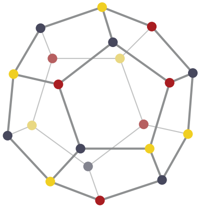
Algorithms for polynomial instances of graph coloring

Flavia Bonomo

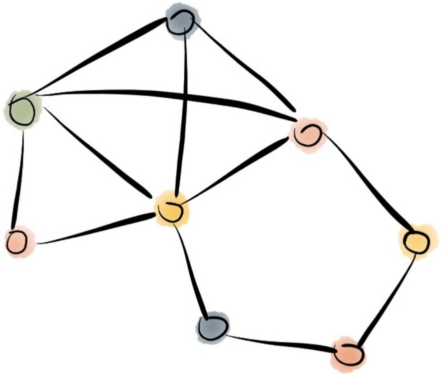
Universidad de Buenos Aires, FCEyN, Departamento de Computación / CONICET-Universidad de Buenos Aires, Instituto de Investigación en Ciencias de la Computación (ICC), Argentina

ELAVIO 2017



k-colorability

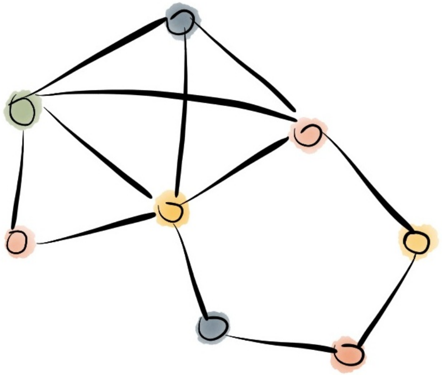
Given a graph G and a number k, decide whether G is k-colorable.



For $k \leq 2$: Just check if G is bipartite (breadth-first search).
Polynomial.

k-colorability

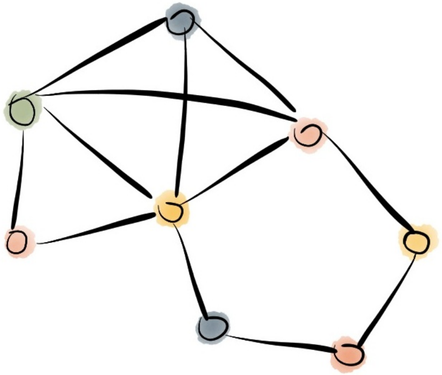
Given a graph G and a number k, decide whether G is k-colorable.



For $k \leq 2$: Just check if G is bipartite (breadth-first search).
Polynomial.

k-colorability

The k-coloring problem of a graph is NP-complete for every $k \geq 3$.



What about restriction to special graph classes?

k-colorability in special graph classes

- Polynomial for perfect graphs (Grötschel, Lovász, and Schrijver 1984).
- Polynomial (with simple algorithms) for subclasses of perfect graphs, like chordal graphs, interval graphs, cographs.
- Polynomial for proper circular-arc graphs (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- NP-complete for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- NP-complete for P_5-free graphs (Kral, Kratochvíl, Tuza, and Woeginger 2001).

k-colorability in special graph classes

- Polynomial for perfect graphs (Grötschel, Lovász, and Schrijver 1984).
- Polynomial (with simple algorithms) for subclasses of perfect graphs, like chordal graphs, interval graphs, cographs.
- Polynomial for proper circular-arc graphs (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- NP-complete for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- NP-complete for P_5-free graphs (Kral, Kratochvíl, Tuza, and Woeginger 2001).

k-colorability in special graph classes

- Polynomial for perfect graphs (Grötschel, Lovász, and Schrijver 1984).
- Polynomial (with simple algorithms) for subclasses of perfect graphs, like chordal graphs, interval graphs, cographs.
- Polynomial for proper circular-arc graphs (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- NP-complete for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- NP-complete for P_5-free graphs (Kral, Kratochvíl, Tuza, and Woeginger 2001).

k-colorability in special graph classes

- Polynomial for perfect graphs (Grötschel, Lovász, and Schrijver 1984).
- Polynomial (with simple algorithms) for subclasses of perfect graphs, like chordal graphs, interval graphs, cographs.
- Polynomial for proper circular-arc graphs (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- NP-complete for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- NP-complete for P_5-free graphs (Kral, Kratochvíl, Tuza, and Woeginger 2001).

k-colorability in special graph classes

- Polynomial for perfect graphs (Grötschel, Lovász, and Schrijver 1984).
- Polynomial (with simple algorithms) for subclasses of perfect graphs, like chordal graphs, interval graphs, cographs.
- Polynomial for proper circular-arc graphs (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- NP-complete for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- NP-complete for P_5-free graphs (Kral, Kratochvíl, Tuza, and Woeginger 2001).

k-colorability in special graph classes (fixed $k \geq 3$)

- Polynomial for perfect graphs (just check for K_{k+1}).
- k-colorability for fixed k is polynomial for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- k-colorability for fixed k is linear for proper circular-arc graphs (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).
- NP-complete for planar graphs, even for planar 4-regular graphs (Dailey 1980).
- NP-complete for triangle-free graphs (even for $k = 3$) (Maffray and Preissmann 1996)
- How about forbidding other subgraphs?

k-colorability in special graph classes (fixed $k \geq 3$)

- Polynomial for perfect graphs (just check for K_{k+1}).
- k-colorability for fixed k is polynomial for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- k-colorability for fixed k is linear for proper circular-arc graphs (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).
- NP-complete for planar graphs, even for planar 4-regular graphs (Dailey 1980).
- NP-complete for triangle-free graphs (even for $k = 3$) (Maffray and Preissmann 1996)
- How about forbidding other subgraphs?

k-colorability in special graph classes (fixed $k \geq 3$)

- Polynomial for perfect graphs (just check for K_{k+1}).
- k-colorability for fixed k is polynomial for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- k-colorability for fixed k is linear for proper circular-arc graphs (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).
- NP-complete for planar graphs, even for planar 4-regular graphs (Dailey 1980).
- NP-complete for triangle-free graphs (even for $k = 3$) (Maffray and Preissmann 1996)
- How about forbidding other subgraphs?

k-colorability in special graph classes (fixed $k \geq 3$)

- Polynomial for perfect graphs (just check for K_{k+1}).
- k-colorability for fixed k is polynomial for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- k-colorability for fixed k is linear for proper circular-arc graphs (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).
- NP-complete for planar graphs, even for planar 4-regular graphs (Dailey 1980).
- NP-complete for triangle-free graphs (even for $k = 3$) (Maffray and Preissmann 1996)
- How about forbidding other subgraphs?

k-colorability in special graph classes (fixed $k \geq 3$)

- Polynomial for perfect graphs (just check for K_{k+1}).
- k-colorability for fixed k is polynomial for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- k-colorability for fixed k is linear for proper circular-arc graphs (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).
- NP-complete for planar graphs, even for planar 4-regular graphs (Dailey 1980).
- NP-complete for triangle-free graphs (even for $k = 3$) (Maffray and Preissmann 1996)
- How about forbidding other subgraphs?

k-colorability in special graph classes (fixed $k \geq 3$)

- Polynomial for perfect graphs (just check for K_{k+1}).
- k-colorability for fixed k is polynomial for circular-arc graphs (Garey, Johnson, Miller, and Papadimitriou 1980).
- k-colorability for fixed k is linear for proper circular-arc graphs (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).
- NP-complete for planar graphs, even for planar 4-regular graphs (Dailey 1980).
- NP-complete for triangle-free graphs (even for $k = 3$) (Maffray and Preissmann 1996)
- How about forbidding other subgraphs?

k-colorability in H-free graphs

A graph is H-free if it does not contain H as an induced subgraph.

k-colorability in H-free graphs

A graph is H-free if it does not contain H as an induced subgraph.

Theorem (Král, Kratochvíl, Tuza, and Woeginger 2001, Lozin and Kamiński 2007)

If H is a graph that contains a cycle, then k-colorability is NP-complete in the class of H-free graphs, for $k \geq 3$.

k-colorability in H-free graphs

A graph is H-free if it does not contain H as an induced subgraph.

Theorem (Král, Kratochvíl, Tuza, and Woeginger 2001, Lozin and Kamiński 2007)

If H is a graph that contains a cycle, then k-colorability is NP-complete in the class of H-free graphs, for $k \geq 3$.

And:

Theorem (Holyer 1981, Leven and Galil 1983)

Let F be a forest with a vertex of degree ≥ 3. Then k-colorability is NP-complete in the class of F-free graphs, for $k \geq 3$.

k-colorability in H-free graphs

A graph is H-free if it does not contain H as an induced subgraph.

Theorem (Král, Kratochvíl, Tuza, and Woeginger 2001, Lozin and Kamiński 2007)

If H is a graph that contains a cycle, then k-colorability is NP-complete in the class of H-free graphs, for $k \geq 3$.

And:

Theorem (Holyer 1981, Leven and Galil 1983)

Let F be a forest with a vertex of degree ≥ 3. Then k-colorability is NP-complete in the class of F-free graphs, for $k \geq 3$.

This leads to the study of P_t-free graphs, where P_t is the path on t vertices.

k-colorability in P_t-free graphs

Complexity of k-colorability in the class of P_t-free graphs:

k\t	4	5	6	7	8	...
3	$O(m)$ [1]	$O(n^\alpha)$ [4]	$O(mn^\alpha)$ [5]	P [6]	?	...
4	$O(m)$ [1]	P [2]	?	NPC [3]	NPC	...
5	$O(m)$ [1]	P [2]	NPC [3]	NPC	NPC	...
6	$O(m)$ [1]	P [2]	NPC	NPC	NPC	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

[1] Chvátal 1984, Corneil, Perl, and Stewart 1984.
[2] Hoàng, Kamiński, Lozin, Sawada, and Shu 2010.
[3] Huang 2013.
[4] Mellin 2002.
[5] Randerath and Schiermeyer 2004.
[6] B., Chudnovsky, Maceli, Schaudt, Stein, and Zhong 2015.

k-colorability in P_t-free graphs

Complexity of k-colorability in the class of P_t-free graphs:

k\t	4	5	6	7	8	...
3	$O(m)$ [1]	$O(n^\alpha)$ [4]	$O(mn^\alpha)$ [5]	P [6]	?	...
4	$O(m)$ [1]	P [2]	?	NPC [3]	NPC	...
5	$O(m)$ [1]	P [2]	NPC [3]	NPC	NPC	...
6	$O(m)$ [1]	P [2]	NPC	NPC	NPC	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

[1] Chvátal 1984, Corneil, Perl, and Stewart 1984.

[2] Hoàng, Kamiński, Lozin, Sawada, and Shu 2010.

[3] Huang 2013.

[4] Mellin 2002.

[5] Randerath and Schiermeyer 2004.

[6] B., Chudnovsky, Maceli, Schaudt, Stein, and Zhong 2015.

[recommended lectures] - Golovach, Johnson, Paulusma, and Song, A Survey on the Computational Complexity of Colouring Graphs with Forbidden Subgraphs
- Hell and Huang, Complexity of coloring graphs without paths and cycles

Let's start by chordal graphs

- A graph is chordal if it contains no induced C_n, $n \geq 4$, that is, if every cycle of length at least 4 has a chord.
- Also called triangulated or rigid circuit.

- [recommended lecture] Blair and Peyton, An introduction to chordal graphs and clique trees

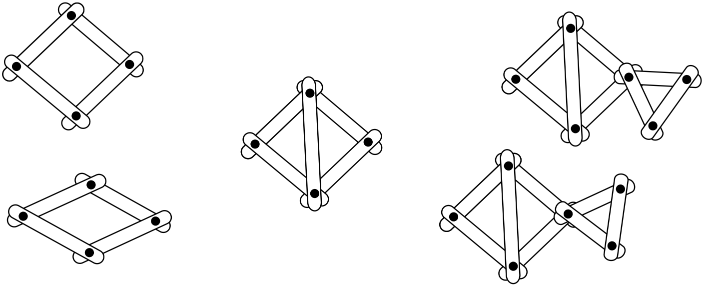
Let's start by chordal graphs

- A graph is chordal if it contains no induced C_n, $n \geq 4$, that is, if every cycle of length at least 4 has a chord.
- Also called triangulated or rigid circuit.

- [recommended lecture] Blair and Peyton, An introduction to chordal graphs and clique trees

Let's start by chordal graphs

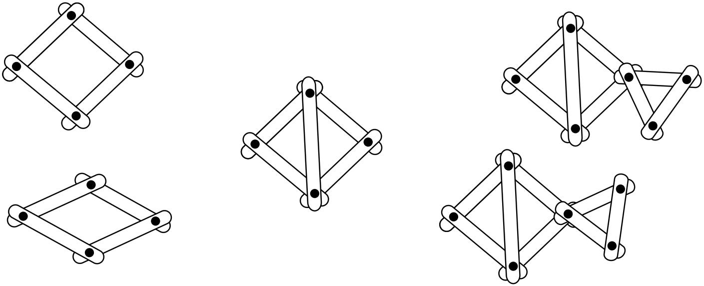
- A graph is chordal if it contains no induced C_n, $n \geq 4$, that is, if every cycle of length at least 4 has a chord.
- Also called triangulated or rigid circuit.



- [recommended lecture] Blair and Peyton, An introduction to chordal graphs and clique trees

Let's start by chordal graphs

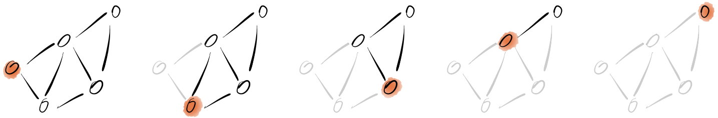
- A graph is chordal if it contains no induced C_n, $n \geq 4$, that is, if every cycle of length at least 4 has a chord.
- Also called triangulated or rigid circuit.



- [recommended lecture] Blair and Peyton, An introduction to chordal graphs and clique trees

Perfect elimination ordering

- A vertex v is simplicial if $N[v]$ induces a complete subgraph on G.
- An ordering $v_1, v_2, \ldots, v_n$ of the vertices of a graph G is a perfect elimination ordering if, for every $2 \leq i \leq n-2$ v_i is simplicial in $G[v_i, v_{i+1}, \ldots, v_n]$.



Perfect elimination ordering

Theorem (Dirac, 1961)

Every chordal graph has a simplicial vertex. If it is not complete, then it has two non-adjacent simplicial vertices.

Theorem (Fulkerson and Gross, 1965)

A graph is chordal if and only if it has a PEO.

Perfect elimination ordering

Theorem (Dirac, 1961)

Every chordal graph has a simplicial vertex. If it is not complete, then it has two non-adjacent simplicial vertices.

Theorem (Fulkerson and Gross, 1965)

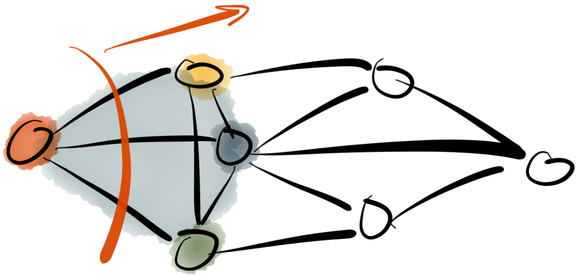
A graph is chordal if and only if it has a PEO.

Algorithmic problems in chordal graphs

How can we solve maximum clique and minimum coloring on chordal graphs?

Algorithmic problems in chordal graphs

How can we solve maximum clique and minimum coloring on chordal graphs?



Algorithmic problems in chordal graphs

Let v be a simplicial vertex:

either v belongs to the maximum clique or not... but v belongs to just one maximal clique! so...

$$\omega(G) = \max\{|N[v]|, \omega(G - v)\}$$

Note: there is a linear number of maximal cliques!

We can extend an optimum coloring of $G - v$ to G without adding colors unless $\chi(G - v) < d(v)$. But in that case we add one new color and, as $N[v]$ is a clique, it is optimum. So...

$$\chi(G) = \max\{|N[v]|, \chi(G - v)\}$$

Chordal graphs are perfect:
For every chordal graph G, $\omega(G) = \chi(G)$, and it holds also for the induced subgraphs because the class is hereditary.

Algorithmic problems in chordal graphs

Let v be a simplicial vertex:

either v belongs to the maximum clique or not... but v belongs to just one maximal clique! so...

$$\omega(G) = \max\{|N[v]|, \omega(G - v)\}$$

Note: there is a linear number of maximal cliques!

We can extend an optimum coloring of $G - v$ to G without adding colors unless $\chi(G - v) < d(v)$. But in that case we add one new color and, as $N[v]$ is a clique, it is optimum. So...

$$\chi(G) = \max\{|N[v]|, \chi(G - v)\}$$

Chordal graphs are perfect:
For every chordal graph G, $\omega(G) = \chi(G)$, and it holds also for the induced subgraphs because the class is hereditary.

Algorithmic problems in chordal graphs

Let v be a simplicial vertex:

either v belongs to the maximum clique or not... but v belongs to just one maximal clique! so...

$$\omega(G) = \max\{|N[v]|, \omega(G - v)\}$$

Note: there is a linear number of maximal cliques!

We can extend an optimum coloring of $G - v$ to G without adding colors unless $\chi(G - v) < d(v)$. But in that case we add one new color and, as $N[v]$ is a clique, it is optimum. So...

$$\chi(G) = \max\{|N[v]|, \chi(G - v)\}$$

Chordal graphs are perfect:
For every chordal graph G, $\omega(G) = \chi(G)$, and it holds also for the induced subgraphs because the class is hereditary.

Algorithmic problems in chordal graphs

Let v be a simplicial vertex:

either v belongs to the maximum clique or not... but v belongs to just one maximal clique! so...

$$\omega(G) = \max\{|N[v]|, \omega(G - v)\}$$

Note: there is a linear number of maximal cliques!

We can extend an optimum coloring of $G - v$ to G without adding colors unless $\chi(G - v) < d(v)$. But in that case we add one new color and, as $N[v]$ is a clique, it is optimum. So...

$$\chi(G) = \max\{|N[v]|, \chi(G - v)\}$$

Chordal graphs are perfect:
For every chordal graph G, $\omega(G) = \chi(G)$, and it holds also for the induced subgraphs because the class is hereditary.

Algorithmic problems in chordal graphs

Let v be a simplicial vertex:

either v belongs to the maximum clique or not... but v belongs to just one maximal clique! so...

$$\omega(G) = \max\{|N[v]|, \omega(G - v)\}$$

Note: there is a linear number of maximal cliques!

We can extend an optimum coloring of $G - v$ to G without adding colors unless $\chi(G - v) < d(v)$. But in that case we add one new color and, as $N[v]$ is a clique, it is optimum. So...

$$\chi(G) = \max\{|N[v]|, \chi(G - v)\}$$

Chordal graphs are perfect:
For every chordal graph G, $\omega(G) = \chi(G)$, and it holds also for the induced subgraphs because the class is hereditary.

Algorithmic problems in chordal graphs

Let v be a simplicial vertex:

either v belongs to the maximum clique or not... but v belongs to just one maximal clique! so...

$$\omega(G) = \max\{|N[v]|, \omega(G - v)\}$$

Note: there is a linear number of maximal cliques!

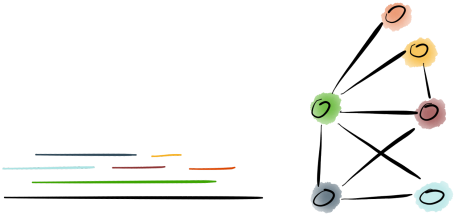
We can extend an optimum coloring of $G - v$ to G without adding colors unless $\chi(G - v) < d(v)$. But in that case we add one new color and, as $N[v]$ is a clique, it is optimum. So...

$$\chi(G) = \max\{|N[v]|, \chi(G - v)\}$$

Chordal graphs are perfect:
For every chordal graph G, $\omega(G) = \chi(G)$, and it holds also for the induced subgraphs because the class is hereditary.

Interval graphs

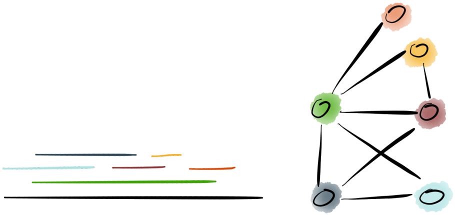
- An interval graph is the intersection graph of intervals in a line.



- They are a subclass of chordal graphs.
- Which are the perfect elimination orderings?
- The right-end ordering of the vertices of an interval graph is a perfect elimination ordering.

Interval graphs

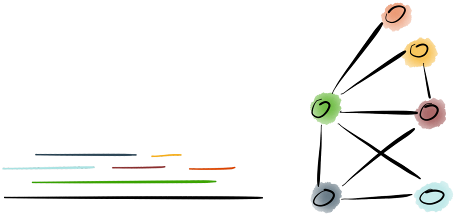
- An interval graph is the intersection graph of intervals in a line.



- They are a subclass of chordal graphs.
- Which are the perfect elimination orderings?
- The right-end ordering of the vertices of an interval graph is a perfect elimination ordering.

Interval graphs

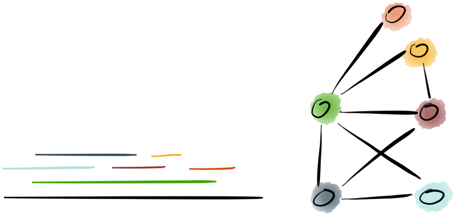
- An interval graph is the intersection graph of intervals in a line.



- They are a subclass of chordal graphs.
- Which are the perfect elimination orderings?
- The right-end ordering of the vertices of an interval graph is a perfect elimination ordering.

Interval graphs

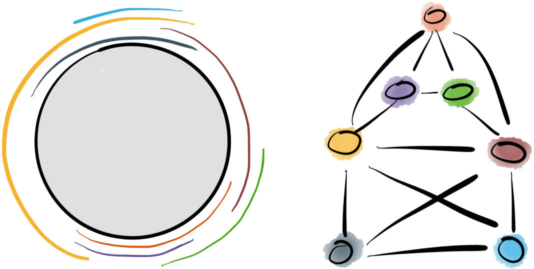
- An interval graph is the intersection graph of intervals in a line.



- They are a subclass of chordal graphs.
- Which are the perfect elimination orderings?
- The right-end ordering of the vertices of an interval graph is a perfect elimination ordering.

Circular-arc graphs

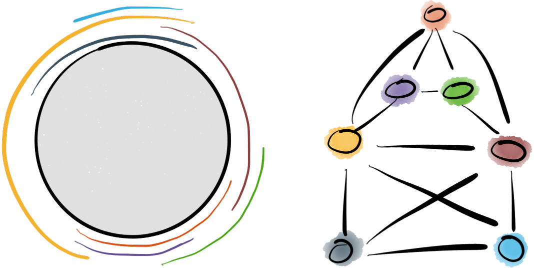
- A circular-arc graph is the intersection graph of arcs on a circle.



- Model for scheduling problems where there in no "night stop".
- *k*-coloring (fixed *k*): polynomial (Garey, Johnson, Miller, and Papadimitriou 1980).
- Coloring: NP-complete (Garey, Johnson, Miller, and Papadimitriou 1980).

Circular-arc graphs

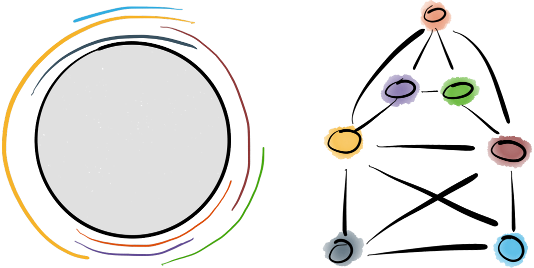
- A circular-arc graph is the intersection graph of arcs on a circle.



- Model for scheduling problems where there in no “night stop”.
- k-coloring (fixed k): polynomial (Garey, Johnson, Miller, and Papadimitriou 1980).
- Coloring: NP-complete (Garey, Johnson, Miller, and Papadimitriou 1980).

Circular-arc graphs

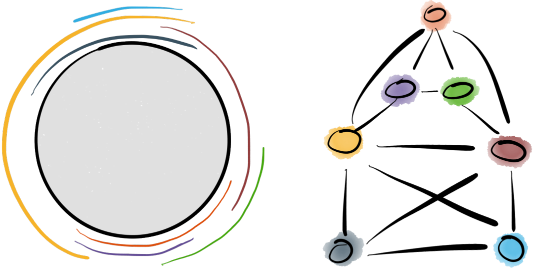
- A circular-arc graph is the intersection graph of arcs on a circle.



- Model for scheduling problems where there in no "night stop".
- k-coloring (fixed k): polynomial (Garey, Johnson, Miller, and Papadimitriou 1980).
- Coloring: NP-complete (Garey, Johnson, Miller, and Papadimitriou 1980).

Circular-arc graphs

- A circular-arc graph is the intersection graph of arcs on a circle.



- Model for scheduling problems where there in no "night stop".
- k-coloring (fixed k): polynomial (Garey, Johnson, Miller, and Papadimitriou 1980).
- Coloring: NP-complete (Garey, Johnson, Miller, and Papadimitriou 1980).

Proper circular-arc graphs

- A circular-arc graph is proper if it admits a model with no arc contained in another.
- Coloring: polynomial (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- k-coloring (fixed k): linear (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).

The algorithm by Orlin, Bonuccelli, and Bovel proves first some theoretical properties on the optimal solutions for coloring proper circular-arc graphs, and then models the problem as an integer programming one, and observes that the model corresponds to a shortest path problem in an auxiliary graph.

Proper circular-arc graphs

- A circular-arc graph is proper if it admits a model with no arc contained in another.
- Coloring: polynomial (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- k-coloring (fixed k): linear (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).

The algorithm by Orlin, Bonuccelli, and Bovel proves first some theoretical properties on the optimal solutions for coloring proper circular-arc graphs, and then models the problem as an integer programming one, and observes that the model corresponds to a shortest path problem in an auxiliary graph.

Proper circular-arc graphs

- A circular-arc graph is proper if it admits a model with no arc contained in another.
- Coloring: polynomial (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- k-coloring (fixed k): linear (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).

The algorithm by Orlin, Bonuccelli, and Bovel proves first some theoretical properties on the optimal solutions for coloring proper circular-arc graphs, and then models the problem as an integer programming one, and observes that the model corresponds to a shortest path problem in an auxiliary graph.

Proper circular-arc graphs

- A circular-arc graph is proper if it admits a model with no arc contained in another.
- Coloring: polynomial (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- k-coloring (fixed k): linear (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).

The algorithm by Orlin, Bonuccelli, and Bovel proves first some theoretical properties on the optimal solutions for coloring proper circular-arc graphs, and then models the problem as an integer programming one, and observes that the model corresponds to a shortest path problem in an auxiliary graph.

Proper circular-arc graphs

- A circular-arc graph is proper if it admits a model with no arc contained in another.
- Coloring: polynomial (Orlin, Bonuccelli, and Bovel 1981, Shih and Hsu 1989).
- k-coloring (fixed k): linear (Teng and Tucker 1985, Bhattacharya, Hell, and Jing 1996).

The algorithm by Orlin, Bonuccelli, and Bovel proves first some theoretical properties on the optimal solutions for coloring proper circular-arc graphs, and then models the problem as an integer programming one, and observes that the model corresponds to a shortest path problem in an auxiliary graph.

Cographs

- A cograph is a graph with no induced P_4 (chordless path on four vertices).

- Property: If G is a non-trivial cograph, then either G or $\bar{G}$ is non-connected.

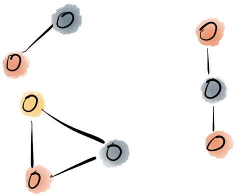
Cographs

- A cograph is a graph with no induced P_4 (chordless path on four vertices).

- Property: If G is a non-trivial cograph, then either G or $\bar{G}$ is non-connected.

Cographs

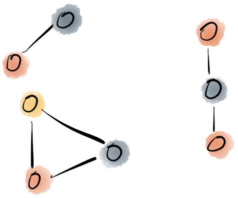
- In the first case, G is the union of two smaller cographs ($G = G_1 \cup G_2$).



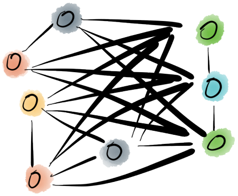
- In the second case, G is the join of two smaller cographs ($G = G_1 \vee G_2$).

Cographs

- In the first case, G is the union of two smaller cographs ($G = G_1 \cup G_2$).

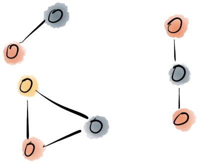


- In the second case, G is the join of two smaller cographs ($G = G_1 \vee G_2$).

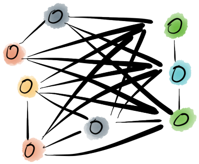


Algorithmic problems in cographs

- Let G_0 be the trivial graph. For coloring,
  $\chi(G_0) = 1$,
  $\chi(G_1 \cup G_2) = \max\{\chi(G_1), \chi(G_2)\}$, and
  $\chi(G_1 \vee G_2) = \chi(G_1) + \chi(G_2)$.



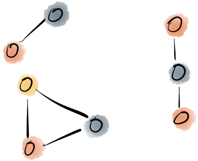
- To compute a maximum clique, $\omega(G_0) = 1$,
  $\omega(G_1 \cup G_2) = \max\{\omega(G_1), \omega(G_2)\}$, and
  $\omega(G_1 \vee G_2) = \omega(G_1) + \omega(G_2)$.



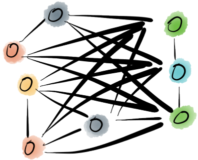
In particular, it can be seen that cographs are perfect.

Algorithmic problems in cographs

- Let G_0 be the trivial graph. For coloring,
  $\chi(G_0) = 1$,
  $\chi(G_1 \cup G_2) = \max\{\chi(G_1), \chi(G_2)\}$, and
  $\chi(G_1 \vee G_2) = \chi(G_1) + \chi(G_2)$.



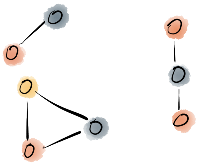
- To compute a maximum clique, $\omega(G_0) = 1$,
  $\omega(G_1 \cup G_2) = \max\{\omega(G_1), \omega(G_2)\}$, and
  $\omega(G_1 \vee G_2) = \omega(G_1) + \omega(G_2)$.



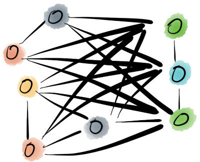
In particular, it can be seen that cographs are perfect.

Algorithmic problems in cographs

- Let G_0 be the trivial graph. For coloring,
  $\chi(G_0) = 1$,
  $\chi(G_1 \cup G_2) = \max\{\chi(G_1), \chi(G_2)\}$, and
  $\chi(G_1 \vee G_2) = \chi(G_1) + \chi(G_2)$.



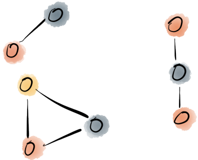
- To compute a maximum clique, $\omega(G_0) = 1$,
  $\omega(G_1 \cup G_2) = \max\{\omega(G_1), \omega(G_2)\}$, and
  $\omega(G_1 \vee G_2) = \omega(G_1) + \omega(G_2)$.



In particular, it can be seen that cographs are perfect.

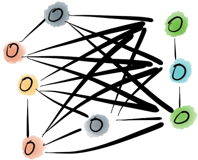
Algorithmic problems in cographs

- Let G_0 be the trivial graph. For coloring,
  $\chi(G_0) = 1$,
  $\chi(G_1 \cup G_2) = \max\{\chi(G_1), \chi(G_2)\}$, and
  $\chi(G_1 \vee G_2) = \chi(G_1) + \chi(G_2)$.



- To compute a maximum clique, $\omega(G_0) = 1$,
  $\omega(G_1 \cup G_2) = \max\{\omega(G_1), \omega(G_2)\}$, and
  $\omega(G_1 \vee G_2) = \omega(G_1) + \omega(G_2)$.

  In particular, it can be seen that cographs are perfect.



Another algorithm for coloring....

The greedy algorithm based on picking maximum stable sets works for cographs!! (even picking just maximal ones!)

Lemma

Let G be a cograph. Then every maximal stable set of G intersects every maximal clique of G.

Proof. By induction, using the decomposition. For the trivial graph is true. If $G = G_1 \cup G_2$, every maximal stable set is composed by a maximal stable set of G_1 and a maximal stable set of G_2, and by inductive hypothesis the part in G_i intersects all the maximal cliques of G_i, for $i = 1, 2$, and these are exactly the maximal cliques of G. If $G = G_1 \vee G_2$, every maximal stable set of G is either a maximal stable set of G_1 or a maximal stable set of G_2, but every maximal clique is composed by a maximal clique of G_1 and a maximal clique of G_2, so by inductive hypothesis the stable set intersects the clique, either in its G_1-part or in its G_2-part. □

Using the lemma, the result follows by induction on $\omega(G)$.

Another algorithm for coloring....

The greedy algorithm based on picking maximum stable sets works for cographs!! (even picking just maximal ones!)

Lemma

Let G be a cograph. Then every maximal stable set of G intersects every maximal clique of G.

Proof. By induction, using the decomposition. For the trivial graph is true. If $G = G_1 \cup G_2$, every maximal stable set is composed by a maximal stable set of G_1 and a maximal stable set of G_2, and by inductive hypothesis the part in G_i intersects all the maximal cliques of G_i, for $i = 1, 2$, and these are exactly the maximal cliques of G. If $G = G_1 \vee G_2$, every maximal stable set of G is either a maximal stable set of G_1 or a maximal stable set of G_2, but every maximal clique is composed by a maximal clique of G_1 and a maximal clique of G_2, so by inductive hypothesis the stable set intersects the clique, either in its G_1-part or in its G_2-part. $\square$

Using the lemma, the result follows by induction on $\omega(G)$.

Another algorithm for coloring....

The greedy algorithm based on picking maximum stable sets works for cographs!! (even picking just maximal ones!)

Lemma

Let G be a cograph. Then every maximal stable set of G intersects every maximal clique of G.

Proof. By induction, using the decomposition. For the trivial graph is true. If $G = G_1 \cup G_2$, every maximal stable set is composed by a maximal stable set of G_1 and a maximal stable set of G_2, and by inductive hypothesis the part in G_i intersects all the maximal cliques of G_i, for $i = 1, 2$, and these are exactly the maximal cliques of G. If $G = G_1 \vee G_2$, every maximal stable set of G is either a maximal stable set of G_1 or a maximal stable set of G_2, but every maximal clique is composed by a maximal clique of G_1 and a maximal clique of G_2, so by inductive hypothesis the stable set intersects the clique, either in its G_1-part or in its G_2-part. □

Using the lemma, the result follows by induction on $\omega(G)$.

Another algorithm for coloring....

The greedy algorithm based on picking maximum stable sets works for cographs!! (even picking just maximal ones!)

Lemma

Let G be a cograph. Then every maximal stable set of G intersects every maximal clique of G.

Proof. By induction, using the decomposition. For the trivial graph is true. If $G = G_1 \cup G_2$, every maximal stable set is composed by a maximal stable set of G_1 and a maximal stable set of G_2, and by inductive hypothesis the part in G_i intersects all the maximal cliques of G_i, for $i = 1, 2$, and these are exactly the maximal cliques of G. If $G = G_1 \vee G_2$, every maximal stable set of G is either a maximal stable set of G_1 or a maximal stable set of G_2, but every maximal clique is composed by a maximal clique of G_1 and a maximal clique of G_2, so by inductive hypothesis the stable set intersects the clique, either in its G_1-part or in its G_2-part. □

Using the lemma, the result follows by induction on $\omega(G)$.

Another algorithm for coloring....

The greedy algorithm based on picking maximum stable sets works for cographs!! (even picking just maximal ones!)

Lemma

Let G be a cograph. Then every maximal stable set of G intersects every maximal clique of G.

Proof. By induction, using the decomposition. For the trivial graph is true. If $G = G_1 \cup G_2$, every maximal stable set is composed by a maximal stable set of G_1 and a maximal stable set of G_2, and by inductive hypothesis the part in G_i intersects all the maximal cliques of G_i, for $i = 1, 2$, and these are exactly the maximal cliques of G. If $G = G_1 \vee G_2$, every maximal stable set of G is either a maximal stable set of G_1 or a maximal stable set of G_2, but every maximal clique is composed by a maximal clique of G_1 and a maximal clique of G_2, so by inductive hypothesis the stable set intersects the clique, either in its G_1-part or in its G_2-part. $\square$

Using the lemma, the result follows by induction on $\omega(G)$.

Yet one more....

The greedy algorithm (give to v the first color not used by a neighbor) also works for cographs for any vertex order!!

Just note that the set of vertices receiving color i is a maximal stable set in the subgraph of G induced by the vertices receiving color at least i.

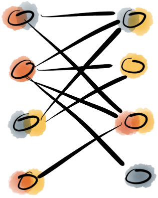
Yet one more....

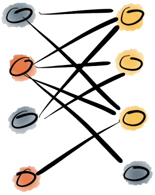
The greedy algorithm (give to v the first color not used by a neighbor) also works for cographs for any vertex order!!

Just note that the set of vertices receiving color i is a maximal stable set in the subgraph of G induced by the vertices receiving color at least i.

The list-coloring problem

In order to take into account particular constraints arising in practical settings, more elaborate models of vertex coloring have been defined in the literature. One of such generalized models is the list-coloring problem, which considers a prespecified set of available colors for each vertex.





The list-coloring problem

- The list-coloring problem is NP-complete for perfect graphs, and is also NP-complete for many subclasses of perfect graphs, including cographs, proper interval graphs, and bipartite graphs.
- Trees and complete graphs are two classes of graphs where the list-coloring problem can be solved in polynomial time. In the first case it can be solved using dynamic programming techniques (Jansen and Scheffler, 1997). In the second case, the problem can be reduced to the maximum matching problem in bipartite graphs.

Back to 3-coloring P_7-free graphs, a bit more general: list 3-coloring P_7-free graphs

We actually solve the list 3-colorability problem, where every vertex is equipped with a subset of $\{1, 2, 3\}$ of admissible colors.

It is not always the case that an algorithm for k-coloring can be generalized to list k-coloring: in the class of $\{P_6, C_5\}$-free graphs for example, 4-coloring can be solved in polynomial time (Chudnovsky, Maceli, Stacho and Zhong, 2014), while the list 4-coloring problem is NP-complete (Huang, Johnson and Paulusma, 2014).

List 3-coloring P_7-free graphs

Theorem (BCMSSZ, 2015)

Given a P_7-free graph G, the list 3-coloring problem can be decided, and a coloring can be found, in $O(n^{21}(n+m))$ time.

The algorithm is based on structural analysis, controlled enumeration, and reduction to 2-SAT, that can be solved in $O(m+n)$ time (Vizing 1976, Erdős, Rubin and Taylor, 1979, Aspvall, Plass and Tarjan, 1979).

List 3-coloring P_7-free graphs

Theorem (BCMSSZ, 2015)

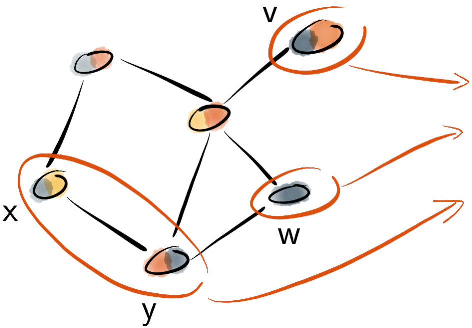
Given a P_7-free graph G, the list 3-coloring problem can be decided, and a coloring can be found, in $O(n^{21}(n+m))$ time.

The algorithm is based on structural analysis, controlled enumeration, and reduction to 2-SAT, that can be solved in $O(m+n)$ time (Vizing 1976, Erdős, Rubin and Taylor, 1979, Aspvall, Plass and Tarjan, 1979).

From list 3-coloring to 2-list-coloring

- k-list-coloring: all the lists are of size at most k
- list k-coloring: the union of the lists is contained in $\{1, \ldots, k\}$ (more restrictive)

We have a list 3-coloring instance and we want to reduce it to a (polynomial) family of 2-list-coloring instances, because 2-list-coloring reduces to 2-SAT.



$(v_r \vee v_b)$

w_b

$(\neg x_b \vee \neg y_b)$

List 3-coloring P_7-free graphs

- We first reduce the problem to a polynomial number of instances of a variation of the 2-list-coloring problem, where we have a family of sets of vertices and we ask each set to be monochromatic.
- We can reduce that problem to 2-list-coloring by contracting each set into a single vertex whose list is the intersection of the lists (we don't need to keep any property of the graph to solve 2-list-coloring).

Some considerations

- We will disregard the original lists L^* until the end, and then on each 2-list-coloring instance we will intersect the current list of each vertex with L^*.
- We will also, during the process, update the lists of neighbors of a vertex using BFS just from some vertices and avoiding some special sets.
- In that way, during the process, the number of colors on the list of a vertex tells us something about its neighbors in the graph.

Some considerations

- We will disregard the original lists L^* until the end, and then on each 2-list-coloring instance we will intersect the current list of each vertex with L^*.
- We will also, during the process, update the lists of neighbors of a vertex using BFS just from some vertices and avoiding some special sets.
- In that way, during the process, the number of colors on the list of a vertex tells us something about its neighbors in the graph.

Some considerations

- We will disregard the original lists L^* until the end, and then on each 2-list-coloring instance we will intersect the current list of each vertex with L^*.
- We will also, during the process, update the lists of neighbors of a vertex using BFS just from some vertices and avoiding some special sets.
- In that way, during the process, the number of colors on the list of a vertex tells us something about its neighbors in the graph.

A useful tool

A dominating set is a set of vertices S such that $S \cup N(S) = V(G)$. We will use the following theorem.

Theorem (Camby and Schaudt, 2014)

For all $t \geq 3$, any connected P_t-free graph has a connected dominating set whose induced subgraph is either P_{t-2}-free, or isomorphic to P_{t-2}.

Corollary

Every connected P_7-free graph has either a connected 2-dominating set of size at most 3 or a complete subgraph of 4 vertices. The set or the subgraph can be found in $O(n^3 m)$ time, given an n-vertex graph.

A useful tool

A dominating set is a set of vertices S such that $S \cup N(S) = V(G)$. We will use the following theorem.

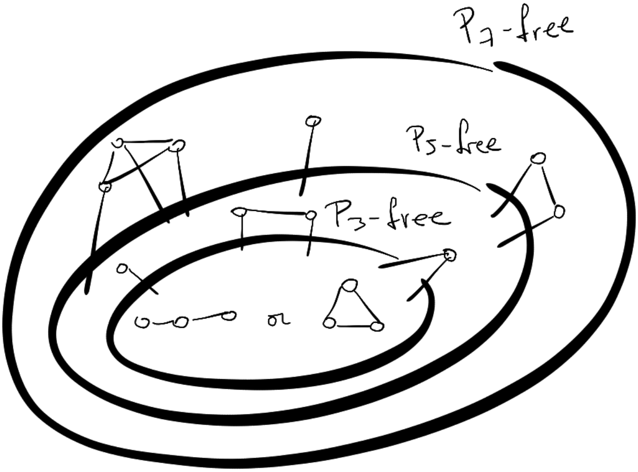
Theorem (Camby and Schaudt, 2014)

For all $t \geq 3$, any connected P_t-free graph has a connected dominating set whose induced subgraph is either P_{t-2}-free, or isomorphic to P_{t-2}.

Corollary

Every connected P_7-free graph has either a connected 2-dominating set of size at most 3 or a complete subgraph of 4 vertices. The set or the subgraph can be found in $O(n^3 m)$ time, given an n-vertex graph.

Nested dominating sets



Main idea

The idea is to start with those 3 vertex as a seed, and for every possible coloring of them, branch into instances such that each instance has a strictly greater seed, the vertices of the seed have a fixed color, and iterate a bounded number of times until obtaining a polynomial number of instances such that, for each of them, the vertices outside the seed neighbourhood having lists of size 3 are a stable set.

We will do that in such a way that there is a coloring for the original instance if and only there is a coloring for at least one of the new (refined) instances.

Main idea

The idea is to start with those 3 vertex as a seed, and for every possible coloring of them, branch into instances such that each instance has a strictly greater seed, the vertices of the seed have a fixed color, and iterate a bounded number of times until obtaining a polynomial number of instances such that, for each of them, the vertices outside the seed neighbourhood having lists of size 3 are a stable set.

We will do that in such a way that there is a coloring for the original instance if and only there is a coloring for at least one of the new (refined) instances.

Growing the seed

- For each color i, we compute the set of paths $x - y - z$ with with $x \in N(S)$, $|L(y)| = 3$ and $z \notin S \cup N(S)$, and such that $i \notin L(x)$.
- We will order the paths non-decreasingly by the number of vertices w (if any) such that $w - x - y - z$ is an induced path.
- We can compute and sort the paths in $O(n^4)$ time, and this order of the paths induces an order on the set of vertices y.
- We then enumerate some partial colorings of those paths and update the lists and the seed in order to create the refined instances.
- We iterate this process twice, and prove that after that no such path exists.

Growing the seed

- For each color i, we compute the set of paths $x - y - z$ with with $x \in N(S)$, $|L(y)| = 3$ and $z \notin S \cup N(S)$, and such that $i \notin L(x)$.
- We will order the paths non-decreasingly by the number of vertices w (if any) such that $w - x - y - z$ is an induced path.
- We can compute and sort the paths in $O(n^4)$ time, and this order of the paths induces an order on the set of vertices y.
- We then enumerate some partial colorings of those paths and update the lists and the seed in order to create the refined instances.
- We iterate this process twice, and prove that after that no such path exists.

Growing the seed

- For each color i, we compute the set of paths $x - y - z$ with with $x \in N(S)$, $|L(y)| = 3$ and $z \notin S \cup N(S)$, and such that $i \notin L(x)$.
- We will order the paths non-decreasingly by the number of vertices w (if any) such that $w - x - y - z$ is an induced path.
- We can compute and sort the paths in $O(n^4)$ time, and this order of the paths induces an order on the set of vertices y.
- We then enumerate some partial colorings of those paths and update the lists and the seed in order to create the refined instances.
- We iterate this process twice, and prove that after that no such path exists.

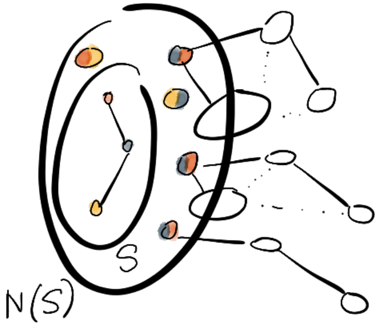
Growing the seed

- For each color i, we compute the set of paths $x - y - z$ with with $x \in N(S)$, $|L(y)| = 3$ and $z \notin S \cup N(S)$, and such that $i \notin L(x)$.
- We will order the paths non-decreasingly by the number of vertices w (if any) such that $w - x - y - z$ is an induced path.
- We can compute and sort the paths in $O(n^4)$ time, and this order of the paths induces an order on the set of vertices y.
- We then enumerate some partial colorings of those paths and update the lists and the seed in order to create the refined instances.
- We iterate this process twice, and prove that after that no such path exists.

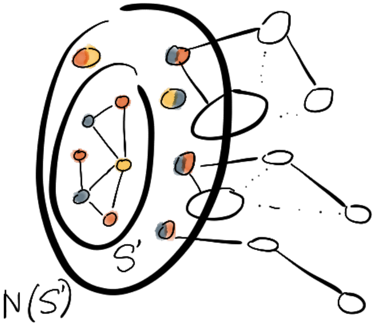
Growing the seed

- For each color i, we compute the set of paths $x - y - z$ with with $x \in N(S)$, $|L(y)| = 3$ and $z \notin S \cup N(S)$, and such that $i \notin L(x)$.
- We will order the paths non-decreasingly by the number of vertices w (if any) such that $w - x - y - z$ is an induced path.
- We can compute and sort the paths in $O(n^4)$ time, and this order of the paths induces an order on the set of vertices y.
- We then enumerate some partial colorings of those paths and update the lists and the seed in order to create the refined instances.
- We iterate this process twice, and prove that after that no such path exists.

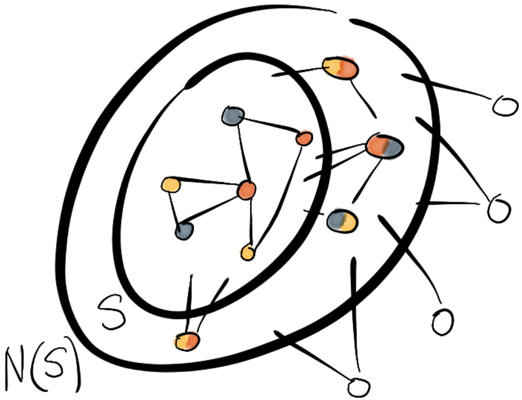
Growing the seed



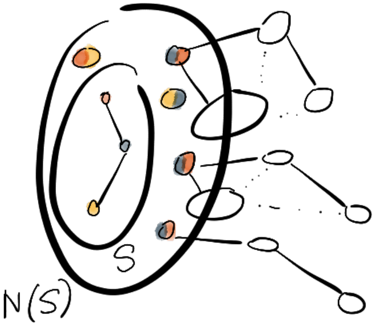
Growing the seed



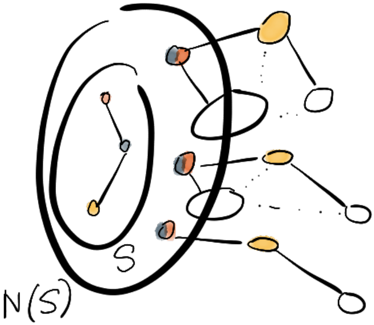
Growing the seed



Growing the seed

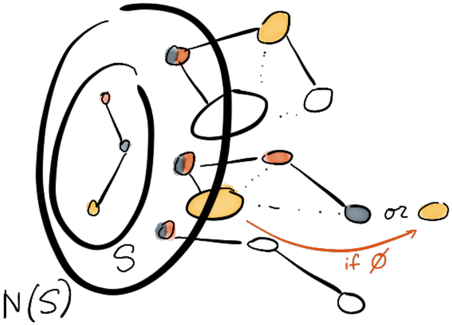


Growing the seed



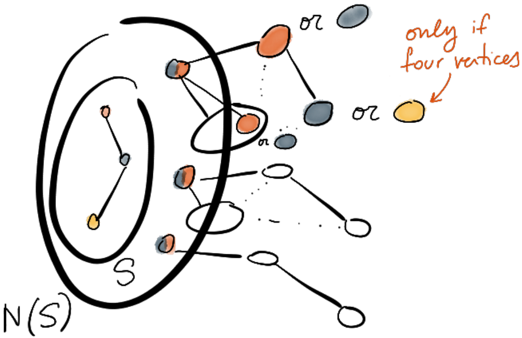
1 instance

Growing the seed



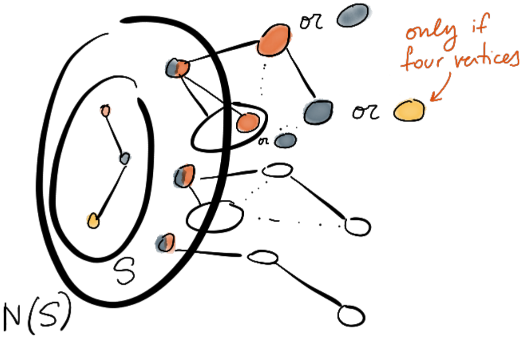
$O(n)$ instances

Growing the seed



$O(n^2)$ instances

Growing the seed



Combining each of the possibilities for each of the 3 types of vertices in $N(S)$ gives $O(n^2) \times O(n^2) \times O(n^2) = O(n^6)$ instances.

The knockout

We prove that two steps of the procedure are enough, so we have in total $O(n^6) \times O(n^6) = O(n^{12})$ instances, that we intersect with the original lists L^*. After applying some preprocessing rules, we are under the assumptions of the following lemma.

Lemma

Let G be a connected P_7-free graph with a list $L(v) \subseteq \{1,2,3\}$ for each vertex v. Let S be a seed of G such that the set X of vertices having lists of size 3 is stable and anticomplete to $V(G) \setminus (S \cup N(S) \cup X)$. Then we can decide whether G has a coloring for L, and find it, in $O(n^9(n+m))$ time.

Since we have $O(n^{12})$ instances to consider, the total running time amounts to $O(n^{21}(n+m))$.

The proof relies on technical lemmas on types of colorings.

The knockout

We prove that two steps of the procedure are enough, so we have in total $O(n^6) \times O(n^6) = O(n^{12})$ instances, that we intersect with the original lists L^*. After applying some preprocessing rules, we are under the assumptions of the following lemma.

Lemma

Let G be a connected P_7-free graph with a list $L(v) \subseteq \{1, 2, 3\}$ for each vertex v. Let S be a seed of G such that the set X of vertices having lists of size 3 is stable and anticomplete to $V(G) \setminus (S \cup N(S) \cup X)$. Then we can decide whether G has a coloring for L, and find it, in $O(n^9(n + m))$ time.

Since we have $O(n^{12})$ instances to consider, the total running time amounts to $O(n^{21}(n + m))$.

The proof relies on technical lemmas on types of colorings.

The knockout

We prove that two steps of the procedure are enough, so we have in total $O(n^6) \times O(n^6) = O(n^{12})$ instances, that we intersect with the original lists L^*. After applying some preprocessing rules, we are under the assumptions of the following lemma.

Lemma

Let G be a connected P_7-free graph with a list $L(v) \subseteq \{1, 2, 3\}$ for each vertex v. Let S be a seed of G such that the set X of vertices having lists of size 3 is stable and anticomplete to $V(G) \setminus (S \cup N(S) \cup X)$. Then we can decide whether G has a coloring for L, and find it, in $O(n^9(n+m))$ time.

Since we have $O(n^{12})$ instances to consider, the total running time amounts to $O(n^{21}(n+m))$.

The proof relies on technical lemmas on types of colorings.

The knockout

We prove that two steps of the procedure are enough, so we have in total $O(n^6) \times O(n^6) = O(n^{12})$ instances, that we intersect with the original lists L^*. After applying some preprocessing rules, we are under the assumptions of the following lemma.

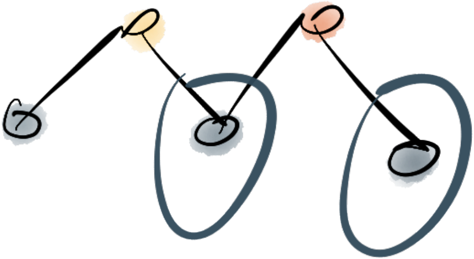
Lemma

Let G be a connected P_7-free graph with a list $L(v) \subseteq \{1, 2, 3\}$ for each vertex v. Let S be a seed of G such that the set X of vertices having lists of size 3 is stable and anticomplete to $V(G) \setminus (S \cup N(S) \cup X)$. Then we can decide whether G has a coloring for L, and find it, in $O(n^9(n+m))$ time.

Since we have $O(n^{12})$ instances to consider, the total running time amounts to $O(n^{21}(n+m))$.

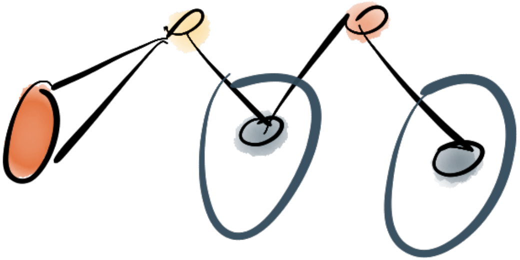
The proof relies on technical lemmas on types of colorings.

Sketches of the types of colorings



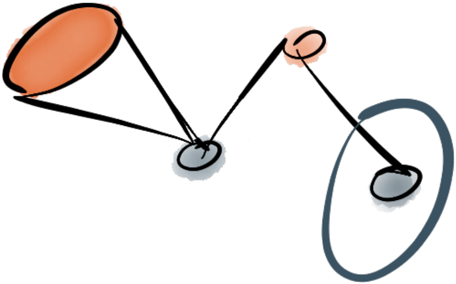
Type A coloring w.r.t blue: $O(n^3)$ instances

Sketches of the types of colorings



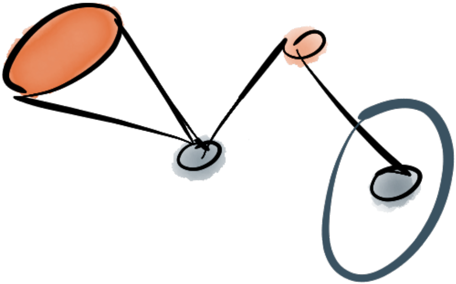
Type B coloring w.r.t blue: $O(n^2)$ instances

Sketches of the types of colorings



Type C coloring w.r.t blue: $O(n^2)$ instances

Sketches of the types of colorings



Type C coloring w.r.t blue: $O(n^2)$ instances

Combining the $O(n^3)$ instances w.r.t the 3 colors, this gives $O(n^9)$ instances to solve.

Triangle-free case

Theorem (BCMSSZ, 2015)

Given a $\{P_7$,triangle$\}$-free graph G, the list 3-coloring problem can be decided, and a coloring can be found, in $O(n^5(n+m))$ time. If G is bipartite, then the complexity drops to $O(n^2(n+m))$.

The algorithm is again based on a structural analysis, controlled enumeration, and reduction to 2-SAT, but the ideas and proofs get simpler.

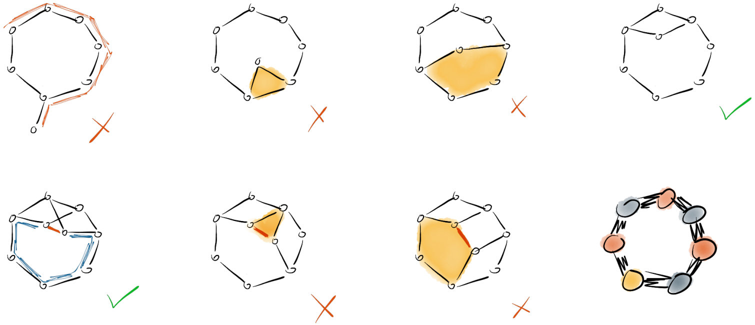
Triangle-free case

We will show how to solve 3-coloring, and how to adapt it to the list version. We thank Daniël Paulusma for pointing out to us that the algorithm for the cases of the C_7 and the C_5 could be trivially adapted to list 3-coloring.

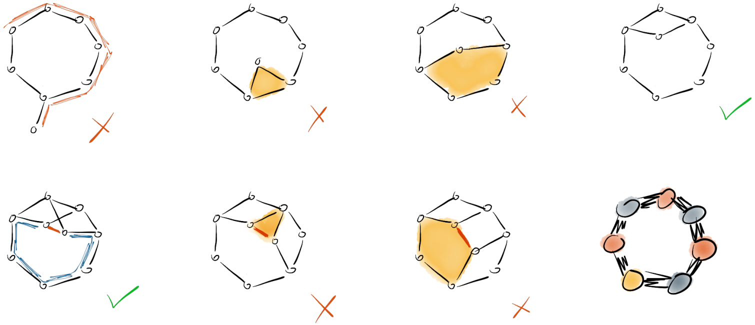
We (quickly) identify three cases:

- the graph is bipartite, so the 3-coloring is trivial (we will deal with the list 3-coloring separately);
- the graph has no induced C_5 but an induced C_7: in this case the graph, after identifying false twins, is C_7 (so the problem and its list version are easy);
- the graph contains an induced C_5: this is the interesting case.

(Triangle, C_5)-free with an induced C_7



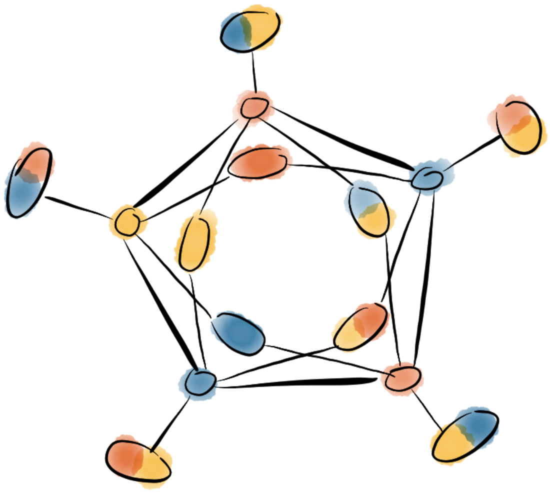
(Triangle, C_5)-free with an induced C_7

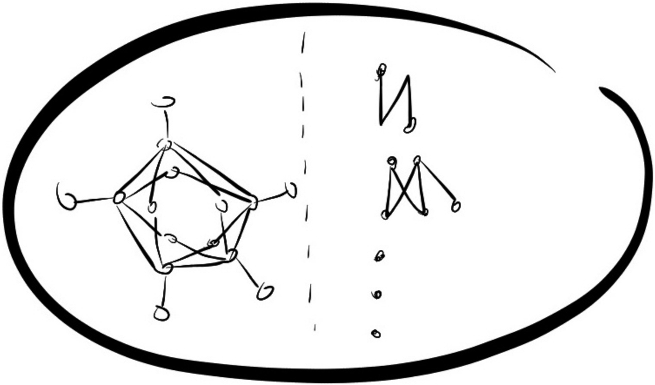


For list 3-coloring we identify, on each class of false twins, those vertices having the same list. The obtained graph has at most 49 vertices.

Triangle-free with an induced C_5

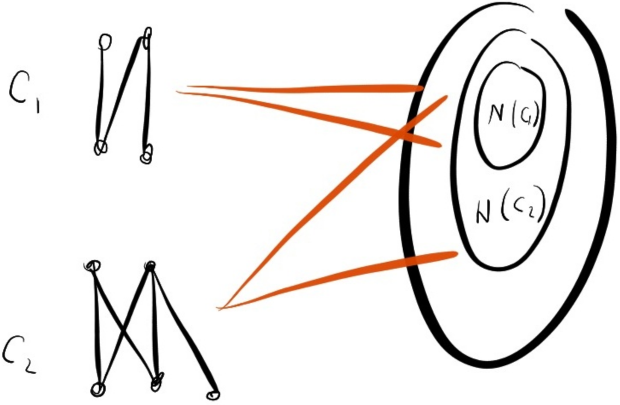
First we determine the core structure of the input graph, for each (valid) 3-coloring of the C_5.



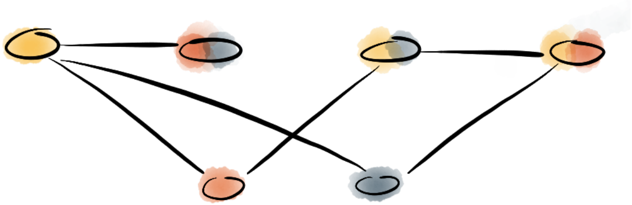


Outside its core structure, the graph is bipartite.

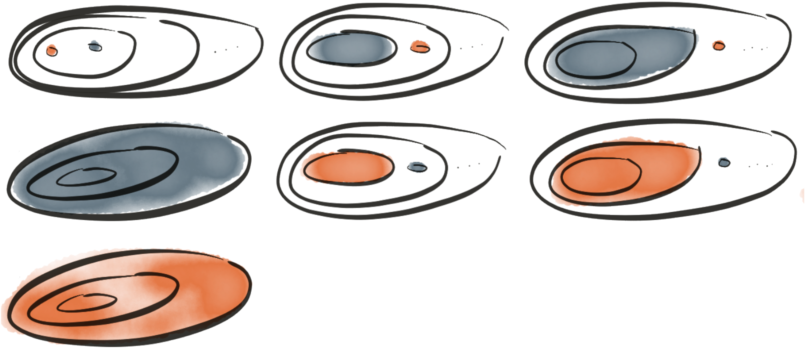
The non-trivial components outside the core structure are well-behaved with respect of the sets of the core structure.



If we have two vertices of different color in the common neighborhood of all the edges (in the core part), each vertex of those edges has at most two colors left.



We enumerate then some partial colorings of the core structure that extend to every possible coloring of the core.



Triangle-free with an induced C_5

- First we enumerate some partial colorings of the sets of the core structure.
- These are determined by the nested neighborhoods of the non-trivial components in each of the "mixed" sets.
- Each partial coloring leaves an instance in which every vertex has at most two admissible colors, or there is a color missing in the list of all its neighbors, and it can be safely use it.
- We reduce then the instance to a 2-SAT problem, that can be solved in $O(n+m)$ time.
- As we have $O(n^5)$ instances, the overall complexity is $O(n^5(n+m))$.

Triangle-free with an induced C_5

- First we enumerate some partial colorings of the sets of the core structure.
- These are determined by the nested neighborhoods of the non-trivial components in each of the "mixed" sets.
- Each partial coloring leaves an instance in which every vertex has at most two admissible colors, or there is a color missing in the list of all its neighbors, and it can be safely use it.
- We reduce then the instance to a 2-SAT problem, that can be solved in $O(n+m)$ time.
- As we have $O(n^5)$ instances, the overall complexity is $O(n^5(n+m))$.

Triangle-free with an induced C_5

- First we enumerate some partial colorings of the sets of the core structure.
- These are determined by the nested neighborhoods of the non-trivial components in each of the "mixed" sets.
- Each partial coloring leaves an instance in which every vertex has at most two admissible colors, or there is a color missing in the list of all its neighbors, and it can be safely use it.
- We reduce then the instance to a 2-SAT problem, that can be solved in $O(n+m)$ time.
- As we have $O(n^5)$ instances, the overall complexity is $O(n^5(n+m))$.

Triangle-free with an induced C_5

- First we enumerate some partial colorings of the sets of the core structure.
- These are determined by the nested neighborhoods of the non-trivial components in each of the "mixed" sets.
- Each partial coloring leaves an instance in which every vertex has at most two admissible colors, or there is a color missing in the list of all its neighbors, and it can be safely use it.
- We reduce then the instance to a 2-SAT problem, that can be solved in $O(n+m)$ time.
- As we have $O(n^5)$ instances, the overall complexity is $O(n^5(n+m))$.

Triangle-free with an induced C_5

- First we enumerate some partial colorings of the sets of the core structure.
- These are determined by the nested neighborhoods of the non-trivial components in each of the "mixed" sets.
- Each partial coloring leaves an instance in which every vertex has at most two admissible colors, or there is a color missing in the list of all its neighbors, and it can be safely use it.
- We reduce then the instance to a 2-SAT problem, that can be solved in $O(n + m)$ time.
- As we have $O(n^5)$ instances, the overall complexity is $O(n^5(n + m))$.

List 3-coloring bipartite graphs

- We first preprocess the graph by eliminating dominated vertices with lists of size 3 (here *dominated* means v and w non adjacent, $N(v) \subseteq N(w)$). We leave just one copy of false twins sets.
- Then we either find two vertices such that every vertex in the graph having a list of size 3 is adjacent to one of them (we can color those two vertices to get 2-SAT instances),
- or we find an induced C_6 (in linear time) with three independent vertices having a list of size 3.
- We define sets of vertices with respect to their neighbors in the C_6, similarly to the C_5 case, and prove some structural properties.

List 3-coloring bipartite graphs

- We first preprocess the graph by eliminating dominated vertices with lists of size 3 (here *dominated* means v and w non adjacent, $N(v) \subseteq N(w)$). We leave just one copy of false twins sets.
- Then we either find two vertices such that every vertex in the graph having a list of size 3 is adjacent to one of them (we can color those two vertices to get 2-SAT instances),
- or we find an induced C_6 (in linear time) with three independent vertices having a list of size 3.
- We define sets of vertices with respect to their neighbors in the C_6, similarly to the C_5 case, and prove some structural properties.

List 3-coloring bipartite graphs

- We first preprocess the graph by eliminating dominated vertices with lists of size 3 (here *dominated* means v and w non adjacent, $N(v) \subseteq N(w)$). We leave just one copy of false twins sets.
- Then we either find two vertices such that every vertex in the graph having a list of size 3 is adjacent to one of them (we can color those two vertices to get 2-SAT instances),
- or we find an induced C_6 (in linear time) with three independent vertices having a list of size 3.
- We define sets of vertices with respect to their neighbors in the C_6, similarly to the C_5 case, and prove some structural properties.

List 3-coloring bipartite graphs

- We first preprocess the graph by eliminating dominated vertices with lists of size 3 (here *dominated* means v and w non adjacent, $N(v) \subseteq N(w)$). We leave just one copy of false twins sets.
- Then we either find two vertices such that every vertex in the graph having a list of size 3 is adjacent to one of them (we can color those two vertices to get 2-SAT instances),
- or we find an induced C_6 (in linear time) with three independent vertices having a list of size 3.
- We define sets of vertices with respect to their neighbors in the C_6, similarly to the C_5 case, and prove some structural properties.

List 3-coloring bipartite graphs

- We define four types of colorings of the C_6.
- We show first that we can test if a type 1 coloring of a cycle can be extended to the whole graph in $O(n+m)$ time.
- Next, we deal with the "parity" case in which all vertices with lists of size 3 have the same parity, and there we can test type 2 or type 3 colorings in $O(n+m)$ time.
- The whole parity case can be reduced to testing $O(n)$ times type 1, type 2 or type 3 colorings, giving a complexity of $O(n(n+m))$.
- In the general case, testing type 2 or type 3 colorings reduces to the parity case ($O(n(n+m))$).
- The general problem can be reduced to testing $O(n)$ times if type 1, type 2 or type 3 colorings, giving a complexity of $O(n^2(n+m))$.

List 3-coloring bipartite graphs

- We define four types of colorings of the C_6.
- We show first that we can test if a type 1 coloring of a cycle can be extended to the whole graph in $O(n+m)$ time.
- Next, we deal with the "parity" case in which all vertices with lists of size 3 have the same parity, and there we can test type 2 or type 3 colorings in $O(n+m)$ time.
- The whole parity case can be reduced to testing $O(n)$ times type 1, type 2 or type 3 colorings, giving a complexity of $O(n(n+m))$.
- In the general case, testing type 2 or type 3 colorings reduces to the parity case ($O(n(n+m))$).
- The general problem can be reduced to testing $O(n)$ times if type 1, type 2 or type 3 colorings, giving a complexity of $O(n^2(n+m))$.

List 3-coloring bipartite graphs

- We define four types of colorings of the C_6.
- We show first that we can test if a type 1 coloring of a cycle can be extended to the whole graph in $O(n+m)$ time.
- Next, we deal with the "parity" case in which all vertices with lists of size 3 have the same parity, and there we can test type 2 or type 3 colorings in $O(n+m)$ time.
- The whole parity case can be reduced to testing $O(n)$ times type 1, type 2 or type 3 colorings, giving a complexity of $O(n(n+m))$.
- In the general case, testing type 2 or type 3 colorings reduces to the parity case ($O(n(n+m))$).
- The general problem can be reduced to testing $O(n)$ times if type 1, type 2 or type 3 colorings, giving a complexity of $O(n^2(n+m))$.

List 3-coloring bipartite graphs

- We define four types of colorings of the C_6.
- We show first that we can test if a type 1 coloring of a cycle can be extended to the whole graph in $O(n+m)$ time.
- Next, we deal with the "parity" case in which all vertices with lists of size 3 have the same parity, and there we can test type 2 or type 3 colorings in $O(n+m)$ time.
- The whole parity case can be reduced to testing $O(n)$ times type 1, type 2 or type 3 colorings, giving a complexity of $O(n(n+m))$.
- In the general case, testing type 2 or type 3 colorings reduces to the parity case ($O(n(n+m))$).
- The general problem can be reduced to testing $O(n)$ times if type 1, type 2 or type 3 colorings, giving a complexity of $O(n^2(n+m))$.

List 3-coloring bipartite graphs

- We define four types of colorings of the C_6.
- We show first that we can test if a type 1 coloring of a cycle can be extended to the whole graph in $O(n+m)$ time.
- Next, we deal with the “parity” case in which all vertices with lists of size 3 have the same parity, and there we can test type 2 or type 3 colorings in $O(n+m)$ time.
- The whole parity case can be reduced to testing $O(n)$ times type 1, type 2 or type 3 colorings, giving a complexity of $O(n(n+m))$.
- In the general case, testing type 2 or type 3 colorings reduces to the parity case ($O(n(n+m))$).
- The general problem can be reduced to testing $O(n)$ times if type 1, type 2 or type 3 colorings, giving a complexity of $O(n^2(n+m))$.

List 3-coloring bipartite graphs

- We define four types of colorings of the C_6.
- We show first that we can test if a type 1 coloring of a cycle can be extended to the whole graph in $O(n+m)$ time.
- Next, we deal with the "parity" case in which all vertices with lists of size 3 have the same parity, and there we can test type 2 or type 3 colorings in $O(n+m)$ time.
- The whole parity case can be reduced to testing $O(n)$ times type 1, type 2 or type 3 colorings, giving a complexity of $O(n(n+m))$.
- In the general case, testing type 2 or type 3 colorings reduces to the parity case ($O(n(n+m))$).
- The general problem can be reduced to testing $O(n)$ times if type 1, type 2 or type 3 colorings, giving a complexity of $O(n^2(n+m))$.

Open problems

- Is there a t such that 3-colorability is NP-complete in P_t-free graphs?
- Is k-colorability FPT in the class of P_5-free graphs?
- Is 4-colorability poly-time solvable in P_6-free graphs?
- Is list 3-colorability poly-time solvable in P_8-free bipartite graphs?