Combinatorial Algorithms for Distributed Graph Coloring

Leonid Barenboim* and Michael Elkin**

Department of Computer Science, Ben-Gurion University of the Negev, POB 653, Beer-Sheva 84105, Israel.
{leonidba,elkinm}@cs.bgu.ac.il

Abstract. Numerous problems in Theoretical Computer Science can be solved very efficiently using powerful algebraic constructions. Computing shortest paths, constructing expanders, and proving the PCP Theorem, are just a few examples of this phenomenon. The quest for combinatorial algorithms that do not use heavy algebraic machinery, but have the same (or better) efficiency has become a central field of study in this area. Combinatorial algorithms are often simpler than their algebraic counterparts. Moreover, in many cases, combinatorial algorithms and proofs provide additional understanding of studied problems. In this paper we initiate the study of combinatorial algorithms for Distributed Graph Coloring problems. In a distributed setting a communication network is modeled by a graph $G = (V, E)$ of maximum degree Δ. The vertices of G host the processors, and communication is performed over the edges of G. The goal of distributed vertex coloring is to color V with $(\Delta + 1)$ colors such that any two neighbors are colored with distinct colors. Currently, efficient algorithms for vertex coloring that require $O(\Delta + \log^* n)$ time are based on the algebraic algorithm of Linial [18] that employs set-systems. The best currently-known combinatorial set-system free algorithm, due to Goldberg, Plotkin, and Shannon [14], requires $O(\Delta^2 + \log^* n)$ time. We significantly improve over this by devising a *combinatorial* $(\Delta + 1)$-coloring algorithm that runs in $O(\Delta + \log^* n)$ time. This exactly matches the running time of the best-known *algebraic* algorithm. In addition, we devise a tradeoff for computing $O(\Delta \cdot t)$-coloring in $O(\Delta/t + \log^* n)$ time, for almost the entire range $1 < t < \Delta$. We also compute a Maximal Independent Set in $O(\Delta + \log^* n)$ time on general graphs, and in $O(\log n / \log\log n)$ time on graphs of bounded arboricity. Prior to our work, these results could be only achieved using algebraic techniques. We believe that our algorithms are more suitable for real-life networks with limited resources, such as sensor, ad-hoc, and mobile networks.

* Supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

** This research has been supported by the Binational Science Foundation, grant No. 2008390. Additional funding was provided by the Lynne and William Frankel Center for Computer Sciences.

1 Introduction

1.1 Algebraic versus Combinatorial Algorithms

It is a common scenario in Theoretic Computer Science that very strong results are achieved by powerful non-combinatorial techniques. In many occasions consequent research focuses on devising *combinatorial* counterparts to these results. This quest for combinatorial algorithms is often justified by the desire to obtain better understanding of the problem at hand. In some cases it also leads to more efficient and simple algorithms.

One notable example of such development is the celebrated *PCP theorem*. This famous result was achieved [3, 12] by algebraic techniques. Recently a significant research effort was invested in devising a combinatorial proof for this result [10, 21]. Another important example is *expanders* and *Ramanujan* graphs. Near-optimal algebraic constructions of expanders were devised in [1, 19]. Reingold, Vadhan and Wigderson [26] devised the first *combinatorial* construction of expanders. Though the parameters of these combinatorial constructions are somewhat inferior to algebraic ones, the techniques that were developed as a part of this effort turned out to be useful for devising a log-space S-T-connectivity algorithm [25]. Also, consequently to the work of [26], improved combinatorial constructions of expanders and near-Ramanujan graphs were devised in [8].

This phenomenon occurs also in *approximation* algorithms. Linear and semidefinite programming is an extremely powerful algebraic technique in this area. However, it is an active line of research to explore how well one can do *without linear programming*. Yet another example of this phenomenon is algorithms for computing (almost) *shortest paths*. Very efficient algorithms for this problem were achieved about twenty years ago via fast matrix multiplication. Recently, combinatorial algorithms for this problem were devised, which, in some scenarios, outperform their algebric counterparts.

1.2 Distributed Coloring

We study the *distributed coloring* problem. We are given an n-vertex unweighted undirected graph $G = (V, E)$, with each vertex $v \in V$ hosting a processor. The processors share no common memory. They communicate with each other by sending short messages (of size $O(\log n)$ each) over the edges of E. The communication is synchronous, i.e., it occurs in discrete rounds. All vertices wake up simultaneously. Each vertex $v \in V$ has a unique identity number ($Id(v)$). For simplicity we assume that all identifiers are from the range $\{1, 2, ..., n\}$. All algorithms extend to larger ranges of identifiers.

Denote by Δ the maximum degree of G. In the $(\Delta+1)$-*coloring* problem the objective is to color G with $\Delta+1$ colors *legally*, i.e., in such a way that for every edge $e = (u, v)$, the endpoints u and v will get distinct colors. The *running time* of an algorithm is the number of rounds that elapse until all vertices compute their final colors. Another closely related problem is the Maximal Independent Set (henceforth, MIS) problem. In this problem we want to compute a subset $U \subseteq V$ of independent vertices (i.e., for every $u, u' \in U$, there is no edge (u, u') in the graph) with the property that for every vertex $v \in V \setminus U$, there exists a neighbor $u \in U$ of v.

These two problems are widely considered to be among the most fundamental distributed problems. Recently, a significant progress was achieved in devising deterministic distributed algorithms for them. Specifically, the authors of the current paper [6], and independently Kuhn [16], devised a $(\Delta+1)$-coloring algorithm with running time $O(\Delta+\log^* n)$. These algorithms also directly give rise to algorithms that compute MIS within the same time. Both papers [6, 16] also devised a tradeoff, and showed that for any constant ϵ, $0<\epsilon<1$, a $\Delta^{1+\epsilon}$-coloring can be computed in $O(\Delta^{1-\epsilon}+\log^* n)$ time. (The results in [16] are, in fact, even more general than this; they show that for any parameter t, $1 \leq t \leq \Delta$, a Δ/t-coloring can be computed in $O(\Delta \cdot t+\log^* n)$ time.) Finally, on graphs of small *arboricity* [1], the authors of the current paper devised in [5] an algorithm that computes $(\Delta+1)$-coloring and MIS in $O(\frac{\log n}{\log\log n})$ time.

All these results rely heavily on an algorithm of Linial [18], that computes an $O(\Delta^2)$-coloring within $\log^* n+O(1)$ time. The latter seminal result relies, in turn, on an algebraic construction of Erdős, Frankl and Füredi [11] of set-systems with certain special and very useful properties. Moreover, the algorithm of Kuhn [16] takes this algebraic technique one step further, and devises an algebraic construction of sets of functions that are tailored for the needs of his coloring algorithm. We remark also that even previous to the work of [6, 16] $(\Delta+1)$-coloring algorithms relied on algebraic techniques. Specifically, the $(\Delta+1)$-coloring and MIS algorithms of Kuhn and Wattenhofer [17] that require $O(\Delta\log\Delta+\log^* n)$ time rely on Linial's algorithm. To the best of our knowledge, the best currently known deterministic algorithm of this type that does not rely on Linial's method is the algorithm due to Goldberg, Plotkin and Shannon [14]. The latter algorithm requires, however, $O(\Delta^2+\log^* n)$ time.

The basic question that we investigate in the current paper is whether algebraic techniques are indeed necessary for devising efficient deterministic coloring and MIS algorithms. We demonstrate that it is not the case, and devise *combinatorial* (we also call them *set-system free*) coloring and MIS algorithms whose performance matches the state-of-the-art. Specifically, one of our new combinatorial algorithms computes a $(\Delta+1)$-coloring and MIS within $O(\Delta+\log^* n)$ time, Another one provides a tradeoff and computes a $\Delta^{1+\epsilon}$-coloring in $O(\Delta^{1-\epsilon}+\log^* n)$ time, for any constant ϵ, $0<\epsilon<1$. We also devise combinatorial $(\Delta+1)$-coloring and MIS algorithms for graphs of small arboricity that run in $O(\frac{\log n}{\log\log n})$ time.

We believe that the value of these results is two-fold. First, it addresses the aforementioned question, and shows that like in the context of PCP and expanders, one also can get rid of algebraic techniques in the context of distributed deterministic coloring. By this our algorithms seem to uncover a new understanding of the nature of the explored problems. Second, since our algorithms are combinatorial, they are much easier for implementation by not-very-mathematically-inclined programmers. In addition, the combinatorial nature of our algorithms enables for more efficient implementation in terms of local computation. While the latter is usually suppressed when analyzing distributed al-

[1] See Section 2 for its definition.

gorithms, it may become crucial when running the algorithms on certain simple real-world communication devices, such as sensors or antennas.

1.3 Our Techniques

The most tempting approach to the problem of devising combinatorial coloring algorithms is to devise a combinatorial counterpart to Linial's algorithm. However, we were not able to accomplish this. Instead we observe that some of the aforementioned coloring algorithms [5, 6] start with computing an $O(\Delta^2)$-coloring via Linial's algorithm, and then employ this coloring in a way that can be relatively easily made combinatorial. Our new algorithms invoke neither the algorithm of Linial itself, nor a combinatorial analogue of it. Instead, in our algorithms we start with partitioning the edge set of the graph into Δ forests (using Panconesi-Rizzi algorithm [22]). We then color each forest separately, and combine colorings of pairs of forest. In this way we obtain an $O(\Delta^2)$-coloring of each of the $\Delta/2$ pairs of forests. Next, we employ combinatorial algorithms to manipulate the colorings in each of these pairs of forests, and to obtain, rougly speaking, a $(\Delta+1)$-coloring for each pair. We then merge these pairs again, and manipulate the resulting coloring again. We iterate in this way up until we get a unified coloring for the entire graph. Obviously, this schematic outline suppresses many technical details. Also, this scheme does not achieve our best results, which we obtain by using more sophisticated combinatorial methods. Nevertheless, it seems to capture the basic ideas behind our combinatorial algorithms.

1.4 Related Work

Goldberg et al. [13] (based on [9]) devised $(\Delta+1)$-coloring and MIS algorithms that require $O(\Delta^2+\log^* n)$ time. Linial [18] stregthened this result, and showed that an $O(\Delta^2)$-coloring can be computed in $\log^* n + O(1)$ time. Kuhn and Wattenhofer [17] improved the running time of [13] to $O(\Delta \log \Delta + \log^* n)$. The latter was further improved to $O(\Delta + \log^* n)$ in [6, 16]. On graphs of arboricity $a \leq \log^{1/2-\epsilon} n$, for a constant $\epsilon > 0$, [5] devised $(\Delta+1)$-coloring and MIS algorithms that require $O(\frac{\log n}{\log\log n})$ time. A variety of additional algorithms and tradeoffs for coloring graphs of small arboricity were devised in [5, 7].

Awerbuch, Goldberg, Luby and Plotkin [4] devised deterministic $(\Delta+1)$-coloring and MIS algorithms that require $2^{O(\sqrt{\log n \log\log n})}$ time. The latter was improved to $2^{O(\sqrt{\log n})}$ by Panconesi and Srinivasan [23]. Randomized algorithms with logarithmic running time were devised by Luby [20], and by Alon, Babai, and Itai [2]. Kothapalli et al. [15] showed that an $O(\Delta)$-coloring can be computed in $O(\sqrt{\log n})$ randomized time. Recently, Schneider and Wattenhofer [27] showed that $(\Delta+1)$-coloring can be computed in randomized $O(\log\Delta + \sqrt{\log n})$ time. They have also showed a tradeoff between the number of colors and running time with very efficient algorithms for $O(\Delta + \log n)$-coloring.

2 Preliminaries

2.1 Definitions and notation

Unless the base value is specified, all logarithms in this paper are to base 2. The *degree* of a vertex v in a graph $G = (V, E)$, denoted *deg(v)*, is the number of

edges incident to v. A vertex u such that $(u,v) \in E$ is called a *neighbor* of v in G. The maximum degree of a vertex in G, denoted $\Delta = \Delta(G)$, is defined by $\Delta(G) = \max_{v \in V} deg(v)$. A *forest* is an acyclic subgraph. A *tree* is a connected acyclic subgraph. A forest $\mathcal{F}$ can also be represented as a collection of vertex-disjoint tress $\mathcal{F} = \{T_1, T_2, ..., T_k\}$. A tree T is said to be oriented if (1) there is a designated *root* vertex $rt \in V(T)$, (2) every vertex $v \in V(T)$ knows whether v is the root or not, and, in the latter case, v knows which of its neighbors is the parent $\pi(v)$ of v. (The parent $\pi(v)$ of v is the unique neighbor of v that lies on the (unique) path in T connecting v with rt.) A forest $\mathcal{F} = \{T_1, T_2, ..., T_k\}$ is said to be *oriented* if each of the trees $T_1, T_2, ..., T_k$ is oriented. A *Forest-Decomposition* of a graph $G = (V, E)$ is an edge partition such that each subgraph forms an oriented forest. The *arboricity* of a graph G is the minimal number a such that the edge set of G can be covered with at most a edge disjoint forests.
A mapping $\varphi : V \to \mathbb{N}$ is called a *coloring*. A coloring that satisfies $\varphi(v) \neq \varphi(u)$ for each edge $(u,v) \in E$ is called a *legal coloring*. For a positive integer k, a k-coloring φ is a legal coloring that employs at most k colors, i.e., for each vertex v, $\varphi(v) \in \{1, 2, ..., k\}$.
For two positive integers m and p, an *m-defective p-coloring* of a graph G is a (not necessarily legal) coloring of the vertices of G using p colors, such that each vertex has at most m neighbors colored by its color. Note that each color class in an m-defective coloring induces a graph of maximum degree m.

2.2 Coloring Procedures

In this section we summarize several well-known results that are used in the current paper. Some of our algorithms use as a black-box a procedure due to Kuhn and Wattenhofer [17]. This procedure accepts as input a graph G with maximum degree Δ, and an initial legal m-coloring, and it produces a $(\Delta+1)$-coloring of G within time $(\Delta+1) \cdot \lceil \log(m/(\Delta+1)) \rceil = O(\Delta \cdot \log(m/\Delta))$. We will refer to this procedure as *KW iterative procedure*, or *KW Procedure*. For future reference we summarize this in the next lemma.

Lemma 2.1. *[17] Given a legal m-coloring of a graph with maximum degree Δ, KW procedure produces a $(\Delta+1)$-coloring within $O(\Delta \cdot \log(m/\Delta))$ time.*

We also use a Δ-forest-decomposition procedure due to Panconesi and Rizzi [22]. (A similar construction was used also by Goldberg et al. [14].) This procedure accepts as input a graph G and computes a forest-decomposition with at most Δ forests in $O(1)$ time. We will refer to this procedure as PR *Δ-Forest-Decomposition* Procedure, or *PR Procedure*.

Next, we summarize the properties of several additional procedures that are used by our algorithms. In these procedures, as well as in our algorithms, we use the following naming conventions. If a procedure involves set-systems and requires a legal coloring as input, then the suffix -SL is added to its name. If the procedure involves set-systems, but does not require a coloring as input, then the suffix -SET is added to its name. If the procedure is set-system free, but it requires a legal coloring as input, then the suffix -LEG is added to its name. The next lemma summarizes the properties of procedures Delta-Col-SL

and Trade-Col-SL for computing legal colorings, and Procedure Defective-Col-LEG for computing defective colorings, devised in [6].

Lemma 2.2. *[6] **(1)** Procedure Delta-Col-SL invoked on an input graph G with an initial $O(\Delta^2)$-coloring computes a $(\Delta+1)$-coloring in $O(\Delta)$ time.*
***(2)** Given a graph G with an $O(\Delta^2)$ coloring, and an arbitrary parameter t, $1 < t \leq \Delta^{1/4}$, Procedure Trade-Col-SL computes an $O(\Delta \cdot t)$-coloring in time $O(\Delta/t)$.*
***(3)** Procedure Defective-Col-LEG accepts as input a graph G with an initial $(c' \cdot \Delta^k)$-coloring, for some constants $c' > 0$ and $k \geq 2$, and two parameters p, q such that $0 < p^2 < q$. It computes a $(\frac{\log(c' \cdot \Delta^k)}{\log(q/p^2)} \cdot \Delta/p)$-defective p^2-coloring in time $(\frac{\log(c' \cdot \Delta^k)}{\log(q/p^2)}) \cdot O(q)$. Moreover, Procedure Defective-Col-LEG is set-system-free.*

3 The Generic Method

Distributed computation of a $(\Delta+1)$-coloring in general graphs is a challenging task. However, for certain graph families very efficient, and even optimal, algorithms are known. In particular, the algorithm of Goldberg and Plotkin [13] [1] is applicable for oriented forests. (Henceforth, the GP algorithm.) The GP algorithm computes a 3-coloring of a forest in $O(\log^* n)$ time. Using PR Procedure the edge set of any graph can be partitioned into Δ oriented forests. Our algorithms start by partitioning a graph into Δ forests, and computing a 3-coloring in each forest, in parallel. Then these colorings are efficiently merged into a single unified $(\Delta+1)$-coloring.

We begin with describing a procedure called *Procedure Pair-Merge* that combines the colorings of two edge-disjoint subgraphs. Procedure Pair-Merge accepts as input a graph $G = (V, E)$, such that E is partitioned into two edge-disjoint subsets E' and E''. It also accepts two legal colorings φ' and φ'' for the graphs $G' = (V, E')$ and $G'' = (V, E'')$, respectively. The colorings φ' and φ'' employ at most c' and c'' colors, respectively. Procedure Pair-Merge returns a new coloring φ for G such that for every $v \in V$, $\varphi(v) = c'' \cdot (\varphi'(v) - 1) + \varphi''(v)$. The new coloring can be seen as an ordered pair $< \varphi'(v), \varphi''(v) >$. This completes the description of the procedure. Observe that invoking Procedure Pair-Merge requires no communication whatsoever. The properties of the procedure are summarized in the next lemma.

Lemma 3.1. *Procedure Pair-Merge produces a legal $(c' \cdot c'')$-coloring of G.*

Next, we describe a generic method [2], called Generic-Merge, for merging the coloring of ℓ edge disjoint subgraphs of G, for a positive integer ℓ. Suppose that we are given an edge partition $E_1, E_2, ..., E_\ell$ of E. Moreover, for $1 \leq i \leq \ell$, we are given a legal coloring φ_i of $G_i = (V, E_i)$ that employs at most c_i colors, $c_i \geq \Delta+1$. In addition, the method Generic-Merge employs an auxilary coloring

[1] The algorithm of [13] is based on an earlier algorithm of Cole and Vishkin [9].
[2] We refer to a procedure as *generic method* if it accepts another procedure as input.

procedure called Procedure Reduce(H, α, β). We will later use the method Generic-Merge with a number of different instantiations of Procedure Reduce. However, in all its instantiations Procedure Reduce accepts as input a graph H with a legal α-coloring, and computes a legal β-coloring of H. Procedure Reduce requires that $\alpha > \beta \geq \Delta+1$. The method Generic-Merge also accepts as input a positive integer parameter d, $1 \leq d \leq \min\{c_i | 1 \leq i \leq \ell\}$. Roughly speaking, the parameter d determines the ratio between α and β. To summarize, the method Generic-Merge accepts as input a partition $E_1, E_2, ..., E_\ell$ of E, a legal c_i-coloring φ_i of E_i, for each $i = 1, 2, ..., \ell$, a procedure Reduce, and a parameter d. It returns a legal coloring φ of the entire input graph G.

The method Generic-Merge proceeds in phases. In each phase pairs of subgraphs are merged using Procedure Pair-Merge, in parallel. As a result we obtain fewer subgraphs, but a greater number of colors is employed in each subgraph. The number of colors is then reduced using Procedure Reduce, which is invoked in parallel on the merged subgraphs. This process of pairing subgraphs, merging their colorings, and reducing the number of employed colors is repeated for $\lceil \log \ell \rceil$ phases, until all subgraphs are merged into the original input graph G.

Algorithm 1 Method Generic-Merge($G_1, G_2, ..., G_\Delta, \varphi_1, \varphi_2, ..., \varphi_\Delta, c_1, c_2, ..., c_\Delta$, Procedure Reduce, d)

1: $\ell := \Delta$ /* ℓ is the number of subgraphs */
2: **while** $\ell > 1$ **do**
3: **for** $i := 1, 2, ..., \lfloor \ell/2 \rfloor$, in parallel **do**
4: $\{G'_i, \varphi'_i\}$:= Pair-Merge($G_{2i-1}, G_{2i}, \varphi_{2i-1}, \varphi_{2i}, c_{2i-1}, c_{2i}$)
5: $\{G_i, \varphi_i\}$:= Reduce(G'_i, $\alpha_i := c_{2i-1} \cdot c_{2i}$, $\beta_i := \lfloor \alpha_i/d \rfloor$)
6: $c_i := \lfloor c_{2i-1} \cdot c_{2i}/d \rfloor$
7: **end for**
8: **if** ℓ is odd **then**
9: $\{G_{\lceil \ell/2 \rceil}, \varphi_{\lceil \ell/2 \rceil}, c_{\lceil \ell/2 \rceil}\} := \{G_\ell, \varphi_\ell, c_\ell\}$
10: **end if**
11: $\ell := \lceil \ell/2 \rceil$
12: **end while**
13: return $\{G_1, \varphi_1, c_1\}$

In the first phase of Generic-Merge the pairs (G_1, G_2), (G_3, G_4),...,$(G_{\ell-1}, G_\ell)$ are merged into the subgraphs $G'_1, G'_2, ..., G'_{\lceil \ell/2 \rceil}$ by using Procedure Pair-Merge. (In other words, $G'_i = (V, E_{2i-1} \bigcup E_{2i})$, for $i = 1, 2, ..., \lfloor \ell/2 \rfloor$. If ℓ is odd then we define $G_{\ell+1} = (V, \emptyset)$, $c_{\ell+1} = 1$, and, consequently, $G'_{\lceil \ell/2 \rceil} = G_\ell$.) Set $\alpha_i = c'_i = c_{2i-1} \cdot c_{2i}$, and $\beta_i = \lfloor \alpha_i/d \rfloor$. (Intuitively, α_i is an upper bound on the number of colors used by the coloring that Procedure Pair-Merge produces for the subgraph G'_i. Procedure Reduce transforms this coloring into a β_i-coloring, with $\beta_i = \lfloor \alpha_i/d \rfloor$.) Next, Procedure Reduce(G'_i, α_i, β_i) is executed in parallel, for $i = 1, 2, ..., \lfloor \ell/2 \rfloor$. In general, a phase proceeds as follows. Suppose that the previous phase has produced the subgraphs $G'_1, G'_2, ..., G'_{\ell'}$, such that each subgraph is

colored with at most $c'_1, c'_2, ..., c'_{\ell'}$ colors, respectively. Then the subgraphs G'_{2i-1} and G'_{2i} are merged into the subgraph G''_i, and Procedure Reduce(G''_i, $\alpha'_i = c'_{2i-1} \cdot c'_{2i}$, $\beta'_i = \lfloor c'_{2i-1} \cdot c'_{2i}/d \rfloor$) is executed in parallel, for $i = 1, 2, ..., \lfloor \ell'/2 \rfloor$. If ℓ is odd then $G''_{\lceil \ell'/2 \rceil} = G'_{\ell'}$. In this case the coloring of G'_ℓ is also used for $G''_{\lceil \ell'/2 \rceil}$, instead of invoking Procedure Reduce on it. The method Generic-Merge terminates once all subgraphs are merged into a single graph, that is, the input graph G. This completes the description of the method Generic-Merge.

Lemma 3.2. *The method Generic-Merge produces a legal* $(c_1 \cdot c_2 \cdot .... \cdot c_\ell)/d^{\,\ell-1}$ *coloring* φ *of the input graph* G.

4 Coloring Algorithms

4.1 Procedure Simple-Col

In this section we present our first algorithm that employs the method Generic-Merge, called *Procedure Simple-Col*. The method Generic-Merge accepts as input a partition of the input graph such that each subgraph in the partition is legally colored. In order to compute such a partition, we invoke the PR Procedure. Recall that this procedure computes a Δ-forest-decomposition of G. In other words, the procedure outputs an edge partition $\{G_1, G_2, ..., G_\Delta\}$, $G_i = (V, E_i), i \in \{1, 2, ..., \Delta\}$, such that each subgraph in this partition is an oriented forest. Next, each forest is colored with 3 colors using the GP algorithm. The Δ invocations of the GP algorithm are performed in parallel. They result in legal colorings $\varphi_1, \varphi_2, ..., \varphi_\Delta$. Since $\Delta \geq 2$, each coloring φ_i, $i = 1, 2, ..., \Delta$, employs at most $\Delta + 1$ colors.

Recall that the method Generic-Merge also accepts as input parameter a procedure (Procedure Reduce) for reducing the number of colors in a given coloring of a graph. In Procedure Simple-Col we employ the KW iterative procedure as Procedure Reduce. (The KW iterative procedure accepts as input a graph H with an α-coloring, $\alpha \geq \Delta + 1$, and computes a $(\Delta + 1)$-coloring of H in time $O(\Delta \log(\alpha/\Delta))$.) We invoke Generic-Merge on the partition $\{G_1, G_2, ..., G_\Delta\}$ with the colorings $\varphi_1, \varphi_2, ..., \varphi_\Delta$ such that $c_1 = c_2 = ... = c_\Delta = \Delta + 1$. Finally, the parameter d is set to $\Delta + 1$. Consequently, in each phase of Generic-Merge pairs of $(\Delta + 1)$-colored subgraphs are merged into $(\Delta + 1)^2$-colored subgraphs, and then the number of colors in each subgraph is reduced back to $\Delta + 1$. Once Generic-Merge terminates, the input graph is legally colored using $\Delta + 1$ colors. The pseudocode of the procedure follows. Its properties are summarized below.

Algorithm 2 Procedure Simple-Col(G)

1: $\{G_1, G_2, ..., G_\Delta\} := \Delta$-Forests-Decomposition($G$) /*using PR Procedure */
2: **for** $i = 1, 2, ..., \Delta$ in parallel **do**
3: $\varphi_i :=$ 3-color(G_i) /*using GP algorithm */
4: $c_i := \Delta + 1$
5: **end for**
6: Generic-Merge($G_1, G_2, ..., G_\Delta, \varphi_1, \varphi_2, ..., \varphi_\Delta, c_1, c_2, ..., c_\Delta$, KW procedure, $\Delta + 1$)

Theorem 4.1. *Procedure Simple-Col produces a legal $(\Delta+1)$-coloring of G. Its running time is $O(\Delta \log^2 \Delta) + \log^* n$. This procedure is set-system free.*

4.2 Procedures Poly-Col and Fast-Col

Our algorithm consists of two major stages. In the first stage we compute a $\Delta^{O(1)}$-coloring, and in the second stage we reduce the number of colors from $\Delta^{O(1)}$ to $(\Delta+1)$. In the existing $(\Delta+1)$-coloring algorithms that run in $O(\Delta)+\log^* n$ time [6, 16], both these stages employ set systems. In fact, as far as we know, currently there is no known set-system free $\Delta^{O(1)}$-coloring algorithm that runs within $O(\Delta)+\log^* n$ time. The situation is somewhat better in the context of the second stage, as there is a known set-system free algorithm (KW Procedure) that accepts a $\Delta^{O(1)}$-coloring as input and returns a $(\Delta+1)$-coloring. However, its running time ($O(\Delta \log \Delta)$) is higher than the desired bound of $O(\Delta) + \log^* n$. Therefore, we speed up both the first and the second stages of the aforementioned scheme, and achieve a set-system free $(\Delta+1)$-coloring algorithm with running time $O(\Delta + \log^* n)$.

Before we begin with the description of our new algorithm, we provide a brief survey of several known (not set-system free) algorithms (due to [6]) that employ the two-stage technique described above. In the sequel, we modify these algorithms, and eliminate the steps that employ set-systmes. Then we employ the modified versions for devising our new results. We start with sketching Procedure Defective-Col-SET from [6], that accepts as input a graph G and two parameters p and q, and returns a defective coloring of G.

Algorithm 3 Procedure Defective-Col-SET (G, p, q)

1: ϑ := an $O(\Delta^2)$-coloring of G /* using set-systems */
2: ψ := Defective-Col-LEG (G, ϑ, p, q) /* set-system free */
3: return ψ

Set $p = \Delta^\epsilon$, $q = \Delta^{3\epsilon}$, for an arbitrarily small constant $\epsilon > 0$. By Lemma 2.2 (3), Procedure Defective-Col-SET invoked with these parameters computes an $O(\Delta/p)$-defective p^2-coloring of G.

Next we sketch a procedure devised in [6] for computing a legal $O(\Delta^{5/4})$-coloring from a legal $O(\Delta^2)$-coloring in $O(\Delta^{3/4})$ time. This procedure is called Procedure Trade-Col-SL. (The properties of this procedure in its general form are summarized in Lemma 2.2 (2). In the current discussion we fix the parameter t to be equal to $\Delta^{1/4}$.) Procedure Trade-Col-SL accepts as input a graph G and a legal $O(\Delta^2)$-coloring ϑ of G. The procedure proceeds as follows. First, it computes an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring of the input graph using Procedure Defective-Col-LEG. (See Lemma 2.2 (3).) To this end we set $p = \Delta^{1/4}$, $q = \Delta^{3/4}$. (Normally, it is preceded by a step in which an $O(\Delta^2)$-coloring is computed via Linial's algorithm, i.e., using set systems. In the current version of the procedure, this coloring is assumed to be provided as a part of the input.) The defective coloring returned by the invocation of Procedure Defective-Col-LEG

induces a *vertex* partition into $O(\Delta^{1/2})$ subgraphs such that each subgraph has maximum degree $O(\Delta^{3/4})$. Next, all subgraphs are legally colored with distinct palettes of size $O(\Delta^{3/4})$ for each subgraph, in parallel. Then these colorings are combined into a unified legal $O(\Delta^{5/4})$-coloring. This completes the description of Procedure Trade-Col-SL. Similarly to Algorithm 3, the computation is divided into stages that either involve set-systems or are set-systems free.

Algorithm 4 Procedure Trade-Col-SL (G , ϑ)

1: ψ := Defective-Col-LEG (G, ϑ, $p := \Delta^{1/4}$, $q := \Delta^{3/4}$) /* set-system free; see Lemma 2.2 (3) */
/* ψ is an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring of G */
/* ψ induces a vertex partition into $O(\Delta^{1/2})$ subgraphs, each with max. degree $O(\Delta^{3/4})$ */
2: **for** each subgraph G_i induced by color classes of ψ, in parallel **do**
3: φ_i := color G_i with $O(\Delta^{3/2})$ colors using Linial alg.[18] /* using set-systems */
4: φ'_i := Delta-Col-SL(G_i, φ_i) /* using set-systems; see Lemma 2.2 (1) */
/* φ'_i is an $O(\Delta^{3/4})$-coloring of G_i */
5: **end for**
6: **for** $i = 1, 2, ...$ in parallel **do**
7: combine all colorings φ'_i into a unified legal $O(\Delta^{5/4})$-coloring φ of G
/*set-system free */
8: **end for**
9: return φ

Next, we turn to describing our new set-system free algorithm for computing an $O(\Delta^{5/4})$-coloring from scratch. Our algorithm employs the method Generic-Merge, that was described in Section 3. In Section 4.1 the KW Procedure was used as an instantiation for Procedure Reduce in the method Generic-Merge. This time a different procedure is used as an instantiation for Procedure Reduce. This procedure reduces the number of colors from $c^2 \cdot \Delta^{5/2}$ to $c \cdot \Delta^{5/4}$, for a positive constants c. Its running time is $O(\Delta^{3/4} \log \Delta) = o(\Delta / \log \Delta)$. Consequently, the $\lceil \log \Delta \rceil$ phases of Generic-Merge require overall time of $o(\Delta)$. For $i = 1, 2, .., \lceil \log \Delta \rceil$, the colorings of subgraphs in the partition of phase i employ at most $(c \cdot \Delta^{5/4})$ colors each. In phase i, pairs of $(c \cdot \Delta^{5/4})$-colored subgraphs are merged into $(c^2 \cdot \Delta^{5/2})$-colored subgraphs, and the number of colors is then reduced back to $(c \cdot \Delta^{5/4})$ in each subgraph. Once the method Generic-Merge terminates, the input graph is colored with $(c \cdot \Delta^{5/4})$ colors.

We use a modified version of Procedure Trade-Col-SL (Algorithm 4), as an instantiation for Procedure Reduce. Given an $O(\Delta^2)$-coloring as input, Procedure Trade-Col-SL computes an $O(\Delta^{5/4})$-coloring in $O(\Delta^{3/4})$ time. Some of its steps employ set-systems. (See Algorithm 4.) Consequently, the original Procedure Trade-Col-SL cannot be used. We perform two modifications in the procedure. First, the steps that employ set-systems are replaced by analogous set-system free steps. Second, we show that the procedure computes an $O(\Delta^{5/4})$-coloring

from an $O(\Delta^k)$-coloring for any constant $k \geq 2$, not only $k = 2$. We henceforth refer to the modified procedure as *Procedure Mod-Trade-LEG*.

Procedure Trade-Col-SL employs set-systems for computing $O(\Delta^{3/2})$-colorings of subgraphs G_i (line 3 of Algorithm 4). In addition, it employs set-systems in the invocation of Procedure Delta-Col-SL for computing legal colorings of subgraphs with linear number of colors (and in linear time) in the degree of the subgraphs (line 4 of Algorithm 4). In Procedure Mod-Trade-LEG we omit the step of computing $O(\Delta^{3/2})$-coloring of subgraphs G_i. Instead of this step, φ_i is set as the initial coloring ϑ, for all i. For each $i = 1, 2, ..., O(\Delta^{1/2})$, let $\Delta_i = \Theta(\Delta^{3/4})$ be an upper bound on the maximum degree $\Delta(G_i)$ of the graph G_i. Since ϑ is an $O(\Delta^2)$-coloring, it is also an $O(\Delta_i^{8/3})$-coloring of G_i. Next, we replace the step of coloring G_i using Procedure Delta-Col-SL with an invocation of the KW iterative procedure, which is set-systems free. This procedure can start (see Lemma 2.1) with an arbitrarily large number of colors, as long as it is polynomial in the maximum degree of the underlying graph. However, as a result, the running time of procedure Mod-Trade-LEG grows by a multiplicative factor of $\log \Delta$, and becomes $O(\Delta^{3/4} \log \Delta)$.

The original Procedure Trade-Col-SL accepts as input a $(c' \cdot \Delta^2)$-coloring ϑ, for a positive constant c', and reduces it into an $O(\Delta^{5/4})$-coloring. Once line 3 of Algorithm 4 is skipped[1], the coloring ϑ is used only in one step of Procedure Trade-Col-SL, specifically, in the step that computes the $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring (line 1 of Algorithm 4). In this step a procedure called Procedure Defective-Col-LEG is invoked with two input parameters $p = \Delta^{1/4}$ and $q = \Delta^{\epsilon} \cdot p^2$, for an arbitrary small positive constant $\epsilon \leq 1/4$. Procedure Defective-Col-LEG employs the coloring ϑ to compute a $(\frac{\log(c' \cdot \Delta^2)}{\log(q/p^2)} \cdot \Delta/p)$-defective p^2-coloring in time $(\frac{\log(c' \cdot \Delta^2)}{\log(q/p^2)}) \cdot O(q)$. Procedure Defective-Col-LEG can employ any legal t-coloring φ' instead of the $(c' \cdot \Delta^2)$-coloring ϑ. In this case, by Lemma 2.2 (3), it computes a $(\frac{\log t}{\log(q/p^2)} \cdot \Delta/p)$-defective p^2-coloring in time $(\frac{\log t}{\log(q/p^2)}) \cdot O(q)$. In particular, if φ' is a $\Delta^{O(1)}$-coloring, then Procedure Defective-Col-LEG employs φ' to compute a $(\frac{\log(\Delta^{O(1)})}{\log(q/p^2)} \cdot \Delta/p)$-defective p^2-coloring in time $(\frac{\log(\Delta^{O(1)})}{\log(q/p^2)}) \cdot O(q)$. In other words, if Procedure Defective-Col-LEG employs a $\Delta^{O(1)}$-coloring, then it computes an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring in time $O(q) = O(\Delta^{\epsilon} \cdot p^2) = O(\Delta^{1/2+\epsilon})$.

The modified Procedure Mod-Trade-LEG proceeds as follows. It accepts as input a $\Delta^{O(1)}$-coloring ϑ. First, it computes an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring ψ of the input graph in time $O(\Delta^{1/2+\epsilon})$ by Procedure Defective-Col-LEG as was described above. Next, the subgraphs $G_1, G_2, ...$ induced by color classes of ψ are legally colored with distinct palettes of size $O(\Delta^{3/4})$ each, using the KW iterative procedure in parallel on all subgraphs, in time $O(\Delta^{3/4} \cdot \log(\frac{\Delta^{O(1)}}{\Delta^{3/4}})) = O(\Delta^{3/4} \log \Delta)$. (Observe that the KW iterative procedure invoked

[1] Line 3 of Algorithm 4 invokes the algorithm of Linial. To speed up the computation of $O(\Delta^{3/2})$-coloring φ_i, the algorithm of Linial employs the $O(\Delta^2)$-coloring ϑ.

on a subgraph G_i needs an initial coloring ϑ_i to start working. Here we restrict the coloring ϑ of the entire graph G to the vertex set V_i of the subgraph G_i. The resulting restricted coloring is called ϑ_i, and is employed by the KW iterative procedure as an initial coloring.) Then the colorings produced by the invocations of the KW iterative procedure are combined into a unified legal $O(\Delta^{5/4})$ coloring. The combining step requires no communication whatsoever. This completes the description of the procedure. Its pseudocode is provided below. It properties are given in the next lemma.

Algorithm 5 Procedure Mod-Trade-LEG (G , ϑ)

1: ψ :=Defective-Col-LEG (G, ϑ, $\Delta^{1/4}$, $\Delta^{3/4}$) /* ψ is an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring */
2: **for** each subgraph G_i induced by ψ, in parallel **do**
3: φ'_i := color G_i with $O(\Delta^{3/4})$ colors using the KW iterative procedure
/* use the restriction ϑ_i of the coloring ϑ to the vertex set V_i of G_i as initial coloring */
4: **end for**
5: **for** $i = 1, 2, ...,$ in parallel **do**
6: combine all colorings φ'_i into a unified legal $O(\Delta^{5/4})$-coloring φ of G
7: **end for**
8: return φ

Lemma 4.2. *Procedure Mod-Trade-LEG invoked with a $\Delta^{O(1)}$-coloring ϑ as input computes an $O(\Delta^{5/4})$-coloring φ in time $O(\Delta^{3/4}\log\Delta)$. Specifically, if ϑ is a $(c' \cdot \Delta^k)$-coloring, for constants $c' > 0$, $k \geq 2$, then φ employs at most $(\frac{\log(c'\cdot\Delta^k)}{\log\Delta^\epsilon} \cdot \Delta^{5/4}) = O(\frac{k}{\epsilon} \cdot \Delta^{5/4})$ colors.*

Suppose that Procedure Mod-Trade-LEG is invoked with a $(c^2\cdot\Delta^{5/2})$-coloring as input, for a positive constant c to be determined later. Then it computes a $(\frac{\log(c^2\cdot\Delta^{5/2})}{\log\Delta^\epsilon}\cdot\Delta^{5/4})$-coloring φ, for an arbitrary positive constant $\epsilon \leq 1/4$. For any constant c such that $c \geq \frac{2\log c+5/2}{\epsilon}$ it holds that $\frac{\log(c^2\cdot\Delta^{5/2})}{\log\Delta^\epsilon} \leq c$. (To satisfy the condition set $\epsilon = 1/8$, and c to be a sufficiently large constant.) Consequently, the resulting coloring φ is a $\lfloor c \cdot \Delta^{5/4} \rfloor$-coloring.

Next, we present a set-system free procedure, called Procedure Poly-Col, that computes an $O(\Delta^{5/4})$-coloring of the input graph *from scratch*, in time $\tilde{O}(\Delta^{3/4}) + \log^* n$. (This is in contrast to Procedure Defective-Col-LEG that accepts as input a legal $O(\Delta^2)$-coloring ϑ.) Procedure Poly-Col is very similar to Procedure Simple-Col (Algorithm 2.) The main difference is that in step 8 Procedure Poly-Col invokes the method Generic-Merge with Procedure Mod-Trade-LEG as an instantiation for Procedure Reduce instead of the KW Procedure. In addition, the variables $c_1, c_2, ..., c_\Delta$, and d are set to $\lfloor c \cdot \Delta^{5/4} \rfloor$ instead of $\Delta + 1$, where c is a constant as above. The pseudocode of the procedure is given below. Its properties are summarized in the next lemma.

Algorithm 6 Procedure Poly-Col(G)

1: $\{G_1, G_2, ..., G_\Delta\} :=$ Δ-Forest-Decomposition(G) /* using PR Procedure */
2: **for** $i = 1, 2, ..., \Delta$ in parallel **do**
3: $c_i := \left\lfloor c \cdot \Delta^{5/4} \right\rfloor$; $\varphi_i :=$ 3-color(G_i) /* using GP Alg.; each G_i is a forest */
4: **end for**
5: $d := \left\lfloor c \cdot \Delta^{5/4} \right\rfloor$
6: Generic-Merge(P,$G_1, G_2, ..., G_\Delta, \varphi_1, \varphi_2, ..., \varphi_\Delta, c_1, c_2, ..., c_\Delta$,Procedure Mod-Trade-LEG, d)

Lemma 4.3. *Procedure Poly-Col computes from scratch a legal coloring of G that employs at most $c \cdot \Delta^{5/4} = O(\Delta^{5/4})$ colors. Its running time is $O(\Delta^{3/4} \log^2 \Delta + \log^* n)$. Moreover, Procedure Poly-Col is set-system free.*

In what follows we devise a set-system free procedure for computing a $(\Delta+1)$-coloring in $O(\Delta + \log^* n)$ time. In [6] the authors of the current paper devised a procedure, called *Procedure Delta-Color*, (henceforth, *Procedure Delta-Color-SL*), that accepts as input a graph G and a γ-coloring, $\gamma = O(\Delta^2)$, and computes a $(\Delta + 1)$-coloring of G in $O(\Delta)$ time. (See Lemma 2.2.) However, Procedure Delta-Col-SL employs set-systems. Next, we overview Procedure Delta-Col-SL from [6]. This procedure works as follows. If the number of colors in the coloring it accepts as input is $\gamma = O(\Delta)$, then a $(\Delta + 1)$-coloring of G is computed directly from the γ-coloring using the KW iterative procedure within $O(\Delta)$ time. Otherwise, the graph G is partitioned into vertex-disjoint subgraphs with maximum degrees $d < \Delta$, by invoking Procedure Defective-Col-LEG. Next, for each subgraph, an $O(d^2)$-coloring is computed using set-systems, by Linial's algorithm [18]. Then Procedure Delta-Col-SL is invoked recursively on each of the subgraphs. As a result we obtain a $(d + 1)$-coloring for each subgraph. (The recursion depth is $\lfloor \log^* \Delta \rfloor$. For $j = 1, 2, ..., \lfloor \log^* \Delta \rfloor$, in recursion level $\lfloor \log^* \Delta \rfloor - j$ it holds that $d = d_j = \Theta(\Delta / \Pi_{i=j}^{\log^* \Delta}(\log^{(i)} \Delta))$, i.e., $d_j = \Omega(\Delta^{1-\epsilon})$ for any constant $\epsilon > 0$.) These colorings are then merged into a unified legal $(\Delta+1)$-coloring of the input graph. This completes the description of Procedure Delta-Col-SL. The running time of a recursion level j, $j = 1, 2, ..., \lfloor \log^* \Delta \rfloor$, is $O(d_j \cdot \log^{(j)} \Delta)$. Consequently, the overall running time is $O(\sum_{j=1}^{\log^* \Delta}(d_j \cdot \log^{(j)} \Delta)) = O(\Delta)$.

The only step in Procedure Delta-Col-SL that employs set-systems is the step that computes $O(d^2)$-colorings. Specifically, let $d_j = c \cdot \Delta / \Pi_{i=j}^{\log^* \Delta}(\log^{(i)} \Delta)$, for some fixed positive constant c. For $j = 1, 2, ..., \lfloor \log^* \Delta \rfloor$, Procedure Delta-Col-SL computes $O(d_j^2)$-colorings of subgraphs of degree $O(d_j)$. To this end it employs the algorithm of Linial [18]. We argue that if Procedure Delta-Col-SL accepts an $O(\Delta^{5/4})$-coloring instead of an $O(\Delta^2)$-coloring, then employing set systems is no longer required. Observe that as $d_j = \Omega(\Delta^{1-\epsilon})$, for any constant $\epsilon > 0$, it follows that for a sufficiently large Δ, $\Delta^{5/4} \leq d_j^2$, for all $j = 1, 2, ..., \lfloor \log^* \Delta \rfloor$. Hence an $O(\Delta^{5/4})$-coloring is in particular an $O(d_j^2)$-coloring for all $j = 1, 2, ..., \lfloor \log^* \Delta \rfloor$. Therefore, invoking Procedure Delta-Col-SL with an $O(\Delta^{5/4})$-coloring as input instead of an $O(\Delta^2)$-coloring eliminates the need of computing $O(d_j^2)$-colorings

of subgraphs. Indeed, for a subgraph H of degree d_j, $j \in \{1, 2, ..., \lfloor \log^* \Delta \rfloor\}$, the input coloring is already an $O(d_j^2)$-coloring of H.

To summarize, we obtained a variant of Procedure Delta-Col-SL, to which we will refer as Procedure Mod-Delta-LEG. This procedure accepts as input a graph G of maximum degree Δ, and an $O(\Delta^{5/4})$-coloring ϑ for G. The procedure returns a $(\Delta+1)$-coloring, and it does so within $O(\Delta)$ time. (Observe that the running time of Procedure Mod-Delta-LEG is not greater than the running time of Procedure Delta-Col-SL when invoked with an $O(\Delta^{5/4})$-coloring as input. The two procedures perform exactly the same steps, except that Procedure Mod-Delta-LEG skips the invocation of the algorithm of Linial.)

To complete the algorithm it is only left to combine Procedure Poly-Col (that computes an $O(\Delta^{5/4})$-coloring from scratch) with Procedure Mod-Delta-LEG that reduces the number of colors to $\Delta+1$. The resulting procedure will be referred to as Procedure Fast-Col. It accepts as input a graph G, and performs two steps. In the first step it computes an $O(\Delta^{5/4})$-coloring ϑ using Procedure Poly-Col. In the second step it invokes the set-system free variant (Procedure Mod-Delta-LEG) of Procedure Delta-Col-SL with the coloring ϑ as input. This procedure outputs a $(\Delta+1)$-coloring. The running time of Procedure Fast-Col is the sum of the running times of Procedure Poly-Col and Procedure Mod-Delta-LEG. The former is, by Lemma 4.3, $O(\Delta^{3/4} \log^2 \Delta) + \log^* n$, and the latter is $O(\Delta)$. Hence the overall running time of Procedure Fast-Col is $O(\Delta) + \log^* n$.

Theorem 4.4. *Procedure Fast-Col computes a $(\Delta+1)$-coloring of the input graph G in $O(\Delta + \log^* n)$ time. Moreover, this computation is set-system free.*

Our results give rise to set-system free algorithms for a variety of problems. (Detailed proofs of these results will be provided in the full version of this paper.)

Corollary 4.5. ***(1)*** *For an arbitrarily small constant $\epsilon > 0$, and a parameter t, $1 < t \leq \Delta^{1-\epsilon}$, one can compute an $O(\Delta \cdot t)$-coloring in $O((\Delta/t) + \log^* n)$ time.* ***(2)*** *A Maximal Independent Set can be computed in $O(\Delta + \log^* n)$ time.* ***(3)*** *For the family of graphs with bounded arboricity $a(G) = o(\sqrt{\log n})$ a Maximal Independent Set and $(\Delta+1)$-coloring can be computed in sublogarithmic time. Results* ***(1),(2)*** *and* ***(3)*** *are obtained using set-system free algorithms.*

References

1. N. Alon. Eigen-values and expanders. *Combinatorica* 6(2):83-96, 1986.
2. N. Alon, L. Babai, and A. Itai. A fast and simple randomized parallel algorithm for the maximal independent set problem. *J. Algorithms*, 7(4):567–583, 1986.
3. S. Arora, and S. Safra. Probabilistic Checking of Proofs: A New Characterization of NP. *Journal of the ACM*, 45(1):70-122, 1998.
4. B. Awerbuch, A. V. Goldberg, M. Luby, and S. Plotkin. Network decomposition and locality in distributed computation. In *Proc. of the 30th IEEE Annual Symposium on Foundations of Computer Science*, pages 364–369, October 1989.
5. L. Barenboim, and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. In *Proc. of the 27th ACM Symp. on Principles of Distributed Computing*, pages 25–34, 2008.

6. L. Barenboim, and M. Elkin. Distributed $(\Delta + 1)$-coloring in linear (in Δ) time. In *Proc. of the 41th ACM Symp. on Theory of Computing*, pages 111-120, 2009. See also *http://arXiv.org/abs/0812.1379v2*, 2008.
7. L. Barenboim, and M. Elkin. Deterministic distributed vertex coloring in polylogarithmic time. In *Proc. of the 29th ACM Symp. on Principles of Distributed Computing*, pages 410-419, 2010.
8. A. Ben-Aroya, A. Ta-Shma. A combinatorial construction of almost-Ramanujan graphs using the zig-zag product. In *Proc. of the 40th ACM Symp. on Theory of Computing*, 325-334, 2008.
9. R. Cole, and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32–53, 1986.
10. I. Dinur, and O. Reingold. Assignment Testers: Towards a Combinatorial Proof of the PCP Theorem. *SIAM Journal on Computing*, 36(4):975-1024, 2006.
11. P. Erdős, P. Frankl, and Z. Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel Journal of Mathematics*, 51:79–89, 1985.
12. U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *JACM*, 43(2):268-292, 1996.
13. A. Goldberg, and S. Plotkin. Efficient parallel algorithms for $(\Delta+1)$- coloring and maximal independent set problem. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 315–324, 1987.
14. A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. *SIAM Journal on Discrete Mathematics*, 1(4):434-446, 1988.
15. K. Kothapalli, C. Scheideler, M. Onus, and C. Schindelhauer. Distributed coloring in $\tilde{O}(\sqrt{\log n})$ bit rounds. In *20th International Parallel and Distributed Processing Symposium*, 2006.
16. F. Kuhn. Weak graph colorings: distributed algorithms and applications. In *Proc. 21st ACM Symp. on Parallel Algorithms and Architectures*, pages (138–144), 2009.
17. F. Kuhn, and R. Wattenhofer. On the complexity of distributed graph coloring. In *Proc. of the 25th ACM Symp. on Principles of Distributed Computing*, pages 7–15, 2006.
18. N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21(1):193–201, 1992.
19. A. Lubotzky, R. Philips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8:261–277, 1988.
20. M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15:1036-1053, 1986.
21. O. Meir. Combinatorial PCPs with Efficient Verifiers. In *Proc. of the 50th Annual IEEE Symp. on Foundations of Computer Science*, pages 463-471, 2009.
22. A. Panconesi, and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001.
23. A. Panconesi, and A. Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):581-592, 1995.
24. D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.
25. O. Reingold. Undirected ST-connectivity in log-space. In *Proc. of the 37th ACM Symp. on Theory of Computing*, pages 376-385, 2005.
26. O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Proc. of the 41st Annual IEEE Symp. on Foundations of Computer Science*, pages 3-13, 2000.
27. J. Schneider, and R. Wattenhofer. A New Technique For Distributed Symmetry Breaking. In *Proc. of the 29th ACM Symp. on Principles of Distributed Computing*, pages 257-266, 2010.