Recoloring k-trees

Jozef Jirásek
jirasekjozef@gmail.com
Pavel Klavík
pavel@klavik.cz

April 29, 2008

Abstract

1 Introduction

Consider we have a d-degenerate graph G with n vertices, and a two correct colorings of G by $d + c$ colors. We ask how many operations (in relation to n) consisting of changing the color of one vertex are necessary to change one coloring of G to the other, such that G is correctly colored during the entire process. For $c = 1$ this is not always possible (consider a complete graph with $d + 1$ vertices). For $c = 2$ it can be easily proved that there always exists such a sequence of steps, although the maximum number of steps might be exponential in relation to number of vertices of G.

Consider two special subclasses of d-degenerate graphs, partial k-trees and planar graphs. In the first part of the article we will describe method to recolor partial k-trees in time $\Theta(n^2)$. Firstly consider just k-paths because they are simple and their recoloring is almost same like partial k-trees. Then we will show you one really ugly 2-path that requires $\Omega(n^2)$ steps to recolor.

In the end of this article we will show you way to prove that within planar graphs we need only to consider triangulations. We use typical trick to add vertices and edges to each face of the planar graph to create a triangulation. But it have to be done carefully because we need not to block both colorings.

2 Definitions

Within the article we consider simple connected graphs. If graph is not connected, we can simply recolor each component independently. A *d-degenerate graph* is a graph that can be reduced to the empty graph by removing vertices of degree at most d.

A *coloring* of graph G by c colors is a function $f : V(G) \to \{1, \ldots, c\}$. A *correct coloring* is a coloring such that for no two connected vertices u and v holds $f(u) = f(v)$. A graph G with such a function f is a *colored graph* G_f. We will denote

the values of f by capital letters $A, B, \ldots$. The intuitive definition of coloring will be often used, such as "vertex u has a color A" or "change the color of u from A to B."

The *recoloring* of a colored graph G_f to a different coloring G_h is a series of *steps*, where each steps consists of changing the value of $f(v)$ for one vertex $v \in V(G)$, such that after every step G is correctly colored. Before the first step the coloring of G is f, after the last step it is h. The *length* of such recoloring is the number of steps it contains.

3 Recoloring of d-degenerate graphs

A d-degenerate graph can always be colored by $d+1$ colors, but there might not always exist a recoloring between two such colorings. Consider a complete graph with $d+1$ vertices, it is clearly d-degenerate and changing the color of any vertex will create an incorrect coloring.

Theorem 1. *Consider d-degenerate graph G and his two colorings with $d+2$ colors. There always exists some recoloring between them within exponential number of steps.*

Proof: We will prove it by induction for number of vertices of graph. For $n = 1$ it is simple, any graph with only one vertex can be recolored. We want to prove $n+1 \leftarrow n$. Lets take degenerated vertex v and rest of the graph on n vertex, call it G'. We know that there exists a recoloring for G'. We try to use this recoloring for graph but color of vertex v can collide with colors of G'. But there is always one free color for v to swap, because we have $d+2$ colors, but v has only d neighbours. So if color of v collides than we swap it and continue within the recoloring procedure of G'. This also proves the exponential amount of steps, for each new vertex we can change his color between all steps of recoloring of graph without it. Q.E.D.

4 Recoloring k-paths

A k-path is a graph created from a single path (or 1-path) by adding edges between each two vertices which are distant at most k vertices on the path.

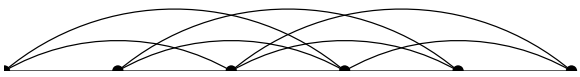

Figure 1: A 3-path on 6 vertices.

Each k-path is also k-degenerate, because we can remove the vertices of degree k from one end of the path.

4.1 Coloring tree

If we have a fixed coloring of several (at least k) vertices from one end of a k-path G, the first still uncolored vertex v can get only one of two colors. It has k neighbors

with already fixed colors, and those from a clique, therefore each of them must have a different color. As we have only $k + 2$ colors and cannot use any of the k colors given to the neighbors of v, v must have one of the two remaining colors. It is at figure 2 on the left.

Next vertex u has also fixed k vertices above, so we can choose from two colors. But which colors are blocked depends on color of v, so we have four possible situation. Look at figure 2 on the right. If we continue same way, we can build recursively *coloring tree* T for k-path.

Figure 2: Recoloring tree for 2-path – depth 1 (left) and 2 (right).

Each vertex u in k-path G corresponds with one level of vertices in coloring tree T. Each path from root to leaf in T is one correct coloring of k-path. Also, we can see that the possible choice of colors for one vertex depends only on colors of k previous vertices. We will note colors of these vertices in lower-left index, to stress the fact that the last color is the actual color the vertex will have.

Figure 3: Recoloring tree for 2-path – really deep one.

Lemma 2. *After $k + 1$ levels, the whole tree structure begins to repeat itself in exactly two subtrees of T.*

Proof: Because the color of a vertex v depends only on the colors of k previous vertices on the path, the tree structure of T starting at vertex v is determined by the combination

of the color of v and colors of k previous vertices. Therefore, if these colors appear on any vertex of T, the subtree starting with that vertex will be the same. It is also easy to see that given a vertex t of T, the lower index of both children of t will be the same as the lower index of t without the first color, and with the color of t as the last color. Q.E.D.

Therefore, if we want the combination of colors to repeat, we have to reroll through all those colors to get the same combination again, which takes $k + 1$ steps. Because we have two possible colors to begin with, and then we have to follow the sequence we attempt to reach, there are exactly two ways how it can be done, resulting in two subtrees with the same structure as the original tree. For an example, see the topmost or bottommost path in the previous image.

We can choose order of subtrees, so the repeating structures will always be to the leftmost and the rightmost subtree, as shown at figure 3.

4.2 Recoloring the k-path

Now we have established that for a given k-path G with several vertices at one end already colored we can uniquely construct a coloring tree T, and that every correct coloring of G corresponds with one path from the root of T to one of its leaves. If we imagine this path as a series of directions to "go left" or "go right" as we progress through the tree, we can represent it as a sequence of left and right arrows with length equal to the depth of T. And because every such path produces an unique sequence of arrows, and every such sequence describes a correct path, and therefore a valid coloring of G, we can state following.

Theorem 3. *Every valid coloring of G with fixed color of at least k vertices from the end can be represented as a sequence of left and right arrows.*

Proof: Each valid coloring of G is represented by one path from root to leaf in coloring tree T. And each path in T can be represented as a unique sequence of left and right arrows. Q.E.D.

We want to change color of one vertex. Consider all of the colors of vertices before that fixed. But the path of current coloring must pass through the leftmost or rightmost subtree, because otherwise we cannot swap the color of this vertex.

Generally, if there is a sequence of $k + 1$ subsequent arrows pointing in one direction, we can find a place in the coloring tree T where the path goes $k + 1$ times left (or right) and crosses two vertices with identical labels, and therefore identical subtrees. Then, as shown above, we can change the color of one vertex in G while maintaining correct coloring, and is it easy to see that the arrows in the sequence will flip. Therefore:

Theorem 4. *Changing the color of a vertex in G is equivalent to flipping a sequence of $k + 1$ equally directed arrows in the arrow representation of the coloring.*

Proof: To sum up, every correct coloring of G can be represented by a sequence of arrows, and we are allowed to flip $k + 1$ consecutive arrows pointing in the same direction in a single step of changing the color of one vertex of G. Recoloring G from

Figure 4: Swapping one coloring of k-path ...

one coloring to another is therefore equivalent to changing one sequence of arrows into another one, using only the described operation. Q.E.D.

Figure 5: ... to another coloring.

Assume for a while that we can find a sequence of $k+1$ consecutive arrows with the same orientation somewhere in the sequence, somewhere after the one we want to flip. Let's take the first such sequence, and flip it according to Theorem 4. Because it was the first such sequence, the arrow preceding it must have pointed in the other direction. Therefore, after the flipping, its direction will be the same as direction of the arrows in the flipped sequence. Now, we can repeat the procedure, starting with the first arrow of the newly-created sequence. Every operation happens closer to the one arrow we want to change, because there is always at least one arrow added to the beginning of the sequence being flipped. We can continue in this manner until we finally flip the arrow we were considering. At most, we do $\mathcal{O}(n)$ such flips, as with each one we move at least one arrow (i.e. one vertex) closer.

When we flip that one arrow, we can add it to the beginning which is already correctly oriented, and proceed to change the next one, if it does not yet have the desired orientation. The following figure shows the procedure to flip one arrow (fourth in the row) for $k = 2$. The arrows being flipped are underlined.

$$
\begin{array}{l|l}
\Rightarrow \Rightarrow \Leftarrow & \Leftarrow \Leftarrow \Rightarrow \Leftarrow \Rightarrow \Rightarrow \underline{\Leftarrow \Leftarrow \Leftarrow} \Rightarrow \Leftarrow \\
\Rightarrow \Rightarrow \Leftarrow & \Leftarrow \Leftarrow \Rightarrow \Leftarrow \underline{\Rightarrow \Rightarrow \Rightarrow} \Rightarrow \Rightarrow \Rightarrow \Leftarrow \\
\Rightarrow \Rightarrow \Leftarrow & \Leftarrow \Leftarrow \Rightarrow \underline{\Leftarrow \Leftarrow \Leftarrow} \Leftarrow \Rightarrow \Rightarrow \Rightarrow \Leftarrow \\
\Rightarrow \Rightarrow \Leftarrow & \Leftarrow \Leftarrow \underline{\Rightarrow \Rightarrow \Rightarrow} \Rightarrow \Leftarrow \Rightarrow \Rightarrow \Rightarrow \Leftarrow \\
\Rightarrow \Rightarrow \Leftarrow & \underline{\Leftarrow \Leftarrow \Leftarrow} \Leftarrow \Leftarrow \Rightarrow \Leftarrow \Rightarrow \Rightarrow \Rightarrow \Leftarrow
\end{array}
$$

$$
\Rightarrow \Rightarrow \Leftarrow \Rightarrow \ \Big| \ \Rightarrow \Rightarrow \Leftarrow \Leftarrow \Rightarrow \Leftarrow \Rightarrow \Rightarrow \Rightarrow \Leftarrow
$$

We can now repeat the process for the next arrow in the sequence, if it is oriented in the wrong direction. In a similar manner we can turn each arrow in $\mathcal{O}(n)$ steps, and therefore recolor the whole graph in $\mathcal{O}\left(n^2\right)$ steps.

4.3 Edge conditions

So far, we were always considering vertices far enough from either end of the path. That is, for every vertex being recolored we assumed existence of at least k vertices in both direction on the path to guarantee an unique way of recoloring. But for ends of path we can imagine some "virtual" vertices there. They will have fixed color which we need for both colorings of G.

We also assumed that in the sequence arrows there are always $k + 1$ consecutive arrows. If there is no such subsequence we can easily create it at the end of path. Or we can use our imagination of virtual vertices, their arrows can be oriented any way we need, so we have always $k + 1$ consecutive arrows.

5 Recoloring k-trees

We will now extend this method to k-trees, because it is almost the same. A *(complete) k-tree* is built similarly to a k-path, from a tree graph (or 1-tree) by adding edges from

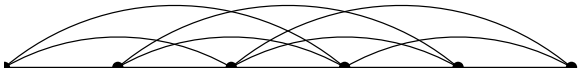

Figure 6: 3-tree

each vertex to its parent, parent of its parent, parent of parent of its parent, etc. up to k predecessors.

But if we look at k-tree, it is just a few k-paths connected. Important here is that no two k-paths influence recoloring of each other. These two k-path have some vertices in common, but they also have to be colored same and some part is different, but we can color each alone as normal k-path. We will just flip arrows in each k-path and then we can flip arrows in common vertices.

Therefore, again, we can flip one arrow (or change the color of one vertex, considering colors of its ancestors fixed) in $\mathcal{O}(n)$ time, as in the worst case we will have to flip every sequence of $k+1$ consecutive arrows in every branch once, this corresponds to changing the color of every vertex. If we begin at the root of the tree and progress towards the leaves, for example using breadth-first method, we will use $\mathcal{O}(n)$ operations for every vertex, and therefore $\mathcal{O}\left(n^2\right)$ steps to recolor the whole k-tree.

6 Partial k-trees

For partial k-trees is situation little more complicated. A *partial (or incomplete) k-tree* is recording to his name a connected subgraph of k-tree.

The most notable difference between partial and complete k-trees is same as for every graph G and his subgraph G'. Any coloring of G is coloring of G', but it does not work opposite way. There can exist a coloring of G' that is not correct coloring in G (because some vertices with same color are here connected, but in G' not).

The recoloring method for partial k-trees will not be as straightforward as for complete k-trees. Consider the obvious fact that if we can change a coloring G_f into $G_{f'}$ in c steps, we can go opposite way and change $G_{f'}$ into G_f in c steps too. Then, if we want to change the coloring of G'_f into another coloring G'_g, we will do it in four steps:

- Find a recoloring of G'_f to any correct coloring of the complete k-tree $G_{f'}$.
- Find a recoloring of G'_g to any correct coloring of $G_{g'}$.
- As above, find a recoloring of $G_{f'}$ to $G_{g'}$.
- Now recolor in this way: $G'_f \to G_{f'} \to G_{g'} \to G'_g$.

As every correct coloring of G is also a correct coloring of G', recoloring of $G(f') \to G(g')$ obtained in step three will also be valid in G'.

6.1 Recoloring a partial k-tree into a coloring of a complete k-tree

The last remaining question is how to find a recoloring of $G'(f)$ into some coloring $G(f')$ that would be correct even for a complete k-tree.

Consider changes in the coloring tree $T(G')$ resulting from omitting an edge in G. If we follow the creation of a coloring tree, a vertex which had one or more edge removed will now have more colors it will be able to take, as it has less neighbors restricting certain colors. Therefore, $T(G')$ will no longer be a binary tree, it can contain several vertices with more than two children.

Figure 7: Vertex with more than two children.

Now, every such vertex has two children which are there also in the recoloring tree $T(G)$ of a complete k-tree G. If the current recoloring path is passing through one of them, we can simply forget all the other children as the color of the current vertex in G' is also correct in G. In the other case, we can always pick the child the coloring path is passing through and one of the children in $T(G)$ and forget about the rest of them. This will make the current vertex in $T(G')$ have only two children. If we pick this vertex so that all of its descendants already form a correct coloring tree of a complete k-tree, we can see that it will also form a correct coloring tree if we consider only the two selected children and their subtrees. We can now find a way to switch the first arrow on the path, making the recoloring path go through the child which is also in $T(G)$, therefore fixing the vertex. This can be done in $\mathcal{O}(n)$ steps, as it is the operation of flipping one arrow in a complete k-tree.

Figure 8: We have to consider just two children.

This way, we can always find one such vertex of $T(G')$ from which the path goes to a vertex not in $T(G)$ and fix the vertex so that the path passes only through $T(G)$. Notice also that this will always keep the changing part of the path in $T(G)$, as the operations being executed always work only in $T(G)$.

As we can fix every vertex guiding the path outside of $T(G)$ in $\mathcal{O}(n)$ steps, we can fix the whole path in $T(G')$ to contain only vertices of $T(G)$ in $\mathcal{O}\left(n^2\right)$ steps.

Theorem 5. *Given any two colorings of a partial k-tree G', G'_f and G'_g, we can find a recoloring transforming G'_f into G'_g in $\mathcal{O}\left(n^2\right)$ steps.*

Consider the order of operations given in the list above. We have just shown that the first two will take $\mathcal{O}\left(n^2\right)$ steps, and in sections 4 and 5 we have shown that the third one will also take $\mathcal{O}\left(n^2\right)$ steps. As we can forget multiplicative constants, the whole recoloring process will also take $\mathcal{O}\left(n^2\right)$ steps.

7 Worst case

Now we will present a k-path (which is indeed also a k-tree) that requires $\Omega(n^2)$ steps as the worst case.

`Worst case will be added later.`

8 Planar graphs

Planar graphs are 5-degenerate. We will show you proof of following theorem.

Theorem 6. *There exists a triangulation T for each planar graph G, such that G is subgraph of T and each coloring in G is valid for T.*

Proof: We would like to add one vertex to each face and join him with all vertices of the face to create a triangulation. But because of this we can block some coloring. If face has more than 6 edges and want to be colored by 7 different colors, we can't add new vertex inside it. But we can place inside more than one vertex. We add vertex v and join him only to 6 vertices of face. There is always 7th color for v. We can create smaller face, so we apply same algorithm till we create triangulation of face.

At figure below in this coloring vertex v can have color C.

Figure 9: Planar graph – triangulation of big face.