Coloring perfect graphs with bounded clique number

JGA 2016

Aurélie Lagoutte

Joint work with M. Chudnovsky, P. Seymour and S. Spirkl

G-SCOP, Univ. Grenoble Alpes

November 17, 2016
Paris Dauphine

What's in my thesis?

Interactions between Cliques and Stable sets in a graph

What's in my thesis?

Coloration
Partition into stable sets; relate nb of parts to size of max. clique

Interactions between Cliques and Stable sets in a graph

What's in my thesis?

Interactions between Cliques and Stable sets in a graph

Coloration
Partition into stable sets; relate nb of parts to size of max. clique

Erdos-Hajnal conjecture
In H-free graphs, there always exists a big clique or a big stable set.

What's in my thesis?

Interactions between Cliques and Stable sets in a graph

- **Coloration**
  Partition into stable sets; relate nb of parts to size of max. clique
- **Erdos-Hajnal conjecture**
  In H-free graphs, there always exists a big clique or a big stable set.
- **Clique-Stable set Separation**
  (see next slide)

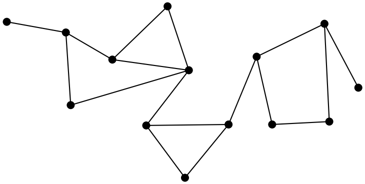
What's in my thesis?

Interactions between Cliques and Stable sets in a graph

- **Coloration**
  Partition into stable sets; relate nb of parts to size of max. clique
- **Erdos-Hajnal conjecture**
  In H-free graphs, there always exists a big clique or a big stable set.
- **Clique-Stable set Separation**
  (see next slide)
- **Pbs around CS-Separation**
  PL for Max. Weighted Stable Set, Constraint Satisfaction Pbs (CSP), Alon-Saks-Seymour conjecture

Clique-Stable set Separation

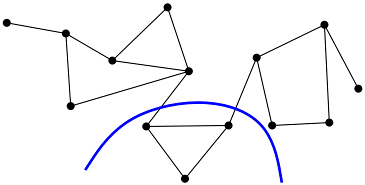
Game on a graph G:



Clique-Stable set Separation

Game on a graph G:

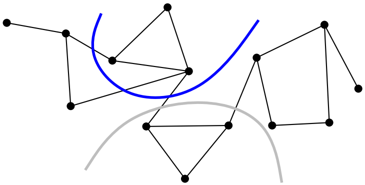
- Pre-processing step: choose some cuts of G.



Clique-Stable set Separation

Game on a graph G:

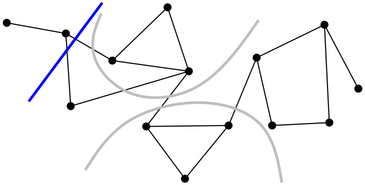
- Pre-processing step: choose some cuts of G.



Clique-Stable set Separation

Game on a graph G:

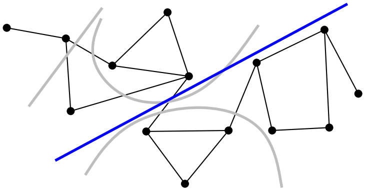
- Pre-processing step: choose some cuts of G.



Clique-Stable set Separation

Game on a graph G:

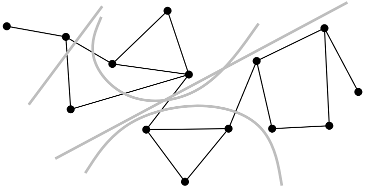
- Pre-processing step: choose some cuts of G.



Clique-Stable set Separation

Game on a graph G:

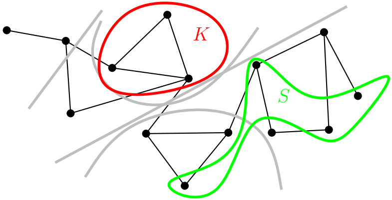
- Pre-processing step: choose some cuts of G.



Clique-Stable set Separation

Game on a graph G:

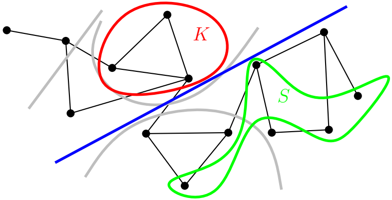
- Pre-processing step: choose some cuts of G.
- An Adversary chooses a clique K and a stable set S that do not intersect.



Clique-Stable set Separation

Game on a graph G:

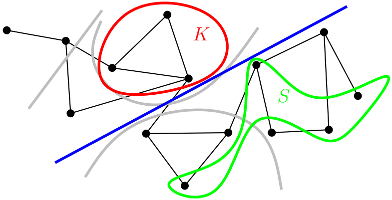
- Pre-processing step: choose some cuts of G.
- An Adversary chooses a clique K and a stable set S that do not intersect.
- I win if I have a cut **separating** K and S
  (=certificate of non-intersection between K and S)



Clique-Stable set Separation

Game on a graph G:

- Pre-processing step: choose some cuts of G.
- An Adversary chooses a clique K and a stable set S that do not intersect.
- I win if I have a cut **separating** K and S
  (=certificate of non-intersection between K and S)

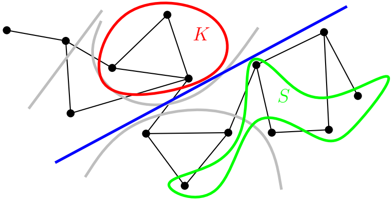


A **CS-Separator** is a family of cuts that ensures me to always win.

Clique-Stable set Separation

Game on a graph G:

- Pre-processing step: choose some cuts of G.
- An Adversary chooses a clique K and a stable set S that do not intersect.
- I win if I have a cut **separating** K and S
  (=certificate of non-intersection between K and S)



A **CS-Separator** is a family of cuts that ensures me to always win.
→ I am allowed to select only **polynomially many** cuts.

Bounds

Yannakakis (1991)

Upper Bound: $\forall G$ there exists a CS-separator of size $\mathcal{O}(n^{\log n})$.

Question: Do perfect graphs admit polynomial-size CS-Separator?

Bounds

Yannakakis (1991)

Upper Bound: $\forall G$ there exists a CS-separator of size $\mathcal{O}(n^{\log n})$.

Question: Do perfect graphs admit polynomial-size CS-Separator?

Huang, Sudakov (2012)

Lower bound : $\Omega(n^{\frac{6}{5}})$ cuts are needed for some graphs

Bounds

Yannakakis (1991)

Upper Bound: $\forall G$ there exists a CS-separator of size $\mathcal{O}(n^{\log n})$.

Question: Do perfect graphs admit polynomial-size CS-Separator?

Huang, Sudakov (2012)

Lower bound : $\Omega(n^{\frac{6}{5}})$ cuts are needed for some graphs

Does there exist for all graph G on n vertices a CS-separator of size poly(n)? Or for which classes of graphs does it exist?

It is known that the following classes of graphs admit poly-size CS-Separator:

- If ω or α is bounded (trivial)
- chordal graphs (linear number of max. cliques)
- comparability graphs (Yannakakis 1991)
- C_4-free graphs (Conseq. of Alekseev 1991)
- P_5-free graphs (Conseq. of Lokshtanov, Vatchelle, Villanger 2014)

It is known that the following classes of graphs admit poly-size CS-Separator:

- If ω or α is bounded (trivial)
- chordal graphs (linear number of max. cliques)
- comparability graphs (Yannakakis 1991)
- C_4-free graphs (Conseq. of Alekseev 1991)
- P_5-free graphs (Conseq. of Lokshtanov, Vatchelle, Villanger 2014)

Joint work with Bousquet, Thomassé ;

It is known that the following classes of graphs admit poly-size CS-Separator:

- If ω or α is bounded (trivial)
- chordal graphs (linear number of max. cliques)
- comparability graphs (Yannakakis 1991)
- C_4-free graphs (Conseq. of Alekseev 1991)
- P_5-free graphs (Conseq. of Lokshtanov, Vatchelle, Villanger 2014)

Joint work with Bousquet, Thomassé ;

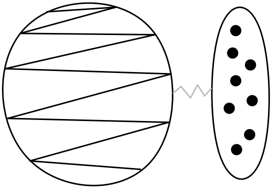
- Random graphs

It is known that the following classes of graphs admit poly-size CS-Separator:

- If ω or α is bounded (trivial)
- chordal graphs (linear number of max. cliques)
- comparability graphs (Yannakakis 1991)
- C_4-free graphs (Conseq. of Alekseev 1991)
- P_5-free graphs (Conseq. of Lokshtanov, Vatchelle, Villanger 2014)

Joint work with Bousquet, Thomassé ;

- Random graphs
- H-free graphs where H is a split graph.



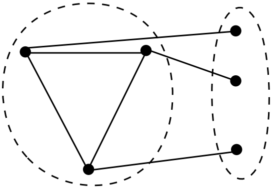
Split graph

It is known that the following classes of graphs admit poly-size CS-Separator:

- If ω or α is bounded (trivial)
- chordal graphs (linear number of max. cliques)
- comparability graphs (Yannakakis 1991)
- C_4-free graphs (Conseq. of Alekseev 1991)
- P_5-free graphs (Conseq. of Lokshtanov, Vatchelle, Villanger 2014)

Joint work with Bousquet, Thomassé ;

- Random graphs
- H-free graphs where H is a split graph.



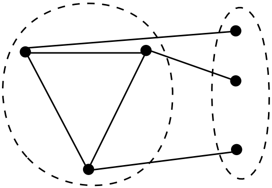
Net

It is known that the following classes of graphs admit poly-size CS-Separator:

- If ω or α is bounded (trivial)
- chordal graphs (linear number of max. cliques)
- comparability graphs (Yannakakis 1991)
- C_4-free graphs (Conseq. of Alekseev 1991)
- P_5-free graphs (Conseq. of Lokshtanov, Vatchelle, Villanger 2014)

Joint work with Bousquet, Thomassé ;

- Random graphs
- H-free graphs where H is a split graph.
- $(P_k, \overline{P_k})$-free graphs



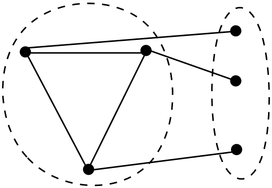
Net

It is known that the following classes of graphs admit poly-size CS-Separator:

- If ω or α is bounded (trivial)
- chordal graphs (linear number of max. cliques)
- comparability graphs (Yannakakis 1991)
- C_4-free graphs (Conseq. of Alekseev 1991)
- P_5-free graphs (Conseq. of Lokshtanov, Vatchelle, Villanger 2014)

Joint work with Bousquet, Thomassé ; and Trunck

- Random graphs
- H-free graphs where H is a split graph.
- $(P_k, \overline{P_k})$-free graphs
- Perfect graphs with no BSP



Net

Does there exist for all graph G on n vertices a CS-separator of size poly(n)?

~~**Does there exist for all graph G on n vertices a CS-separator of size poly(n)?**~~ **No!**

Lower Bound: (Göös 2015): we need $n^{\Omega(\log^{0.128} n)}$ cuts for some graphs.

~~**Does there exist for all graph G on n vertices a CS-separator of size poly(n)?**~~ **No!**

Lower Bound: (Göös 2015): we need $n^{\Omega(\log^{0.128} n)}$ cuts for some graphs.

What now?

~~**Does there exist for all graph G on n vertices a CS-separator of size poly(n)?**~~ **No!**

Lower Bound: (Göös 2015): we need $n^{\Omega(\log^{0.128} n)}$ cuts for some graphs.

What now?

$\rightarrow$ Want to learn more about perfect graphs and try to close the CS-Separation question on them.

Coloring perfect graphs with bounded clique number

JGA 2016

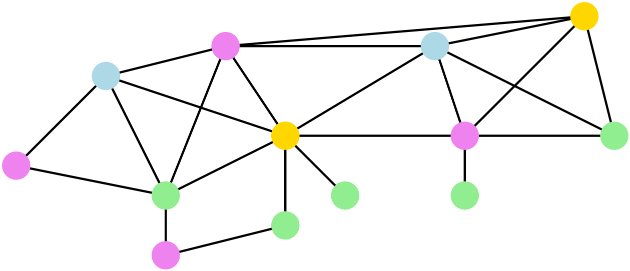
Aurélie Lagoutte

Joint work with M. Chudnovsky, P. Seymour and S. Spirkl

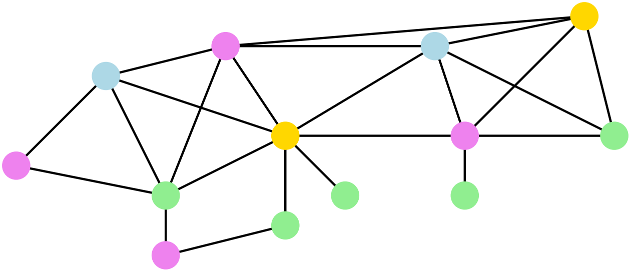
G-SCOP, Univ. Grenoble Alpes

November 17, 2016
Paris Dauphine

A **proper k-coloring** of G is an assignment of colors from $\{1, \dots, k\}$ such that any two adjacent vertices are given different colors.



A **proper k-coloring** of G is an assignment of colors from $\{1, \dots, k\}$ such that any two adjacent vertices are given different colors.

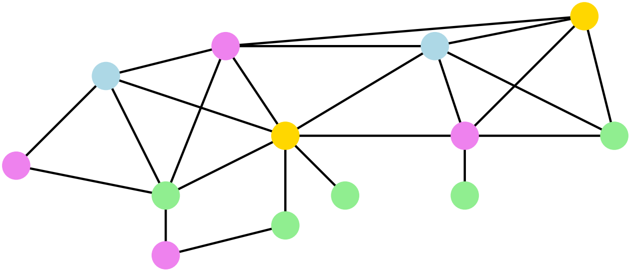


Graph Coloring

Input: A graph G and an integer k.
Output: Does G admits a proper k-coloring?

Graph Coloring is NP-complete.

A **proper** k**-coloring** of G is an assignment of colors from $\{1, \ldots, k\}$ such that any two adjacent vertices are given different colors.



~~Graph Coloring~~ 3-Coloring

Input: A graph G ~~and an integer k.~~

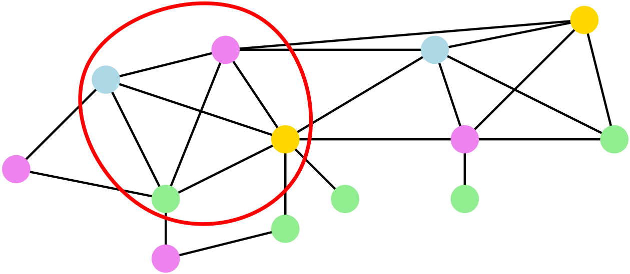
Output: Does G admits a proper ~~k-coloring?~~ 3-coloring?

Graph Coloring is NP-complete. Even 3-Coloring is!

- $\chi(G)$: **chromatic number** of G, i.e. minimum number of color in a proper coloring.
- $\omega(G)$: **clique number**, i.e. size of the largest clique.

- $\chi(G)$: **chromatic number** of G, i.e. minimum number of color in a proper coloring.
- $\omega(G)$: **clique number**, i.e. size of the largest clique.

$$\chi(G) \geq \omega(G)$$

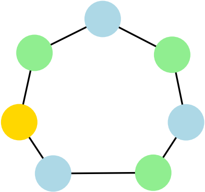


Perfect graph: definition

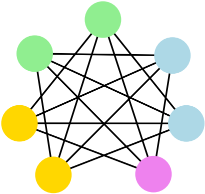
G is **perfect** if and only if $\chi(G) = \omega(G)$ and the equality holds for every induced subgraph H of G.

Perfect graph: definition

G is **perfect** if and only if $\chi(G) = \omega(G)$ and the equality holds for every induced subgraph H of G.



Odd hole C_7



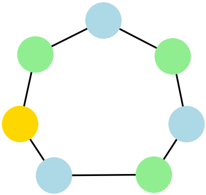
Odd antihole $\overline{C_7}$

Berge's Conjecture (1960's)

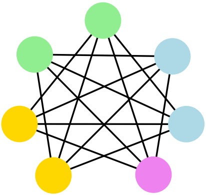
A graph G is perfect if and only if G contains no odd hole and no odd antihole as induced subgraph.

Perfect graph: definition

G is **perfect** if and only if $\chi(G) = \omega(G)$ and the equality holds for every induced subgraph H of G.



Odd hole C_7



Odd antihole $\overline{C_7}$

Berge's Conjecture (1960's) ⇒ Strong Perfect Graph Theorem

A graph G is perfect if and only if G contains no odd hole and no odd antihole as induced subgraph.

Proved in 2002 by Chudnovsky, Robertson, Seymour and Thomas.

What about coloring perfect graphs?

Theorem [Grötschel, Lovász, Schrijver 1981]

The Maximum Weighted Stable Set problem can be solved in polynomial time for perfect graphs.

What about coloring perfect graphs?

Theorem [Grötschel, Lovász, Schrijver 1981]

The Maximum Weighted Stable Set problem can be solved in polynomial time for perfect graphs.

Consequence

There is a polynomial-time algorithm that optimally colors any input perfect graph.

What about coloring perfect graphs?

Theorem [Grötschel, Lovász, Schrijver 1981]

The Maximum Weighted Stable Set problem can be solved in polynomial time for perfect graphs.

Consequence

There is a polynomial-time algorithm that optimally colors any input perfect graph.

$\Rightarrow$ Are we done??

What about coloring perfect graphs?

Theorem [Grötschel, Lovász, Schrijver 1981]

The Maximum Weighted Stable Set problem can be solved in polynomial time for perfect graphs.

Consequence

There is a polynomial-time algorithm that optimally colors any input perfect graph.

$\Rightarrow$ Are we done?? This algorithm uses the ellipsoid method:

- $\Rightarrow$ commonly acknowledged to be unpractical.
- $\Rightarrow$ Theoretical point of view: translates into semi-definite programming and we loose any understanding on the ongoing process.

What about coloring perfect graphs?

Theorem [Grötschel, Lovász, Schrijver 1981]

The Maximum Weighted Stable Set problem can be solved in polynomial time for perfect graphs.

Consequence

There is a polynomial-time algorithm that optimally colors any input perfect graph.

$\Rightarrow$ Are we done?? This algorithm uses the ellipsoid method:

- $\Rightarrow$ commonly acknowledged to be unpractical.
- $\Rightarrow$ Theoretical point of view: translates into semi-definite programming and we loose any understanding on the ongoing process.

Not satisfying! We know so much on perfect graphs that we want a *combinatorial* algorithm.

We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

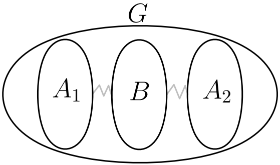
- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

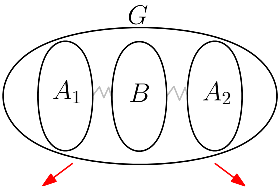


We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

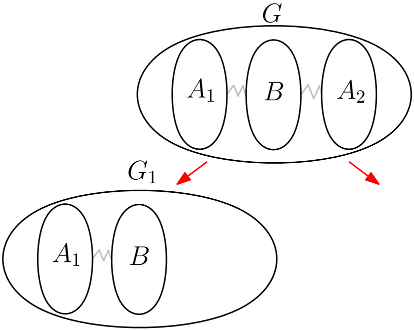


We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

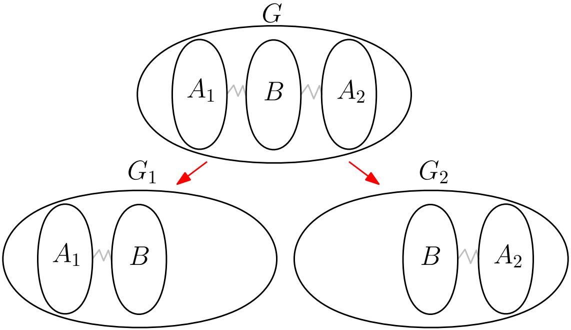


We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.



We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

How to use it?

We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

How to use it?

For structural purposes: want to prove that any Berge graph satisfies some property $\mathcal{P}$ (ex: *is perfect*).

We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

How to use it?

For structural purposes: want to prove that any Berge graph satisfies some property $\mathcal{P}$ (ex: *is perfect*).

Take G a minimal counter-example, i.e. Berge but does not satisfy $\mathcal{P}$.

We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

How to use it?

For structural purposes: want to prove that any Berge graph satisfies some property $\mathcal{P}$ (ex: *is perfect*).

Take G a minimal counter-example, i.e. Berge but does not satisfy $\mathcal{P}$.

- Prove that G cannot be decomposed (get a smaller counter-example)
- Prove that any basic graph satisfies $\mathcal{P}$.

We know *so much*?

Decomposition theorem from [CRST'02]

If G is Berge, then

- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

How to use it?

For structural purposes: want to prove that any Berge graph satisfies some property $\mathcal{P}$ (ex: *is perfect*).

Take G a minimal counter-example, i.e. Berge but does not satisfy $\mathcal{P}$.

- Prove that G cannot be decomposed (get a smaller counter-example)
- Prove that any basic graph satisfies $\mathcal{P}$.

$\Rightarrow$ Contradiction!

How to use it for algorithmic purposes?

Want to compute something (coloring, largest stable set, ...).

How to use it for algorithmic purposes?

Want to compute something (coloring, largest stable set, ...).

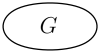
Meta-algorithm:

How to use it for algorithmic purposes?

Want to compute something (coloring, largest stable set, ...).

Meta-algorithm:

1. Construct the decomposition tree:

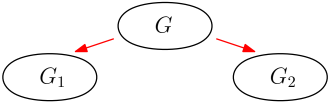


How to use it for algorithmic purposes?

Want to compute something (coloring, largest stable set, ...).

Meta-algorithm:

1. Construct the decomposition tree:

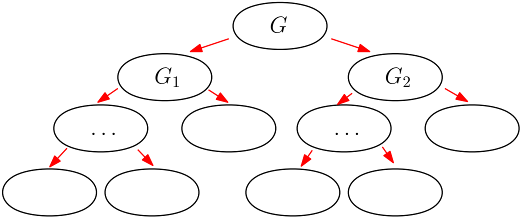


How to use it for algorithmic purposes?

Want to compute something (coloring, largest stable set, ...).

Meta-algorithm:

1. Construct the decomposition tree:

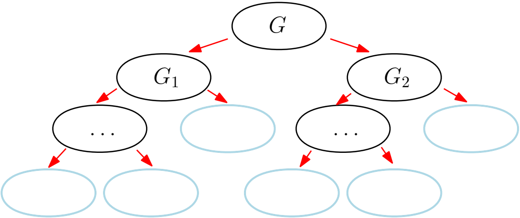


How to use it for algorithmic purposes?

Want to compute something (coloring, largest stable set, ...).

Meta-algorithm:

1. Construct the decomposition tree:



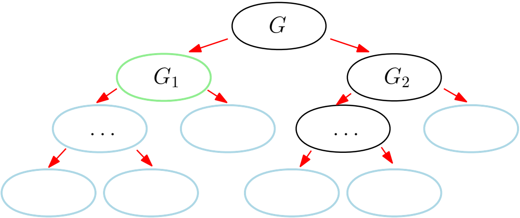
2. Compute what you want on the **leaves** ($\rightarrow$ *basic* graphs).

How to use it for algorithmic purposes?

Want to compute something (coloring, largest stable set, ...).

Meta-algorithm:

1. Construct the decomposition tree:



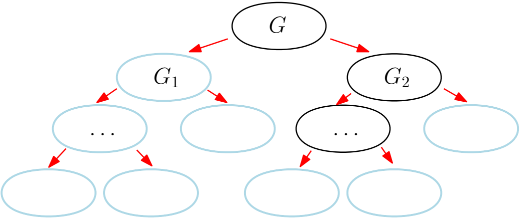
2. Compute what you want on the **leaves** ($\rightarrow$ *basic* graphs).
3. From bottom to top: **combine solutions for children** to get a solution for the father.

How to use it for algorithmic purposes?

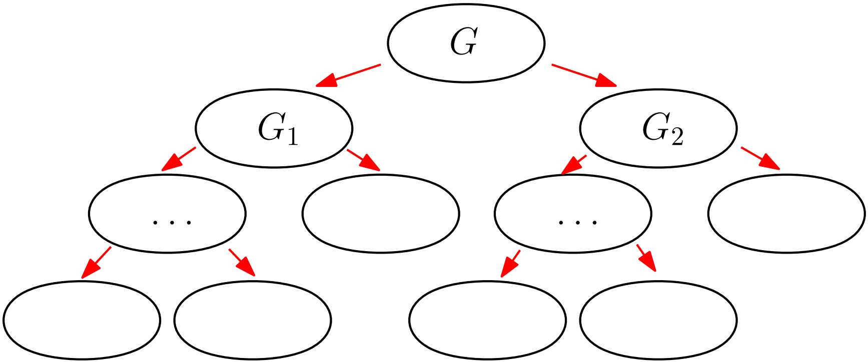
Want to compute something (coloring, largest stable set, ...).

Meta-algorithm:

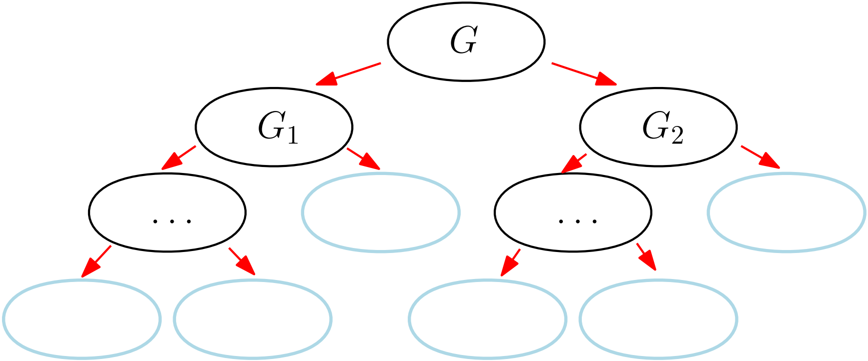
1. Construct the decomposition tree:



2. Compute what you want on the **leaves** ($\rightarrow$ *basic* graphs).
3. From bottom to top: **combine solutions for children** to get a solution for the father.

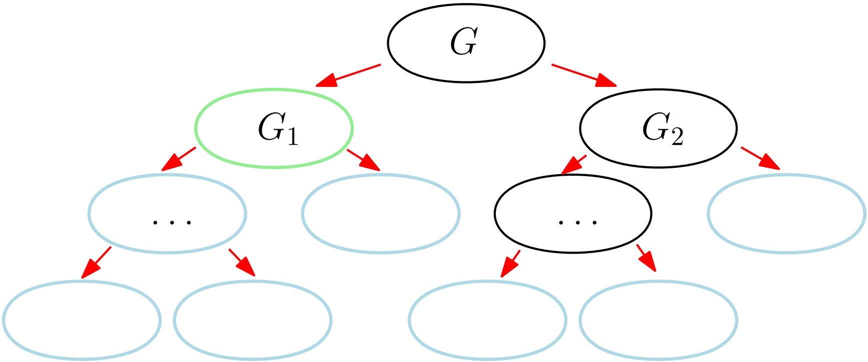


Hence four intermediate steps to reach:



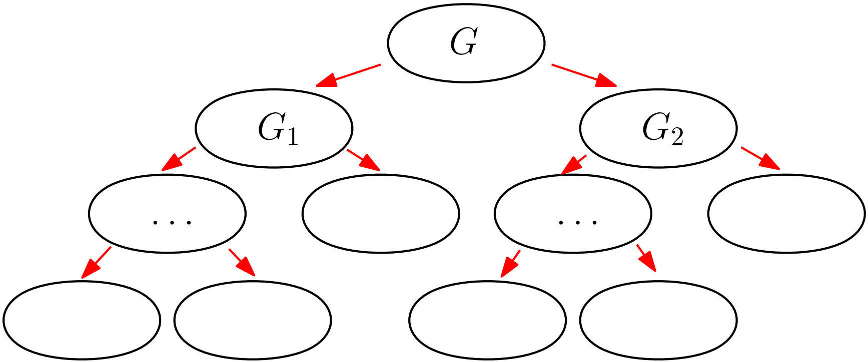
Hence four intermediate steps to reach:

- Know how to directly solve the problem on leaves.



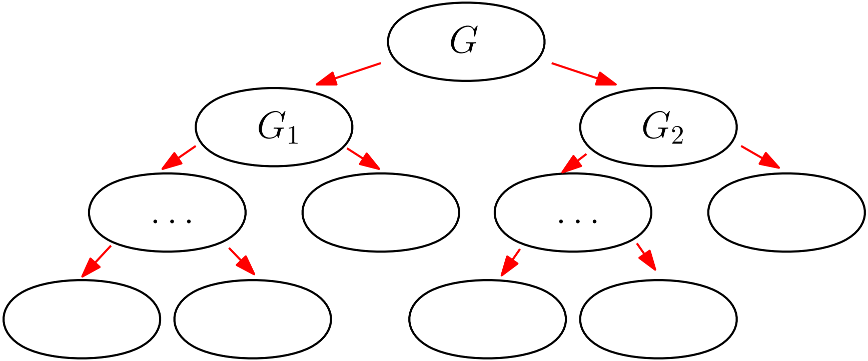
Hence four intermediate steps to reach:

- Know how to directly solve the problem on leaves.
- Know how to go from children to father (combining solutions).



Hence four intermediate steps to reach:

- Know how to directly solve the problem on leaves.
- Know how to go from children to father (combining solutions).
- Bound the size of the tree by a polynomial in n.



Hence four intermediate steps to reach:

- Know how to directly solve the problem on leaves.
- Know how to go from children to father (combining solutions).
- Bound the size of the tree by a polynomial in n.
- Know how to algorithmically construct the tree.

Our result

Theorem [Chudnovsky, L., Seymour, Spirkl]

We design an algorithm with the following specification:

Algorithm $\mathcal{A}_k$:

Input: A perfect graph G with $\omega(G) \leq k$.

Output: A proper coloring of G with $\chi(G) = \omega(G)$ colors.

Running time: $\mathcal{O}(n^{(\omega(G)+1)^2})$

Our result

Theorem [Chudnovsky, L., Seymour, Spirkl]

We design an algorithm with the following specification:
Algorithm $\mathcal{A}_k$:
Input: A perfect graph G with $\omega(G) \leq k$.
Output: A proper coloring of G with $\chi(G) = \omega(G)$ colors.
Running time: $\mathcal{O}(n^{(\omega(G)+1)^2})$

We proceed by induction on $k \rightarrow$ we can call $\mathcal{A}_{k-1}$ when needed.

Previous results in this direction:

Previous results in this direction:

A combinatorial algorithm that optimally colors:

- *any Berge graph with no BSP*

  [Chudnovsky, Trotignon, Trunkc, Vušković 2015]

- *any C_4-free Berge graph*

  [Chudnovsky, Lo, Maffray, Trotignon, Vušković 2015^+]

Outline

Five intermediate steps to reach:

- Describe the decomposition tree that is used
- Know how to algorithmically construct the tree
- Know how to directly solve the problem on leaves
- Bound the size of the tree by a polynomial in n
- Know how to go from children to father (combining solutions)

Decomposing perfect graphs

Decomposition theorem

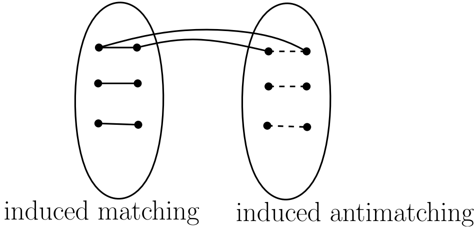
If G is perfect, then at least one of the following holds:

Decomposing perfect graphs

Decomposition theorem

If G is perfect, then at least one of the following holds:

- G or $\overline{G}$ lies in one of the following classes: bipartite graphs, line graphs of a bipartite graph, double split.



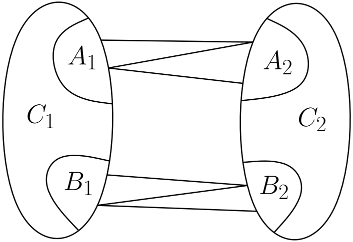
Double-Split

Decomposing perfect graphs

Decomposition theorem

If G is perfect, then at least one of the following holds:

- G or $\overline{G}$ lies in one of the following classes: bipartite graphs, line graphs of a bipartite graph, double split.
- G or $\overline{G}$ admits a decomposition by 2-join,



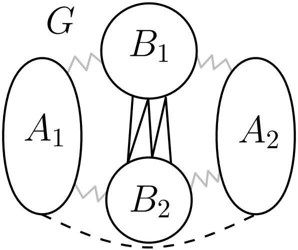
2-join

Decomposing perfect graphs

Decomposition theorem

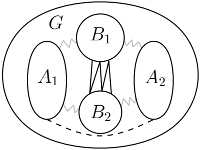
If G is perfect, then at least one of the following holds:

- G or $\overline{G}$ lies in one of the following classes: bipartite graphs, line graphs of a bipartite graph, double split.
- G or $\overline{G}$ admits a decomposition by 2-join,
- G admits a decomposition by balanced skew partition (BSP).



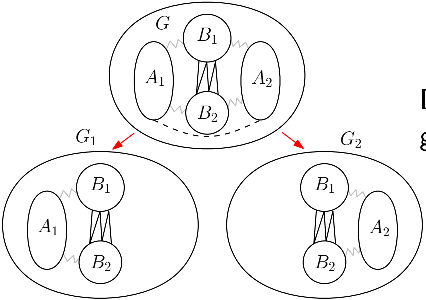
Skew partition

Our decomposition tree



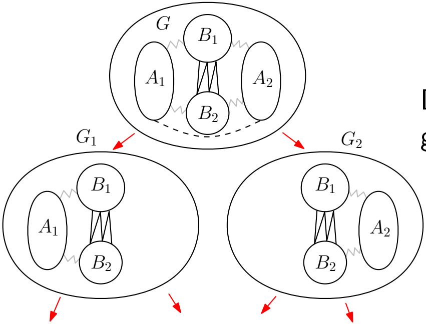
Decompose along BSP until the graph:

Our decomposition tree



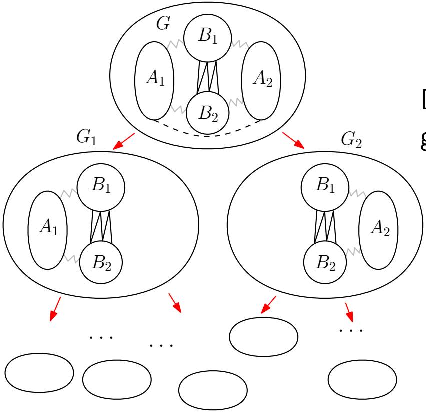
Decompose along BSP until the graph:

Our decomposition tree



Decompose along BSP until the graph:

Our decomposition tree



Decompose along BSP until the graph:

- admits no BSP,
- or is not anticonnected,
- or has clique number $< k$,
- or has bounded size $< 2k$.

Outline

Five intermediate steps to reach:

- ✓ Describe the decomposition tree that is used
- Know how to algorithmically construct the tree
- Know how to directly solve the problem on leaves
- Bound the size of the tree by a polynomial in n
- Know how to go from children to father (combining solutions)

How to algorithmically construct the tree?

Find a BSP in polynomial time?

How to algorithmically construct the tree?

Find a BSP in polynomial time?

Theorem [Chudnovsky, L., Seymour, Spirkl]

There is an algorithm that, given as input a perfect graph G, outputs a BSP of G or asserts that there is none.

How to algorithmically construct the tree?

Find a BSP in polynomial time?

Theorem [Chudnovsky, L., Seymour, Spirkl]

There is an algorithm that, given as input a perfect graph G, outputs a BSP of G or asserts that there is none.

Previous results:

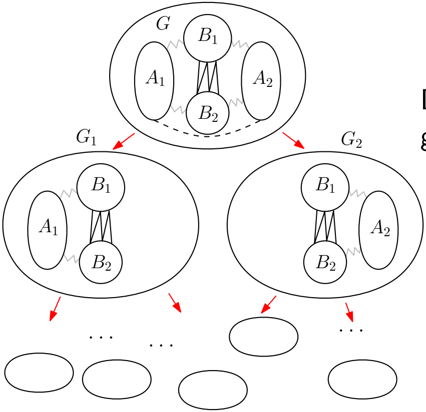
- Deciding if a graph has a BSP is NP-complete. [Trotignon 08]
- A poly-time algorithm that decides if a **perfect** graph has a BSP can be done in polynomial-time (but, if yes, the algo does not output such a partition). [Trotignon 08]
- A poly-time algo that decides if a graph has a skew partition and, if yes, outputs such a partition. [Kennedy & Reed 08]

Outline

Five intermediate steps to reach:

- ✓ Describe the decomposition tree that is used
- ✓ Know how to algorithmically construct the tree
- Know how to directly solve the problem on leaves
  $\rightarrow$ time $\mathcal{O}(n^{\max(7,\omega(G)^2)})$
- Bound the size of the tree by a polynomial in n
- Know how to go from children to father (combining solutions)

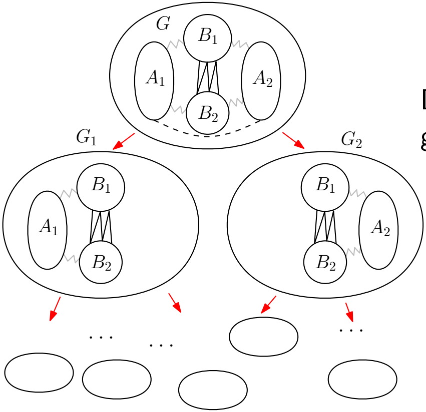
Our decomposition tree



Decompose along BSP until the graph:

- admits no BSP,
- or is not anticonnected,
- or has clique number $< k$,
- or has bounded size $< 2k$.

Our decomposition tree

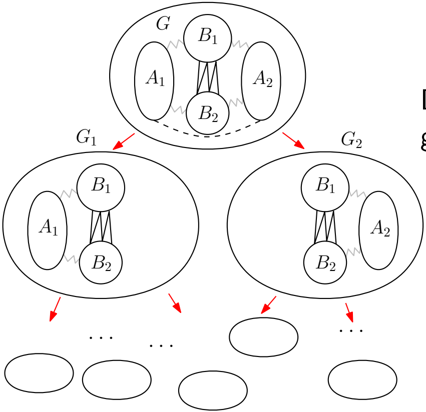


Decompose along BSP until the graph:

- admits no BSP,
- or is not anticonnected,
- or has clique number $< k$,
- or has bounded size $< 2k$.

Color with CTTV algo $\rightarrow \mathcal{O}(n^7)$

Our decomposition tree

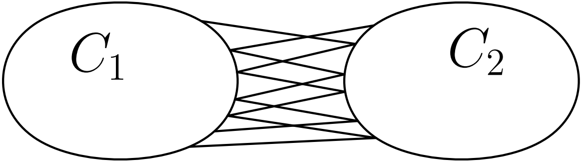


Decompose along BSP until the graph:

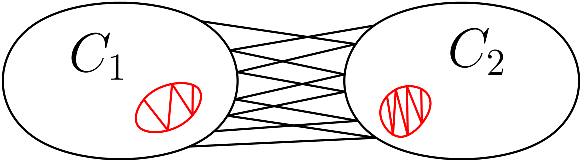
- admits no BSP,
- or is not anticonnected,
- or has clique number $< k$,
- or has bounded size $< 2k$.

Color with CTTV algo $\rightarrow \mathcal{O}(n^7)$
Color with $\mathcal{A}_{k-1} \rightarrow \mathcal{O}(n^{\omega(G)^2})$

If G is not anticonnected:



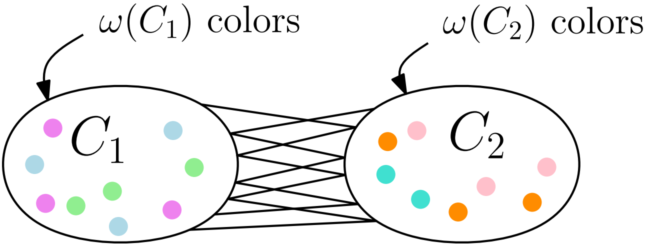
If G is not anticonnected:



$$\omega(C_1) + \omega(C_2) = \omega(G)$$

$$\omega(C_1), \omega(C_2) < \omega(G)$$

If G is not anticonnected:

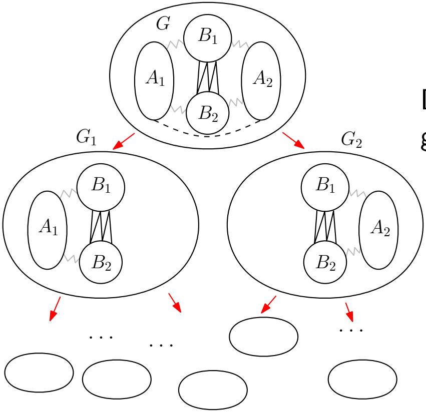


$$\omega(C_1) + \omega(C_2) = \omega(G)$$

$$\omega(C_1), \omega(C_2) < \omega(G)$$

Color each side with $\mathcal{A}_{k-1}$

Our decomposition tree

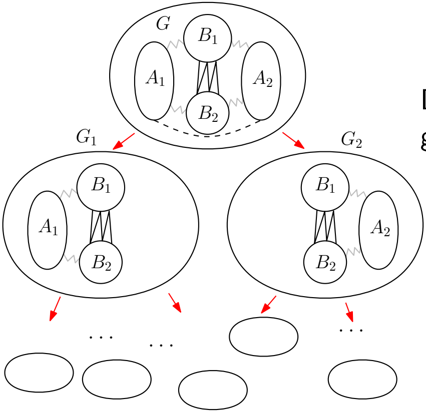


Decompose along BSP until the graph:

- admits no BSP,
- or is not anticonnected,
- or has clique number $< k$,
- or has bounded size $< 2k$.

Color with CTTV algo $\to \mathcal{O}(n^7)$
Color with $\mathcal{A}_{k-1} \to \mathcal{O}(n^{\omega(G)^2})$

Our decomposition tree



Decompose along BSP until the graph:

- admits no BSP,
- or is not anticonnected,
- or has clique number $< k$,
- or has bounded size $< 2k$.

Color with CTTV algo $\rightarrow \mathcal{O}(n^7)$
Color with $\mathcal{A}_{k-1} \rightarrow \mathcal{O}(n^{\omega(G)^2})$
Easy to color in $f(k)$

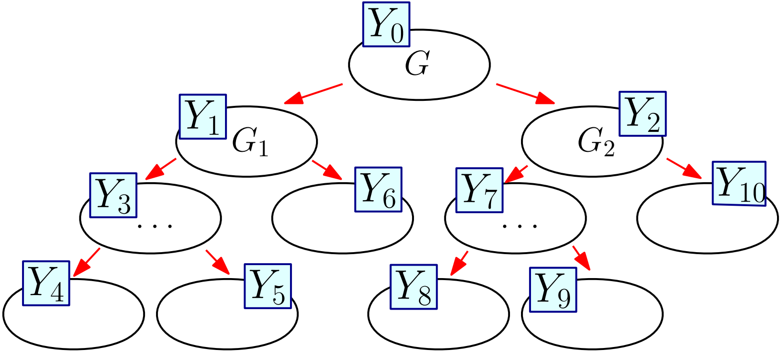
Outline

Five intermediate steps to reach:

- ✓ Describe the decomposition tree that is used
- ✓ Know how to algorithmically construct the tree
- ✓ Know how to directly solve the problem on leaves
  $\rightarrow$ time $\mathcal{O}(n^{\max(7,\omega(G)^2)})$
- Bound the size of the tree by a polynomial in n
- Know how to go from children to father (combining solutions)

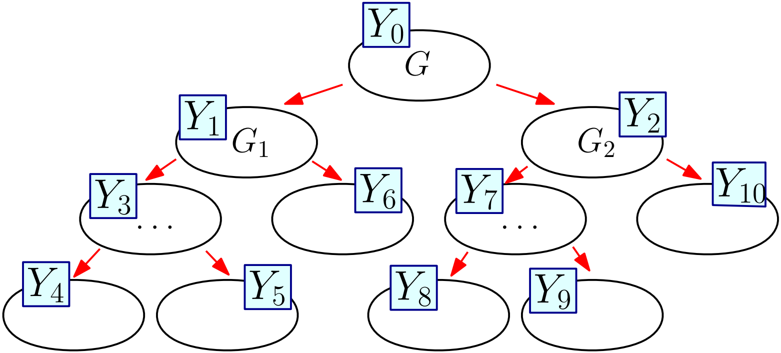
How to bound the size of the tree?

Label each node of the tree with some well-chosen $Y \subseteq V(G)$:



How to bound the size of the tree?

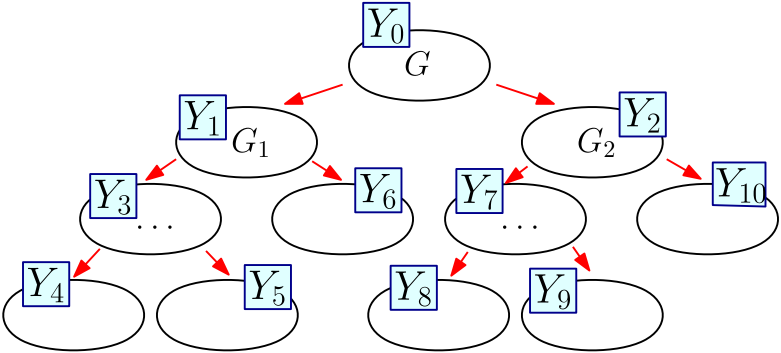
Label each node of the tree with some well-chosen $Y \subseteq V(G)$:



- Each label is different from every other labels,

How to bound the size of the tree?

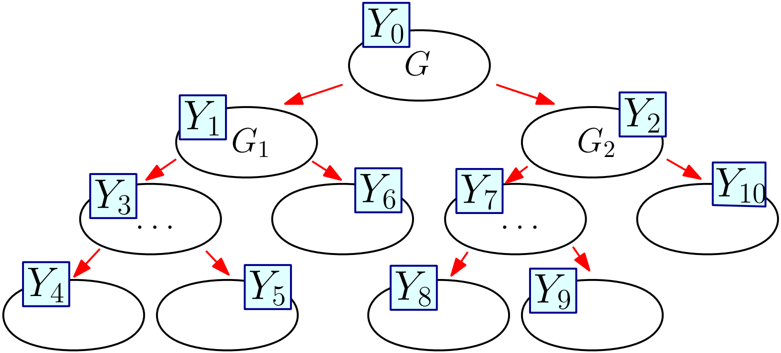
Label each node of the tree with some well-chosen $Y \subseteq V(G)$:



- Each label is different from every other labels,
- The number of candidates for labeling is bounded by a polynomial.

How to bound the size of the tree?

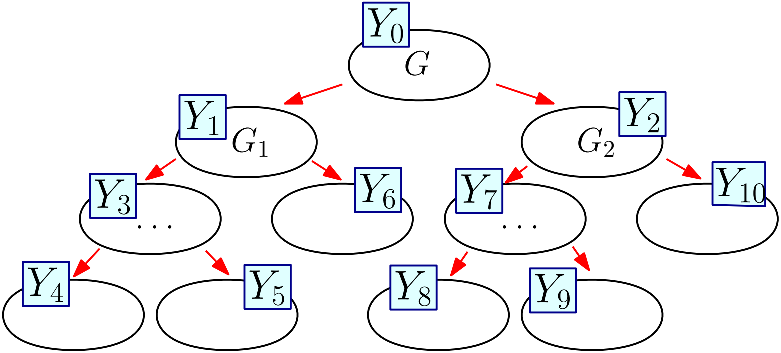
Label each node of the tree with some well-chosen $Y \subseteq V(G)$:



- Each label is different from every other labels,
- The number of candidates for labeling is bounded by a polynomial.

How to bound the size of the tree?

Label each node of the tree with some well-chosen $Y \subseteq V(G)$:



- Each label is different from every other labels,
- The number of candidates for labeling is bounded by a polynomial.

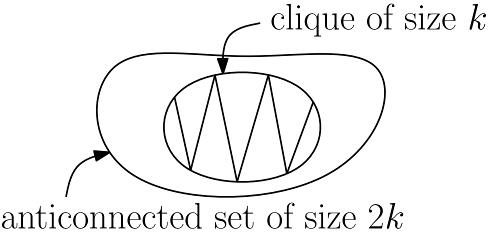
$\Rightarrow$ Bounds the number of nodes by a polynomial.

Key ingredient: k-pellet

Definition: k-pellet

A subset $Y \subseteq V(G)$ is a k-pellet if

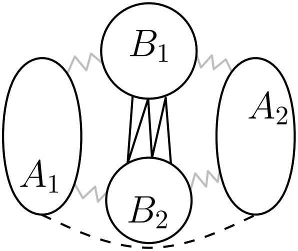
- Y contains a clique of size k,
- Y is anticonnected,
- and $|Y| = 2k$.



Number of $\omega(G)$-pellets: at most $\mathcal{O}(n^{2\omega(G)})$!!

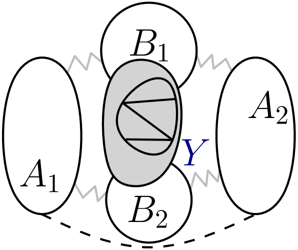
Good property of $\omega(G)$-pellet

An $\omega(G)$-pellet can not lie in the middle part $B_1 \cup B_2$ of a BSP.



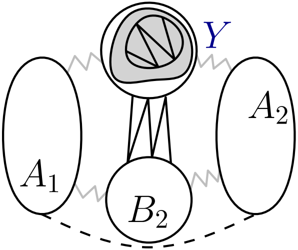
Good property of $\omega(G)$-pellet

An $\omega(G)$-pellet can not lie in the middle part $B_1 \cup B_2$ of a BSP.



Good property of $\omega(G)$-pellet

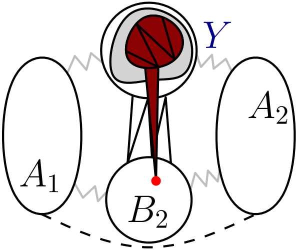
An $\omega(G)$-pellet can not lie in the middle part $B_1 \cup B_2$ of a BSP.



Y is anticonnected.

Good property of $\omega(G)$-pellet

An $\omega(G)$-pellet can not lie in the middle part $B_1 \cup B_2$ of a BSP.



Y is anticonnected.

Y contains a clique of size $\omega(G)$ and any $v \in B_2$ is complete to it.

$\Rightarrow$ Contradiction!

Unique labeling

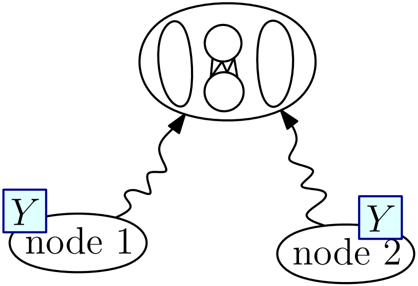
Two nodes getting the same label Y?

Unique labeling

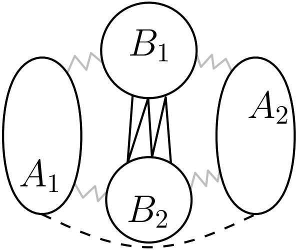
Two nodes getting the same label Y?

Unique labeling

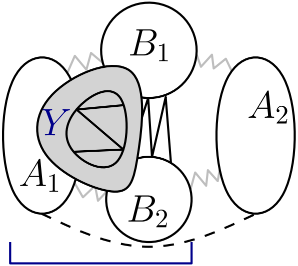
Two nodes getting the same label Y?



Where is Y?

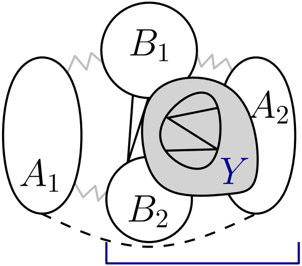


Where is Y?



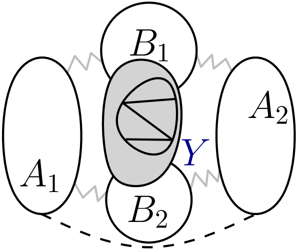
Y appears in left descendants.

Where is Y?



Y appears in left descendants.
Y appears in right descendants.

Where is Y?



Y appears in left descendants.
Y appears in right descendants.
$\Rightarrow$ Contradiction!

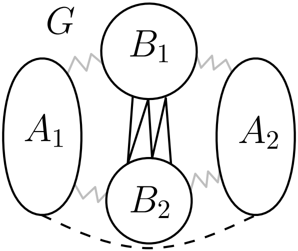
Outline

Five intermediate steps to reach:

- ✓ Describe the decomposition tree that is used
- ✓ Know how to algorithmically construct the tree
- ✓ Know how to directly solve the problem on leaves
  $\rightarrow$ time $\mathcal{O}(n^{\max(7,\omega(G)^2)})$
- ✓ Bound the size of the tree by a polynomial in n
- Know how to go from children to father (combining solutions)

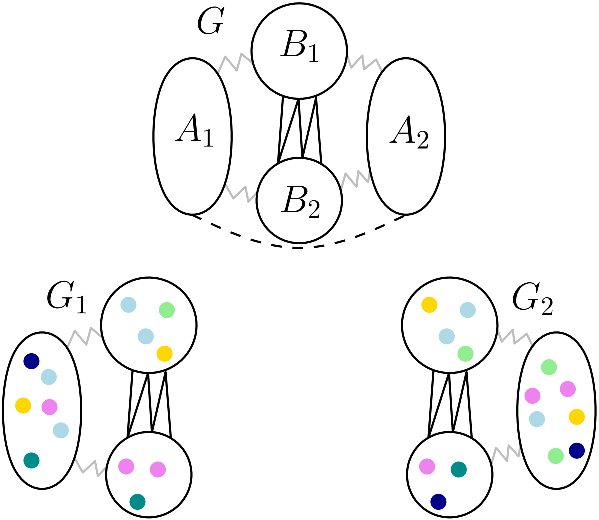
How to combine solutions?

Problem when gluing solutions



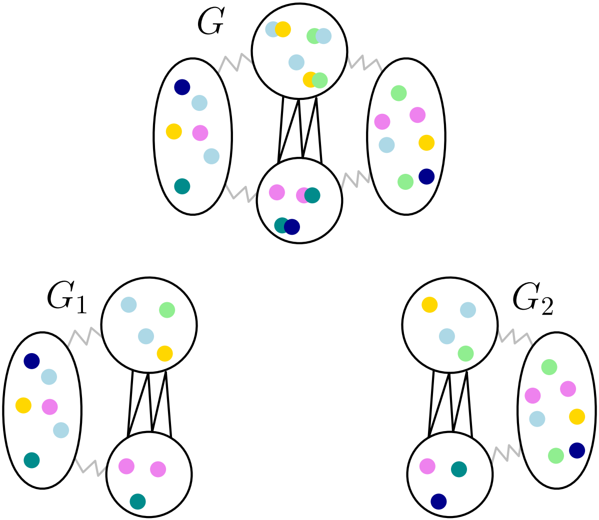
How to combine solutions?

Problem when gluing solutions



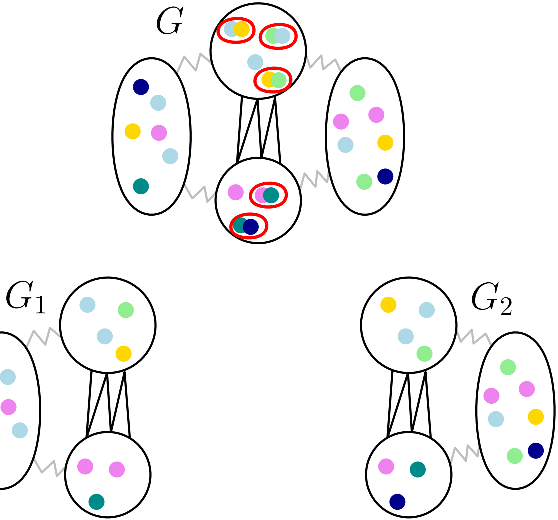
How to combine solutions?

Problem when gluing solutions



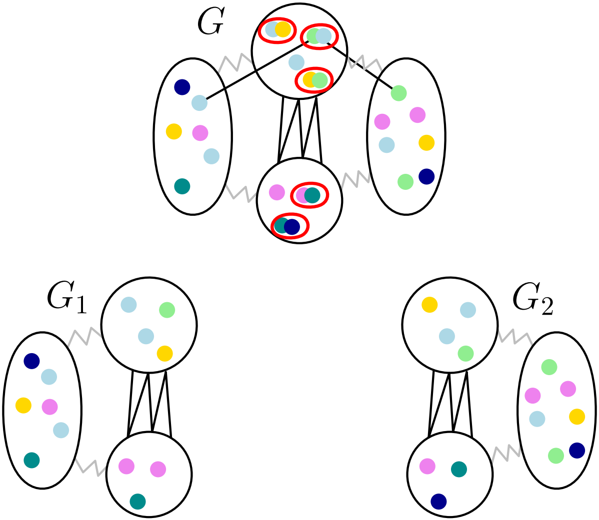
How to combine solutions?

Problem when gluing solutions



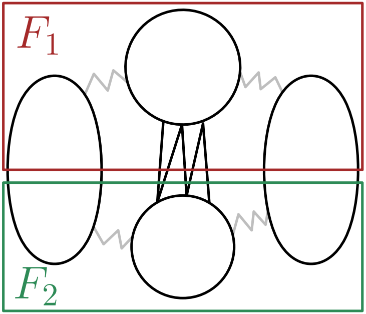
How to combine solutions?

Problem when gluing solutions



Goal: Find a partition in two sets (F_1, F_2):

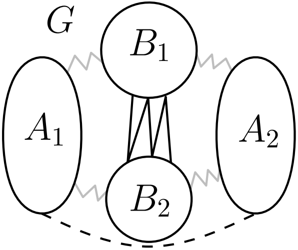
- $\omega(F_1) = k_1 < \omega(G)$;
- $\omega(F_2) = k_2 < \omega(G)$;
- $k_1 + k_2 = \omega(G)$.



Then we will call $\mathcal{A}_{k-1}$ on F_1 and F_2.

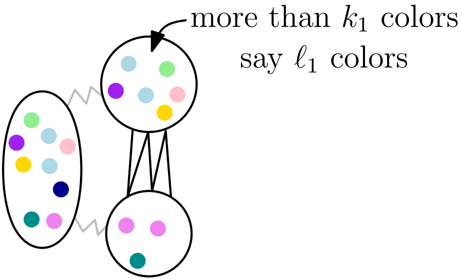
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).



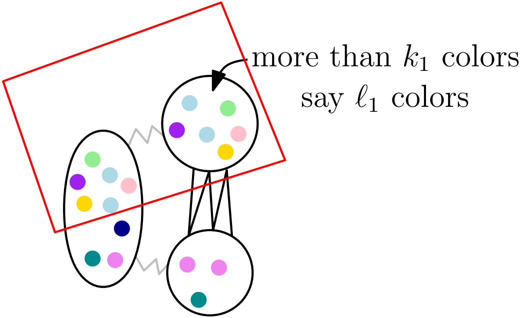
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).
2. Consider the solution on the left part:



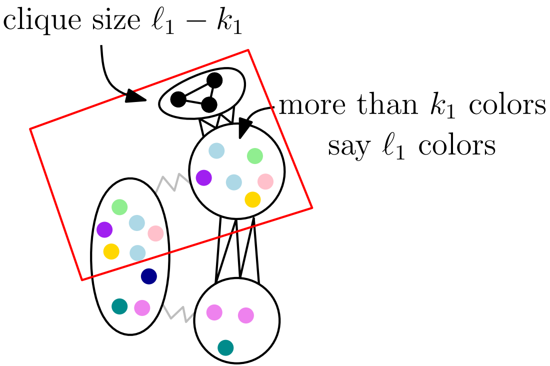
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).
2. Consider the solution on the left part:



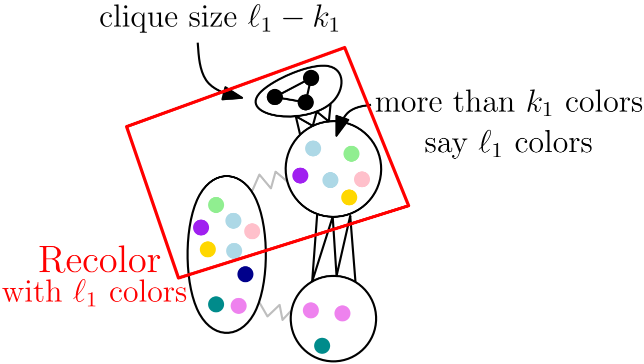
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).
2. Consider the solution on the left part:



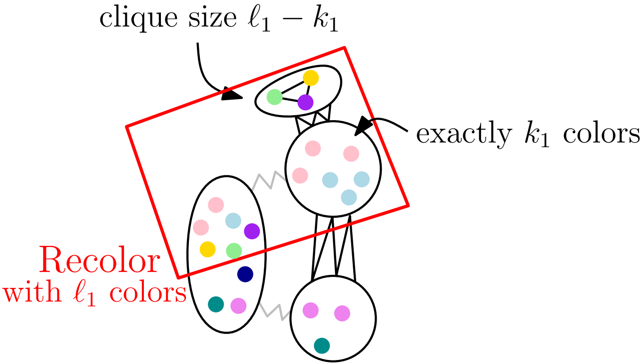
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).
2. Consider the solution on the left part:



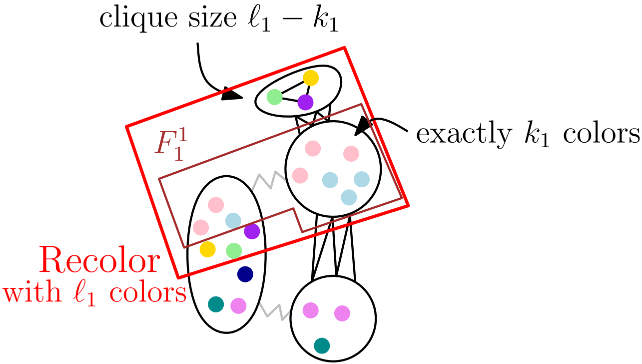
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).
2. Consider the solution on the left part:



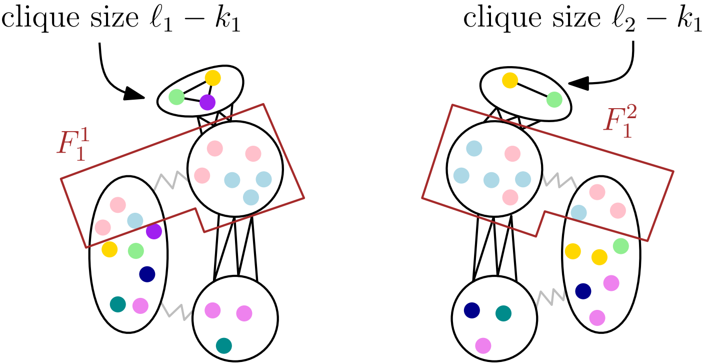
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).
2. Consider the solution on the left part:



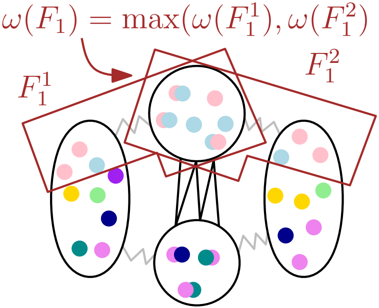
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).
2. Consider the solution on the left part:
3. Do the same on the right part and set $F_1 = F_1^1 \cup F_1^2$.



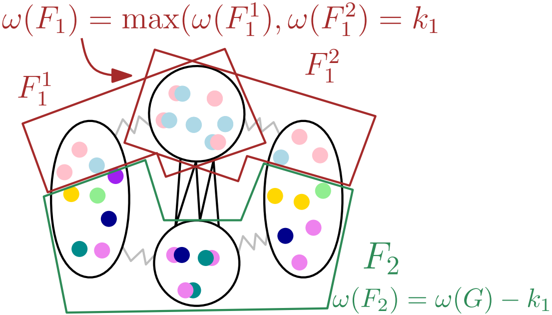
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).
2. Consider the solution on the left part:
3. Do the same on the right part and set $F_1 = F_1^1 \cup F_1^2$.



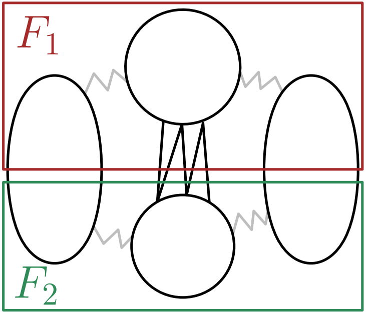
How to find such a partition (F_1, F_2)?

1. Compute $k_1 = \omega(B_1) < k$ (test every X s.t. $|X| < k$).
2. Consider the solution on the left part:
3. Do the same on the right part and set $F_1 = F_1^1 \cup F_1^2$.



Goal: Find a partition in two sets (F_1, F_2):

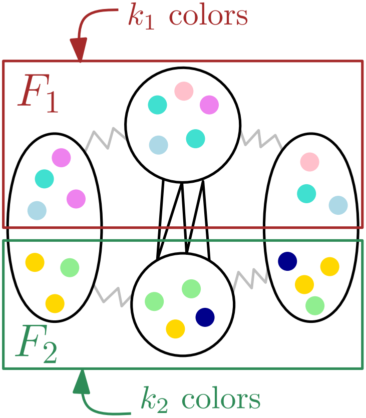
- $\omega(F_1) = k_1 < \omega(G)$;
- $\omega(F_2) = k_2 < \omega(G)$;
- $k_1 + k_2 = \omega(G)$.



Call $\mathcal{A}_{k-1}$ on F_1 and F_2. This gives a proper $\omega(G)$-coloring of G.

Goal: Find a partition in two sets (F_1, F_2):

- $\omega(F_1) = k_1 < \omega(G)$;
- $\omega(F_2) = k_2 < \omega(G)$;
- $k_1 + k_2 = \omega(G)$.



Call $\mathcal{A}_{k-1}$ on F_1 and F_2. This gives a proper $\omega(G)$-coloring of G.

Outline

Five intermediate steps to reach:

- ✓ Describe the decomposition tree that is used
- ✓ Know how to algorithmically construct the tree
- ✓ Know how to directly solve the problem on leaves
  $\rightarrow$ time $\mathcal{O}(n^{\max(7,\omega(G)^2)})$
- ✓ Bound the size of the tree by a polynomial in n
- ✓ Know how to go from children to father (combining solutions)
  $\rightarrow$ time $\mathcal{O}(n^{2\omega(G)})$

Outline

Five intermediate steps to reach:

- ✓ Describe the decomposition tree that is used
- ✓ Know how to algorithmically construct the tree
- ✓ Know how to directly solve the problem on leaves
  $\rightarrow$ time $\mathcal{O}(n^{\max(7,\omega(G)^2)})$
- ✓ Bound the size of the tree by a polynomial in n
- ✓ Know how to go from children to father (combining solutions)
  $\rightarrow$ time $\mathcal{O}(n^{2\omega(G)})$

$\rightarrow$ Algorithm $\mathcal{A}_k$ is well-defined and runs in time $\mathcal{O}(n^{(\omega(G)+1)^2})$

Perspectives

Ultimate aim: Color perfect graphs in the general case!

Perspectives

Ultimate aim: Color perfect graphs in the general case!

- How to bound the size of the tree?
- Could we modify the decomposition theorem?
- Could we get a FPT algorithm with parameter $\omega(G)$?

Perspectives

Ultimate aim: Color perfect graphs in the general case!

- How to bound the size of the tree?
- Could we modify the decomposition theorem?
- Could we get a FPT algorithm with parameter $\omega(G)$?

Thank you for your attention!