Understanding and implementation of the algorithm for three-coloring in triangle-free planar graphs

Bachelor Thesis Defense

Qianli Wang

06.05.2021

Outline

- Introduction
- Safety of multigrams
- Main argument for detecting multigrams
- Proof of Grötzsch's theorem
- Implementation

Introduction

Definition 1:

Given a graph $G = (V, E)$ and a positive integer K such that $K < |V|$. The graph *K-COLORING problem* is to determine whether there exists a conflict-free vertex coloring using K colors or less.

- What we focus on is $3 - COLORING$ problem. However, $3 - COLORING$ problem is *NP-complete*.
- Assume that the given graph $G = (V, E)$ is planar and triangle-free.

Introduction

- In 1959, Grötzsch's theorem stated 3-colorability of a planar and triangle-free graph.
- In 2003, Thomassen's proofs can be developed an algorithm for 3-coloring in $\mathcal{O}(n^2)$ time.
- In 2004, Kowalik found a $\mathcal{O}(nlogn)$-time algorithm.
- In 2011, Dvorak, Kawaravayashi and Thomas designed a linear-time algorithm.

Introduction

Definition 2:

Given a graph $G = (V, E)$, *identifying a pair of vertices* means that the two selected vertices $u, v \in V(G)$ will be glued as a vertex s and the neighbor of s is the union of u, v's neighbors. So we get a new graph $G' = (V', E')$, where $V' = V \backslash \{u, v\} \cup \{s\}$ and E' will be obtained by deleting parallel edges from E after gluing the vertices.

Introduction - Core idea

(1) **Detection and Reduction**

(2) **Reconstruction**

Safety of multigrams

Definition 3:

A *tetragram* is a sequence (v_1, v_2, v_3, v_4) of vertices of G such that it can build a facial cycle in this listed order. Analogously, we can define a *hexagram* $(v_1, v_2, v_3, v_4, v_5, v_6)$. And a *pentagram* $(v_1, v_2, v_3, v_4, v_5)$ is also defined likewise and v_1, v_2, v_3, v_4 have degree exactly three.

Definition 4.1 (Safety of tetragrams and hexagrams):

Assume that $k = 4, 5, 6$ and $(v_1, v_2, ..., v_k)$ be a tetra-, penta- or hexagram in a triangle-free planar graph G. The tetragram or hexagram is *safe* if every path in G of length at most three with ends v_1 and v_3 is a subgraph of cycle $v_1 v_2 ... v_k$.

Safety of multigrams

Definition 4.2 (Safety of pentagrams):

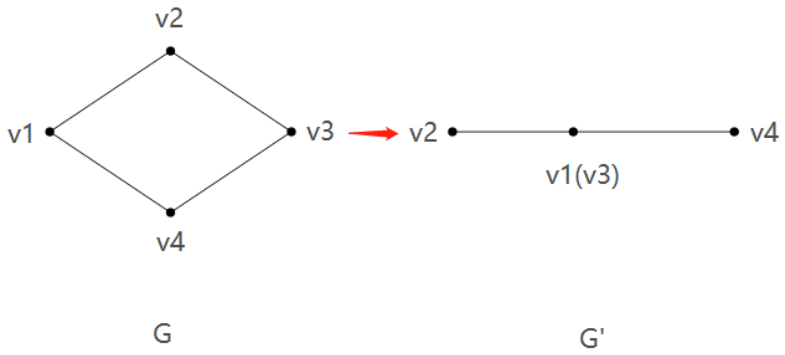
Let x_i be the neighbor of v_i different from v_{i-1} and v_{i+1}, where $v_0 = v_5$ and $\forall x_i \notin \{v_1, v_2, v_3, v_4, v_5\}, i \in \{1, 2, 3, 4\}$. A pentagram $(v_1, v_2, v_3, v_4, v_5)$ is *safe*, if

- the vertices x_1, x_2, x_3, x_4 are pairwise distinct and pairwise non-adjacent, and
- there is no path in $G \backslash \{v_1, v_2, v_3, v_4\}$ of length at most three from x_2 to v_5, and
- every path in $G \backslash \{v_1, v_2, v_3, v_4\}$ of length at most three from x_3 to x_4 has length exactly two, and its completion via the path $x_3 v_3 v_4 x_4$ results in a facial cycle of length five in G.

Safety of multigrams

How to identify vertices in multigrams?

For *tetragram* (v_1, v_2, v_3, v_4), $G^{'}$ can be obtained by identifying vertices v_1 with v_3.



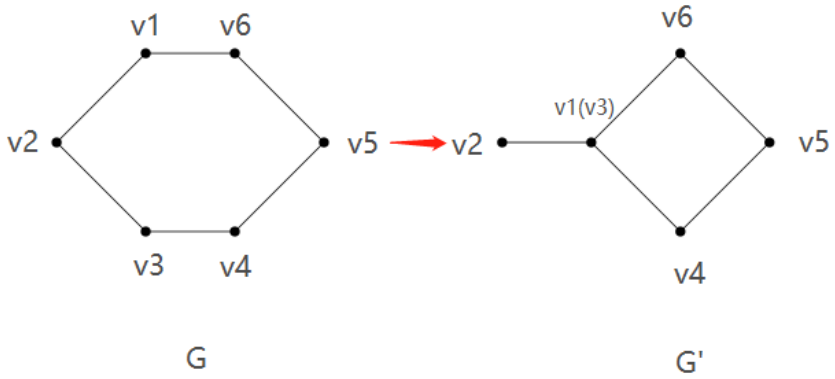
Reconstruction Step:

In graph $G^{'}$, let c_1 be the color of v_2, c_2 be the color of v_1 and c_3 be the color of v_4.

Safety of multigrams

How to identify vertices in multigrams?

For *hexagram* $(v_1, v_2, v_3, v_4, v_5, v_6)$, $G^{'}$ can be obtained by identifying vertices v_1 with v_3.



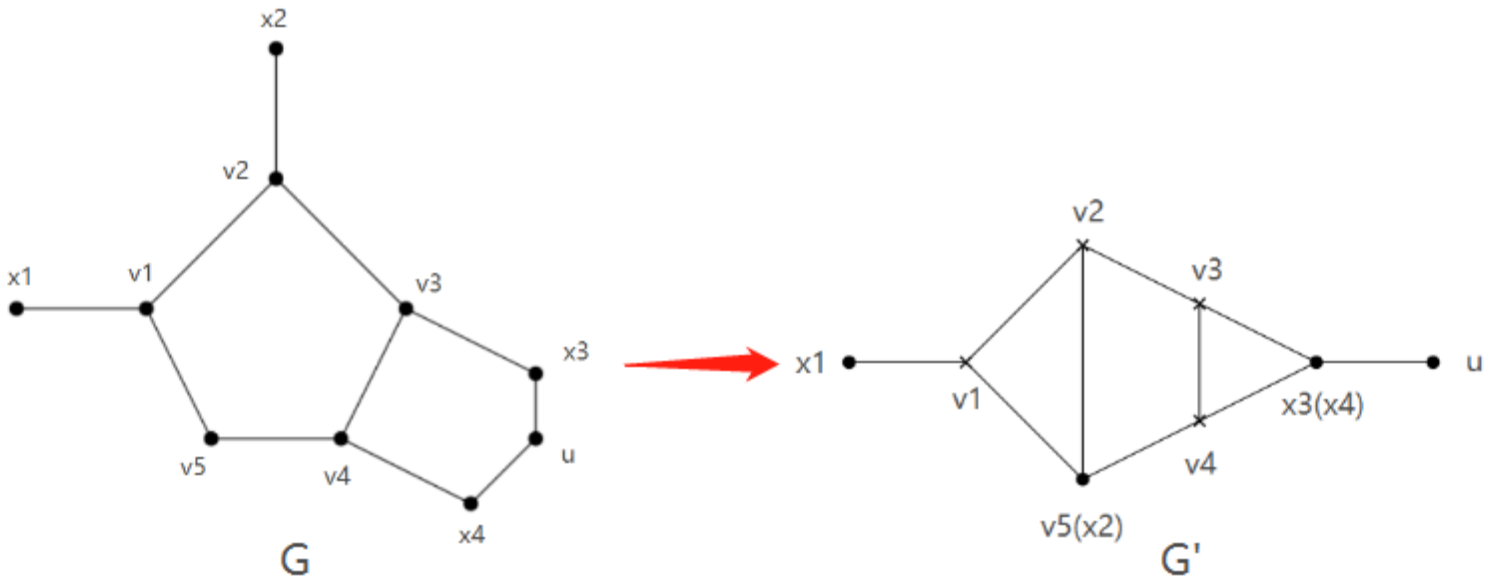
Reconstruction Step:

In graph $G^{'}$, let c_1 be the color of v_2, c_2 be the color of v_1. Assume that the color of v_4 and v_6 is c_3 and the color of v_5 is c_1. Apparently, the color c_2 can be designated for v_3 as well.

Safety of multigrams

How to identify vertices in multigrams?

For pentagram $(v_1, v_2, v_3, v_4, v_5)$, G' will be attained from $G \backslash \{v_1, v_2, v_3, v_4\}$ by identifying v_5 with x_2 and x_3 with x_4.



Reconstruction Step:

In graph G', let c_1 be the color of x_1, c_2 the color of x_2 and v_5, and c_3 the color of x_3 and x_4. Consider the following cases:

- $c_1 = c_2$, we will color the remaining vertices in this order (v_4, v_3, v_2, v_1). Since there are at most two remaining colors when v_i is colored, we can simply choose the third color for it.
- $c_2 = c_3$, similarly, we will color the vertices in the reversed order as in the first case.
- $c_1 \neq c_2$ and $c_2 \neq c_3$, we can color v_2 with c_1, v_1 with c_3, v_3 with c_2 and v_4 with c_1.

Demonstration

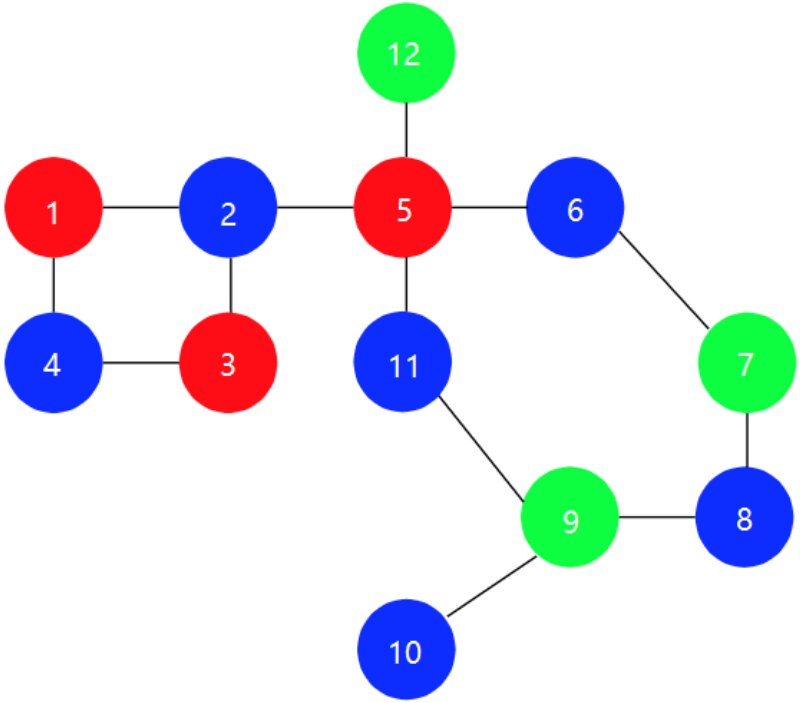


Figure: The input graph

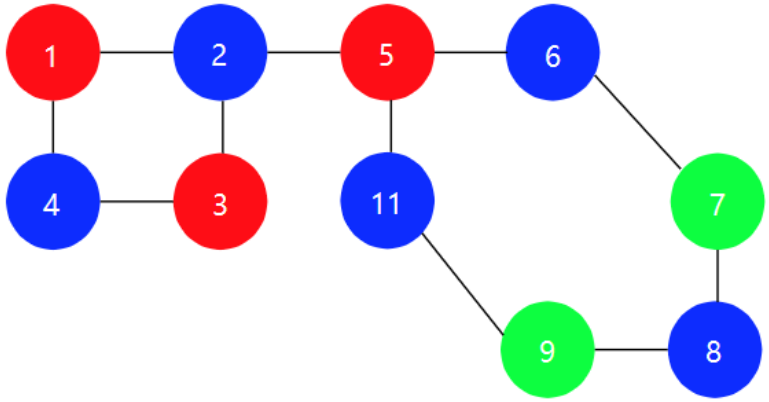


Figure: Remove vertices with degree one

Demonstration

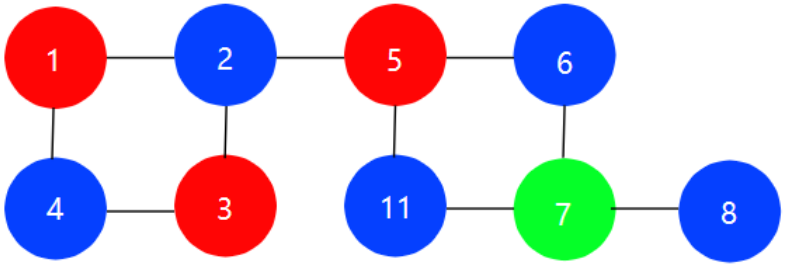


Figure: Identify vertices 7 and 9

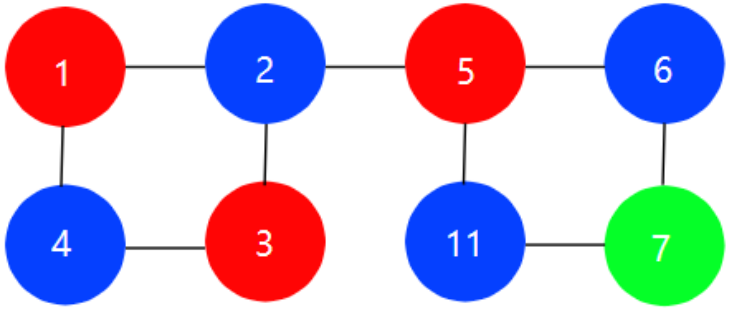


Figure: Remove vertices with degree one

Demonstration

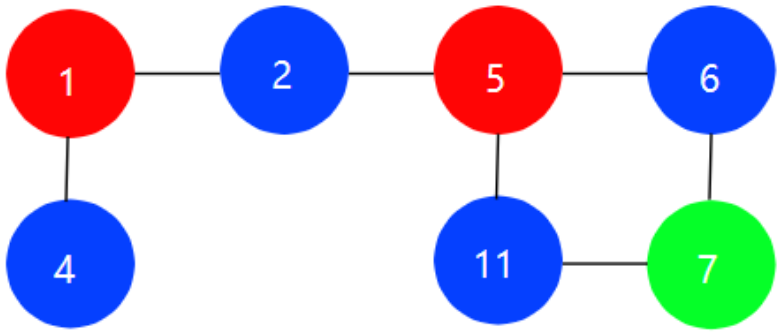


Figure: Identify vertices 1 and 3

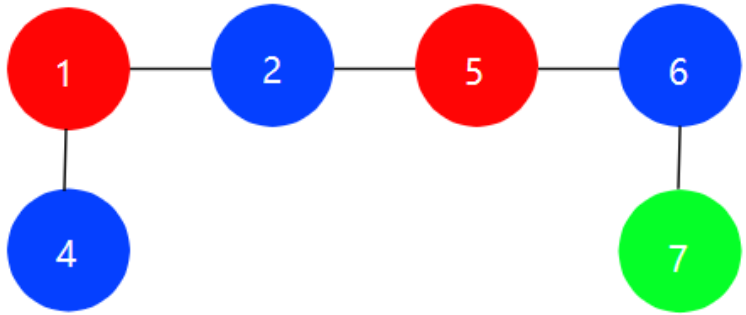


Figure: Identify vertices 6 and 11

Main argument for detecting multigrams

Lemma 1:

Every triangle-free plane graph G of minimum degree at least three has a safe tetragram, a safe pentagram or a safe hexagram.

Proof of Grötzsch's Theorem

Grötzsch's Theorem:

Every triangle-free planar graph is 3-colorable.

Proof of Grötzsch's Theorem:

Let G as stated to be a triangle-free graph.

(1) If there is a vertex v with degree one or two, delete it. While reconstructing, give v a color different from its neighbor(s).

(2) If the minimum degree is at least three, apply Lemma 1 on the graph.

Implementation

Brute Force Algorithm:

(1) Start with a vertex $v \in V$ and just go through all its adjacent vertices which should be pairwise distinct. In this step, we will use multiple inner for-loops to achieve that.

(2) Check the condition whether the detected multigram is safe or not.

(3) If the founded multigram is safe, then identify vertices.

(4) Reconstruct the coloring of the graph

Testing safety:

- Found multigram is tetragram/hexagram. We can use the same way by using two for-loops to go through all neighbors and neighbors of neighbors so that we can check that there is no path of length at most three from v_1 to v_3 which is not part of the tetragram/hexagram. This can be done in $\mathcal{O}(m^2)$ time, where $m = |E|$.
- Found multigram is pentagram. As mentioned in the former case, we can also use two for-loops to check whether there is no path in $G \backslash \{v_1, v_2, v_3, v_4\}$ of length at most three from x_2 to v_5, and every path in $G \backslash \{v_1, v_2, v_3, v_4\}$ of length at most three from x_3 to x_4 has length exactly two. This can be done in $\mathcal{O}(m^2)$ time as well.

Implementation - Running time analyse

Observation:

The running time of the Brute Force algorithm is $\mathcal{O}(m^8)$ in the worst case.

Java Implementation

```
The info of input graph:
Tetragram:
[6, 10, 11, 12]
Pentagram:
[1, 2, 3, 4, 5]
Hexagram:
Number of tetragram: 1
Number of pentagram: 1
Number of hexagram: 0
Remaining #vertex: 12

Identified vertices: [6, 11] in 0-th iteration
Tetragram:
Pentagram:
[1, 2, 3, 4, 5]
Hexagram:
Number of tetragram: 0
Number of pentagram: 1
Number of hexagram: 0
Remaining #vertex: 11

Identified vertices: [7, 5] in 1-th iteration
Identified vertices: [8, 9] in 2-th iteration
Tetragram:
Pentagram:
Hexagram:
Number of tetragram: 0
Number of pentagram: 0
Number of hexagram: 0
Remaining #vertex: 5
```

```
Vertex 0 --->  Color 0
Vertex 1 --->  Color 0
Vertex 2 --->  Color 0
Vertex 3 --->  Color 0
Vertex 4 --->  Color 0
Vertex 5 --->  Color 0
Vertex 6 --->  Color 0
Vertex 7 --->  Color 0
Vertex 8 --->  Color 0
Vertex 9 --->  Color 0
Vertex 10 --->  Color 1
Vertex 11 --->  Color 0
Vertex 12 --->  Color 1
************************
Vertex 0 --->  Color 0
Vertex 1 --->  Color 1
Vertex 2 --->  Color 2
Vertex 3 --->  Color 1
Vertex 4 --->  Color 2
Vertex 5 --->  Color 0
Vertex 6 --->  Color 0
Vertex 7 --->  Color 0
Vertex 8 --->  Color 0
Vertex 9 --->  Color 0
Vertex 10 --->  Color 1
Vertex 11 --->  Color 0
Vertex 12 --->  Color 1
************************
```

Implementation - Virtualization

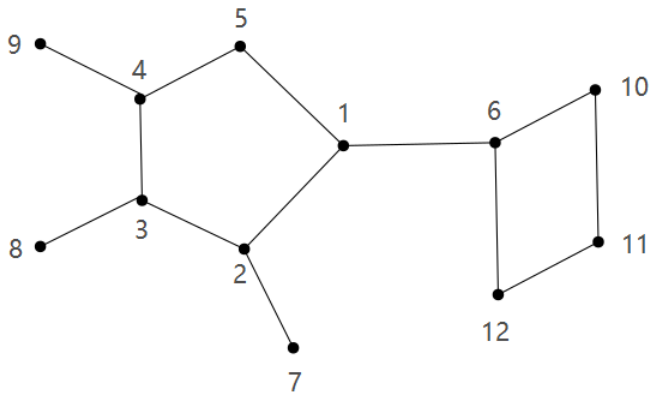


Figure: The input graph

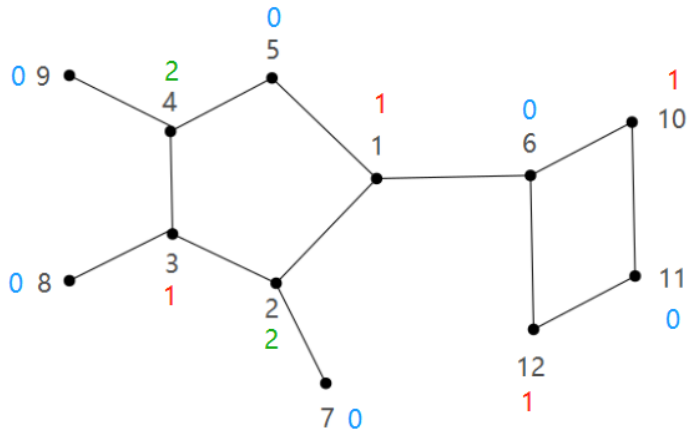


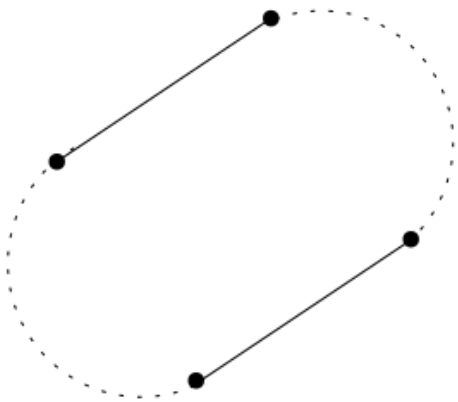
Figure: The proper coloring

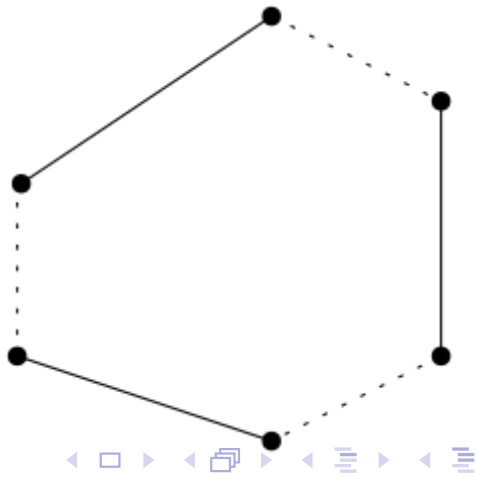
Implementation - Improvments

- **Tetragram.**
  (1) Use two loops to go through all edges $e_1, e_2 \in E$, where $e_1 \neq e_2$ and check whether they form a cycle.
  (2) Check the condition of safety by using BFS.

  The running time of this improved algorithm for detecting tetragrams is $\mathcal{O}(m^3)$.
- **Hexagram.**
  (1) Use three loops to go through all edges $e_1, e_2, e_3 \in E$, where $e_1 \neq e_2$ and $e_2 \neq e_3$ and check whether they form a cycle.
  (2) Check the condition of safety by using BFS analogously.

  Totally, the running time is $\mathcal{O}(m^4)$.





Implementation - Improvments

- **Pentagram.** We use two loops to go through all edges $e_1, e_2 \in E$, where $e_1 \neq e_2$ and another loop to go through all remaining vertices. Then, we apply BFS to check wether there is a path in $G \backslash \{v_1, v_2, v_3, v_4\}$ of length at most three from x_2 to v_5 and every path in $G \backslash \{v_1, v_2, v_3, v_4\}$ of length at most three has length exactly two. Therefore, the running time is $\mathcal{O}(m^4)$.

Conclusion:

As a result, the running time can be improved to $\mathcal{O}(m^4)$ in the worst case.

End

Thank you for your attention!