PARAMETERIZED GRAPH ALGORITHMS USING RESTRICTED MODULAR PARTITIONS

Manuel Lafond,

Weidong Luo

Université de Sherbrooke, Canada

In this talk

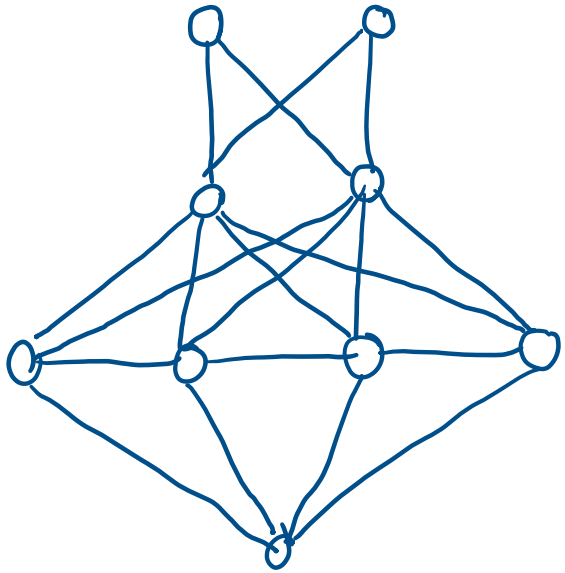
1) Modular-width and parameterized algorithms
2) $\mathcal{G}$-modular cardinality

Paper published in the proceedings of Mathematical Foundations of Computer Science (MFCS) 2023

Let G be a simple undirected graph.

Module of G: subset $X \subseteq V(G)$ of vertices that all have the same neighbors and non-neighbors outside of X.

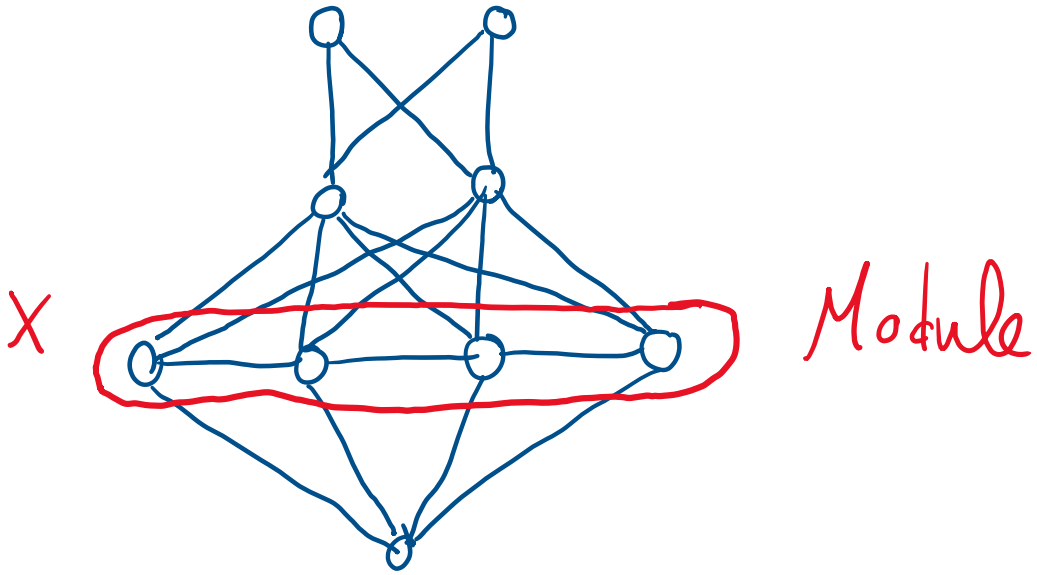
- Between two disjoint modules X, Y, either all edges between X and Y are present, or none.



Let G be a simple undirected graph.

Module of G: subset $X \subseteq V(G)$ of vertices that all have the same neighbors and non-neighbors outside of X.

- Between two disjoint modules X, Y, either all edges between X and Y are present, or none.

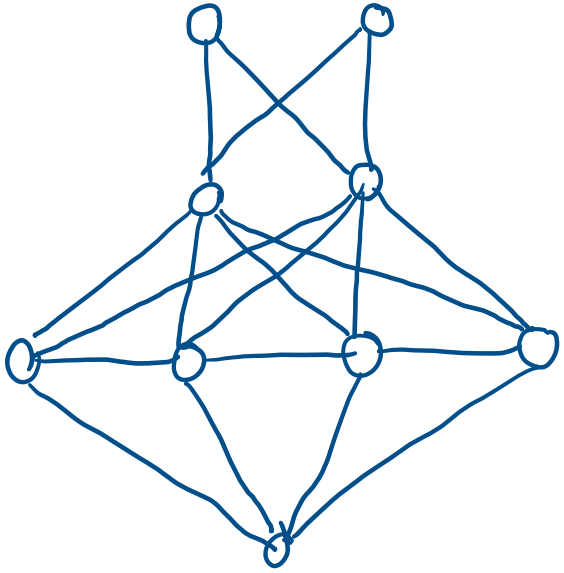


Let G be a simple undirected graph.

Module of G: subset $X \subseteq V(G)$ of vertices that all have the same neighbors and non-neighbors outside of X.

- Between two disjoint modules X, Y, either all edges between X and Y are present, or none.

Modular partition: partition of $V(G)$ into at least two modules, or $V(G)$ if G has a single vertex.

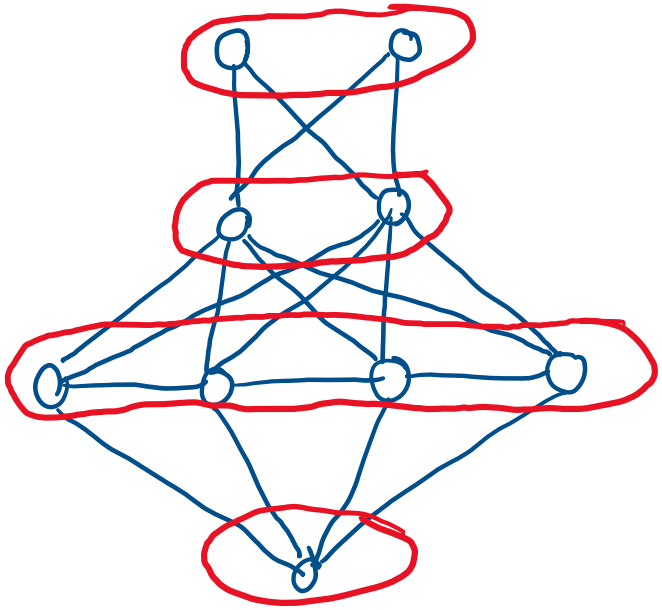


Let G be a simple undirected graph.

Module of G: subset $X \subseteq V(G)$ of vertices that all have the same neighbors and non-neighbors outside of X.

- Between two disjoint modules X, Y, either all edges between X and Y are present, or none.

Modular partition: partition of $V(G)$ into at least two modules, or $V(G)$ if G has a single vertex.

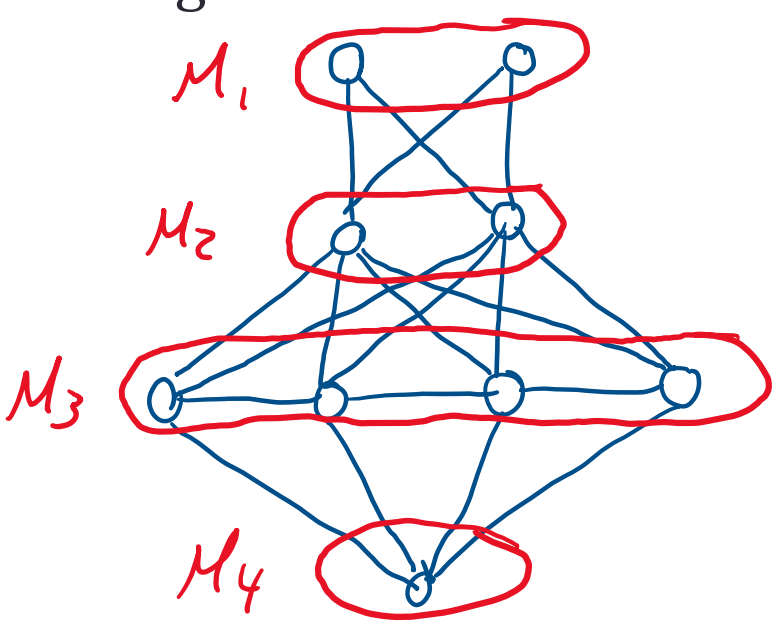


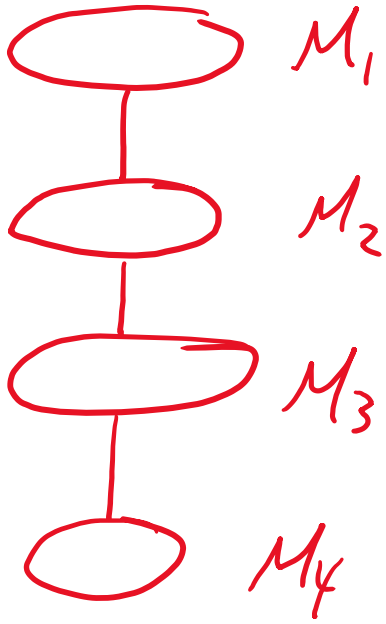
Let G be a simple undirected graph.

Module of G: subset $X \subseteq V(G)$ of vertices that all have the same neighbors and non-neighbors outside of X.

- Between two disjoint modules X, Y, either all edges between X and Y are present, or none.

Modular partition: partition of $V(G)$ into at least two modules, or $V(G)$ if G has a single vertex.





Let G be a simple undirected graph.

Module of G: subset $X \subseteq V(G)$ of vertices that all have the same neighbors and non-neighbors outside of X.

- Between two disjoint modules X, Y, either all edges between X and Y are present, or none.

Modular partition: partition of $V(G)$ into at least two modules, or $V(G)$ if G has a single vertex.

Modular-width: maximum size of the smallest modular partition among all induced subgraphs of G.

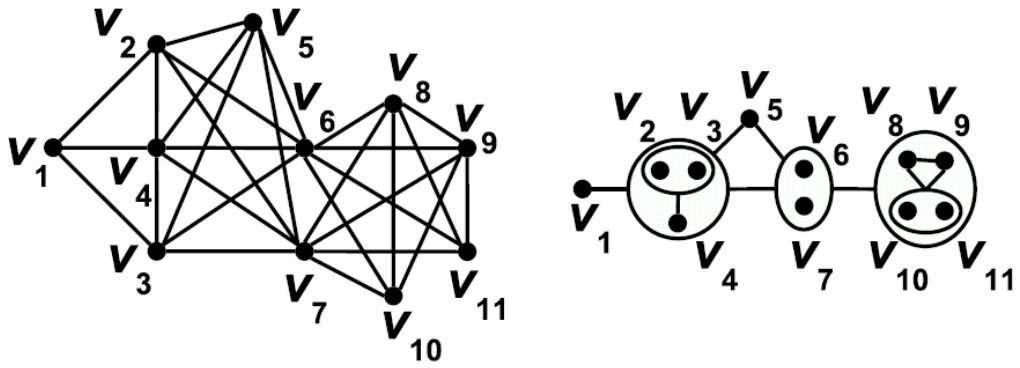
- [Gajarsky, Lampis & Ordyniak, 2013]

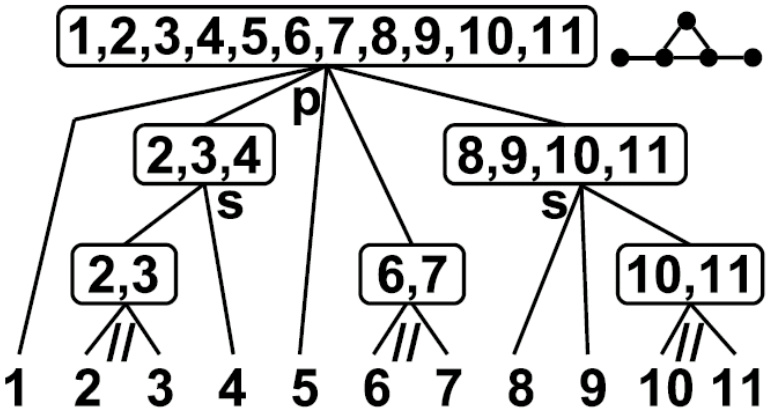
The modular-width of G is denoted $mw(G)$. (or just mw)

Modular-width: $V(G)$ can be partitioned into (at most) $mw(G)$ modules. In turn, each module M_i can be partitioned into (at most) $mw(G)$ modules of $G[M_i]$. And so on...

This creates a modular tree decomposition.

$mw(G) =$ maximum number of children of a node in this tree.



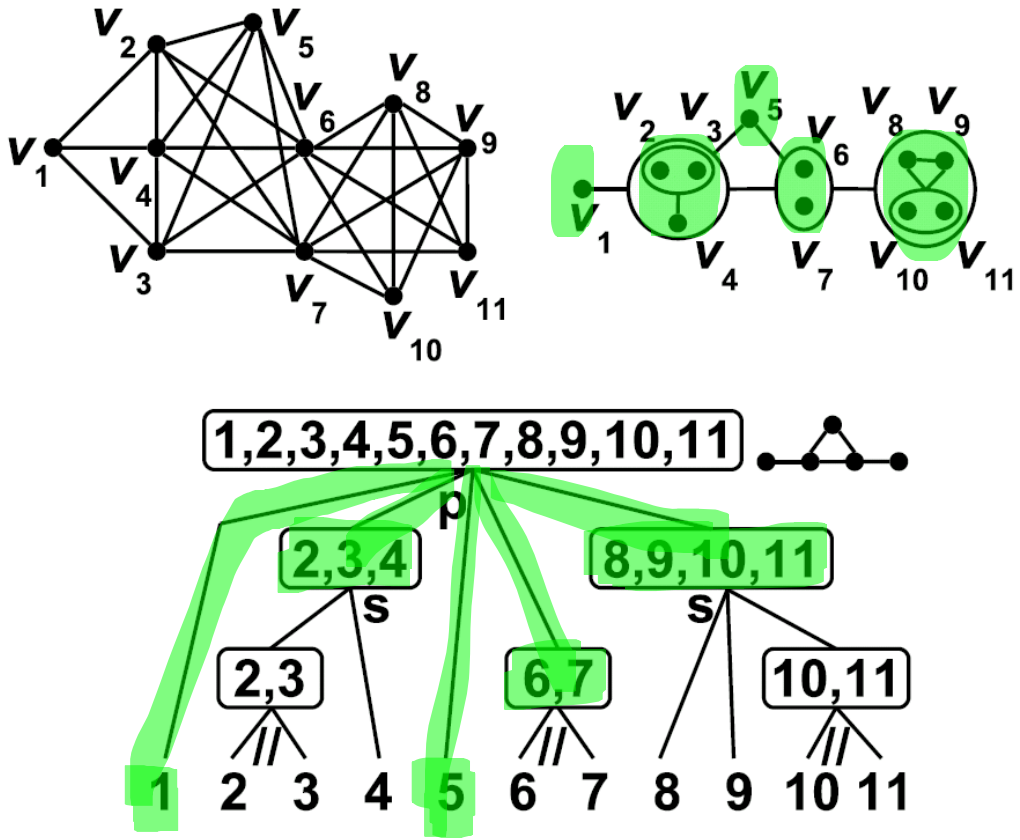


mw = 5

Modular-width: $V(G)$ can be partitioned into (at most) $mw(G)$ modules. In turn, each module M_i can be partitioned into (at most) $mw(G)$ modules of $G[M_i]$. And so on...

This creates a modular tree decomposition.

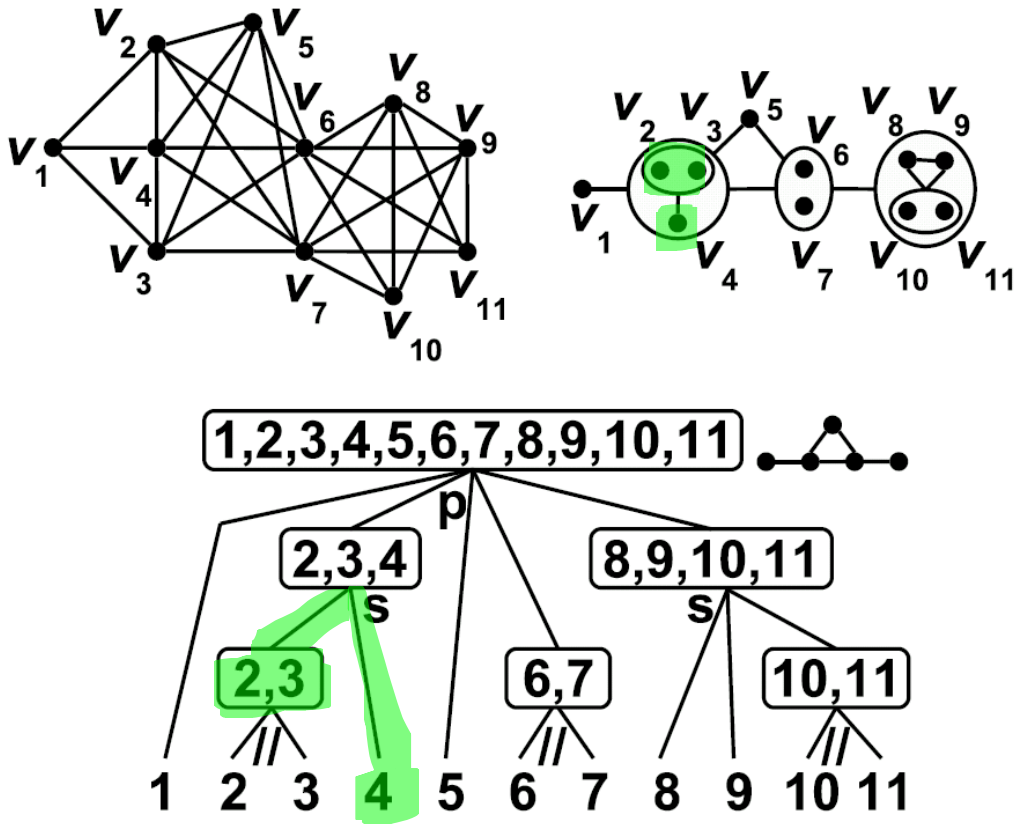
$mw(G) =$ maximum number of children of a node in this tree.



Modular-width: $V(G)$ can be partitioned into (at most) $mw(G)$ modules. In turn, each module M_i can be partitioned into (at most) $mw(G)$ modules of $G[M_i]$. And so on...

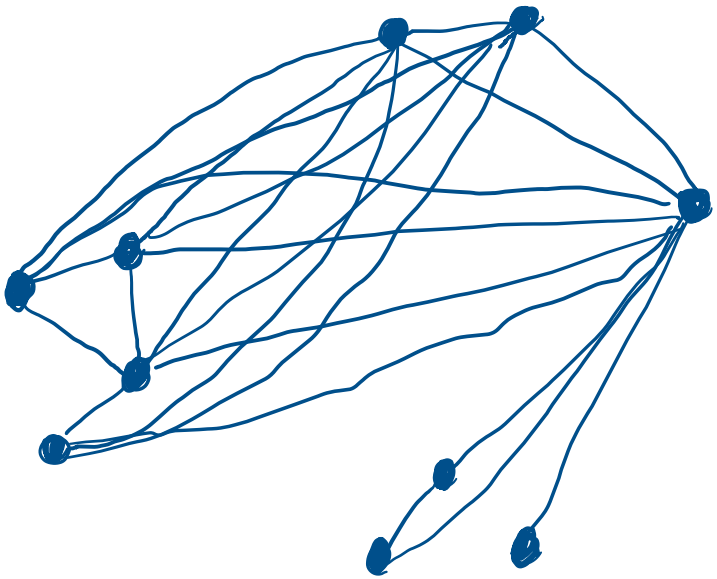
This creates a modular tree decomposition.

$mw(G) =$ maximum number of children of a node in this tree.



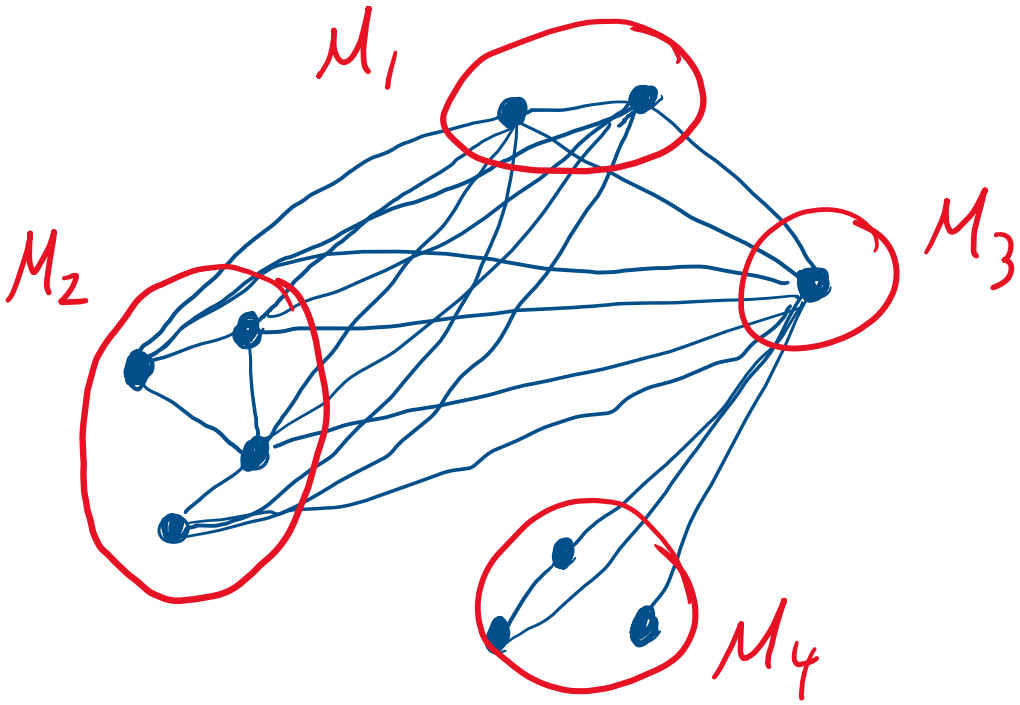
Suppose we want to solve the MAX-CLIQUE problem on G.

1. Find a modular partition P of G with at most $mw(G)$ modules.
2. Recursively find a MAX-CLIQUE C_i in each module $M_i \in P$.
3. Let G_P be the graph in which $V(G) = P$ and M_i, M_j share an edge iff all edges between the two are in G. Each vertex M_i has weight $|Ci|$.
4. Let C' be a clique of maximum weight in G'.
5. Return $\bigcup_{M_i \in C'} C_i$



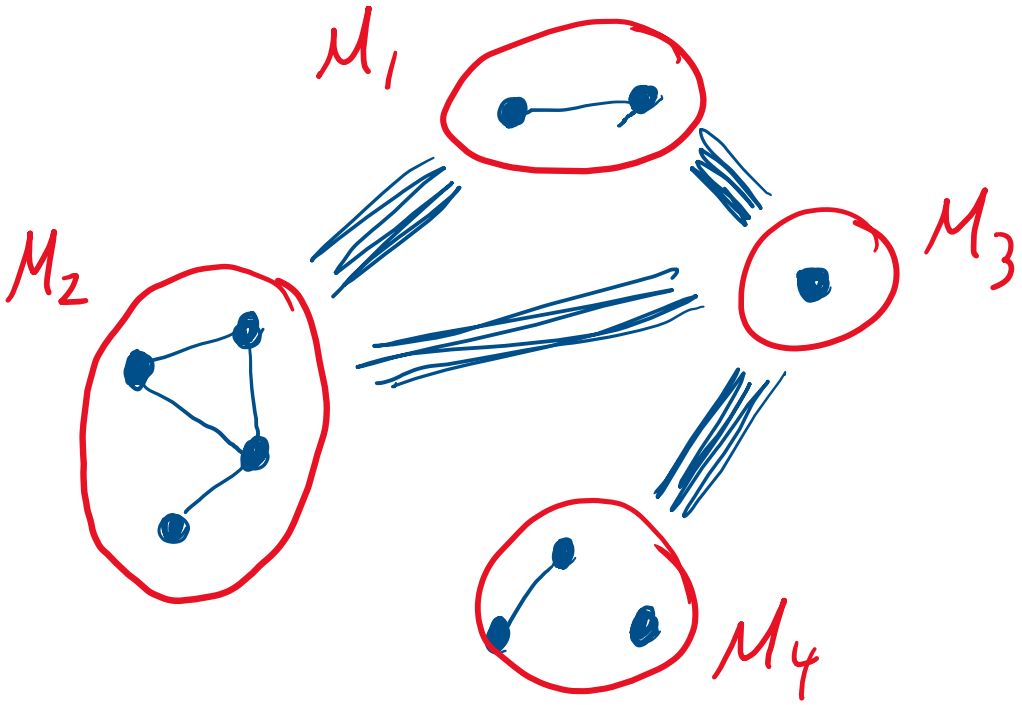
Suppose we want to solve the MAX-CLIQUE problem on G.

1. Find a modular partition P of G with at most $mw(G)$ modules.
2. Recursively find a MAX-CLIQUE C_i in each module $M_i \in P$.
3. Let G_P be the graph in which $V(G) = P$ and M_i, M_j share an edge iff all edges between the two are in G. Each vertex M_i has weight $|Ci|$.
4. Let C' be a clique of maximum weight in G'.
5. Return $\bigcup_{M_i \in C'} C_i$



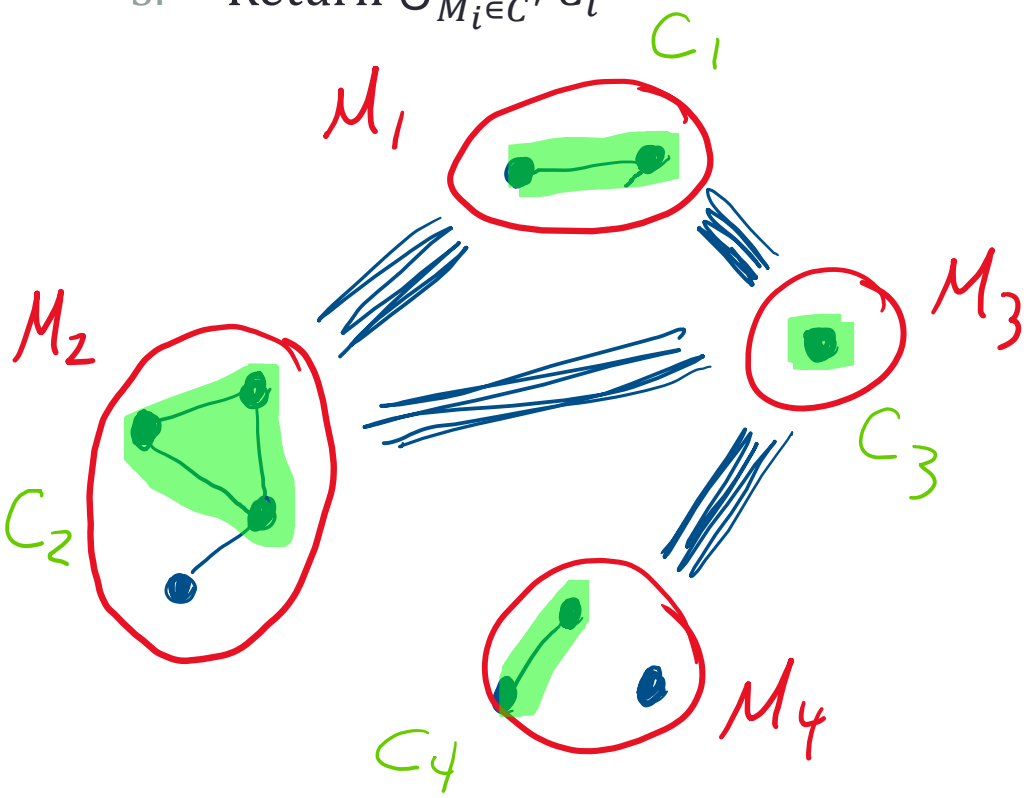
Suppose we want to solve the MAX-CLIQUE problem on G.

1. Find a modular partition P of G with at most $mw(G)$ modules.
2. Recursively find a MAX-CLIQUE C_i in each module $M_i \in P$.
3. Let G_P be the graph in which $V(G) = P$ and M_i, M_j share an edge iff all edges between the two are in G. Each vertex M_i has weight $|Ci|$.
4. Let C' be a clique of maximum weight in G'.
5. Return $\bigcup_{M_i \in C'} C_i$



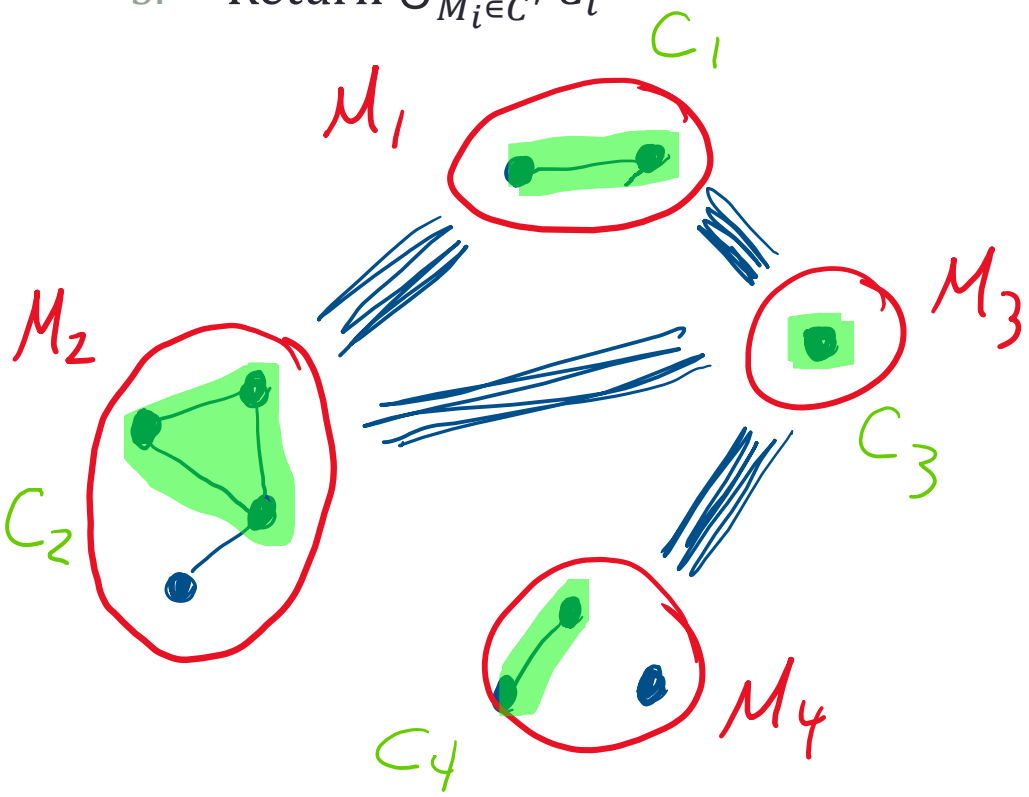
Suppose we want to solve the MAX-CLIQUE problem on G.

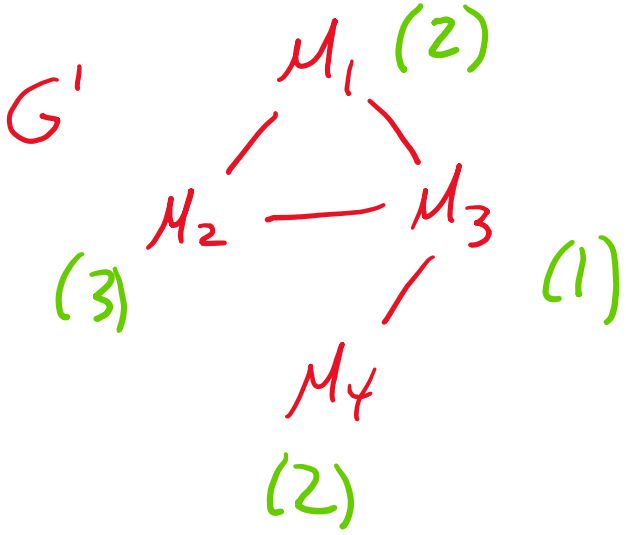
1. Find a modular partition P of G with at most $mw(G)$ modules.
2. Recursively find a MAX-CLIQUE C_i in each module $M_i \in P$.
3. Let G_P be the graph in which $V(G) = P$ and M_i, M_j share an edge iff all edges between the two are in G. Each vertex M_i has weight $|Ci|$.
4. Let C' be a clique of maximum weight in G'.
5. Return $\bigcup_{M_i \in C'} C_i$



Suppose we want to solve the MAX-CLIQUE problem on G.

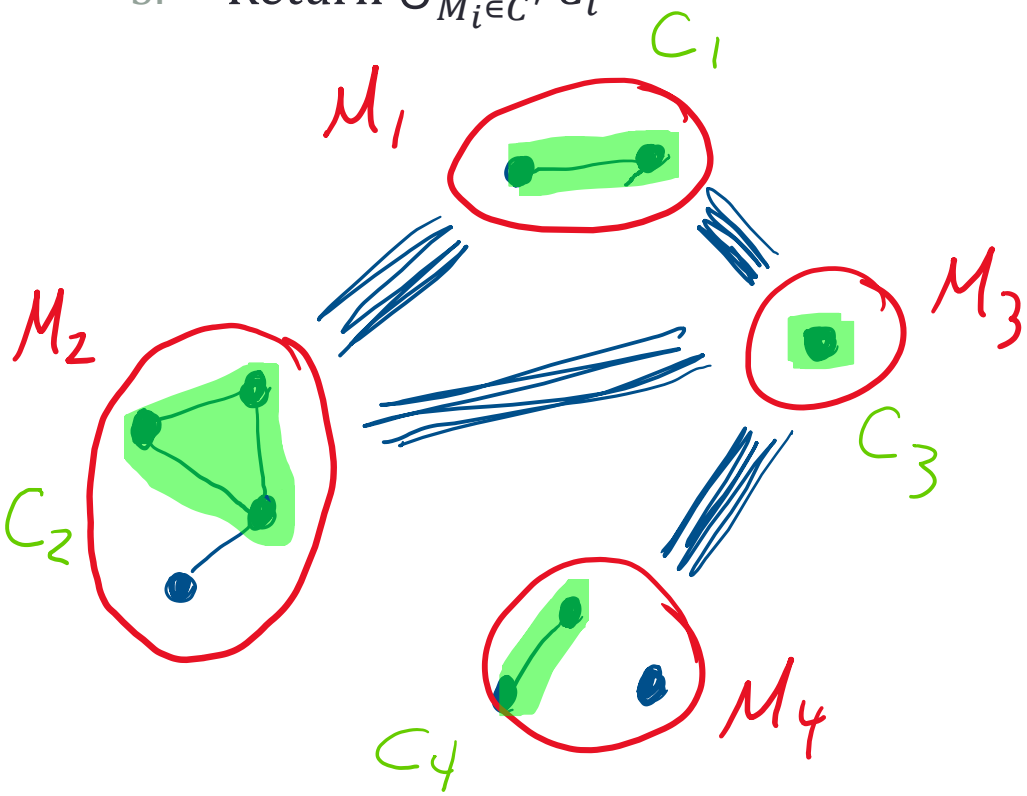
1. Find a modular partition P of G with at most $mw(G)$ modules.
2. Recursively find a MAX-CLIQUE C_i in each module $M_i \in P$.
3. Let G_P be the graph in which $V(G) = P$ and M_i, M_j share an edge iff all edges between the two are in G. Each vertex M_i has weight $|Ci|$.
4. Let C' be a clique of maximum weight in G'.
5. Return $\bigcup_{M_i \in C'} C_i$

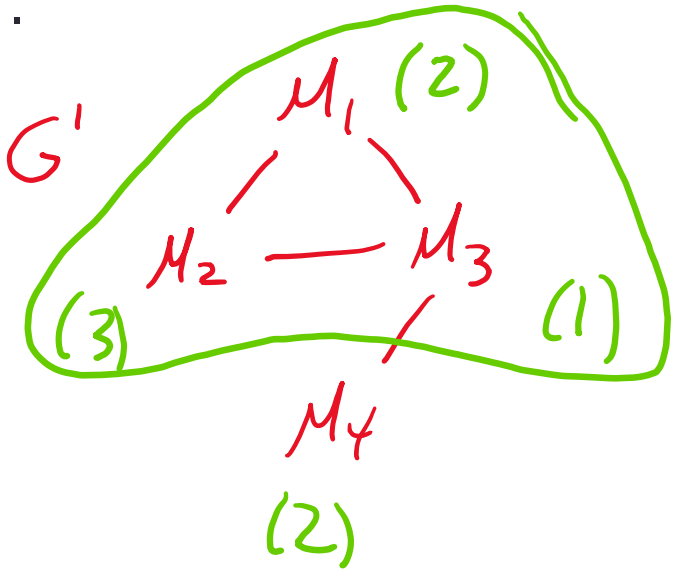




Suppose we want to solve the MAX-CLIQUE problem on G.

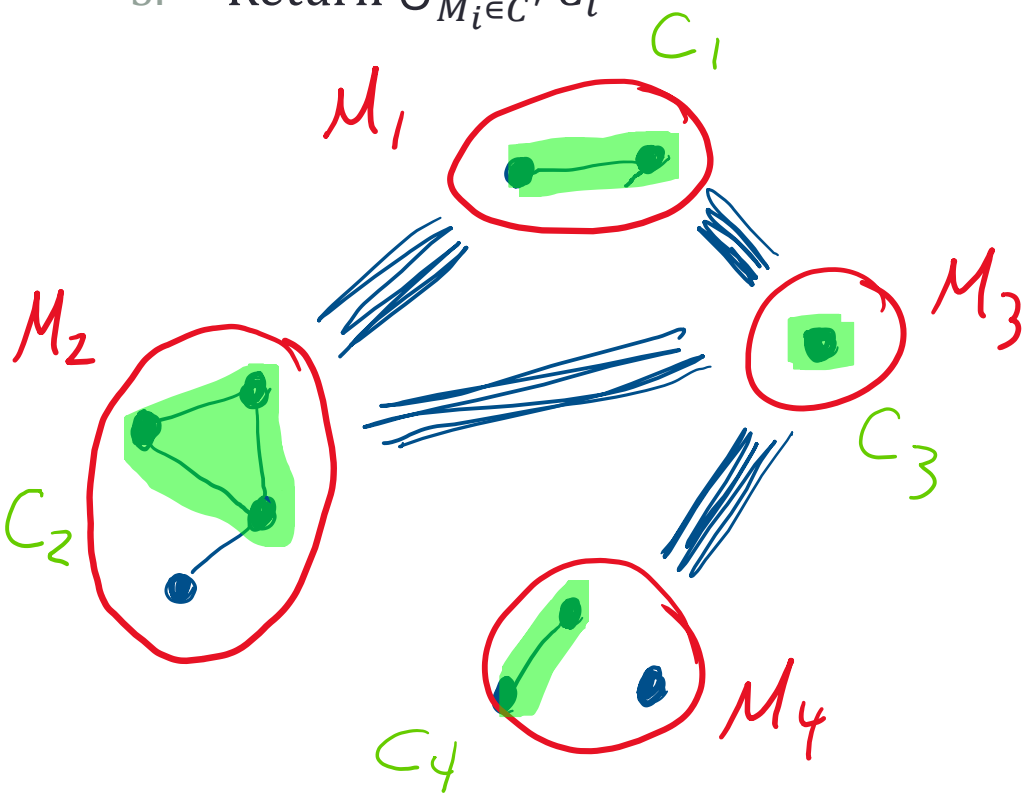
1. Find a modular partition P of G with at most $mw(G)$ modules.
2. Recursively find a MAX-CLIQUE C_i in each module $M_i \in P$.
3. Let G_P be the graph in which $V(G) = P$ and M_i, M_j share an edge iff all edges between the two are in G. Each vertex M_i has weight $|Ci|$.
4. Let C' be a clique of maximum weight in G'.
5. Return $\bigcup_{M_i \in C'} C_i$

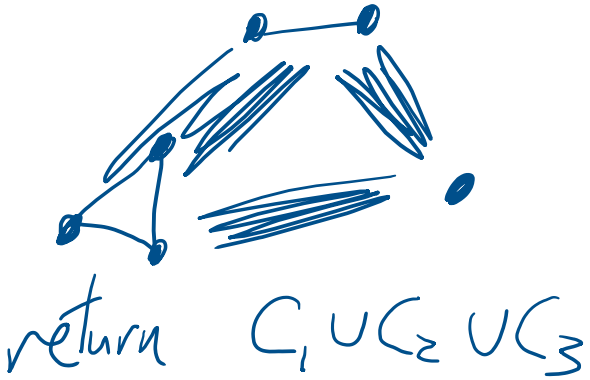




Suppose we want to solve the MAX-CLIQUE problem on G.

1. Find a modular partition P of G with at most $mw(G)$ modules.
2. Recursively find a MAX-CLIQUE C_i in each module $M_i \in P$.
3. Let G_P be the graph in which $V(G) = P$ and M_i, M_j share an edge iff all edges between the two are in G. Each vertex M_i has weight $|Ci|$.
4. Let C' be a clique of maximum weight in G'.
5. Return $\bigcup_{M_i \in C'} C_i$





Suppose we want to solve the MAX-CLIQUE problem on G.

1. Find a modular partition P of G with at most $mw(G)$ modules.
2. Recursively find a MAX-CLIQUE C_i in each module $M_i \in P$.
3. Let G_P be the graph in which $V(G) = P$ and M_i, M_j share an edge iff all edges between the two are in G. Each vertex M_i has weight $|Ci|$.
4. Let C' be a clique of maximum weight in G'.
5. Return $\bigcup_{M_i \in C'} C_i$

Step 1 can be done in time $O(n^2)$

Steps 3-4 can be done in time $O(2^{mw})$

Considering every recursion, total time is $O(2^{mw}n^3)$

Considering every recursion, total time is $O(2^{mw}\ n^3)$

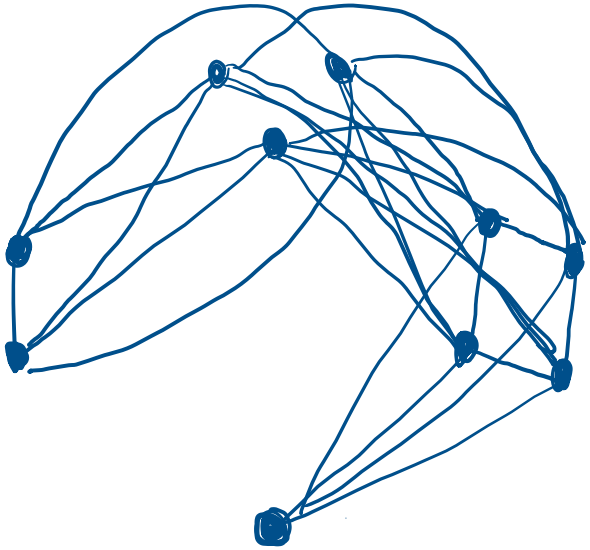
MAX-CLIQUE is fixed-parameter tractable (FPT) in parameter mw

FPT means complexity has the form $f(mw)\ poly(n)$

DO NOT WANT: $O(n^{mw})$

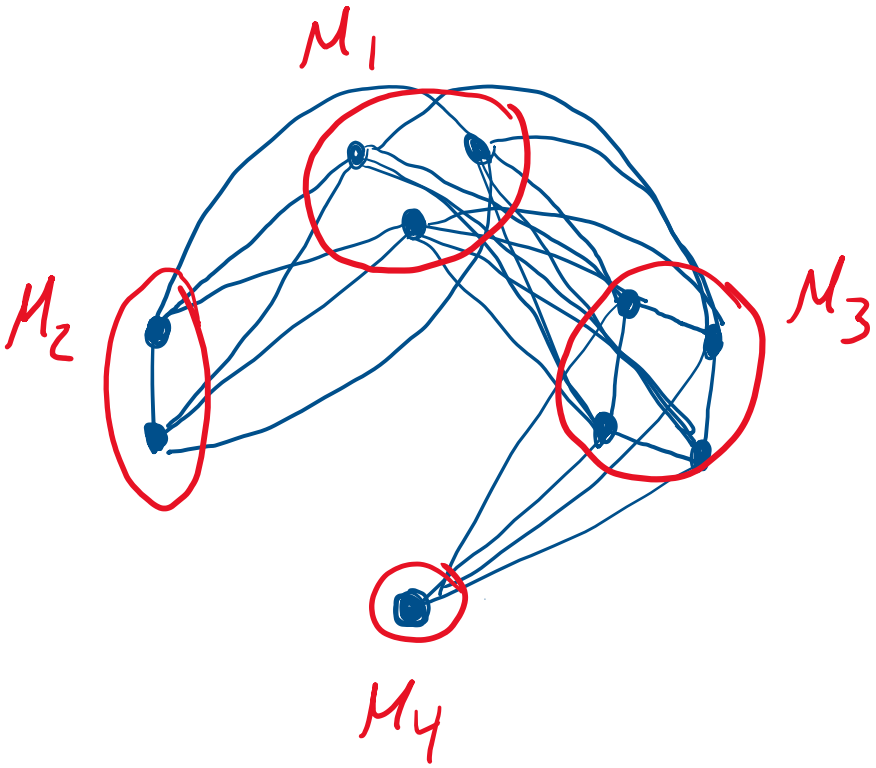
3-COLORING

- We want to color $V(G)$ with $\{1,2,3\}$ such that no edge has two ends of the same color.
- For each module M_i and each subset $S \subseteq \{1,2,3\}$, ask M_i the question "hey, can you color yourself with only colors of S"?



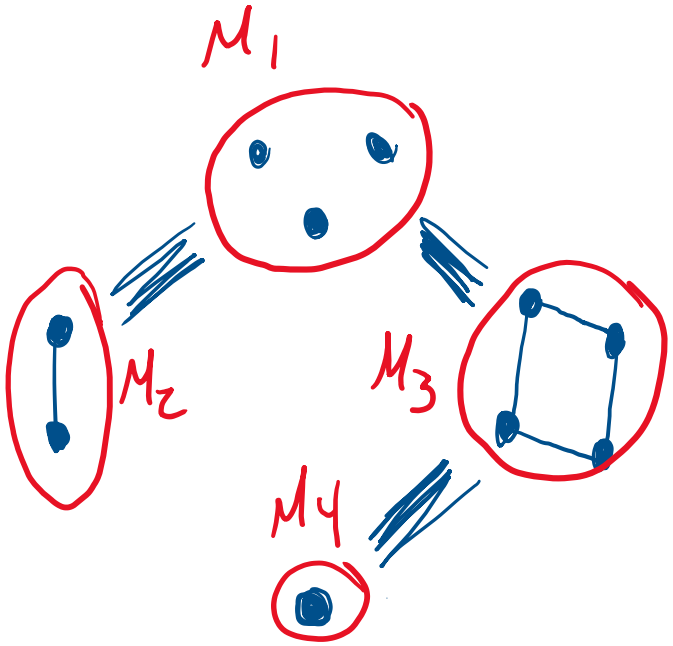
3-COLORING

- We want to color $V(G)$ with $\{1,2,3\}$ such that no edge has two ends of the same color.
- For each module M_i and each subset $S \subseteq \{1,2,3\}$, ask M_i the question "hey, can you color yourself with only colors of S"?



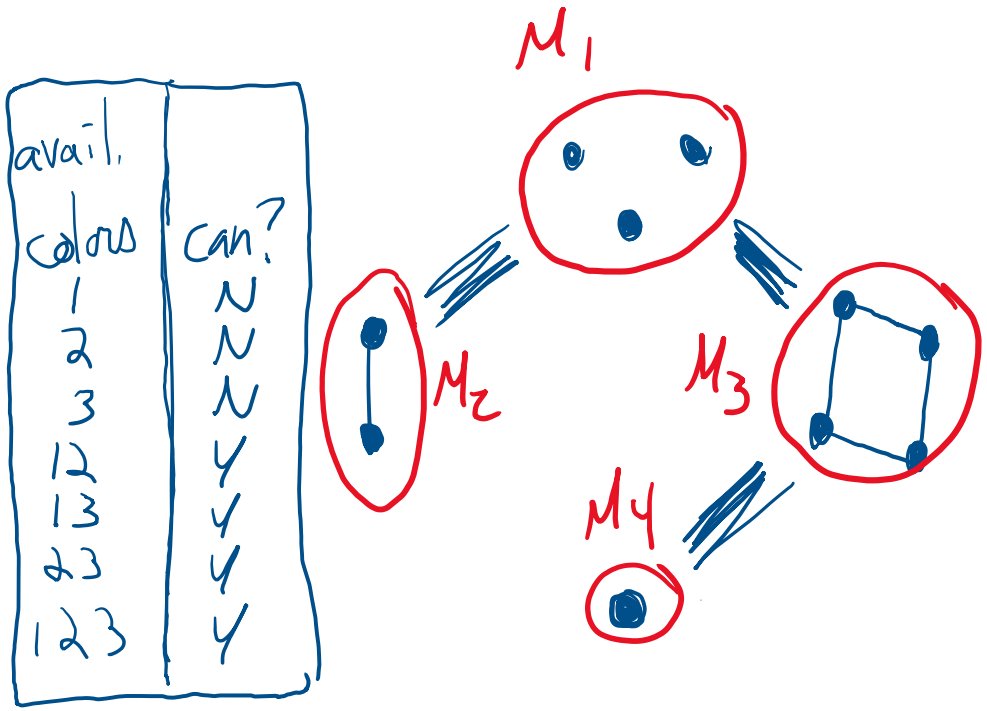
3-COLORING

- We want to color $V(G)$ with $\{1,2,3\}$ such that no edge has two ends of the same color.
- For each module M_i and each subset $S \subseteq \{1,2,3\}$, ask M_i the question "hey, can you color yourself with only colors of S"?



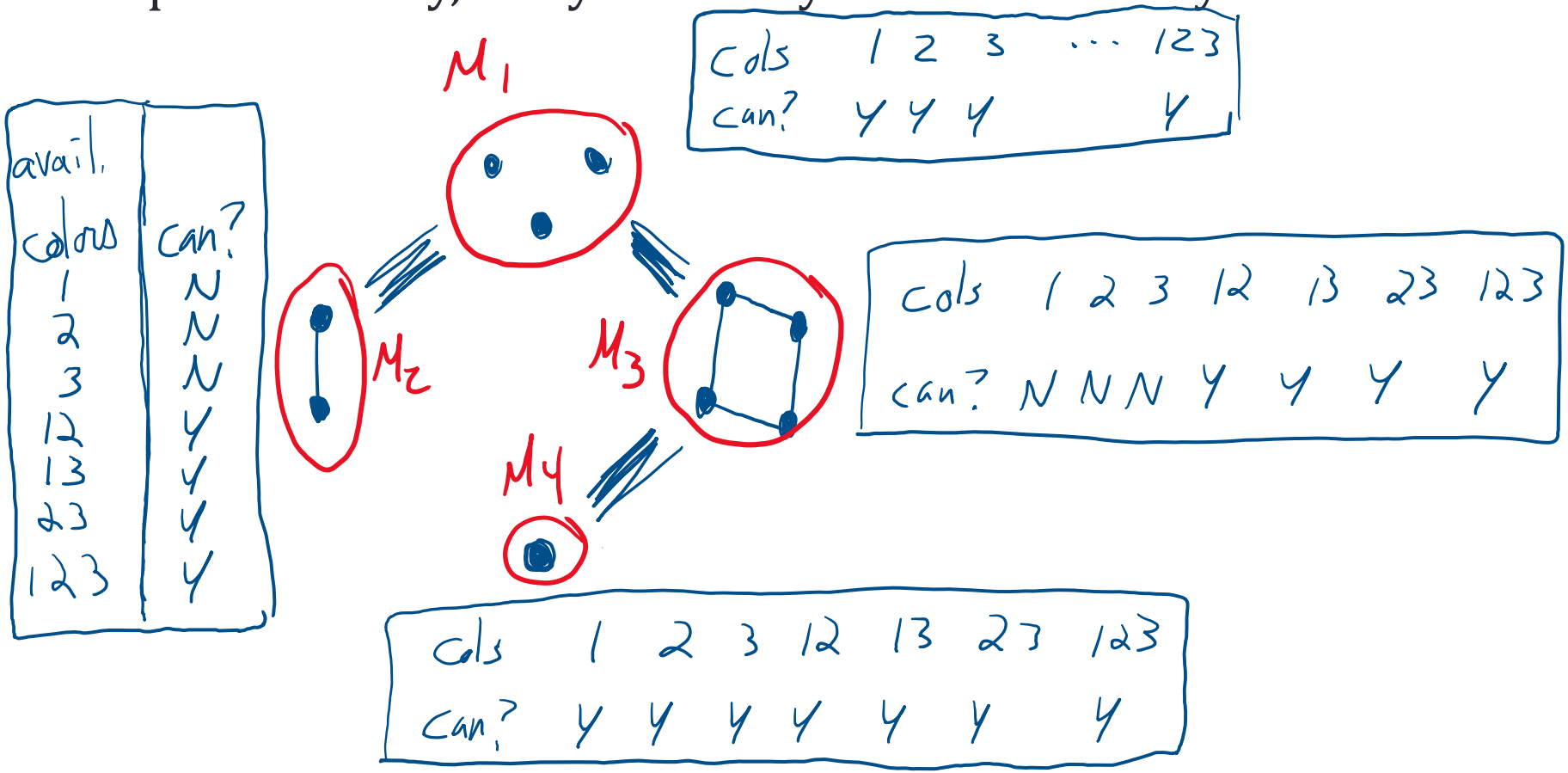
3-COLORING

- We want to color $V(G)$ with $\{1,2,3\}$ such that no edge has two ends of the same color.
- For each module M_i and each subset $S \subseteq \{1,2,3\}$, ask M_i the question "hey, can you color yourself with only colors of S"?



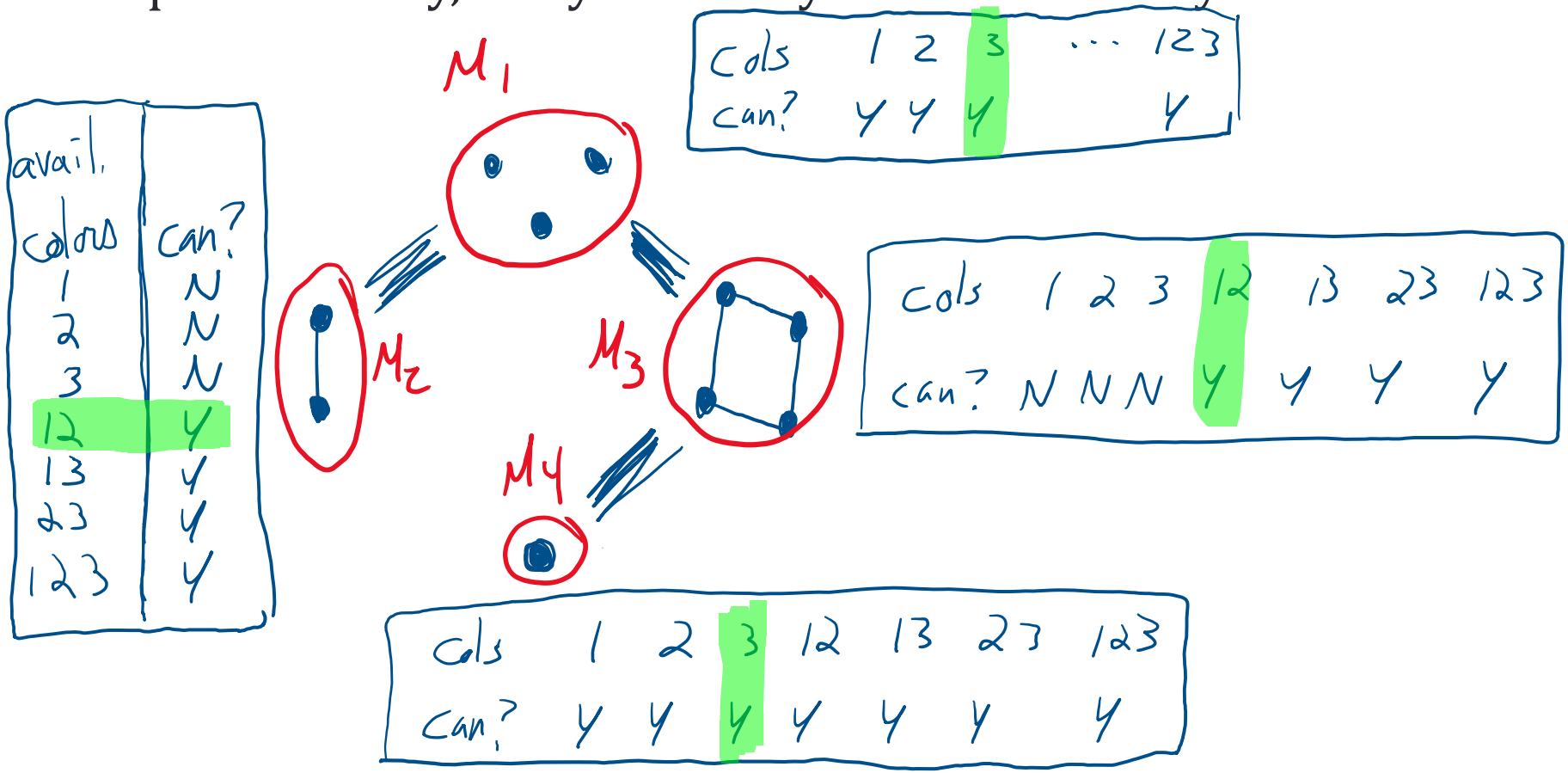
3-COLORING

- We want to color $V(G)$ with $\{1,2,3\}$ such that no edge has two ends of the same color.
- For each module M_i and each subset $S \subseteq \{1,2,3\}$, ask M_i the question "hey, can you color yourself with only colors of S"?



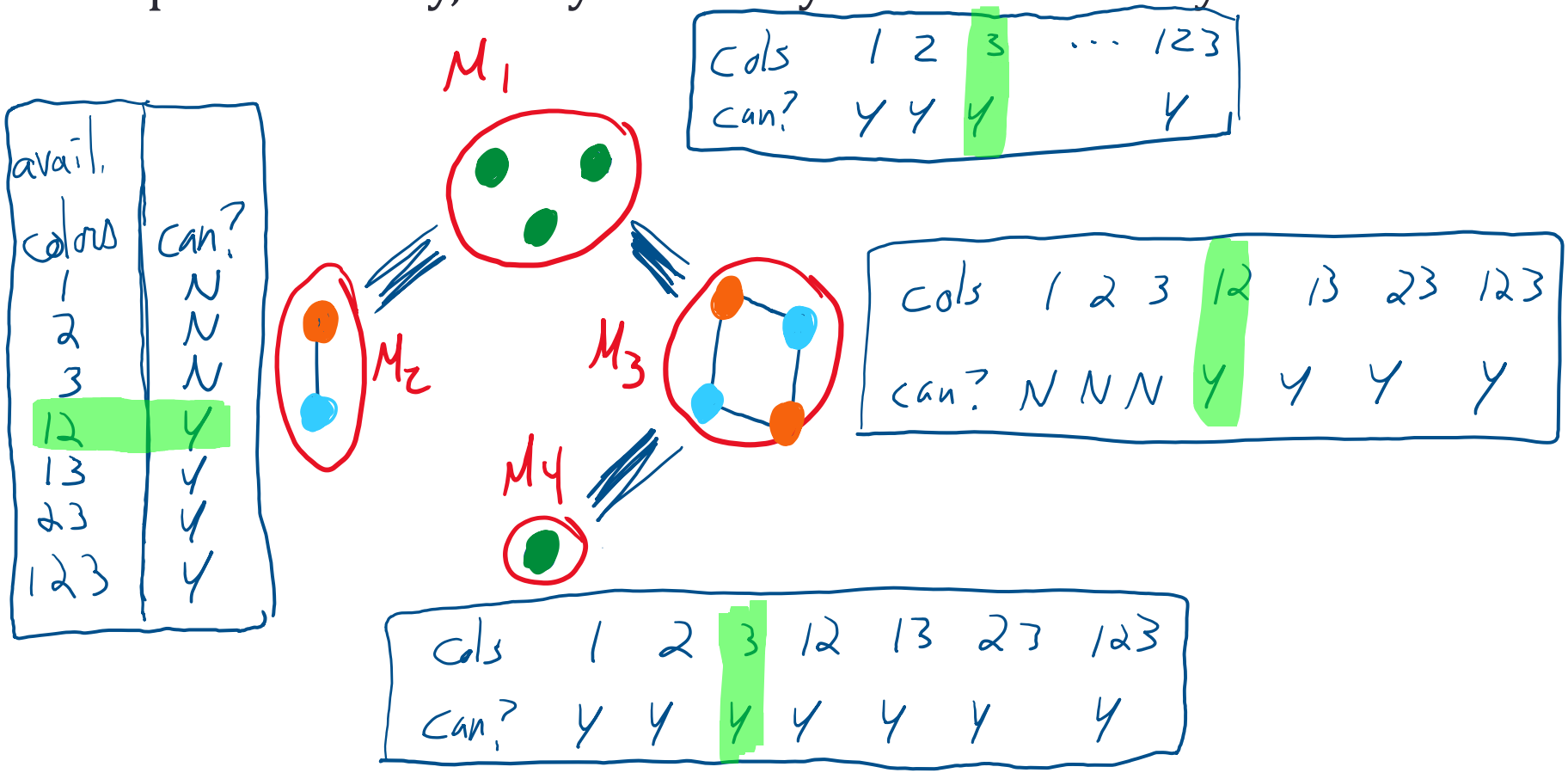
3-COLORING

- We want to color $V(G)$ with {1,2,3} such that no edge has two ends of the same color.
- For each module M_i and each subset $S \subseteq \{1,2,3\}$, ask M_i the question "hey, can you color yourself with only colors of S"?



3-COLORING

- We want to color $V(G)$ with {1,2,3} such that no edge has two ends of the same color.
- For each module M_i and each subset $S \subseteq \{1,2,3\}$, ask M_i the question "hey, can you color yourself with only colors of S"?



3-COLORING

- We want to color $V(G)$ with $\{1,2,3\}$ such that no edge has two ends of the same color.
- For each module M_i and each subset $S \subseteq \{1,2,3\}$, ask M_i the question "hey, can you color yourself with only colors of S"?
- $V(G)$ can be 3-colored iff each M_i can be colored with some color subset S_i such that M_i, M_j sharing all edges implies S_i, S_j are disjoint.

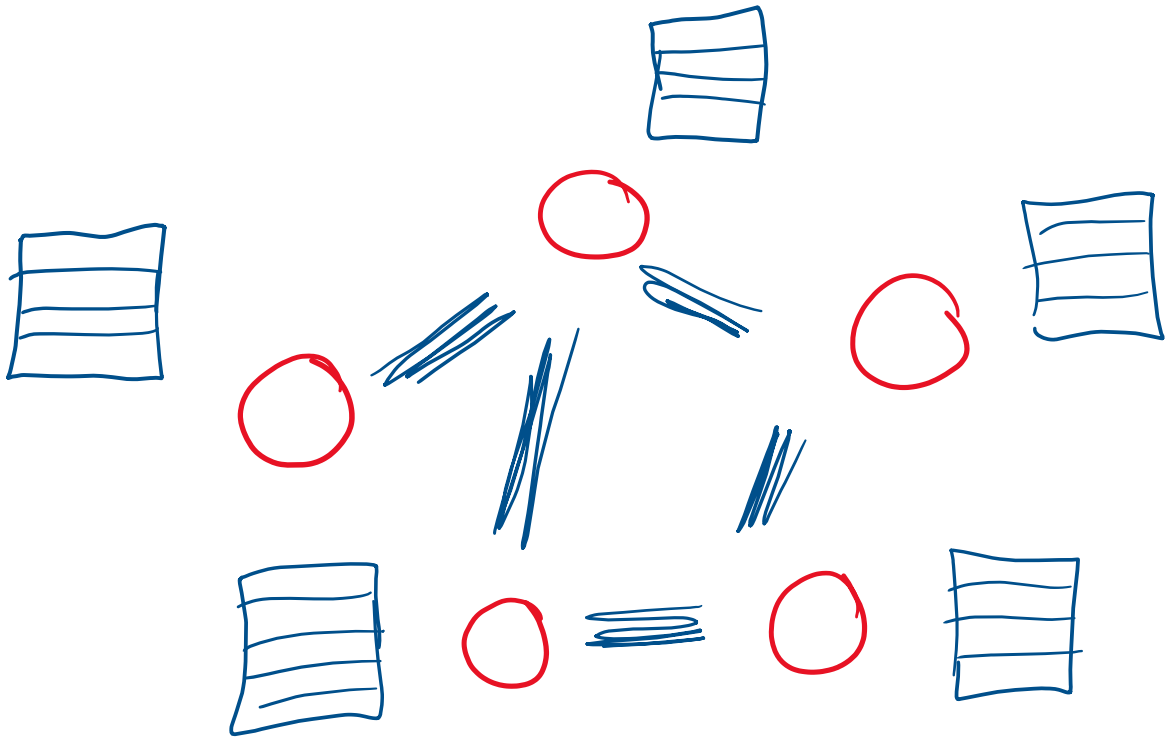
- Complexity: around $O(n\ 7^{mw})$
- 7 entries per module, try each of the 7^{mw} combinations

We call this a "solution table" algorithm.

For each M_i, compute a list of sub-solutions.

Check if the solutions from all the M_i's can be merged.

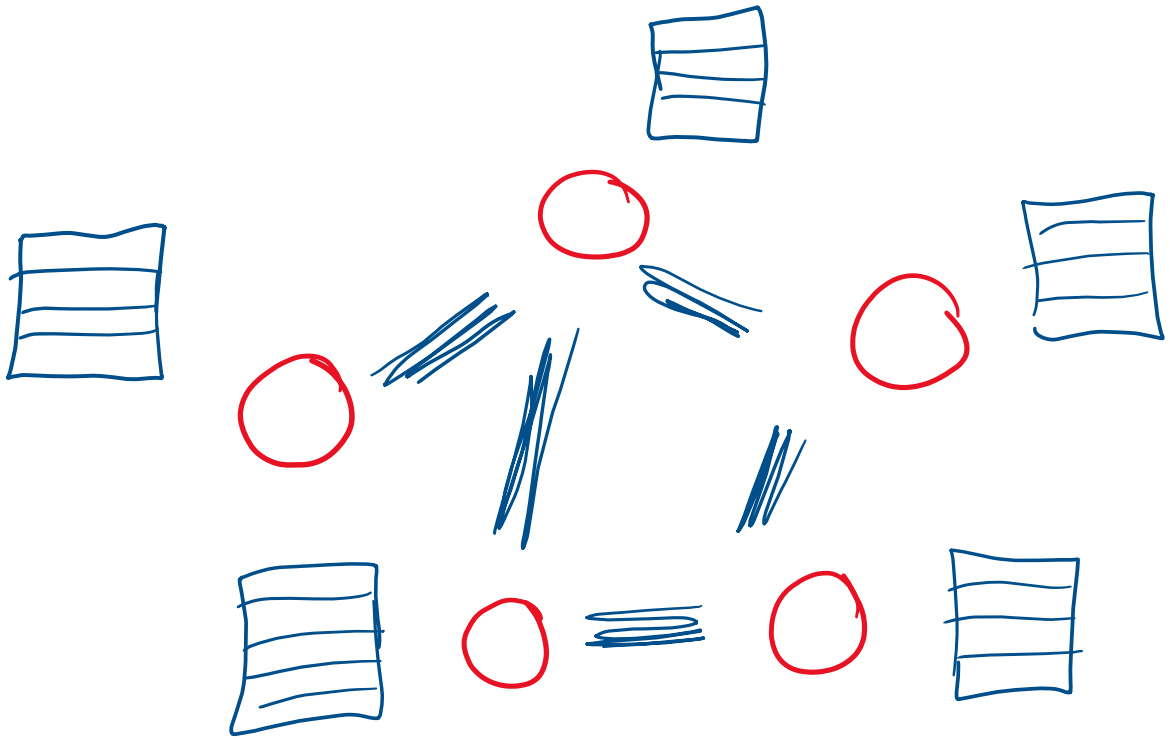
Most FPT algorithms in mw use this idea.



The important property here is

« Each M_i allows computing a list of sub-solutions efficiently, and these sub-solutions can be combined efficiently ».

- Works because each M_i satisfies « bounded modular-width ».
- Could we replace M_i with another graph class?
  - Still works if each M_i is a cograph, a clique, a forest, ...



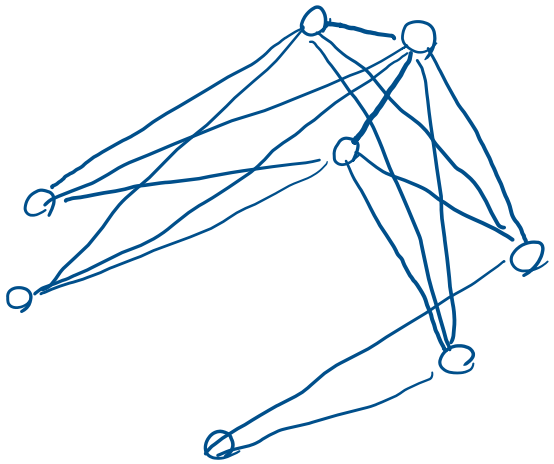
Let $\mathcal{G}$ be a graph class.

The **$\mathcal{G}$-modular cardinality** of a graph G, denoted $\mathcal{G}$-mc, is the minimum size of a modular partition P of $V(G)$ such that, for each module $M \in P$, $G[M]$ is in the class $\mathcal{G}$.

Let $\mathcal{G}$ be a graph class.

The **$\mathcal{G}$-modular cardinality** of a graph G, denoted $\mathcal{G}$-mc, is the minimum size of a modular partition P of $V(G)$ such that, for each module $M \in P$, $G[M]$ is in the class $\mathcal{G}$.

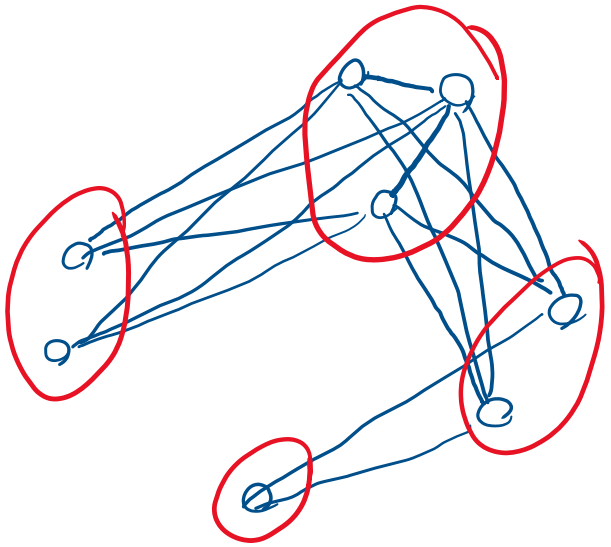
$\mathcal{G}$ = edgeless



Let $\mathcal{G}$ be a graph class.

The **$\mathcal{G}$-modular cardinality** of a graph G, denoted $\mathcal{G}$-mc, is the minimum size of a modular partition P of $V(G)$ such that, for each module $M \in P$, $G[M]$ is in the class $\mathcal{G}$.

$\mathcal{G}$ = edgeless

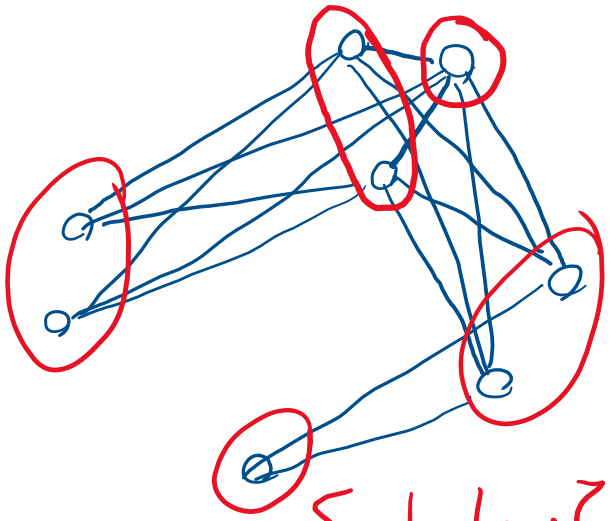


standard modular partition

Let $\mathcal{G}$ be a graph class.

The **$\mathcal{G}$-modular cardinality** of a graph G, denoted $\mathcal{G}$-mc, is the minimum size of a modular partition P of $V(G)$ such that, for each module $M \in P$, $G[M]$ is in the class $\mathcal{G}$.

$\mathcal{G}$ = edgeless



$\mathcal{G}$-modular partition

{edgeless}-mc = 5

Let $\mathcal{G}$ be a graph class.

The **$\mathcal{G}$-modular cardinality** of a graph G, denoted $\mathcal{G}$-mc, is the minimum size of a modular partition P of $V(G)$ such that, for each module $M \in P$, $G[M]$ is in the class $\mathcal{G}$.

e.g. : $\mathcal{G}$ = chordal graphs → partition G into modules, each inducing a chordal graph.

Generalizes previously known parameters:

- neighborhood diversity ⇔ {cliques ∪ edgeless}-mc [Lampis 2012]
- iterated type partition ⇔ {cographs}-mc [Cordaso & al 2020]
- {cliques}-mc used for clustering algorithms in [Fomin & al 2021]
- {bounded rank-width}-mc studied in [Kwon & al 2022]

Question 1: for which graph class $\mathcal{G}$ can $\mathcal{G}$-mc be computed in polynomial time?

Question 2: for an algorithmic problem X, which graph class $\mathcal{G}$ allows an FPT algorithm in parameter $\mathcal{G}$ –mc?

FPT: complexity $f(\mathcal{G}\text{-}mc)\ poly(n)$

e.g. $O(2^{\mathcal{G}\text{-mc}}\ n^2)$

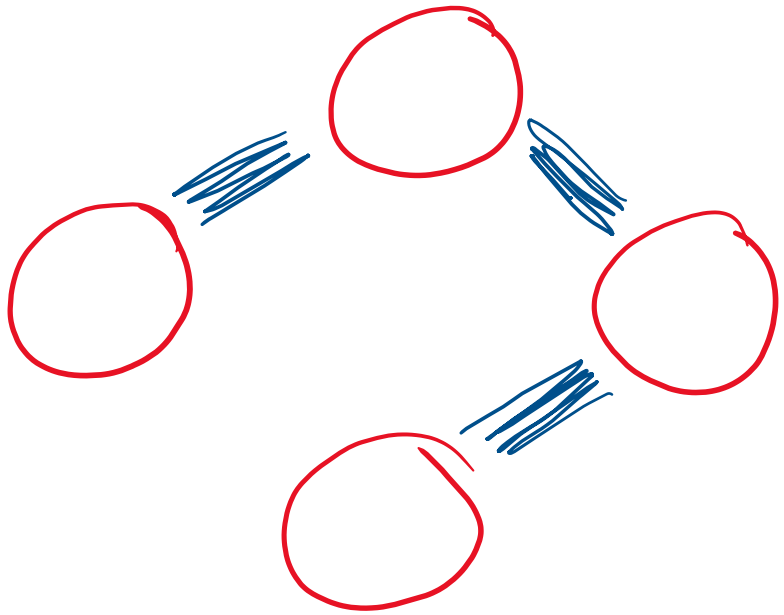
Question 1: for which graph class $\mathcal{G}$ can $\mathcal{G}$-mc be computed in polynomial time?

For a graph class $\mathcal{G}$ and a given a graph G:

Easy case: G and its complement are connected.

1. Find a minimum-size modular partition P of $V(G)$.
2. Find the $\mathcal{G}$-modules in each $G[M]$ recursively, $M \in P$.
3. Return the union of all the $\mathcal{G}$-modules found.

Intuition: this works because we cannot do better than P since it partitions minimally without $\mathcal{G}$ membership restriction. The best we can do is partition each $G[M]$ into $\mathcal{G}$-modules.

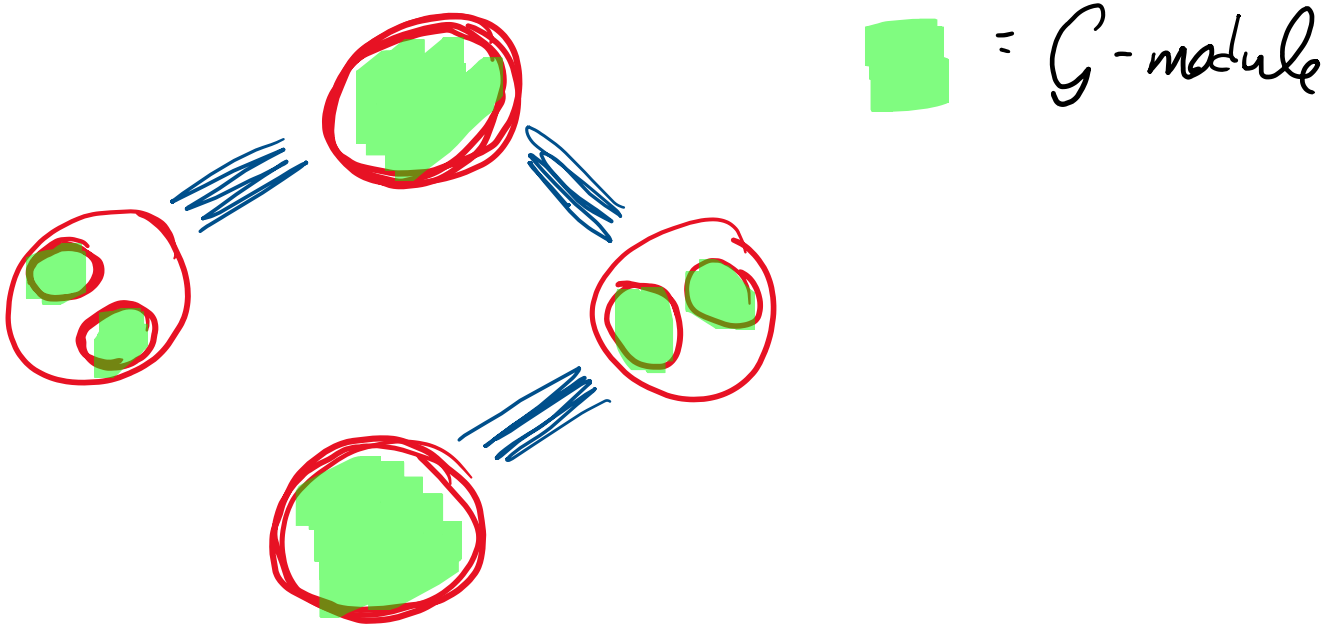


For a graph class $\mathcal{G}$ and a given a graph G:

Easy case: G and its complement are connected.

1. Find a minimum-size modular partition P of $V(G)$.
2. Find the $\mathcal{G}$-modules in each $G[M]$ recursively, $M \in P$.
3. Return the union of all the $\mathcal{G}$-modules found.

Intuition: this works because we cannot do better than P since it partitions minimally without $\mathcal{G}$ membership restriction. The best we can do is partition each $G[M]$ into $\mathcal{G}$-modules.

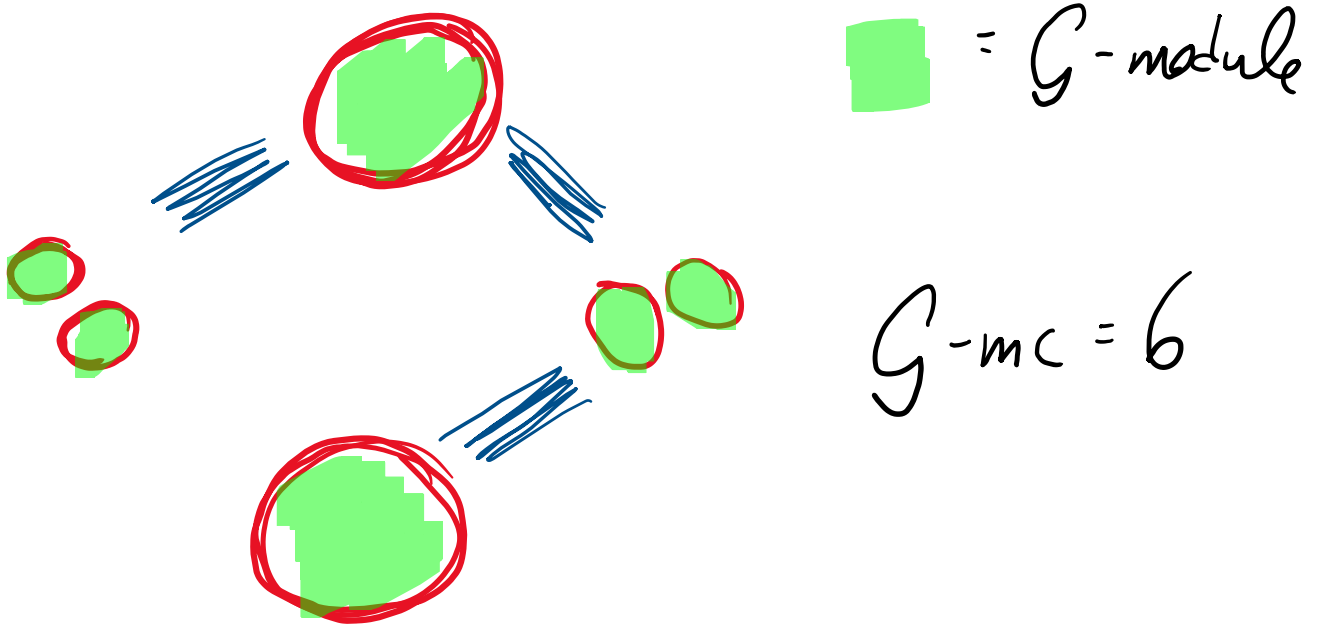


For a graph class $\mathcal{G}$ and a given a graph G:

Easy case: G and its complement are connected.

1. Find a minimum-size modular partition P of $V(G)$.
2. Find the $\mathcal{G}$-modules in each $G[M]$ recursively, $M \in P$.
3. Return the union of all the $\mathcal{G}$-modules found.

Intuition: this works because we cannot do better than P since it partitions minimally without $\mathcal{G}$ membership restriction. The best we can do is partition each $G[M]$ into $\mathcal{G}$-modules.



Tough case: G is disconnected (or its complement is disconnected).

1. Assume that each connected component is in $\boldsymbol{\mathcal{G}}$.
2. Then merge the connected components optimally while staying in the graph class $\boldsymbol{\mathcal{G}}$.

Tough case: G is disconnected (or its complement is disconnected).

1. Assume that each connected component is in $\mathcal{G}$.
2. Then merge the connected components optimally while staying in the graph class $\mathcal{G}$.

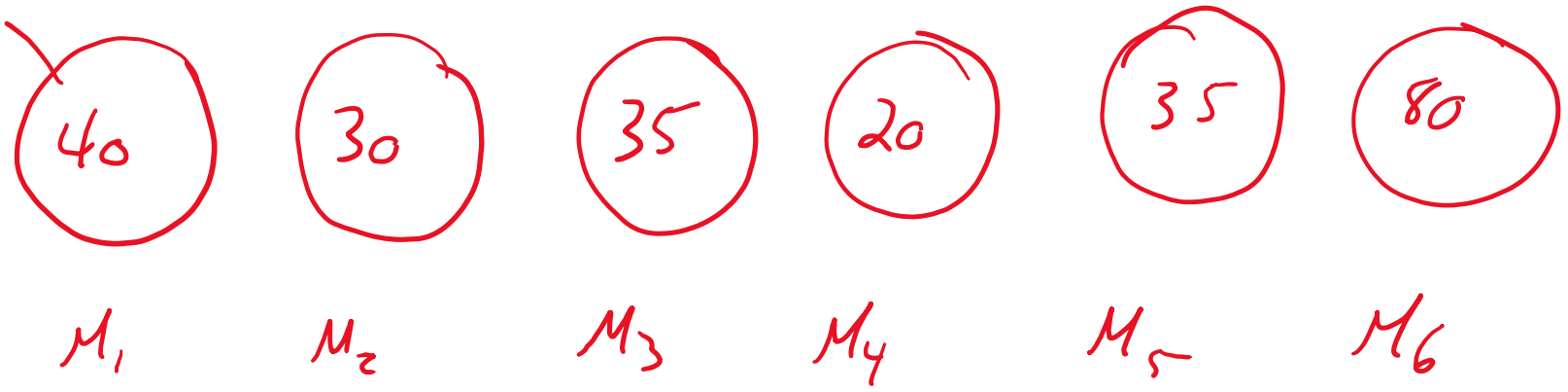
$\mathcal{G}$ = "graphs with ≤ 100 vertices".

Tough case: G is disconnected (or its complement is disconnected).

1. Assume that each connected component is in $\mathcal{G}$.
2. Then merge the connected components optimally while staying in the graph class $\mathcal{G}$.

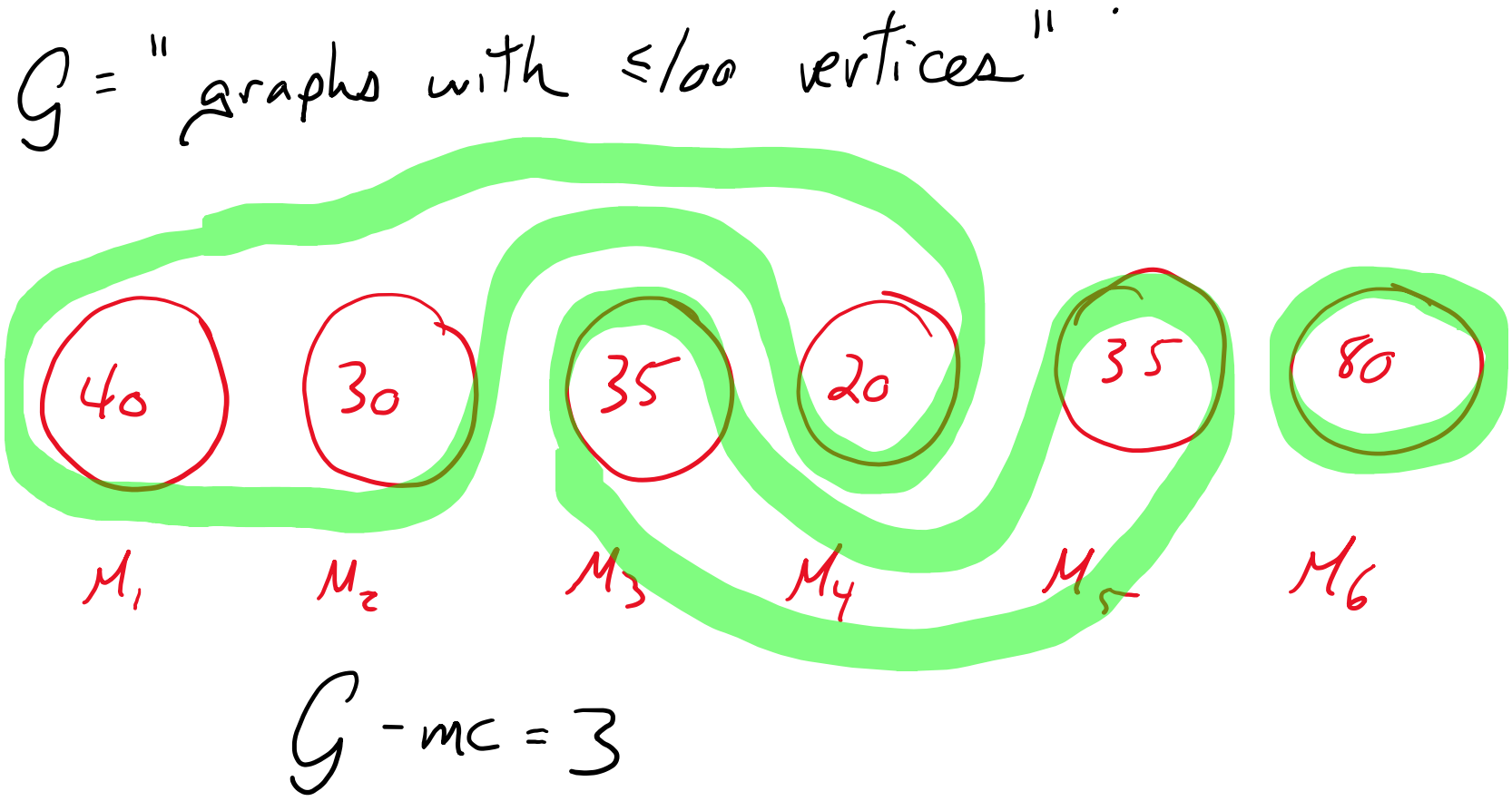
$\mathcal{G}$ = "graphs with ≤ 100 vertices"

#vertices in module



Tough case: G is disconnected (or its complement is disconnected).

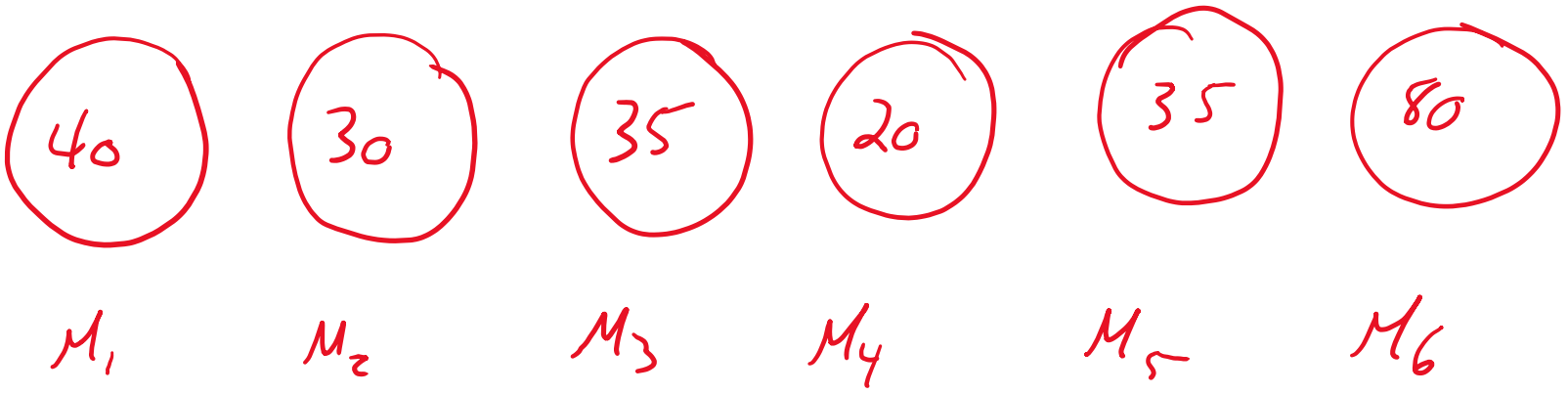
1. Assume that each connected component is in $\mathcal{G}$.
2. Then merge the connected components optimally while staying in the graph class $\mathcal{G}$.



Tough case: G is disconnected (or its complement is disconnected).

1. Assume that each connected component is in $\mathcal{G}$.
2. Then merge the connected components optimally while staying in the graph class $\mathcal{G}$.

G = "graphs with ≤ 100 vertices"



G - mc = null

Theorem: if $\mathcal{G}$ is a *hereditary* graph class such that

- membership in $\mathcal{G}$ can be done in polynomial time; and
- the optimal merging of connected components can be performed in polynomial time,

then $\mathcal{G}$-mc can be computed in polynomial time.

Theorem: if $\mathcal{G}$ is a *hereditary* graph class such that

- membership in $\mathcal{G}$ can be done in polynomial time; and
- the optimal merging of connected components can be performed in polynomial time,

then $\mathcal{G}$-mc can be computed in polynomial time.

Corollary: if $\mathcal{G}$ is one of {cographs},{cliques},{edgeless},{stars},{bounded-mw},... then $\mathcal{G}$-mc can be computed in polynomial time.

Theorem: if $\mathcal{G}$ is a *hereditary* graph class such that

- membership in $\mathcal{G}$ can be done in polynomial time; and
- the optimal merging of connected components can be performed in polynomial time,

then $\mathcal{G}$-mc can be computed in polynomial time.

Corollary: if $\mathcal{G}$ is one of {cographs},{cliques},{edgeless},{stars},{bounded-mw}, then $\mathcal{G}$-mc can be computed in polynomial time.

Question: characterize polytime graph classes (dichotomy)?

Question: is there a polytime recognizable graph class for which $\mathcal{G}$-mc is NP-hard to compute?

Question 2: for an algorithmic problem X, which graph class $\boldsymbol{\mathcal{G}}$ allows an FPT algorithm in parameter $\boldsymbol{\mathcal{G}}$–mc?

FPT: complexity $f(\boldsymbol{\mathcal{G}}\text{-}mc)\ poly(n)$

e.g. $O(2^{\boldsymbol{\mathcal{G}}\text{-mc}}\ n^2)$

Question 2: for an algorithmic problem X, which graph class $\mathcal{G}$ allows an FPT algorithm in parameter $\mathcal{G}$–mc?

FPT: complexity $f(\mathcal{G}\text{-}mc)\ poly(n)$

e.g. $O(2^{\mathcal{G}\text{-mc}}\ n^2)$

If MAX-CLIQUE is polytime (or FPT) on $\mathcal{G}$, solution table idea still works.

- MAX-CLIQUE is FPT in $\mathcal{G}$-mc when $\mathcal{G}$ is any combination of {cliques},{cographs},{edgeless},{forests},{bounded-degree},{bounded treewidth}, …

Similar techniques also work with 3-COLORING, VERTEX-COVER, LONGEST-PATH, and many others.

Question 2: for an algorithmic problem X, which graph class $\mathcal{G}$ allows an FPT algorithm in parameter $\mathcal{G}$–mc?

FPT: complexity $f(\mathcal{G}\text{-}mc)\ poly(n)$

e.g. $O(2^{\mathcal{G}\text{-mc}}\ n^2)$

When does it not work?

Solution tables must be small so that we can brute force over them.

If all useful solution tables are **large**, no efficient algorithm.

Bounded-Degree-Deletion (BDD) problem

Given: a graph G and an integer q

Task: delete a minimum number of vertices so that the resulting graph has maximum degree q (or less).

Example: when $q = 0$, this is vertex-cover.

Delete vertices in the cover → results in graph of max degree 0.

For general q, the "only" solution table idea that seems to work:

- Ask each $\boldsymbol{\mathcal{G}}$-module M_i,

  "hey, if I let you delete k vertices, what's the smallest max degree you can achieve?"

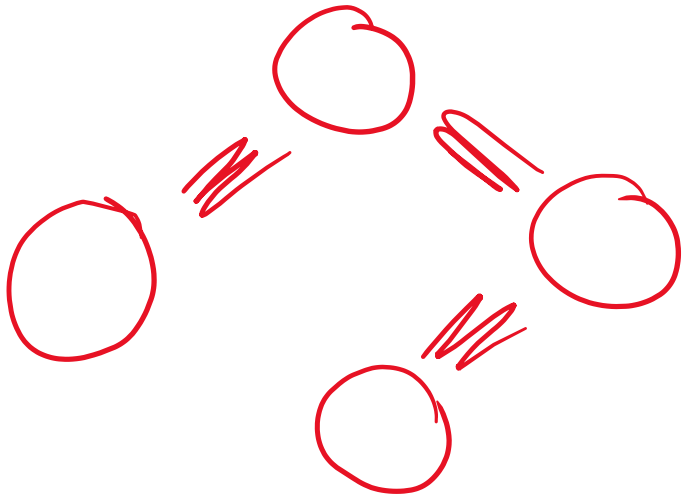
- $M_i[0]$ = smallest possible max degree if we delete 0 vertices
- $M_i[1]$ = smallest possible max degree if we delete 1 vertex
- **...**
- $M_i[|M_i|]$ = smallest possible max degree if we delete all vertices

For general q, the "only" solution table idea that seems to work:

- Ask each $\boldsymbol{\mathcal{G}}$-module M_i,

  "hey, if I let you delete k vertices, what's the smallest max degree you can achieve?"

- $M_i[0]$ = smallest possible max degree if we delete 0 vertices
- $M_i[1]$ = smallest possible max degree if we delete 1 vertex
- ...
- $M_i[|M_i|]$ = smallest possible max degree if we delete all vertices

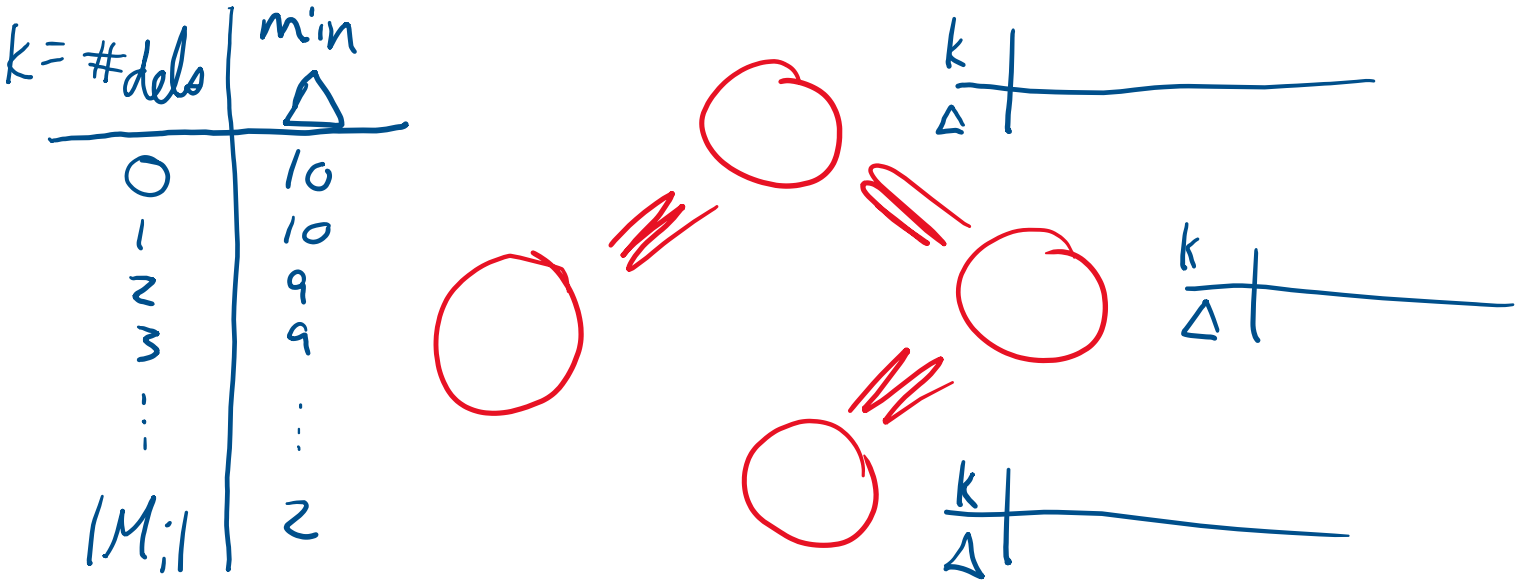


For general q, the "only" solution table idea that seems to work:

- Ask each $\mathcal{G}$-module M_i,

  "hey, if I let you delete k vertices, what's the smallest max degree you can achieve?"

- $M_i[0]$ = smallest possible max degree if we delete 0 vertices
- $M_i[1]$ = smallest possible max degree if we delete 1 vertex
- ...
- $M_i[|M_i|]$ = smallest possible max degree if we delete all vertices

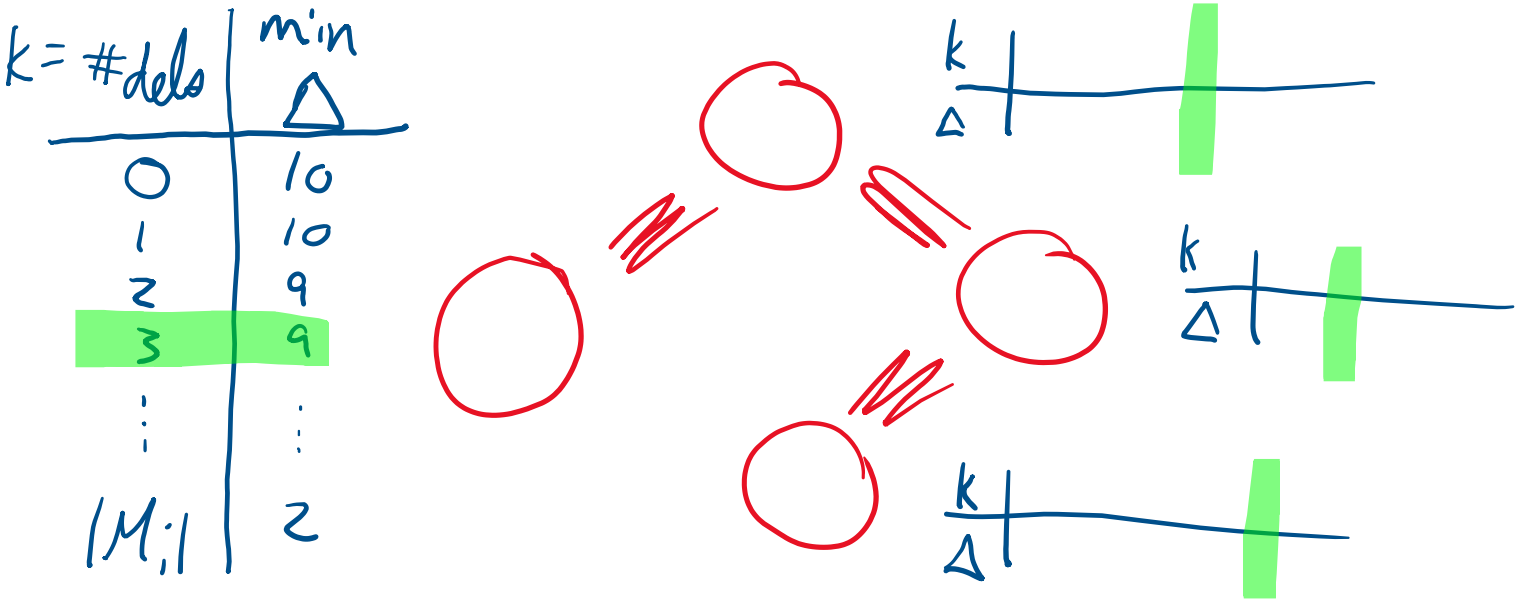


For general q, the "only" solution table idea that seems to work:

- Ask each $\boldsymbol{\mathcal{G}}$-module M_i,

  "hey, if I let you delete k vertices, what's the smallest max degree you can achieve?"

- $M_i[0]$ = smallest possible max degree if we delete 0 vertices
- $M_i[1]$ = smallest possible max degree if we delete 1 vertex
- ...
- $M_i[|M_i|]$ = smallest possible max degree if we delete all vertices



Idea: try all combinations of #deletions in the modules, check if they result in max degree at most q.

Idea: try all combinations of #deletions in the modules, check if they result in max degree at most q.

Problem: if each module has size around $\sim n$, the number of combinations is $\sim n^{\mathcal{G}\text{-mc}}$

This is not $f(\mathcal{G}\text{-}mc)\ poly(n)$.

Idea: try all combinations of #deletions in the modules, check if they result in max degree at most q.

Problem: if each module has size around $\sim n$, the number of combinations is $\sim n^{\mathcal{G}\text{-mc}}$

This is not $f(\mathcal{G}\text{-}mc)\ poly(n)$.

Is there a more clever solution table idea? Probably not:

Idea: try all combinations of #deletions in the modules, check if they result in max degree at most q.

Problem: if each module has size around $\sim n$, the number of combinations is $\sim n^{\mathcal{G}\text{-mc}}$

This is not $f(\mathcal{G}\text{-}mc)\ poly(n)$.

Is there a more clever solution table idea? Probably not:

Theorem: Bounded-degree deletion is W[1]-hard in parameter $\mathcal{G}$-mc, where $\mathcal{G}$ = "disjoint stars" or any graph class that contains it.

Generic hardness results

▶ **Theorem 10.** *The (α, β)-Linear Degree Domination problem is W[1]-hard parameterized in the following cases:*

1. *$\alpha = 0$, β is in the input, and the parameter is the $(\mathcal{G} \cup \mathcal{I})$-modular cardinality, where $\mathcal{G}$ is any graph class that admits arbitrary degree retention tables;*
2. *α is any fixed constant in the interval $(0, 1)$, β is any fixed constant in $\mathbb{Z}$, and the parameter is the* `stars`*-modular cardinality;*
3. *$\alpha = 1$, β is in the input, and the parameter is the $(\mathcal{G} \cup \mathcal{I})$-modular cardinality, where $\mathcal{G}$ is any graph class that admits arbitrary degree deletion tables.*

Generic hardness results

▶ **Theorem 10.** *The (α, β)-Linear Degree Domination problem is W[1]-hard parameterized in the following cases:*

1. *$\alpha = 0$, β is in the input, and the parameter is the $(\mathcal{G} \cup \mathcal{I})$-modular cardinality, where $\mathcal{G}$ is any graph class that admits arbitrary degree retention tables;*
2. *α is any fixed constant in the interval $(0, 1)$, β is any fixed constant in $\mathbb{Z}$, and the parameter is the* `stars`*-modular cardinality;*
3. *$\alpha = 1$, β is in the input, and the parameter is the $(\mathcal{G} \cup \mathcal{I})$-modular cardinality, where $\mathcal{G}$ is any graph class that admits arbitrary degree deletion tables.*

Generic positive results

▶ **Definition 20.** *Let $\mathcal{G}$ be a polynomial-time recognizable graph class. For $G \in \mathcal{G}$, suppose that $f^G(x)$ is the degree deletion function of $G = (V, E)$, where $x \in [0, |V|]$. We say that $\mathcal{G}$ admits a succinct solution table for* Bounded Degree Deletion *if, for every $G \in \mathcal{G}$, there exists a function $g^G : \mathbb{R} \to \mathbb{R}$ that satisfies at least one of the following conditions:*

1. *g^G is a constant piecewise linear function such that $f^G(x) = g^G(x)$ for every integer $x \in [0, |V(G)|]$,*
2. *g^G is a piecewise convex linear function such that $f^G(x) = \lceil g^G(x) \rceil$ for every integer $x \in [0, |V(G)|]$.*

Moreover, g^G can be described and constructed in polynomial time with respect to $|V(G)|$.

▶ **Theorem 21.** *Let $\mathcal{G}$ be a graph class that admits a succinct solution table for* Bounded Degree Deletion*. Assume a minimum $\mathcal{G}$-modular partition is given. Then,* Bounded Degree Deletion *is FPT parameterized by $\mathcal{G}$-modular cardinality.*

Generic positive results

▶ **Definition 20.** *Let $\mathcal{G}$ be a polynomial-time recognizable graph class. For $G \in \mathcal{G}$, suppose that $f^G(x)$ is the degree deletion function of $G = (V, E)$, where $x \in [0, |V|]$. We say that $\mathcal{G}$ admits a succinct solution table for* Bounded Degree Deletion *if, for every $G \in \mathcal{G}$, there exists a function $g^G : \mathbb{R} \to \mathbb{R}$ that satisfies at least one of the following conditions:*

1. *g^G is a constant piecewise linear function such that $f^G(x) = g^G(x)$ for every integer $x \in [0, |V(G)|]$,*
2. *g^G is a piecewise convex linear function such that $f^G(x) = \lceil g^G(x) \rceil$ for every integer $x \in [0, |V(G)|]$.*

Moreover, g^G can be described and constructed in polynomial time with respect to $|V(G)|$.

▶ **Theorem 21.** *Let $\mathcal{G}$ be a graph class that admits a succinct solution table for* Bounded Degree Deletion. *Assume a minimum $\mathcal{G}$-modular partition is given. Then,* Bounded Degree Deletion *is FPT parameterized by $\mathcal{G}$-modular cardinality.*

(α, β)-LDD problem	$\alpha = 0$ (k-dom)	$\alpha \in (0, 1)$ (α-dom $+\beta$)	$\alpha = 1$ (BDD)	Parameters
β is in the input	W[1]-h	W[1]-h	W[1]-h*	cw
	W[1]-h	W[1]-h	W[1]-h	mw
	W[1]-h	W[1]-h	W[1]-h	cograph-mc
	W[1]-h	W[1]-h	W[1]-h	itp
	W[1]-h	W[1]-h	W[1]-h	stars-mc
	open	open	FPT	cluster-mc
	FPT	FPT	FPT	nd
β is any constant	FPT	W[1]-h	FPT	cw
	FPT	W[1]-h	FPT	mw
	FPT	W[1]-h	FPT	cograph-mc
	FPT	W[1]-h	FPT	itp
	FPT	W[1]-h	FPT	stars-mc
	FPT	open	FPT	cluster-mc
	FPT	FPT	FPT	nd

Conclusion

- $\mathcal{G}$-modular decompositions can be useful for algorithms

- Questions for specific $\mathcal{G}$'s:
  - Structure of graphs with small $\mathcal{G}$-mc? Or high $\mathcal{G}$-mc?
  - e.g. forced induced subgraph characterization of graphs with $mw(G) = n$ [Chudnovsky, Kim, Oum & Seymour, 2015]
- Questions for generic $\mathcal{G}$'s:
  - Meta-theory for which $\mathcal{G}$'s admit efficient algorithms (or not)

- Open: unknown FPT complexity in mw (and some $\mathcal{G}$-mc)
  - MAX-CUT, MIN-EDGE-DOMINATING SET, TRIANGLE PACKING