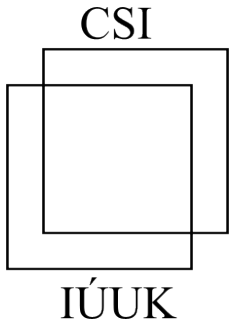
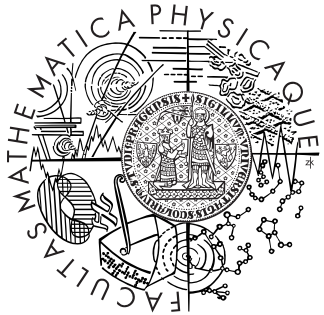
List 3-Colouring is polynomial on $(P_2 + P_5)$-Free and $(P_3 + P_4)$-Free Graphs.

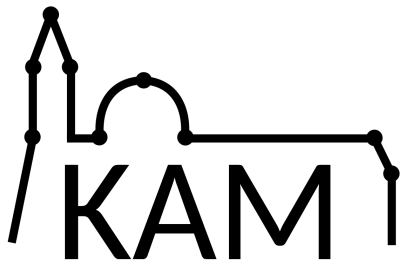
Tereza Klimošová, Josef Malík, Tomáš Masařík,
Jana Novotná, Daniël Paulusma, Veronika Slívová

Faculty of Mathematics and Physics,
Charles University,
Prague, Czech Republic.

Noon lecture 2018,
Prague, Czech Republic





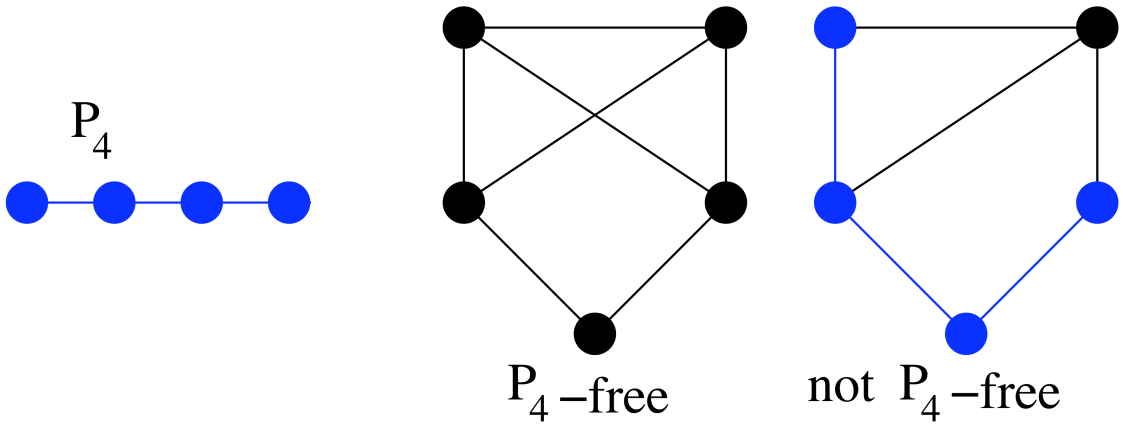


Preliminaries

- A graph G is H-free if G has no induced subgraph isomorphic to H.

Preliminaries

- A graph G is H-free if G has no induced subgraph isomorphic to H.

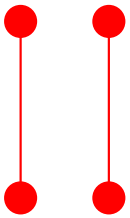


Preliminaries

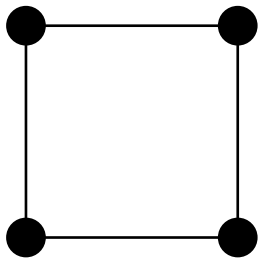
- A graph G is H-free if G has no induced subgraph isomorphic to H.
- A linear forest is a disjoint union of paths.

Preliminaries

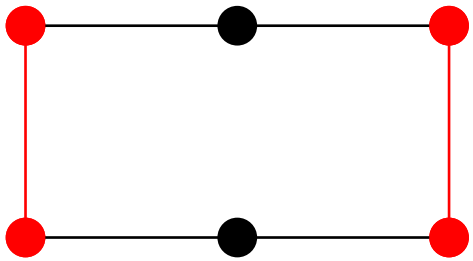
- A graph G is H-free if G has no induced subgraph isomorphic to H.
- A linear forest is a disjoint union of paths.



$2P_2$



$2P_2$-free



not $2P_2$-free

Preliminaries

- A graph G is H-free if G has no induced subgraph isomorphic to H.
- A linear forest is a disjoint union of paths.

k-Colouring: Given a graph G, does there exists a colouring of G which uses at most k colours?

Preliminaries

- A graph G is H-free if G has no induced subgraph isomorphic to H.
- A linear forest is a disjoint union of paths.

k-Colouring: Given a graph G, does there exists a colouring of G which uses at most k colours?

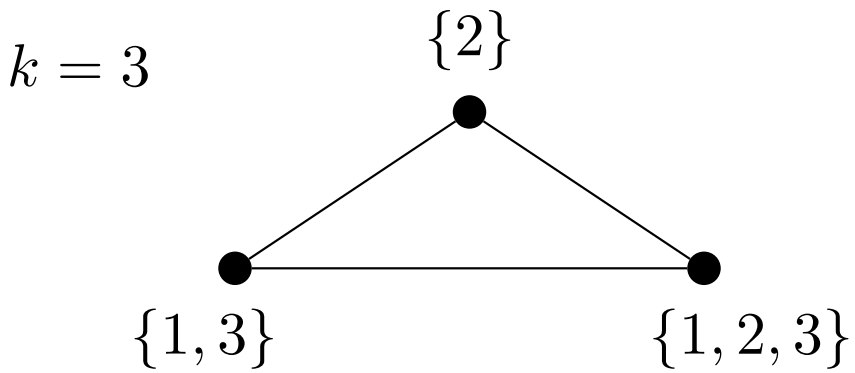
List k-Colouring: Given a graph G and a list assignment L, a subset of colours $\{1, \ldots, k\}$, does there exist a colouring of G that respects L?

Preliminaries

- A graph G is H-free if G has no induced subgraph isomorphic to H.
- A linear forest is a disjoint union of paths.

k-Colouring: Given a graph G, does there exists a colouring of G which uses at most k colours?

List k-Colouring: Given a graph G and a list assignment L, a subset of colours $\{1, \ldots, k\}$, does there exist a colouring of G that respects L?

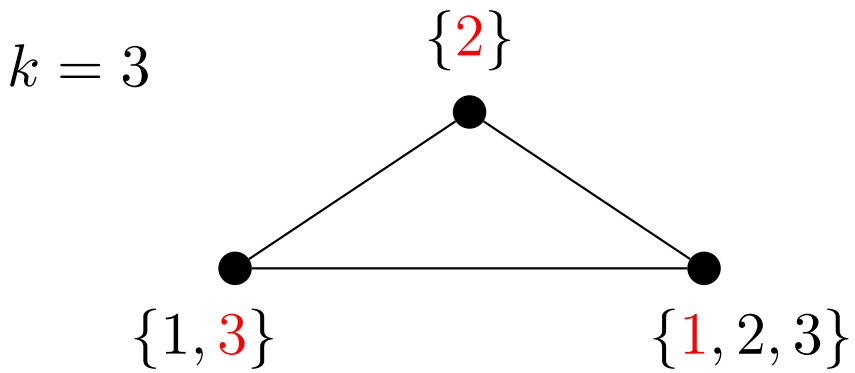


Preliminaries

- A graph G is H-free if G has no induced subgraph isomorphic to H.
- A linear forest is a disjoint union of paths.

k-Colouring: Given a graph G, does there exists a colouring of G which uses at most k colours?

List k-Colouring: Given a graph G and a list assignment L, a subset of colours $\{1, \ldots, k\}$, does there exist a colouring of G that respects L?



Preliminaries

- A graph G is H-free if G has no induced subgraph isomorphic to H.
- A linear forest is a disjoint union of paths.

k-Colouring: Given a graph G, does there exists a colouring of G which uses at most k colours?

List k-Colouring: Given a graph G and a list assignment L, a subset of colours $\{1, \ldots, k\}$, does there exist a colouring of G that respects L?

- k-List Colouring: $\forall v \in V : |L(v)| \leq k$.

State of the art (k-Colouring on H-free graphs)

- For every $k \geq 3$, k-Colouring on H-free graphs is NP-complete if H contains
  - an induced claw [Holyer 81 & Leven, Galil 83] or

State of the art (k-Colouring on H-free graphs)

- For every $k \geq 3$, k-Colouring on H-free graphs is NP-complete if H contains
  - an induced claw [Holyer 81 & Leven, Galil 83] or
  - a cycle [Emden-Weinert, Hougardy, Kreuter 98].
-

State of the art (k-Colouring on H-free graphs)

- For every $k \geq 3$, k-Colouring on H-free graphs is NP-complete if H contains
  - an induced claw [Holyer 81 & Leven, Galil 83] or
  - a cycle [Emden-Weinert, Hougardy, Kreuter 98].
- Hence, it remains to consider the case where H is a linear forest

State of the art (k-Colouring on H-free graphs)

- For every $k \geq 3$, k-Colouring on H-free graphs is NP-complete if H contains
  - an induced claw [Holyer 81 & Leven, Galil 83] or
  - a cycle [Emden-Weinert, Hougardy, Kreuter 98].
- Hence, it remains to consider the case where H is a linear forest
  - paths (P_t-free graphs),
  - disjoint union of more than one path.

State of the art (P_t-free graphs)

	k-Colouring			
t	$k=3$	$k=4$	$k=5$	$k\geq 6$
$t\leq 5$	P	P	P	P
$t=6$	P	P	NP-c	NP-c
$t=7$	P	NP-c	NP-c	NP-c
$t\geq 8$	?	NP-c	NP-c	NP-c

	List k-Colouring			
t	$k=3$	$k=4$	$k=5$	$k\geq 6$
$t\leq 5$	P	P	P	P
$t=6$	P	NP-c	NP-c	NP-c
$t=7$	P	NP-c	NP-c	NP-c
$t\geq 8$	?	NP-c	NP-c	NP-c

State of the art (linear forests)

Theorem [survey: Golovach, Johnson, Paulusma, Song 17]

For a graph H with $|V(H)| \leq 6$, 3-Colouring and List 3-Colouring are polynomial-time solvable on H-free graphs if and only if H is a linear forest.

D. Paulusma, C&C 2017

3-Coloring for H-free graphs is classified if $|V(H)| \leq 7$ except when

- $H = P_2 + P_5$
- $H = P_3 + P_4$.

Our results

Theorem

List 3-Colouring is polynomial-time solvable for $(P_2 + P_5)$-free graphs and for $(P_3 + P_4)$-free graphs.

Our results

Theorem

List 3-Colouring is polynomial-time solvable for $(P_2 + P_5)$-free graphs and for $(P_3 + P_4)$-free graphs.

- Corollary: For a graph H with $|V(H)| \leq 7$, 3-Colouring and List 3-Colouring are polynomial-time solvable on H-free graphs if and only if H is a linear forest.

Our results

Theorem

List 3-Colouring is polynomial-time solvable for $(P_2 + P_5)$-free graphs and for $(P_3 + P_4)$-free graphs.

- Corollary: For a graph H with $|V(H)| \leq 7$, 3-Colouring and List 3-Colouring are polynomial-time solvable on H-free graphs if and only if H is a linear forest.

Theorem

5-Colouring is NP-complete for $(P_3 + P_5)$-free graphs.

Our results

Theorem

List 3-Colouring is polynomial-time solvable for $(P_2 + P_5)$-free graphs and for $(P_3 + P_4)$-free graphs.

- Corollary: For a graph H with $|V(H)| \leq 7$, 3-Colouring and List 3-Colouring are polynomial-time solvable on H-free graphs if and only if H is a linear forest.

Theorem

5-Colouring is NP-complete for $(P_3 + P_5)$-free graphs.

Theorem (Chudnovsky, Huang, Spirkl, Zhong 2018+)

List 3-Colouring is polynomial-time solvable for $(kP_3 + P_6)$-free graphs.

Part of the proof

for $(P_2 + P_5)$-free and $(P_3 + P_4)$-free graphs

Simple example of our techniques

Idea

Find a small (constant size) set of vertices

Simple example of our techniques

Idea

Find a small (constant size) set of vertices → try all possibilities to colour this set

Simple example of our techniques

Idea

Find a small (constant size) set of vertices → try all possibilities to colour this set → obtain a smaller instance

Simple example of our techniques

Idea

Find a small (constant size) set of vertices → try all possibilities to colour this set → obtain a smaller instance

Simple example

- If G has a constant size Dominating set,

Simple example of our techniques

Idea

Find a small (constant size) set of vertices → try all possibilities to colour this set → obtain a smaller instance

Simple example

- If G has a constant size Dominating set,
  - (i.e., every vertex is in D or in the neighbourhood of D)

Simple example of our techniques

Idea

Find a small (constant size) set of vertices → try all possibilities to colour this set → obtain a smaller instance

Simple example

- If G has a constant size Dominating set,
  - (i.e., every vertex is in D or in the neighbourhood of D)
- dominating: If we colour D, we reduce all lists of vertices in G at least by one

Simple example of our techniques

Idea

Find a small (constant size) set of vertices → try all possibilities to colour this set → obtain a smaller instance

Simple example

- If G has a constant size Dominating set,
  - (i.e., every vertex is in D or in the neighbourhood of D)
- dominating: If we colour D, we reduce all lists of vertices in G at least by one (instance of 2-list coloring).

Simple example of our techniques

Idea

Find a small (constant size) set of vertices → try all possibilities to colour this set → obtain a smaller instance

Simple example

- If G has a constant size Dominating set,
  - (i.e., every vertex is in D or in the neighbourhood of D)
- dominating: If we colour D, we reduce all lists of vertices in G at least by one (instance of 2-list coloring).
- constant size: we can try all possible colourings of D

Simple example of our techniques

Idea

Find a small (constant size) set of vertices → try all possibilities to colour this set → obtain a smaller instance

Simple example

- If G has a constant size Dominating set,
  - (i.e., every vertex is in D or in the neighbourhood of D)
- dominating: If we colour D, we reduce all lists of vertices in G at least by one (instance of 2-list coloring).
- constant size: we can try all possible colourings of D → we can examine all of them.

Proof: Used tools

Theorem [Bonomo, Chudnovsky, Maceli, Schaudt, Stein, Zhong 17]

List 3-Colouring is polynomial-time solvable for P_7-free graphs.

Proof: Used tools

Theorem [Bonomo, Chudnovsky, Maceli, Schaudt, Stein, Zhong 17]

List 3-Colouring is polynomial-time solvable for P_7-free graphs.

Theorem [Edwards 86]

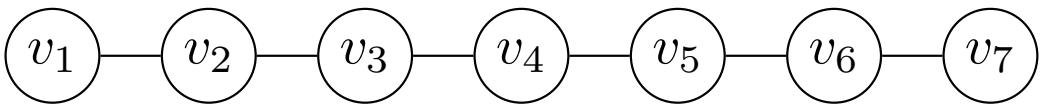
2-List Colouring is polynomial-time solvable for general graphs.

Proof Sketch

- We are using $(P_3 + P_5)$ (whenever we are able to do so)

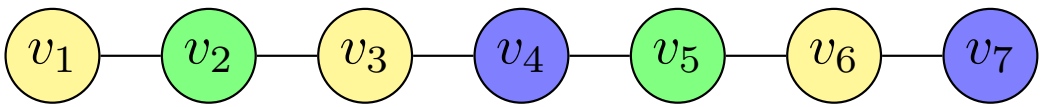
Proof Sketch

- We are using $(P_3 + P_5)$ (whenever we are able to do so)
- Assume that G contains an induced P_7,



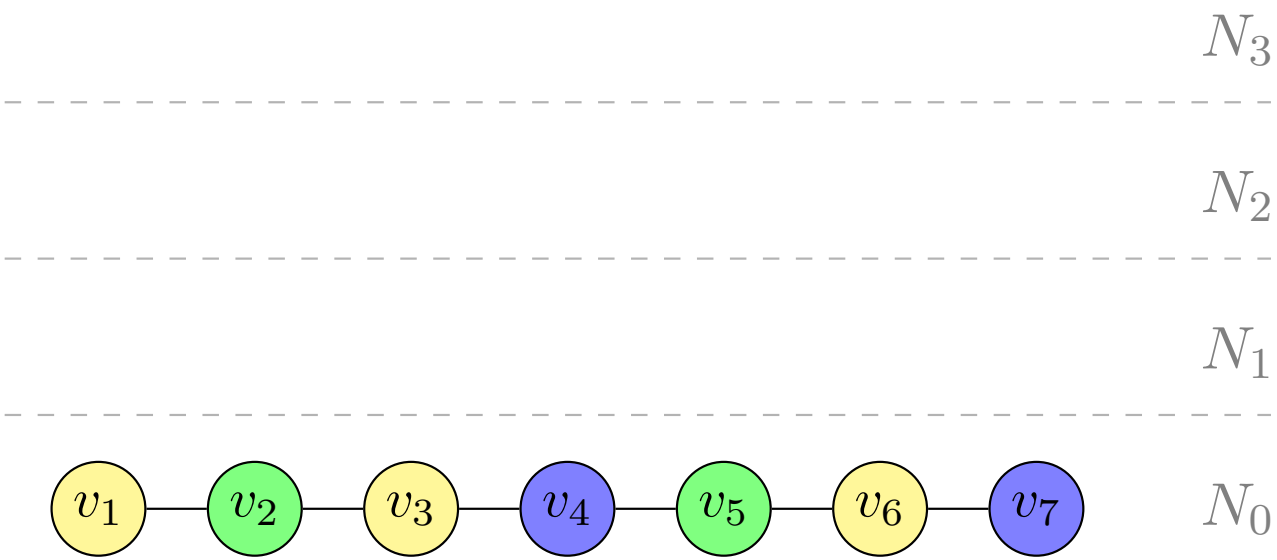
Proof Sketch

- We are using $(P_3 + P_5)$ (whenever we are able to do so)
- Assume that G contains an induced P_7,
- Consider every possibility of colouring the vertices of this P_7,



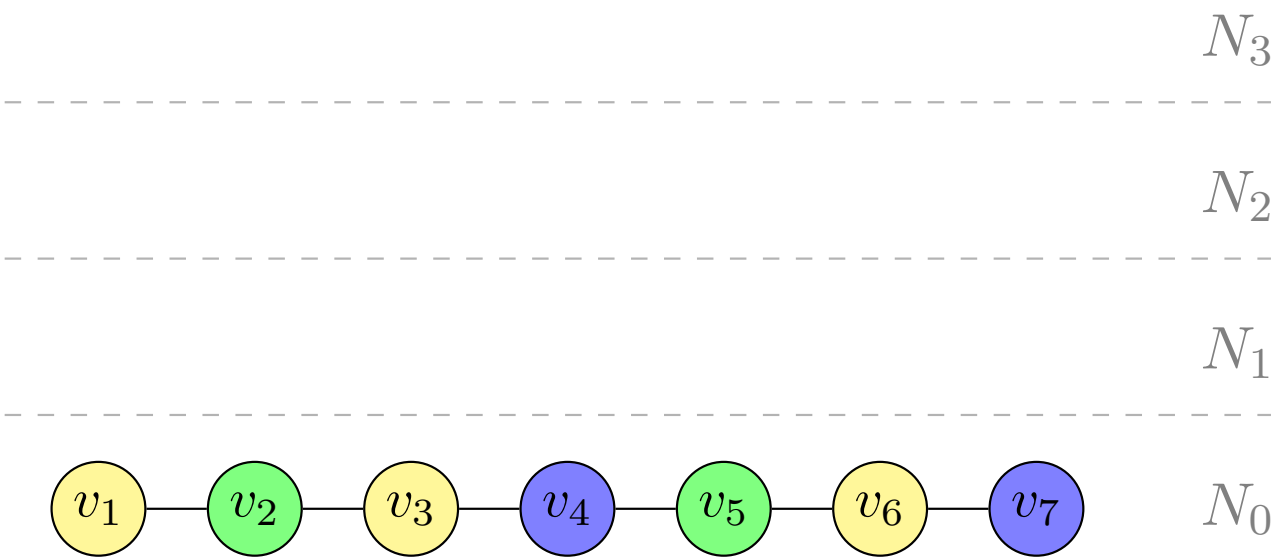
Proof Sketch

- We are using $(P_3 + P_5)$ (whenever we are able to do so)
- Assume that G contains an induced P_7,
- Consider every possibility of colouring the vertices of this P_7,
- Partition vertices of G into four "layers" according to their distance to the P_7,



Proof Sketch

- We are using $(P_3 + P_5)$ (whenever we are able to do so)
- Assume that G contains an induced P_7,
- Consider every possibility of colouring the vertices of this P_7,
- Partition vertices of G into four "layers" according to their distance to the P_7,
- Reduce to a polynomial number of instances of 2-List Colouring.



Active vertices

- We want to reduce vertices with lists of size 3,

Active vertices

- We want to reduce vertices with lists of size 3,
  $\rightarrow$ they are only in N_2.

Active vertices

- We want to reduce vertices with lists of size 3,
  $\rightarrow$ they are only in N_2.

Definition

We call a vertex u in N_2 and its neighbours in N_1 active if $|L(u)| = 3$.

- We use A_1, A_2 for the set of all active vertices in N_1, N_2.

Active vertices

- We want to reduce vertices with lists of size 3,
  $\rightarrow$ they are only in N_2.

Definition

We call a vertex u in N_2 and its neighbours in N_1 active if $|L(u)| = 3$.

- We use A_1, A_2 for the set of all active vertices in N_1, N_2.

Goal

Deactivate all active vertices.

Overview of the polynomial algorithm

- Preprocessing
- Phase 1: Reduce private neighbours of non-adjacent vertices in P_7
- Phase 2: Two types of lists in the first layer
- Phase 3: One type of lists in the first layer

Overview of the polynomial algorithm

- Preprocessing
- Phase 1: Reduce private neighbours of non-adjacent vertices in P_7
- Phase 2: Two types of lists in the first layer
- Phase 3: One type of lists in the first layer

Preprocessing

- Set of rules which are applied exhaustively.

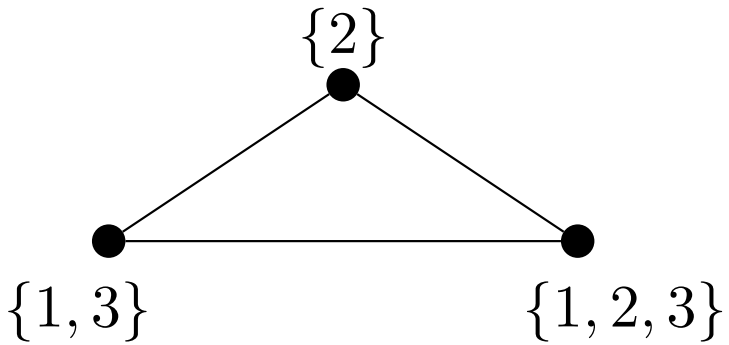
Preprocessing

- Set of rules which are applied exhaustively.
- Examples

Preprocessing

- Set of rules which are applied exhaustively.
- Examples

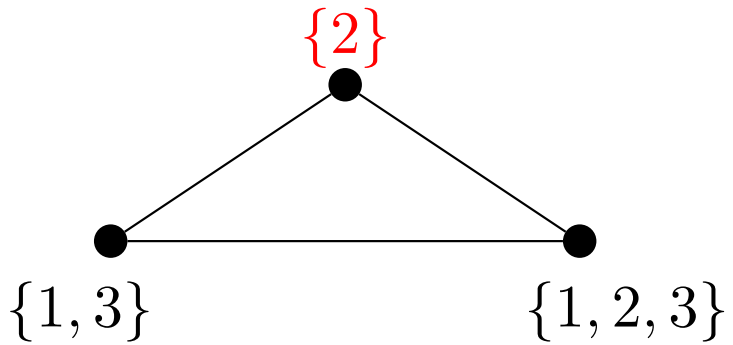
  Rule 5. (single colour propagation) If u and v are adjacent, $|L(u)| = 1$, and $L(u) \subseteq L(v)$, then set $L(v) := L(v) \setminus L(u)$.



Preprocessing

- Set of rules which are applied exhaustively.
- Examples

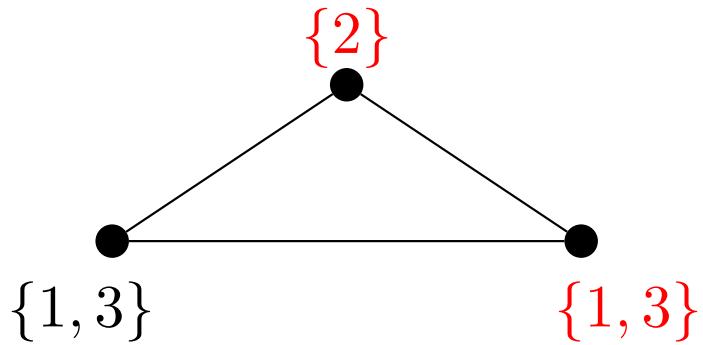
  Rule 5. (single colour propagation) If u and v are adjacent, $|L(u)| = 1$, and $L(u) \subseteq L(v)$, then set $L(v) := L(v) \setminus L(u)$.



Preprocessing

- Set of rules which are applied exhaustively.
- Examples

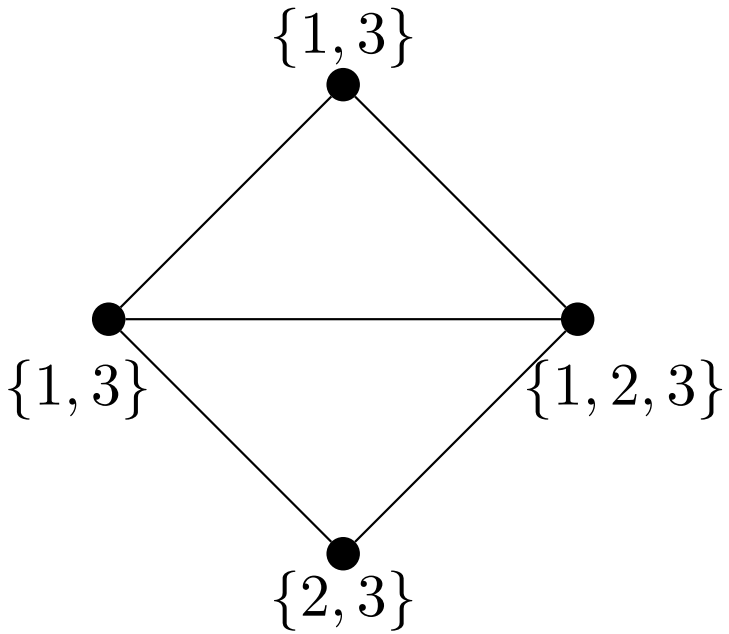
  Rule 5. (single colour propagation) If u and v are adjacent, $|L(u)| = 1$, and $L(u) \subseteq L(v)$, then set $L(v) := L(v) \setminus L(u)$.



Preprocessing

- Set of rules which are applied exhaustively.
- Examples

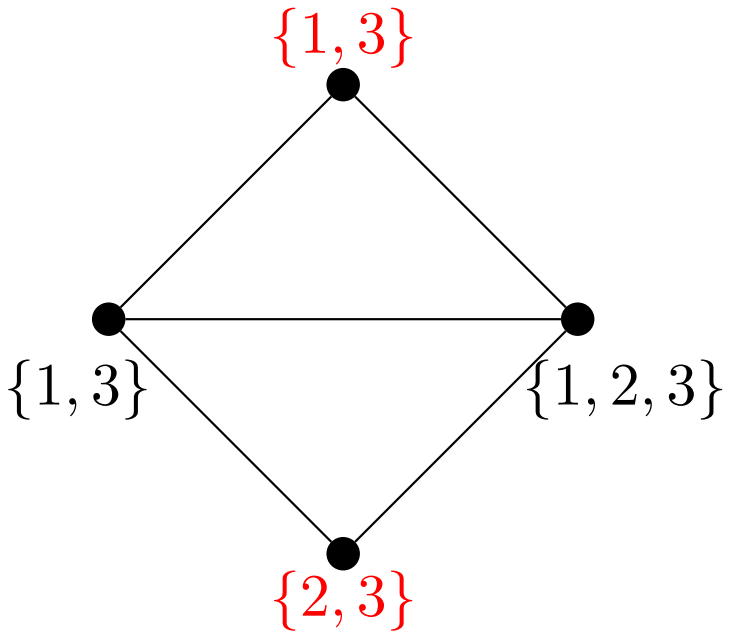
  Rule 6. (diamond colour propagation) If u and v are adjacent and share two common neighbours x and y with $L(x) \neq L(y)$, then set $L(x) := L(x) \cap L(y)$ and $L(y) := L(x) \cap L(y)$.



Preprocessing

- Set of rules which are applied exhaustively.
- Examples

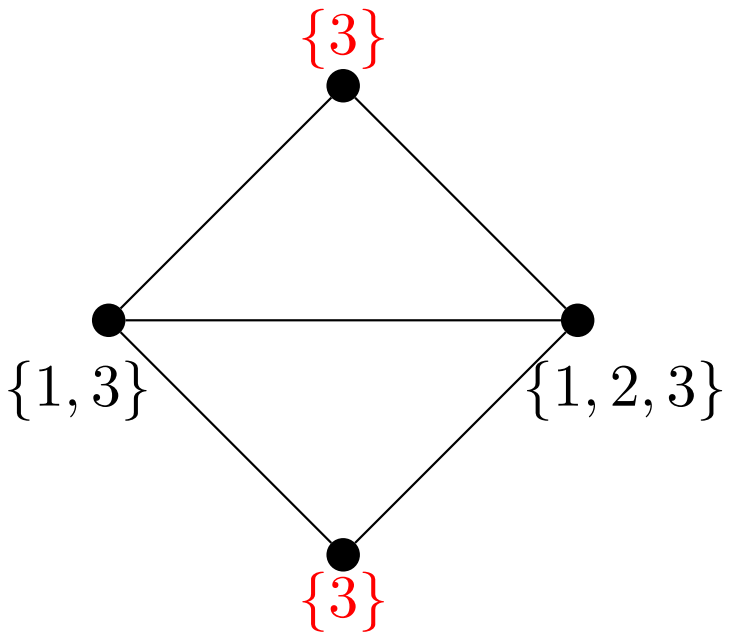
  Rule 6. (diamond colour propagation) If u and v are adjacent and share two common neighbours x and y with $L(x) \neq L(y)$, then set $L(x) := L(x) \cap L(y)$ and $L(y) := L(x) \cap L(y)$.



Preprocessing

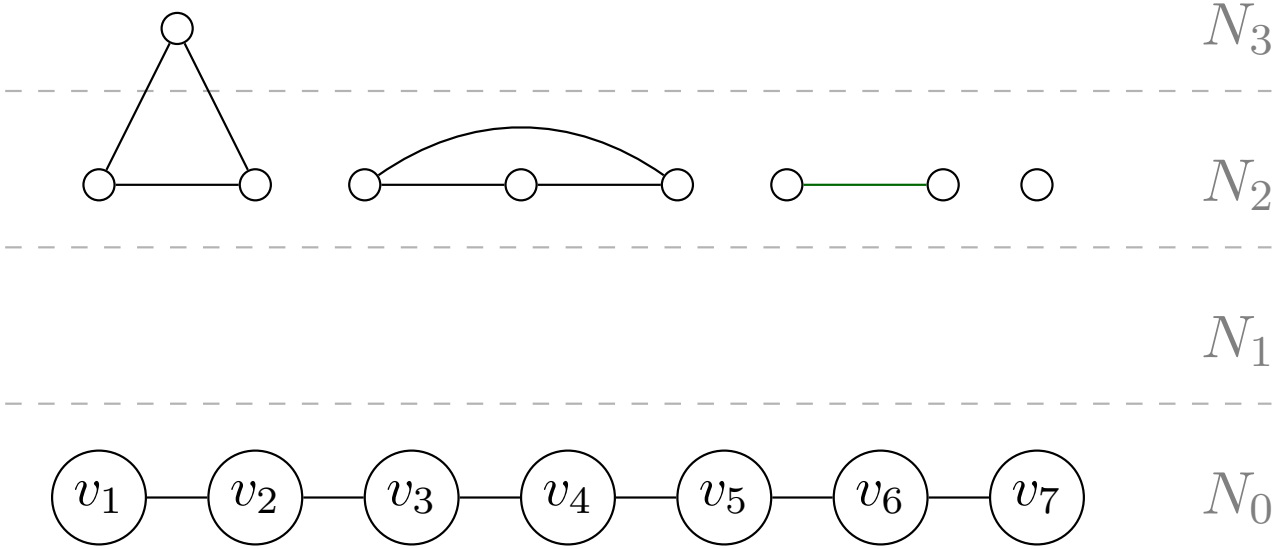
- Set of rules which are applied exhaustively.
- Examples

  Rule 6. (diamond colour propagation) If u and v are adjacent and share two common neighbours x and y with $L(x) \neq L(y)$, then set $L(x) := L(x) \cap L(y)$ and $L(y) := L(x) \cap L(y)$.



Preprocessing

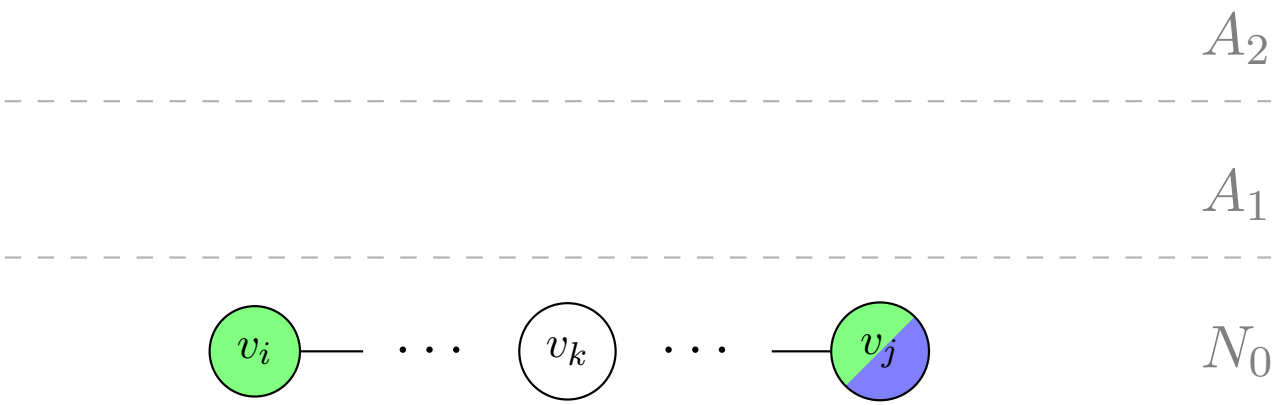
- Set of rules which are applied exhaustively.
- Examples
- We can enforce strong structure on $N_2 \cup N_3$:



Overview of the polynomial algorithm

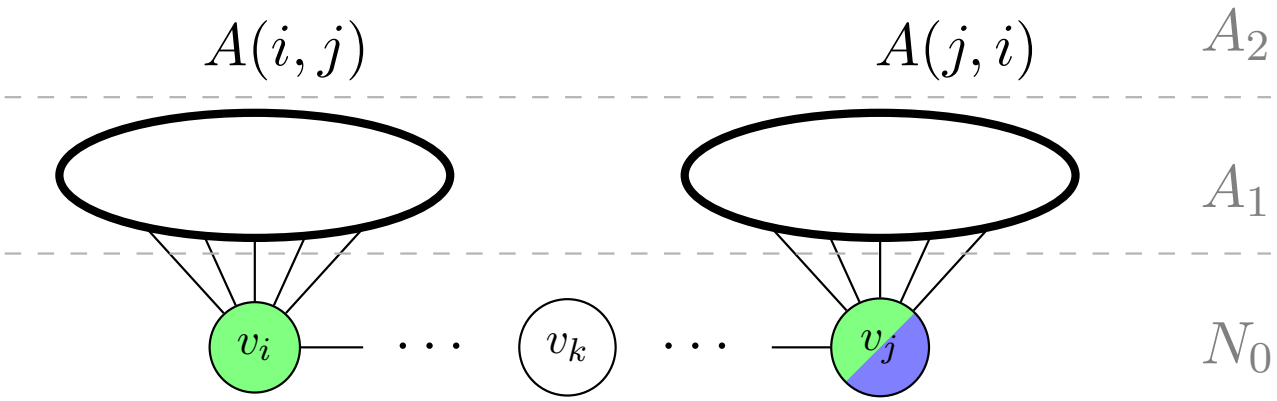
- ✓ Preprocessing
- Phase 1: Reduce private neighbours of non-adjacent vertices in P_7
- Phase 2: Two types of lists in the first layer
- Phase 3: One type of lists in the first layer

Phase I



Phase I

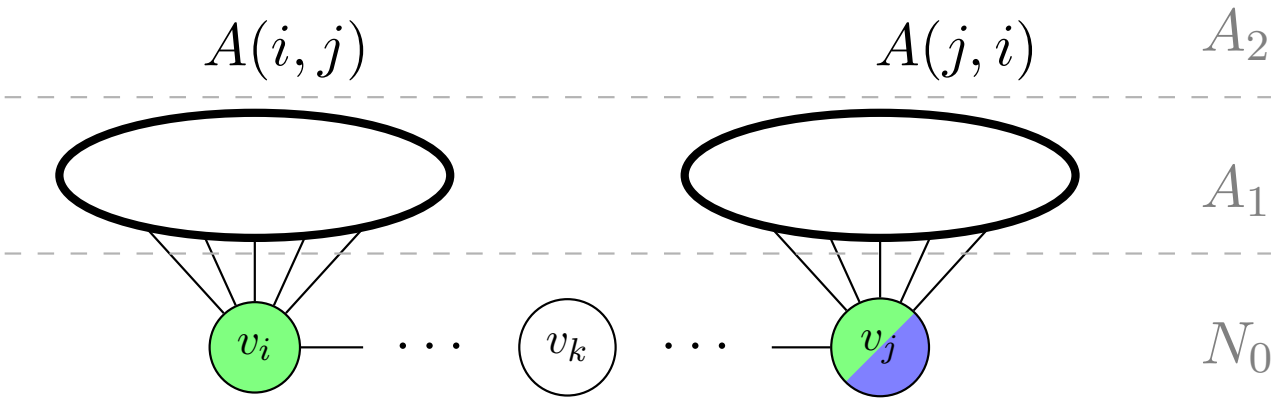
- $A(i,j)\ldots$ set of active vertices in N_1 adjacent to v_i and nonadjacent to v_j, similarly $A(j,i)$.



Phase I

Goal

To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

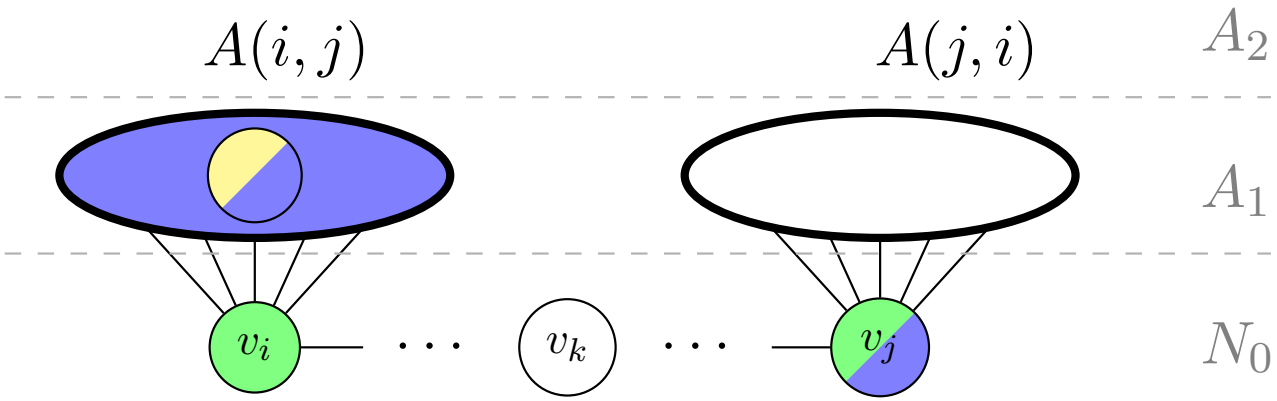


Phase I

Goal

To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

- Branching: at most one vertex in $A(i,j)$ is yellow $\rightarrow$ $n+1$ branches,

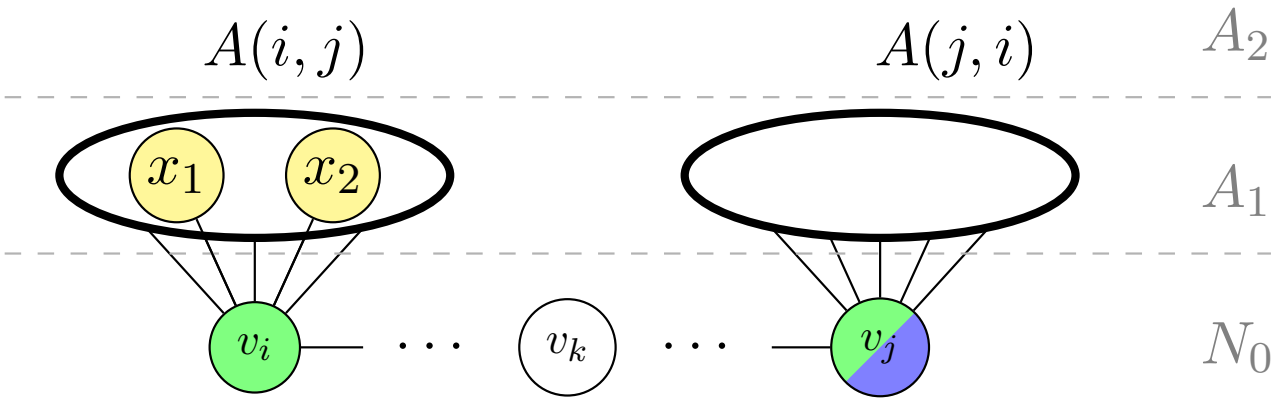


Phase I

Goal

To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

- Branching: at most one vertex in $A(i,j)$ is yellow $\rightarrow$ $n+1$ branches, or two vertices are yellow $\rightarrow$ n^2 branches.

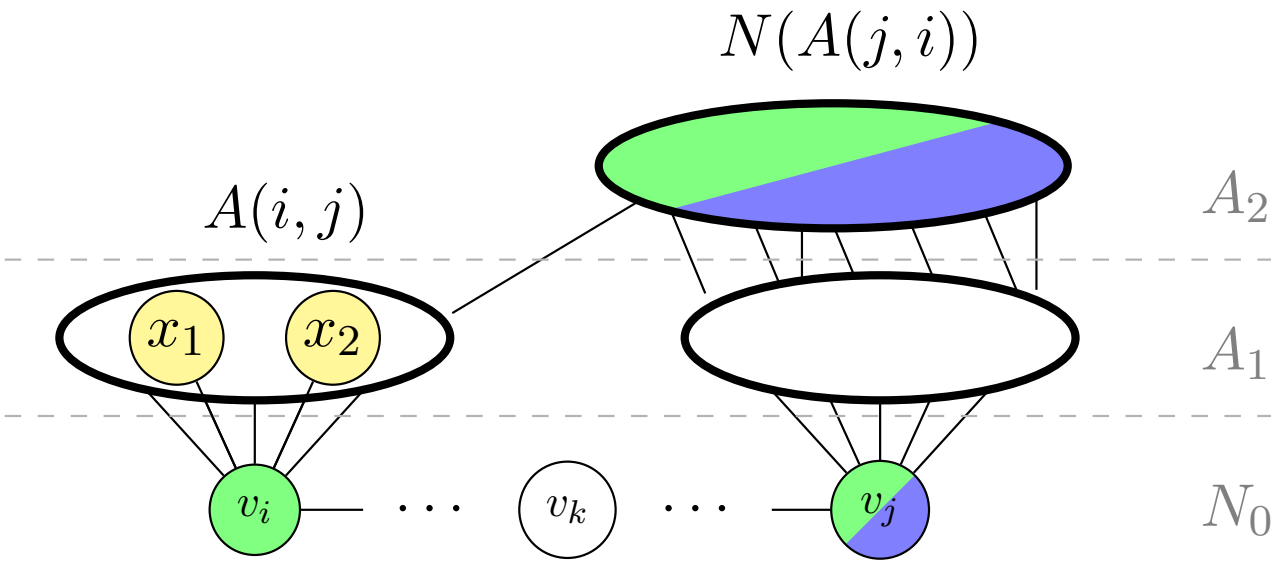


Phase I

Goal

To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

- Branching: Each vertex in $N(A(j,i)) \cap A_2$ given the color of v_j $\rightarrow$ one branch,

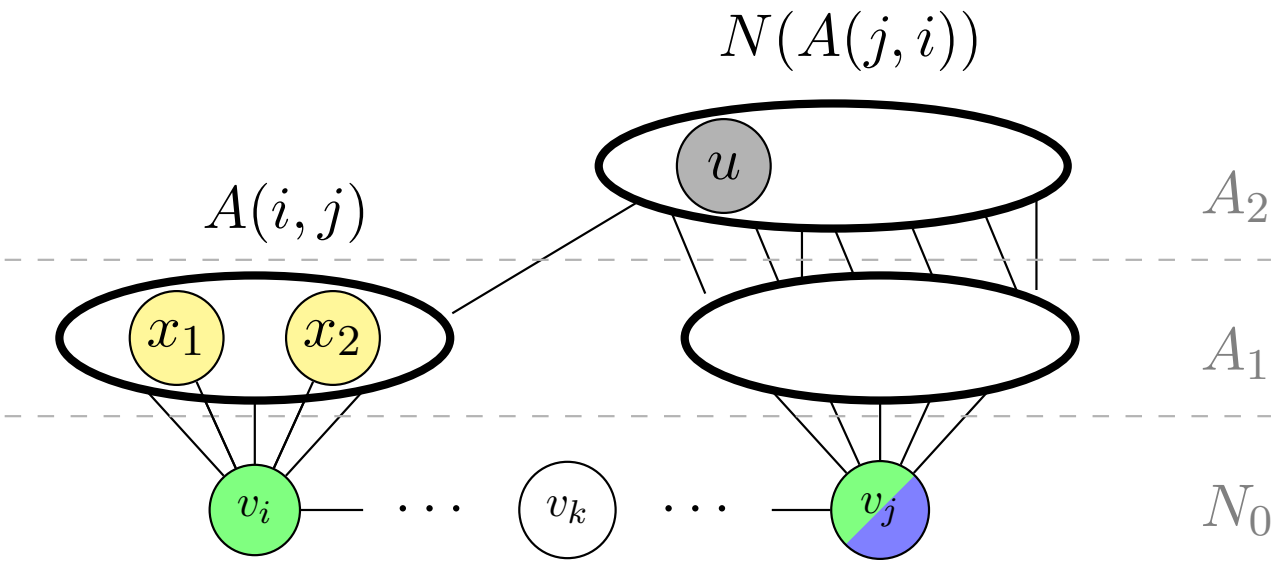


Phase I

Goal

To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

- Branching: Each vertex in $N(A(j,i)) \cap A_2$ given the color of v_j
  $\rightarrow$ one branch, or one vertex in $N(A(j,i)) \cap A_2$ has different color
  $\rightarrow 2n$ branches.

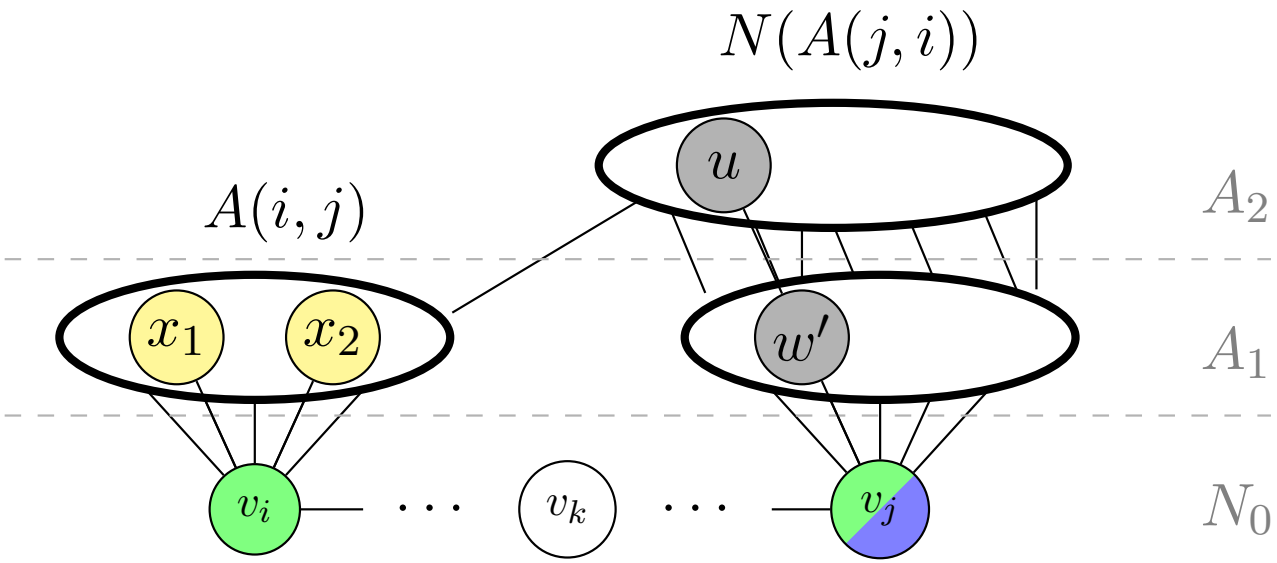


Phase I

Goal

To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

- Either $A(j,i)$ is deactivated, or we find an induced $P_5 + P_3$.

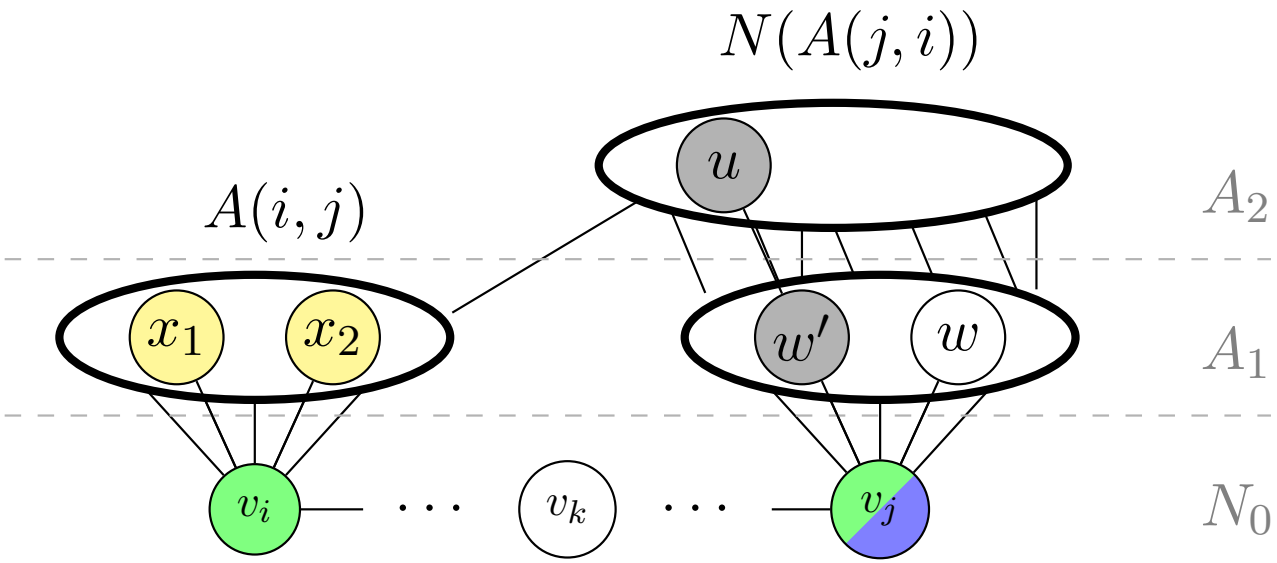


Phase I

Goal

To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

- Either $A(j,i)$ is deactivated, or we find an induced $P_5 + P_3$.

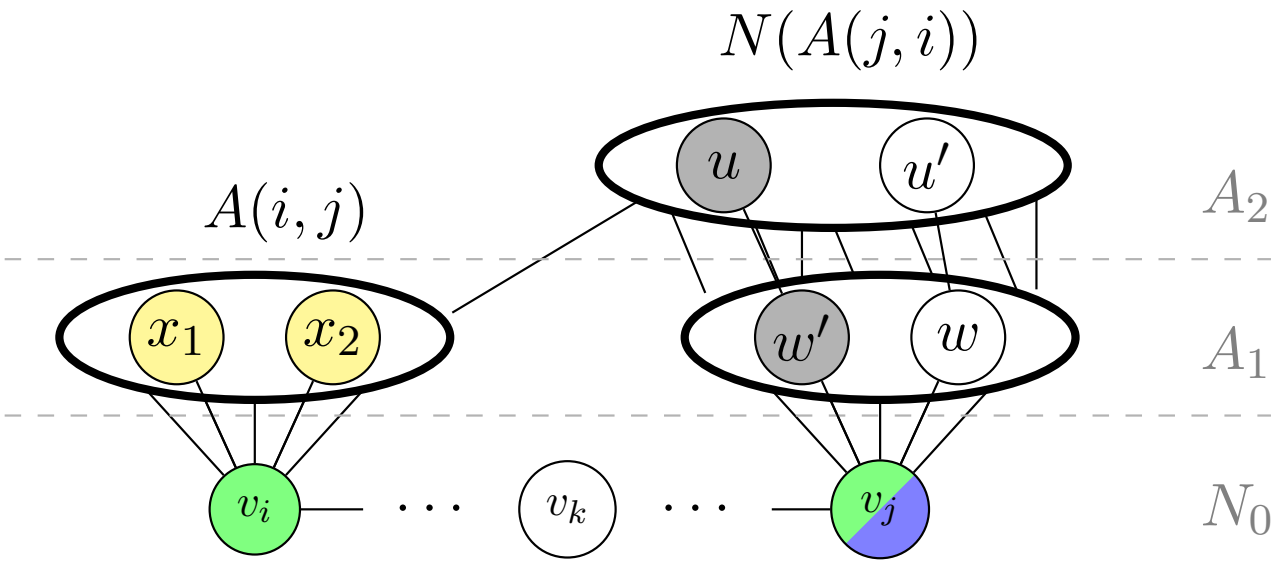


Phase I

Goal

To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

- Either $A(j,i)$ is deactivated, or we find an induced $P_5 + P_3$.

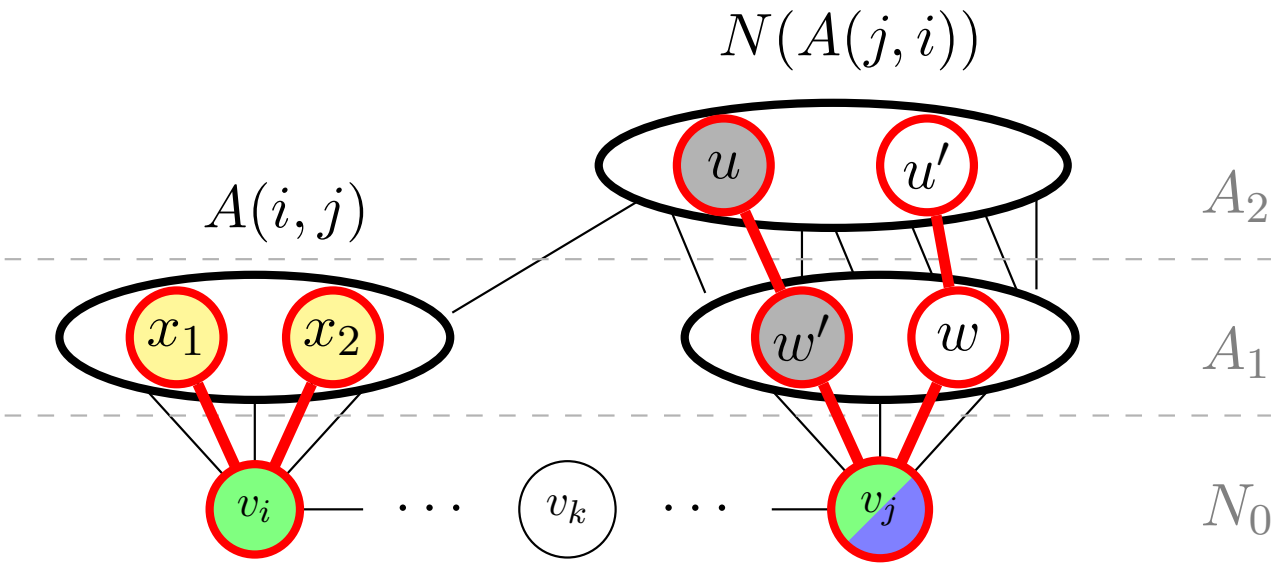


Phase I

Goal

To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

- Either $A(j,i)$ is deactivated, or we find an induced $P_5 + P_3$.

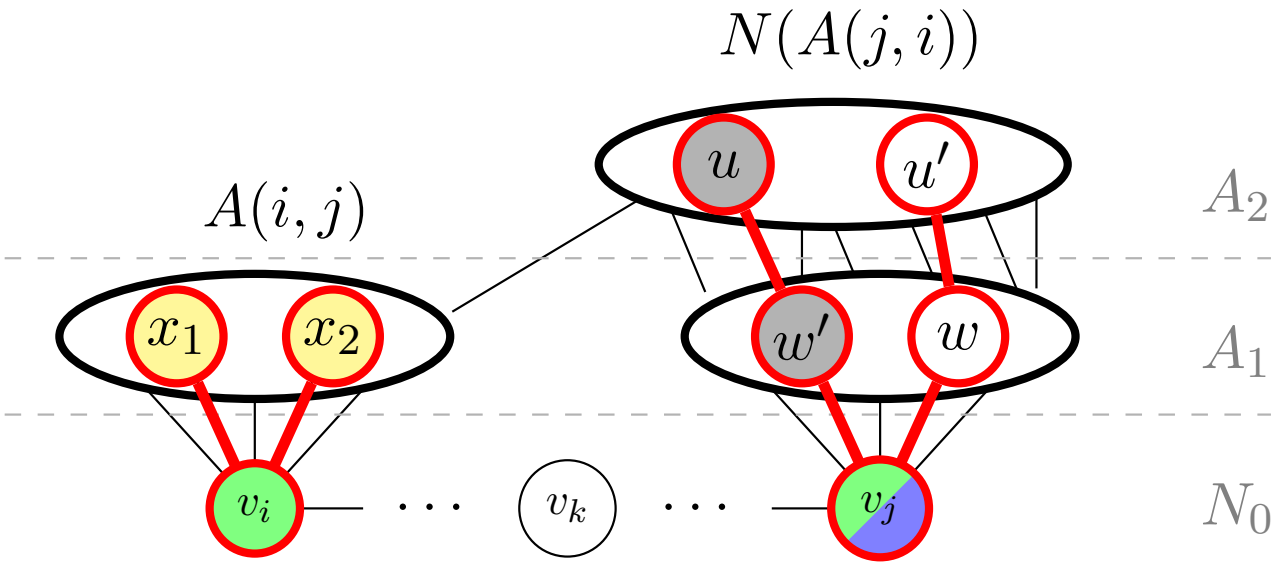


Phase I

Goal

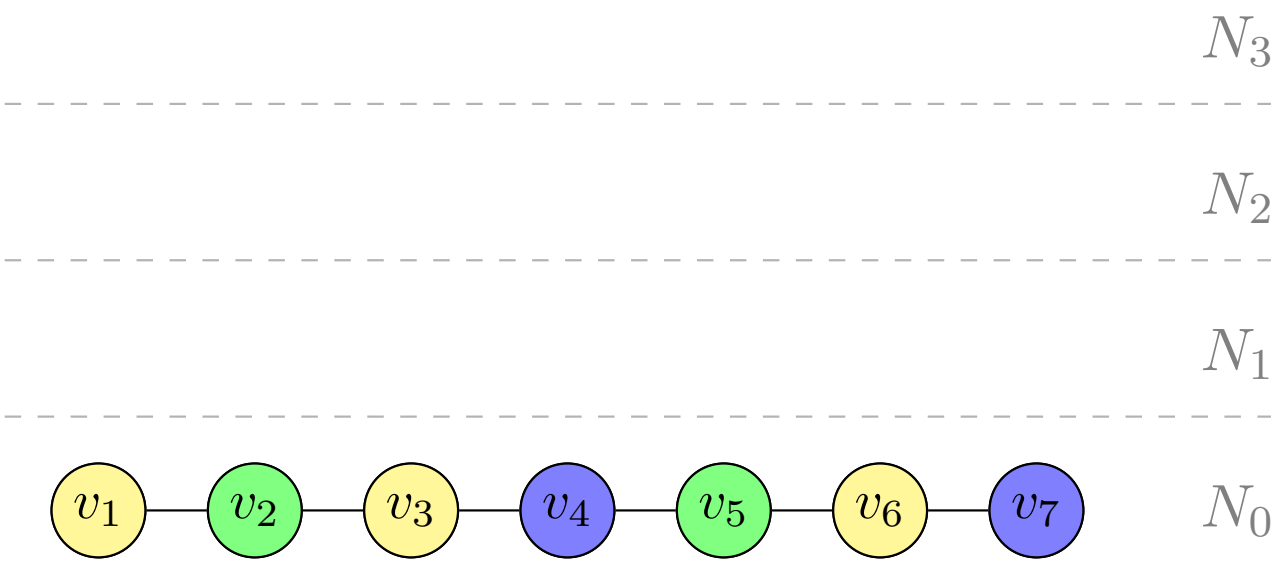
To deactivate at least one of $A(i,j), A(j,i)$ for nonadjacent v_i, v_j.

- $A(i,j)$ or $A(j,i)$ can be deactivated for all nonadjacent vertices v_i, $v_j \in N_0$



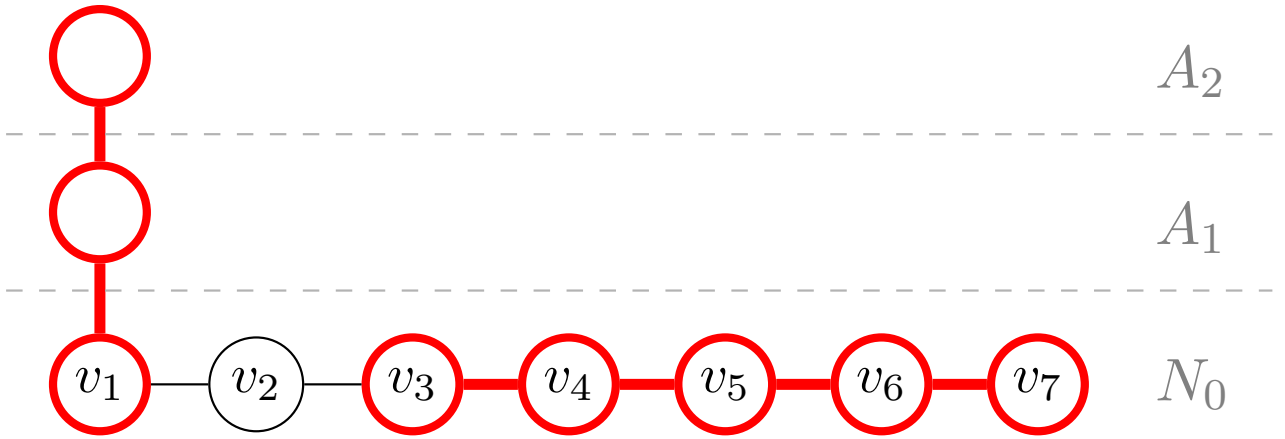
Phase I: Consequences

- $A(i,j)$ or $A(j,i)$ is empty for nonadjacent v_i, v_j
  $\rightarrow$ at most two types of lists of $\{$●,●$\}$, $\{$●,●$\}$, $\{$●,●$\}$ can occur in A_1,



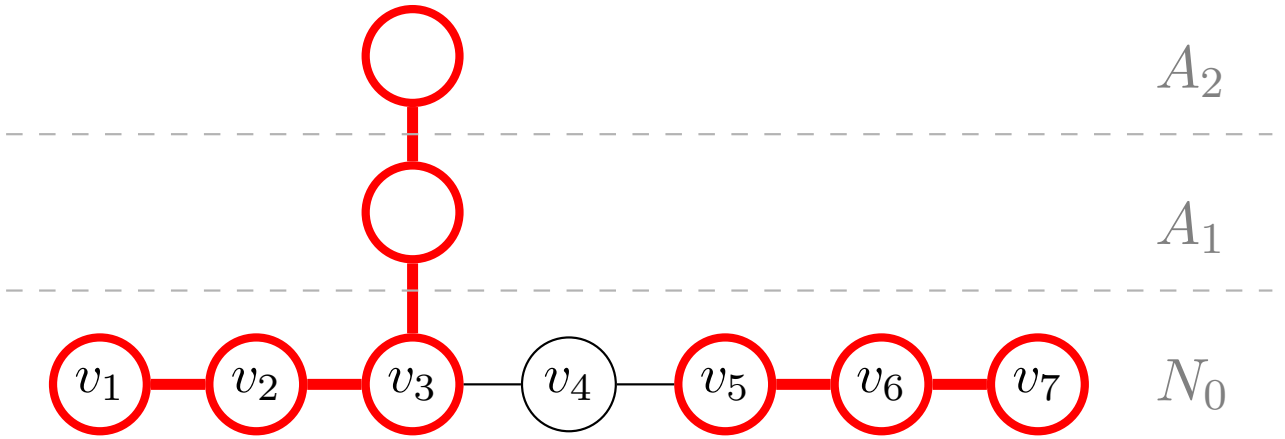
Phase I: Consequences

- $A(i,j)$ or $A(j,i)$ is empty for nonadjacent v_i, v_j
  $\rightarrow$ at most two types of lists of $\{$●,●$\}$, $\{$●,●$\}$, $\{$●,●$\}$ can occur in A_1,
- neighbours of v_i in A_1, $i \in \{1,3,5,7\}$, have another neighbour in N_0.



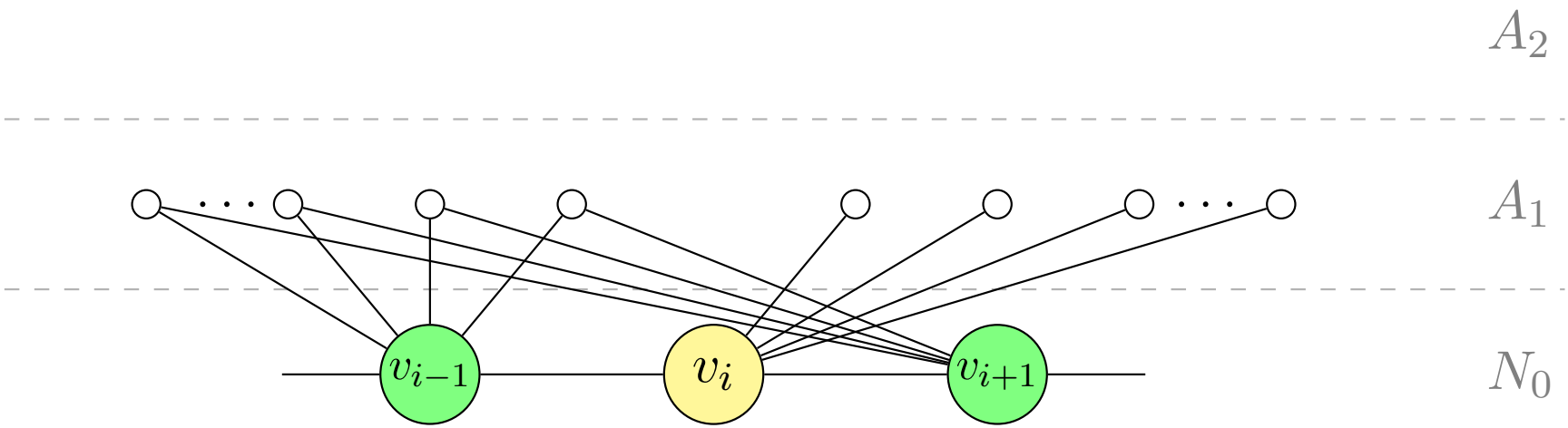
Phase I: Consequences

- $A(i,j)$ or $A(j,i)$ is empty for nonadjacent v_i, v_j
  $\rightarrow$ at most two types of lists of $\{$●,●$\}$, $\{$●,●$\}$, $\{$●,●$\}$ can occur in A_1,
- neighbours of v_i in A_1, $i \in \{1,3,5,7\}$, have another neighbour in N_0.



Overview of the polynomial algorithm

- ✓ Preprocessing
- ✓ Phase 1: Reduce private neighbours of non-adjacent vertices in P_7
- Phase 2: Two types of lists in the first layer ({●,●},{●,●})
- Phase 3: One type of lists in the first layer({●,●})



Overview of the polynomial algorithm

- ✓ Preprocessing
- ✓ Phase 1: Reduce private neighbours of non-adjacent vertices in P_7
- ✓ Phase 2: Two types of lists in the first layer ($\{$  ,  $\}$, $\{$  ,  $\}$)
- Phase 3: One type of lists in the first layer ($\{$  ,  $\}$)

- Colour all vertices in A_2 by the colour which is not in A_1.

Overview of the polynomial algorithm

- ✓ Preprocessing
- ✓ Phase 1: Reduce private neighbours of non-adjacent vertices in P_7
- ✓ Phase 2: Two types of lists in the first layer ($\{$  ,  $\}$, $\{$  ,  $\}$)
- Phase 3: One type of lists in the first layer ($\{$  ,  $\}$)

$(P_2 + P_5)$-free graphs

- N_2 is an independent set.

Overview of the polynomial algorithm

- ✓ Preprocessing
- ✓ Phase 1: Reduce private neighbours of non-adjacent vertices in P_7
- ✓ Phase 2: Two types of lists in the first layer ($\{$  ,  $\}$, $\{$,  $\}$)
- Phase 3: One type of lists in the first layer ($\{$  ,  $\}$)

$(P_2 + P_5)$-free graphs

- N_2 is an independent set.
- Colour all vertices in A_2 by the colour which is not in A_1.

Overview of the polynomial algorithm

- ✓ Preprocessing
- ✓ Phase 1: Reduce private neighbours of non-adjacent vertices in P_7
- ✓ Phase 2: Two types of lists in the first layer ($\{$  ,  $\}$, $\{$,  $\}$)
- Phase 3: One type of lists in the first layer ($\{$  ,  $\}$)

$(P_2 + P_5)$-free graphs

- N_2 is an independent set.
- Colour all vertices in A_2 by the colour which is not in A_1.

Thank you for your attention!