Coloring perfect graphs with bounded clique number

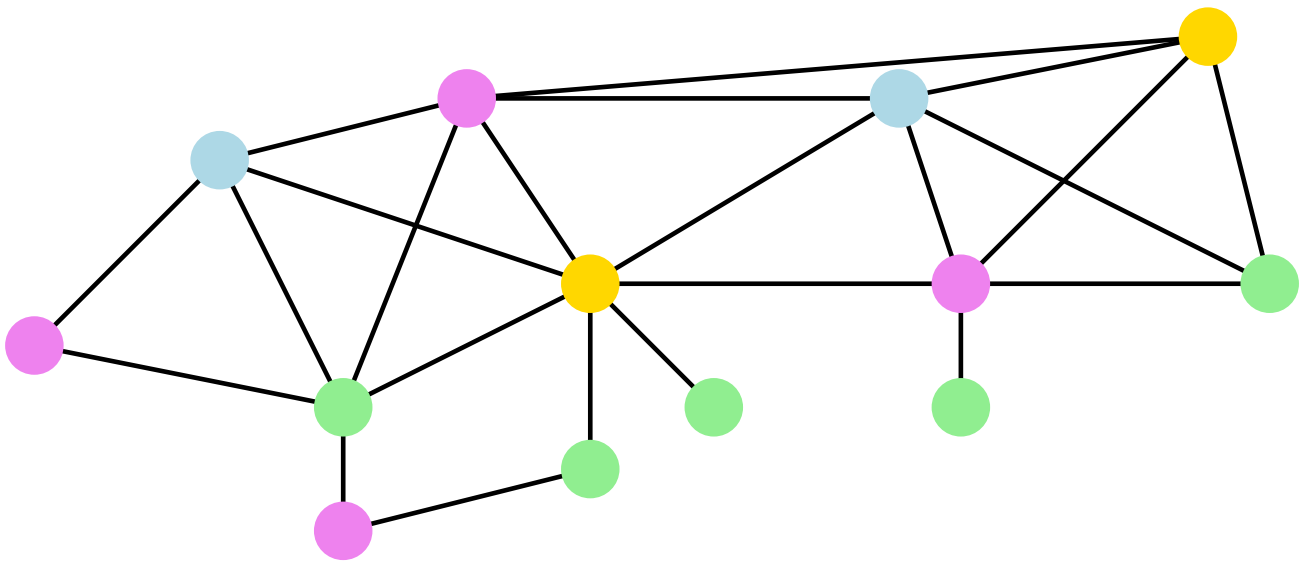
Joint work with M. Chudnovsky, P. Seymour and S. Spirkl

Aurélie Lagoutte

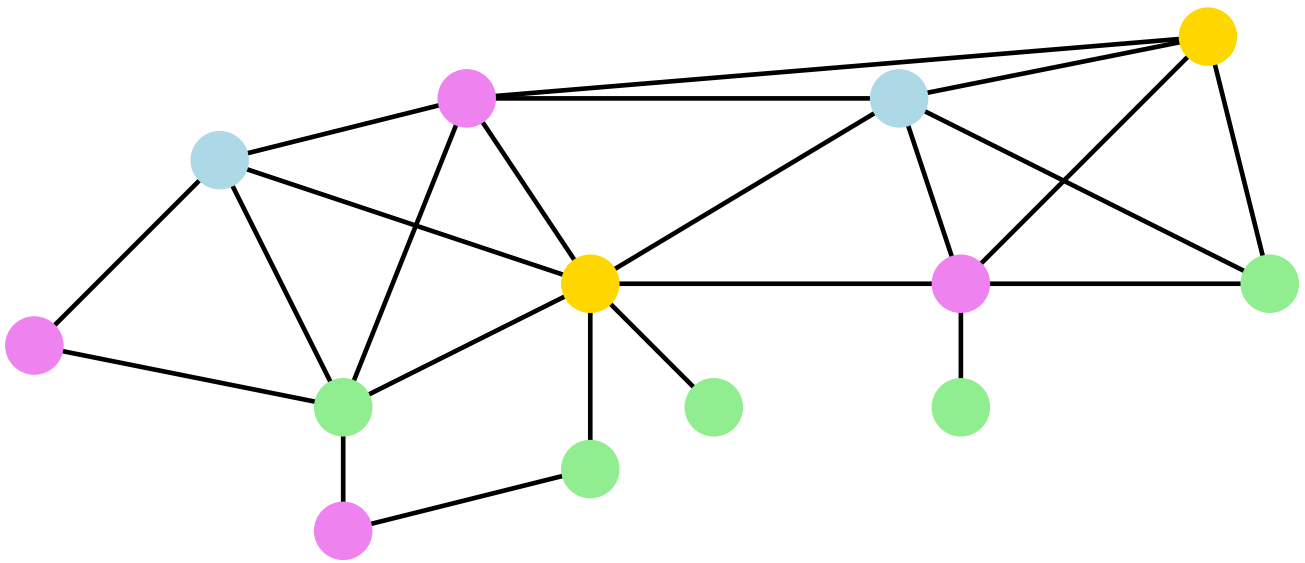
LIP, ENS Lyon

January 28, 2016
Séminaire AIGCo, LIRMM

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.



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.

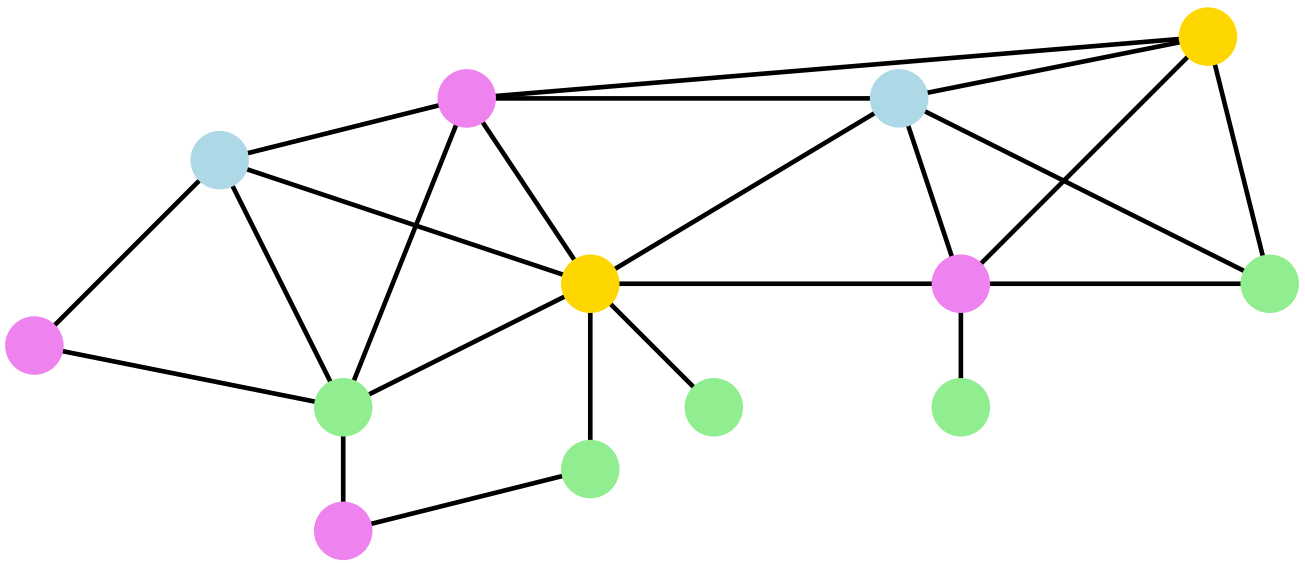


Graph Coloring

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.~~

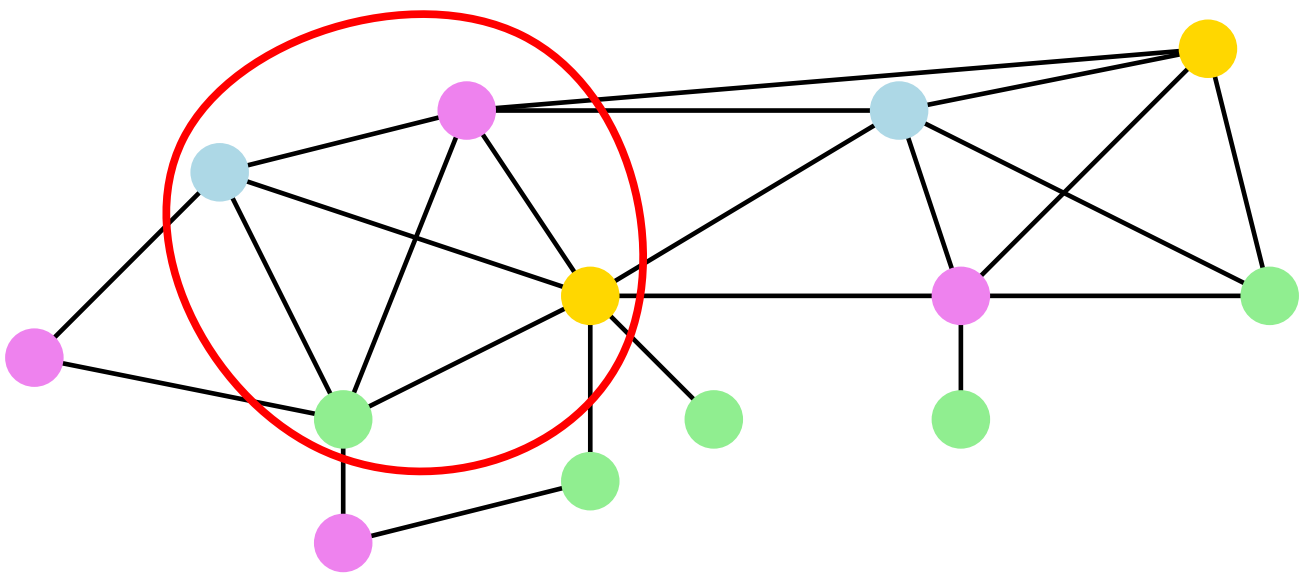
Output: Does G admits a proper ~~k-coloring?~~ 3-coloring?

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.

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

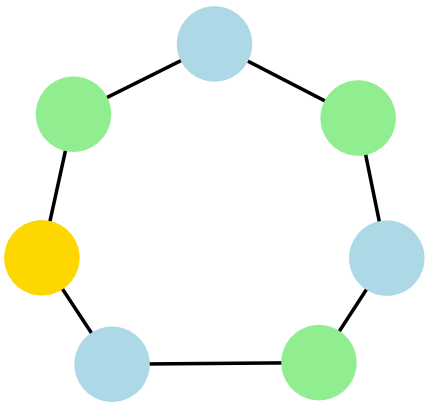


Perfect graph: definition

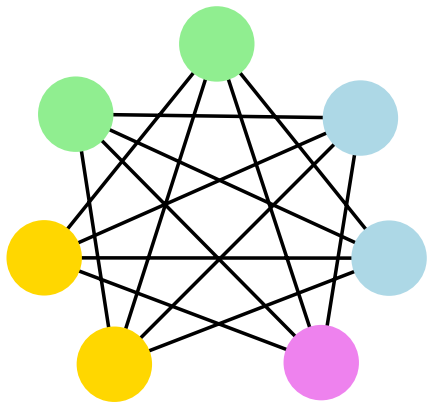
G is **perfect** if and only if $\chi(G) = \omega(G)$ and the equality holds for every induced subgraph H of G.

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



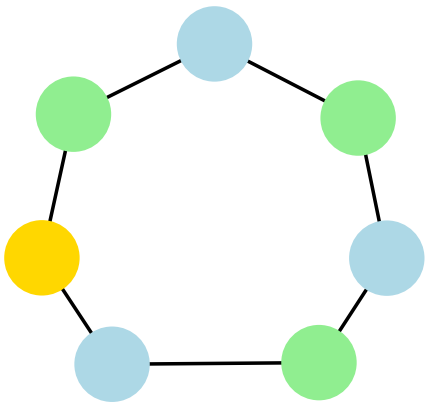
Odd antihole $\overline{C_7}$

Berge's Conjecture (1960's)

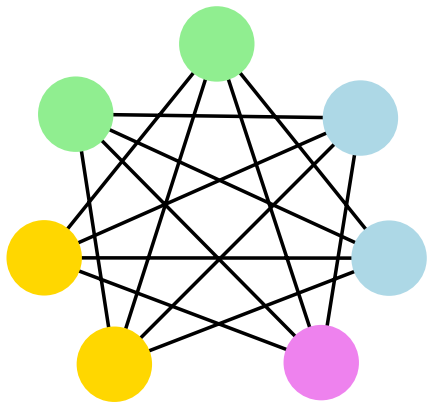
A graph G is perfect if and only if G contains no odd hole and no odd antihole as induced subgraph ($=G$ is *Berge*).

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) $\Rightarrow$ 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 ($=G$ is *Berge*).

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

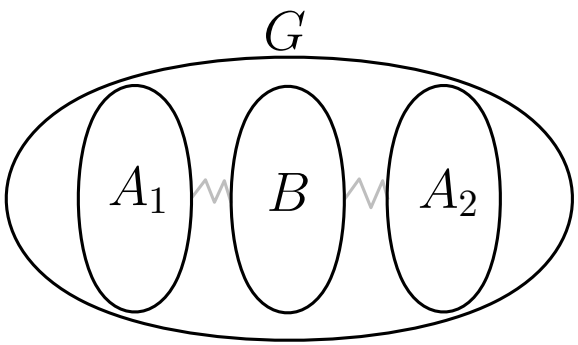
- either G is *basic* (bipartite, line graph of bipartite, ...),
- or G can be *decomposed* in a given way.

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.

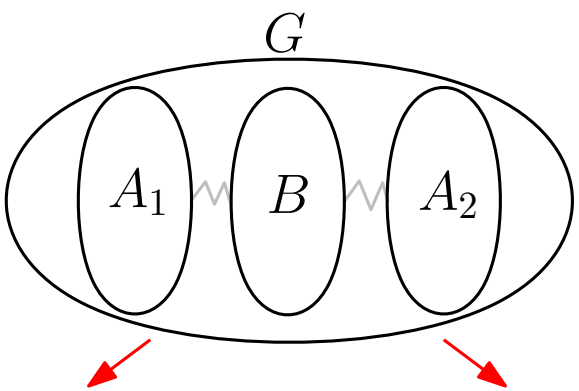


We know *so much*?

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.

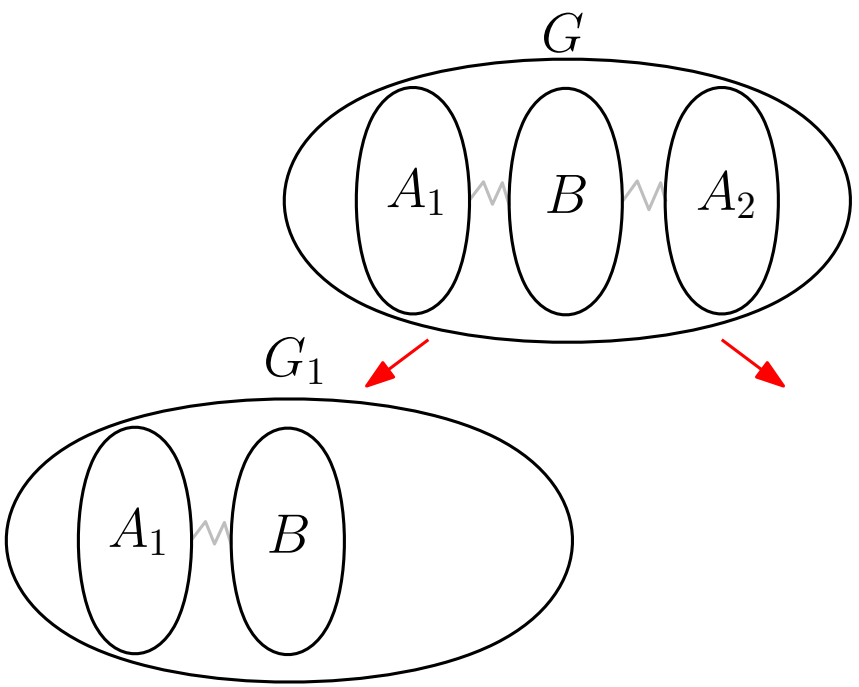


We know *so much*?

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.

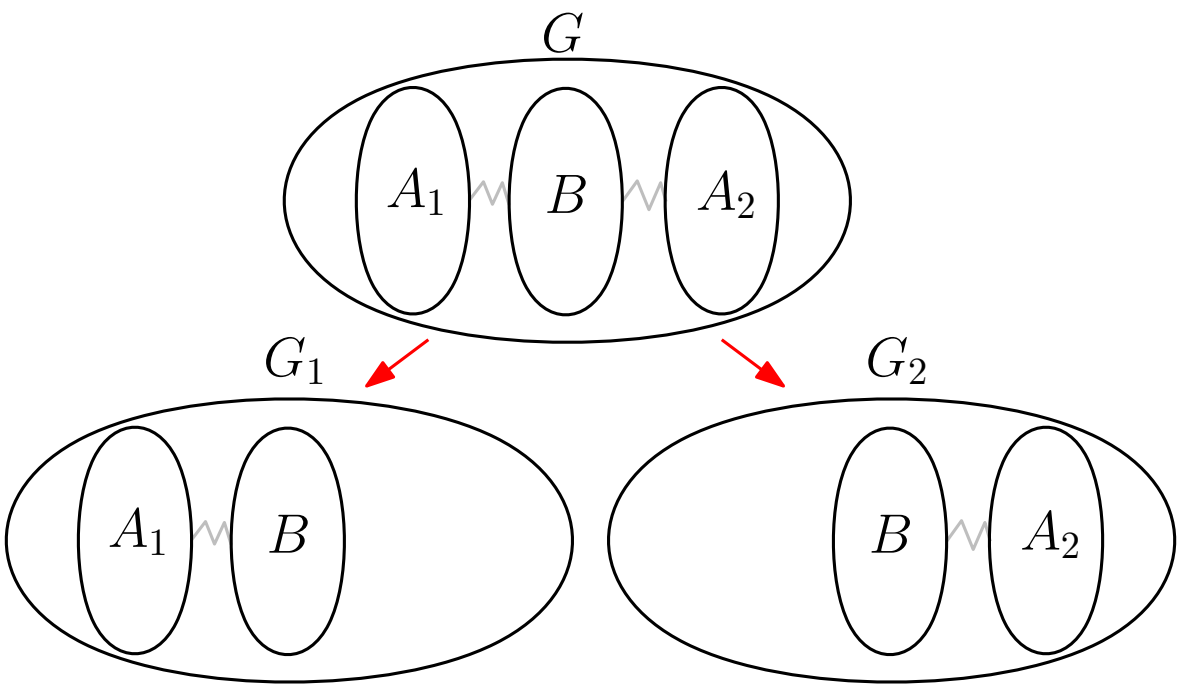


We know *so much*?

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 $\mathcal{P}$.

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 $\mathcal{P}$.

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 $\mathcal{P}$.

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 $\mathcal{P}$.

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, ...).

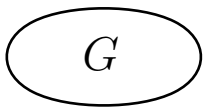
Meta-algorithm:

How to use it for algorithmic purposes?

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

Meta-algorithm:

1. Construct the decomposition tree:

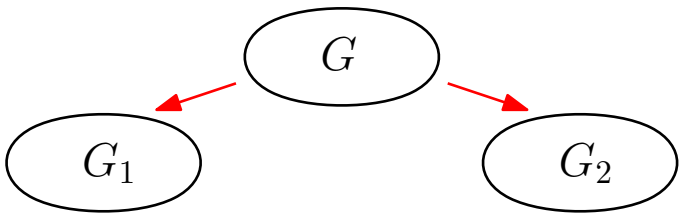


How to use it for algorithmic purposes?

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

Meta-algorithm:

1. Construct the decomposition tree:

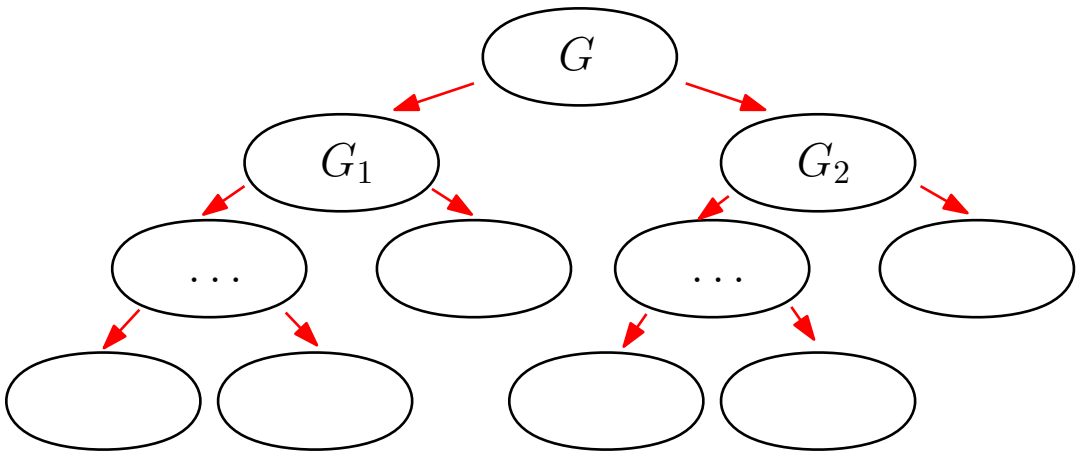


How to use it for algorithmic purposes?

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

Meta-algorithm:

1. Construct the decomposition tree:

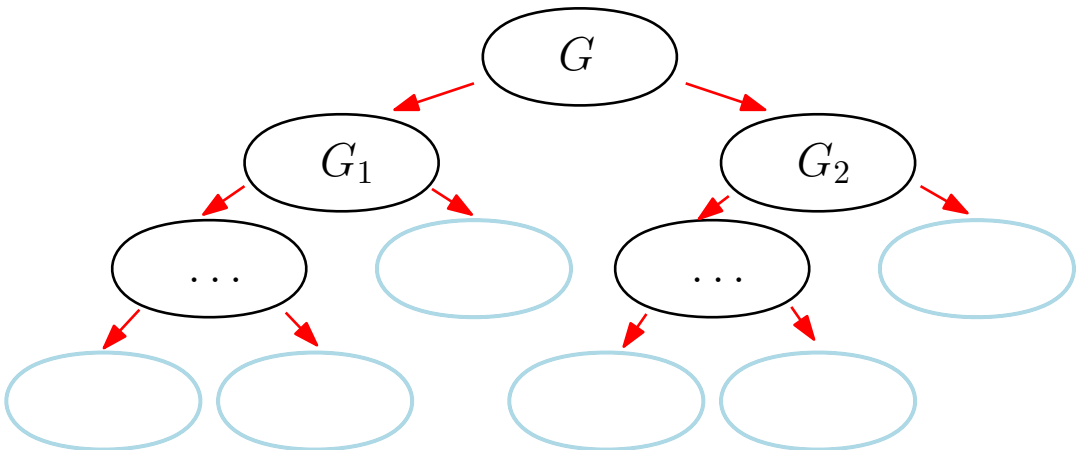


How to use it for algorithmic purposes?

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

Meta-algorithm:

1. Construct the decomposition tree:



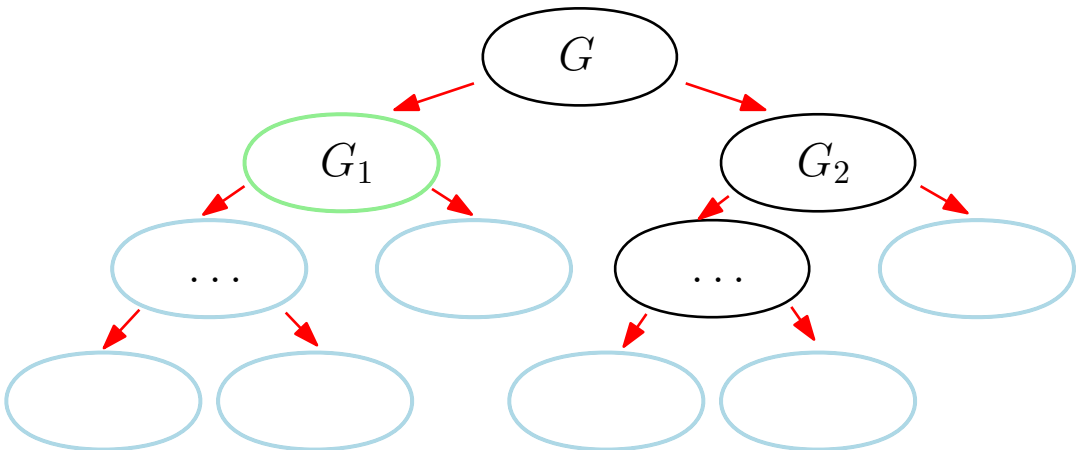
2. Compute what you want on the **leaves** ($\rightarrow$ *basic* graphs).

How to use it for algorithmic purposes?

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

Meta-algorithm:

1. Construct the decomposition tree:



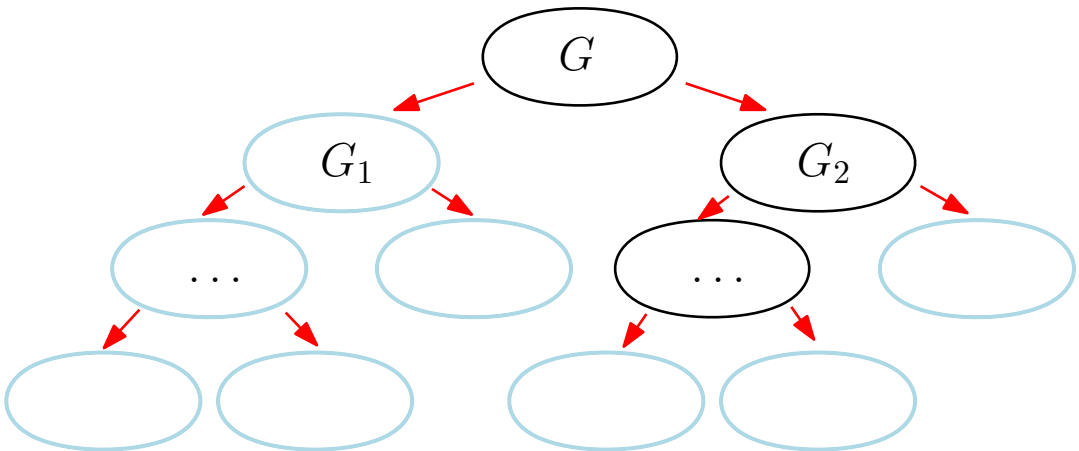
2. Compute what you want on the **leaves** ($\rightarrow$ *basic* graphs).
3. From bottom to top: **combine solutions for children** to get a solution for the father.

How to use it for algorithmic purposes?

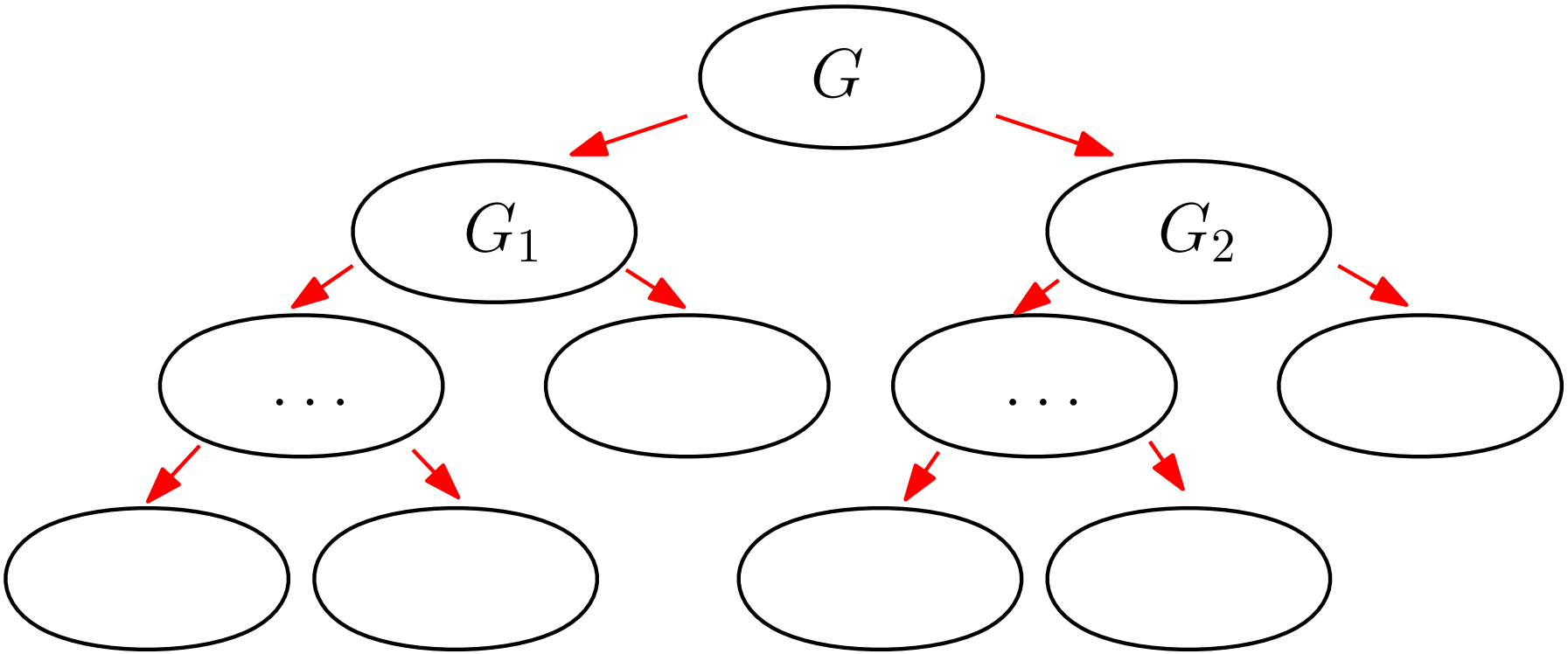
Want to compute something (coloring, largest stable set, ...).

Meta-algorithm:

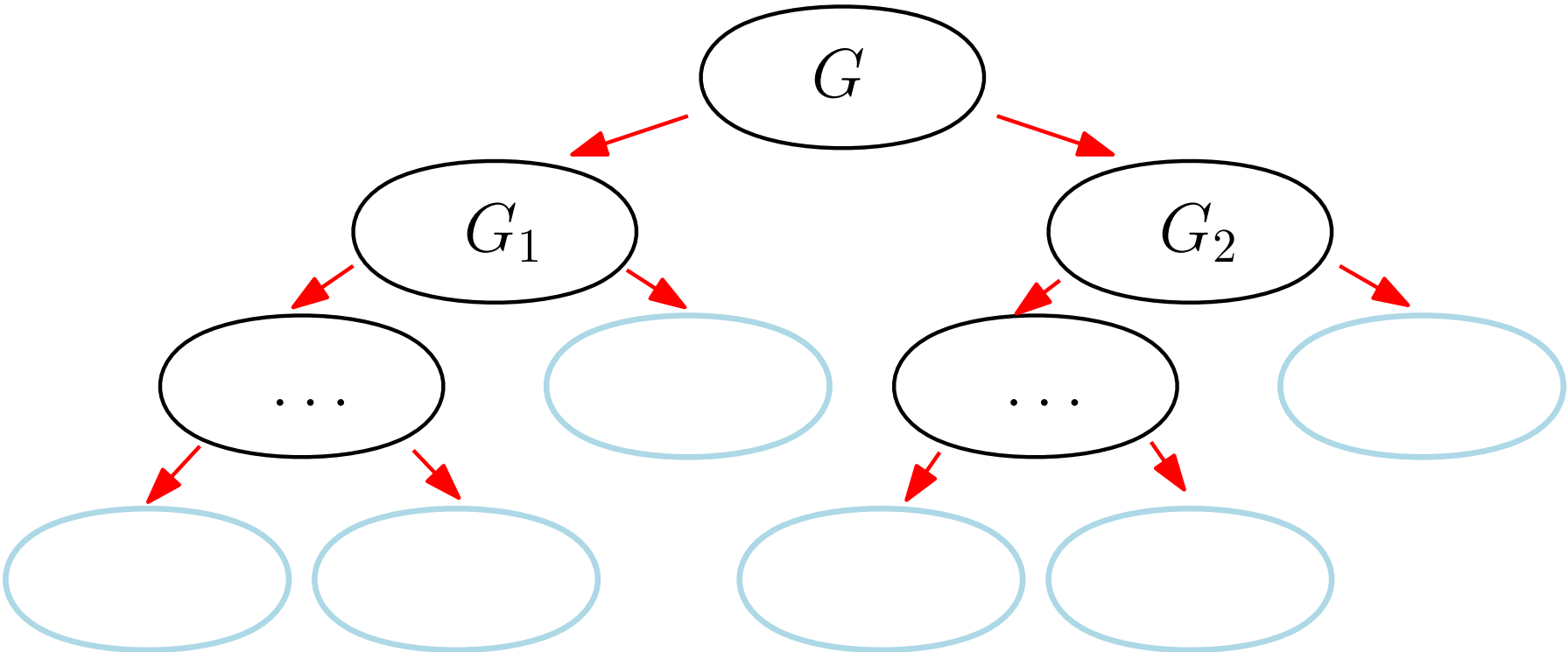
1. Construct the decomposition tree:



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.

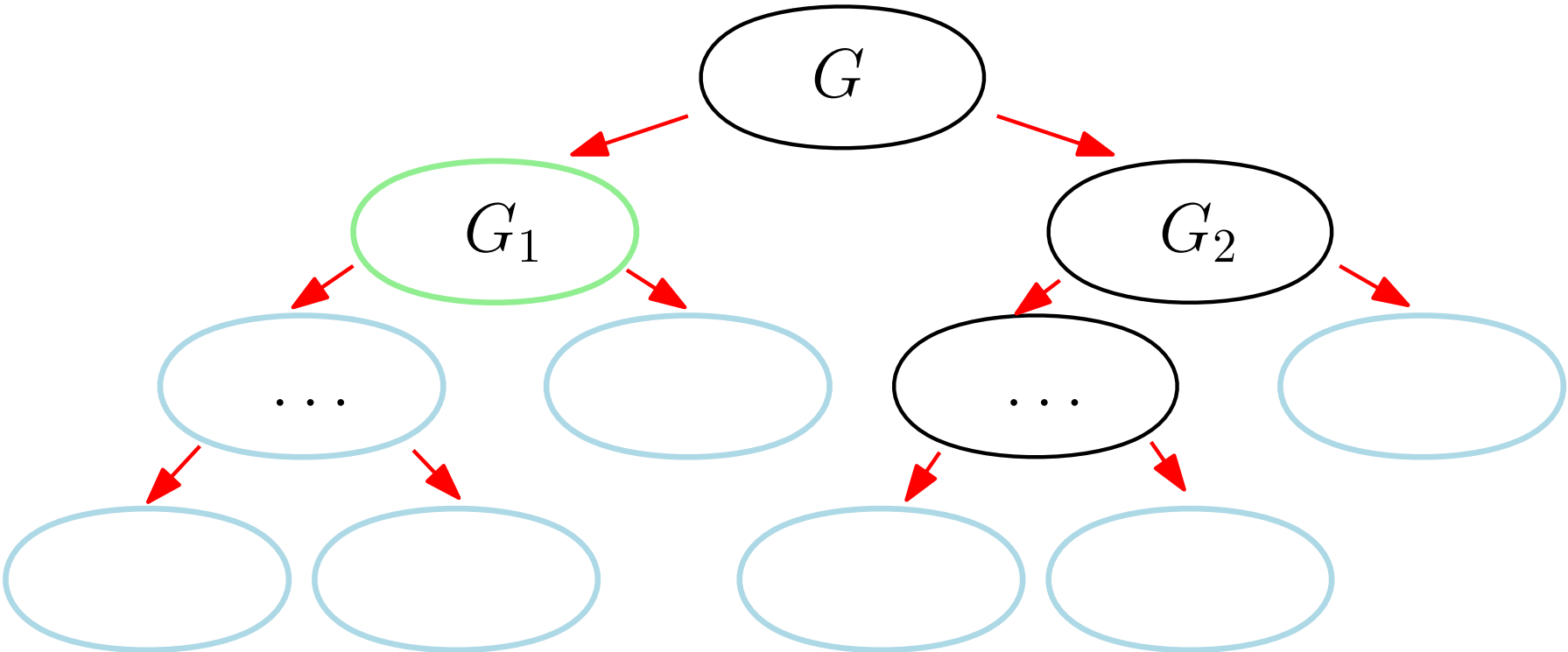


Hence four intermediate steps to reach:



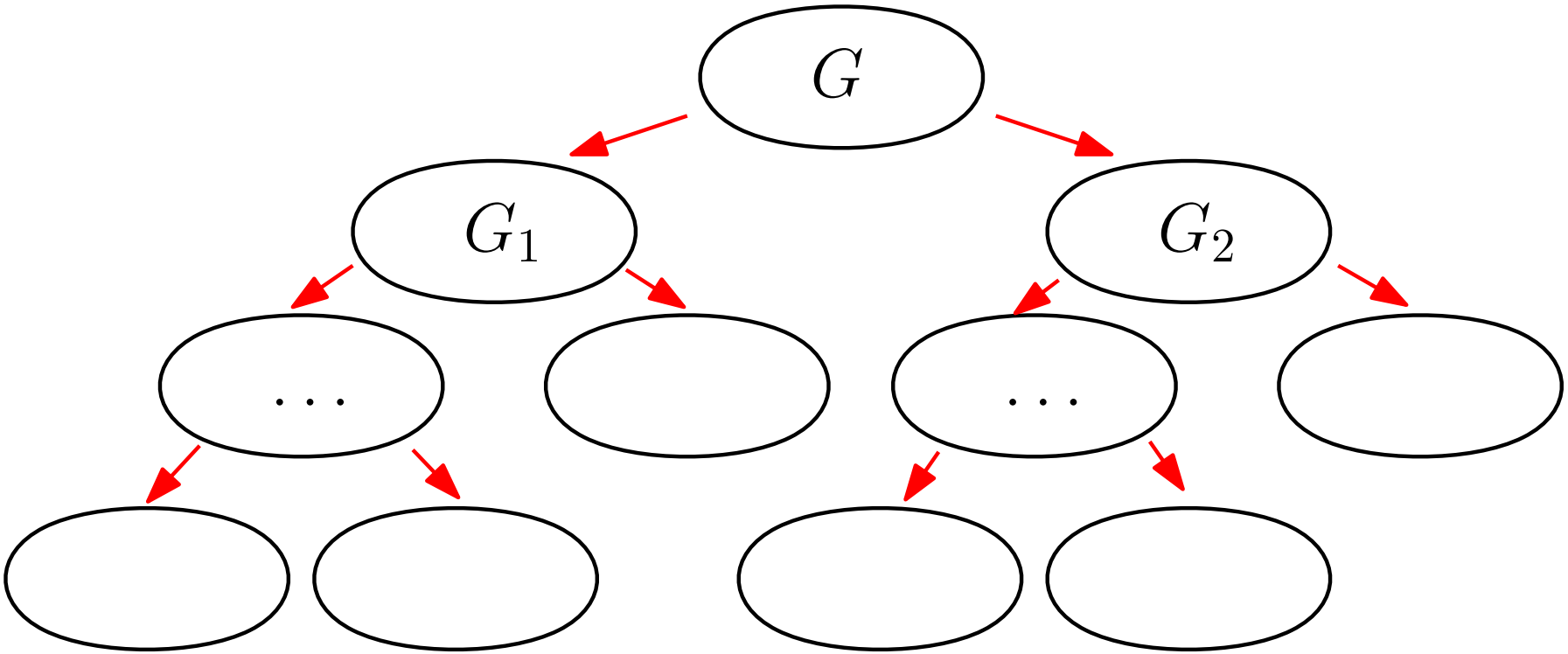
Hence four intermediate steps to reach:

- Know how to directly solve the problem on leaves.



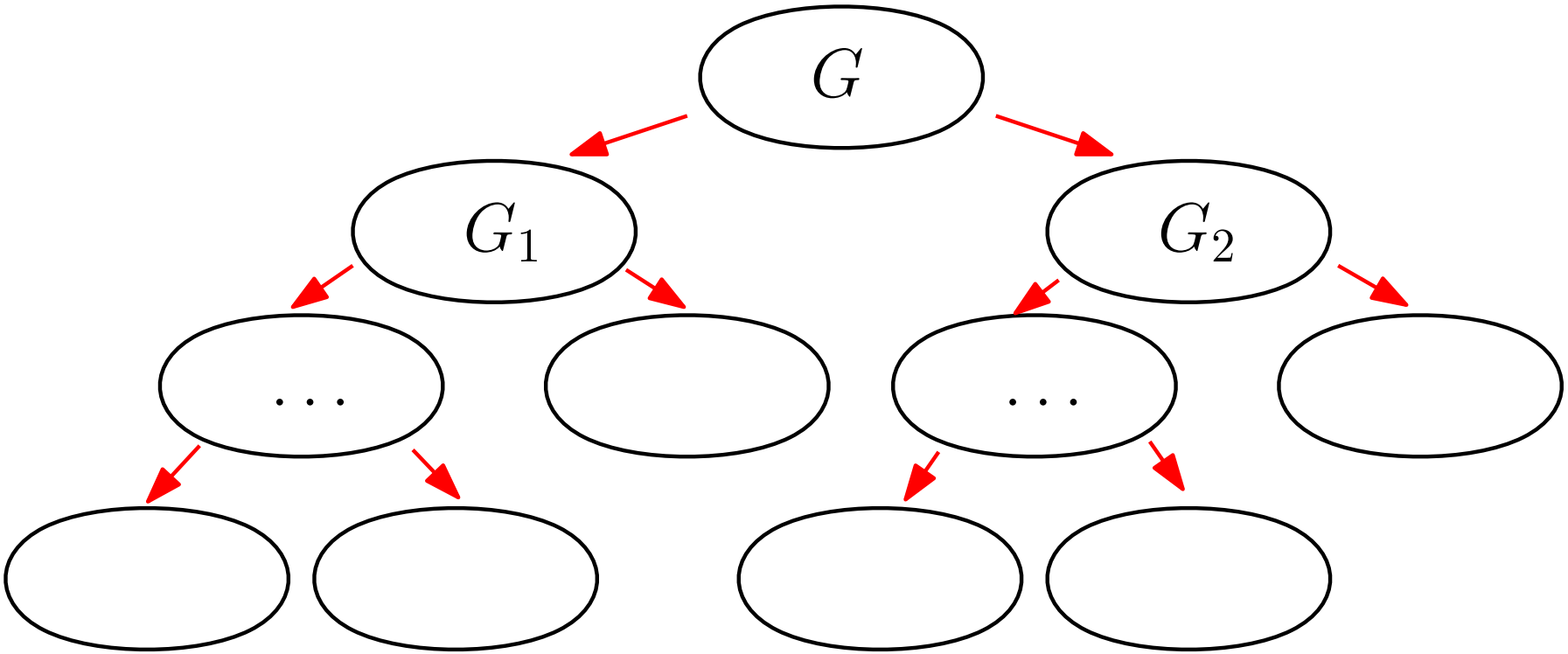
Hence four intermediate steps to reach:

- 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$^+$]

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:

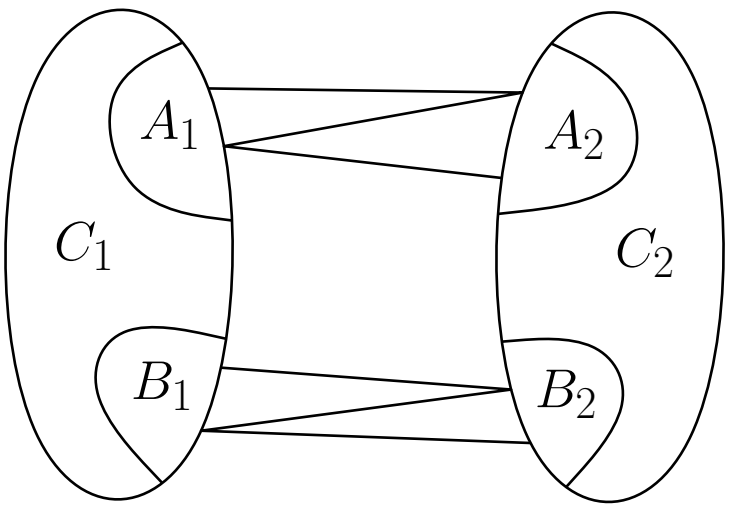
- G or $\overline{G}$ lies in one of the following classes: bipartite graphs, line graphs of a bipartite graph, double split.

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,



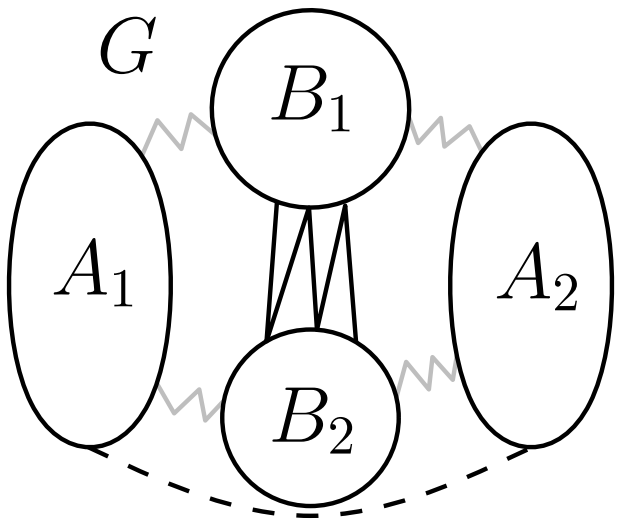
2-join

Decomposing perfect graphs

Decomposition theorem

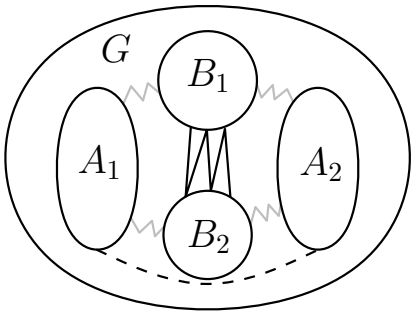
If G is perfect, then at least one of the following holds:

- 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).



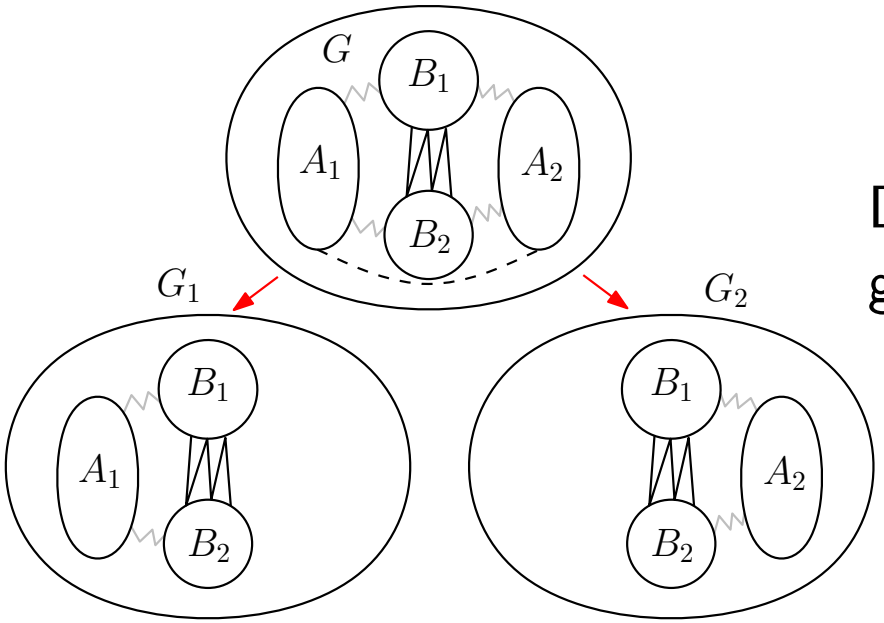
Skew partition

Our decomposition tree



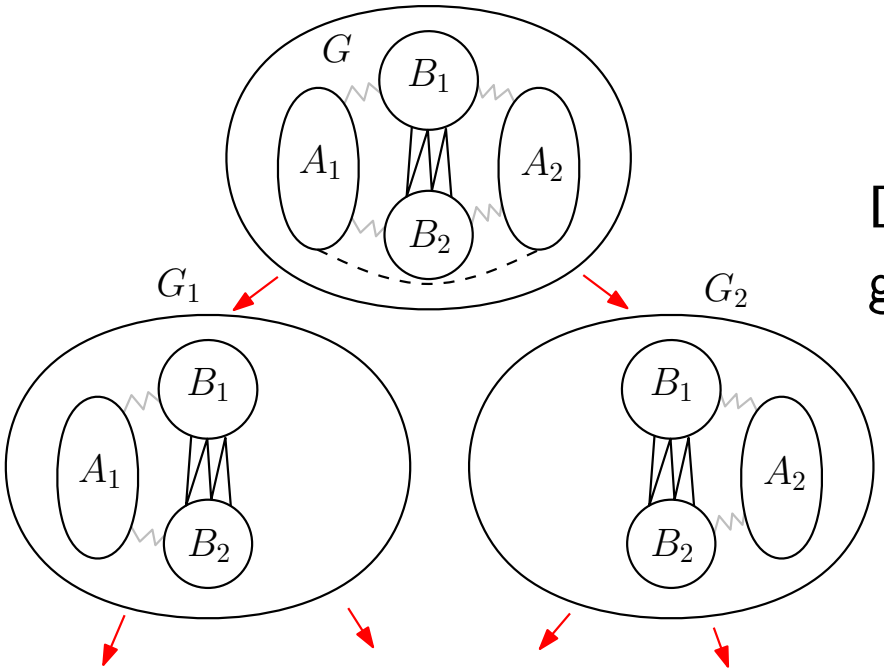
Decompose along BSP until the graph:

Our decomposition tree



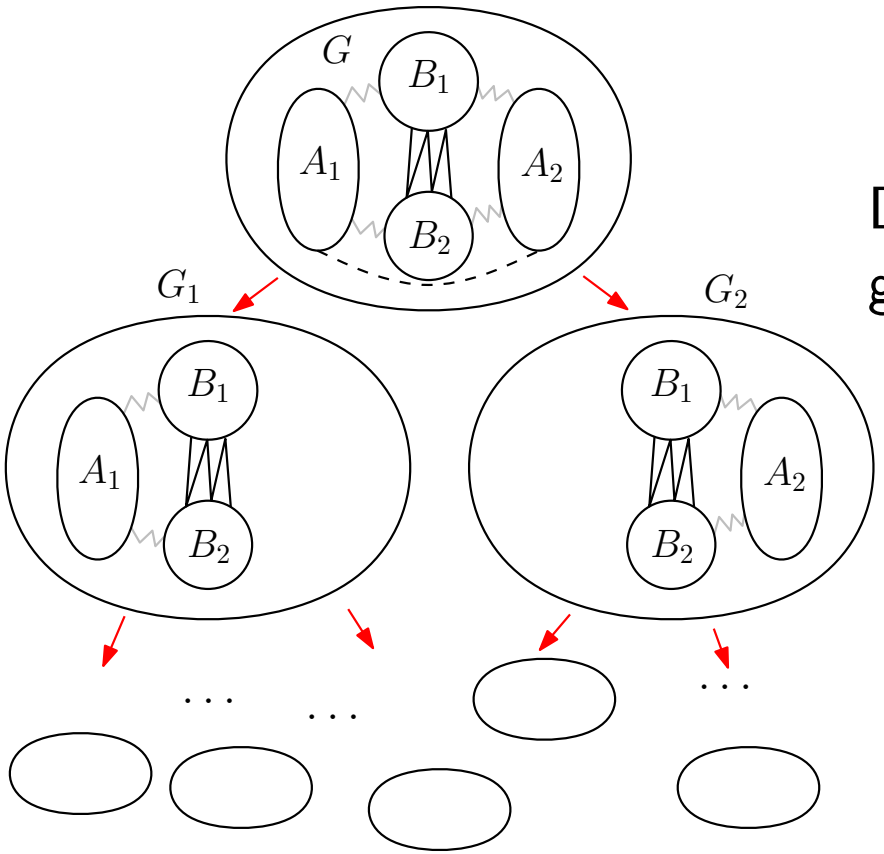
Decompose along BSP until the graph:

Our decomposition tree



Decompose along BSP until the graph:

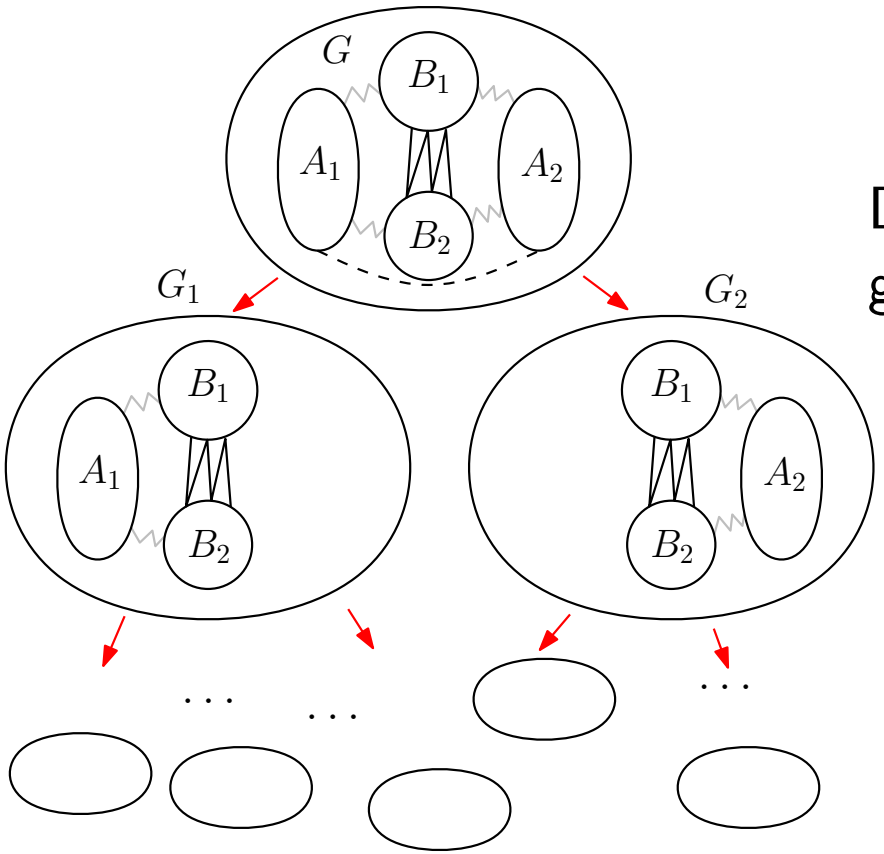
Our decomposition tree



Decompose along BSP until the graph:

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

Our decomposition tree

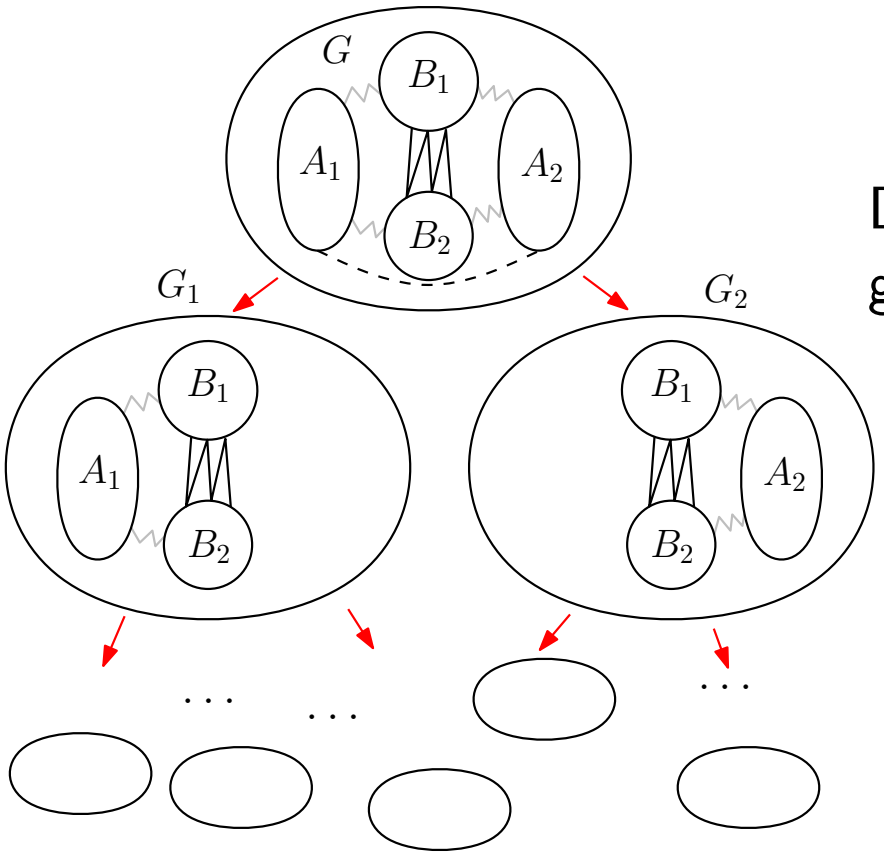


Decompose along BSP until the graph:

- 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

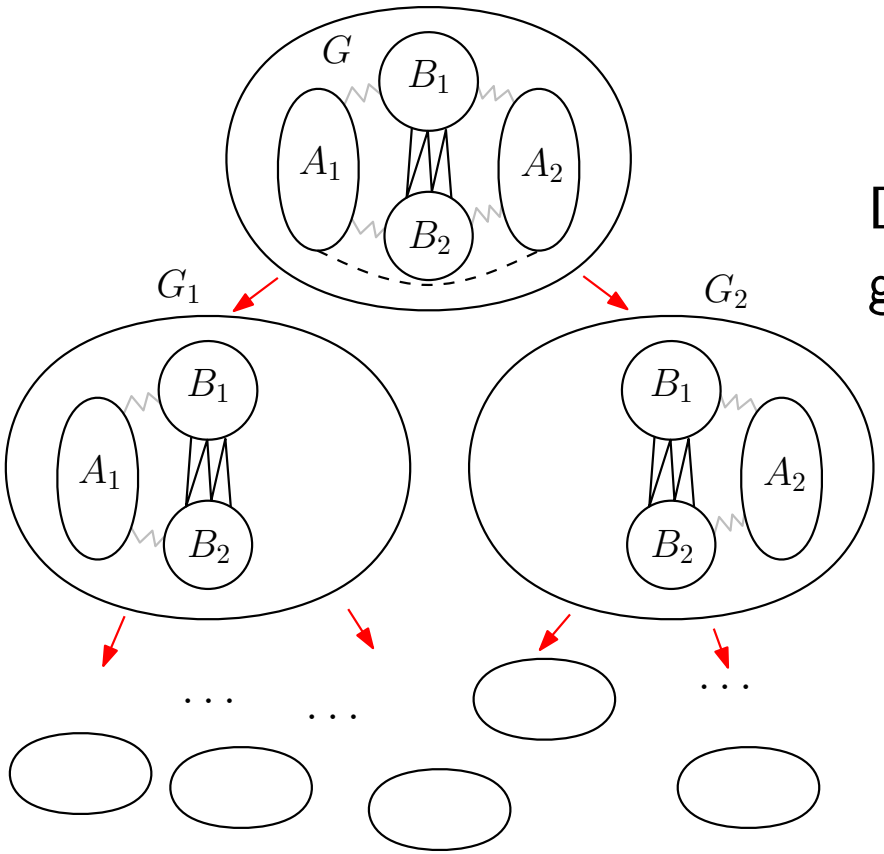


Decompose along BSP until the graph:

- 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 $\mathcal{A}_{k-1} \rightarrow \mathcal{O}(n^{\omega(G)^2})$

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

Outline

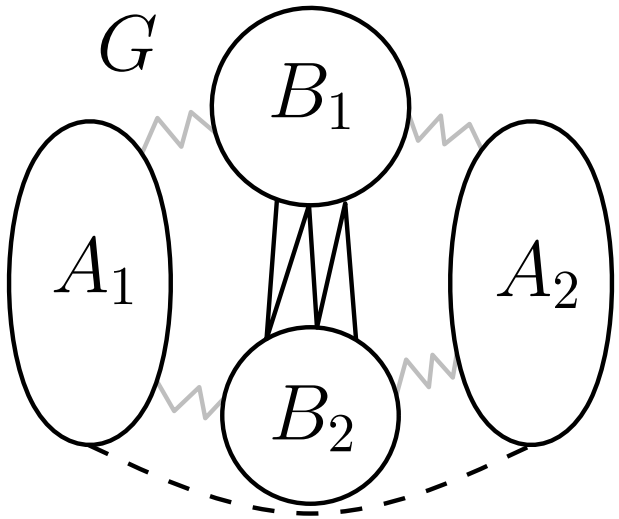
Four intermediate steps to reach:

- ✓ Know how to directly solve the problem on leaves
  $\rightarrow$ time $\mathcal{O}(n^{\max(7,\omega(G)^2)})$
- 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

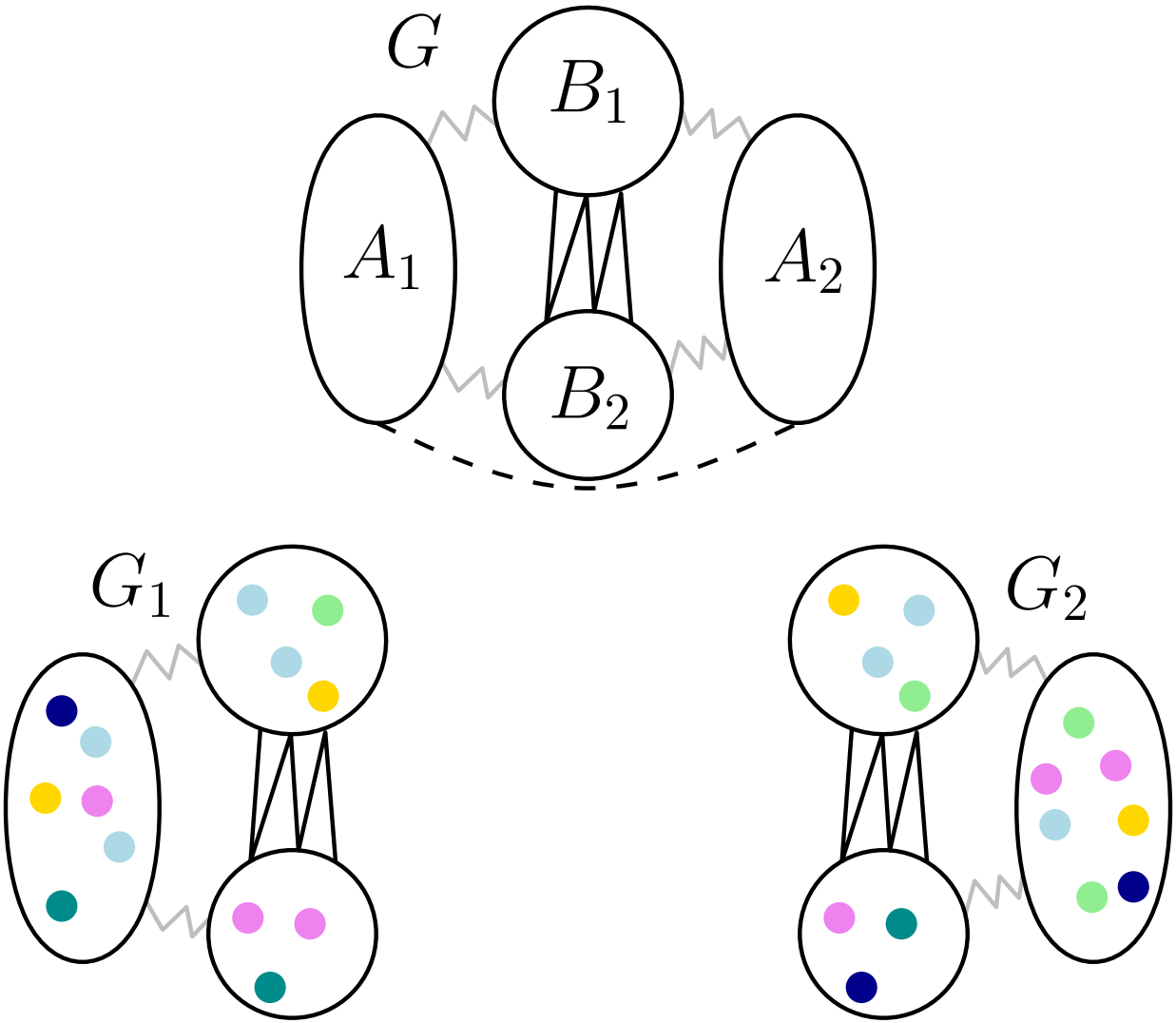
How to combine solutions?

Problem when gluing solutions



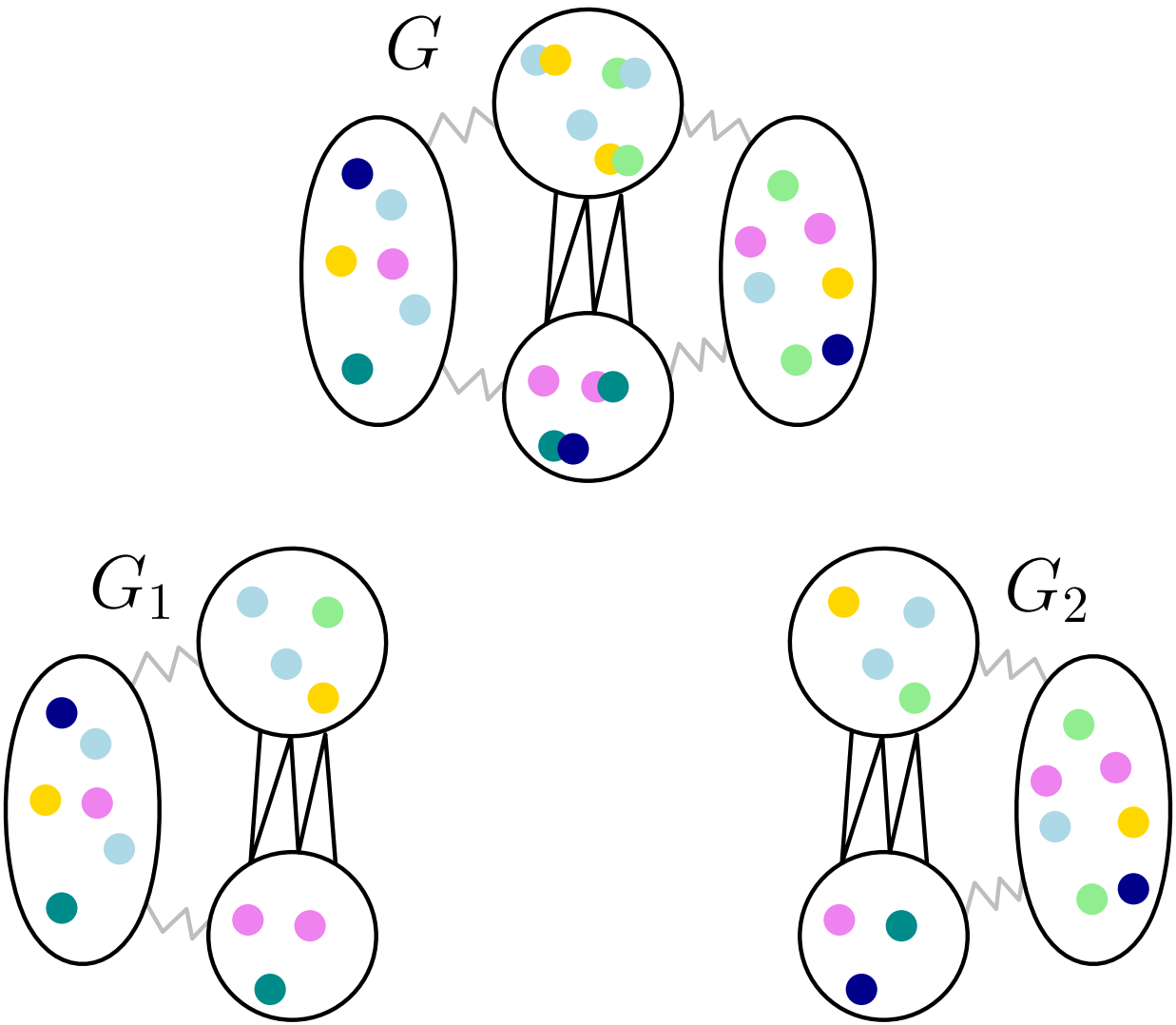
How to combine solutions?

Problem when gluing solutions



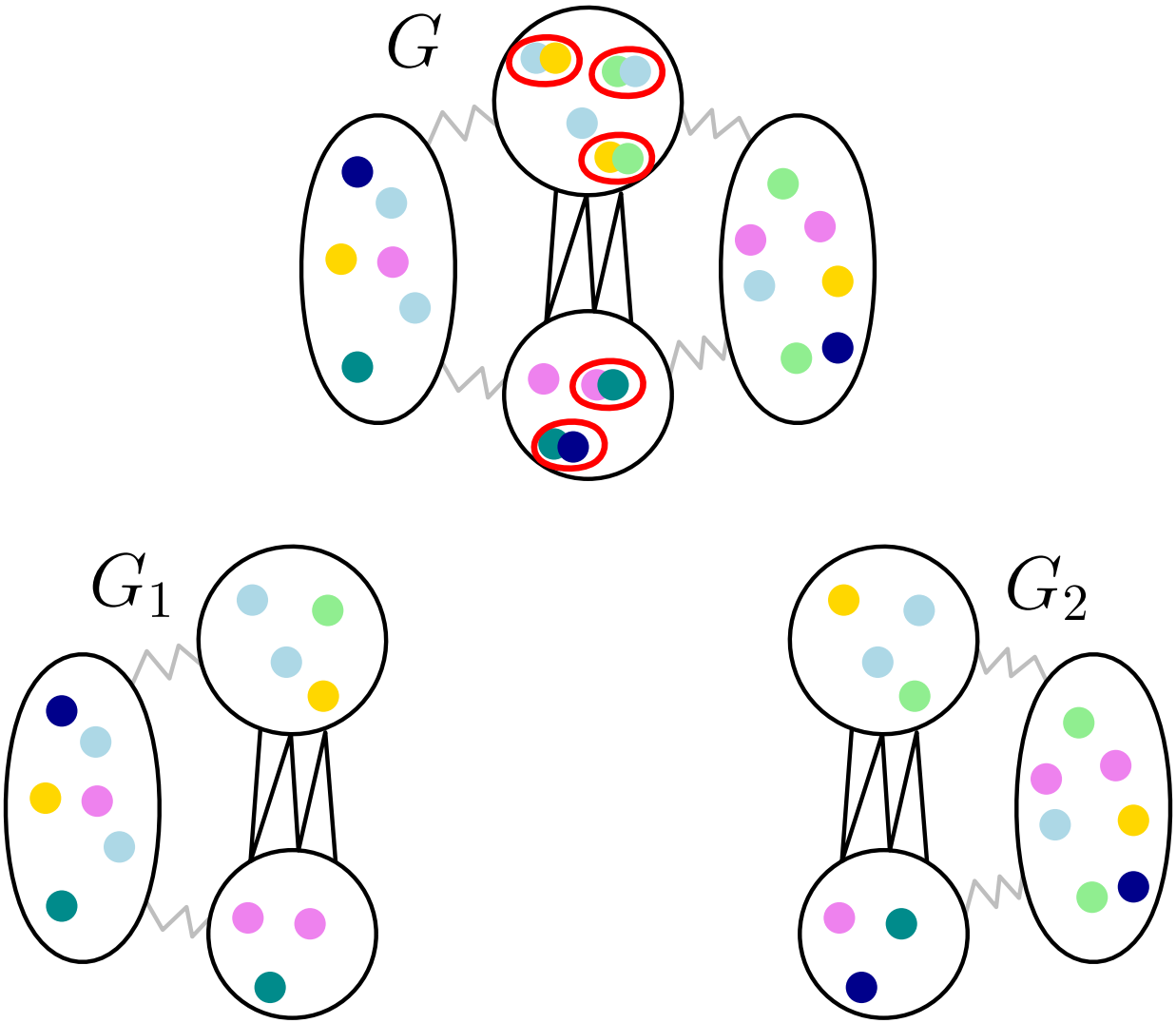
How to combine solutions?

Problem when gluing solutions



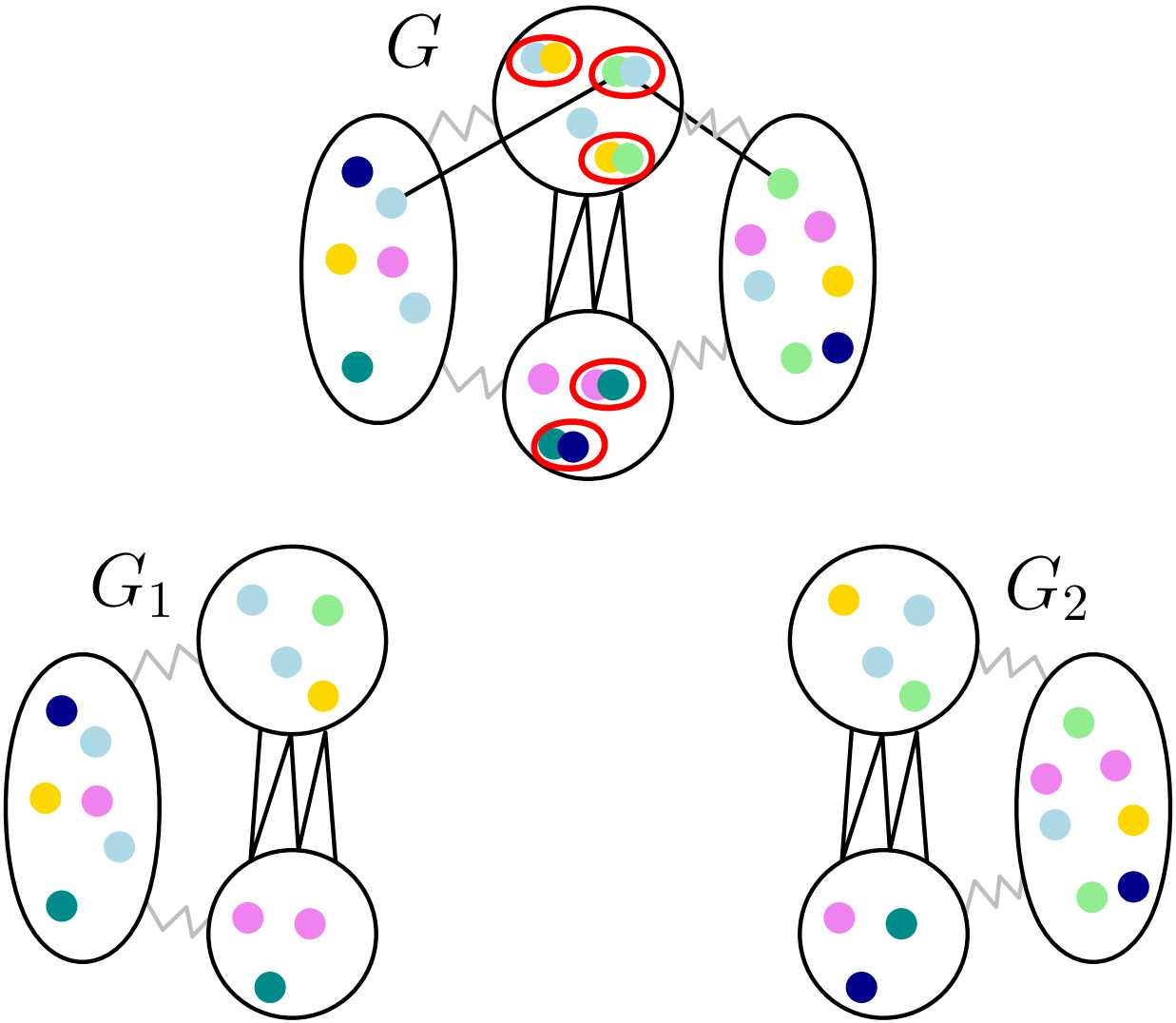
How to combine solutions?

Problem when gluing solutions



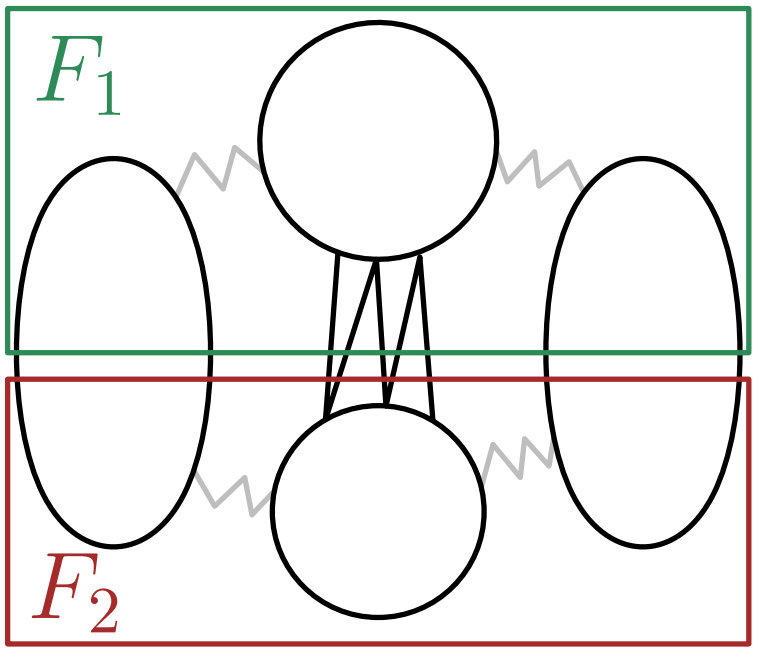
How to combine solutions?

Problem when gluing solutions



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

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



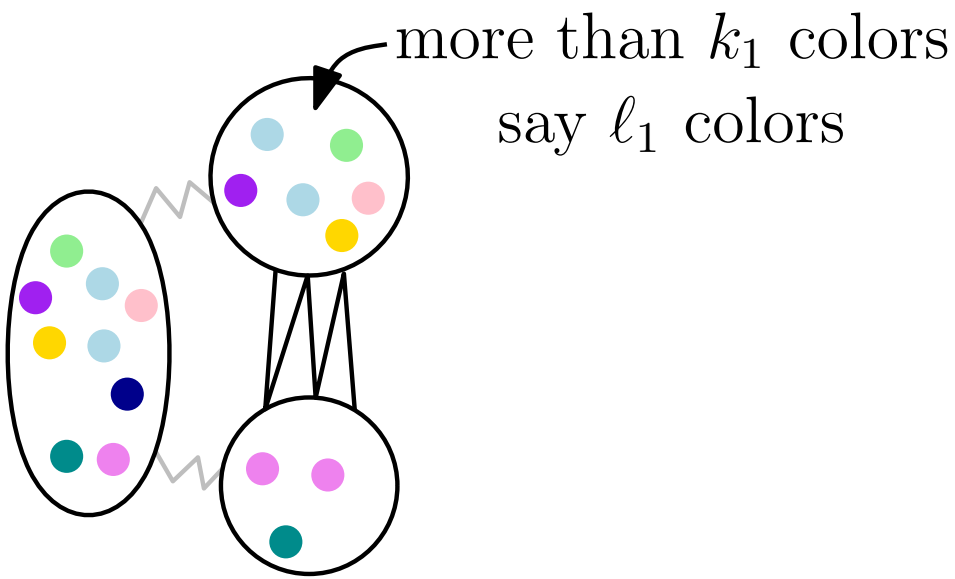
Call $\mathcal{A}_{k-1}$ on F_1 and F_2. This gives a proper $\omega(G)$-coloring of G.

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

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

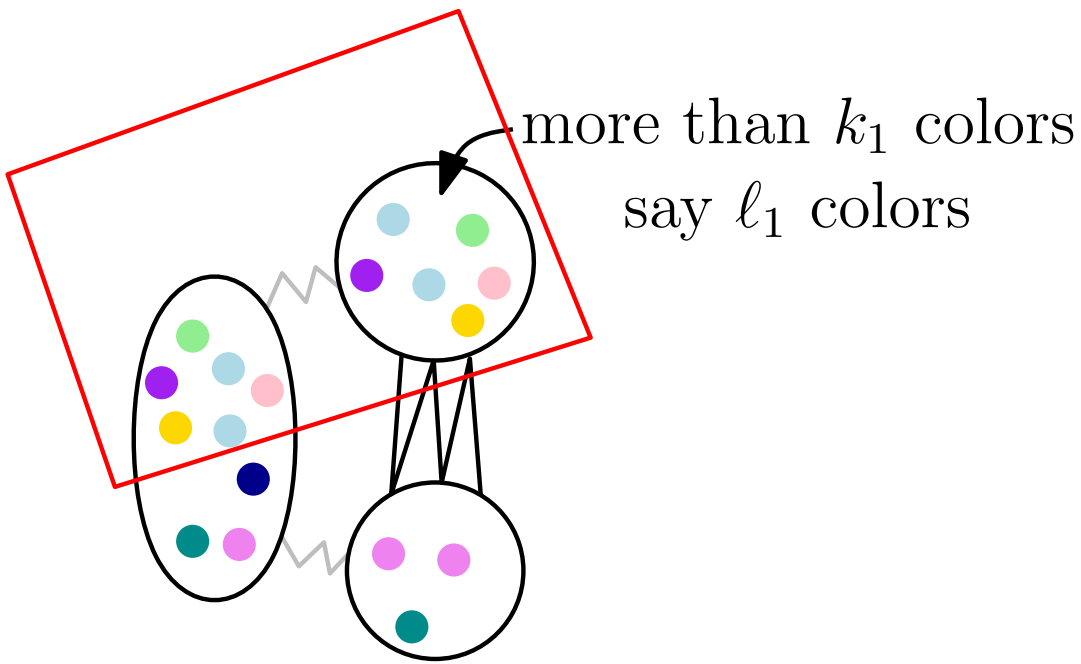
How to find such a partition (F_1, F_2)?

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



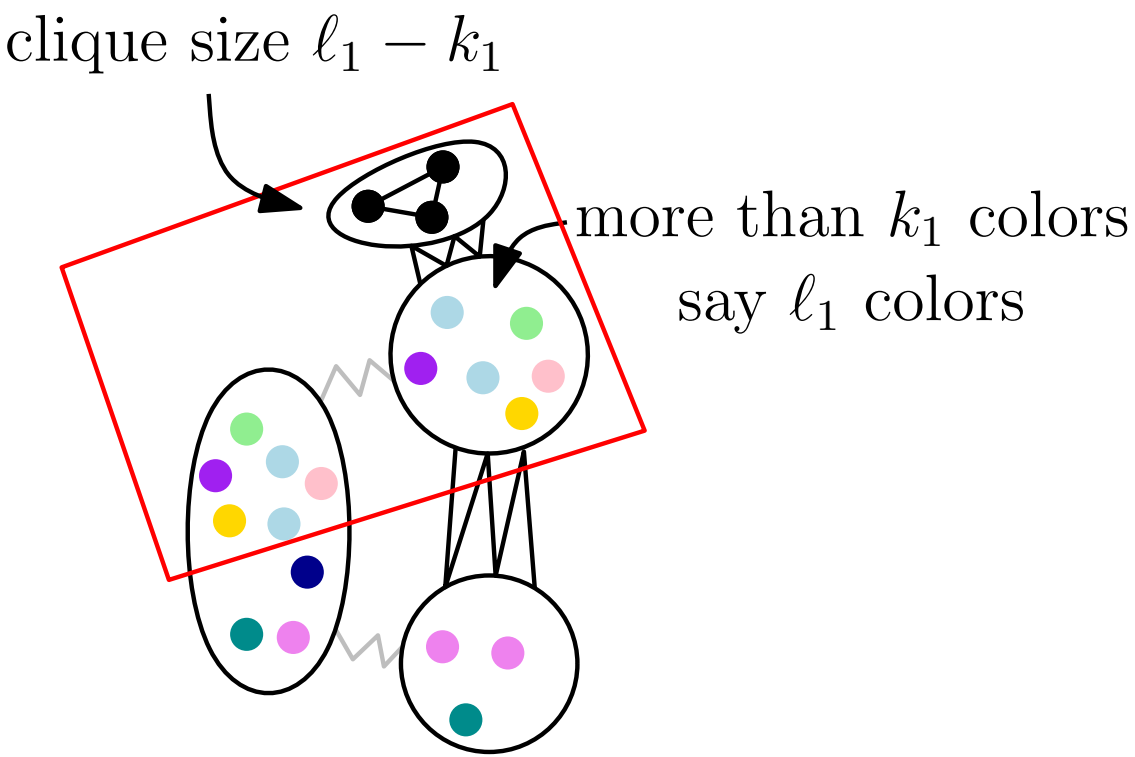
How to find such a partition (F_1, F_2)?

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



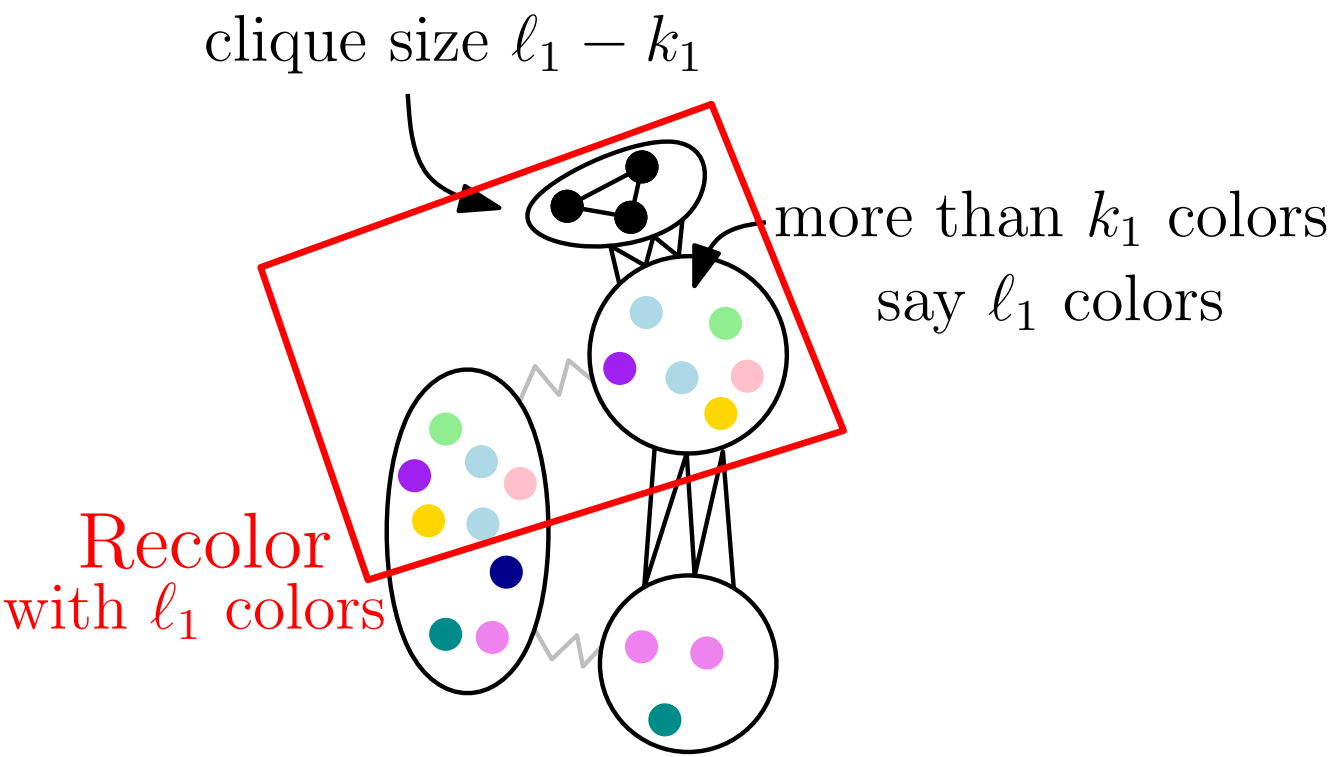
How to find such a partition (F_1, F_2)?

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



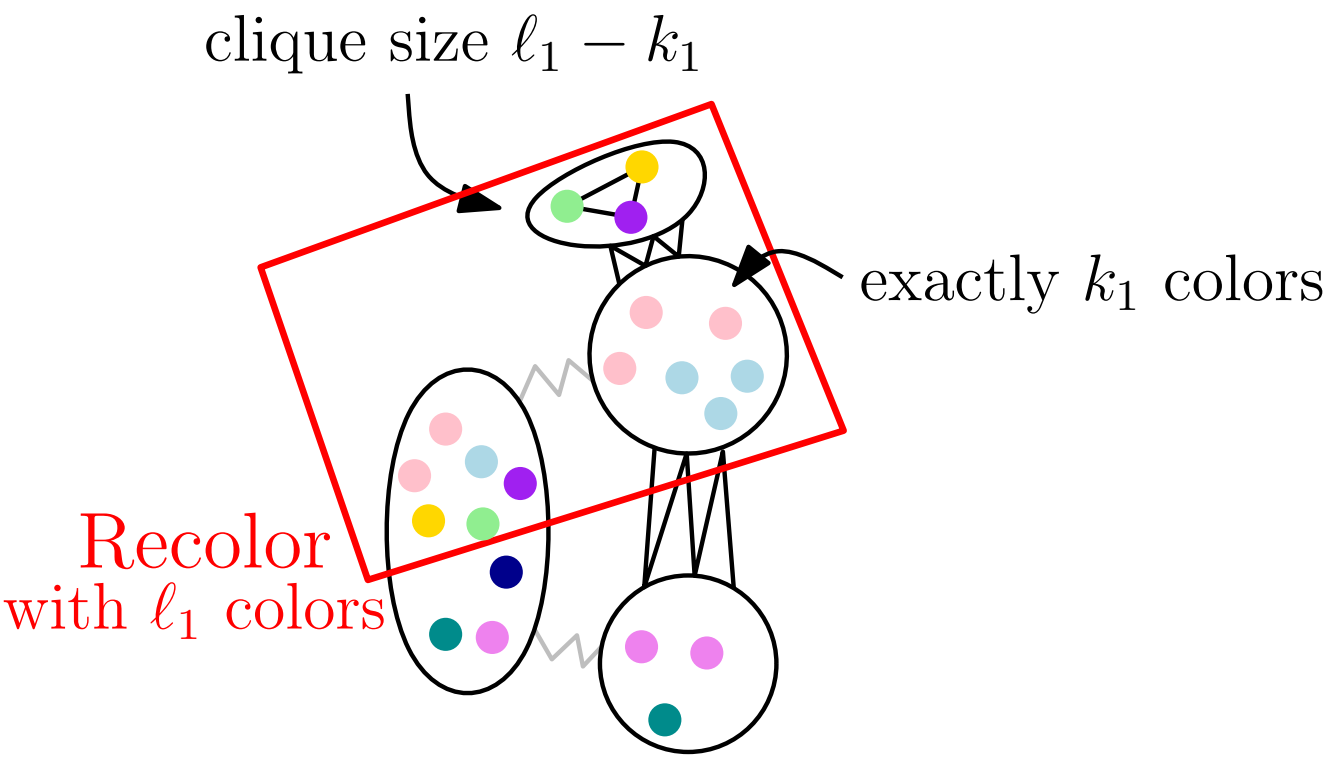
How to find such a partition (F_1, F_2)?

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



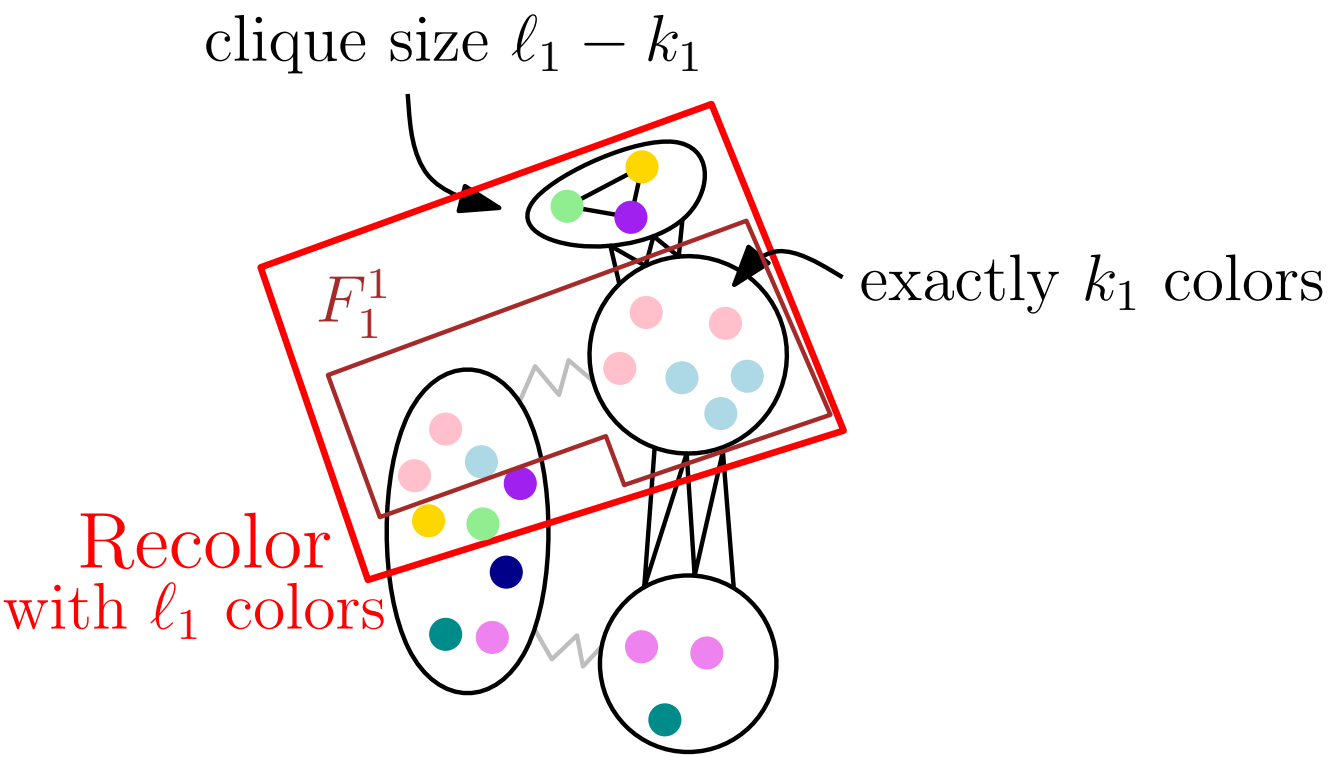
How to find such a partition (F_1, F_2)?

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



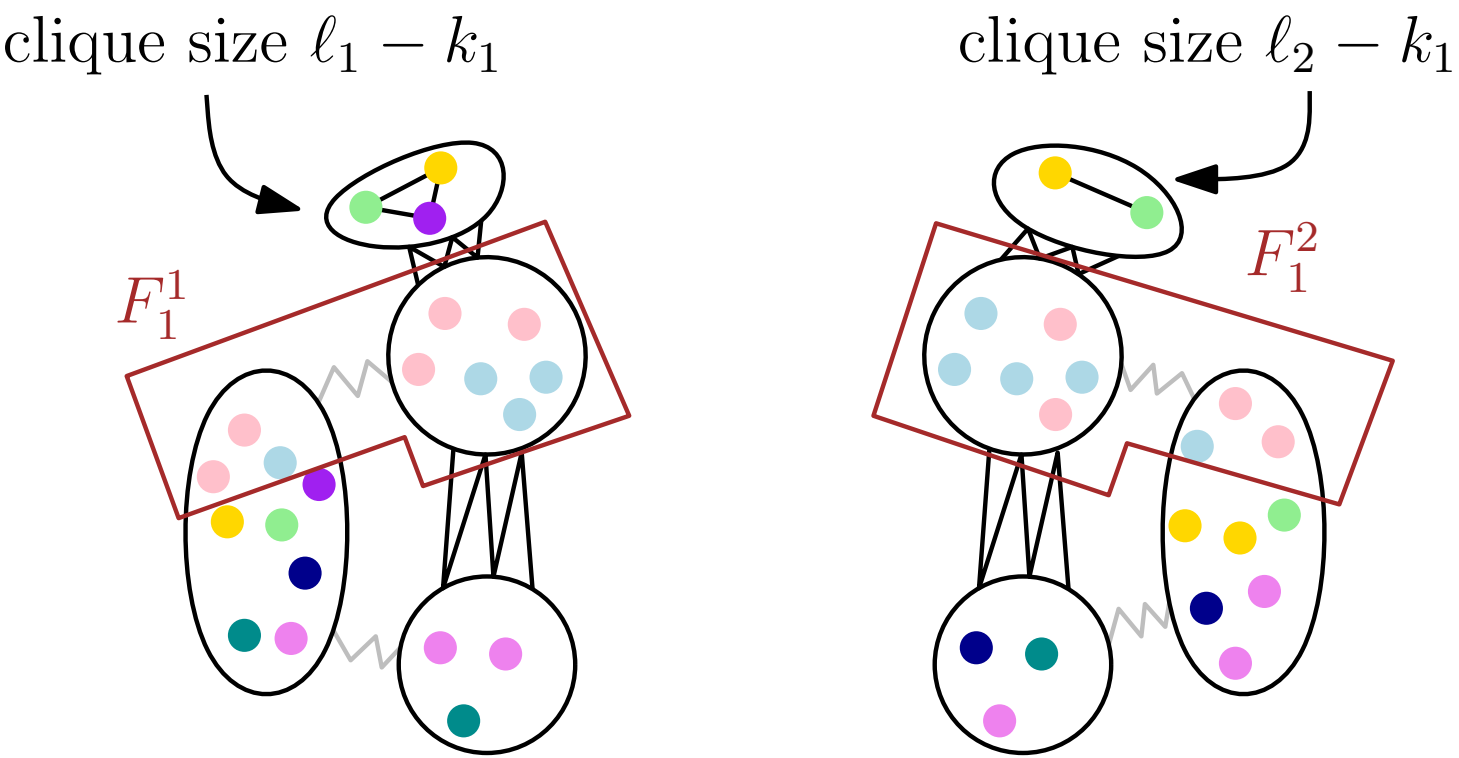
How to find such a partition (F_1, F_2)?

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



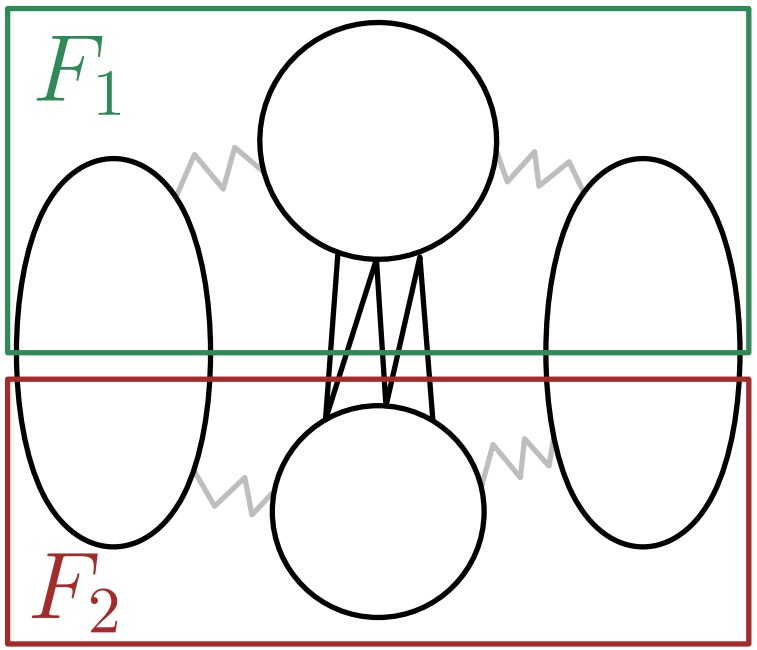
How to find such a partition (F_1, F_2)?

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 $F_1 = F_1^1 \cup F_1^2$.



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

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



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

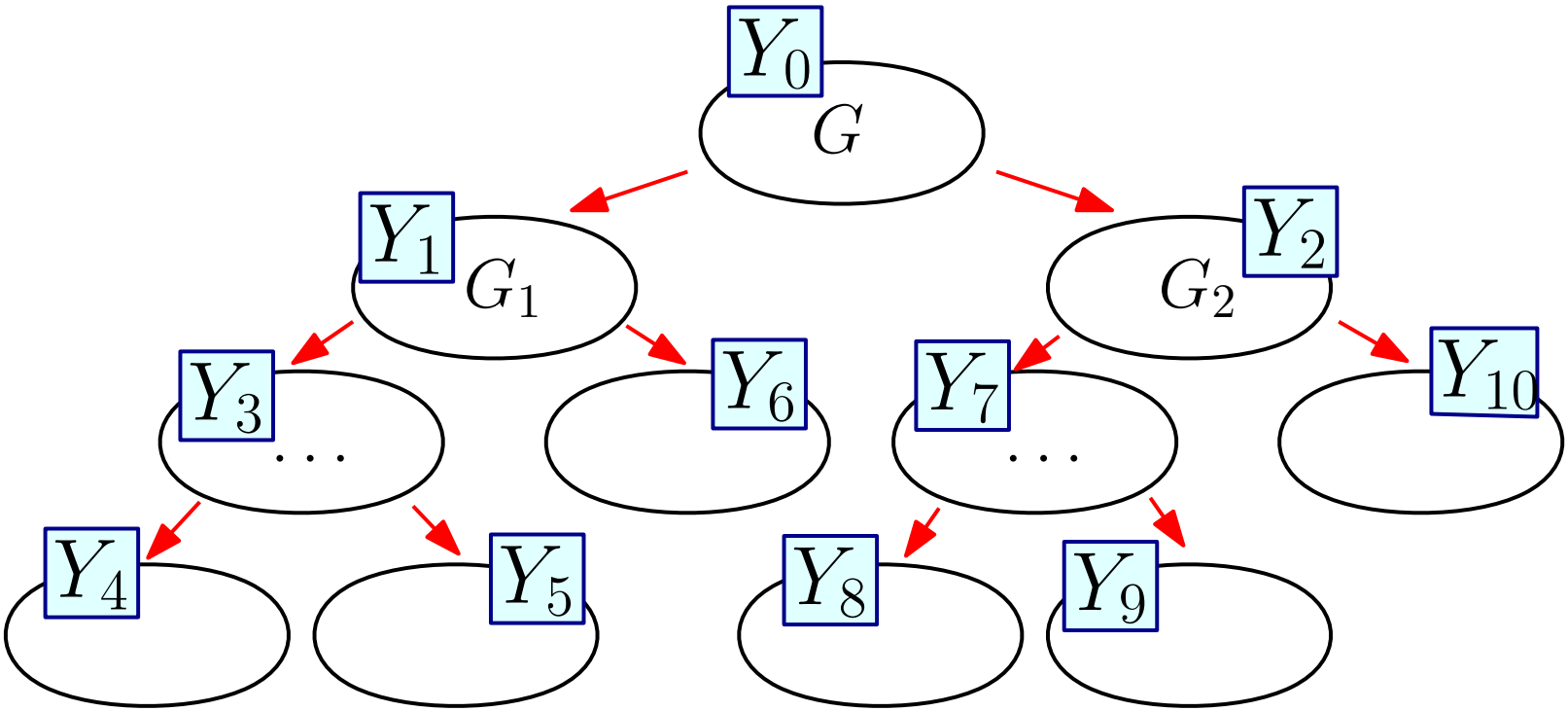
Outline

Four intermediate steps to reach:

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

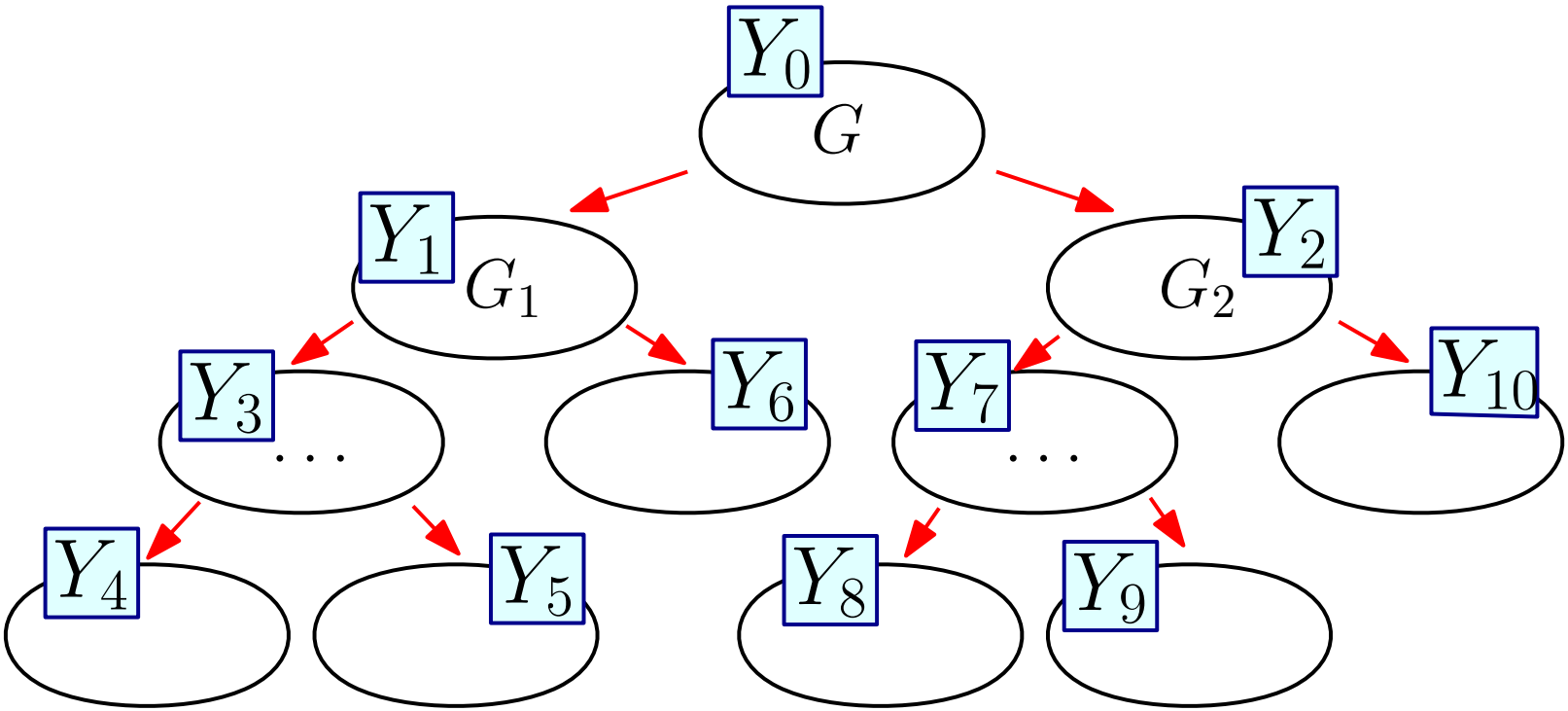
How to bound the size of the tree?

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



How to bound the size of the tree?

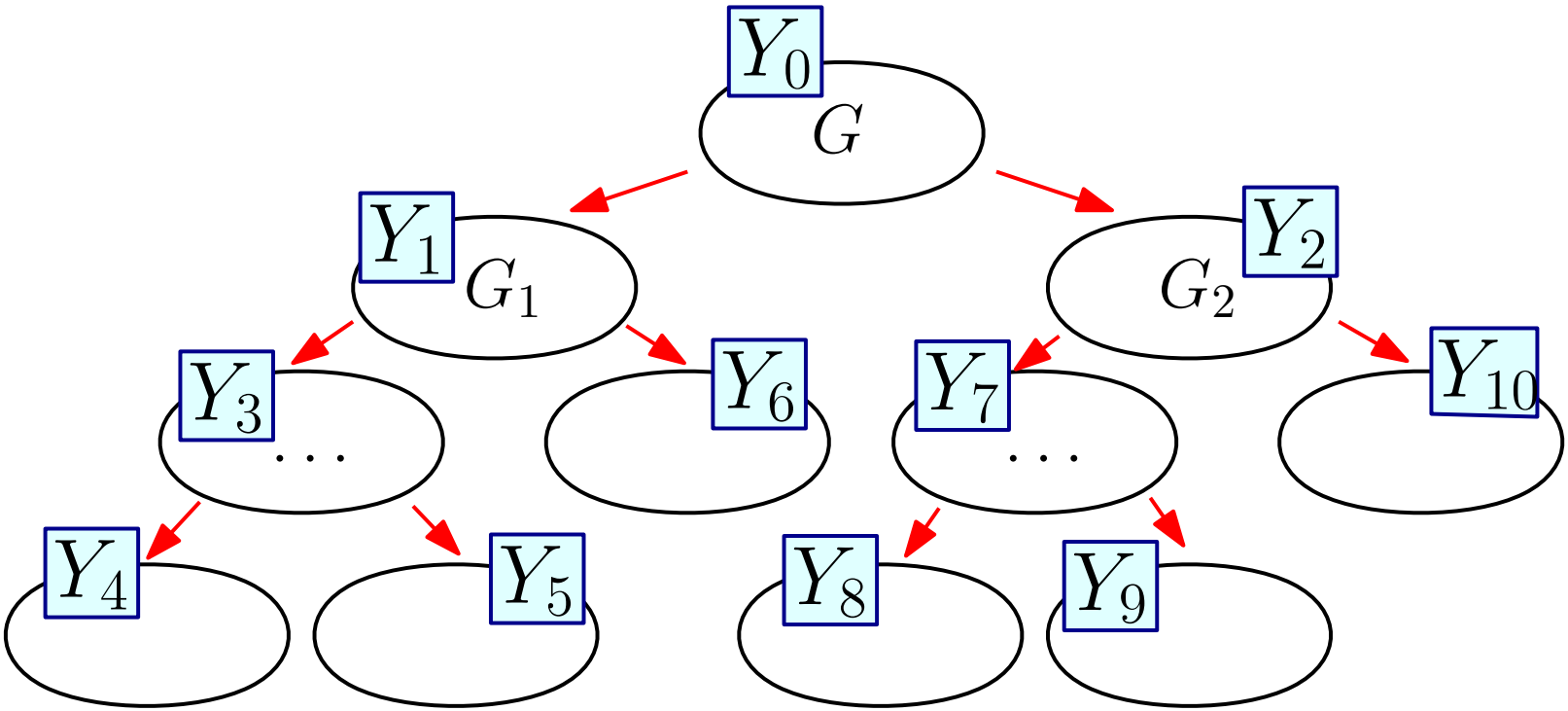
Label each node of the tree with some well-chosen $Y \subseteq V(G)$:



- Each label is different from every other labels,

How to bound the size of the tree?

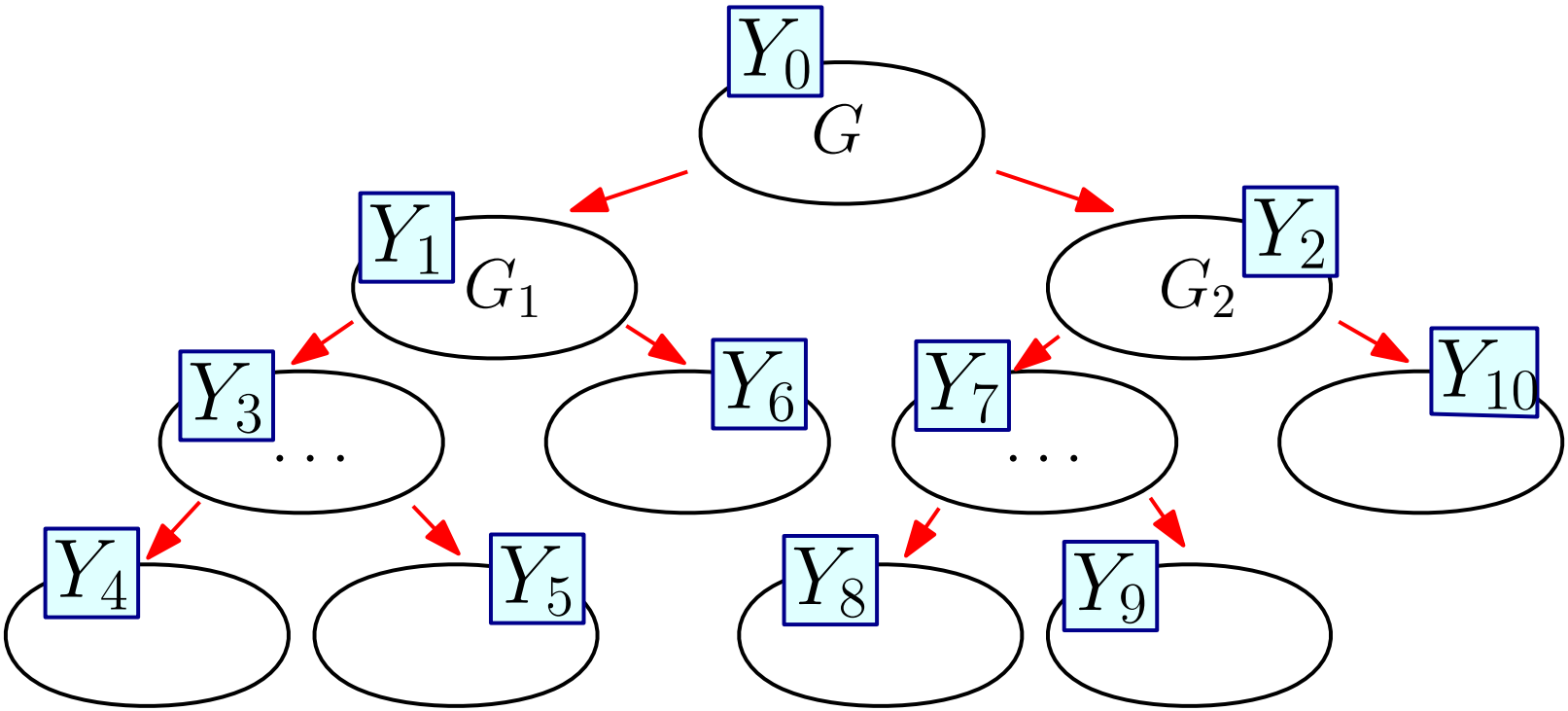
Label each node of the tree with some well-chosen $Y \subseteq V(G)$:



- 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?

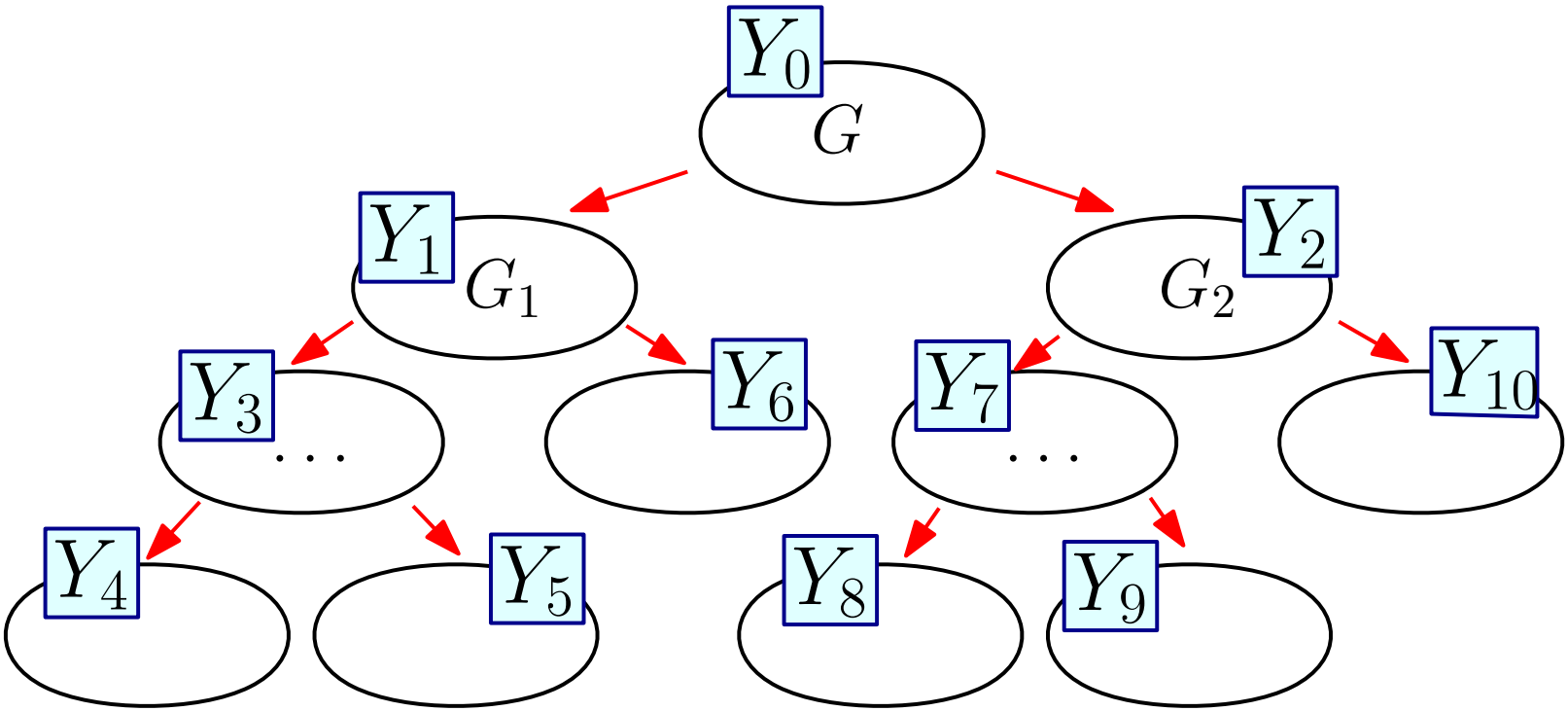
Label each node of the tree with some well-chosen $Y \subseteq V(G)$:



- 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.

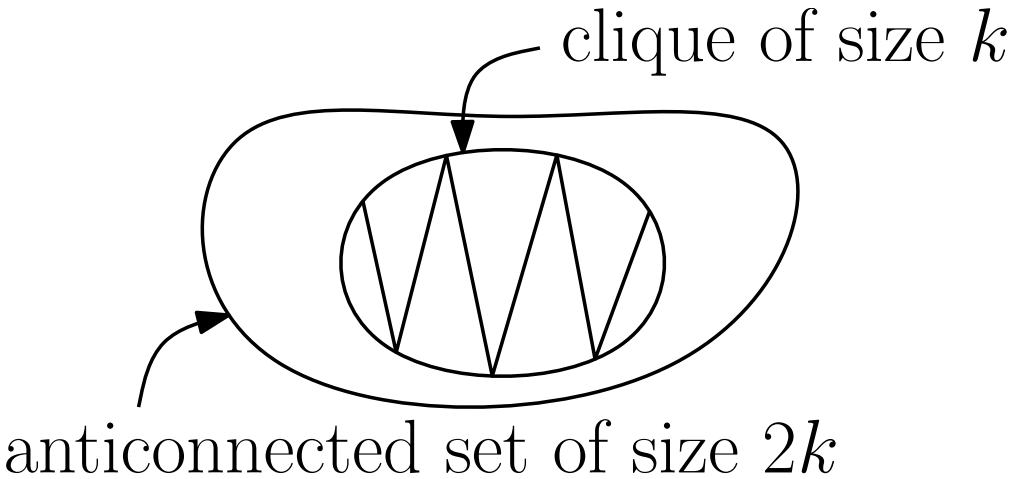
$\Rightarrow$ Bounds the number of nodes by a polynomial.

Key ingredient: k-pellet

Definition: k-pellet

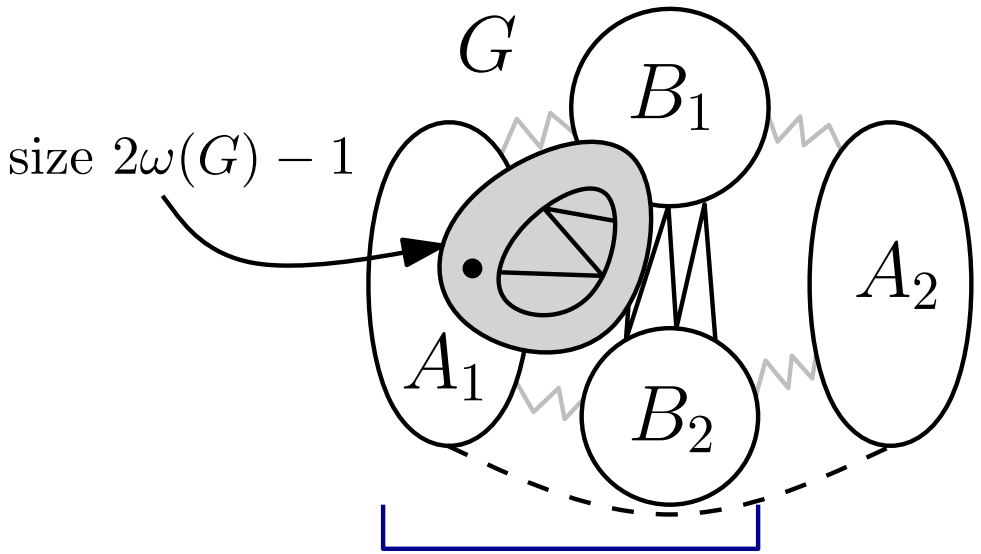
A subset $Y \subseteq V(G)$ is a k-pellet if

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



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

At each node decomposing along a BSP, arbitrarily choose an $\omega(G)$-pellet that will be broken.

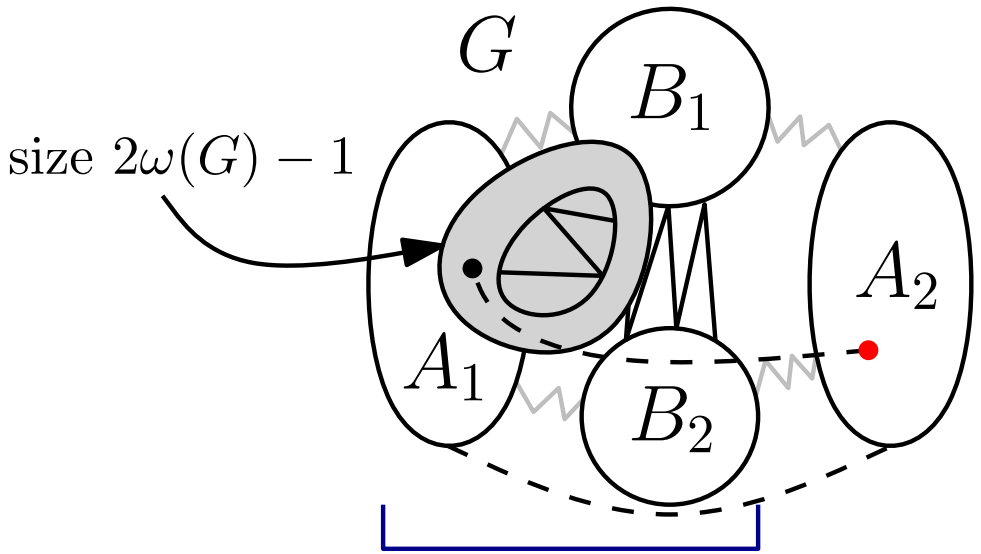


Definition: k-pellet

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

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

At each node decomposing along a BSP, arbitrarily choose an $\omega(G)$-pellet that will be broken.



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 $|Y| = 2k$.

Unique labeling

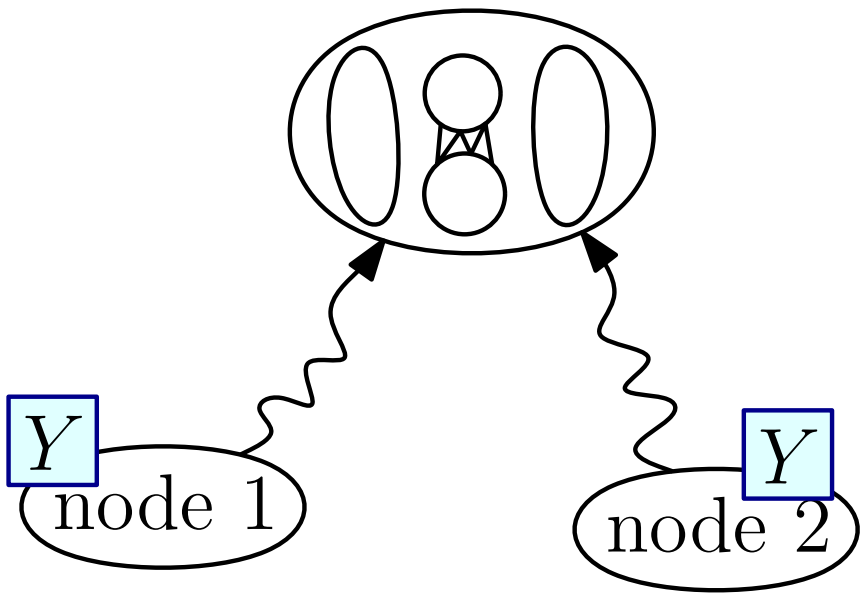
Two nodes getting the same label Y?

Unique labeling

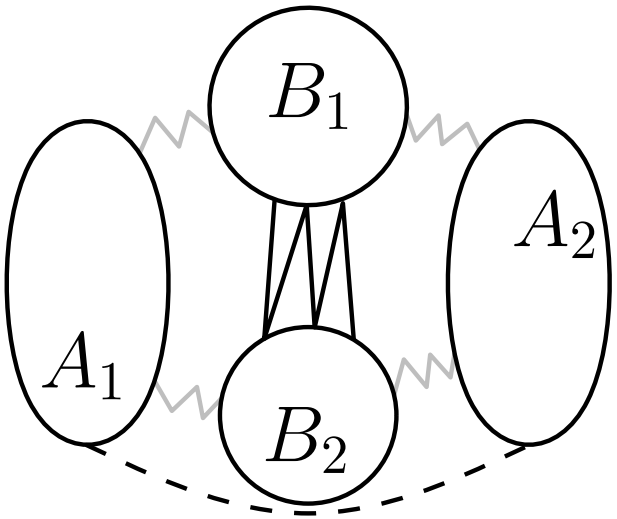
Two nodes getting the same label Y?

Unique labeling

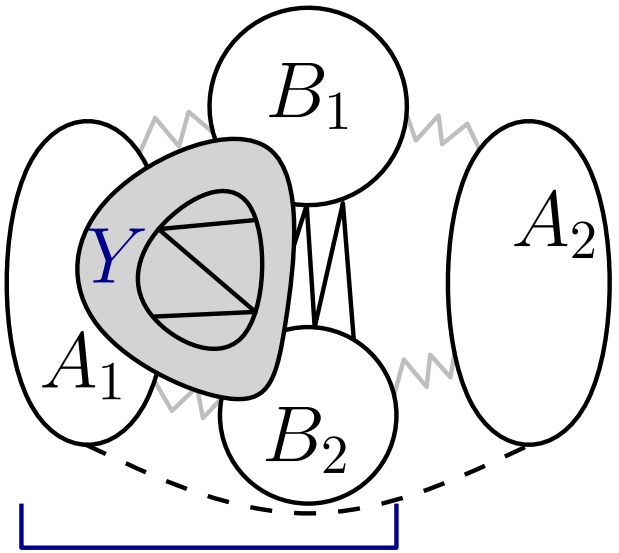
Two nodes getting the same label Y?



Where is Y?

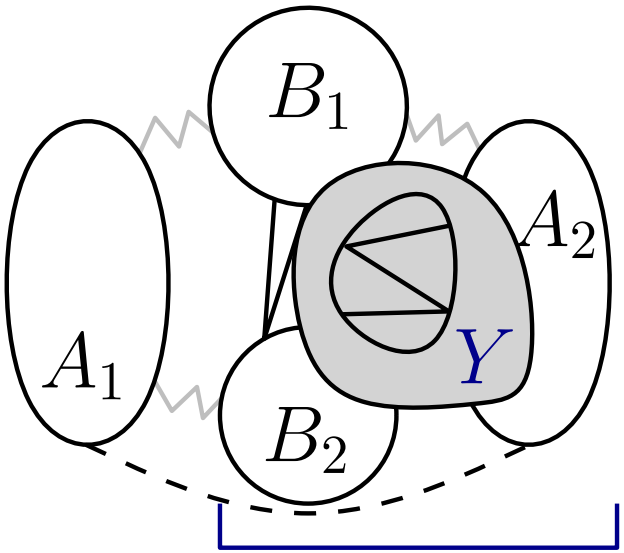


Where is Y?



Y appears in left descendants.

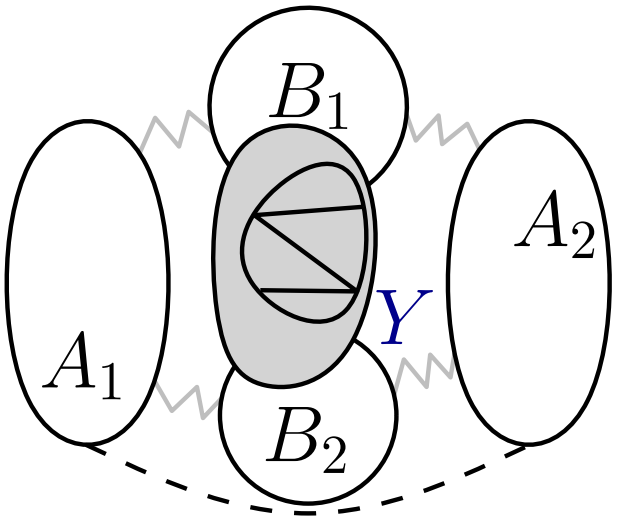
Where is Y?



Y appears in left descendants.

Y appears in right descendants.

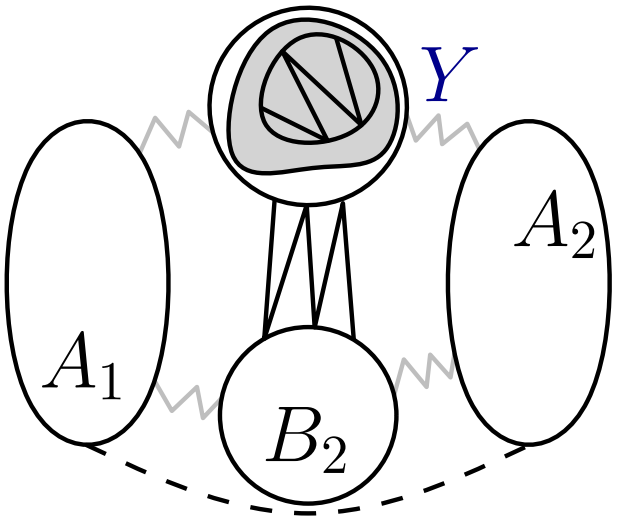
Where is Y?



Y appears in left descendants.

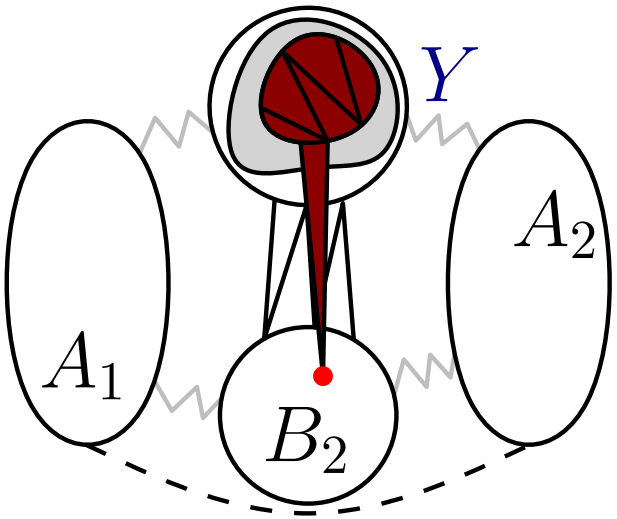
Y appears in right descendants.

Where is Y?



Y appears in left descendants.
Y appears in right descendants.
Y is anticonnected.

Where is Y?



Y appears in left descendants.
Y appears in right descendants.
Y is anticonnected.
Y contains a clique of size $\omega(G)$ and any $v \in B_2$ is complete to it.
$\Rightarrow$ Contradiction!

Outline

Four intermediate steps to reach:

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

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

Four intermediate steps to reach:

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

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!