Efficiently Computing Vertex Arboricity of Planar Graphs *

Abhik Roychoudhury *Susmita Sur-Kolay* †

Abstract

Acyclic-coloring of a graph $G = (V, E)$ is a partitioning of V, such that the induced subgraph of each partition is acyclic. The minimum number of such partitions of V is defined as the vertex arboricity of G. An $O(n)$ algorithm ($n = |V|$) for acyclic-coloring of planar graphs with 3 colors is presented. Next, an $O(n^2)$ heuristic is proposed which produces a valid acyclic-2-coloring of a planar graph, if one exists (since there are planar graphs with arboricity 3). We also prove that our heuristic is guaranteed to produce a valid acyclic-2-coloring for all outerplanar graphs (which are of arboricity 2) in $O(n)$ time. Finally, some experimental results for our acyclic 3-coloring and 2-coloring algorithms, along with future directions are presented.

Keywords : Vertex arboricity, Planar graph, Graph coloring,
Testing of sequential circuits.

Contact author : Abhik Roychoudhury
E-mail : abhik@cs.sunysb.edu
Phone : (516) 632-8470
Fax : (516) 632-8334

*A preliminary version of this paper was published in "Foundations of Software Technology and Theoretical Computer Science" (FST & TCS), 1995.

†The first author is currently at : Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794, USA; E-mail : abhik@cs.sunysb.edu. The second author's address is : Department of Computer Science and Engg, Jadavpur University, Calcutta 700032, India. E-mail : elcv9602@isical.ernet.in, dgd@jadav.ernet.in

1 Introduction

In this paper, the problem of determining the vertex arboricity of finite planar graphs efficiently is studied [12]. Vertex arboricity of a graph is the minimum number of vertex-disjoint forests into which the graph can be decomposed. All vertices of a forest can be assigned the same color, thus producing no monochromatic cycles in the graph. This problem is also equivalent to a generalized vertex coloring problem [2] and will be henceforth referred to as *acyclic-coloring* [7].

Formally, let $G = (V, E)$ be a finite undirected (directed) graph. An acyclic-k-coloring $color : V \to \{1, 2, \ldots, k\}$ is a partitioning of V into $\{V_1, V_2, \ldots, V_k\}$ such that for all i, the subgraph G_{V_i} induced by V_i is a forest. Any vertex $v \in V_i$ is assigned color i, i.e. $color(v) := i$. The minimum value of k for which such partitioning exists, is called the *vertex arboricity* ρ of G.

The immediate motivation of this problem comes from the domain of *design for testability* in VLSI circuits [1]. A clocked sequential circuit is represented by a graph G whose vertices correspond to the flip-flops of the circuit and an $< i, j >$ edge implies the existence of a combinational path from flip-flop i to flip-flop j. The partitions obtained by acyclic-coloring of such a graph correspond to the independent clocks of the circuit. Each independent clock corresponds to a *test mode* of the circuit. In any particular test mode, the flip-flops in its own partition change states while the other flip-flops in the circuit maintain their states. An effective acyclic-coloring algorithm can minimize the number of partitions, and hence the number of test modes.

The acyclic-k-coloring problem for general digraphs is NP-complete [6]. A depth-first-search based simple heuristic for acyclic-coloring any digraph was suggested in [1]. Nevertheless, the fact that $\rho \leq 3$ for any planar graph [3, 7], leads us to the following pertinent question: is the acyclic-coloring problem polynomial-time solvable for planar graphs? Determining the vertex arboricity of maximal planar graphs is known to be NP-Hard [8]. This would effectively mean that determining whether $\rho = 2$ for a planar graph is, in general, not polynomial time solvable. In this paper, we give an algorithm to compute an acyclic coloring of planar graphs using 3 colors. We also suggest an efficient (and sound) heuristic to compute the acyclic coloring with 2 colors. Parallel algorithms to obtain acyclic coloring of planar graphs were reported independently in [4, 5].

A simple linear time algorithm for acyclic-coloring of planar graphs using 3 colors (along with a proof of correctness) is presented in Section 2. Section 3 provides a polynomial-time heuristic to compute the acyclic-2-coloring of a planar graph. The method discussed is sound but incomplete, and a proof of soundness is also furnished. Moreover, we formally prove that our heuristic can always produce a valid acyclic-2-coloring for certain classes of planar graphs (namely outerplanar graphs) in linear time. Experimental results for both the algorithms appear in Section 4 along with concluding remarks.

For graph-theoretic terminologies used without definition in this paper, the reader is referred to [9]. The induced subgraph formed by the neighbors of a vertex v is called the *Neighbor Induced Subgraph (NIG)* of v. With respect to a given acyclic-coloring, the number of vertices in $NIG(v)$ which have been colored i, is denoted by $N_i(v)$. A path or a cycle is said to be *i-monochromatic* if all the vertices on it are assigned the color i.

2 Acyclic-3-Coloring Algorithm for Planar Graphs

Since all planar graphs have $\rho \leq 3$ [3, 7], an acyclic-3-coloring exists always. In fact, it was proven in [7] that there exists an acyclic-3-coloring of any planar graph where each partition induces a linear forest. But no efficient algorithm follows from these results.

Our algorithm for acyclic-3-coloring a planar graph is a recursive one, similar to that for vertex-coloring [10]. In each recursive call, a vertex v of degree at most 5 (always exists in a planar graph) is deleted, until a sufficiently small graph G_s with at most 5 vertices is obtained. While deleting the vertex v, the vertices of $NIG(v)$ which have not yet been deleted, are stored.

The vertices of G_s are colored by procedure $N\ddot{a}ive\_Color\_3$ as follows: (a) If $\Delta(G_s)$ is 0 or 1, all its vertices are assigned color 1. (b) Otherwise, it is shown below that a vertex $u \in G_s$ exists such that $2 \leq \deg(u) \leq 3$. Assign color 1 to u, color 2 to two neighbors of u and color 3 to all other vertices in G_s.

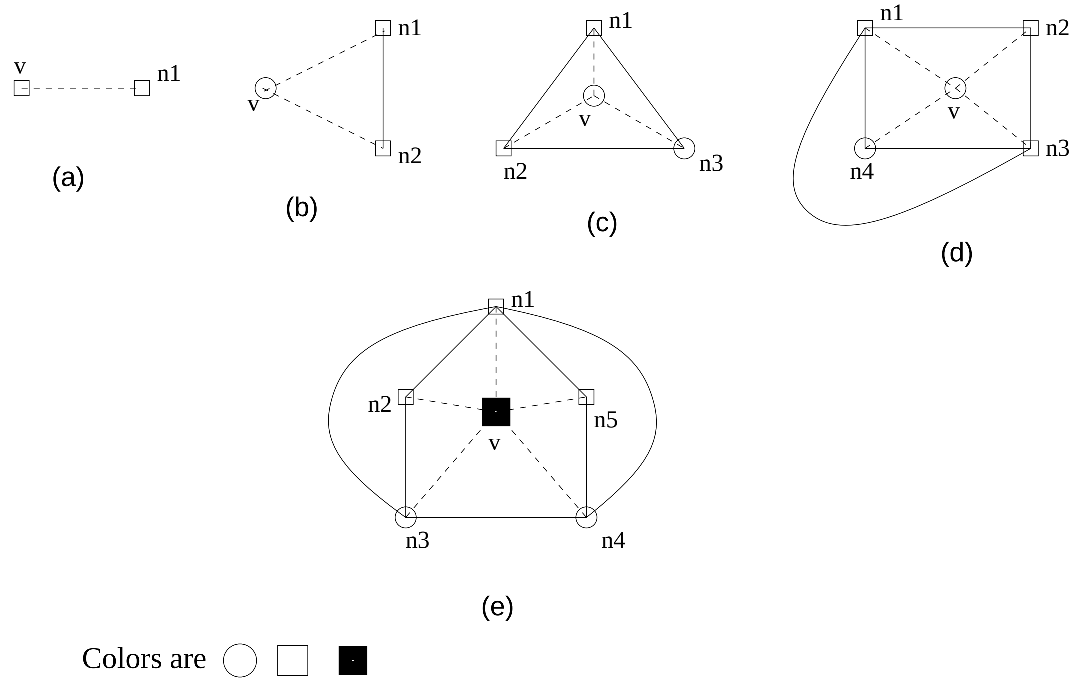

Figure 1: Five cases of NIGs for acyclic 3-coloring

The recursive calls are completed by coloring the deleted vertex v in each call. The color i is given to v (for $i \in \{1, 2, 3\}$) such that there is at most one vertex in $NIG(v)$ with color i. Figs. 1a–1e show five NIGs and the corresponding acyclic-colorings; all possible other NIGs are subgraphs of one of these five. Since a vertex of degree at most 5 is always deleted, it is clear that $|V_{NIG}| \leq 5$. The complete algorithm is presented below in pseudocode.

procedure Acyclic_3_Color (G);

```
begin
    if |V| ≤ 5 then
        Näive_Color_3(G)
    else begin
        Find a vertex v in G with degree ≤ 5;
        Keep track of the neighbors of v;
        Acyclic_3_Color(G − v);
        Color v by considering the color of its neighbors;
    end;
```

end;

Lemma 1 *Algorithm Näive_Color_3 outputs an acyclic-3-coloring for any planar graph G_s with 5 or fewer vertices in constant time.*

Proof: If all vertices of G_s have degree 0 or 1, then G_s has no cycles. In the second case, if the existence of u is established, then the Lemma is proven since the color assignment does not produce any monochromatic cycles. Suppose contrary to the hypothesis, all vertices of G_s have degree 4 then G_s must have 5 vertices. But a planar graph of 5 vertices can have a maximum of 9 edges whereas the sum of the degrees of all vertices is 20, hence violating the Handshaking Lemma [9] for any graph. Therefore, vertex u exists. $\square$

Theorem 1 *Algorithm Acyclic_3_Color produces an acyclic-3-coloring for any planar graph with n vertices in $O(n)$ time.*

Proof: The correctness of the algorithm is established from the following observations:
i) Since 3 colors are available, by pigeon-hole-principle there exists no coloring of the vertices of $NIG(v)$ for which : $|N_i(v)| \geq 2, i = 1,2,3$. Thus, if $|N_j(v)| < 2$ for some $j \in \{1,2,3\}$, then v can be assigned color j to complete that recursive call and no monochromatic cycles are introduced.
ii) The given algorithm halts. In each recursive call, a vertex is removed from the graph and ultimately when the number of vertices is 5 or less, by Lemma 1 *Näive_Color_3* terminates correctly.

Regarding the time complexity, each recursive call can be performed in constant time and there are $O(n)$ recursive calls. Step 4 is accomplished by always choosing the minimum degree vertex. Before invoking *Acyclic_3_Color*, the vertices are kept sorted by bucket-sort in order of their degrees. Deletion of a vertex then causes minor adjustments. Step 5 takes constant time since $NIG(v)$ has at most 5 vertices. Step 7 involves computing $N_i(v)$ by counting the number of neighbors with color i $(i = 1,2,3)$ where $V_{NIG(v)}| \leq 5$. $\square$

3 Acyclic-2-Coloring for Planar Graphs

An example of a planar graph with $\rho = 3$ is the geometric dual of the Tutte graph [3]. Determination of ρ of planar graphs is equivalent to testing whether an acyclic-2-coloring exists and producing it. If the arboricity is 3 then the algorithm reports failure in yielding a valid acyclic-2-coloring. Moreover, for outerplanar graphs (which always have $\rho = 2$), the algorithm never reports failure in constructing an acyclic-2-coloring.

3.1 Overview

The proposed algorithm colors the vertices of a planar graph G in a step-by-step fashion, coloring one vertex at each recursive call. At each stage, it is ensured that there exists no monochromatic cycle among the vertices already colored. A list of monochromatic paths for each color, obtained so far, is also stored. Initially, the two lists of monochromatic paths are empty. Since any cycle in G is entirely within a biconnected component (block) of G, the first step comprises in dividing G into its blocks. Next, the blocks of G are colored one by one in such a sequence that the cutpoints can be colored consistently. Each block is colored recursively by finding a vertex v of minimum degree which is certainly at most 5, keeping track of its not yet deleted neighbors, and deleting v. This continues until a sufficiently small graph G_s of at most 5 vertices is obtained, which is then colored naively. A recursive call is completed with coloring the deleted vertex by considering the colors of its neighbors and the set of monochromatic paths produced so far. This may induce limited backtracking if monochromatic cycles are obtained. The algorithm halts if elimination of these monochromatic cycles fails.

Näive_Color_2 is very similar to *Näive_Color_3* excepting in the case where there is a vertex u with degree 2 or 3. Here u is colored 1, two of its neighbors are assigned color 2 and all other vertices of G_s are given color 1.

It is to be noted that if the input graph G is divided into blocks $B_1, B_2, \ldots, B_n$, then while B_k is being colored, after the deletion of a finite number of vertices from B_k, the remaining graph could turn out to have cutpoints. The next recursive call will then again divide this graph into blocks $B_{k,1}, B_{k,2}, \ldots, B_{k,m}$. Then in our algorithm, coloring of $B_{k,i}, \{i = 1, 2, .., m\}$ are completed and thus the coloring of B_k is obtained. Only then can the coloring of B_{k+1} proceed.

The following sections discuss this algorithm in details. Section 3.2 gives a safe ordering of blocks for consistent coloring of cutpoints. The various cases of $NIG(v)$ during backtracking and the respective methods for elimination of monochromatic cycles appear in Section 3.3. The criterion for reporting failure, i.e., inability to compute an acyclic-2-coloring is given in Section 3.4. An idea about storing monochromatic paths and updating schemes is given in Section 3.5. Section 3.6 deals with the correctness proof and complexity analysis of the algorithm. Finally section 3.7 provides a formal proof that the algorithm will always successfully produce an acyclic-2-coloring for outerplanar graphs.

The algorithm is presented below in pseudo code.

BEGIN
  Read(G); % the input graph is read %
  Set list of monochromatic paths to ϕ*;*
  Acyclic_2_Color(G);
END.

procedure Acyclic_2_Color(G);
  begin
1. *Divide G into blocks;*
2. *Find a sequence σ in which the blocks are to be colored;*
3. *For each block $B_i \in \sigma$ do*
  begin
4. *Keep track of any already colored cutpoint c of B_i;*
5. *if $|V_{B_i}| \leq 5$ then*
6. *Näive_Color_2*(B_i)
7. *else begin*
8. *Find a vertex v in B_i of minimum degree; % deg(v) $\leq$ 5%*
9. *Keep track of the neighbors of v;*
10. *Acyclic_2_Color($B_i - v$);*
11. *Color v by considering the color of its neighbors;*
12. *Eliminate monochromatic cycles by*
  first invoking "Toggle_Iterate" and
  then invoking "Break_Cut_Cycle";
13. *if failure occurs in Step 12 then report and exit;*

14. *Update the lists of monochromatic paths;*
end;
15. *if color(c) is different now, then toggle all vertices on B_i;*
end;
end;

3.2 Safe Ordering of Blocks

Our strategy is to obtain acyclic 2-coloring for the blocks separately. However, the cutpoints need to be consistently colored. An illustration of inconsistent coloring of cutpoints for the graph in Fig. 2a is given in Fig. 2b where a cutpoint may have been assigned different colors when the two blocks sharing it were colored separately. Changing the color of the cutpoint in any of these two blocks results in a monochromatic cycle. In order to ensure a consistent coloring the blocks should be colored in a particular sequence which is called a *safe ordering*. By this ordering, at the start of coloring any block, at most one cutpoint is previously colored. It is shown next that such an ordering always exists.

Lemma 2 *There exists a safe ordering of blocks of a graph such that when the coloring of the vertices of a particular block commences, there exists at most one vertex in that component which is already colored.*

Proof: Let us define an ordering σ as follows and then show that it is safe. For a graph G, its block-cutpoint tree [9] $BC(G)$ (Fig. 2c) has vertices corresponding to the blocks and the cutpoints of G and an undirected edge (c, B) exists if cutpoint c is on block B. Initially, the sequence σ is empty. A pruning step on $BC(G)$ involves (i) removing all its pendant vertices (these correspond to blocks of G only) from $BC(G)$, (ii) adding the list of these blocks to the beginning of σ, and (iii) removing the pendant vertices of the resultant graph (these correspond to cutpoints of G only). This pruning step is repeated until the resultant graph is empty or has one vertex which again corresponds to the central block of G. This block is added to the beginning of σ.

Since $BC(G)$ is a tree, it has no cycles and if the central block is treated as its root, any other block has a unique parent which is pruned after it and thus colored before it according to σ (Fig. 2d). Hence, when coloring of any block begins, only the cutpoint shared with its unique parent is previously colored. $\square$

3.3 Completion of Recursive Call by Coloring a Vertex

Before invoking the recursive call in step 10 of our acyclic-2-coloring algorithm, the NIG of the deleted vertex v is stored in Step 9 because on returning from the recursive call, the color of v in Step 11 depends on $NIG(v)$ and its acyclic 2-coloring (Fig.3), so that no monochromatic cycles are introduced.

If $N_k(v) < 2$ for some k, then $color(v) := k$ (Figs. 3a and 3b). If for both colors $k = 1, 2$, there are two vertices n_{i_k} and n_{j_k} in $NIG(v)$ which have been colored k but there is no k-monochromatic path in $G - v$ between n_{i_k} and n_{j_k} for at least some k, then $color(v) := k$.

In the remaining case, for both k there is *exactly one* k-monochromatic path of length 1 (i.e., direct edge) or more between n_{i_k} and n_{j_k} due to the assumption that the graph $G - v$ already has a valid acyclic-2-coloring when v is being colored. There is no option but to backtrack and $NIG(v)$ is referred to as the *backtracking NIG*. The major intricacy of our algorithm lies in effective

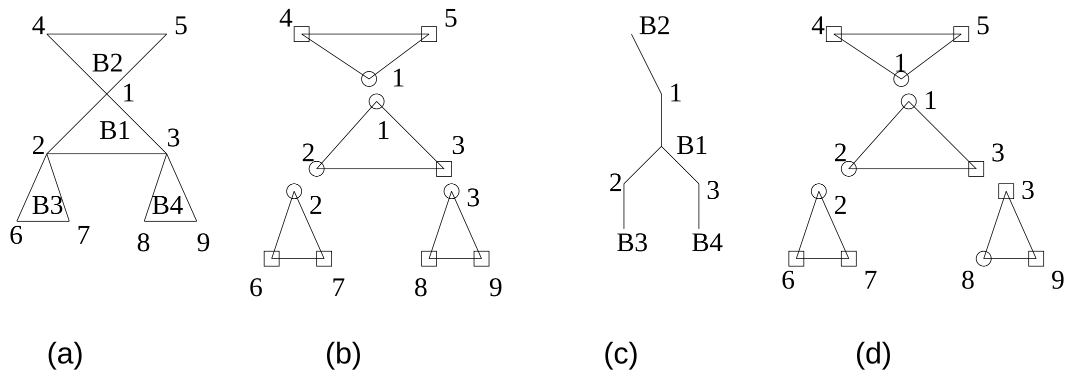

Figure 2: (a) Separable graph (b) Inconsistent coloring (c) Block-cutpoint tree (d) Consistent coloring

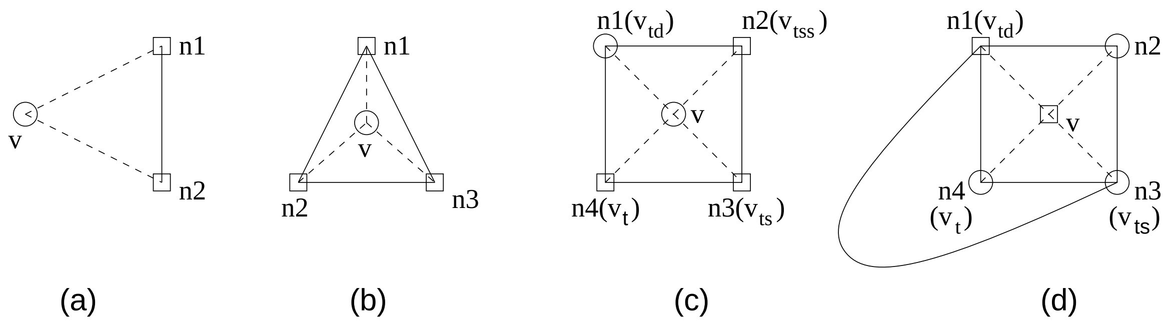

Figure 3: NIGs for acyclic-2-coloring (a)K_2 (b) K_3 (c) C_4 (d)$K_4 - x$

backtracking. The operation starts with setting $color(v)$ to $color(n_i)$ where n_i is in $NIG(v)$, and changing $color(n_i)$ to $3 - color(n_i)$; this is called *toggling* the color of n_i. The choice of n_i depends on the backtracking $NIG(v)$. But toggling of n_i may give rise to monochromatic cycles now. Monochromatic cycles obtained while backtracking can be classified as follows:

Definition 1 *A* **block cycle** *of a toggled vertex n_i in $NIG(v)$ is a monochromatic cycle containing n_i and at least one other vertex in $NIG(v)$.*

Definition 2 *A* **cut cycle** *of a toggled vertex n_i in $NIG(v)$ is a monochromatic cycle containing n_i and no other vertex in $NIG(v)$.*

Definition 3 *A* **cut-chain cycle** *of a toggled vertex n_i in $NIG(v)$ is a monochromatic cut cycle which is formed by breaking a cut cycle of n_i and contains none of the vertices in $NIG(v)$.*

If all block and cut cycles are eliminated efficiently without re-toggling any vertex in G, then backtracking ends and coloring of v is complete. Detection of failure necessitates maintaining a list of vertices toggled upto now. The salient cases of backtracking $NIG(v)$ are taken up below to discuss the choice of n_i to be toggled and subsequent elimination of monochromatic cycles.

Case 1: $|V_{NIG}| = 4$.
Here $N(c_1) = N(c_2) = 2$ and color of any vertex of minimum degree in $NIG(v)$ can be toggled. If there are more than one (at most two) choices and backtracking from one fails, then backtracking from the other is started. Two major subcases for backtracking are illustrated in Figs. 3c & 3d

after the color of vertex n_4 in $NIG(v)$ has been toggled. The method for eliminating block cycles and cut cycles in each case are described below. Let us assume without loss of generality that $color(n_4)$ is 2 after toggling. The following notations are used:
v_t: the toggled vertex, (n_4),
v_{td} : the differently colored neighbor of v_t, (n_1),
v_{ts} : the similarly colored neighbor of v_t, (n_3),
v_{tss} : the other neighbor of v_{ts} and v_{td} colored similarly to v_t, (n_2),
$C_i(v_t)$: the i-th monochromatic cycle through v_t,
$v_i(v_t)$: the neighbors of v_t on a cycle $C_i(v_t)$, excepting v_{ts} if $C_i(v_t)$ is a block cycle of v_t in which case it contains the edge (v_t, v_{ts});
togg_set : set of vertices whose colors were toggled earlier.

Case 1.1 : $NIG = K_4 - x$.

(a) Elimination of block cycles (Fig. 4): The key idea behind eliminating 2-colored block cycles due to toggling of v_t is to toggle the colors of $v_i(v_t)$ for all $i = \{1, 2, \ldots, p\}$ to color 1. It is to be observed that G being planar, for any plane embedding of it, only one of these, say $v_j(v_t)$ — either the first or the last one in clockwise order in the embedding of G — can create one or more 1-colored cut cycles through v_{td}. Once again, planarity guarantees that only u_l and u_r, the first and the last neighbor of $v_j(v_t)$ in clockwise order, may form 2-colored cut cycles through v_t or v_{ts}. In the next iteration, all the neighbors of $v_j(v_t)$ excepting one of u_l and u_r (any one if neither has been toggled before) are toggled to color 2. However, if either of these two distinct vertices has a direct edge to v_t or v_{ts}, as the case may be, then all neighbors of $v_j(v_t)$ on 1-colored paths to v_{td} have to be toggled. Ultimately, if no more monochromatic cycles are created, the procedure halts successfully. It exits unsuccessfully if at any stage, an already toggled vertex must be toggled for elimination of a monochromatic cycle. These ideas could be presented in a semi-formal manner as follows.

Algorithm Toggle_Iterate

(Eliminates block cycles due to toggling of $color(v_t)$)

Input : $G, v_t, v_{ts}, v_{td}, NIG(v)$, monochromatic paths formed by S, the set of already colored vertices, togg_set.

Output : A 2-coloring of $S \cup v$, possibly including cut-cycles, and the set of toggled vertices.

Technique:

```
T1. If v_t has one or more color(v_t) monochromatic paths to v_ts then
        begin
T2.             for all paths do toggle color(v_i);
T3.             u_l := first just toggled neighbor of v_t in clockwise order;
                u_r := last just toggled neighbor of v_t in clockwise order;
                    ( in the embedding of G )
        end
T4. for other_node := {v_t, v_ts} do
        begin
                check_node :=v_td ; clr := 3 - color(v_t);
T5.             while ( either u_l or u_r has two or more clr-paths to check_node) do
                    %these are cut cycles%
                    begin
T6.                     u := vertex having two or more clr-paths to check_node;
T7.                     U := neighbors of u on clr-paths to check_node
                            in clockwise order ;
T8.                     u_l := first element of U; u_r := last element of U;
T9.                     if check_node ∈ U then toggle all vertices in U - check_node
                                % toggling check_node is avoided%
T10.                        else toggle all vertices in U except either u_l or u_r %not both%;
T11.                    Swap check_node and other_node; clr := 3 - clr;
                    end
        end
```

Lemma 3 **Toggle_Iterate** *runs in* $O(n)$ *time when a vertex* v *in a graph of* n *vertices with* $NIG(v) = K_4 - x$ *is to be acyclic-2-colored and breaks all block cycles if it does not have to re-toggle.*

Proof: In the recursive algorithm for acyclic-2-coloring, there are no monochromatic cycles before v_t is toggled. Any block cycle resulting from toggling of v_t in $NIG = K_4 - x$, must have v_t and v_{ts} on it. All such block cycles are eliminated in Step T2. The remaining steps are for eliminating cut cycles arising due to toggling in Step T2. This procedure stops when either no more new cut cycles are created or some vertex has to be re-toggled.

In a planar graph, the number of distinct paths between two vertices is $O(n)$. So Step 2 may need $O(n)$ time. There are two iterations of the *for* loop in Step T5. Each iteration involves a *while* loop but since this inner loop toggles already colored vertices, it can do so at most $O(n)$ times over all its iterations. Hence, the total time complexity in the worst case is linear in n, the number of vertices in G. $\square$

(b) Elimination of cut cycles: Fig. 5a gives a glimpse of the possible cut-cycles created due to toggling $color(v_t)$. It should be noted that many of the cut cycles get eliminated in the process

of eliminating block-cycles, e.g. if in Fig. 5a, the path p_1 (marked in bold) is 2-monochromatic, then the cut cycle $(n_4, v_i, \ldots, v_j, n_4)$ gets broken as well. $(v_i, \ldots, v_j)$ implies the existence of a monochromatic path between v_i and v_j) But the ones in Fig. 5b need separate treatment because cut-chain cycles appear.

Algorithm Break_Cut_Cycle;

Inputs : $G, v_t, v_{ts}, v_{td}, NIG(v)$, monochromatic paths formed by S, the set of vertices colored so far, toggset;

Output : Valid acyclic-2-coloring of $S \cup \{v\}$

Technique :

B1. just_toggled := v_t; clr := color(v_t); check_node := v_{td}; other_node := v_{ts};

B2. for all $v_i \in$ just_toggled do

begin

B3. U_{v_i} := list of neighbors of v_i having color clr and clr-colored paths among them, in clockwise order;

B4. if $\forall i, U_{v_i} = \phi$, then return; %no more cut or cut-chain cycles%

B5. else toggle all u_i except either the first or last in U_{v_i};
% Taking caution not to re-toggle any vertex %

end;

B6. just_toggled := $\bigcup_k U_{v_k}$;

B7. if there is a cut cycle through any u_i and check_node then toggle neighbors of u_i as in Steps T7 to T11 in Toggle_Iterate;

B8. Swap check_node and other_node; clr := 3 - clr;

B9. Go to Step B2 for checking any further cut-chain-cycles.

end;

Lemma 4 Break_Cut_Cycle *runs in $O(n)$ time when a vertex v in a graph of n vertices with $NIG(v) = K_4 - x$ is to be acyclic-2-colored and breaks all cut- cycles and cut-chain-cycles which are not broken by Toggle_Iterate if it does have to re-toggle any vertex.*

Proof: Similar arguments as in Lemma 3. □

Case 1.2 : $NIG = C_4$ or its subgraphs.

(a) Elimination of block-cycles: The technique adopted is similar to *Toggle_Iterate* with the addition that 2-monochromatic paths between neighbor of a just toggled vertex to both v_{ts} and v_{tss} are to be detected [11]. Thus the for loop in Step T5 is for v_t, v_{ts}, v_{tss}. However, if a 2-monochromatic cycle containing v_{tss} is found at any stage, then in the subsequent iterations it is not necessary to search for cycles containing v_{ts} because of planarity.

(b) Elimination of cut-cycles: The technique is similar to the algorithm *Break_Cut_Cycle*.

Case 2: $|V_{NIG}| = 5$ (Fig. 6). Here $N_i(v) \geq 2 : i = 1, 2$. For each color, a monochromatic path with endpoints e_1 and e_2 where $e_1, e_2 \in V_{NIG}$ is tested for. If such paths exist for both the colors, then obviously backtracking is needed. The criteria for selecting the vertex $n_i \in V_{NIG}$ to be toggled differs from case to case, and is included in the discussion for each individual case. The elimination of cut-cycles for this case is similar to the algorithm *Break_Cut_Cycle* discussed earlier. Therefore only the block cycles need to be discussed. The different NIGs in this case are as follows.

Case 2.1: $NIG = K_1 + (C_4 - x)$. (Fig. 6a)

Here, the vertex $n_i \in V_{NIG}$ to be toggled must satisfy :

a) $\deg(n_i) = 2$ in $NIG(v)$;

b) the neighbors of n_i are *differently* colored;
c) $N_{color(n_i)} = 2$.
If there is no vertex satisfying all three conditions enlisted above, then it has to be checked whether the degree 4 vertex (n_1 in Fig.6a) has at least two same colored neighbors. If so, then n_1 is toggled otherwise any vertex satisfying conditions (a) and (b) but not (c) is toggled. The elimination of block-cycles here is exactly similar to the case $NIG = K_4 - x$.

Case 2.2 : $NIG = (K_1 + (C_4 - x)) - x$. (Fig. 6b)
The three conditions for selecting of $n_i \in V_{NIG}$ to be toggled for proper coloring of v in case 2.1 above are also applicable here. But if no vertex satisfying all three conditions exists, then n_i is a vertex meeting conditions (a), (b) and also the condition:
c') the two neighbors of n_i have no edge between them.

Possible subcases :

Case 2.2.1 : There is the edge (v_{ts}, v_{td}) (Fig. 7a).
Elimination of block cycles is done by algorithm *Toggle_Iterate* discussed earlier in Case 1.1.

Case 2.2.2 : No edge between v_{ts} and v_{td} (Fig. 7b).
Some modifications in *Toggle_Iterate* algorithm are required to tackle the situation. These are same as those mentioned in Case 1.2.

Case 2.3 : $NIG = C_5$ or its subgraphs. (Fig. 6c)
This is the most difficult situation with largest number of possibilities. To simplify the situation, the conditions for choosing a vertex $n_i \in V_{NIG}$ whose color is to be toggled are the following:
i) $\deg(n_i) = 2$;
ii) neighbors of n_i are *similarly* colored;
iii) $N_{color(n_i)} = 3$.
It can be easily seen that in the C_5 backtracking NIG, such a vertex n_i will always exist. This choice gives us an additional advantage as well in shortening the monochromatic paths obtained after coloring of v. The elimination of block cycles is done by making a few additions to the procedure *Toggle_Iterate* as follows [11]. This procedure is invoked twice, once with v_{td1} as check_node and then with v_{td2}. The basic strategy is similar to that for Case 1.1 but here monochromatic paths between the toggled vertex v_t and both of its neighbors v_{ts1} and v_{ts2} are relevant for creation of block cycle. If in this process, there is a vertex u which has edges to both v_{ts1} and v_{ts2} and is toggled to $color(v_t)$, then vertex v_{ts2} (v_{ts1}) is toggled where check_node is v_{td1} (v_{td2}) and invoke Toggle_Iterate with $v_t := v_{ts2}$ (v_{ts1}); $v_{td} := u$; $v_{ts} := v_{td1}$ (v_{td2}).

3.4 Monochromatic Path Management

The lists of monochromatic paths formed by vertices colored so far are consulted during the vital operation of backtracking. Here the storage and updation of monochromatic paths are discussed briefly. This covers Step 14 of our acyclic-2-coloring algorithm.

For both the colors, a set S of existing monochromatic paths for which the following invariant holds, is stored: For any path $p_i \in S$, there exists no $p_j \in S$, $(j \neq i)$, such that $p_i \subseteq p_j$.

Without loss of generality, let us consider the updation of 1-monochromatic paths. First, the 1-monochromatic paths are scanned for vertices that have just been toggled from 1 to 2. As and when these are found, the corresponding 1-monochromatic paths are broken. Let this set of broken paths be denoted as P_1. Then, the different paths in the induced subgraph formed by set of vertices just toggled from 2 to 1(say $S_{2,1}$) are built. Let this set of paths be called $P_{2,1}$. This is followed by merging of P_1 with $P_{2,1}$. Finally, if v has been colored 1, necessary changes are made in the 1-monochromatic paths.

3.5 Correctness and Time Complexity

Theorem 2 *Algorithm* **Acyclic-2-Color** *reports failure if $\rho(G) > 2$. Also, any coloring produced by the algorithm is valid and is produced in $O(n^2)$ time.*

Proof: The proof is based on the following facts.
(a) The 2-acyclic-coloring algorithm halts.
In each recursive call, a vertex is deleted from a biconnected component of the input graph. Ultimately, when the number of vertices ≤ 5, $N\ddot{a}ive\_Color\_2$ is invoked. Then, there are two options : i) the algorithm reports failure while trying to color a vertex and thereby complete a recursive call; ii) the recursive calls are completed for each block, after which the algorithm terminates successfully.
In either case, the 2-coloring algorithm halts.
(b) Algorithm $N\ddot{a}ive\_Color\_2$ introduces no monochromatic cycles.
Let G_s be the small graph with which $N\ddot{a}ive\_Color\_2$ is invoked. Then, the only possibility of monochromatic cycles exists when for a color, say 1, N_1 is 3. If v_{start} is the first vertex in G_s to be colored, then $color(v_{start}) := 1$. Since, two of the neighbors of v_{start} are colored with 2, v_{start} can have at most one neighbor with color 1. Hence, no cycles are introduced.
(c) Given the coloring of the neighbors of a deleted vertex v, if no failure is reported, the coloring of v introduces no monochromatic cycles.
This follows from Lemmata 3 and 4.
(d) Given a valid coloring of the biconnected components, a valid coloring of the entire graph can be obtained.
This is based on the existence of safe ordering in Lemma 2.
(e) If the input graph is not 2-colorable, then a failure is reported.
From facts (a), (b), (c) and (d) above, it follows that if no failure is reported, the algorithm produces a valid 2-coloring. But if no failure is reported for an acyclic 3-colorable graph, then one of the two partitions produced by the algorithm must include a monochromatic cycle, i.e the coloring must be invalid. The proof follows by contradiction.
(f) Any coloring produced by the algorithm is done in $O(n^2)$ time (n = Number of vertices in input graph)
There are $O(n)$ recursive calls in Acyclic-2-color. In each call, step 1 which basically performs depth-first search on a planar graph, takes $O(n)$ time. Step 8 is also linear time in the worst case and so are steps 12, 14 and 15. The remaining steps require constant time. The overall time complexity is thus $O(n^2)$.

Observation (e) proves that Acyclic-2-color reports failure if $\rho(G) > 2$. Observations (a), (b), (c) and (d) prove that any 2-coloring produced by Acyclic-2-color is valid. Observation (f) proves that any coloring produced is accomplished in $O(n^2)$ time. This completes the proof. □

3.6 Acyclic-2-coloring for outerplanar graphs

Our acyclic-2-coloring algorithm always successfully produces a valid 2-coloring for a special class of graphs, namely *outerplanar graphs*. In fact, we prove that for this class of planar graphs, our algorithm will never backtrack, *i.e.* the color assigned to a vertex will never be toggled.

A graph is outerplanar if and only if it has no subgraph *homeomorphic*[1] to K_4 or $K_{2,3}$ except

[1] Two graphs are homeomorphic if both can be obtained from the same graph by a sequence of subdivisions of lines [9]

$K_4 - x$ [9]. Note that, therefore we produce valid 2-colorings for all K_4- and $K_{2,3}$-free graphs as well, as in [4].

Theorem 3 *Algorithm Acyclic_2_color(G) will always produce a valid 2 coloring for G, if G is outerplanar.*

Proof : If the coloring of all the blocks are correct, the coloring of the graph is produced correctly by Acyclic_2_color (Lemma 2). Also, a graph G is outerplanar iff all its blocks are outerplanar [9]. Hence, it is sufficient to show that while coloring a vertex v in an outerplanar block B in step 11 of Acyclic_2_color, the color of none of the neighbors of v is toggled. In that case, Acyclic_2_color will never invoke "Toggle_Iterate" or "Break_Cut_Cycle" while coloring an outerplanar graph G.

Now, since we are considering an outerplanar block B, clearly

1. $\forall u \in V(B)\ degree(u) \geq 2$ (B is a block)
2. $\exists v \in V(B)$ s.t. $degree(v) = 2$. (This is because B is outerplanar. Even maximal outerplanar graphs are guaranteed to have at least 2 vertices of degree 2 [9].)

Therefore, the minimum degree vertex v chosen in step 8 of Acyclic_2_color is always of degree exactly 2 (for any invocation of Acyclic_2_color). Hence, when we color v at each recursive call, by looking at its neighbors, we have exactly 2 neighbors to consider,- a case which does not lead to any toggling of assigned colorings. However, the only case when Acyclic_2_color(G) can report failure even when $\rho(G) = 2$ is when in the process of coloring G, it attempts to toggle the color of a vertex whose color assignment was previously toggled. Since for outerplanar graphs, color assignments are never toggled, such a situation can never arise. Also, we know from theorem 2 that any coloring produced by Acyclic_2_color is valid. This completes the proof.

□

Complexity Analysis :
A minor modification of Acyclic_2_color(G) can ensure that the average case complexity of 2-coloring of outerplanar graphs is $O(n)$ where $n = |V(G)|$. This can be explained as follows. In general, there are $O(n)$ recursive calls. In each recursive call, step 1 (dividing G into blocks), step 8 (obtaining the minimum degree vertex), step 12 (elimination of monochromatic cycles created due to toggling of color assignments), step 14 (updating the list of monochromatic paths) and step 15 (toggling the color of all the vertices of a block so as to obtain consistent coloring of a cutpoint) take $O(n)$ time (refer Theorem 2 part(f)) for general planar graphs.

Now, while coloring an outerplanar graph G, we will require a depth-first search to find the blocks of G in the first recursive call. However, in the subsequent recursive calls, we are always coloring an outerplanar block. To find out the number of biconnected components obtained by the deletion of a degree 2 vertex from an outerplanar block (recall that we always delete a vertex of degree 2 at every recursive call), we observe that any outerplanar block B is hamiltonian, *i.e.* all vertices of B lie on a cycle. Let these vertices be $\{v_1, v_2, ..., v_n\}$. Without loss of generality, assume that v_1 is of degree 2 with neighbors v_2 and v_n, and $B \not\equiv C_n$. Now, we can split $V(B - v_1)$ into the following subsets :
$U_1 = \{v_2, v_3, ..., v_k\}$, $U_2 = \{v_m, v_{m+1}, ..., v_n\}$ and $U_3 = V(B - v_1) - U_1 - U_2$
where $k \geq 2$ is the largest possible integer such that the induced subgraph of U_1 in $B - v_1$ is a chain. Similarly, $m \leq n$ is the smallest possible integer, such that the induced subgraph of U_2 in $B - v_1$ is a chain. The induced subgraph of U_3 is then a block. The induced subgraphs of U_1 and U_2 are clearly free from cycles. Hence they can be colored in $O(|U_1| + |U_2|)$ time and we can avoid issuing any further recursive calls to perform this coloring. Moreover, in the average case,

$|U_1| = O(1)$ and $|U_2| = O(1)$, Hence, unless $B \equiv C_k$ (*i.e.* the outerplanar block concerned is a cycle of k vertices, for some k), the identification of any new block resulting from the deletion of a minimum degree vertex from an outerplanar block can be performed in $O(1)$ time. If we have $B \equiv C_k$, then $|U_1| \equiv |U_2|$, and we can color $B - v_1$ in $O(k)$ *i.e.* $O(|V(B - v_1)|)$ time. Hence, if G is outerplanar, the separation of G into its blocks in step 1 through a depth-first search is required only in the first recursive call and not subsequently.

Obtaining the minimum degree vertex (step 8) can be performed in $O(1)$ time by keeping the vertices of G sorted (on the basis of their degrees) prior to computing Acyclic_2_color(G) (as was done in Acyclic_3_color). Since color assignments are never toggled, therefore step 12 is never executed in Acyclic_2_color(G) when G is outerplanar. Moreover, the list of monochromatic paths can also be updated in $O(1)$ time since we always color a vertex of degree 2 in each recursive call. Finally, step 15 never needs to be invoked except in the first recursive call. Hence, if G is outerplanar, for each recursive call issued in computing *Acyclic_2_color*(G) the following property holds :

- Either, it takes $O(1)$ time (average-case) before it issues the subsequent recursive call, and it also takes $O(1)$ time after returning from that recursive call
- Or, it is of the form *Acyclic_2_color*(B), and it is computed in $O(|V(B)|)$ without issuing any further recursive calls

Hence, the average-case complexity for acyclic-2-coloring an outerplanar graph G by our method is $O(n)$, where $n = |V(G)|$.

4 Experimental Results and Future Work

Some of the results obtained by applying the algorithms on various types of planar graphs are shown in Fig. 8, 9 and 10. The algorithm Acyclic_2_Color reported failure on the geometric dual of the Tutte graph, which is known to have $\rho = 3$.

The acyclic-2-coloring algorithm can be improved by using a more efficient data structure for storing monochromatic paths and reducing the overhead of re-determining blocks after a vertex is deleted. There is scope for parallelizing the algorithm since in the safe ordering of blocks, all blocks which are siblings (having same parent) in the block-cutpoint tree can be processed independently.

The algorithms described in this paper evidently work for planar digraphs as well. It may be worth investigating the class of planar graphs (digraphs) without any cycles (directed) of length 3. The ultimate goal is however to obtain the acyclic coloring of general digraphs with colors less than the known bound of $1 + \lfloor \frac{max(\delta(G'))}{2} \rfloor$ where the maximum is taken over all induced subgraphs G' of G.

References

[1] V.D. Agrawal, S. Seth, and J.S. Deogun. Design for testability and test generation with 2 clocks. *Univ. of Nebraska-Lincoln, Dept. of Comp. Sc. Engg, Report series #102*, 1990.

[2] I. Broere and C.M. Mynhardt. Generalized colorings of planar and outerplanar graphs. *Y. Alavi et al. eds., Graph Theory with Applications to Algorithms and Computer Science(Wiley, New York)*, pages 151–161, 1985.

[3] G. Chartrand and H.V. Kronk. The point arboricity of planar graphs. *Journal of London Math. Soc.*, 44(4), 1969.

[4] Zhi-Zhong Chen and Xin He. NC algorithms for partitioning planar graphs into induced forests and approximating NP-hard problems. *Proc. 21st Intl. Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Vol 1017*, 1995.

[5] Zhi-Zhong Chen and Xin He. Parallel complexity of partitioning planar graphs into induced forests. *Discrete Applied Mathematics*, 69, 1996.

[6] M. Garey and D.S. Johnson. *Computers and Intractability: A complete guide to NP-completeness*. Freeman, 1979.

[7] W. Goddard. Acyclic colorings of planar graphs. *Discrete Mathematics 91*, pages 91–94, 1991.

[8] S. L. Hakimi and E. F. Schmeichel. A note on the vertex arboricity of a graph. *SIAM J. on Discrete Mathematics*, 2(1), 1989.

[9] F. Harary. *Graph theory*. Addison Wesley Publishing Co., 1969.

[10] T. Nishizeki and N. Chiba. Planar graphs: theory and algorithms. *North Holland Mathematics Studies 140*, 1988.

[11] A. Roychoudhury. Cycle-coloring of planar and general graphs with applications to VLSI testing. *Project Report, Dept. of Computer Sc. Engg., Jadavpur University, India*, 1995.

[12] A. Roychoudhury and S. Sur-Kolay. Efficient algorithms for vertex arboricity of planar graphs. *Proceedings of FST&TCS 1995, Lecture Notes in Computer Science, Vol 1026*, pages 37–51, 1995.

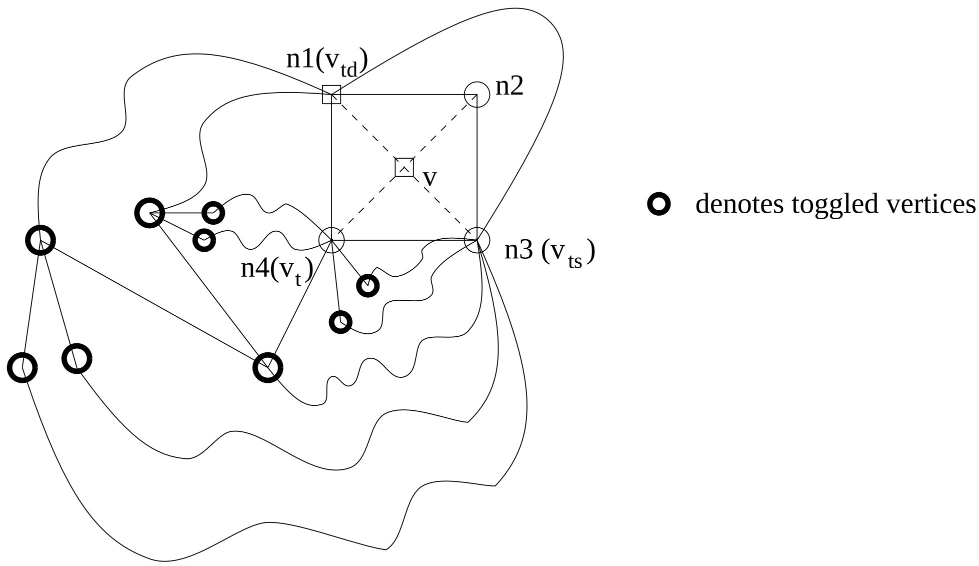

Figure 4: Elimination of block cycles

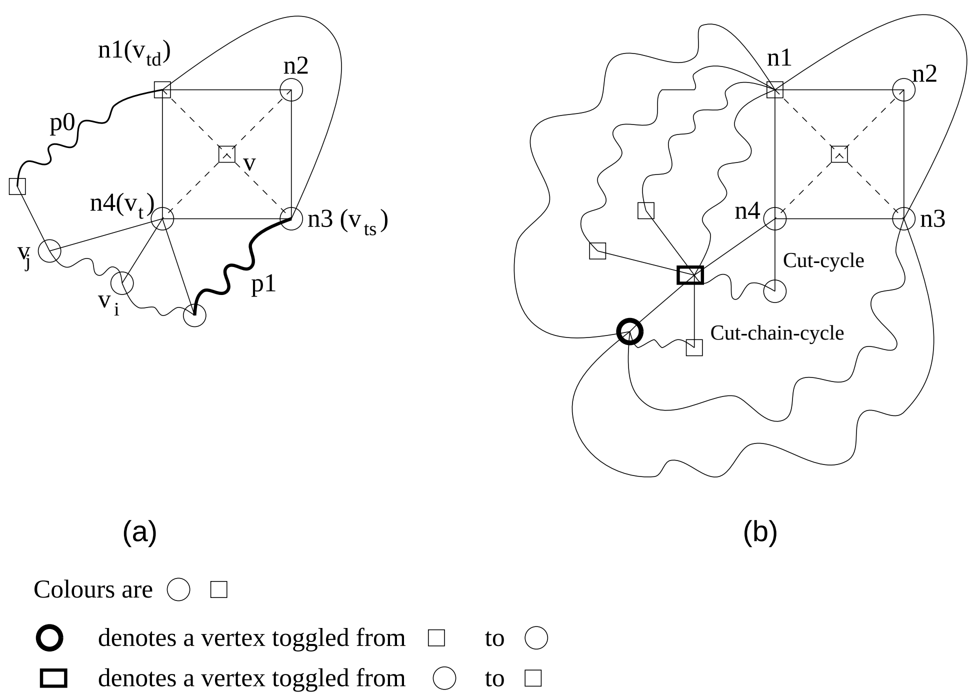

Figure 5: Elimination of (a) cut-cycles (b) cut-chain-cycles

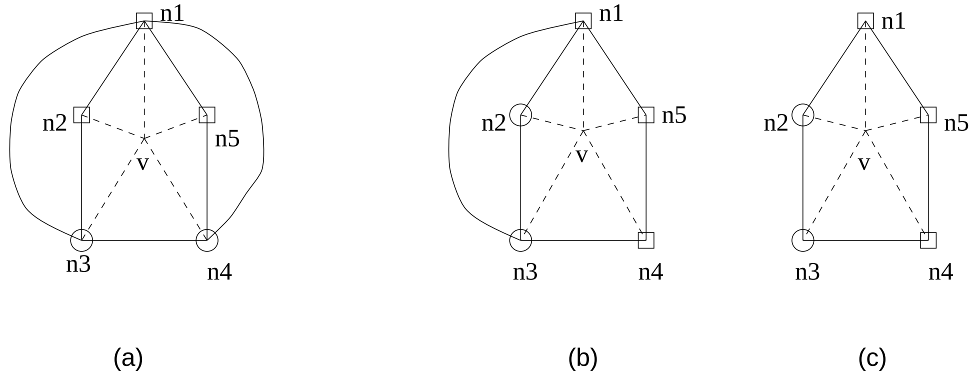

Figure 6: NIGs for acyclic-2-coloring with 5 vertices

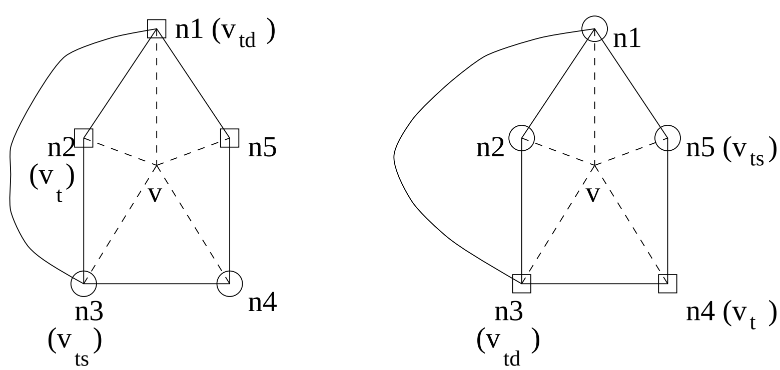

Figure 7: Two subcases of Fig. 6b (before toggling)

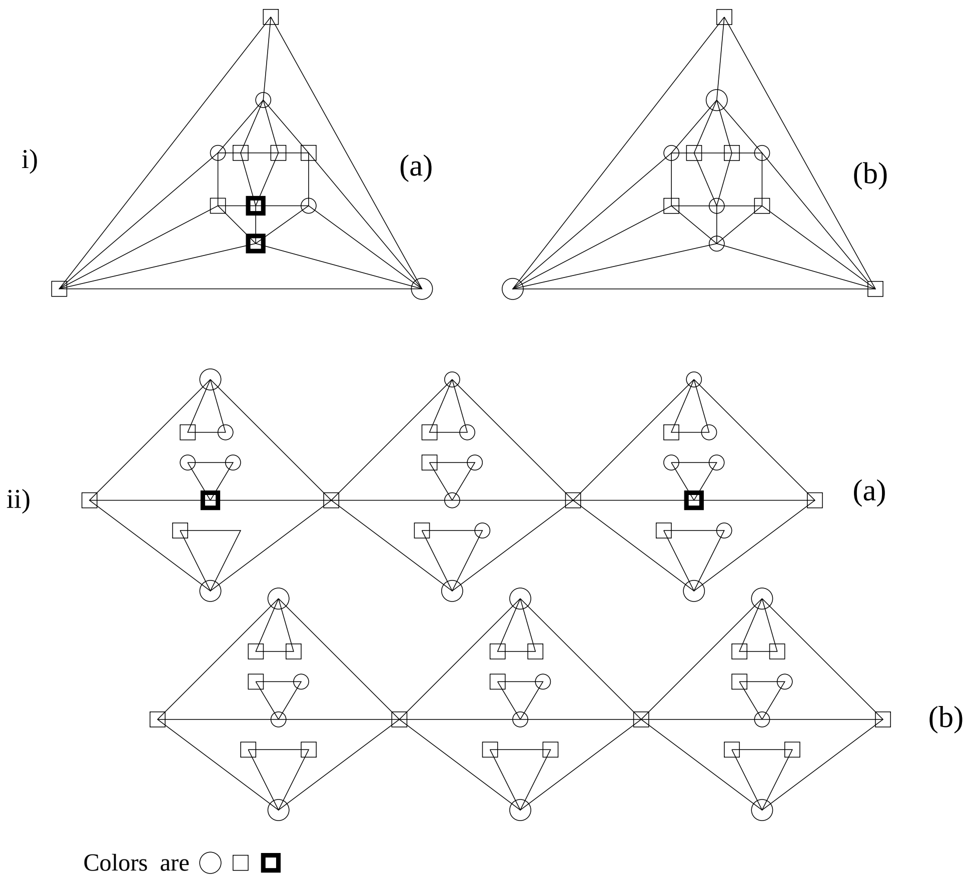

Figure 8: Experimental results with (a) 3 colors (b) 2 colors on : i) Polyhedron ii) Separable graph

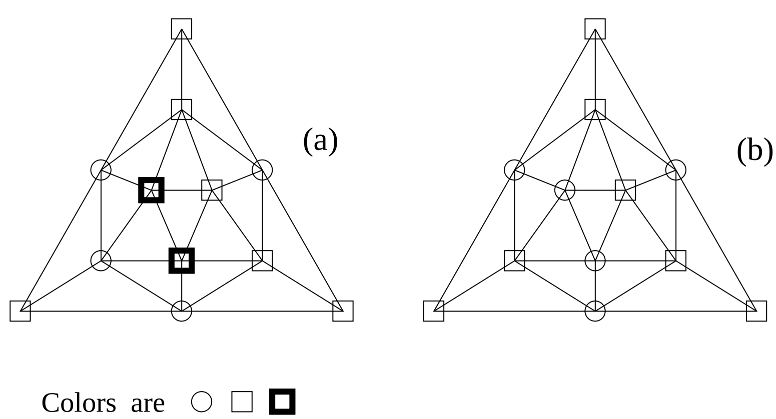

Figure 9: Experimental results with (a) 3 colors (b) 2 colors on a maximal planar graph

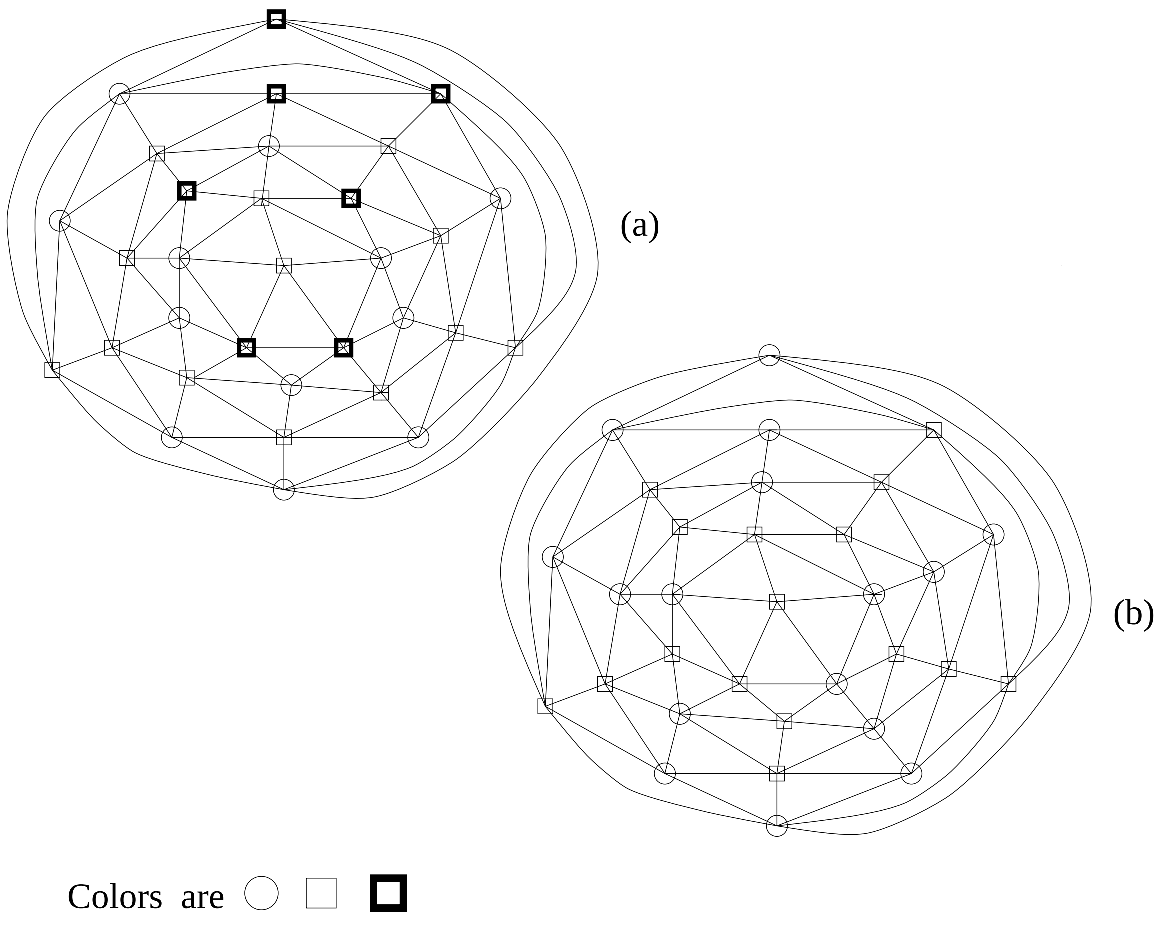

Figure 10: Results for acyclic coloring of a 5 connected planar graph: (a) 3 colors (b) 2 colors