International Journal of Foundations of Computer Science

Coloring Algorithms on Subcubic Graphs*

SAN SKULRATTANAKULCHAI
and
HAROLD N. GABOW
Department of Computer Science, University of Colorado at Boulder, Boulder CO 80309-0430 USA
{skulratt,hal}@cs.colorado.edu

ABSTRACT

We present efficient algorithms for three coloring problems on subcubic graphs. (A *subcubic graph* has maximum degree at most three.) The first algorithm is for 4-edge coloring, or more generally, 4-list-edge coloring. Our algorithm runs in linear time, and appears to be simpler than previous ones. The second algorithm is the first randomized EREW PRAM algorithm for the same problem. It uses $O(n/\log n)$ processors and runs in $O(\log n)$ time with high probability, where n is the number of vertices of the graph. The third algorithm is the first linear-time algorithm to 5-total-color subcubic graphs. The fourth algorithm generalizes this to get the first linear-time algorithm to 5-list-total-color subcubic graphs. Our sequential algorithms are based on a method of ordering the vertices and edges by traversing a spanning tree of a graph in a bottom-up fashion. Our parallel algorithm is based on a simple decomposition principle for subcubic graphs.

Keywords: List Edge Coloring, Total Coloring, List Total Coloring, Subcubic Graphs

1. Introduction

We present efficient algorithms for three coloring problems on subcubic graphs. (A subcubic graph has maximum degree at most three.) The problems we attack are by now well-studied generalizations of standard vertex and edge coloring [19]. The first algorithm is for 4-edge coloring, or more generally, 4-list-edge coloring. Our algorithm runs in linear time, and appears to be simpler than previous ones. The second algorithm is the first randomized EREW PRAM algorithm for the same problem. It uses $O(n/\log n)$ processors and runs in $O(\log n)$ time with high probability, where n is the number of vertices of the graph. The third algorithm is the first linear-time algorithm to 5-total-color subcubic graphs. The fourth algorithm generalizes this to the first linear-time algorithm to 5-list-total-color subcubic graphs. Our sequential algorithms are based on a method of ordering the vertices and edges by traversing a spanning tree of a graph in a bottom-up fashion. Our parallel algorithm is based on a simple decomposition principle for subcubic graphs. We now discuss these problems and previous work in detail.

*A preliminary version of this paper appeared in *Proceedings of the 8th Annual International Computing & Combinatorics Conference.*

1.1. Background

All *graphs* treated in this paper can have parallel edges. We use the term *simple graphs* to denote the ones without parallel edges. A graph is *cubic* if every vertex has degree three. A graph is *subcubic* if none of its vertices has degree more than three [22]. Subcubic graphs are also called *at most cubic* in the literature [16]. We prefer the former term since it reminds us of the fact that a subcubic graph is always an induced subgraph of some cubic graph. (And furthermore, a simple subcubic graph is an induced subgraph of some simple cubic graph.) Subcubic graphs are interesting and important theoretically and in practice [16, 8, 3] for several reasons. First, graphs of maximum degree three are often the borderline cases between the hard and easy problems. Most **NP**-hard problems and some **APX**-hard problems do not become easier even when restricted to such graphs [15, 16, 1, 24, 25], but become polynomial-time solvable for graphs of smaller maximum degree [15, 16]. Second, several graph problems admit linear-time transformations from a general graph to a cubic or subcubic graph such that a solution to the transformed graph yields a solution to the general graph. Examples are the famous Four Color Problem [34] and the Maximum Matching Problem [2]. Third, some real-world applications are problems on subcubic graphs [3, 8]. There are a number of reasons why subcubic graphs are good models for these real-world problems, but in the area of computational geometry it is often due to the fact that the dual of a plane triangulation is cubic.

Reference [38] introduces a method for decomposing subcubic graphs that seems to be well-suited for solving the edge coloring problem. We show in a preliminary version of this paper [13] that the method is also useful for designing both sequential and parallel algorithms for the list-edge and list-total coloring problems. In this paper we introduce a method of ordering the vertices and edges by traversing a spanning tree in a bottom-up fashion. This method, called the B-ordering method, conceptually simplifies our sequential algorithms previously presented in [13]. It is also used in [37] to obtain linear-time algorithms for acyclic coloring problems.

1.2. Graph Coloring

We follow the terminology of [5]. Let $G = (V, E)$ be a loopless (multi)graph having n vertices and m edges. A *total coloring* is a map $\varphi : V \cup E \to \mathbb{N}$ satisfying (i) no adjacent vertices or edges have the same image, and (ii) the image of each vertex is distinct from the images of edges incident with it. To *k-total-color* G is to find a total coloring whose image is included in $[k] = \{1, 2, \ldots, k\}$. The graph is *$k$-total-colorable* when such a map exists. The *total chromatic number* χ'' is the least k for which G is k-total-colorable.

Let X stand for either V, E, or $V \cup E$; and let $\lambda : X \to 2^{\mathbb{N}}$ be an assignment of lists of colors to elements of X. A *λ-coloring on X* is a map $\varphi : X \to \mathbb{N}$ such that $\varphi(x) \in \lambda(x)$ for all $x \in X$, and $\varphi(x_1) = \varphi(x_2)$ implies x_1 is neither adjacent nor incident to x_2. A *λ-vertex-coloring* is a λ-coloring on V. A *λ-edge-coloring* is a λ-coloring on E. A *λ-total-coloring* is a λ-coloring on $V \cup E$.

A graph is *k-choosable* if there exists a λ-vertex-coloring for any λ satisfying $|\lambda(v)| = k$ for all $v \in V$. The notions of *k-edge-choosability* and *k-total-choosability* are defined similarly. The *choice number* (or *list chromatic number*) χ_ℓ is the least k for which G is k-choosable. The *list chromatic index* χ'_ℓ is the least k for which G is k-edge-choosable. The *total choosability* χ''_ℓ is the least k for which G is k-total-choosable.

1.3. List Edge Coloring

By Vizing's Theorem [42] the chromatic index χ' of any simple graph with maximum degree Δ is either Δ or $\Delta + 1$. A simple graph can be edge-colored using $\Delta + 1$ colors in $O(m\sqrt{n \log n})$ time [12]. However, Holyer [18] shows that deciding whether the chromatic index of a given simple cubic graph equals 3 or 4 is **NP**-complete [15]. Reference [38] gives an algorithm to 4-edge-color any (not necessarily simple) subcubic graph in $O(n)$ time.

The **List Edge Coloring Conjecture** (LECC) states that the list chromatic index χ'_ℓ of a graph equals its chromatic index χ' (see [4, 9]). Graphs that are known to satisfy the LECC are the bipartite graphs [14], certain families of 1-factorable graphs [10], line-perfect graphs [31], multicircuits [44], simple series-parallel graphs [23], and simple outerplanar graphs [23]. Clearly $\chi'_\ell \geq \chi'$. By Vizing's Theorem [42], a simple graph would satisfy $\chi'_\ell \leq \Delta + 1$ if the LECC were true. Even this upper bound has not been established for all simple graphs.

The authors of reference [22] show subcubic graphs satisfy this upper bound, i.e., they are 4-edge-choosable. They study subcubic graphs with *halfedges*, i.e., edges with only one endpoint. They obtain their result through case-by-case analysis of 4-edge-coloring of paths, cycles with halfedges, and some special types of graphs; reduction of the input graph to a specific form; and coloring procedures that avoid known obstructions. Even though the algorithm derived from their proof has linear time bound, it appears to be too complicated for practical use.

Independently, Vizing [43] and Erdős *et al.* [11], prove the following list version of Brooks' theorem [7]: *the choice number of any connected simple graph that is neither complete nor an odd cycle does not exceed its maximum degree.* Reference [39] gives an $O(m + n)$-time algorithm to Δ-list-vertex-color any graph that satisfies the hypotheses of the above theorem. Suppose G is subcubic. By reducing the problem of list-edge-coloring G to that of list-vertex-coloring its line graph and using the above theorem we see that G is 4-edge-choosable. Thus G can be 4-list-edge-colored in $O(n)$ time by executing the algorithm of [39] on its line graph.

We will present a direct, simple, $O(n)$-time sequential algorithm, and the first randomized $O(n)$-work, $O(\log n)$-time with high probability, EREW PRAM algorithm to 4-list-edge-color subcubic graphs.

1.4. Total Coloring

The **Total Coloring Conjecture** (TCC) states that the total chromatic number χ'' of a simple graph is at most $\Delta + 2$. Clearly $\chi'' \geq \Delta + 1$. However, deciding

whether or not a given simple cubic, bipartite graph satisfies $\chi'' = \Delta + 1$ is **NP**-complete [36, 35]. In fact, the problem of determining the total chromatic number of a k-regular bipartite graph is **NP**-hard, for each fixed $k \geq 3$ [30]. The TCC has been shown to hold for some families of graphs [45]. Rosenfeld [33] shows it holds for subcubic graphs. Vijayaditya [41] shows it holds for simple subcubic graphs. The proof of [33] yields a super-linear time algorithm because it has to find a shortest cycle in each recursive step. The algorithm in [41] requires a routine to find a perfect matching in a bridgeless, cubic graph. The current best known algorithm [3] to find such a matching runs in $O(n \log^4 n)$ time.

We will present the first $O(n)$-time algorithm to 5-total-color subcubic graphs. Our algorithm has nothing in common with [33] or [41]. Moreover, it works on any graphs, not necessarily simple ones.

1.5. List Total Coloring

The **List Total Coloring Conjecture** (LTCC) states that the total choosability χ''_ℓ of any graph equals its total chromatic number χ'' [21, 6]. Clearly $\chi''_\ell \geq \chi''$. The LTCC is wide open. Graphs that are known to satisfy the LTCC are the outerplanar graphs [20] and the multicircuits [26, 27]. A simple graph would satisfy $\chi''_\ell \leq \Delta + 2$ if the TCC and the LTCC were both true. This upper bound has not been established for all simple graphs.

The authors of reference [21] show subcubic graphs satisfy this upper bound, i.e., they are 5-total-choosable. They claim their proof gives a polynomial-time algorithm, without specifying the degree of the polynomial. Their algorithm has super-linear running time because it has to find a shortest cycle in each recursive step.

We will present the first $O(n)$-time algorithm to 5-list-total-color subcubic graphs. It does not use the result of [21]. Our 5-total coloring algorithm is in fact a special case of our 5-list-total coloring algorithm. We present it separately and before the list version because it is simpler.

1.6. Organization

This paper is organized as follows. The rest of this section gives notations and defines terms to be used in later sections. Section 2 presents our high level principles: the B-ordering method, the subcubic graph decomposition theorem, and some list vertex and list edge coloring lemmas of cycles. The results therein are used in Sections 3 & 4. Section 3 concerns the list edge coloring algorithms. Section 4 concerns the total and list total coloring algorithms. Section 5 concludes our paper.

Definitions. A *triple bond* is a graph consisting of two vertices and three parallel edges. A *cycle* is a connected graph every vertex of which has degree two. It is *even* if it has an even number of vertices, and *odd* otherwise. Note that a cycle can have length two. Let C be a cycle in G. A *chord* is an edge of G joining two vertices of C but is itself not an edge of C. An edge e is a *pendant edge* of cycle C if exactly one of e's endpoints is on C. A *k-cycle* is a cycle on k vertices. We write

a labeled k-cycle as either $\langle v_1, v_2, \ldots, v_k, v_1\rangle$ or $\langle v_1, e_1, \ldots, v_k, e_k, v_1\rangle$, depending, respectively, on whether we are interested in the vertices v_i only, or in both the vertices v_i and the edges e_i.

The word *"(color) list"* as used in the list coloring problems actually means "(color) set." We use the terms *neighbor* and *available color* at several places in this paper. Let x be either a vertex or an edge and let $\lambda(x)$ be its list of colors. By a *neighbor* of x we mean any vertex/edge adjacent/incident to x. During the execution of an algorithm, a color $\alpha \in \lambda(x)$ is *available for* x if no neighbor of x has yet been assigned color α by the algorithm; it is *unavailable* otherwise.

2. Basic Techniques

2.1. B-ordering

It seems Lovász [28, 29] originated the idea of solving a coloring problem by cleverly imposing a coloring order on the graph elements to be colored. He used it to prove Brooks' theorem [7] on vertex coloring:*the chromatic number of any connected simple graph that is neither complete nor an odd cycle does not exceed its maximum degree.* Our B-ordering method is a generalization of Lovász's to both edge coloring and total coloring.

Consider a connected graph $G = (V, E)$. Let T be a spanning tree of G with root x. A *B-ordering of $V \cup E$ with respect to x* is an ordering of the vertices and edges as $y_1, y_2, \ldots, y_{m+n}$ such that the nontree edges come last and the remaining order is gotten by traversing T bottom up, visiting each tree edge right after its child vertex, and visiting each vertex right after having visited all its child edges. (In all other respects the ordering is arbitrary.)

Lemma 1 *Consider a connected (multi)graph G. Let $y_1, y_2, \ldots, y_{m+n}$ be a B-ordering with respect to a vertex x. The following statements hold.*

(1) *y_1 is the vertex x.*

(2) *y_2 is an edge incident with x.*

(3) *If $k > 2$, then any vertex or edge y_k has a neighboring vertex y_i with $i < k$ and a neighboring edge y_j with $j < k$.*

A B-ordering of G can be obtained in $O(m + n)$ time.

Proof. Statements (1) & (2) clearly hold. Statement (3) holds if y_k is a vertex since every vertex other than the root has a parent vertex and a parent edge. It holds if y_k is an edge for the same reason unless y_k is incident to the root. In that case use (2) to get y_j. A spanning tree of G can be found by any standard search technique in $O(m + n)$ time. □

Property (3) of Lemma 1 is what makes B-ordering useful. Consider the problem of 4-list edge coloring. We are given a subcubic graph G whose every vertex has its own list of four colors, and we are to list-edge-color G. Assume without loss of generality that G is connected. Any edge of G has at most 4 neighboring edges. Suppose we "greedily" color the edges in decreasing B-order, i.e., we assign to each edge, whenever possible, any color available for it. Property (3) guarantees that all

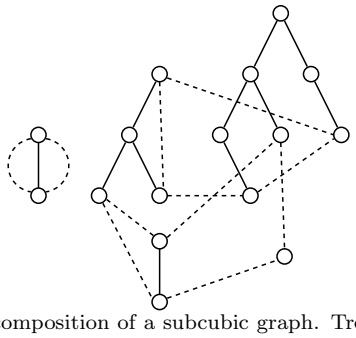

Fig. 1. Example decomposition of a subcubic graph. Tree edges are rendered solid. Cycle edges are rendered dashed. The 5-cycle has a chord. By making the chord a cycle edge and changing the two leftmost cycle edges into tree edges, we get a chordless decomposition on the larger connected component of the graph.

edges except possibly the very first one (in B-order) can be colored! In fact, this idea alone already gives us an $O(n)$-time 4-list edge coloring algorithm if G is not cubic: *simply B-order the edges with respect to some vertex x with $d(x) < 3$, and greedily color them in backward B-order.* Greedy coloring works correctly on such graphs because the very first edge in B-order has at most three edge neighbors, so, at all times, it has at most three colored neighbors. In Section 3 we will augment this greedy approach to get a full 4-list edge coloring algorithm.

Now consider the problem of 5-list total coloring. We are given a subcubic graph G whose every vertex and every edge has its own list of five colors, and we are to list-total-color G. Assume without loss of generality that G is connected. Any vertex of G has at most 3 neighboring vertices and at most 3 neighboring edges. Any edge of G has exactly 2 neighboring vertices and at most 4 neighboring edges. Suppose we "greedily" color the vertices and edges in decreasing B-order, i.e., we assign to each vertex/edge, whenever possible, any color available for it. Property (3) guarantees that all vertices/edges except possibly the very first edge (in B-order) and the very first vertex (in B-order) can be colored! In fact, this idea alone already gives us an $O(n)$-time 5-list total coloring algorithm if G is not cubic: *simply B-order the vertices and edges with respect to some vertex x with $d(x) < 3$, and greedily color them in backward B-order.* Greedy coloring works correctly on such graphs because, at the time the algorithm is about to color it, the very first edge (in B-order) has at most four colored neighbors, and the same holds for the very first vertex. In Section 4 we will augment this greedy approach to get a full 5-list total coloring algorithm.

2.2. Subcubic Graph Decomposition

Theorem 1 *A subcubic graph G can be decomposed into edge-disjoint subgraphs $\mathcal{C}$ and $\mathcal{T}$, where $\mathcal{C}$ is a collection of vertex-disjoint cycles, and $\mathcal{T}$ is a subcubic forest. (See Fig. 1.) Furthermore, G admits a decomposition without chords unless it contains a triple bond.*

Proof. Let $\mathcal{C}$ be a maximal collection of edge-disjoint cycles in G. Any two cycles

of $\mathcal{C}$ are vertex-disjoint since G is subcubic. $\mathcal{T} = G - E(\mathcal{C})$ is a forest and obviously subcubic. If G contains no triple bond then we can choose the cycles of $\mathcal{C}$ to be chordless. □

The above decomposition theorem for subcubic graphs first appears in [38]. Note that decomposition of a subcubic graph into trees and cycles is not unique in general. Nevertheless, we can classify the edges as either *cycle edges* or *tree edges* for a given decomposition.

2.3. List Colorings of Cycles

Lemma 2 *Let C be a cycle with color lists $L(\cdot)$ assigned to its vertices. If every list has at least two colors, then C is L-vertex-colorable in linear time unless C is odd and all lists are the same list of size two.*

Proof. Procedure 1 produces an L-vertex-coloring of C if it runs to completion. Its input consists of a labeled cycle $C = \langle v_1, v_2, \ldots, v_k, v_1\rangle$, color lists $L(\cdot)$, and a color $\alpha \in L(v_1)$.

Procedure 1
1. assign color α to v_1
2. **for** $i \leftarrow 2$ **to** $k-1$ **do**
3. assign to v_i any color in $L(v_i)$ distinct from that of v_{i-1}
4. assign to v_k any color in $L(v_k)$ distinct from that of v_{k-1} or v_1

Statements 1 & 3 can always be carried out since $|L(v_i)| \geq 2$ for all i. We show Statement 4 can always be carried out also by judiciously labeling C and picking the color α.

First suppose $|L(x)| > 2$ for some x. Label C so that $x = v_k$. Pick any $\alpha \in L(v_1)$. At most two colors are unavailable for v_k in Statement 4. Since $|L(v_k)| > 2$, Statement 4 can be carried out. Next suppose some adjacent vertices x, y satisfy $L(x) \neq L(y)$, say that $L(y) \setminus L(x) \neq \emptyset$. Label C so that $x = v_k$ and $y = v_1$. Pick $\alpha \in L(v_1) \setminus L(v_k)$. Since Statement 1 assigns α to v_1 and $|L(v_k) \setminus \{\alpha\}| = |L(v_k)| \geq 2$, Statement 4 can be carried out. The remaining possibility is for every vertices x, y to satisfy $L(x) = L(y)$ and $|L(x)| = 2$. Assume C is even or else there is nothing to prove. Choose any valid labeling of C. Pick any $\alpha \in L(v_1)$. Suppose $L(v_i) = \{\alpha, \beta\}$ for all i. It is easy to see the vertices are colored alternately α and β.

The time is clearly linear. □

Lemma 3 *Let $C = \langle v_1, v_2, \ldots, v_k, v_1\rangle$ be an odd cycle. For each i let $L(v_i)$ be a nonempty list of colors for vertex v_i such that $L(v_1) \subset L(v_2)$, $|L(v_2)| = 2$, $L(v_i) \supseteq L(v_2)$ for $3 \leq i < k$, and $L(v_k) \setminus L(v_2) \neq \emptyset$. Then C is L-vertex-colorable in linear time.*

Proof. By throwing away all excess colors, we may assume $L(v_i) = L(v_2)$ for all $3 \leq i < k$. Color $v_1, v_2, \ldots, v_k$ in that order, using any color available for each vertex. We now show this coloring procedure never fails. Say that $L(v_1) = \{\alpha\}$ and $L(v_i) = \{\alpha, \beta\}$ for all $2 \leq i < k$. When v_k is about to be colored, one of its neighbor v_1 is colored α and its other neighbor v_{k-1} is colored β. Since

$L(v_k) \setminus \{\alpha, \beta\} \neq \emptyset$, there is some color that can be assigned to v_k. The time is clearly linear. □

To edge-color a graph is the same as to vertex-color its line graph. Since the line graph of a cycle is a cycle, these corollaries follow from the above lemmas.

Corollary 1 *Let C be a cycle with color lists $L(\cdot)$ assigned to its edges. If every list has at least two colors, then C is L-edge-colorable in linear time unless C is odd and all lists are the same list of size two.* □

Corollary 2 *Let $C = \langle v_1, e_1, \ldots, v_k, e_k, v_1 \rangle$ be an odd cycle. For each i let $L(e_i)$ be a nonempty list of colors for edge e_i such that $L(e_1) \subset L(e_2)$, $|L(e_2)| = 2$, $L(e_i) \supseteq L(e_2)$ for $3 \leq i < k$, and $L(e_k) \setminus L(e_2) \neq \emptyset$. Then C is L-edge-colorable in linear time.* □

3. List Edge Coloring Algorithms

Let G be a subcubic graph every edge of which has its own list of four colors. Assume without loss of generality that G is connected.

3.1. Sequential Algorithm

We have already described in Section 2 how to 4-list-edge color G if it is not cubic. So assume G is cubic. If G is a triple bond, then it can clearly be 4-list-edge-colored. So assume further that G is not a triple bond. Since its *minimum degree* δ equals 3, graph G contains some cycle, and thus it contains some chordless cycle since it is not a triple bond. Let C be a chordless cycle. Let G' be the graph obtained from G by contracting C to a vertex $[C]$. B-order the edges of G' with respect to $[C]$. For each edge e of G' in backward B-order except the very first edge e_1, assign to e any color that is available for it (in G). It remains to color e_1 and the edges of C. In G, only two edges adjacent to e_1 are colored since e_1 is a pendant edge of C. Thus, two colors are available for e_1 (in G). If C is even, we color e_1 using any color available for it. If C is odd, we color e_1 by choosing a color available for it in such a way that it does not result in all edges of C having the same list of 2 available colors. Now color the edges of C by invoking Corollary 1.

We now prove the algorithm correct by showing it never gets stuck. Observe that if edges e_i and e_j are adjacent in G' and e_j is not incident with the root $[C]$, then e_i and e_j are adjacent in G as well. This fact and property (3) of Lemma 1 imply that the algorithm does not get stuck on any edge not incident with the root $[C]$. Every edge that is incident with $[C]$ is adjacent in G to two uncolored edges on the cycle C, so it has two available colors when the time comes for the algorithm to color it. Hence the algorithm does not get stuck on it either. The edges of C can be colored using their available colors because all hypotheses of Corollary 1 are satisfied. The running time is $O(m + n)$, which is $O(n)$ since G is subcubic.

3.2. Parallel Algorithm

We now describe a randomized EREW PRAM algorithm for 4-list edge coloring. It uses $O(n/\log n)$ processors and runs in $O(\log n)$ time with high probability. The

algorithm has two stages. The first stage obtains, by Theorem 1, a decomposition of G into a collection $\mathcal{C}$ of cycles and a forest $\mathcal{T}$. The second stage does the coloring. The trees are colored first; the cycles are colored later.

Here is how to decompose G into cycles $\mathcal{C}$ and forest $\mathcal{T}$. First find a spanning tree T of G, using the randomized $O(\log n)$-time, $O(m+n)$-work, EREW PRAM algorithm of [17]. This algorithm assumes the input graph G is specified using doubly-linked adjacency lists, with each edge uv appearing on the list of u having a pointer to its opposite edge vu on the list of v. It returns a spanning tree T by marking all edges of T as tree edges. Now, each nontree edge e_i $(1 \leq i \leq m-n+1)$ has an associated fundamental cycle $C(e_i)$. Let $\mathcal{C}$ be the direct sum, i.e., the mod-2 sum, of all fundamental cycles $C(e_1), C(e_2), \ldots, C(e_{m-n+1})$. Then $\mathcal{C}$ is a collection of edge-disjoint cycles. In fact, $\mathcal{C}$ is a collection of vertex-disjoint cycles since G is subcubic. Also, $\mathcal{C}$ contains all the nontree edges e_i. So we need only compute $\mathcal{C}$ and output the decomposition consisting of cycles $\mathcal{C}$ and forest $\mathcal{T} = T - \{\text{edges of } \mathcal{C}\}$.

To find the edges of $\mathcal{C}$, we have to compute, for each tree edge f, the parity of the number of fundamental cycles $C(e_i)$ containing f. Edge f is in $\mathcal{C}$ if and only if this number is odd. This can be done as follows. First make T into a rooted out-tree. For every tree edge $f = wv$, where w is the parent of v, compute the sum, over all descendants u of v, of the vertex degrees $d_{G-T}(u)$ in the graph $G-T$. Edge f is in $\mathcal{C}$ if and only if this sum is odd. Computing this sum for all tree edges can be done on an EREW PRAM in $O(\log n)$ time and $O(n)$ work by parallel prefix computation on the Euler tour of T [40, 32]. Once the edges of $\mathcal{C}$ and $\mathcal{T}$ are known, we create an adjacency list representation of $\mathcal{T}$, and a doubly-linked circular list representation for each cycle in $\mathcal{C}$.

To list-edge-color a tree in $\mathcal{T}$, number every vertex by its depth (using a parallel prefix computation on the Euler tour). Each vertex r of even depth will color the subtree of ≤ 6 edges that descend from it but no other even-depth vertex. The procedure is as follows. Let the subtree consist of edges ra_i for $i = 1, 2$, and $a_i a_{ij}$ for $i = 1, 2$ and $j = 1, 2$. Some of these edges may not exist. The coloring is in 2 parallel steps.

Step 1. There are two substeps.

1. Assign each edge ra_i 2 colors from its list, such that the 4 colors assigned to these 2 edges are distinct. This can be done since all lists have length 4.

2. For $i = 1, 2$ do

(a) Let c_1, c_2 be the colors on ra_i.

(b) Assign color d_j to $a_i a_{ij}$ for $j = 1, 2$, where d_j is in the edge's list and c_1, c_2, d_1, d_2 are 4 distinct colors. This can be done since all lists have length 4.

Step 2. Color ra_i with c_1 or c_2, whichever is distinct from the color of r's parent edge (which was chosen in substep 2(b) of Step 1).

This tree coloring procedure is clearly an EREW PRAM algorithm. We make the coloring of $\mathcal{T}$ work-optimal as follows. Assign each processor $\log n$ vertices (so that $n/\log n$ processors are used in total). Each processor looks at each of its assigned vertices. It performs the above coloring steps for all even-depth vertices; it does nothing for odd-depth vertices. Note that depths enable the algorithm to

decide which edges go down the tree.

Each cycle $C \in \mathcal{C}$ is colored by invoking Corollary 1. For each $e \in E(C)$, let $L(e)$ be the set of colors available for e. Any edge of C has at least 2 available colors since it has four colors in its list and it is adjacent to at most 2 colored edges (because each of its endpoints is adjacent to at most one colored edge). So Corollary 1 applies unless C is a *bad cycle*, i.e., it is odd and every edge in it has the same two colors available. Suppose C is a bad cycle. Then every vertex of C has degree 3 (in G), and it is incident with a pendant edge or a chord. Let e be a pendant edge (chord) of C. Since C is a bad cycle, e is adjacent to exactly two (four) edges of C if e is a pendant edge (chord). Thus, recoloring e changes the available colors for some but not all edges of C. So again Corollary 1 applies.

Given the doubly-linked circular list representation of each cycle in $\mathcal{C}$, we can easily implement the entire cycle coloring process in $O(\log n)$ parallel time and $O(n)$ work using standard techniques as mentioned above.

4. Total Coloring Algorithms

Let G be a subcubic graph every vertex/edge x of which has its own list $\lambda(x)$ of five colors. We have to find a λ-total-coloring. To specialize the following description to the 5-total-coloring problem, simply set $\lambda(x) = [5]$ for all x.

Assume without loss of generality that G is connected. We have already described in Section 2 how to 5-list-total color G if it is not cubic. So assume G is cubic. If G is a triple bond, then clearly it can be 5-list-total-colored. So assume further that G is not a triple bond. Since $\delta = 3$, graph G contains some cycle, and thus G contains some chordless cycle since it is not a triple bond. Let C be a chordless cycle. Let G' be the graph obtained from G by contracting C to a vertex $[C]$. B-order the vertices and edges of G' with respect to $[C]$. For each vertex or edge y_i of G' in backward B-order except vertex $[C]$ and edges incident with it, assign to y_i any color that is available for it. Property (3) of Lemma 1 guarantees this is always possible.

It remains to show we can complete the coloring by extending it to cover the pendant edges & the vertices and edges of C, using their lists of available colors.

Let us adopt the following naming convention. Suppose cycle C is labeled $\langle v_1, e_1, \ldots, v_k, e_k, v_1\rangle$. Denote the pendant edge incident with v_i by f_i, and name f_i's other endpoint w_i. Observe that no w_i is the same as any v_i since C is chordless. However, it is possible that $w_i = w_j$ for some distinct i, j. For each vertex/edge x, write $L(x)$ for the set of colors still available for x. If x is colored, let $\varphi(x)$ denote its color. When $i = 1$ (resp. $i = k$), vertex v_{i-1} (resp. v_{i+1}) refers to v_k (resp. v_1); similarly for vertex w_{i-1} (resp. w_{i+1}) and edge f_{i-1} (resp. f_{i+1}).

Since G is cubic, every vertex x on C is incident with a pendant edge xy with y already colored. Choose any valid labeling of cycle C. Observe that each f_i has at most three colored neighbors; and thus $|L(f_i)| \geq 2$ in the current coloring φ.

The rest of the 5-total-coloring algorithm is described in Subsection 4.1. The rest of the λ-total-coloring algorithm is described in Subsection 4.2.

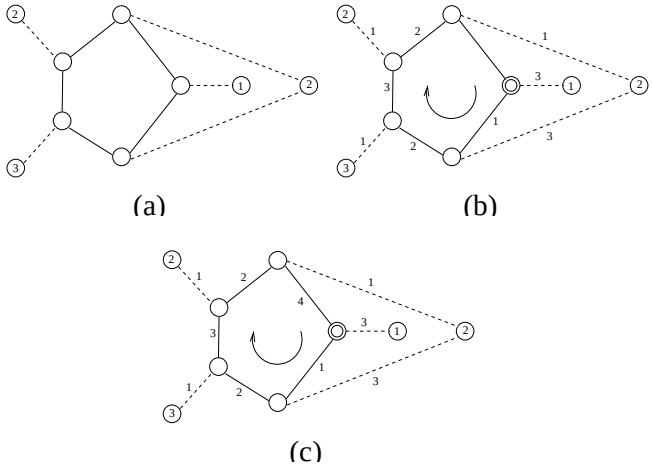

Fig. 2. Example Case 1 of the 5-total-coloring algorithm. (a) Original configuration. (b) After execution of the first for loop of Procedure 2. (c) After the second for loop. Vertex v_1 is doubly circled. The indices of vertices & edges of the cycle increase in the direction indicated by the arrow.

4.1. The 5-total-coloring Algorithm

There are two cases to consider. In Case 1 we first color all the uncolored (pendant & cycle) edges but leave all the vertices uncolored. However, we make sure that each of the vertices has 2 colors available. We then color the vertices by invoking Lemma 2 if possible. If not possible, we recolor one cycle edge so that the vertices can be colored by invoking Lemma 3. In Case 2, we first color all the uncolored pendant edges and cycle vertices but leave all the cycle edges uncolored. However, we make sure that each of the cycle edges has at least 2 colors available. We then color the cycle edges by invoking Corollary 1 if possible. If not possible, we recolor one cycle vertex so that the cycle edges can be colored by invoking Corollary 2. Here are the details.

Case 1 Some index i satisfies $\varphi(w_i) \neq \varphi(w_{i+1})$. By relabeling if necessary, we may assume that $\varphi(w_k) \neq \varphi(w_1)$. (See Fig. 2.) We will color all the f_i and e_i so that, for all i, either $\varphi(e_i) = \varphi(w_i)$ or $\varphi(e_{i-1}) = \varphi(w_i)$. We do this using Procedure 2.

Procedure 2
$\varphi(f_1) \leftarrow$ any color in $L(f_1) \setminus \{\varphi(w_k)\}$
$\varphi(e_1) \leftarrow \varphi(w_1)$
for $i \leftarrow 2$ **to** k **do** {
  $\varphi(f_i) \leftarrow$ any color in $L(f_i)$
  if $\varphi(w_i) \in L(e_i)$ **then**
    $\varphi(e_i) \leftarrow \varphi(w_i)$ }
for $i \leftarrow 2$ **to** k **do**
  if e_i is still uncolored **then**
    $\varphi(e_i) \leftarrow$ any color in $L(e_i)$

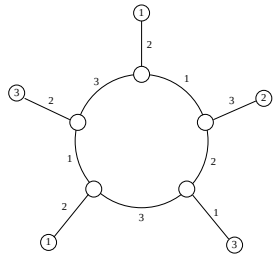

Fig. 3. Example obstruction in Case 1 of the 5-total-coloring algorithm, when an edge-colored cycle cannot be extended to a total coloring of that cycle. Changing the color of a single edge on the cycle allows such an extension to be carried out.

It is easy to see that Procedure 2 never gets stuck and it extends φ to cover all f_i and e_i. Procedure 2 assigns the color of w_1 to edge e_1, and for all $i = 2$ to k in order, the if statement in its first for loop assigns the color of w_i to edge e_i if it is still available for e_i. Clearly this color can be unavailable for e_i only if e_{i-1} and w_i have the same color. Thus, Procedure 2 results in $|L(v_i)| = 2$ for every i, because 4 neighbors of v_i, viz., w_i, f_i, e_{i-1}, and e_i, are colored, with exactly two of them colored the same (either $\varphi(e_i) = \varphi(w_i)$ or $\varphi(e_{i-1}) = \varphi(w_i)$).

If all the hypotheses of Lemma 2 are satisfied, then φ can be extended to a total-coloring of C and we are done. So suppose C is odd and all $L(v_i)$ consist of the same two colors. Consider edge e_k. Since $L(v_k) = L(v_1)$, we have $\{\varphi(e_{k-1}), \varphi(f_k)\} = \{\varphi(e_1), \varphi(f_1)\}$. Hence there exists some color $\alpha \in L(e_k)$ different from the current color $\varphi(e_k)$ of e_k. Change the color of e_k to α. This affects $L(v_k)$ and $L(v_1)$ but leaves the remaining $L(v_i)$ intact. Following this color change, either all the hypotheses of Lemma 2 or all the hypotheses of Lemma 3 are satisfied. (See Fig. 3.) Note that we may have to relabel the cycle in the opposite orientation to get the hypotheses of Lemma 3 to hold. Therefore, φ can be extended to a total-coloring of C.

Case 2 All w_i receive the same color. (See Fig. 4.) Color all the f_i and v_i using Procedure 3.

Procedure 3
$\varphi(f_1) \leftarrow$ any color in $L(f_1)$
for $i \leftarrow 2$ **to** k **do** {
$\varphi(v_i) \leftarrow \varphi(f_{i-1})$
$\varphi(f_i) \leftarrow$ any color in $L(f_i)$ }
if $\varphi(f_k) \neq \varphi(f_1)$ **then**
$\varphi(v_1) \leftarrow \varphi(f_k)$
else
$\varphi(v_1) \leftarrow$ any color in $L(v_1)$

It is easy to see that Procedure 3 never gets stuck and it extends φ to cover all f_i and v_i. It results in either

(i) $\varphi(f_i) = \varphi(v_{i+1})$ and $\varphi(f_i) \neq \varphi(f_{i+1})$ for all $1 \leq i \leq k$, or

(ii) $\varphi(f_k) = \varphi(f_1)$, but $\varphi(f_i) = \varphi(v_{i+1})$ and $\varphi(f_i) \neq \varphi(f_{i+1})$ for all $1 \leq i < k$.

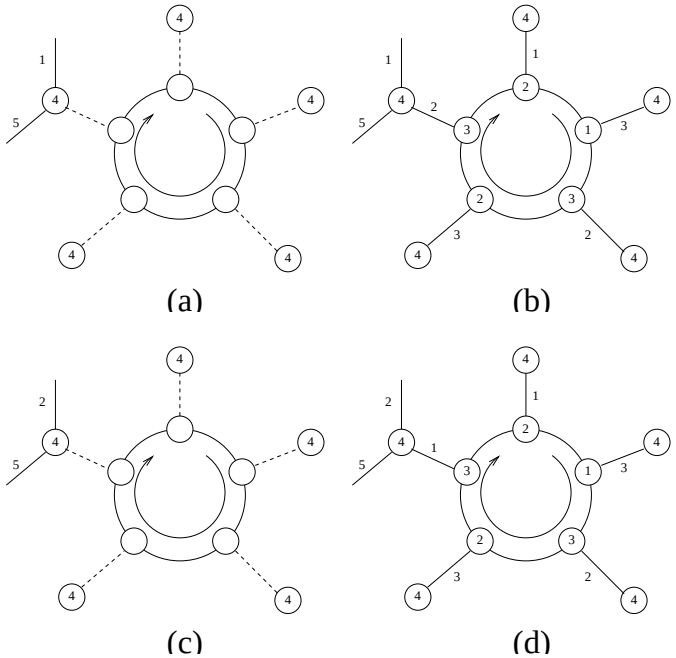

Fig. 4. Example Case 2 of the 5-total-coloring algorithm. Two possible scenarios (i)-(ii): **(i)** Original configuration in (a), and after Procedure 3 in (b). **(ii)** Original configuration in (c), and after Procedure 3 in (d).

In both cases we have $|L(e_i)| \geq 2$ for all $1 \leq i \leq k$. If all the hypotheses of Corollary 1 are satisfied, then φ can be extended to a total-coloring of C and we are done. So suppose C is odd and all $L(e_i)$ consist of the same two colors, say α and β. Let γ, δ, and ϵ be the colors of f_1, f_2, and v_1 respectively. By the supposition and by the way Procedure 3 works, we see that $3 \mid k$ and for all $1 \leq i \leq k$ we have

$$\varphi(f_i) = \varphi(v_{i+1}) = \begin{cases} \gamma, & \text{if } i \equiv 1 \pmod 3, \\ \delta, & \text{if } i \equiv 2 \pmod 3, \\ \epsilon, & \text{if } i \equiv 0 \pmod 3. \end{cases} \tag{1}$$

(See Fig. 5.) Moreover, we must be in case (i) above.

Change the color of v_2 to the only remaining color in $[5] \setminus \{\gamma, \delta, \epsilon, \varphi(w_2)\}$. This changes $L(e_1)$ and $L(e_2)$ but leaves the remaining $L(e_i)$ intact. Following this color change, all the hypotheses of Corollary 2 are satisfied. Note that we have to relabel the cycle to get the hypotheses of Corollary 2 to hold. Therefore, φ can be extended to a 5-total-coloring of C.

This completes the 5-total-coloring algorithm.

4.2. The λ-total-coloring Algorithm

To understand the algorithm of this subsection, one needs to study the relationships that hold among the color lists of w_i, f_i, v_i, and e_i $(1 \leq i \leq k)$, and the effects of recoloring on these lists. First some definitions.

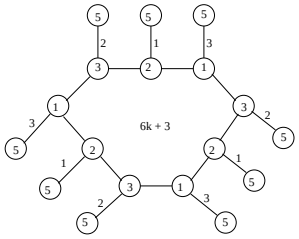

Fig. 5. Example obstruction in Case 2 of the 5-total-coloring algorithm, when a vertex-colored cycle cannot be extended to a total coloring of that cycle. Changing the color of a single vertex on the cycle allows such an extension to be carried out.

Definitions Let $1 \leq i \leq k$. The neighbors of vertex v_i that are not vertices of cycle C feature prominently in our algorithm. These are w_i, f_i, e_{i-1}, and e_i. Any of these neighbors is a *safety witness for* v_i if it is colored with a color not in $\lambda(v_i)$. Edge e_{i-1} or e_i can also be a *safety witness for* v_i if it is colored with the same color as $\varphi(w_i)$. Vertex v_i is *safe* if it has a safety witness; it is *unsafe* otherwise.

Similarly, the neighbors of edge e_i that are not edges of C feature prominently in our algorithm. These are f_i, v_i, f_{i+1}, and v_{i+1}. Any of these neighbors is a *safety witness for* e_i if it is colored with a color not in $\lambda(e_i)$. Any of these neighbors can also be a *safety witness for* e_i if it is colored and it participates in one of the following conditions: **(a)** $\varphi(f_i) = \varphi(f_{i+1})$ **(b)** $\varphi(f_i) = \varphi(v_{i+1})$ **(c)** $\varphi(v_i) = \varphi(f_{i+1})$. Edge e_i is *safe* if it has a safety witness; it is *unsafe* otherwise.

Define $\sigma(i) = \big(\lambda(e_i) \setminus \lambda(v_i)\big) \cup \big(\lambda(e_i) \cap \{\varphi(w_i)\}\big)$. The motivation for the definition of $\sigma(i)$ is that assigning a color in $\sigma(i)$ to edge e_i will make v_i safe. Note that our coloring algorithm will maintain the following invariant.

Invariant: $\varphi(x) \in \lambda(x)$ for any colored vertex/edge x, and no two neighbors receive the same color.

Claim 1 *Suppose for all $1 \leq i \leq k$ it is true that f_i and e_i are colored but v_i is not. Then any vertex v_i is safe if and only if $|L(v_i)| \geq 2$. The following conditions are equivalent.*

- $|L(v_i)| = 2$.
- *v_i has exactly one safety witness.*
- $\big\{\varphi(w_i), \varphi(f_i), \varphi(e_{i-1}), \varphi(e_i)\big\} \cap \lambda(v_i)$ *has exactly 3 colors.*

Proof. By definitions of safe vertex and safety witness, and by the fact that the algorithm maintains the Invariant at all time. □

Claim 2 *Suppose for all $1 \leq i \leq k$ it is true that f_i and v_i are colored but e_i is not. Then any edge e_i is safe if and only if $|L(e_i)| \geq 2$. The following conditions are equivalent.*

- $|L(e_i)| = 2$.
- *either exactly one of $\varphi(f_i)$, $\varphi(v_i)$, $\varphi(f_{i+1})$, $\varphi(v_{i+1})$ is not a color in $\lambda(e_i)$, or they are all in $\lambda(e_i)$ and exactly one condition in **(a)**–**(c)** in the definitions of safety witness for e_i holds.*
- $\big\{\varphi(f_i), \varphi(v_i), \varphi(f_{i+1}), \varphi(v_{i+1})\big\} \cap \lambda(e_i)$ *has exactly 3 colors.*

Proof. By definition of safe edge and by the fact that the algorithm maintains the Invariant at all time. □

Claim 3 *For any pair of vertices/edges x, y and for any $1 \leq i \leq k$ we have*
(i) $|\lambda(x) \setminus \lambda(y)| = |\lambda(y) \setminus \lambda(x)|$.
(ii) $\sigma(i) \subseteq \lambda(e_i)$ *and assigning any color in* $\sigma(i)$ *to* e_i *makes* v_i *safe.*
(iii) *If* $\varphi(w_i) \in \lambda(v_i)$ *then* $\sigma(i)$ *is nonempty.*
(iv) *If* v_i *is unsafe then* $\sigma(i)$ *is nonempty.*

Proof. (i) This holds since $|\lambda(x)| = 5$ for every vertex/edge x.
(ii) By definitions of σ and safety witness for v_i.
(iii) By definition of σ and part (i).
(iv) If v_i is unsafe, then w_i is not a safety witness for v_i, i.e., $\varphi(w_i) \in \lambda(v_i)$, and we are done by part (iii). □

Claim 4 *Assume some vertex v_i is safe, and w_i, f_i, e_{i-1}, e_i are colored. Suppose it is possible to change the color of e_i, while maintaining the Invariant and keeping the colors of w_i, f_i, e_{i-1} the same. There are two possibilities.*

(I) Suppose e_i is a safety witness for v_i. After recoloring, the set $L(v_i)$ satisfies either (i) $|L(v_i)| = 1$ by having a color removed from the old $L(v_i)$, or (ii) $L(v_i)$ stays the same. Case (i) occurs if and only if the color used to recolor e_i belongs to the old $L(v_i)$.

(II) Suppose e_i is not a safety witness for v_i. After recoloring, the set $L(v_i)$ satisfies either (i) $|L(v_i)| = 2$ by having exactly one color in the old $L(v_i)$ changed to the old color of e_i, or (ii) $|L(v_i)| = 3$ by having the old color of e_i added to the old $L(v_i)$. Case (i) occurs if and only if the color used to recolor e_i belongs to the old $L(v_i)$.

Proof. Let $L(v_i) = \{\alpha, \beta\}$. Let γ be the current color of e_i. Let δ be the color to be used for recoloring e_i.

Table 1 shows what can happen if e_i is a safety witness for v_i. There are 2 choices of γ; with 3 choices of δ for the first choice of γ, and 2 choices of δ for the second choice of γ. The last column gives the values of $L(v_i)$ after recoloring for all possible combinations of γ and δ.

Table 1. Case (I) of Claim 4: e_i is a safety witness for v_i

$\gamma \notin \lambda(v_i)$	$\delta \notin \lambda(v_i)$	$\{\alpha, \beta\}$
	$\delta \in \lambda(v_i)$, $\delta = \varphi(w_i)$	$\{\alpha, \beta\}$
	$\delta \in \lambda(v_i)$, $\delta \neq \varphi(w_i)$	$\{\alpha\}$ or $\{\beta\}$
$\gamma = \varphi(w_i)$	$\delta \notin \lambda(v_i)$	$\{\alpha, \beta\}$
	$\delta \in \lambda(v_i)$, $\delta \neq \varphi(w_i)$	$\{\alpha\}$ or $\{\beta\}$

Table 2 shows what can happen if e_i is not a safety witness for v_i. Since e_i is not a witness for v_i, we have $\gamma \in \lambda(v_i)$ and $\gamma \neq \varphi(w_i)$. There are 3 choices of δ. The last column gives the values of $L(v_i)$ after recoloring for all possible choices of δ.

Our claim is easily verified with the help of these two tables. □

Note that the roles of e_{i-1} and e_i in Claim 4 are completely interchangeable. In fact, Claim 4 has a symmetric version whose only difference is that e_{i-1} and e_i

Table 2. Case (II) of Claim 4: e_i is not a safety witness for v_i

$\gamma \in \lambda(v_i)$, $\gamma \neq \varphi(w_i)$	$\delta \notin \lambda(v_i)$	$\{\alpha, \beta, \gamma\}$
	$\delta \in \lambda(v_i)$, $\delta = \varphi(w_i)$	$\{\alpha, \beta, \gamma\}$
	$\delta \in \lambda(v_i)$, $\delta \neq \varphi(w_i)$	$\{\alpha, \gamma\}$ or $\{\beta, \gamma\}$

are interchanged. From now on whenever we invoke Claim 4 we will not make a distinction between these two versions and will let context decide which version is meant.

We are now ready to continue with the algorithm. By Claim 3(iii), $\sigma(i)$ is nonempty if $\varphi(w_i) \in \lambda(v_i)$. Therefore, $\sigma(i) \neq \emptyset$ in Cases 1 & 2 below. The following three cases thus exhaust all possibilities.

1. Some index i satisfies $\varphi(w_i) \notin \lambda(v_i)$ or $L(f_i) \setminus \lambda(v_i) \neq \emptyset$.

2. Not 1, and some valid labeling of C has some index i satisfying $\sigma(i) \neq \sigma(i-1)$ or $|\sigma(i)| > 1$.

3. Not 1, and any valid labeling of C has all $\sigma(i)$ as the same one-color set.

Note that for any vertex x on C, there are two possible valid labelings of C with x labeled v_1. This is because one can tour C starting from x in either clockwise or counterclockwise direction.

We will invoke Lemma 3 and Corollary 2 several times in the following description. In all such situations, we want to apply the lemma/corollary to some labeled cycle. However, we may have to relabel the cycle before we can apply the statement of the lemma/corollary. To avoid monotony, we will drop the necessary phrase "By relabeling if necessary," in all such situations. Note that relabeling may involve changing the direction of traversal.

Our strategy is as follows. In Cases 1 & 2 we will first color all the f_i and e_i so as to make all v_i safe, and then extend φ to cover all the v_i as well, after making any necessary color changes on some (no more than two) edges. In Case 3 we will first color all the f_i and v_i so as to make all e_i safe, and then extend φ to cover all the e_i as well, after making any necessary color change on a vertex. Here are the details.

Case 1 Some index i satisfies $\varphi(w_i) \notin \lambda(v_i)$ or $L(f_i) \setminus \lambda(v_i) \neq \emptyset$. By relabeling if necessary, we may assume that $i = k$ satisfies the condition. Color all the f_i and e_i using Procedure 4.

Procedure 4
  for $i \leftarrow 1$ **to** k **do** {
    if $L(f_i) \setminus \lambda(v_i) \neq \emptyset$ **then**
      $\varphi(f_i) \leftarrow$ any color in $L(f_i) \setminus \lambda(v_i)$
    else
      $\varphi(f_i) \leftarrow$ any color in $L(f_i)$
    if v_i is unsafe /* always false if $i = k$ */ **then**
      $\varphi(e_i) \leftarrow$ any color in $\sigma(i)$ }
  for $i \leftarrow 1$ **to** k **do**
    if e_i is still uncolored **then**
      $\varphi(e_i) \leftarrow$ any color in $L(e_i)$

Case 2 $\varphi(w_i) \in \lambda(v_i)$ and $L(f_i) \subseteq \lambda(v_i)$, and some valid labeling of C has some index i satisfying $\sigma(i) \neq \sigma(i-1)$ or $|\sigma(i)| > 1$. By relabeling if necessary, we may assume that $i = 1$ satisfies the condition. Let $\zeta \in \sigma(1)$ and $\eta \in \sigma(k)$ be distinct colors. Define a function c by setting $c(1) = \zeta$, $c(k) = \eta$, and setting $c(i)$ to be some color in $\sigma(i)$ for each $1 < i < k$. The function c is well-defined since $\sigma(i) \neq \emptyset$ for every i (by Claim 3(iii)). Color all the f_i and e_i using Procedure 5.

Procedure 5
$\varphi(f_1) \leftarrow$ any color in $L(f_1) \setminus \{c(k)\}$
$\varphi(e_1) \leftarrow c(1)$
for $i \leftarrow 2$ **to** k **do** {
  $\varphi(f_i) \leftarrow$ any color in $L(f_i)$
  if $c(i) \in L(e_i)$ **then**
    $\varphi(e_i) \leftarrow c(i)$ }
for $i \leftarrow 2$ **to** k **do**
  if e_i is still uncolored **then**
    $\varphi(e_i) \leftarrow$ any color in $L(e_i)$

By Claims 1 & 3(ii) & 3(iv), we see that Procedures 4 & 5 extend φ to cover all f_i and e_i and make all v_i safe, while maintaining the Invariant. If all the hypotheses of Lemma 2 are satisfied, then φ can be extended to a λ-total-coloring of C and we are done. So suppose C is odd and all $L(v_i)$ consist of the same two colors, say α and β. Consider edge e_i that is not a safety witness for both v_i and v_{i+1}. Existence of such an edge is guaranteed because $k \geq 3$, and $|L(v_i)| = 2$ for all i (so every v_i has exactly one safety witness by Claim 1). By relabeling if necessary, we may assume e_i is not a safety witness for v_i.

First suppose $L(e_i) \supset \{\varphi(e_i)\}$. Change the color of e_i to some other color in $L(e_i)$. This color change affects only lists $L(v_i)$ and $L(v_{i+1})$. By Claim 4, either Lemma 2 or Lemma 3 applies after the color change, irrespective of whether e_i is a safety witness for v_{i+1} or not.

Next suppose $L(e_i) = \{\varphi(e_i)\}$, but $L(e_{i+1}) \supset \{\varphi(e_{i+1})\}$. Say γ is the current color of e_{i+1} and δ is some other color in $L(e_{i+1})$. Recolor e_{i+1} by δ. If either Lemma 2 or Lemma 3 applies, then we are done. Otherwise, we must have $\delta \in \{\alpha, \beta\}$ and γ must have been a safety witness for both v_{i+1} and v_{i+2}. In that case, also recolor e_i by γ. Now Lemma 3 applies (use Claim 4).

Last suppose $L(e_i) = \{\varphi(e_i)\}$ and $L(e_{i+1}) = \{\varphi(e_{i+1})\}$. Then we can switch the colors of e_i and e_{i+1} and still maintain the Invariant! Lemma 2 applies after the color switch (use Claim 4).

Case 3 $\varphi(w_i) \in \lambda(v_i)$ and $L(f_i) \subseteq \lambda(v_i)$, and any valid labeling of C has all $\sigma(i)$ as the same one-color set, say $\{\alpha\}$. This case can happen only when there exist distinct colors $\beta, \gamma, \delta, \epsilon$ all different from α such that for each i we have $\lambda(e_i) = \{\alpha, \beta, \gamma, \delta, \epsilon\}$ and either $\lambda(v_i) = \lambda(e_i)$ or $\big(\{\alpha\} = \lambda(e_i) \setminus \lambda(v_i)$ and $\{\varphi(w_i)\} = \lambda(v_i) \setminus \lambda(e_i)\big)$. Color all the f_i and v_i using Procedure 3.

It is easy to see that Procedure 3 extends φ to cover all f_i and v_i while maintaining the Invariant. It also results in safe e_i for every i (use Claim 2). If all the hypotheses of Corollary 1 are satisfied, then φ can be extended to a λ-total-coloring

of C and we are done. So suppose C is odd and all $L(e_i)$ consist of the same two colors, say α and β. By this supposition and by the way Procedure 3 works, we see that $3 \mid k$ and there exist distinct colors γ, δ, ϵ all different from α, β such that Equation (1) holds for all i.

Change the color of v_2 to the only color in $\lambda(v_2) \setminus \{\gamma, \delta, \epsilon, \varphi(w_2)\}$. This changes $L(e_1)$ and $L(e_2)$ but leaves the remaining $L(e_i)$ intact. Following this color change, all the hypotheses of Corollary 2 are satisfied. So φ can be extended to a λ-total-coloring of C.

This completes the λ-total-coloring algorithm.

5. Conclusion

We propose two general methods and use them to design several efficient coloring algorithms for subcubic graphs. The first method, called B-ordering, orders the vertices and edges of any graph in such a way that we can color all or almost all of the graph elements in a semi-greedy fashion. The second method is a simple structure theorem for subcubic graphs whose consideration leads to efficient coloring algorithms, both sequential and parallel. Using the methods, we are able to obtain $O(n)$-time sequential algorithms for the following problems on subcubic graphs: (1) 4-list-edge-coloring (2) 5-total-coloring (3) 5-list-total-coloring. Our algorithm for problem (1) is simpler than previous algorithms. Our algorithms for problems (2) & (3) are the first $O(n)$-time algorithms. For problem (1), we also obtain the first randomized EREW PRAM algorithm that uses $O(n/\log n)$ processors and runs in $O(\log n)$ time with high probability. We expect to use both methods to solve more coloring problems. We also anticipate the use of the second method to attack non-coloring problems on subcubic graphs.

References

1. Paola Alimonti and Viggo Kann. Hardness of approximating problems on cubic graphs. *Theoretical Computer Science*, 237:123–134, 2000.
2. Therese C. Biedl. Linear reductions of maximum matching. Technical Report CS-2000-17, University of Waterloo Department of Computer Science, October 2000.
3. Therese C. Biedl, Prosenjit Bose, Erik D. Demaine, and Anna Lubiw. Efficient algorithms for Petersen's Matching Theorem. *Journal of Algorithms*, 38:110–134, 2001.
4. Béla Bollobás and A. J. Harris. List-colourings of graphs. *Graphs and Combinatorics*, 1:115–127, 1985.
5. J. Adrian Bondy and U. S. R. Murty. *Graph Theory with Applications.* The Macmillan Press Ltd, 1976.
6. Oleg V. Borodin, Alexandr V. Kostochka, and Douglas R. Woodall. List edge and list total colourings of multigraphs. *Journal of Combinatorial Theory Series B*, 71:184–204, 1997.
7. R. L. Brooks. On colouring the nodes of a network. *Proceedings of the Cambridge Philosophical Society. Mathematical and Physical Sciences*, 37:194–197, 1941.
8. Tiziana Calamoneri. *Does Cubicity Help to Solve Problems?* PhD thesis, Università Degli Studi di Roma "La Sapienza", 1997. IX-97-2.

9. Amanda G. Chetwynd and Roland Häggkvist. A note on list-colorings. *Journal of Graph Theory*, 13(1):87–95, 1989.
10. M. Ellingham and L. A. Goddyn. List edge colourings of some 1-factorable multigraphs. *Combinatorica*, 16(3):343–352, 1996.
11. Paul Erdős, A. L. Rubin, and H. Taylor. Choosability in graphs. In *Proceedings of the West-Coast Conference on Combinatorics, Graph Theory and Computing*, volume XXVI of *Congressus Numerantium*, pages 125–157, Arcata, California, 1979.
12. Harold N. Gabow, Takao Nishizeki, Oded Kariv, Daniel Leven, and Osamu Terada. Algorithms for edge-coloring graphs. Technical Report TRECIS-8501, Tohoku University, 1985.
13. Harold N. Gabow and San Skulrattanakulchai. Coloring algorithms on subcubic graphs. In Oscar H. Ibarra and Louxin Zhang, editors, *Proceedings of the 8th Annual International Computing & Combinatorics Conference*, volume 2387 of *Lecture Notes in Computer Science*, pages 67–76, Berlin, 2002. Springer-Verlag.
14. Fred Galvin. The list chromatic index of a bipartite multigraph. *Journal of Combinatorial Theory Series B*, 63:153–158, 1995.
15. Michael R. Gary and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co., San Francisco, CA, 1979.
16. Raymond Greenlaw and Rossella Petreschi. Cubic graphs. *ACM Computing Surveys*, 27(4):471–495, 1995.
17. Shay Halperin and Uri Zwick. Optimal randomized EREW PRAM algorithms for finding spanning forests. *Journal of Algorithms*, 39:1–46, 2001.
18. Ian J. Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.
19. Tommy R. Jensen and Bjarne Toft. *Graph Coloring Problems*. John Wiley & Sons, New York, New York, 1995.
20. Martin Juvan and Bojan Mohar. List colorings of outerplanar graphs. Unpublished Manuscript, 1996.
21. Martin Juvan, Bojan Mohar, and Riste Škrekovski. List total colorings of graphs. *Combinatorics, Probability & Computing*, 7:181–188, 1998.
22. Martin Juvan, Bojan Mohar, and Riste Škrekovski. On list edge-colorings of subcubic graphs. *Discrete Mathematics*, 187:137–149, 1998.
23. Martin Juvan, Bojan Mohar, and Robin Thomas. List edge-colorings of series-parallel graphs. *The Electronic Journal of Combinatorics*, 6:#R42, 1999.
24. Viggo Kann. Maximum bounded 3-dimensional matching is MAX SNP-complete. *Information Processing Letters*, 37:27–35, 1991.
25. Viggo Kann. *On the Approximability of NP-Complete Optimization Problems*. PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1992.
26. Alexandr V. Kostochka and Douglas R. Woodall. Total choosability of multicircuits I. *Journal of Graph Theory*, 40:26–43, 2002.
27. Alexandr V. Kostochka and Douglas R. Woodall. Total choosability of multicircuits II. *Journal of Graph Theory*, 40:44–67, 2002.
28. László Lovász. Three short proofs in graph theory. *Journal of Combinatorial Theory Series B*, 19:269–271, 1975.
29. László Lovász. *Combinatorial Problems and Exercises*. North-Holland, Amsterdam, second edition, 1993.

30. Colin J. H. McDiarmid and Abdón Sánchez-Arroyo. Total colouring regular bipartite graphs is NP-hard. *Discrete Mathematics*, 124:155–162, 1994.
31. Dale Peterson and Douglas R. Woodall. Edge-choosability in line-perfect multigraphs. *Discrete Mathematics*, 202:191–199, 1999.
32. John H. Reif, editor. *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers, California, 1993.
33. M. Rosenfeld. On the total coloring of certain graphs. *Israel Journal of Mathematics*, 9:396–402, 1971.
34. Thomas L. Saaty and Paul C. Kainen. *The Four-Color Problem, Assaults and Conquest*. Dover Publications, Inc., New York, second edition, 1986.
35. Abdón Sánchez-Arroyo. Determining the total colouring number is NP-hard. *Discrete Mathematics*, 78:315–319, 1989.
36. Abdón Sánchez-Arroyo. Total colourings and complexity. Master's thesis, University of Oxford, 1989.
37. San Skulrattanakulchai. Acyclic colorings of subcubic graphs. Manuscript.
38. San Skulrattanakulchai. 4-edge-coloring graphs of maximum degree 3 in linear time. *Information Processing Letters*, 81:191–195, 2002.
39. San Skulrattanakulchai. Δ-list vertex coloring in linear time. In Martti Penttonen and Erik Meineche Schmidt, editors, *Proceedings of the 8th Scandinavian Workshop on Algorithm Theory*, volume 2368 of *Lecture Notes in Computer Science*, pages 240–248, Berlin, 2002. Springer-Verlag.
40. Robert E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14:862–874, 1985.
41. N. Vijayaditya. On total chromatic number of a graph. *Journal of the London Mathematical Society*, 3(2):405–408, 1971.
42. Vadim G. Vizing. On an estimate of the chromatic class of a p-graph. *Metody Diskret. Analiz.*, 3:25–30, 1964. (In Russian).
43. Vadim G. Vizing. Coloring the vertices of a graph in prescribed colors. *Metody Diskret. Anal. v Teorii Kodov i Schem*, 29:3–10, 1976.
44. Douglas R. Woodall. Edge-choosability of multicircuits. *Discrete Mathematics*, 202:271–277, 1999.
45. Hian-Poh Yap. *Total Colourings of Graphs*, volume 1623 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1996.