The stubborn problem is stubborn no more

a polynomial algorithm for 3-compatible colouring and the stubborn list partition problem

Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, Jakub Onufry Wojtaszczyk

Faculty of Mathematics, Informatics and Mechanics
University of Warsaw

22nd August 2011,
Young Researchers' Forum

Problem definition

3-Compatible colouring

Input: A complete undirected graph $G = (V, E)$ and a function $\mathcal{C} : E \to \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$

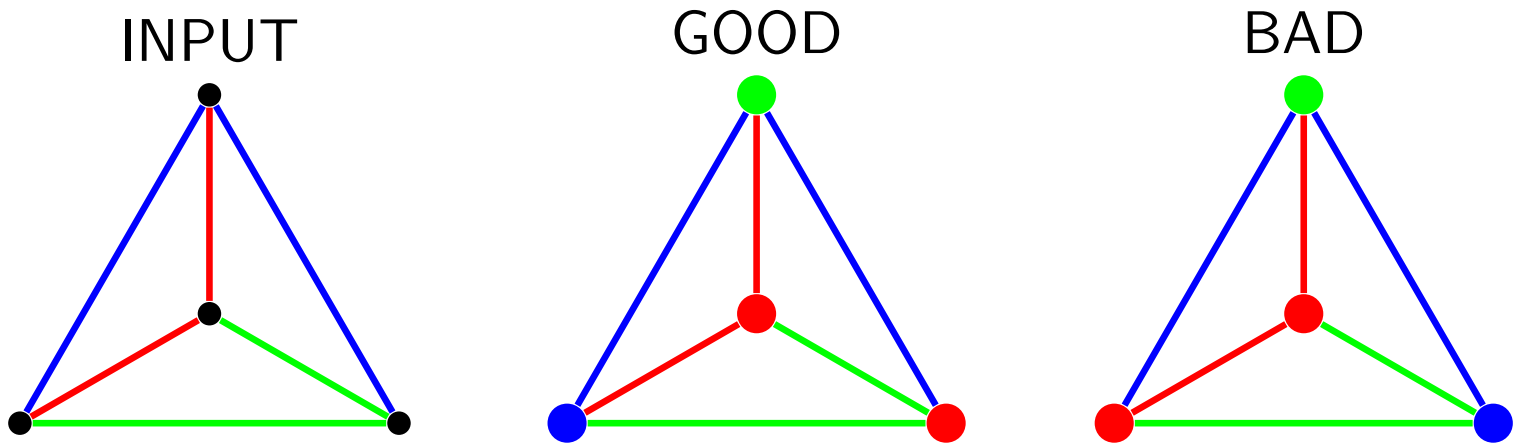
Question: Does there exist a function $\phi : V \to \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$ such that for each edge $uv \in E$ either $\phi(u) \neq \mathcal{C}(uv)$ or $\phi(v) \neq \mathcal{C}(uv)$

Problem definition

3-Compatible colouring

Input: A complete undirected graph $G = (V, E)$ and a function $\mathcal{C} : E \to \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$

Question: Does there exist a function $\phi : V \to \{\mathcal{R}, \mathcal{G}, \mathcal{B}\}$ such that for each edge $uv \in E$ either $\phi(u) \neq \mathcal{C}(uv)$ or $\phi(v) \neq \mathcal{C}(uv)$



Motivation

- 3-Compatible colouring was known to be as hard as the Stubborn problem, an irritating case resisting classification of List Matrix Partition problems by Cameron et al. (SODA 2004).

Motivation

- 3-Compatible colouring was known to be as hard as the Stubborn problem, an irritating case resisting classification of List Matrix Partition problems by Cameron et al. (SODA 2004).
- Feder et al. (SODA 2005) developed a quasipolynomial algorithm with running time $n^{O(\log n / \log \log n)}$.

Motivation

- 3-Compatible colouring was known to be as hard as the Stubborn problem, an irritating case resisting classification of List Matrix Partition problems by Cameron et al. (SODA 2004).
- Feder et al. (SODA 2005) developed a quasipolynomial algorithm with running time $n^{O(\log n/\log\log n)}$.
- Developing a polynomial algorithm for 3-Compatible colouring became a nurturing open problem that inspired a lot of research.

Motivation

- 3-Compatible colouring was known to be as hard as the Stubborn problem, an irritating case resisting classification of List Matrix Partition problems by Cameron et al. (SODA 2004).
- Feder et al. (SODA 2005) developed a quasipolynomial algorithm with running time $n^{O(\log n/\log\log n)}$.
- Developing a polynomial algorithm for 3-Compatible colouring became a nurturing open problem that inspired a lot of research.
- **Our result:** an $O(n^7)$ algorithm.

Warmup: two colours

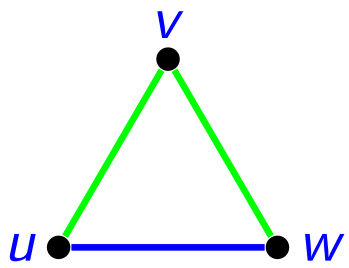
2-Compatible colouring

Input: A complete undirected graph $G = (V, E)$ and a function $\mathcal{C} : E \to \{\mathcal{G}, \mathcal{B}\}$

Question: Does there exist a function $\phi : V \to \{\mathcal{G}, \mathcal{B}\}$ such that for each edge $uv \in E$ either $\phi(u) \neq \mathcal{C}(uv)$ or $\phi(v) \neq \mathcal{C}(uv)$

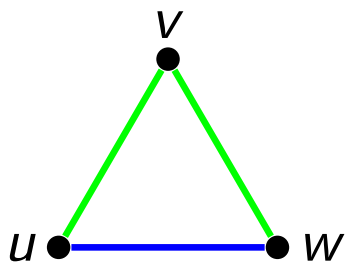
Two colours: solution

- **Observation:** If $\mathcal{C}(uv) = \mathcal{C}(vw) \neq \mathcal{C}(uw)$, then v has determined colour.



Two colours: solution

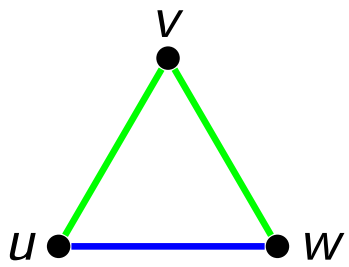
- **Observation:** If $\mathcal{C}(uv) = \mathcal{C}(vw) \neq \mathcal{C}(uw)$, then v has determined colour.



- **Algorithm:**

Two colours: solution

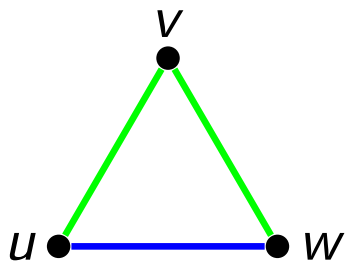
- **Observation:** If $\mathcal{C}(uv) = \mathcal{C}(vw) \neq \mathcal{C}(uw)$, then v has determined colour.



- **Algorithm:**
  - Until the graph becomes a monochromatic clique, find such a triangle, determine the colour of the vertex and propagate this information by deleting vertices with already decided colours.

Two colours: solution

- **Observation:** If $\mathcal{C}(uv) = \mathcal{C}(vw) \neq \mathcal{C}(uw)$, then v has determined colour.



- **Algorithm:**
  - Until the graph becomes a monochromatic clique, find such a triangle, determine the colour of the vertex and propagate this information by deleting vertices with already decided colours.
  - Then we have $m + 1$ solutions: all the vertices of the second colour, or just one of the first.

Three colours: overview

- We introduce vertices one by one, each time trying to obtain some feasible colouring of the clique induced by already introduced ones.

Three colours: overview

- We introduce vertices one by one, each time trying to obtain some feasible colouring of the clique induced by already introduced ones.
- When we introduce the next vertex, we try 3 possibilities of its colour. If one succeeds, we immediately get to the next step.

Three colours: overview

- We introduce vertices one by one, each time trying to obtain some feasible colouring of the clique induced by already introduced ones.
- When we introduce the next vertex, we try 3 possibilities of its colour. If one succeeds, we immediately get to the next step.
- **Goal:** we have (some) feasible colouring of a subclique (denoted previous) and we seek a new colouring with one more vertex.

Three colours: overview

- We introduce vertices one by one, each time trying to obtain some feasible colouring of the clique induced by already introduced ones.
- When we introduce the next vertex, we try 3 possibilities of its colour. If one succeeds, we immediately get to the next step.
- **Goal:** we have (some) feasible colouring of a subclique (denoted $\mathrm{previous}$) and we seek a new colouring with one more vertex.
- **Note:** In the new colouring all the vertices can be repainted; the previous colouring is just a 'hint'.

List version

Each vertex v has a list $L(v)$ of available colours.

List version

Each vertex v has a list $L(v)$ of available colours.

- If there is a vertex with $|L(v)| = 0$, answer NO.

List version

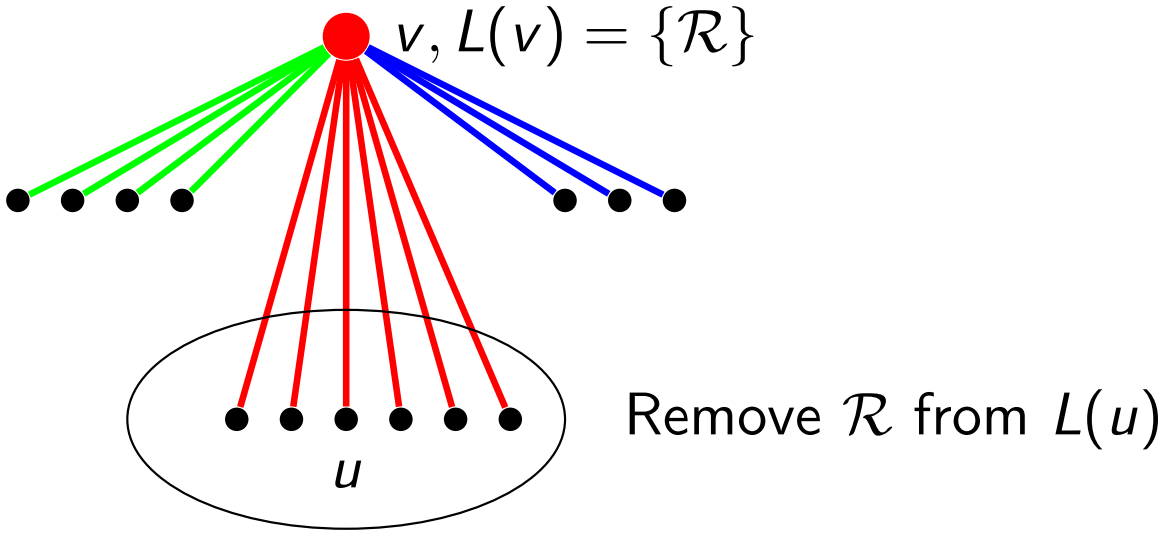
Each vertex v has a list $L(v)$ of available colours.

- If there is a vertex with $|L(v)| = 0$, answer NO.
- If there is a vertex with $|L(v)| = 1$, reduce the instance.

List version

Each vertex v has a list $L(v)$ of available colours.

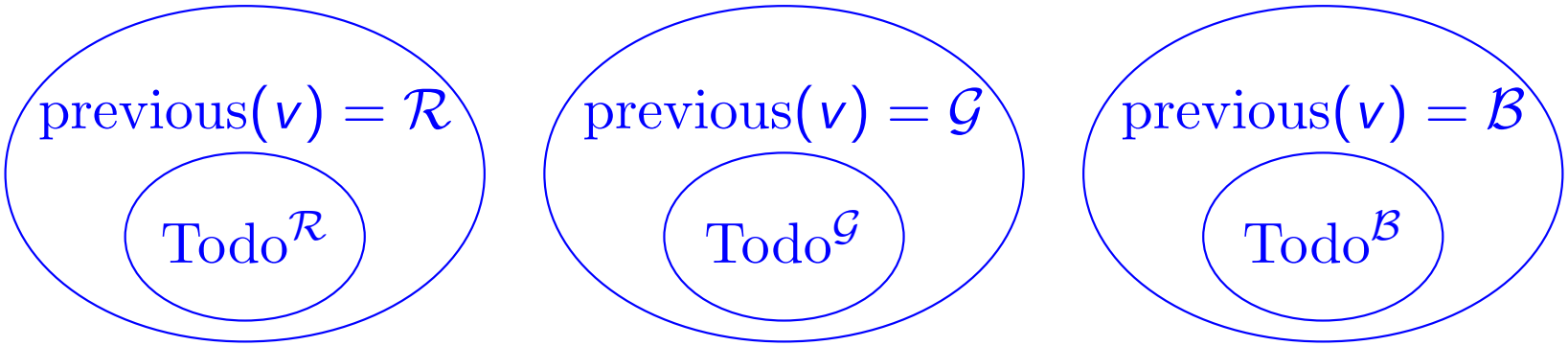
- If there is a vertex with $|L(v)| = 0$, answer NO.
- If there is a vertex with $|L(v)| = 1$, reduce the instance.



Structure of the problem

We define subsets of vertices according to their lists of available colours and the previous colour.

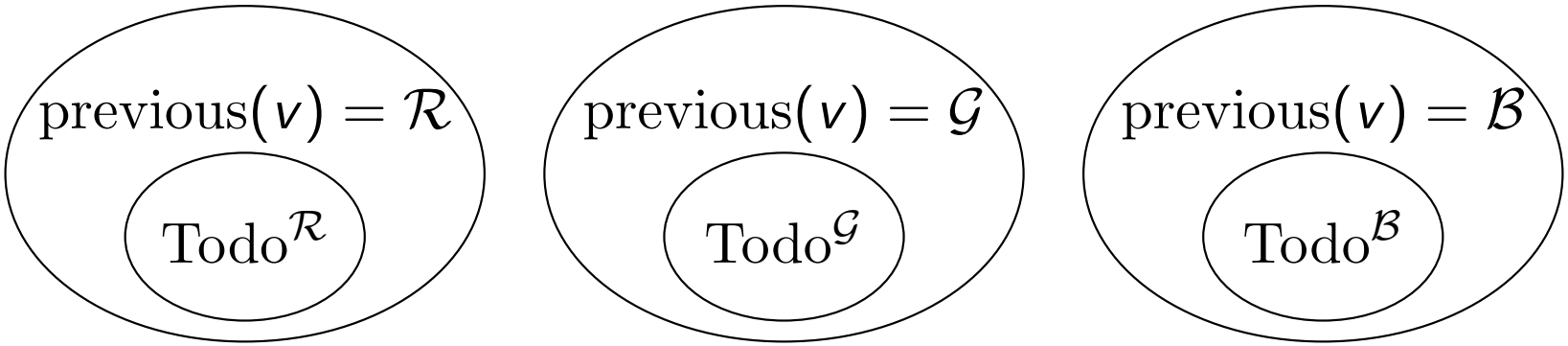
$$\text{Todo}^x = \{v : \text{previous}(v) = x, |L(v)| = 2, x \notin L(v)\}$$



Structure of the problem

We define subsets of vertices according to their lists of available colours and the previous colour.

$$\mathrm{Todo}^x = \{v : \mathrm{previous}(v) = x, |L(v)| = 2, x \notin L(v)\}$$



Intuition: Todo^x are the vertices that need to be repainted.

Resolving Todo

$$\text{Todo}^x = \{v : \text{previous}(v) = x, |L(v)| = 2, x \notin L(v)\}$$

Resolving Todo

$$\mathrm{Todo}^x = \{v : \mathrm{previous}(v) = x, |L(v)| = 2, x \notin L(v)\}$$

Observation 1

If all Todo sets are empty then we return $\phi(v) := \mathrm{previous}(v)$.

Resolving Todo

$$\mathrm{Todo}^x = \{v : \mathrm{previous}(v) = x, |L(v)| = 2, x \notin L(v)\}$$

Observation 1

If all Todo sets are empty then we return $\phi(v) := \mathrm{previous}(v)$.

Observation 2

In $E(\mathrm{Todo}^x)$ there are no edges of colour x, hence $G[\mathrm{Todo}^x]$ is an instance of 2-Compatible colouring.

Resolving Todo

- As long as some Todo is not monochromatic, we can extract a vertex from it, just as in the two colour case.

Resolving Todo

- As long as some Todo is not monochromatic, we can extract a vertex from it, just as in the two colour case.
- If all are monochromatic cliques, we become stuck...

Resolving Todo

- As long as some Todo is not monochromatic, we can extract a vertex from it, just as in the two colour case.
- If all are monochromatic cliques, we become stuck...
- And now the magic begins.

Resolving Todo

- As long as some Todo is not monochromatic, we can extract a vertex from it, just as in the two colour case.
- If all are monochromatic cliques, we become stuck...
- And now the magic begins.
- We do simple branching: take some nonempty Todo^x and branch into $|\mathrm{Todo}^x| + 1$ possibilities.

Time analysis

How much information we gained in each branch?

Time analysis

How much information we gained in each branch?

We gain on reducing the number of vertices with full lists.

Time analysis

How much information we gained in each branch?

We gain on reducing the number of vertices with full lists.

$$\mathrm{Free}^x = \{v : \mathrm{previous}(v) = x, |L(v)| = 3\}$$

Time analysis

How much information we gained in each branch?

We gain on reducing the number of vertices with full lists.

$$\mathrm{Free}^x = \{v : \mathrm{previous}(v) = x, |L(v)| = 3\}$$

Key Lemma

If we branch on colourings of Todo^x, then in all new instances the sets Free^x are disjoint.

Time analysis: proof of the Key Lemma

Key Lemma

If we branch on colourings of Todo^x then in all new instances the sets Free^x are disjoint.

Time analysis: proof of the Key Lemma

Key Lemma

If we branch on colourings of Todo^x then in all new instances the sets Free^x are disjoint.

- Assume a vertex v_1 that stays in Free^x in at least two branches.

Time analysis: proof of the Key Lemma

Key Lemma

If we branch on colourings of Todo^x then in all new instances the sets Free^x are disjoint.

- Assume a vertex v_1 that stays in Free^x in at least two branches.
- Let ϕ, ϕ' be two different 2-compatible colourings on Todo^x in those branches.

Time analysis: proof of the Key Lemma

Key Lemma

If we branch on colourings of Todo^x then in all new instances the sets Free^x are disjoint.

- Assume a vertex v_1 that stays in Free^x in at least two branches.
- Let ϕ, ϕ' be two different 2-compatible colourings on Todo^x in those branches.
- Let v_2 be a vertex such that $\phi(v_2) \neq \phi'(v_2)$.

Time analysis: proof of the Key Lemma

Key Lemma

If we branch on colourings of Todo^x then in all new instances the sets Free^x are disjoint.

- Assume a vertex v_1 that stays in Free^x in at least two branches.
- Let ϕ, ϕ' be two different 2-compatible colourings on Todo^x in those branches.
- Let v_2 be a vertex such that $\phi(v_2) \neq \phi'(v_2)$.
- Since $\mathcal{C}(v_1 v_2) \neq x$, w.l.o.g. we may assume $\mathcal{C}(v_1 v_2) = \phi(v_2)$, hence v_1 is not free in the branch corresponding to ϕ, a contradiction.

Time analysis

- We bound the number of leaves of the branch tree by $O(n^3)$ using a potential function

$$\pi = (\mathrm{Free}^{\mathcal{R}} + \mathrm{Todo}^{\mathcal{R}} + 1)(\mathrm{Free}^{\mathcal{G}} + \mathrm{Todo}^{\mathcal{G}} + 1)(\mathrm{Free}^{\mathcal{B}} + \mathrm{Todo}^{\mathcal{B}} + 1).$$

Time analysis

- We bound the number of leaves of the branch tree by $O(n^3)$ using a potential function

$$\pi = (\mathrm{Free}^{\mathcal{R}} + \mathrm{Todo}^{\mathcal{R}} + 1)(\mathrm{Free}^{\mathcal{G}} + \mathrm{Todo}^{\mathcal{G}} + 1)(\mathrm{Free}^{\mathcal{B}} + \mathrm{Todo}^{\mathcal{B}} + 1).$$

- Vertices from Todo^x can originate only in Free^x, so π does not increase when reducing vertices with fixed colour.

Time analysis

- We bound the number of leaves of the branch tree by $O(n^3)$ using a potential function

$$\pi = (\mathrm{Free}^{\mathcal{R}} + \mathrm{Todo}^{\mathcal{R}} + 1)(\mathrm{Free}^{\mathcal{G}} + \mathrm{Todo}^{\mathcal{G}} + 1)(\mathrm{Free}^{\mathcal{B}} + \mathrm{Todo}^{\mathcal{B}} + 1).$$

- Vertices from Todo^x can originate only in Free^x, so π does not increase when reducing vertices with fixed colour.
- Due to the Key Lemma, in the branching step the sum of the potentials in the branches is at most the prior potential.

Time analysis

- We bound the number of leaves of the branch tree by $O(n^3)$ using a potential function

$$\pi = (\mathrm{Free}^{\mathcal{R}} + \mathrm{Todo}^{\mathcal{R}} + 1)(\mathrm{Free}^{\mathcal{G}} + \mathrm{Todo}^{\mathcal{G}} + 1)(\mathrm{Free}^{\mathcal{B}} + \mathrm{Todo}^{\mathcal{B}} + 1).$$

- Vertices from Todo^x can originate only in Free^x, so π does not increase when reducing vertices with fixed colour.
- Due to the Key Lemma, in the branching step the sum of the potentials in the branches is at most the prior potential.
- The time complexity is $O(n^7)$ in total.

Thank you for your attention

Questions?