LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-320

Efficient Parallel Algorithms for $(\Delta+1)$-Coloring and Maximal Independent Set Problems

Andrew V. Goldberg
Serge A. Plotkin

January 1987

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Abstract

We describe an efficient technique for breaking symmetry in parallel. The technique works especially well on rooted trees and on graphs with a small maximum degree. In particular, we can find a maximal independent set on a constant-degree graph in $O(\lg^* n)$ time on an EREW PRAM using a linear number of processors. We show how to apply this technique to construct more efficient parallel algorithms for several problems, including coloring of planar graphs and $(\Delta+1)$-coloring of constant-degree graphs. We also prove lower bounds for two related problems.

1 Introduction

Some problems for which trivial sequential algorithms exist appear to be much harder to solve in a parallel framework. Therefore, new methods are needed for design of efficient parallel algorithms. A known example of a problem with a trivial sequential algorithm which is hard to solve in parallel, is the problem of finding a maximal independent set in a graph [18]. This problem was shown to be in the class NC by Karp and Wigderson [12]. A simple randomized algorithm for the problem is due to Luby [15]. Recently M. Goldberg and Spencer [9] gave a deterministic algorithm for the problem that runs in polylogarithmic time using a linear number of processors.

The study of the maximal independent set problem shows the importance of techniques for breaking symmetry in parallel. The symmetry-breaking comes up in many other parallel algorithms as well. In many cases, however, it is enough to be able to break symmetry in special kinds of graphs. The performance of the resulting algorithm improves if we can solve the special case of symmetry-breaking more efficiently.

In this paper we present a technique for breaking symmetry. In particular, we give an $O(\lg^* n)$ time algorithm to 3-color a rooted tree. This techniques can be viewed as a generalization of the deterministic coin-flipping technique of Cole and Vishkin [5]. To show the usefulness of our technique, we present the following algorithms. All of the presented algorithms use a linear number of processors.

- For graphs whose maximum degree is a constant Δ, we give an $O(\Delta \lg \Delta \lg^* n)$ algorithm for $(\Delta+1)$-coloring and for finding a maximal independent set on an EREW PRAM.
- We give an algorithm to 7-color a planar graph. This algorithm, and the maximal independent set (for planar graphs) algorithm based on it, run in $O(\lg n \lg^* n)$ time on a CRCW PRAM and in $O(\lg^2 n)$ time on an EREW PRAM. We also give an $O(\lg^3 n \lg^* n)$ CRCW algorithm to 5-color a planar graph.
- We give a $O(\lg n \lg^* n)$ algorithm for finding a maximal matching in a planar graph on a CRCW PRAM.
- For general graphs we give an $O(\Delta^2 \lg n)$ algorithm for $(\Delta+1)$-coloring and for finding a maximal independent set on EREW PRAM.

The above stated results improve the running time and processor bounds for the respective problems. The fastest previously known algorithm for $(\Delta+1)$-coloring [15], in the case of constant-degree graphs, runs in $O(\lg n)$ time, and the deterministic version of this algorithm requires n^3 processors. The 5-coloring algorithm for planar graphs, due to Boyar and Karloff, [4] runs in $O(\lg^3 n)$ time, and the deterministic version of this algorithm requires n^3 processors. The $O(\lg^3 n)$ running time of the maximal matching algorithm due to Israeli and Shiloach [10] can be reduced to $O(\lg^2 n)$ in the restricted case of planar graphs, but our algorithm is faster.

Although in this paper we have limited ourselves to the application of our techniques for the design of of parallel algorithms for the PRAM model of computation, the same techniques can be applied in a distributed model of computation [1,7]. Moreover, the $O(\lg^* n)$ lower bound, given by Linial [14] for the maximal independent set problem on a chain in the distributed model, shows that our symmetry-breaking technique is optimal in this model.

The fact that a rooted tree can be 3-colored in $O(\lg^* n)$ time raises the question whether a rooted tree can be 2-colored within the same time complexity. We answer this question by giving an $\Omega(\lg n / \lg\lg n)$ lower bound for 2-coloring of a rooted tree.

We also present an $\Omega(\lg n / \lg\lg n)$ lower bound for finding a maximal independent set in a general graph, thus answering the question posed by Luby [15].

Some of the results presented here were obtained independently by Shannon [16].

2 Definitions and Notation

This sections describes the assumptions about the computational model, and introduces the notation used throughout the paper. In this paper we use n to denote the number of vertices and m to denote the number of edges in a graph. We use Δ to denote the maximum degree of the graph.

Given a graph $G = (V, E)$, we say that a subset of nodes $I \in V$ is *independent* if no two nodes in I are adjacent. A *coloring* of a graph G is an assignment $C : V \rightarrow N$ of positive integers (colors) to nodes of the graph. A coloring is *valid* if no two adjacent nodes have the same color. The i^{th} bit in the color of a node v is denoted by $C_v(i)$. A subset of edges $M \in E$ is a matching if any two distinct edges in M have no nodes in common.

The following problems are discussed in the paper:

- The vertex-coloring (VC) problem: find a valid coloring of a given graph that uses at most $\Delta+1$ colors.
- The maximal independent set (MIS) problem: find a maximal independent set of vertices in a given graph.
- The maximal matching (MM) problem: find a maximal matching in a given graph.

We make a distinction between *unrooted* and *rooted* trees. In a rooted tree, each nonroot node knows which of its neighbors is its parent.

The following notation is used:

$$\begin{aligned} \lg x &= \log_2 x \\ \lg^{(1)} x &= \lg x \\ \lg^{(i)} x &= \lg\lg^{(i-1)} x \\ \lg^* x &= \min\{i \mid \lg^{(i)} x \leq 2\} \end{aligned}$$

We assume a PRAM model of computation where each processor is capable of executing simple word and bit operations. The word width is assumed to be $O(\lg n)$. The word operations we use include bit-wise boolean operations, integer comparisons, and unary-to-binary conversion. In addition, we assume that each processor has a unique *identification number* $O(\lg n)$ bits wide, which we denote by PE-ID. We use exclusive-read, exclusive-write (EREW) PRAM, concurrent-read, exclusive-write (CREW) PRAM, and concurrent-read, concurrent-write (CRCW) PRAM as appropriate. All lower bounds are proven for a CRCW PRAM with a polynomial number of processors.

3 Coloring Rooted Trees

This section describes an $O(\lg^* n)$ time algorithm for 3-coloring rooted trees. First we describe an $O(\lg^* n)$ time algorithm for 6-coloring rooted trees. Then we show how to transform a 6-coloring of a rooted tree into 3-coloring in constant time.

The procedure *6-Color-Rooted-Tree* is shown in Figure 1. This procedure accepts a rooted tree $T = (V, E)$ and 6-colors it in time $O(\lg^* n)$. Starting from the valid coloring given by the processor ID's, the procedure iteratively reduces the number of bits in the color descriptions by recoloring each non-root node v with the color obtained by concatenating the index of a bit in which C_v differs from $C_{father(v)}$ and the value of this bit. The root r appends $C_r[0]$ to 0.

Theorem 1 *The algorithm* 6-Color-Rooted-Tree *produces a valid 6-coloring of a tree in* $O(\lg^* n)$ *time on a CREW PRAM using* $O(n)$ *processors.*

```
PROCEDURE 6-Color-Rooted-Tree
L ← ⌈lg n⌉
for all v ∈ V in parallel do C_v ← PE-ID(v)   ;;; initial coloring
while L > ⌈lg L + 1⌉ for all v ∈ V in parallel do
        if v is the root
          then do
                i_v ← 0
                b_v ← C_v(0)
                end
          else do
                i_v ← min{i | C_v(i) ≠ C_father(v)(i)}
                b_v ← C_v(i_v)
                end
        C_v ← b_v i_v
end
```

Figure 1: The Coloring Algorithm for Rooted Trees

Proof: First we prove by induction that the coloring computed by the algorithm is valid, and then we prove the upper bound on the execution time.

Assume that the coloring C is valid at the beginning of an iteration, and show that the coloring at the end of the iteration is also valid. Let v and w be two adjacent nodes; without loss of generality assume that v is the father of w. By the algorithm, w chooses some index i such that $C_v(i) \neq C_w(i)$ and v chooses some index j such that $C_v(j) \neq C_{father(v)}(j)$. The new color of w is $\langle i, C_w(i)\rangle$ and the new color of v is $\langle j, C_v(j)\rangle$. If $i \neq j$, the new colors are different and we are done. On the other hand, if $i = j$, then $C_v(i) \neq C_w(i)$ and again the colors are different. Hence, the validity of the coloring is preserved.

Now we show that the algorithm terminates after $O(\lg^* n)$ iterations. Let L_k denote the number of bits in the representation of colors after k iterations. For $k = 1$ we have

$$\begin{aligned} L_1 &= \lceil \lg L \rceil + 1 \\ &\leq 2\lceil \lg L \rceil \end{aligned}$$

if $\lceil \lg L \rceil \geq 1$.

Assume for some k we have $L_{k-1} \leq 2\lceil \lg^{(k-1)} L \rceil$ and $\lceil \lg^{(k)} L \rceil \geq 2$. Then

$$\begin{aligned} L_k &= \lceil \lg L_{k-1} \rceil + 1 \\ &\leq \lceil \lg(2 \lg^{(k-1)} L) \rceil + 1 \\ &\leq 2\lceil \lg^{(k)} L \rceil \end{aligned}$$

Therefore, as long as $\lceil \lg^{(k)} L \rceil \geq 2$,

$$L_k \leq 2\lceil \lg^{(k)} L \rceil.$$

Hence, the number of bits in the representation of colors L_k decreases until, after $O(\lg^* n)$ iterations, $\lceil \lg^{(k)} L \rceil$ becomes 1 and L_k reaches the value of 3 (the solution of $L = \lceil \lg L \rceil + 1$). Another iteration of the algorithm produces a 6-coloring: 3 possible values of the index i_v and 2 possible values of the bit b_v. The algorithm terminates at this point.

We use concurrent-read capability to broadcast the newly computed color C_v to all the sons of v; no concurrent-write capabilities are required. For constant-degree trees the concurrent-read capability is not needed either. ∎

As we have shown, a rooted tree can be 6-colored quickly. A natural question to ask at this point is whether one can use less colors and still stay within the same complexity bounds. The following theorem answers this question.

Theorem 2 *A rooted tree can be 3-colored in $O(\lg^* n)$ CREW PRAM time using $O(n)$ processors.*

Proof: The algorithm *3-Color-Rooted-Tree* presented in Figure 2 starts by using the previously described algorithm to 6-color the tree and then recolors it in 3 colors in constant time.

The algorithm recolors the nodes colored with *bad* colors 3, 4, and 5, into *good* colors 0, 1, 2 as follows. First, each node is recolored in the color of its father, so that any two nodes with the same father have the same color. The root, which

has no father, recolors itself with a color different from its current color. Next, the algorithm removes the color from every node that has a bad color and has a neighbor with a good color. These nodes become uncolored. Every node v that still has a color C_v is recolored in the color $C_v \bmod 3$; this gets rid of the remaining bad colors. Note that this coloring has the nice property that for any node v, all of the sons of v that are colored, must be colored identically.

The resulting coloring is valid, but not all nodes are colored. By the construction, every uncolored node has at least one colored neighbor. Therefore, if there are two nodes v and w, such that $v = \mathit{father}(w)$ and both nodes are uncolored, then $\mathit{father}(v)$ is colored and $\mathit{sons}(w)$ are colored too. The algorithm colors v with a color different from $C_{\mathit{sons}(v)}$ and from $C_{\mathit{father}(v)}$. Such a color always exists because there are 3 different colors to choose from and all the colored sons of v are colored with the same color. Finally, the algorithm colors w with a color different from both C_v and $C_{\mathit{sons}(w)}$. Every step of the *3-Color-Rooted-Tree* algorithm can be executed in constant time except for the first one, in which we color the tree with 6 colors. Hence, the total running time of the algorithm is $O(\lg^* n)$. ∎

Any tree can be 2-colored. In fact, it is easy to 2-color a tree in polylogarithmic time. For example, one can use treefix operations [13] to compute the distance from each node to the root, and color even level nodes with one color and odd level nodes with the other color. It is harder to find a 2-coloring of a rooted tree in parallel, however, than it is to find a 3-coloring of a rooted tree. In section 7 we show a lower bound of $\Omega(\lg n / \lg\lg n)$ on 2-coloring of a directed list by a CRCW PRAM with a polynomial number of processors, which implies the same lower bound for rooted trees.

4 Coloring Constant-Degree Graphs

The method for coloring rooted trees described in the previous section is a generalization of the deterministic coin-flipping technique described in [5]. The method can be generalized even further [8] to color constant-degree graphs in a constant

PROCEDURE *3-Color-Rooted-Tree*
$C \leftarrow$ 6-Color-Rooted-Tree (V, E)
for all $v \in V$, $v \neq$ root **in parallel do**
    $C_v \leftarrow C_{father(v)}$
end
$C_{root} \leftarrow \min\{\ \{0,1,2\} - \{C_{sons(root)}\}\ \}$
$V_1 \leftarrow \{v \mid C_v \leq 2\}$
$V_2 \leftarrow V - V_1$
$V' \leftarrow \{v \mid v \in V_2$ and $\exists (v,w) \in E, w \in V_1\}$;;; *bad*-colored nodes with *good*-colored neighbors
for all $v \in V - V'$ **in parallel do**
    $C_v \leftarrow C_v \bmod 3$
end
for all $v \in V'$ **in parallel do**
    $C_v \leftarrow$ uncolored
end
for all $v \in V'$ **in parallel do**
    if father(v) $\notin V'$
      then do
          $C_v \leftarrow \min\{\ \{0,1,2\} - \{C_{sons(v)}\} - \{C_{father(v)}\}\ \}$
          $V' \leftarrow V' - v$
      end
end
for all $v \in V'$ **in parallel do**
    $C_v \leftarrow \min\{\ \{0,1,2\} - \{C_{sons(v)}\} - \{C_{father(v)}\}\ \}$
end

Figure 2: The 3-coloring Algorithm for Rooted Trees

number of colors. In the generalized algorithm, a current color of a node is replaced by a new color obtained by looking at each neighbor, appending the index of a bit in which the current color of the node is different from the neighbor's color to the value of the bit in the node color, and concatenating the resulting strings. This algorithm runs in $O(\lg^* n)$ time, but the number of colors, although constant, is exponential in the degree of the graph.

In this section we show how to use the procedure *3-Color-Rooted-Tree* described in the previous section to color a constant-degree graph with $(\Delta+1)$ colors, where Δ is the maximum degree of the graph.

First, we describe how to find in constant time a forest in a given graph such that each node with nonzero degree in the graph has nonzero degree in the forest. The removal of the edges of the forest decreases the maximum degree of the remaining graph (unless the maximum degree of the graph is zero). We shall use this property later use to decompose the edges into Δ sets, each set inducing a forest on the nodes of the graph. The procedure *Find-Forest* (see Figure 3) constructs such a forest.

The procedure has two steps. In the first step each node compares the ID's of its neighbors with its own ID. A node that does not have the maximum processor ID among its neighbors chooses an edge that connects it to the neighbor with the largest processor ID. The graph induced by the chosen edges is a forest (the graph has no cycles) and the nodes with the highest processor IDs among their neighbors – local maximums – are roots of the forest. In the second step each root with no sons chooses an edge that connects it to one of its neighbors. The roots are local maximums and are therefore independent. Hence, no new cycles are introduced into the graph induced by the chosen edges.

The algorithm *Color-Constant-Degree-Graph* that colors constant-degree graph with $(\Delta+1)$ colors is presented in Figure 4. The algorithm consists of two phases. In the first phase we iteratively call the *Find-Forest* procedure, each time removing the edges of the constructed forest. This phase continues until no edges remain, At which point we color all the nodes with one color.

In the second phase we iteratively return the edges of the forests into the graph,

```
PROCEDURE Find-Forest(V,E)
E' ← ∅
R ← ∅
for all v ∈ V in parallel do   ;;; construct the forest – the first step
    if PE-ID(v) is not a local maximum
      then do
            e_v ← (v, w) s.t. (v, w) ∈ E and PE-ID(w) = max{PE-ID(u)|(v, u) ∈ E}
            E' ← E' ∪ e_v
      end
      else do
            R ← R ∪ v
      end
end
for all v ∈ R in parallel do   ;;; get rid of zero-depth trees – the second step
    if ∄(v, w) ∈ E' and ∃(v, w') ∈ E
      then do
            E' ← E' ∪ (v, w')
      end
end
return (E')  ;;; the edges of the forest
```

Figure 3: The Spanning Forest Algorithm

```
PROCEDURE Color-Constant-Degree-Graph
E' ← E
i ← 0
while E' ≠ ∅ do   ;;; the first phase
        E_i ← Find-Forest(V, E')
        E' ← E' − E_i
        i ← i + 1
end
for all v ∈ V in parallel do   ;;; initial coloring
     C(v) ← 1
end
for i ← i − 1 to 0 do   ;;; the second phase
     C' ← 3-Color-Rooted-Tree (V, E_i)
     E' ← E' + E_i
     for k ← 1 to 3 do
          for j ← 1 to Δ+1 do
               V' ← V
               for all v ∈ V' in parallel do
                    if C(v) = j and C'(v) = k
                      then do
                              C(v) ← max {{1, 2, ...Δ+1} − {C(w) | (v, w) ∈ E'}}
                              V' ← V' − v
                      end
               end
          end
     end
end
```

Figure 4: The Recoloring Algorithm for Constant Degree Graphs

each time recoloring the nodes to maintain a consistent coloring. At the beginning of each iteration of this phase, the edges of the current forest (E') are added, making the existing $(\Delta+1)$-coloring inconsistent. This forest is colored with 3 colors using the *3-Color-Rooted-Tree* procedure. Now, each node has two colors – one from the coloring at the previous iteration and one from the coloring of the forest. The pairs of colors form a valid $3(\Delta+1)$-coloring of the graph. The iteration finishes by enumerating the color classes, recoloring each node of the current color with a color from $\{0, ..., \Delta\}$ that is different from the colors of its neighbors (note that we can recolor all the nodes of the same color in parallel because they are independent).

Theorem 3 *The algorithm* Color-Constant-Degree-Graph *runs in* $O(\Delta \lg \Delta(\Delta + \lg^* n))$ *time and colors the graph with* $(\Delta+1)$ *colors.*

Proof: At each iteration all edges of the spanning forest are removed. From the above discussion it follows that each node that still has neighbors in the beginning of an iteration, has at least one edge removed during that iteration, and therefore its degree decreases. Hence, the first phase of the algorithm terminates in at most Δ iterations.

The second phase terminates in at most Δ iterations as well. Each iteration consists of two stages. First, the current forest is colored using procedure *3-Color-Rooted-Tree*, which takes, by theorem 2, $O(\lg \Delta \lg^* n)$ time on an EREW PRAM (the $\lg \Delta$ factor appears because we do not use the concurrent-read capability). Now we iterate over all the colors. Since in this section we assume that Δ is a constant, each iteration can be done in $O(\lg \Delta)$ time using word operations. Hence, one iteration of the second phase takes $O(\lg \Delta \lg^* n + \Delta \lg \Delta)$ time, leading to an overall $O(\Delta \lg \Delta(\Delta + \lg^* n))$ running time on an EREW PRAM. ∎

Having a $(\Delta+1)$-coloring of a graph enables us to find an MIS in this graph. The following theorem states this fact formally. (We refer to the algorithm described in the proof as *Constant-Degree-MIS* in the subsequent sections.)

Theorem 4 *An MIS in constant-degree graphs can be found in $O(\lg^* n)$ time on an EREW PRAM using $O(n)$ processors.*

Proof: After coloring the graph in a constant number of colors using the procedure *Color-Constant-Degree-Graph*, one can find an MIS by iterating over the colors, taking all the remaining nodes of the current color, adding them to the independent set, and removing them and all their neighbors from the graph. By theorem 3, the coloring of a constant-degree graph takes $O(\lg^* n)$ time on an EREW PRAM. The selection of all nodes with a specific color and the removal of all neighbors of the selected nodes takes constant time. ∎

The proofs of theorems 3 and 4 also imply that the algorithms *Color-Constant-Degree-Graph* and *Constant-Degree-MIS* have a polylogarithmic running times for graphs with polylogarithmic maximum degrees. However, in this case the assumption that the word size is greater then Δ is unreasonable, so the running time of the algorithms becomes $O(\Delta(\Delta^2 + \lg \Delta \lg^* n))$. In section 6 we present an algorithm with better performance for $\Delta = \omega(\lg n)$.

The above algorithms can be implemented in the distributed model of computation [1,7], where processors have fixed connections determined by the input graph. The algorithms in the distributed model achieve the same $O(\lg^* n)$ bound as in the EREW PRAM model. Linial has recently shown [14] that $\Omega(\lg^* n)$ time is required in the distributed model to find a maximal independent set on a chain. Our algorithms are therefore optimal (to within a constant factor) in the distributed model.

5 Algorithms for Planar Graphs

Any planar graph can be 4-colored. However, linear time sequential algorithms are known only for 5-coloring planar graphs. In this section we describe a simple and efficient parallel algorithm that 7-colors a planar graph, and show how to construct a more complicated parallel algorithm to 5-color a planar graph.

```
PROCEDURE 7-Color-Planar-Graph
V' ← V
V_1, V_2, ... V_{lg n} ← ∅
i ← 0
while V' ≠ ∅ for all v ∈ V' do in parallel ;;; first stage
      if Degree(v) ≤ 6
        then do
               V_i ← V_i + v
               V' ← V' − v
        end
      i ← i + 1
end
for  i ← i − 1 to 0 do ;;; second stage
     while V_i ≠ ∅ do
           E_i ← {(v, w) | v, w ∈ V_i ; (v, w) ∈ E}
           I ← Constant-Degree-MIS(V_i, E_i)
           for  all v ∈ I do in parallel
                C_v ← max{{1 ... 7} − {C_w | w ∈ V'; (v, w) ∈ E} }
                end
           V' ← V' + I
           V_i ← V_i − I
     end
end
```

Figure 5: The 7-Coloring Algorithm For Planar Graphs

First we describe an algorithm for 7-coloring of planar graphs. The algorithm, called *7-Color-Planar-Graph*, is shown in Figure 5. The algorithm consists of two stages. In the first stage, we iteratively partition the vertices of the graph into layers. At each iteration we create a new layer consisting of all nodes of the graph with degree 6 or less and delete these nodes from the graph.

The second stage returns the layers to the graph in the order opposite to the order in which the layers are removed. After a layer is returned, it is 7-colored in the way consistent with the coloring of the layers which have been returned and colored in the previous iterations. Note that all the nodes of the returned layer have a degree of at most 6 in the current graph.

The layer is colored by iteratively applying the *Constant-Degree-MIS* procedure to find an MIS in the subgraph induced by the uncolored nodes of the layer, and coloring each of the selected nodes in a color different from its colored neighbors. Since the uncolored nodes have a degree of at most 6 in the current graph, we never need more than 7 colors.

Theorem 5 *The algorithm* 7-Color-Planar-Graph *runs in* $O(\lg n \lg^* n)$ *time on a CRCW PRAM and in* $O(\lg^2 n)$ *time on an EREW PRAM.*

Proof: In a planar graph, at least a constant fraction ($1/7^{\text{th}}$) of nodes have a degree less or equal to 6, and therefore the first stage of the *7-Color-Planar-Graph* algorithm terminates in at most $O(\lg n)$ steps. At each step we have to identify the nodes that have degree less than 7 in the remaining graph. This takes constant time on a CRCW PRAM (assuming that if two or more processors simultaneously write into some location, one of them will succeed) and $O(\lg n)$ time on an EREW PRAM.

In the second stage all the uncolored nodes are of degree less or equal to 6 and therefore, by theorem 4, the procedure *Constant-Degree-MIS* finds, in $O(\lg^* n)$ time, an MIS in the graph induced by these nodes. By the definition of the maximal independent set, when the algorithm colors the MIS, at least one uncolored neighbor of each uncolored node becomes colored. Therefore the second part of the second

stage terminates in at most 7 iterations.

Since the first stage takes $O(\lg n)$ time on a CRCW PRAM and $O(\lg^2 n)$ time on an EREW PRAM, and since each one of the $O(\lg n)$ iterations of the second stage is dominated by a call to *Constant-Degree-MIS*, the total running time is $O(\lg n \lg^* n)$ on a CRCW PRAM and $O(\lg^2 n)$ on an EREW PRAM. ∎

Remark: If, at each stage, instead of removing from the graph all the nodes with degree less than 6, we remove all the nodes with degree less or equal to the average degree, the algorithm described above produces a correct result in polylogarithmic time for any graph G such that the average degree of any node-induced subgraph G' of G is polylogarithmic in the size of G'. This class contains many important subclasses including graphs that are unions of a polylogarithmic number of planar graphs (i.e. graphs with polylogarithmic thickness).

Our techniques together with the ideas presented in [4] can be used to construct a deterministic $O(\log^3 n \lg^* n)$ time algorithm for 5-coloring a planar graph.

The 5-coloring algorithm has two stages. The first stage of the algorithm partitions the graph into layers such that vertices in any layer are independent and have degree of at most 6 in the graph induced by the vertices in its layer and the higher numbered layers. The second stage of the algorithm adds layers one by one, starting from the layer with the highest number, each time recoloring the graph with 5 colors.

Before describing the second stage, we need the following definitions. Let G be a partially colored graph and let c_1 and c_2 be two distinct colors. A *color component* is a connected component of a subgraph of G induced by all vertices of color c_1 and c_2. A *color component flip* is a recoloring of the color component that exchanges colors c_1 and c_2. A color component flip does not affect the validity of coloring.

We can proceed with the description of the second stage of the algorithm. After a layer is added to already colored graph, we first color all vertices that can be colored without changing the existing coloring. This can be done in the same way as in the 7-coloring algorithm. Now all 5 colors are represented among neighbors of each

uncolored vertex. Since the uncolored vertices have degree of at most 6, the results of [4] imply that for every uncolored vertex v there are two colors c_1 and c_2 such that v has exactly one neighbor w_1 of color c_1 and exactly one neighbor w_2 of color c_2. Furthermore, the vertices w_1 and w_2 belong to different color components induced by colors c_1 and c_2. Flipping each one of these color component allows us to color v. The problem is, however, that flipping both color components simultaneously does not allow us to color v. We call such color components *dependent*.

Where as Boyar and Karloff use randomness to deal with this problem, we use our symmetry-breaking techniques as follows. For each pair of distinct colors c_1 and c_2, we construct color components induced by these colors. Then we construct a *dependency graph* with vertices corresponding to the color components and edges corresponding to the dependencies between the color components. Flipping a set of color components that corresponds to an independent set in the dependency graph does not cause conflicts. Suppose we can find an independent set in the dependency graph such that flipping the corresponding set of color components allows us to color a constant set of uncolored vertices. Then in $O(\log n)$ iterations will be able to color all uncolored vertices.

We find such an independent set in the dependency graph as follows. Observe that the dependency graph is planar, so we can 7-color this graph using the 7-Color-Planar-Graph algorithm. Then, for each pair of distinct colors and for each color class of the corresponding dependency graph, we compute the number of uncolored vertices of the original graph which can be colored if the color components corresponding to vertices in the color class are flipped. For each of the 10 possible choices of colors c_1 and c_2 there are 7 color classes, so the total number of times that we count the number of vertices that can be colored if a color class is flipped is 70. Since each uncolored vertex is counted at least once, there is a color class such that flipping all color components in this class allows us to color at least $1/70$ uncolored vertices.

Next we analyze to complexity of the algorithm. The outer loop of the algorithm that iterates over layers is executed $O(\log n)$ times, and the inner loop that colors a constant fraction of uncolored vertices is executed $O(\log n)$ times as well.

Each iteration of the inner loop does 10 connected component computations, 70 enumeration and 10 calls to the 7-Color-Planar-Graph procedure. Since each connected component computation can be done in $O(\log n)$ time on CRCW PRAM using Shiloach-Vishkin algorithm [17], the 7-Color-Planar-Graph procedure is the bottleneck of the inner loop (recall that it runs in $O(\log n \lg^* n)$ time). The overall running time of the algorithm is $O(\log^3 n \lg^* n)$.

The above result is summarized in the following theorem.

Theorem 6 *A planar graph can be 5-colored in $O(\lg^3 n \lg^* n)$ time on a CRCW PRAM using $O(n)$ processors.*

Using the techniques described in this paper it is easy to construct a fast algorithm for finding a maximal matching in planar graph.

Theorem 7 *A maximal matching in planar graph can be found in $O(\lg n \lg^* n)$ time on a CRCW PRAM.*

Proof: First, the algorithm partitions the graph into layers, such that the nodes in a layer are of degree less than 7 in the graph induced by the nodes of this layer and the nodes in the higher-numbered layers. The algorithm proceeds by iteratively returning a layer, finding a maximal matching in the obtained graph, and removing the end-points of the edges in the matching. At the end of each iteration the remaining nodes induce a graph of degree zero and therefore at the beginning of each iteration the maximum degree of the induced graph is 6. Hence, a maximal matching in this graph can be found in $O(\lg^* n)$ time by finding a maximal independent set in the line-graph, which also has a constant maximum degree. Each iteration takes $O(\lg^* n)$ time on a CRCW PRAM and the number of iterations is $O(\lg n)$. This gives $O(\lg n \lg^* n)$ total running time. ∎

6 Coloring Polylogarithmic Degree Graphs

This section describes a coloring algorithm for graphs with maximum degree which is polylogarithmic in the size of the graph. For $\Delta = \omega(\lg n)$, this algorithm has a better performance than the algorithm *Color-Constant-Degree-Graph* described above.

The *Poly-Log-Color* algorithm is shown in Figure 6 and works as follows. First, the graph is partitioned into two subgraphs with approximately equal number of nodes, and the subgraphs are recursively colored in $\Delta+1$ colors. Then we iterate through all the colors of one of the subgraphs, recoloring each node with a color different from the colors of all of its neighbors.

Theorem 8 *The algorithm* Poly-Log-Color *colors a graph with a maximum degree of* Δ *with* $\Delta+1$ *colors in* $O(\Delta^2 \lg n)$ *time.*

Proof: Each time the graph is partitioned into two subgraphs with approximately equal number of nodes and therefore the depth of recursion is $O(\lg n)$. At each recursion level we iterate through all the colors, each iteration dominated by the time to find a color different from the colors of all the neighbors of a node, which takes $O(\Delta)$ time. Hence the total time is $O(\Delta^2 \lg n)$ on a EREW PRAM. ∎

After coloring the graph in $\Delta+1$ colors we can construct an MIS of the graph in $O(\Delta^2)$ time. Hence, an MIS of a graph with a polylogarithmic maximum degree can be found in $O(\Delta^2 \lg n)$ time on EREW PRAM using a linear number of processors.

7 Lower Bounds

In this section we prove two lower bounds for a CRCW PRAM with polynomial number of processors:

- Finding a MIS in a general graph takes $\Omega(\lg n / \lg\lg n)$ time.

```
PROCEDURE Poly-Log-Color (V, E)
partition V into V_r,V_l such that V_r ∪ V_l = V
E_r ← {(v, w) | (v, w) ∈ E; v, w ∈ V_r}
E_l ← {(v, w) | (v, w) ∈ E; v, w ∈ V_l}
C_r ←Poly-Log-Color(V_r, E_r)
C_l ←Poly-Log-Color(V_l, E_l)
V' ← ∅
for all v ∈ V_l in parallel do
    if ∃(v, w) ∈ E such that v ∈ V_l, w ∈ V_r and C_l(v) = C_r(w)
      then do
            V' ← V' ∪ v
      end
for j ← 1 to Δ+1 do
    for all v ∈ V' in parallel do
        if C_l(v) = j
          then do
                C_l(v) ← max {{1, 2, ...Δ+1} − {C(w) | (v, w) ∈ E'}}
                V' ← V' − v
          end
    end
end
```

Figure 6: The Coloring Algorithm for Polylogarithmic Maximum Degree Graphs

- 2-coloring a directed list takes $\Omega(\lg n / \lg\lg n)$ time.

The first lower bound complements the $O(\lg n)$ CRCW PRAM upper bound for the MIS problem that is achieved by Luby's algorithm [15]. The second lower bound complements Theorem 2 in this paper.

Theorem 9 *The running time of any MIS algorithm on a CRCW PRAM with a polynomial number of processors is* $\Omega(\lg n / \lg\lg n)$.

Proof: Given an instance of MAJORITY, we construct an instance of MIS in constant CRCW PRAM time. MAJORITY is harder that PARITY [6], which was proven to take $\Omega(\lg n / \lg\lg n)$ on a CRCW PRAM in [2,3]. Therefore the lower bound claimed in the theorem follows.

Let $x_1, x_2, \ldots, x_n$ be an instance of MAJORITY. We construct a complete bipartite graph $G = (V, E)$ with nodes corresponding to '0' bits of the input on one side and nodes corresponding to '1' bits on the other side.

$$\begin{aligned} V &= \{1, \ldots, n\} \\ E &= \{(i,j) \mid x_i \neq x_j\} \end{aligned}$$

To construct this graph, assign a processor P_{ij} for each pair $1 \leq i < j \leq n$. Then, each processor P_{ij} writes 1 into location M_{ij} if $x_i \neq x_j$ and 0 otherwise.

A maximal matching in a complete bipartite graph is also a maximum one. By constructing a maximal independent set in the line-graph G' of G, one can find a maximal matching in G. To construct the graph G' assign a processor P_{ijk} for each distinct $i, j, k \leq n$. Each P_{ijk} writes 1 into location $M_{(i,j),(j,k)}$ if $M_{ij} = M_{jk} = 1$ and 0 otherwise.

The MAJORITY equals to 1 if and only if there is an unmatched node $i \in G$ such that $x_i = 1$, which can be checked on a CRCW PRAM in constant time. ∎

Theorem 10 *The time to 2-color a directed list on a CRCW PRAM with a polynomial number of processors is* $\Omega(\lg n / \lg\lg n)$.

Proof: We show a constant time reduction from PARITY to the 2-coloring of a directed list. First, we show how to construct, in constant time, a directed list with elements corresponding to all the input bits x_i with value of 1. Let $x_1, x_2, \ldots, x_n$ be an instance of PARITY. Associate a processor P_i with each input cell M_i that initially holds the value of x_i. Associate a set of processors P_i^{jk} with each index i, $1 \leq k \leq j < i$. In one step, each processor P_i^{jk} reads the value of M_k and, if it equals to 1, writes 1 into M_i^j, effectively computing the OR-function on the input values $x_{i-j}, x_{i-j+1}, \ldots, x_{i-1}$. Assign a processor P_i^j to each M_i^j. Each processor P_i^j reads M_i^j and M_i^{j+1} and writes j into M_i' if and only if $M_i^j \neq M_i^{j+1}$. It can be seen that for all $0 \leq i \leq n$, M_i' holds $\max\{j \mid j < i, x_j = 1\}$.

We have constructed a directed list with elements corresponding to all the input bits x_i with value of 1. Assume this list is 2-colored. Then PARITY equals to 1 if and only if both ends of the list are colored in the same color, which can be checked in constant time. ∎

8 Acknowledgments

We would like to thank Charles Leiserson and David Shmoys for fruitful and stimulating discussions, and for their valuable comments on a draft of this paper.

References

[1] B. Awerbuch. Complexity of network syncronization. *Journal of the Association for Computing Machinery*, 32(4):804–823, October 1985.

[2] P. Beame. *Lower Bounds in Parallel Machine Computation.* PhD thesis, University of Toronto, 1986.

[3] P. Beame and J. Hastad. Personal communication. 1986.

[4] J. Boyar and H. Karloff. Coloring planar graphs in parallel. 1986. Unpublished Manuscript.

[5] R. Cole and U. Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. In *Proc. 18th ACM Simp. on Theory of Computing*, pages 206–219, 1986.

[6] M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial time hierarchy. In *Proc. 22nd IEEE Conf on Foundations of Computer Science*, pages 260–270, 1981.

[7] R. G. Gallager, P. A. Humblet, and P. M Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, January 1983.

[8] A. Goldberg and S. Plotkin. Parallel $(\Delta+1)$ coloring of constant-degree graphs. *Information Processing Letters*, 1986. Accepted for publication.

[9] M. Goldberg and T. Spencer. A new parallel algorithm for the maximal independent set problem. 1986. Submitted for publication.

[10] A. Israeli and Y. Shiloach. An improved parallel algorithm for maximal matching. *Information Processing Letters*, 22:57–60, January 1986.

[11] H. J. Karloff. *Fast Parallel Algorithms for Graph-Theoretic Problems: Matching, Coloring, Partitioning.* PhD thesis, University of California, Berkeley, 1985.

[12] R. M. Karp and A. Wigderson. A fast parallel algorithm for the maximal independent set problem. In *Proc. 16th ACM Simp. on Theory of Computing*, pages 266–272, 1984.

[13] C. Leiserson and B. Maggs. Communication-efficient parallel graph algorithms. In *Proc. of International Conference on Parallel Processing*, pages 861–868, 1986.

[14] N. Linial. Personal communication. 1986.

[15] M. Luby. A simple parallel algorithm for the maximal independent set problem. In *Proc. 17th ACM Simp. on Theory of Computing*, pages 1–10, 1985.

[16] G. Shannon. Reduction techniques for designing linear-processor parallel algorithms on sparse graphs. 1986. In prepapation.

[17] Y. Shiloach and U. Vishkin. An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms*, 3:57–67, 1982.

[18] L. G. Valiant. Parallel computation. In *Proc. 7th IBM Simp. on Mathemetical Foundations of Computer Science*, 1982.