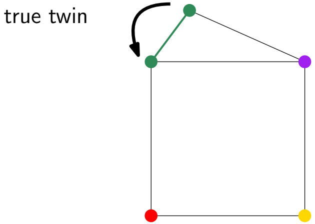
True twins : $N[x] = N[y]$ False twins $N(x) = N(y)$



Cographes & twins

Cographs admit a "twin ordering" on $V(G)$: G can always be obtained from a single vertex by adding a true or a false twin to a existing vertex.

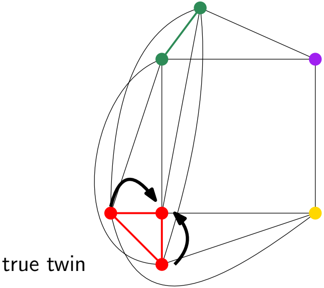
True twins : $N[x] = N[y]$ False twins $N(x) = N(y)$



Cographes & twins

Cographs admit a "twin ordering" on $V(G)$: G can always be obtained from a single vertex by adding a true or a false twin to a existing vertex.

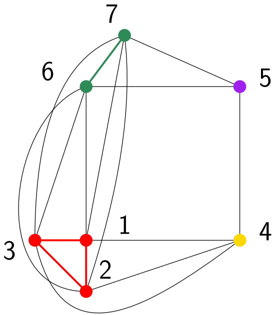
True twins : $N[x] = N[y]$ False twins $N(x) = N(y)$



Cographes & twins

Cographs admit a "twin ordering" on $V(G)$: G can always be obtained from a single vertex by adding a true or a false twin to a existing vertex.

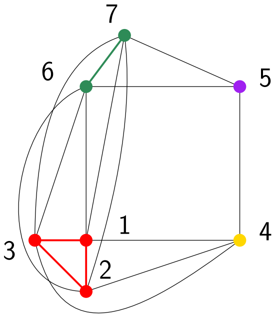
True twins : $N[x] = N[y]$ False twins $N(x) = N(y)$



Cographes & twins

Cographs admit a "twin ordering" on $V(G)$: G can always be obtained from a single vertex by adding a true or a false twin to a existing vertex.

True twins : $N[x] = N[y]$ False twins $N(x) = N(y)$

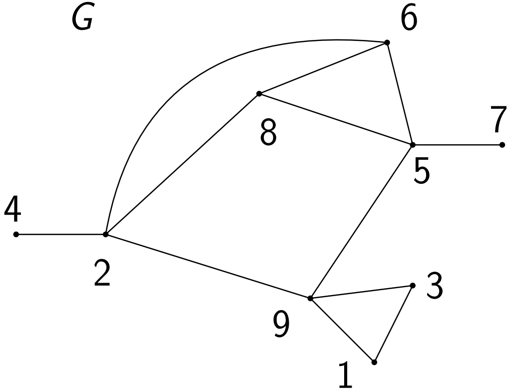


1 4 5 6 2 3 7

Canonical twin ordering

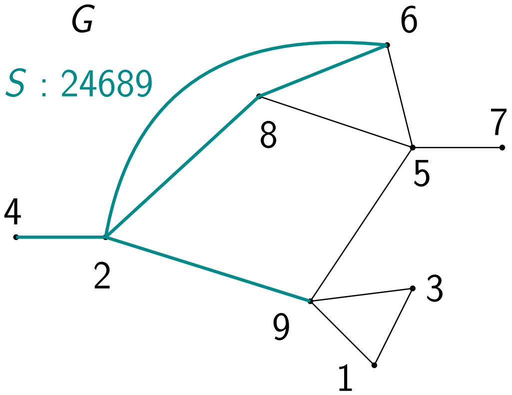
Proximity between two subcographs

Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



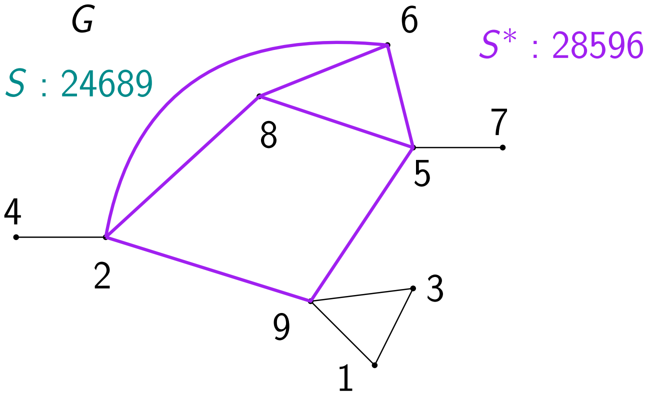
Proximity between two subcographs

Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



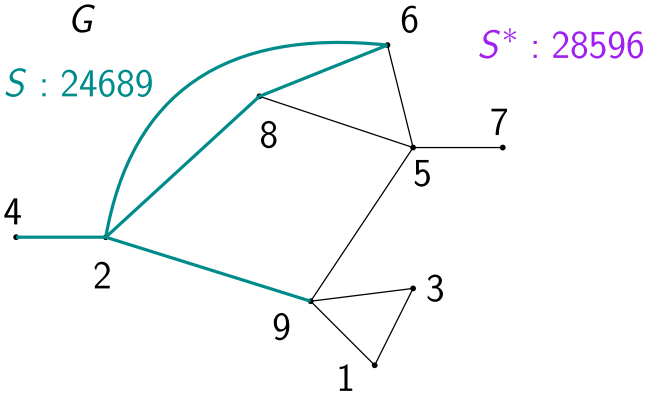
Proximity between two subcographs

Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



Proximity between two subcographs

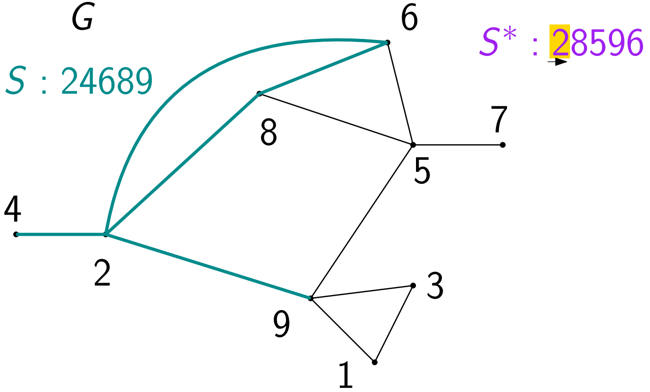
Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



$S \tilde{\cap} S^* = \{ \quad \}$

Proximity between two subcographs

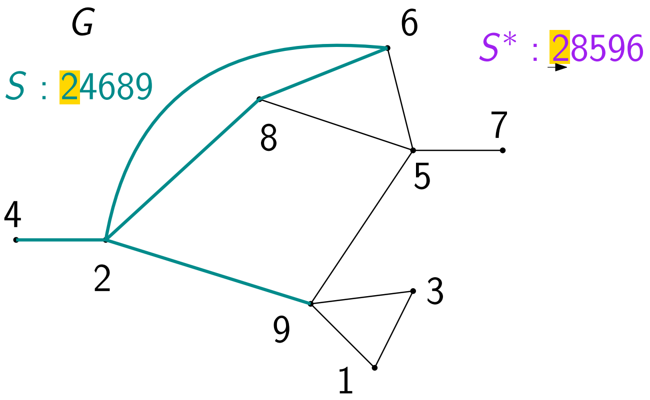
Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



$S \tilde{\cap} S^* = \{ \quad \}$

Proximity between two subcographs

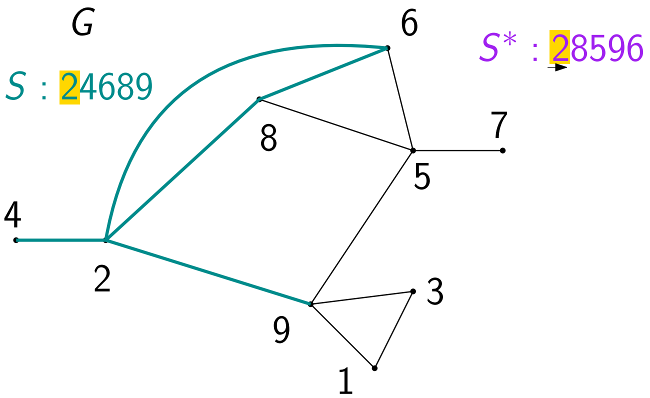
Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



$S \tilde{\cap} S^* = \{ \quad \}$

Proximity between two subcographs

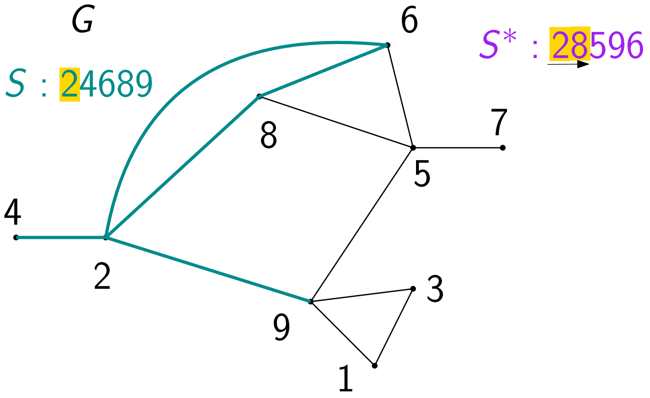
Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



$S \tilde{\cap} S^* = \{\, 2 \quad \}$

Proximity between two subcographs

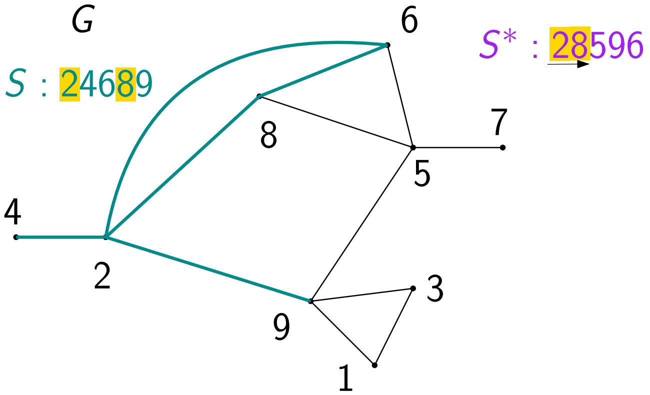
Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



$S \tilde{\cap} S^* = \{\, 2 \quad \}$

Proximity between two subcographs

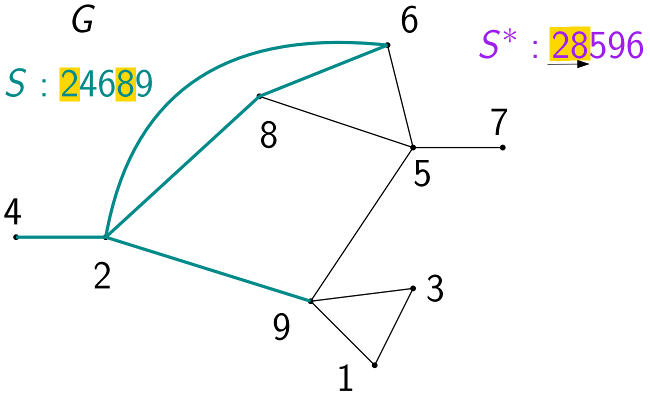
Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



$S \tilde{\cap} S^* = \{\, 2 \quad\quad\quad \}$

Proximity between two subcographs

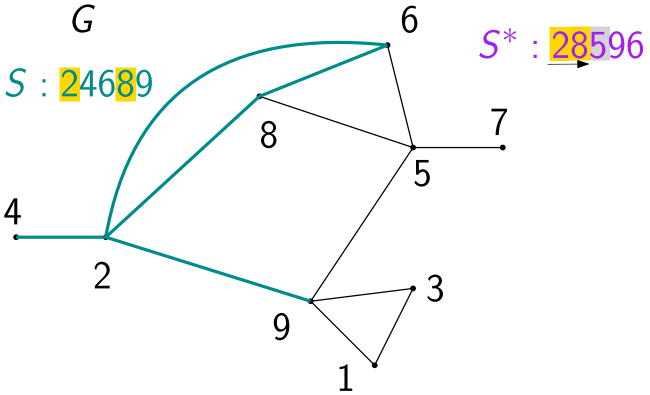
Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



$S \tilde{\cap} S^* = \{\, 2 \,,\, 8 \quad \}$

Proximity between two subcographs

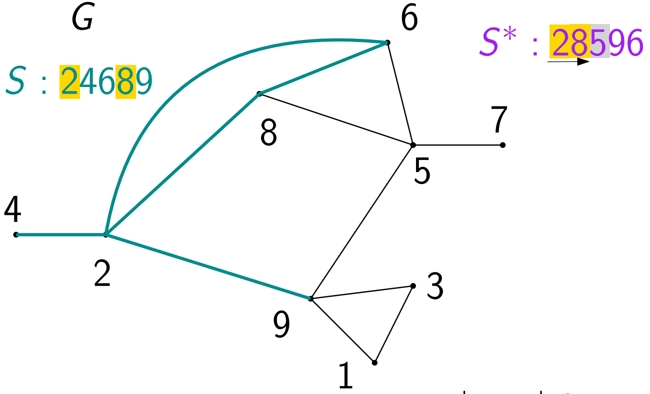
Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



$S \tilde{\cap} S^* = \{\, 2 \,,\, 8 \quad \}$

Proximity between two subcographs

Goal : enumerating all inclusion-wise maximal induced subgraphs of G that are cographs with *Proximity Search*



$$S \tilde{\cap} S^* = \{\, 2\, ,\, 8 \quad \} \rightarrow |\text{prox}|\ 2$$

Neighbors in the solution metagraph

S : 24689 S^* : 28596

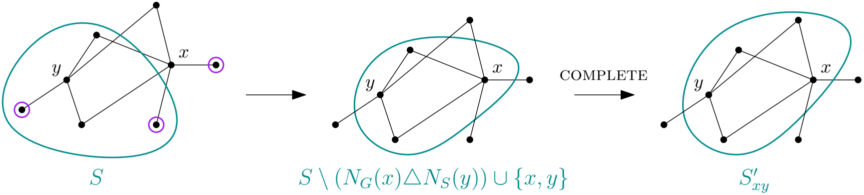
S': keeping 2 and 8, "force" 5 to enter as a (false) twin of 2

Neighbors in the solution metagraph

S : 24689 $\qquad\qquad$ S^* : 28596

S': keeping 2 and 8, "force" 5 to enter as a (false) twin of 2

Force x to enter in the solution as a (false) twin of y:

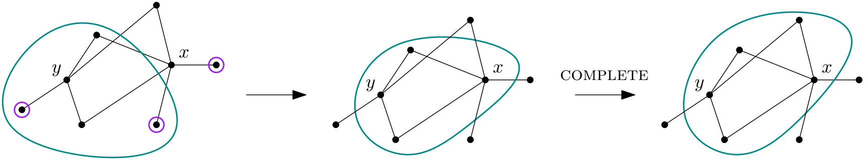


Neighbors in the solution metagraph

S : 24689 S^* : 28596

S': keeping 2 and 8, "force" 5 to enter as a (false) twin of 2

Force x to enter in the solution as a (false) twin of y:



S $S \setminus (N_G(x) \triangle N_S(y)) \cup \{x, y\}$ S'_{xy}

$$Neighbors(S) = \bigcup_{y \in S, x \notin S} S'_{xy}$$

Cograph fixings : recap

We have all the ingredients for Proximity Search to work :

1. Proximity between two solutions S and S^* is defined
2. $Neighbors(S)$ is computable in polytime
3. Lemma proving that we can always find a neighbor with higher proximity with a target solution, thanks to the twin ordering (not shown here)

$\rightarrow$ Enumeration of cograph fixings by inclusion-wise min. deletion of vertices in polynomial delay

Results

[Brosse, L., Limouzy, Mary, Pastor – 2020+]

Prop. Π: being a....	Max. induced subgraphs	Min. deletions	Min. completions
split graph	Poly delay & space [Cao'20$^+$]	Poly delay & space	Poly delay & space *via* autocompl.
cograph	poly delay *via* *Proxi. Search*	Open	Open
threshold graph	Poly delay & space [Cao'20$^+$]	poly delay *via* *Proxi. Search*	poly. delay *via* autocompl.

Results from [Cao20$^+$] are mostly using another method, called *restricted problem*.

(and there are more than displayed in this array).

More

→ Enumerating minimal triangulations of a graph ?

More

→ Enumerating minimal triangulations of a graph ?

Theorem [Brosse, Limouzy, Mary, 2021^{+}]

There exists a polynomial delay polynomial space algorithm to enumerate all inclusion-wise minimal chordal completion of a graph G given in input.

→ *Proximity Search* with careful arguments (to get polynomial space in particular)

More

$\rightarrow$ Enumerating minimal triangulations of a graph ?

Theorem [Brosse, Limouzy, Mary, 2021^{+}]

There exists a polynomial delay polynomial space algorithm to enumerate all inclusion-wise minimal chordal completion of a graph G given in input.

$\rightarrow$ *Proximity Search* with careful arguments (to get polynomial space in particular)

$\rightarrow$ Enumerating all minimal "fixings" of a graph into a Π-graph for any hereditary Π?