

Diversity Coloring for Distributed Data Storage in Networks¹

Anxiao (Andrew) Jiang and Jehoshua Bruck
California Institute of Technology
Pasadena, CA 91125, U.S.A.
{jax, bruck}@paradise.caltech.edu

Abstract — Distributively storing widely shared files with redundancy is an important technique for high performance and fault-tolerance in information networks. This paper proposes a new file storage scheme for encoded files, aiming at satisfying highly varied QoS requirements and guaranteeing graceful performance-degradation while some data become inaccessible. In the scheme, every node can get a file by accessing data in its proximity of a bounded radius; and the variety of data in the proximity increases steadily when the proximity's radius increases.

We formulate the file storage scheme as a new graph coloring problem which we call *diversity coloring*. The diversity coloring problem is shown to be NP-hard for general graphs. An algorithm using a *K-interleaving* technique for obtaining diversity coloring on tree networks is presented. The algorithm is of low complexity and can guarantee to output solutions that minimize the length of the codeword representing the file and also minimize the delay for *any* node to retrieve *any* amount of distinct data. Various other aspects of the algorithm, as well as properties of diversity coloring on trees and more general graphs, are also studied.

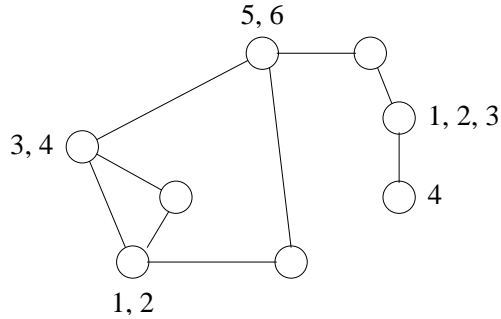
Index Terms — Distributed networks, diversity coloring, file assignment, K-interleaving, tree.

I. INTRODUCTION

The Internet is becoming a more unified massive data-storage system, where information is shared by users distributed all over the world. For a widely shared file, distributively storing multiple copies of it in the network, compared to storing just a single copy, reduces communication cost for file retrieving and improves fault tolerance. There has been considerable research on devising distributed storage schemes for shared files in the Internet [3] [11] [15] [20] [28] [30] [35] [38]. Some of them are for file storage in stable environments, where parameters such as the file-retrieving frequencies are largely constant, and typically use median-type or center-type solutions [7] [9] [14] [17] [18] [19] [23] [36]. The others are more appropriate for unstable environments, where parameters dynamically change and are difficult to predict, and may use certain techniques to get suboptimal but very computationally efficient solutions [11] [20] [33] [38] in order to adapt to the highly dynamic conditions. Examples of those techniques include using greedy methods for file assignment and removal [38], using hash functions to map files to memories [16], and restricting file storage and transmission to a subset of the network of a special topology, e.g., a tree [33]. Many distributed storage systems for the Internet have emerged, including the Distributed-Parallel Storage System (DPSS) [24], the OceanStore system [21] and various caching systems [27] [37].

In this paper we'll propose and study a novel file storage scheme for stable environments. The scheme combines coding with file storage for better performance. Storing encoded files has been adopted in disk storage systems such as RAID [4] [29], in server clusters [6], and in parallel-distributed systems [24] [31]. In all those cases, however, the underlying networks are of highly regular structures, such as the complete graph structure [29] or the hypercube structure [31]. The study on storing encoded files in more irregular networks has been very limited, despite the broad counterpart work done on un-encoded files. Among the few results on storing encoded files in irregular networks, the paper by Naor and Roth [26] is a major one. Our storage model generalizes the model in [26] in order to make it more suitable for real irregular networks such as the Internet, and our storage scheme generalizes the scheme in [26] in order to improve fault-tolerance and quality-of-service (QoS). Because of the relevancy of the work in [26], we briefly introduce the model and the scheme in [26] below.

¹This work was supported in part by the Lee Center for Advanced Networking at the California Institute of Technology.



The codeword has 6 symbols.

Any 4 distinct symbols are sufficient for recovering the file.

For every node, there are at least 4 distinct symbols within 1 hop.

Figure 1: An example of storing an encoded file in a network.

Given a file, we can see it as a string of k symbols, and encode it using an (n, k) code to get a total of n symbols. If the code can correct up to ε erasures, then any $n - \varepsilon$ of those n symbols are sufficient for recovering the file. We can distributively store copies of those n symbols in a network. And a user in need of the file can reconstruct it by retrieving any $n - \varepsilon$ distinct symbols from the network. Clearly a file that is not encoded can be seen as being encoded with a (k, k) code. Therefore storing un-encoded files in networks is a special case of storing encoded files. An error-correcting code increases the variety of data representing the file, therefore can help find solutions of improved performance and enhanced fault-tolerance for a file storage system.

In [26], Naor and Roth model the network as an undirected graph $G = (V, E)$, where V is a set of nodes and E is a set of bidirectional links between the nodes. A file of f bits is encoded with an error-correcting code, and pieces of the codeword are distributively stored on all nodes of the graph. To the end, every node should be able to recover the file by accessing the codeword pieces stored on itself and its adjacent nodes.

Fig. 1 shows an example of the file storage scheme in [26]. In the example the file is encoded into a codeword of 6 symbols, any 4 of which are sufficient for recovering the file. We represent the 6 distinct symbols with numbers 1, 2, 3, 4, 5, and 6. It can be seen that the placement in Fig. 1 does guarantee that for every node, the data stored on itself and its neighbors are sufficient for the file reconstruction.

In [26] the objective is to minimize the total number of bits stored in the network. Paper [26] presents an algorithm which realizes this objective when $f \gg \log \Delta_G$, where Δ_G is the maximum degree in G . The algorithm has time complexity that is polynomial in f and $|V|$.

Below we first give an overview of our file storage model and our scheme. Then we introduce a new interleaving concept which we call the *K-interleaving*. K-interleaving is a method that can be used to solve the file storage problem on tree networks optimally. It places distinct codeword symbols very closely around every network node, thus minimizes the file retrieving delays.

1.1 Model and Scheme

We generalize the network model in [26] in the following ways. First, we associate every edge $e \in E$ with a length $l(e)$, which represents the delay of transmitting the file through the corresponding link. Next, we associate every node with a storage capacity. The reason for that is because in a network, often not all nodes are appropriate for storing data; and for those nodes that can store data, there may be restrictions on the amount of data they can store. Now given a file, we encode it into a codeword of n symbols which can correct up to ε erasures. In [26], the scheme is to store copies of those n symbols on nodes such that every node is able to recover the file by accessing data stored within one hop. In the generalized network model, that's equivalent to setting all the edges' lengths to be 1, and requiring there to be at least $n - \varepsilon$ distinct symbols within distance 1 from each node. In networks like the Internet, usually not all nodes are sources of requests for the file; and for those nodes that request for the file, the file-retrieving delay they can tolerate varies from node to node. Therefore as a generalization of the storage scheme, we associate every node v with a parameter which is the distance within which from v there should be $n - \varepsilon$ distinct

symbols. That parameter varies from node to node; and for a node that doesn't request for the file, that parameter is naturally infinity.

We further generalize the storage scheme for fault-tolerance and QoS. Define the 'neighborhood of a node v of radius r ' as the set of nodes that are within distance r from v . When there are P ($P \geq n - \varepsilon$) distinct symbols in the neighborhood of node v of radius r , node v can retrieve $n - \varepsilon$ distinct symbols from the network for recovering the file within delay r . If $P > n - \varepsilon$, then node v has the freedom to choose which $n - \varepsilon$ distinct symbols to get from this neighborhood, based on the current network condition. This freedom in adaptively retrieving data from a network is important for balancing load and ensuring high availability in data networks [25] [32]. Also if $P > n - \varepsilon$, it means node v is guaranteed to be able to retrieve $n - \varepsilon$ distinct symbols within delay r when the neighborhood of v of radius r suffers the inaccessibility of no more than $P - (n - \varepsilon)$ symbols (e.g., because of data loss or processors being busy). It's very desirable that the number of distinct symbols in the neighborhood of a node grows steadily when the neighborhood's radius increases, so that the delay that the node needs to tolerate for retrieving $n - \varepsilon$ distinct symbols deteriorates gracefully when more and more symbols stored in the network become inaccessible. In our scheme, we associate every node with a function which specifies the numbers of distinct symbols that should appear in the node's neighborhoods of different radii; and those functions can differ from node to node to better accommodate the varied QoS requirements among nodes.

The optimization criterion we're interested in differs from that of [26]. Paper [26] is interested in minimizing the total number of bits stored in the network, while in this paper we shall focus on minimizing the delays of data retrieving — that is, to minimize delays after ensuring that the delays satisfy the QoS requirements and the data stored on every node don't exceed the node's storage capacity.

The formal definition of the file storage scheme will be presented in Section II. Next we introduce the *K-interleaving*.

1.2 K-Interleaving

First we give the definition.

Definition 1 (K-Interleaving): $G = (V, E)$ is a tree, where each vertex $v \in V$ is associated with a non-negative integer $w(v)$ and each edge $e \in E$ is associated with a positive real number $l(e)$ called its 'length'. Every edge $e \in E$ is seen as a string (or curve) of length $l(e)$, where a point on it is just called a 'point' in the usual sense. (A vertex is also seen as a point.) There are n distinct colors, and K is an integer such that $0 \leq K \leq n$. Assign $w(v)$ colors to each vertex $v \in V$. For any point p in G and any integer i ($0 \leq i \leq n$), define $R_{min}(p, i)$ as the minimum value of r such that the vertices at distance no greater than r from p are assigned no less than i colors (no matter if those colors are all distinct or not), and define $R'_{min}(p, i)$ as the minimum value of r such that the vertices at distance no greater than r from p are assigned no less than i distinct colors. If $R_{min}(p, i) = R'_{min}(p, i)$ for any point p and any $0 \leq i \leq K$, then the coloring is called a *K-interleaving*. \square

The following is an example of the K-interleaving.

Example 1: Fig. 2 shows a K-interleaving where $K = 8$. The parameters $w(v)$ for all vertices v are shown in the figure, and $n = 8$ and $K = 8$. The real number beside each edge e is its length $l(e)$. We denote the 8 colors by the numbers 1, 2, \dots , 8. The colors assigned to every vertex is shown in the set beside that vertex (e.g., vertex v_1 has color 2, 4, 6 and 8). It can be verified that $R_{min}(p, i) = R'_{min}(p, i)$ for any point p and any $0 \leq i \leq K$. For example, if we let p be the point on the edge between v_3 and v_4 which is at distance 0.8 to v_4 , and let $i = 3$, then $R_{min}(p, i) = 0.8$ and $R'_{min}(p, i) = 0.8$, so $R_{min}(p, 3) = R'_{min}(p, 3)$. (The vertices within distance 0.8 from p are v_3 and v_4 , which have colors 3, 4 and 2, 6.) Similarly, $R_{min}(p, 8) = R'_{min}(p, 8) = 1.2$. So this coloring is an 8-interleaving. Clearly it's also a 7-interleaving, 6-interleaving, \dots , 0-interleaving.

\square

If we understand the n colors as the n symbols in an (n, k) codeword, and understand the tree $G = (V, E)$ as a network with link delays $l(e)$ ($e \in E$) and nodes' storage capacities $w(v)$ ($v \in V$), then a K-interleaving is just a

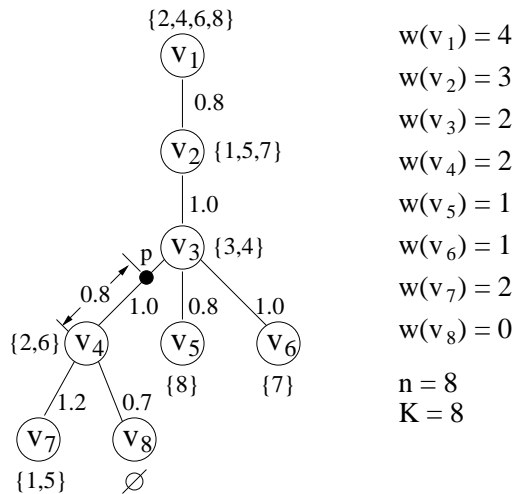


Figure 2: An example of the K -interleaving.

codeword layout which compactly places at least K distinct symbols around every ‘point’ — including the nodes.

There have been a list of interleaving techniques in literature for codeword layout [1] [2] [5] [8] [10] [34], which are mainly for correcting error-bursts of different shapes on one-, two- or three-dimensional arrays. K -interleaving adds to this list as a novel interleaving scheme, and it is for trees. Note that the commonly used one-dimensional interleaving of codewords, which is for combatting burst errors in channels and is of the form ‘-1-2-3-...-n-1-2-3-...-n-’ when n codewords are interleaved, is a special case of the K -interleaving where the tree is reduced to a linear array and K is set to be equal to n .

The importance of K -interleaving is that it can help us realize the file storage scheme being studied in this paper on tree networks, as long as the value of K is set to be large enough. It will be shown that realizing the file storage scheme in general is NP-hard. But when the network is a tree, the scheme can be realized in polynomial time, because as will be shown, a K -interleaving always exists for any K , and it can be computed efficiently. What’s more, by setting K to be equal to n , a K -interleaving corresponds to a *best* solution in the sense that it simultaneously minimizes the delay for *any* node to retrieve *any* number of distinct symbols, because with an n -interleaving, for every node all the n distinct symbols are placed around it as closely as possible. As we will focus a lot on solving the file storage scheme for tree networks, K -interleaving will be studied extensively in this paper.

The rest of this paper is organized as follows. In Section II, the new file storage scheme is formally defined, which is then formulated as a graph coloring problem which we call the *diversity coloring problem*. The diversity coloring problem is shown to be NP-hard for general graphs. In Section III, a K -interleaving algorithm is presented for solving the diversity coloring problem on trees. The algorithm has time complexity $O(|V|^2 + \mathbb{W}|V| + \mathbb{W}K)$, where $|V|$ is the number of vertices in the tree and \mathbb{W} is the total number of colors — distinct or non-distinct — assigned to the tree. The algorithm is *general* in the sense that every K -interleaving is one of the algorithm’s possible outputs. The K -interleaving technique can be used to find both *perfect* diversity coloring — defined as diversity coloring that uses the absolutely minimum number of colors — and *absolutely optimum* file storage solutions — defined as solutions that simultaneously minimize the delay of retrieving any number of distinct codeword symbols by any node in the network. In Section IV, a more efficient K -interleaving algorithm is presented, which has time complexity $O(|V|^2 + \mathbb{W})$. Discussions on diversity coloring on cyclic graphs are presented in Section V. The concluding remarks are presented in Section VI.

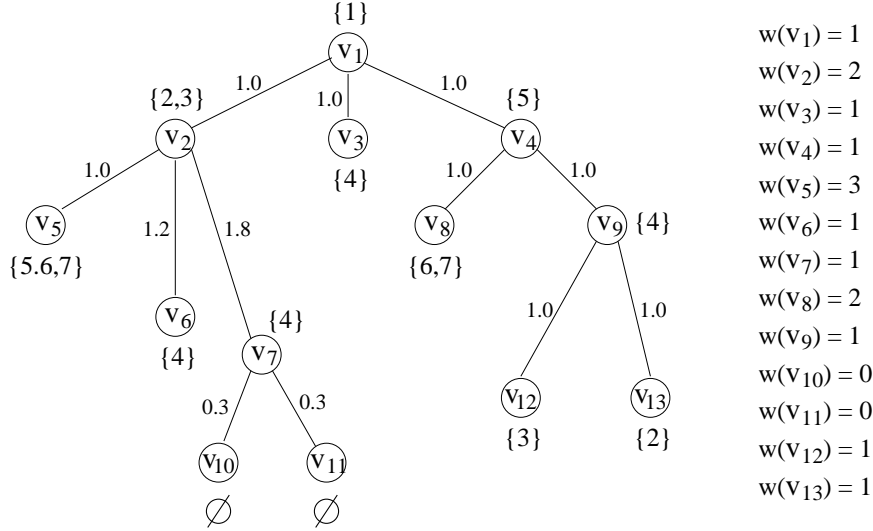


Figure 3: An example of the file storage scheme.

II. FILE STORAGE SCHEME AND DIVERSITY COLORING

2.1 File Storage Scheme

In this section we formally define our file storage scheme for distributively storing an encoded file in a network. We model the network as an undirected graph $G = (V, E)$, where V is a set of nodes and E is a set links between the nodes. A file is encoded into a codeword of n symbols, any $n - \varepsilon$ of which are sufficient for recovering the file.

Each node $v \in V$ is associated with a ‘storage capacity’ $w(v)$. $w(v)$ is an integer and is the maximum number of symbols that can be stored on the node v . Each edge $e \in E$ is associated with a ‘length’ $l(e)$. $l(e)$ is a positive real number and represents the delay of transmitting data over the link e . For any two nodes u and v , we define $d(u, v)$ as the length of the shortest path between u and v , and call it ‘the distance between u and v ’. (If u and v are the same node, then $d(u, v) = 0$.) $d(u, v)$ represents the delay of transmitting data from u to v , or from v to u .

Given a node $v \in V$ and a real number r , we define $\mathcal{N}(v, r)$ as the set of nodes that are within distance r from v , that is, $\mathcal{N}(v, r) = \{u | u \in V, d(u, v) \leq r\}$. We call $\mathcal{N}(v, r)$ the ‘neighborhood of v of radius r ’.

Every node $v \in V$ is associated with a function $M(v, r)$, which is the number of distinct symbols that need to appear in $\mathcal{N}(v, r)$. We can see $M(v, r)$ as a specification of the QoS requirements of node v . As explained in Section I, it’s desirable that the value of $M(v, r)$ would increase steadily as r increases until $M(v, r)$ becomes equal to n , in order to facilitate adaptive data-retrieval and guarantee performance in faulty environments.

The file storage scheme is defined as follows.

Definition 2 (The File Storage Scheme): Assign up to $w(v)$ symbols to each node $v \in V$, such that for any $u \in V$ and any $r \in \mathbb{R}$, nodes in $\mathcal{N}(u, r)$ store no less than $M(u, r)$ distinct symbols in total. (Here for any $u \in V$ and any $r \in \mathbb{R}$, $\sum_{v \in \mathcal{N}(u, r)} w(v) \geq M(u, r)$ and $0 \leq M(u, r) \leq n$.) \square

An example of the above file storage scheme is shown below.

Example 2: A graph $G = (V, E)$ with 13 vertices is shown in Fig. 3. A file is encoded into a codeword of 7 symbols, any 3 of which are sufficient for recovering the file. For $1 \leq i \leq 13$, $w(v_i)$ is shown in Fig. 3, and the number beside each edge is the edge’s length.

For $1 \leq i \leq 4$,

$$M(v_i, r) = \begin{cases} 0 & \text{if } r < 2.0 \\ 3 & \text{if } 2.0 \leq r < 2.5 \\ 5 & \text{if } 2.5 \leq r < 3.0 \\ 7 & \text{if } r \geq 3.0 \end{cases}$$

For $5 \leq i \leq 7$,

$$M(v_i, r) = \begin{cases} 0 & \text{if } r < 2.0 \\ 3 & \text{if } 2.0 \leq r < 3.0 \\ 6 & \text{if } r \geq 3.0 \end{cases}$$

For $8 \leq i \leq 9$, $M(v_i, r) = 0$ for any r .

For $10 \leq i \leq 13$,

$$M(v_i, r) = \begin{cases} 0 & \text{if } r < 2.5 \\ 3 & \text{if } 2.5 \leq r < 3.5 \\ 7 & \text{if } r \geq 3.5 \end{cases}$$

We use integers ‘1, 2, ..., 7’ to denote the 7 symbols. A solution to the file storage scheme is shown in Fig. 3, where the set of integers beside each node indicates the symbols assigned to it — e.g., the node v_2 is assigned symbol ‘2’ and symbol ‘3’. It’s easy to verify that all the conditions in Definition 2 are satisfied here. For example, the nodes in $\mathcal{N}(v_1, 2.0)$ store $7 \geq M(v_1, 2.0) = 3$ distinct symbols in total.

□

For any node v and integer P , the delay for v to retrieve P distinct symbols equals the minimum value of r such that the nodes in $\mathcal{N}(v, r)$ store no less than P distinct symbols in total. If we use $R_{min}(v, P)$ to denote the minimum value of r such that $\sum_{u \in \mathcal{N}(v, r)} w(u) \geq P$, then clearly the delay for v to retrieve P distinct symbols must be no less than $R_{min}(v, P)$. If there exists a solution in which the delay for v to retrieve P distinct symbols equals $R_{min}(v, P)$ for all $v \in V$ and for all integer P , then we call such a solution ‘*absolutely optimum*’. Clearly an *absolutely optimum* solution has superb performance in the sense that it *simultaneously* minimizes the delay for *any* node to retrieve *any* number of distinct symbols. It’s generally very unlikely for an ‘absolutely optimum’ solution to exist if the graph has an irregular structure. However, it’ll be proved later in this paper that if the graph is a tree, then an ‘absolutely optimum’ solution always exists.

2.2 Diversity Coloring

By using n colors to represent the n codeword symbols, we can formulate the file storage scheme in Definition 2 as a graph coloring problem. In this paper we focus on finding file storage solutions that minimize data-retrieving delay, which benefits from assigning as many symbols to each node as possible. Therefore we define the graph coloring problem as follows, and call it the ‘*diversity coloring problem*’.

Definition 3 (The Diversity Coloring Problem):

INSTANCE: n colors and a graph $G = (V, E)$. Every edge $e \in E$ has a positive length $l(e)$. Every vertex $v \in V$ is associated with an integer $w(v)$ and a function $M(v, r)$. (Here for any $v \in V$ and any real number r , $0 \leq w(v) \leq n$, $0 \leq M(v, r) \leq n$, and $\sum_{u \in \mathcal{N}(v, r)} w(u) \geq M(v, r)$.)

QUESTION: How to assign $w(v)$ colors to each vertex $v \in V$, such that for any $u \in V$ and any real number r , the vertices in $\mathcal{N}(u, r)$ has no less than $M(u, r)$ distinct colors in total? (Such a coloring is called a ‘*diversity coloring*’. Here the same color can be assigned to different vertices.) □

In graph theory, it’s usually interesting to find a coloring that uses as few different colors as possible. Clearly if the graph $G = (V, E)$ has a diversity coloring, then in G there are at least $\max_{v \in V, r \in \mathbb{R}} M(v, r)$ distinct colors. If in G there are exactly $\max_{v \in V, r \in \mathbb{R}} M(v, r)$ distinct colors, then the diversity coloring is called a ‘*perfect*’ diversity coloring.

We'll show that the diversity coloring problem is NP-hard for fixed $n \geq 2$. (Clearly this problem is trivial when $n = 1$.) Before that, we first introduce two known NP-complete problems.

Definition 4 (The Graph K -Colorability Problem) [12]:

INSTANCE: A graph $G = (V, E)$. A positive integer $K \leq |V|$.

QUESTION: Is G K -colorable, that is, does there exist a function $f : V \rightarrow \{1, 2, \dots, K\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

Comment: This problem is solvable in polynomial time for $K = 2$, but remains NP-complete for all fixed $K \geq 3$.

□

Definition 5 (The Set Splitting Problem) [12]:

INSTANCE: A collection C of subsets of a finite set S .

QUESTION: Is there a partition of S into two subsets S_1 and S_2 such that no subset in C is entirely contained in either S_1 or S_2 ? □

Now we prove the NP-hardness of the diversity coloring problem.

Theorem 1 *The diversity coloring problem is NP-hard for fixed $n \geq 2$.*

Proof: (i) We first consider the case where n is fixed and $n > 2$.

Let $G = (V, E)$ be a graph, and let $2 < n \leq |V|$ be a positive integer. Assume $V = \{v_1, v_2, \dots, v_{|V|}\}$, and $E = \{e_1, e_2, \dots, e_{|E|}\}$. We construct a graph $H = (V_0, E_0)$ in the following way: $V_0 = \{\hat{v}_1, \hat{v}_2, \dots, \hat{v}_{|V|}, u_1, u_2, \dots, u_{|E|}\}$; for any $1 \leq i \neq j \leq |V|$, $(\hat{v}_i, \hat{v}_j) \in E_0$ if and only if $(v_i, v_j) \in E$; for any $1 \leq i \leq |V|$ and $1 \leq j \leq |E|$, $(\hat{v}_i, u_j) \in E_0$ if and only if in the graph G , v_i is an endpoint of e_j ; for any $1 \leq i \neq j \leq |E|$, $(u_i, u_j) \notin E_0$.

We let all edges of H be of length 1. For $1 \leq i \leq |V|$, we let $w(\hat{v}_i) = 1$; for $1 \leq i \leq |E|$, we let $w(u_i) = 0$. For $1 \leq i \leq |V|$, we let $M(\hat{v}_i, r) = 0$ for any r . For $1 \leq i \leq |E|$, we let $M(u_i, r) = 0$ for any $r < 1$, and let $M(u_i, r) = 2$ for any $r \geq 1$.

Assume we're given n colors, and we assign one color to each vertex \hat{v}_i ($1 \leq i \leq |V|$) and assign no color to any vertex u_j ($1 \leq j \leq |E|$); for $1 \leq i \leq |V|$, we let v_i has the same color as \hat{v}_i . Clearly, every edge in G has two different colors on its endpoints if and only if in the graph H , vertices in $\mathcal{N}(u_i, 1)$ have two different colors for any $1 \leq i \leq |E|$. So G is n -colorable if and only if H has a diversity coloring with n colors. So the K -colorability problem can be reduced to the diversity coloring problem where $n > 2$.

(ii) Next we consider the case where $n = 2$.

Let C be a collection of subsets of a finite set S . Assume $C = \{c_1, c_2, \dots, c_{|C|}\}$, and $S = \{s_1, s_2, \dots, s_{|S|}\}$. We construct a bipartite graph $G = (V, E)$ in the following way: $V = \{v_1, v_2, \dots, v_{|C|}, u_1, u_2, \dots, u_{|S|}\}$; for any $1 \leq i \leq |C|$ and $1 \leq j \leq |S|$, there is an edge between v_i and u_j if and only if $s_j \in c_i$.

We let all edges of G be of length 1. For $1 \leq i \leq |C|$, we let $w(v_i) = 0$; for $1 \leq i \leq |S|$, we let $w(u_i) = 1$. For $1 \leq i \leq |S|$, we let $M(u_i, r) = 0$ for any r . For $1 \leq i \leq |C|$, we let $M(v_i, r) = 0$ for any $r < 1$, and let $M(v_i, r) = 2$ for any $r \geq 1$.

Assume we're given $n = 2$ colors, and we assign one color to each vertex u_i ($1 \leq i \leq |S|$) and assign no color to any vertex v_j ($1 \leq j \leq |C|$). And we split S into two subsets S_1 and S_2 in this way: for any $1 \leq i \neq j \leq |S|$, s_i and s_j are both in S_1 or both in S_2 if and only if u_i and u_j are assigned the same color. Clearly, no subset in C is entirely contained in either S_1 or S_2 if and only if for any $1 \leq i \leq |C|$, vertices in $\mathcal{N}(v_i, 1)$ have 2 distinct colors—which means the coloring is a diversity coloring on G . So the set splitting problem can be reduced to the diversity coloring problem where $n = 2$.

(iii) Whether or not a coloring is a diversity coloring can be verified in polynomial time. So the diversity coloring problem is NP-hard for fixed $n \geq 2$.

□

III. K-INTERLEAVING FOR DIVERSITY COLORING ON TREES

Trees are important network structures. For example, trees are often used by backbone networks, by virtual private networks (VPNs) [22], or as embedded networks for data storage [33]. In this section, the relationship between diversity coloring on trees and a special kind of interleaving called ‘*K-interleaving*’ will be revealed, and a general K-interleaving algorithm will be presented. First we introduce some basic terms.

3.1 Terms

A *point* in a tree $G = (V, E)$ is defined as a point on one of its edges. If $e \in E$ is an edge and p is a point on e , then p is at distance $l(e) \cdot \lambda$ from one of e ’s endpoints and is at distance $l(e) \cdot (1 - \lambda)$ from the other endpoint of e , for some $\lambda \in [0, 1]$. (Here $l(e)$ is the length of e . And as a special case, a vertex is also a point.) For any two points p_1 and p_2 in the tree, $d(p_1, p_2)$ is defined as the length of the shortest path between p_1 and p_2 , and is called the ‘*distance between p_1 and p_2* ’. For any point p and any real number r , $\mathcal{N}(p, r)$ is defined as the set of vertices within distance r from p , namely, $\mathcal{N}(p, r) = \{v | v \in V, d(v, p) \leq r\}$.

For any two points p_1 and p_2 , $\mathcal{C}(p_1, p_2)$ is defined as the unique point at distance $\frac{d(p_1, p_2)}{2}$ from both p_1 and p_2 — that is, $\mathcal{C}(p_1, p_2)$ is the middle-point of the path between p_1 and p_2 . $\mathcal{S}(p_1, p_2)$ is defined as the set of vertices within distance $\frac{d(p_1, p_2)}{2}$ from $\mathcal{C}(p_1, p_2)$, namely, $\mathcal{S}(p_1, p_2) = \mathcal{N}(\mathcal{C}(p_1, p_2), \frac{d(p_1, p_2)}{2})$.

In the diversity coloring problem, each vertex $v \in V$ is to be assigned $w(v)$ colors. We say each vertex v has $w(v)$ *color-slots*. Then assigning $w(v)$ colors to v is equivalent to assigning one color to each of v ’s color-slots. By convention, if $S \subseteq V$ is a set of vertices and $v \in S$, then we say each of v ’s color-slots is also in S . Also by convention, we say the *distance* between two color-slots s' and s'' is the distance between the two vertices that the two color-slots respectively belong to, and denote it by $d(s', s'')$; and similarly, the *distance* between a color-slot s and a point p is the distance between p and the vertex that s belongs to, and is denoted by $d(s, p)$ or $d(p, s)$.

3.2 K-Interleaving

The following is another way to define K-interleaving. It’s clearly equivalent to Definition 1.

Definition 6 (K-Interleaving): $G = (V, E)$ is a tree, where every edge $e \in E$ has a length $l(e)$, and every vertex $v \in V$ has $w(v)$ color-slots. K and n are two integers, and $0 \leq K \leq n$. Given n colors, a coloring which assigns one color to each color-slot is called a *K-Interleaving* if and only if for any point p in the tree and for any real number r , either all the colors assigned to the color-slots in $\mathcal{N}(p, r)$ are distinct, or the color-slots in $\mathcal{N}(p, r)$ are assigned at least K distinct colors. \square

The following lemma shows how for every point in the tree, a K-interleaving compactly places at least K distinct colors around it. Its proof is omitted due to its simplicity.

Lemma 1 *For every point p in the tree $G = (V, E)$, define $R_{min}(p, K)$ as the minimum value of r such that $\sum_{v \in \mathcal{N}(p, r)} w(v) \geq K$. (If $\sum_{v \in V} w(v) < K$, then $R_{min}(p, K) = +\infty$.) Then with a K-interleaving, for any point p and any fixed real number r , all the colors of the color-slots in $\mathcal{N}(p, r)$ are distinct if $0 \leq r < R_{min}(p, K)$, and the color-slots in $\mathcal{N}(p, r)$ have at least K distinct colors if $r \geq R_{min}(p, K)$.*

The following theorem shows the relationship between K-interleaving and diversity color.

Theorem 2 *For the diversity coloring problem for a tree, a K-interleaving is also a diversity coloring if $K \geq \max_{v \in V, r \in \mathbb{R}} M(v, r)$.*

Proof: For every point p in the tree $G = (V, E)$, define $R_{min}(p, K)$ as the minimum value of r such that $\sum_{v \in \mathcal{N}(p, r)} w(v) \geq K$. Assume there is a coloring on G which is a K-interleaving with $K \geq \max_{v \in V, r \in \mathbb{R}} M(v, r)$. By Lemma 1 and Definition 3, for any vertex $v \in V$, vertices in $\mathcal{N}(v, r)$ have $\sum_{u \in \mathcal{N}(v, r)} w(u) \geq M(v, r)$ distinct colors when $r < R_{min}(v, K)$, and have at least $K \geq M(v, r)$ distinct colors when $r \geq R_{min}(v, K)$. So the coloring is also a diversity coloring. So Theorem 2 is proved. \square

Theorem 2 shows that a K -interleaving with a sufficiently large K ensures a diversity coloring on a tree. But does such a K -interleaving always exist? The following of the paper will give a positive answer to this question.

3.3 A General K -Interleaving Algorithm

In this subsection we present a general K -interleaving algorithm. The algorithm is ‘general’ in the sense that the solution space of the K -interleaving problem is exactly the set of possible outputs of the algorithm.

Let $G = (V, E)$ be the tree for which the algorithm is to output a K -interleaving. Denote the root of G by γ , and define $\mathbb{W} = \sum_{v \in V} w(v)$. G has \mathbb{W} color-slots, which are labelled as $s_1, s_2, \dots, s_{\mathbb{W}}$ following this rule: $d(\gamma, s_i) \leq d(\gamma, s_j)$ for any $1 \leq i < j \leq \mathbb{W}$. The algorithm is as follows.

Algorithm 1 [General K -interleaving Algorithm for Tree $G = (V, E)$]

1. Assign K distinct colors to the following K color-slots: s_1, s_2, \dots, s_K , such that no two color-slots among them are assigned the same color. (If there are less than K color-slots in the tree, then simply assign \mathbb{W} distinct colors to all the color-slots, such that no two color-slots are assigned the same color.)

2. For $i = K + 1$ to \mathbb{W} do

{ Let r_{min} denote the smallest value of r such that the set $\{s_j | 1 \leq j \leq i - 1, d(s_i, s_j) \leq r\}$ contains no less than K color-slots.

Assign a color to s_i , such that no color-slot in the set $\{s_j | 1 \leq j \leq i - 1, d(s_i, s_j) < r_{min}\}$ is assigned the same color as s_i .

}

□

Analysis shows that Algorithm 1 has time complexity $O(|V|^2 + \mathbb{W}|V| + \mathbb{W}K)$. We present the complexity analysis in Appendix I.

Below we start proving the correctness of Algorithm 1.

Lemma 2 *Algorithm 1 is used to produce a coloring for the tree $G = (V, E)$. Let’s say s_a is a color-slot of vertex v_A , and s_b is a color-slot of vertex v_B . (Here $1 \leq a \neq b \leq \mathbb{W}$.) If s_a and s_b are assigned the same color, then the color slots in $\mathcal{S}(v_A, v_B)$ are assigned no less than K distinct colors.*

Proof: Without loss of generality, we assume $a < b$. We’ll prove by induction that the lemma holds for any $b \leq \mathbb{W}$.

The K color-slots s_1, s_2, \dots, s_K are all assigned distinct colors. As a null case, the lemma holds when $b \leq K$. We use this as the base case for the induction.

Let i be an integer such that $K + 1 \leq i \leq \mathbb{W}$. Assume that when $b < i$, the lemma holds. Let’s prove that the lemma also holds when $b = i$.

Assume $b = i$; and assume s_a and s_b are assigned the same color. Let \hat{r}_{min} denote the smallest value of r such that the set $\{s_j | 1 \leq j \leq b - 1, d(s_b, s_j) \leq r\}$ contains no less than K color slots. According to Algorithm 1, no color-slot in the set $\{s_j | 1 \leq j \leq b - 1, d(s_b, s_j) < \hat{r}_{min}\}$ is assigned the same color as s_b is. Therefore $d(s_a, s_b) \geq \hat{r}_{min}$. Since $a < b$, the distance between the root and v_A is no greater than the distance between the root and v_B . Therefore the unique point at distance $\frac{d(v_A, v_B)}{2}$ from both v_A and v_B , $\mathcal{C}(v_A, v_B)$, lies on the path between the root and v_B . Define Q as $Q = \{s_j | 1 \leq j \leq b - 1, d(s_b, s_j) \leq \hat{r}_{min}\}$. Consider any color-slot in Q — say it’s s_c and is a color-slot of vertex v_C . Clearly $c < b$. If $\mathcal{C}(v_A, v_B)$ lies on the path between the root and v_C , then $d(\mathcal{C}(v_A, v_B), v_C) \leq d(\mathcal{C}(v_A, v_B), v_B) = \frac{d(v_A, v_B)}{2}$, because the distance between the root and v_C is no greater than the distance between the root and v_B . If $\mathcal{C}(v_A, v_B)$ doesn’t lie on the path between the root and v_C , then again $d(\mathcal{C}(v_A, v_B), v_C) \leq \frac{d(v_A, v_B)}{2}$, because $d(v_C, v_B) \leq \hat{r}_{min} \leq d(v_A, v_B)$ and $d(v_C, v_B) = d(v_C, \mathcal{C}(v_A, v_B)) + d(\mathcal{C}(v_A, v_B), v_B) = d(v_C, \mathcal{C}(v_A, v_B)) + \frac{d(v_A, v_B)}{2}$. So s_c is in the set $\mathcal{S}(v_A, v_B)$. So all the color-slots in Q are in $\mathcal{S}(v_A, v_B)$.

There are at least K color-slots in Q . If all the color-slots in Q have distinct colors, then clearly the color-slots in $\mathcal{S}(v_A, v_B)$ are assigned no less than K distinct colors. Now consider the case where there are two color-slots in Q that have the same color — say those two color-slots are s_f and s_h , which belong to vertex v_F and v_H respectively.

Clearly $f < b$, $h < b$, and both v_F and v_H are in $\mathcal{S}(v_A, v_B)$. Let v_T denote the unique vertex that lies on the following three paths: the path between $\mathcal{C}(v_A, v_B)$ and v_F , the path between $\mathcal{C}(v_A, v_B)$ and v_H , and the path between v_F and v_H . The point $\mathcal{C}(v_F, v_H)$ lies on the path between v_F and v_H ; without loss of generality, let's say $\mathcal{C}(v_F, v_H)$ lies on the path between v_F and v_T . For any color-slot in $\mathcal{S}(v_F, v_H)$ — say it's s_j , which belongs to vertex v_J — we have $d(\mathcal{C}(v_A, v_B), v_J) \leq d(\mathcal{C}(v_A, v_B), \mathcal{C}(v_F, v_H)) + d(\mathcal{C}(v_F, v_H), v_J) \leq d(\mathcal{C}(v_A, v_B), \mathcal{C}(v_F, v_H)) + \frac{d(v_F, v_H)}{2} = d(\mathcal{C}(v_A, v_B), v_T) + d(v_T, \mathcal{C}(v_F, v_H)) + d(\mathcal{C}(v_F, v_H), v_F) = d(\mathcal{C}(v_A, v_B), v_F) \leq \frac{d(v_A, v_B)}{2}$, and therefore s_j is in $\mathcal{S}(v_A, v_B)$. By the induction assumption, the color-slots in $\mathcal{S}(v_F, v_H)$ are assigned no less than K distinct colors. So the color-slots in $\mathcal{S}(v_A, v_B)$ are also assigned no less than K distinct colors.

So the lemma holds when $b = i$. And the proof is completed.

□

Lemma 3 *Assume there is a coloring on the tree $G = (V, E)$, which uses n colors and assigns $w(v)$ colors to every vertex $v \in V$. The coloring is a K -interleaving if and only if the following is true: for any two color-slots s_a and s_b — say they are color-slots of vertex v_A and v_B respectively — if s_a and s_b are assigned the same color, then the color slots in $\mathcal{S}(v_A, v_B)$ are assigned no less than K distinct colors. (Here $1 \leq a \neq b \leq \mathbb{W}$.)*

Proof: First let's prove one direction. Assume the coloring is a K -interleaving, and s_a and s_b are assigned the same color. $\mathcal{S}(v_A, v_B) = \mathcal{N}(\mathcal{C}(v_A, v_B), \frac{d(v_A, v_B)}{2})$, and both s_a and s_b are in $\mathcal{S}(v_A, v_B)$. So by Definition 6, the color-slots in $\mathcal{S}(v_A, v_B)$ are assigned no less than K distinct color.

Next let's prove the other direction. Assume that for any two color-slots s_a and s_b , which belong to vertex v_A and v_B respectively, if s_a and s_b are assigned the same color, then the color-slots in $\mathcal{S}(v_A, v_B)$ are assigned no less than K distinct colors. Let p be an arbitrary point in G , and let r be an arbitrary real number. If not all the color-slots in $\mathcal{N}(p, r)$ are assigned distinct colors, then let s_a and s_b be two color-slots of the same color in $\mathcal{N}(p, r)$. For any vertex $v \in \mathcal{S}(v_A, v_B)$, we have $d(v, p) \leq d(v, \mathcal{C}(v_A, v_B)) + d(\mathcal{C}(v_A, v_B), p) \leq \frac{d(v_A, v_B)}{2} + d(\mathcal{C}(v_A, v_B), p) = \max\{d(v_A, p), d(v_B, p)\} \leq r$. Therefore $\mathcal{S}(v_A, v_B) \subseteq \mathcal{N}(p, r)$. So the color-slots in $\mathcal{N}(p, r)$ are assigned at least K distinct colors. So by Definition 6, the coloring is a K -interleaving.

□

Lemma 2 and Lemma 3 together naturally establish the following conclusion.

Theorem 3 *Algorithm 1 correctly outputs a K -interleaving for the tree $G = (V, E)$.*

Since Algorithm 1 always has an output, we have the following corollary.

Corollary 1 *K -interleaving always exists for trees.*

By Theorem 2, Algorithm 1 can be used to find diversity coloring for trees if $\max_{v \in V, r \in \mathbb{R}} M(v, r) \leq K \leq n$. Assume we have the freedom to choose the values of n and K . Then if we enforce that $n = K = \max_{v \in V, r \in \mathbb{R}} M(v, r)$, Algorithm 1 will output a *perfect* diversity coloring. Note that in this case, the value of n is minimized, where n corresponds to the length of the codeword in the file-storage scheme. We summarize the results in the following corollaries.

Corollary 2 *Algorithm 1 outputs a diversity coloring when $\max_{v \in V, r \in \mathbb{R}} M(v, r) \leq K \leq n$. And Algorithm 1 outputs a perfect diversity coloring when $\max_{v \in V, r \in \mathbb{R}} M(v, r) = K = n$.*

Corollary 3 *Diversity coloring always exists for trees. And perfect diversity coloring always exists for trees.*

3.4 Generality of the General K -Interleaving Algorithm

Different K -interleavings can exist for the same tree, as the following example shows.

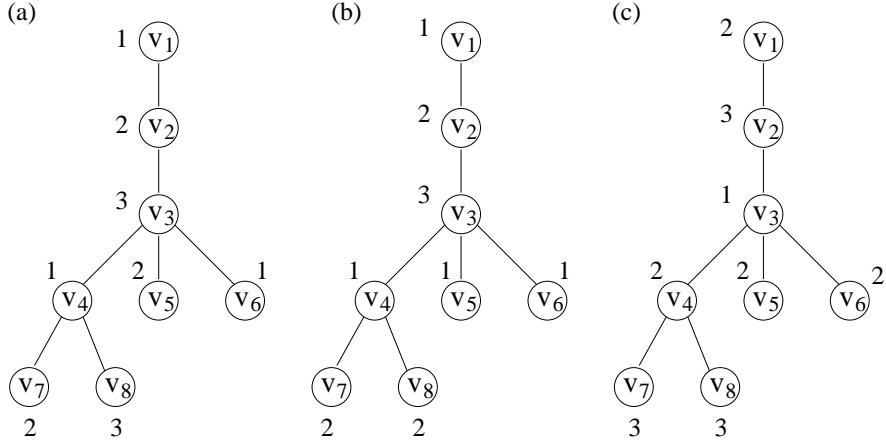


Figure 4: Different K-interleavings on the same tree.

Example 3: Fig. 4 (a), (b) and (c) show the same tree $G = (V, E)$. In the tree, all edges have the same length, and $w(v) = 1$ for any vertex $v \in V$. $n = 3$ colors — denoted by the numbers ‘1’, ‘2’ and ‘3’ — are used to color the tree, and three colorings are shown in (a), (b) and (c) respectively. (The number beside each vertex is the color assigned to it.) It’s simple to verify that each coloring is a K-interleaving where $K = 3$. Clearly the colorings in (a) and (b) can’t be derived from each other through permutation of colors; but the colorings in (b) and (c) can, e.g., the coloring in (c) can be got by changing the colors ‘1’, ‘2’ and ‘3’ in (b) into colors ‘2’, ‘3’ and ‘1’ respectively. \square

When Algorithm 1 colors a tree, first it needs to label the \mathbb{W} color-slots as $s_1, s_2, \dots, s_{\mathbb{W}}$, and then it assigns colors to those color-slots in order. The labelling is usually not unique; and every time the algorithm assigns a color to a color-slot, the color can usually be chosen from a set of more than one color. By labelling the color-slots in different ways and by choosing the color differently while coloring a color-slot, Algorithm 1 can output different K-interleavings. The following theorem shows that in fact, every K-interleaving that exists is a possible output of Algorithm 1, — namely, the set of possible outputs of Algorithm 1 is exactly the whole set of K-interleavings, — and that property holds even if the root of the tree is fixed. (Fixing the root of the tree lessens the number of ways to label the color-slots.)

Theorem 4 *The set of possible outputs of Algorithm 1 is exactly the whole set of K-interleavings for the tree $G = (V, E)$, even if the root of the tree is fixed.*

Proof: We present a sketch of the proof here. A detailed proof for this theorem is presented in Appendix II. By Theorem 3, the set of possible outputs of Algorithm 1 is a subset of the set of K-interleavings. So the only thing left to prove is that every K-interleaving is a possible output of Algorithm 1, even if the root of the tree is fixed.

Assume the root of the tree is fixed. Assume a fixed K-interleaving for the tree $G = (V, E)$ is given. If $\sum_{v \in V} w(v) \leq K$, it’s simple to see that the K-interleaving must have assigned \mathbb{W} distinct colors to the \mathbb{W} color-slots, which is clearly a possible output of Algorithm 1. So from now on we only consider the case where $\mathbb{W} > K$. We’ll prove by induction that for $1 \leq i \leq \mathbb{W}$, there is a set of i color-slots that Algorithm 1 can label as s_1, s_2, \dots, s_i and assign the same colors to as the given K-interleaving does.

Let γ denote the fixed root of the tree. For $1 \leq j \leq \mathbb{W}$, let $R_{min}(\gamma, j)$ denote the minimum value of r such that $\sum_{v \in \mathcal{N}(\gamma, r)} w(v) \geq j$. By Lemma 1, there exist a set of K color-slots that are assigned distinct colors in the given K-interleaving and are at distance no greater than $R_{min}(\gamma, K)$ from γ , and every color-slot at distance less than $R_{min}(\gamma, K)$ from γ is contained in that set. Clearly it’s feasible for Algorithm 1 to label those color-slots as s_1, s_2, \dots, s_K , and assign them the same colors as the given K-interleaving does. So the induction is true for $1 \leq i \leq K$.

Let I be a fixed integer such that $K \leq I < \mathbb{W}$. Assume the following induction assumption is true: when $i = I$, there is a set C of i color-slots which Algorithm 1 can label as s_1, s_2, \dots, s_i and assign the same colors to as the given K-interleaving does. We will prove that this induction assumption is also true when $i = I + 1$.

Let D denote the set of all the \mathbb{W} color-slots in the tree. Let H denote the set of color-slots in $D - C$ at distance $R_{min}(\gamma, I + 1)$ from γ . (There are I color-slots in C . Clearly $H \neq \emptyset$.) Denote the color-slots in H as $c_1, c_2, \dots, c_{|H|}$. For $1 \leq p \leq |H|$, define $r_{min}(p)$ as the smallest value of r such that there are no less than K color-slots in C at distance no greater than r from c_p .

Randomly pick a color-slot c_{j_1} from H . (Here $1 \leq j_1 \leq |H|$.) If at least one color-slot in C at distance less than $r_{min}(j_1)$ from c_{j_1} is assigned the same color as c_{j_1} in the given K-interleaving, then let $t_{k_1} \in C$ denote the color-slot closest to c_{j_1} which is assigned the same color as c_{j_1} in the given K-interleaving. Say c_{j_1} is a color-slot of vertex u_1 , and t_{k_1} is a color-slot of vertex v_1 . Let J denote the set of color-slots in $\mathcal{S}(u_1, v_1)$. The given K-interleaving assigns at least K distinct colors to the color slots in J ; and $J = (J \cap C) \cup (J \cap H)$. Any color-slot in $J \cap C$ is at distance less than $r_{min}(j_1)$ from c_{j_1} , so there exists a color-slot in $J \cap H$ which has a color different from any color-slot in $J \cap C$ in the given K-interleaving — and we denote that color-slot by c_{j_2} . If at least one color-slot in C at distance less than $r_{min}(j_2)$ from c_{j_2} is assigned the same color as c_{j_2} in the given K-interleaving, then some color-slot c_{j_3} can be found through c_{j_2} in the same way the color-slot c_{j_2} is found through c_{j_1} . This process can keep going on and a series of color slots $c_{j_1}, c_{j_2}, c_{j_3}, c_{j_4}, \dots$ will be found. It can be shown that all those color-slots are distinct. Therefore this series is not infinitely long, and eventually some color-slot c_{j_x} ($x \geq 1$) will be found such that no color-slot in C at distance less than $r_{min}(j_x)$ from c_{j_x} is assigned the same color as c_{j_x} in the given K-interleaving. Then it's simple to see that it is feasible for Algorithm 1 to label c_{j_x} as the color-slot s_{I+1} and assign s_{I+1} the same color as the given K-interleaving does, after labelling the color-slots in C as s_1, s_2, \dots, s_I and assigning the same colors to them as the given K-interleaving does. So the induction is also true when $i = I + 1$.

The proof by induction is complete here. Now it's clear that Algorithm 1 can assign the same colors to all the \mathbb{W} color-slots as the arbitrarily given K-interleaving does. So any K-interleaving is a possible output of Algorithm 1, even if the root of the tree is fixed. Therefore this theorem is proved.

□

3.5 Absolutely Optimum Data Storage with K-Interleaving

Say there is a K-interleaving on a tree $G = (V, E)$. As in Section II, let the colors correspond to codeword symbols, and let the path lengths correspond to delays. Then the K-interleaving corresponds to a data-storage solution. For any vertex $v \in V$ and any integer $P \leq K$, let $R_{min}(v, P)$ denote the minimum value of r such that $\sum_{u \in \mathcal{N}(v, r)} w(u) \geq P$; then the delay for vertex v to retrieve P *distinct* codeword symbols is the same as the delay for v to retrieve P *nearest* codeword symbols, which is $R_{min}(v, P)$. So when K equals the number of distinct codeword symbols, the K-interleaving corresponds to an ‘*absolutely optimum*’ data storage solution. The following theorem is clearly true.

Theorem 5 *There always exists an ‘absolutely optimum’ solution for the file storage scheme defined in Section II on tree networks, which simultaneously minimizes the delay for any node to retrieve any number of distinct codeword symbols from the network. An ‘absolutely optimum’ solution can be output by Algorithm 1 by letting $K = n$. (n is the number of distinct codeword symbols.)*

IV. AN EFFICIENT K-INTERLEAVING ALGORITHM

The general K-interleaving algorithm shown in Section III has complexity $O(|V|^2 + \mathbb{W}|V| + \mathbb{W}K)$. In this section we present an algorithm which is less general but has a lower complexity. What's more, the algorithm is very suitable for parallel computation.

Algorithm 2 [Efficient K-interleaving Algorithm for Tree $G = (V, E)$]

1. Denote the root of G by γ . Label the vertices in G as $v_1, v_2, \dots, v_{|V|}$ following this rule: for any $1 \leq i < j \leq |V|$, $d(\gamma, v_i) \leq d(\gamma, v_j)$.
 2. Label the color-slots of G as $s_1, s_2, \dots, s_{\mathbb{W}}$ in the following way: for $1 \leq i \leq |V|$, label the $w(v_i)$ color-slots of vertex v_i as $s_{W_i+1}, s_{W_i+2}, \dots, s_{W_i+w(v_i)}$, where W_i is defined as $W_i = \sum_{1 \leq j \leq i-1} w(v_j)$.
 3. If $\mathbb{W} \leq K$, then assign \mathbb{W} distinct colors to the \mathbb{W} color-slots in the tree such that no two color-slots are assigned the same color, and exit this algorithm. Otherwise go to Step 4.
 4. For $i = 1$ to K , $C_i \leftarrow \{s_i\}$.
 5. For $i = K + 1$ to \mathbb{W} do
 - { Let s_x be the unique color-slot that satisfies the following 2 conditions:
 - (i) $1 \leq x \leq i - 1$;
 - (ii) The set S defined as follows contains exactly $K - 1$ color-slots. S is defined in this way: a color-slot s_j is in S if and only if $1 \leq j \leq i - 1$ and either ' $d(s_i, s_j) < d(s_i, s_x)$ ', or ' $d(s_i, s_j) = d(s_i, s_x)$ and $j < x$ '.
 Let C_t ($1 \leq t \leq K$) be the unique set such that $s_x \in C_t$. $C_t \leftarrow C_t \cup \{s_i\}$.
 - }
 6. Assign K distinct colors to the color-slots in the K sets C_1, C_2, \dots, C_K , such that any two color-slots are assigned the same color if and only if they are in the same set.
-

Example 4: The following example is used to illustrate how Algorithm 2 computes. In the example, the tree $G = (V, E)$ is as shown in Fig. 5. The number beside each edge is the edge's length. The vertex at the top is selected as the root. The 13 vertices in G are labelled as v_1, v_2, \dots, v_{13} which is consistent with the way Algorithm 2 labels vertices — namely, for any $1 \leq i < j \leq |V|$, the distance between the root and v_i is no greater than the distance between the root and v_j . For $1 \leq i \leq 13$, $w(v_i)$ is shown in Fig. 5. Algorithm 2 is used to compute a K-interleaving for the tree using n distinct colors, where $n = 6$ and $K = 5$.

There are $\mathbb{W} = \sum_{v \in V} w(v) = 15$ color-slots in G . Algorithm 2 labels them as s_1, s_2, \dots, s_{15} as shown in the table below.

Vertex v	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}
Color-slots of vertex v	s_1	s_2, s_3	s_4	s_5	s_6, s_7, s_8	s_9, s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}		

Table 1: The labelling of the color-slots

Algorithm 2 then assigns values to 5 sets C_1, C_2, \dots, C_5 as follows: $C_i = \{s_i\}$ for $i = 1, 2, \dots, 5$.

Next, for $i = 6$ to 15, Algorithm 2 finds the color-slot s_x corresponding to i (This is the step 5 in Algorithm 2. Note that s_x changes when i changes.); then s_i is inserted into the set C_t , where C_t ($1 \leq t \leq 5$) is the set that s_x is in (Note that the value of t changes when i changes). This process is illustrated by the table below. (As an example, in the second column of Table 2, $i = 6$, $s_i = s_6$, $s_x = s_5$, $C_t = C_5$. It means when $i = 6$, the corresponding s_x is s_5 . $s_5 \in C_5$, so $C_t = C_5$. As a result, Algorithm 2 inserts s_i — which is s_6 — into C_5 ; then both s_5 and s_6 are in C_5 .)

i	6	7	8	9	10	11	12	13	14	15
s_i	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
s_x	s_5	s_4	s_1	s_4	s_3	s_2	s_7	s_7	s_{10}	s_9
C_t	C_5	C_4	C_1	C_4	C_3	C_2	C_4	C_4	C_3	C_4

Table 2: The execution of the step 5 in Algorithm 2

After the above computing, the values of the 5 sets C_1, C_2, \dots, C_5 become as follows: $C_1 = \{s_1, s_8\}$, $C_2 = \{s_2, s_{11}\}$, $C_3 = \{s_3, s_{10}, s_{14}\}$, $C_4 = \{s_4, s_7, s_9, s_{12}, s_{13}, s_{15}\}$, $C_5 = \{s_5, s_6\}$. Then Algorithm 2 arbitrarily selects 5 distinct colors

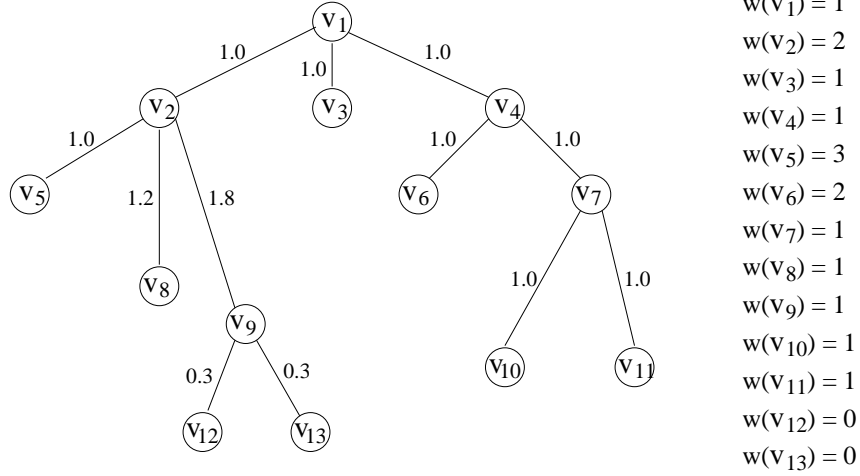


Figure 5: A K -interleaving example using Algorithm 2.

— without loss of generality, say they are color 1, 2, 3, 4 and 5 respectively — and colors the tree with those 5 colors in the following way: for $1 \leq i \leq 5$, assign color i to all the color-slots in C_i . It can be verified that such a coloring is a K -interleaving with $K=5$.

□

It's simple to see that with a K -interleaving output by Algorithm 2, the tree $G = (V, E)$ has exactly K distinct colors.

Analysis shows that Algorithm 2 has time complexity $O(|V|^2 + \mathbb{W})$. And it can be shown that Algorithm 2 is *indeed* more efficient than Algorithm 1, the general K -interleaving algorithm. The complexity analysis is presented in Appendix III.

The major computation in Algorithm 2 is its step 5, where for each color-slot s_i ($K + 1 \leq i \leq \mathbb{W}$), a corresponding color-slot s_x is searched for. Given the tree structure and s_i , the corresponding s_x can be uniquely determined without knowing the coloring on any part of the tree. So Algorithm 2's step 5 can be computed in parallel by up to $\mathbb{W} - K$ computing machines, and then its computational time will be reduced by a factor of $\mathbb{W} - K$.²

Below we start proving the correctness of Algorithm 2.

Lemma 4 *Assume Algorithm 2 is used to produce a K -interleaving for a tree $G = (V, E)$. Algorithm 2 labels the vertices in G as $v_1, v_2, \dots, v_{|V|}$, and labels the color-slots in G as $s_1, s_2, \dots, s_{\mathbb{W}}$. Assume $\mathbb{W} > K$. At the end of Algorithm 2, the K sets C_1, C_2, \dots, C_K is a partition of all the color-slots in G .*

For $K + 1 \leq i \leq \mathbb{W}$, define L_i as an ordered set that sorts the $i - 1$ color-slots s_1, s_2, \dots, s_{i-1} in the following way: for any $1 \leq j \neq k \leq i - 1$, s_j is placed before s_k in the ordered set L_i if ' $d(s_i, s_j) < d(s_i, s_k)$ ' or if ' $d(s_i, s_j) = d(s_i, s_k)$ and $j < k$ '.

Then the following conclusion is true: for any $K + 1 \leq i \leq \mathbb{W}$, the first K color-slots in L_i are evenly distributed in the K sets C_1, C_2, \dots, C_K — that is, for any two color-slots s_j and s_k among the first K color-slots in L_i , if $s_j \in C_{j_0}$ and $s_k \in C_{k_0}$, then $j_0 \neq k_0$.

Proof: For $K + 1 \leq i \leq \mathbb{W}$, define $h(i)$ as a variable that satisfies the following two conditions: (1) $s_{h(i)}$ is one of the first K color-slots in L_i ; (2) for any color-slot s_j that is one of the first K color-slots in L_i , $h(i) \geq j$.

²Processing the color-slots one by one is one straightforward way to implement Algorithm 2's step 5. As shown in Appendix III, a more efficient way to compute is to process all the color-slots of each vertex together. If that method is used, then step 5 can be computed by N_0 computing machines in parallel, and the computational time can be reduced by a factor of approximately N_0 , where N_0 denotes the number of vertices each of which has at least one color-slot.

First we consider the case where i is a fixed integer such that $K + 1 \leq i \leq \mathbb{W}$ and $1 \leq h(i) \leq K$. By the definition of $h(i)$, it's simple to see that $h(i) = K$ and the first K color-slots in L_i is just a permutation of s_1, s_2, \dots, s_K . s_1, s_2, \dots, s_K are evenly distributed in the K sets C_1, C_2, \dots, C_K . Therefore Lemma 4 is true in this case. Now we start proving Lemma 4 by induction.

When $i = K + 1$, $1 \leq h(i) \leq K$. By the above result, the first K color-slots in L_i are evenly distributed in the K sets C_1, C_2, \dots, C_K . We use this as the base case.

Assume i is a fixed integer such that $K + 1 < i \leq \mathbb{W}$; and for $K + 1 \leq j < i$, the first K color-slots in L_j are evenly distributed in the K sets C_1, C_2, \dots, C_K . We need to prove that the first K color-slots in L_i are also evenly distributed in the K sets C_1, C_2, \dots, C_K . Clearly, we only need to consider the case where $K + 1 \leq h(i) \leq i - 1$.

Assume $s_{h(i)}$ is the k_0 -th color-slot in L_i . (Here $1 \leq k_0 \leq K$.) Let B denote the set defined in this way: a color-slot s_k is in B if and only if $1 \leq k \leq i - 1$ and one of the following two conditions is satisfied: (1) $d(s_i, s_k) < d(s_i, s_{h(i)})$, (2) $d(s_i, s_k) = d(s_i, s_{h(i)})$ and $k < h(i)$. Clearly B contains $k_0 - 1$ color-slots, each of which is one of the first $k_0 - 1$ color-slots in L_i ; and each color-slot in B is also in $L_{h(i)}$.

Let u_0 be the unique vertex on all of the following three paths: the path between the root and the vertex that s_i belongs to, the path between the root and the vertex that $s_{h(i)}$ belongs to, and the path between the vertex that s_i belongs to and the vertex that $s_{h(i)}$ belongs to. Let s_y be an arbitrary color-slot in B . Since s_y is one of the first K color-slots in L_i and $y \neq h(i)$, $y < h(i)$. According to the way Algorithm 2 labels color-slots, we know the distance between the root and s_y is no greater than the distance between the root and $s_{h(i)}$ — so if u_0 is on the path between the root and s_y , then $d(u_0, s_y) \leq d(u_0, s_{h(i)})$. If u_0 is not on the path between the root and s_y , we also have $d(u_0, s_y) \leq d(u_0, s_{h(i)})$ because $d(s_i, s_y) \leq d(s_i, s_{h(i)})$, $d(s_i, s_y) = d(s_i, u_0) + d(u_0, s_y)$ and $d(s_i, s_{h(i)}) = d(s_i, u_0) + d(u_0, s_{h(i)})$.

Let s_z be an arbitrary color-slot in $L_{h(i)}$ but not in B . It's not hard to see that u_0 is not on the path between the root and the vertex that s_z belongs to, and $d(u_0, s_z) > d(u_0, s_{h(i)})$. So $d(s_{h(i)}, s_z) = d(s_{h(i)}, u_0) + d(u_0, s_z) > d(s_{h(i)}, u_0) + d(u_0, s_{h(i)}) \geq d(s_{h(i)}, u_0) + d(u_0, s_y) \geq d(s_{h(i)}, s_y)$. Since s_y is an arbitrary color-slot in B , all the color-slots in B are placed before s_z in the ordered set $L_{h(i)}$. Since s_z is an arbitrary color-slot in $L_{h(i)}$ but not in B , the color-slots in B are just the first $k_0 - 1$ color-slots in $L_{h(i)}$ (but the ordering is not specified here). So the first $k_0 - 1$ color-slots in L_i is a permutation of the first $k_0 - 1$ color-slots in $L_{h(i)}$.

Consider the following two cases.

Case 1: In this case, $k_0 = K$. Then $s_{h(i)}$ is the K -th color-slot in L_i . Let C_t ($1 \leq t \leq K$) be the set that the K -th color-slot in $L_{h(i)}$ is in. It's simple to see from Algorithm 2 that $s_{h(i)}$ is also in C_t . By the induction assumption, the first K color-slots in $L_{h(i)}$ are evenly distributed in the K sets C_1, C_2, \dots, C_K . So the first K color-slots in L_i are also evenly distributed in the K sets C_1, C_2, \dots, C_K .

Case 2: In this case, $k_0 < K$. Let s_{q_0} denote the $(k_0 + 1)$ -th color-slot in L_i . Then $d(s_i, s_{q_0}) \geq d(s_i, s_{h(i)})$, and $q_0 < h(i)$. It's not hard to see that u_0 can't be on the path between the root and the vertex that s_{q_0} belongs to, and $d(s_i, s_{q_0}) > d(s_i, s_{h(i)})$. Define D_i to be an ordered set defined as follows: D_i contains $K - k_0$ color-slots, and for $1 \leq j \leq K - k_0$, the j -th color-slot in D is the $(k_0 + j)$ -th color-slot in L_i .

Clearly for any color-slot s_q in D , $q < h(i)$, and $d(s_i, s_q) \geq d(s_i, s_{q_0}) > d(s_i, s_{h(i)})$. Let s_{q_1} denote the K -th color-slot in L_i , which is also the $(K - k_0)$ -th color-slot in D . It's simple to see that a color-slot s_j ($1 \leq j \leq \mathbb{W}$) is in D if and only if the following three conditions are true: (1) $j < h(i)$; (2) $d(s_i, s_j) > d(s_i, s_{h(i)})$; (3) either ' $d(s_i, s_j) < d(s_i, s_{q_1})$ ' or ' $d(s_i, s_j) = d(s_i, s_{q_1})$ and $j \leq q_1$ '. For any color-slot s_q in D , clearly u_0 can't be on the path between the root and the vertex that s_q belongs to. So a color-slot s_j ($1 \leq j \leq \mathbb{W}$) is in D if and only if the following three conditions are true: (1) $j < h(i)$; (2) $d(u_0, s_j) > d(u_0, s_{h(i)})$; (3) either ' $d(u_0, s_j) < d(u_0, s_{q_1})$ ' or ' $d(u_0, s_j) = d(u_0, s_{q_1})$ and $j \leq q_1$ '.

It's been proven that for any color-slot s_z in $L_{h(i)}$ but not in B , u_0 is not on the path between the root and the vertex that s_z belongs to, and $d(u_0, s_z) > d(u_0, s_{h(i)})$. Now it's simple to see that for $1 \leq j \leq K - k_0$, the j -th color-slot in D is the $(k_0 - 1 + j)$ -th color-slot in $L_{h(i)}$.

Let's summarize our results. It's proven that the first $k_0 - 1$ color-slots in L_i is a permutation of the first $k_0 - 1$ color-slots in $L_{h(i)}$. Also clearly for $k_0 + 1 \leq j \leq K$, the j -th color-slot in L_i is the $(j - 1)$ -th color-slot in $L_{h(i)}$, since they are both the same as the $(j - k_0)$ -th color-slot in D . Let $C_{\hat{t}}$ ($1 \leq \hat{t} \leq K$) be the set that the K -th color-slot in $L_{h(i)}$ is in. It's simple to see from Algorithm 2 that $s_{h(i)}$ is also in $C_{\hat{t}}$ — so the k_0 -th color-slot in L_i is in $C_{\hat{t}}$. By the induction assumption, the first K color-slots in $L_{h(i)}$ are evenly distributed in the K sets C_1, C_2, \dots, C_K . So the first K color-slots in L_i are also evenly distributed in the K sets C_1, C_2, \dots, C_K . The analysis of Case 2 ends here.

The proof by induction is complete here. So Lemma 4 is proved.

□

Theorem 6 *Algorithm 2 correctly outputs a K -interleaving for the tree $G = (V, E)$.*

Proof: This theorem can be proved by showing that Algorithm 2 is a special case of Algorithm 1, namely, all the computation in Algorithm 1 is also implemented in Algorithm 2.

Assume Algorithm 2 is used to produce a K -interleaving for a tree $G = (V, E)$. Algorithm 2 labels the vertices in G as $v_1, v_2, \dots, v_{|V|}$, and labels the color-slots in G as $s_1, s_2, \dots, s_{\mathbb{W}}$. At the end of Algorithm 2, the K sets C_1, C_2, \dots, C_K is a partition of all the color-slots in G .

Let γ denote the root of G . By the way Algorithm 2 labels vertices and color-slots, it's simple to see that for $1 \leq i < j \leq \mathbb{W}$, $d(\gamma, s_i) \leq d(\gamma, s_j)$. So the rule on labelling color-slots in Algorithm 1 is also followed in Algorithm 2.

If $\mathbb{W} \leq K$, then Algorithm 2 assigns \mathbb{W} distinct colors to the \mathbb{W} color-slots in G such that no two color-slots are assigned the same color, which is what Algorithm 1 does, too. So if $\mathbb{W} \leq K$, Algorithm 2 will correctly output a K -interleaving for the tree $G = (V, E)$. From now on we only consider the case where $\mathbb{W} > K$.

For $1 \leq i \leq K$, $s_i \in C_i$. For any $1 \leq i \neq j \leq K$, no color-slot in C_i is assigned the same color as any color-slot in C_j . So Algorithm 2 assigns K distinct colors to the following K color-slots: s_1, s_2, \dots, s_K , such that no two color-slots among them are assigned the same color, which is what Algorithm 1's step 1 does, too.

Let y be a fixed integer such that $K + 1 \leq y \leq \mathbb{W}$. Let r_{min} denote the smallest value of r such that the set $\{s_j | 1 \leq j \leq y - 1, d(s_y, s_j) \leq r\}$ contains no less than K color-slots. Define L_y as an ordered set that sorts the $y - 1$ color-slots s_1, s_2, \dots, s_{y-1} in the following way: for any $1 \leq j \neq k \leq y - 1$, s_j is placed before s_k in the ordered set L_y if ' $d(s_y, s_j) < d(s_y, s_k)$ ' or if ' $d(s_y, s_j) = d(s_y, s_k)$ and $j < k$ '. Clearly, the distance between s_y and the K -th color-slot in L_y equals r_{min} , and any color-slot in the set $\{s_j | 1 \leq j \leq y - 1, d(s_y, s_j) < r_{min}\}$ is one of the first $K - 1$ color-slots in L_y .

Assume C_t ($1 \leq t \leq K$) is the set that the K -th color-slot in L_y is in. It's simple to see from Algorithm 2 that $s_y \in C_t$. Let s_q be any color-slot in the set $\{s_j | 1 \leq j \leq y - 1, d(s_y, s_j) < r_{min}\}$. s_q is one of the first $K - 1$ color-slots in L_y . By Lemma 4, $s_q \notin C_t$. So s_y and s_q are assigned different colors by Algorithm 2, which is what Algorithm 1's step 2 does, too.

By now it's proven that all the computation in Algorithm 1 is also implemented in Algorithm 2. Therefore Algorithm 2 is a special case of Algorithm 1. So Algorithm 2 will correctly output a K -interleaving for the tree $G = (V, E)$, just as Algorithm 1 does.

□

V. DISCUSSION

In this paper the diversity coloring problem is studied for data storage in networks. It's shown that a *perfect* diversity coloring always exists on trees. For cyclic graphs, a perfect diversity coloring may also exist if the parameters have certain properties. For example, for the diversity coloring problem where the graph $G = (V, E)$ is an infinite two-dimensional array with all edges of length 1 and with each vertex to be assigned one color, if $\max_{v \in V, r \in \mathbb{R}} M(v, r) = 2m^2 + 2m + 1$ for some fixed $m \in \mathbb{Z}^+$, then a coloring using a close 'sphere packing' shown in [13] will be a perfect diversity coloring, because in the coloring only $2m^2 + 2m + 1$ distinct colors will be actually placed and for every

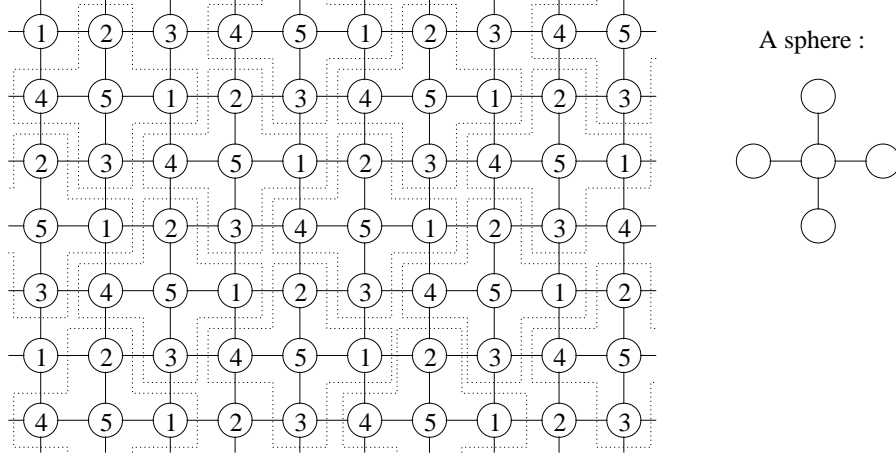


Figure 6: Perfect diversity coloring on 2-dimensional array using close sphere-packing

vertex, its neighborhood of radius m will contain all the $2m^2 + 2m + 1$ distinct colors with each appearing once. Fig. 6 shows such a coloring with $m = 1$. It can be seen that the ‘spheres’, each of which is a neighborhood of radius m , are closely packed.

Many diversity coloring problems for cyclic graphs, however, don’t have ‘perfect’ solutions. As a simple case study we present the following problem on rings. Rings are common and important network topologies, and the problem presented below corresponds to a very natural scenario of data storage.

Definition 7 (A diversity coloring problem on a ring) :

INSTANCE: n colors and a ring $G = (V, E)$. Every edge $e \in E$ is of length 1. For every vertex $v \in V$, $w(v) = 1$, and the function $M(v, r)$ is defined as follows: $M(v, r) = 2\lceil r \rceil + 1$ if $r \leq r_0$; $M(v, r) = 2r_0 + 1$ if $r > r_0$. (Here r_0 is a fixed integer such that $1 \leq 2r_0 + 1 \leq |V|$.)

QUESTION: How to assign $w(v) = 1$ color to each vertex $v \in V$, such that for any $u \in V$ and any real number r , the vertices in $\mathcal{N}(u, r)$ have no less than $M(u, r)$ distinct colors in total? \square

It’s simple to see that the above diversity coloring problem can be reformulated as follows: *how to assign one color to each vertex such that any $2r_0 + 1$ consecutive vertices have $2r_0 + 1$ distinct colors?* Note that r_0 is an integer so $2r_0 + 1$ is odd. By releasing this constraint the following more general problem can be raised: *given a ring $G = (V, E)$ and n distinct colors, how to assign one color to each vertex such that every m consecutive vertices have m distinct colors? (Here m is a fixed integer such that $1 \leq m \leq |V|$.)* The following proposition solves this problem.

Proposition 1 *Given a ring $G = (V, E)$, n distinct colors and a fixed integer m ($1 \leq m \leq |V|$), it’s feasible to assign one color to each vertex such that every m consecutive vertices have m distinct colors if and only if $n \geq \lceil \frac{|V|}{m} \rceil$.*

Proof: First, assume one color is assigned to each vertex such that every m consecutive vertices have m distinct colors. Then for any two vertices of the same color, there must be at least $m - 1$ other vertices between them. So each color is assigned to at most $\lfloor \frac{|V|}{m} \rfloor$ vertices. So the ring G has at least $\lceil \frac{|V|}{m} \rceil$ colors. So $n \geq \lceil \frac{|V|}{m} \rceil$.

Now assume $n \geq \lceil \frac{|V|}{m} \rceil$. Let $a = \lfloor \frac{|V|}{m} \rfloor$, and let $b = |V| \bmod m$. Then $|V| = am + b$ and $0 \leq b < m$. Let $c = \lfloor \frac{b}{a} \rfloor$, let $d = \lceil \frac{b}{a} \rceil$, let $g = b - ac$, and let $h = a - g$. Then $g \geq 0$ and $h > 0$. Also, $(m + c)h + (m + d)g = m(h + g) + ch + dg = ma + \lfloor \frac{b}{a} \rfloor(a - g) + \lceil \frac{b}{a} \rceil g = ma + \lfloor \frac{b}{a} \rfloor a + (\lceil \frac{b}{a} \rceil - \lfloor \frac{b}{a} \rfloor)g = ma + \lfloor \frac{b}{a} \rfloor a + (\lceil \frac{b}{a} \rceil - \lfloor \frac{b}{a} \rfloor)(b - \lfloor \frac{b}{a} \rfloor a) = ma + b = |V|$. Let $m + d$ distinct colors be denoted by integers $1, 2, \dots, m + d$. Color the ring in this way: color $(m + c)h$ consecutive vertices as “ $1, 2, \dots, m + c, 1, 2, \dots, m + c, \dots, 1, 2, \dots, m + c$ ”, and color the following $(m + d)g$ consecutive vertices as “ $1, 2, \dots, m + d, 1, 2, \dots, m + d, \dots, 1, 2, \dots, m + d$ ”. Clearly the ring has $m + d = m + \lceil \frac{b}{a} \rceil = \lceil \frac{am + b}{a} \rceil = \lceil \frac{|V|}{m} \rceil \leq n$ distinct colors; and with such a coloring, every m consecutive vertices have m distinct colors. \square

So for the diversity coloring problem defined in Definition 7, a solution places at least $\lceil \frac{|V|}{\lfloor \frac{|V|}{2r_0+1} \rfloor} \rceil$ distinct colors on the ring; as long as $2r_0 + 1$ can't divide $|V|$, there doesn't exist any perfect diversity coloring as its solution. The proof of Proposition 1 actually presents an algorithm which solves the diversity coloring problem in Definition 7.

It can be shown that for *any* diversity coloring problem for a ring, if $n \geq 2 \cdot \max_{v \in V, r \in \mathbb{R}} M(v, r) - 1$, (here n is the number of colors that can be assigned to the ring, the same notation as in Definition 3), then there exists a solution in which the ring is assigned no more than $2 \cdot \max_{v \in V, r \in \mathbb{R}} M(v, r) - 1$ distinct colors. The reasoning is as follows. Assume a *general* diversity coloring problem as in Definition 3 is given, where the graph $G = (V, E)$ is a ring. Assume $n \geq 2 \cdot \max_{v \in V, r \in \mathbb{R}} M(v, r) - 1$. See every vertex $v \in V$ as having $w(v)$ color-slots. Define \mathbb{W} as $\mathbb{W} = \sum_{v \in V} w(v)$. Label the $|V|$ vertices as $v_1, v_2, \dots, v_{|V|}$, such that v_1 is adjacent to $v_{|V|}$ and v_2 , v_2 is adjacent to v_1 and v_3, \dots , and $v_{|V|}$ is adjacent to $v_{|V|-1}$ and v_1 ; and differentiate the \mathbb{W} color-slots in this way: for $1 \leq i \leq |V|$, say the $w(v_i)$ color-slots of v_i are the $(\sum_{1 \leq j \leq i-1} w(v_j) + 1)$ -th, the $(\sum_{1 \leq j \leq i-1} w(v_j) + 2)$ -th, \dots , the $(\sum_{1 \leq j \leq i-1} w(v_j) + w(v_i))$ -th color-slot in the ring respectively. Let the n distinct colors be denoted by integers $1, 2, \dots, n$. Now color the ring is this way: for the first $|V| - (|V| \bmod \max_{v \in V, r \in \mathbb{R}} M(v, r))$ color-slots in the ring, color them as “ $1, 2, \dots, \max_{v \in V, r \in \mathbb{R}} M(v, r), 1, 2, \dots, \max_{v \in V, r \in \mathbb{R}} M(v, r), \dots, 1, 2, \dots, \max_{v \in V, r \in \mathbb{R}} M(v, r)$ ”; for the last $|V| \bmod \max_{v \in V, r \in \mathbb{R}} M(v, r)$ color-slots in the ring, color them as “ $\max_{v \in V, r \in \mathbb{R}} M(v, r) + 1, \max_{v \in V, r \in \mathbb{R}} M(v, r) + 2, \dots, \max_{v \in V, r \in \mathbb{R}} M(v, r) + (|V| \bmod \max_{v \in V, r \in \mathbb{R}} M(v, r))$ ”. Clearly with such a coloring, the ring has no more than $2 \cdot \max_{v \in V, r \in \mathbb{R}} M(v, r) - 1$ distinct colors; and for any $\max_{v \in V, r \in \mathbb{R}} M(v, r)$ consecutive color-slots, they are assigned $\max_{v \in V, r \in \mathbb{R}} M(v, r)$ distinct colors. For any vertex $\hat{v} \in V$ and for any $\hat{r} \in \mathbb{R}$, the color-slots of the vertices in $\mathcal{N}(\hat{v}, \hat{r})$ must be consecutive. So if there are less than $\max_{v \in V, r \in \mathbb{R}} M(v, r)$ color-slots in $\mathcal{N}(\hat{v}, \hat{r})$, then all the $\sum_{u \in \mathcal{N}(\hat{v}, \hat{r})} w(u)$ color-slots in $\mathcal{N}(\hat{v}, \hat{r})$ are assigned distinct colors; if there are no less than $\max_{v \in V, r \in \mathbb{R}} M(v, r)$ color-slots in $\mathcal{N}(\hat{v}, \hat{r})$, then those color-slots are assigned no less than $\max_{v \in V, r \in \mathbb{R}} M(v, r)$ distinct colors. By the definition of the diversity coloring problem, $\sum_{u \in \mathcal{N}(\hat{v}, \hat{r})} w(u) \geq M(\hat{v}, \hat{r})$; and certainly $\max_{v \in V, r \in \mathbb{R}} M(v, r) \geq M(\hat{v}, \hat{r})$. So the vertices in $\mathcal{N}(\hat{v}, \hat{r})$ have no less than $M(\hat{v}, \hat{r})$ distinct colors in any case. Therefore the coloring is a solution to the *general* diversity coloring problem for a ring.

So a diversity coloring on a ring *never* has to use twice as many as $\max_{v \in V, r \in \mathbb{R}} M(v, r)$ distinct colors. That is not true, however, for general graphs. There exist diversity coloring problems for general graphs where the minimum number of distinct colors a solution needs to use can be arbitrarily greater than $\max_{v \in V, r \in \mathbb{R}} M(v, r)$. Examples of such problems can be easily found.

This section mainly focuses on the minimum number of distinct colors a diversity coloring needs to use. Studying that number is important in both theoretical and application aspects. That number not only is critical for the existence of solutions to diversity coloring but also influences the complexity of finding the solutions. And when the diversity coloring problem is related to the file storage scheme proposed in this paper, that number sets a lower bound on the codeword length of the error-correcting code used. A large codeword length usually implies a small selection of the candidate codes and high decoding complexity. Therefore knowing the lower bound on the codeword length is certainly desirable.

VI. CONCLUDING REMARKS

In this paper a new file storage scheme is proposed. The scheme combines coding with storage for better performance, and aims at satisfying the various QoS requirements of different nodes in both fault-free and faulty environments. The scheme is formulated as a ‘diversity coloring problem’, which is solved for tree networks using a ‘K-interleaving’ technique. The K-interleaving technique can produce both perfect diversity coloring and absolutely optimum file placement solutions. It is also a new member in the family of interleaving techniques for codeword layout. Various other aspects of diversity coloring and K-interleaving are also studied.

APPENDIX I

In this appendix we analyze the complexity of Algorithm 1.

To implement Algorithm 1, it's first necessary to label all the color-slots as $s_1, s_2, \dots, s_{\mathbb{W}}$ based on their distance to the root. This has time complexity $O(|V| \log |V| + \mathbb{W})$.

Assigning colors to the first K color-slots, which are s_1, s_2, \dots, s_K , has time complexity $O(K)$. For any of the remaining $\mathbb{W} - K$ color-slots, say it's s_i , in order to assign a color to it, it's necessary to find out the colors of all the color-slots in the set $\{s_1, s_2, \dots, s_{i-1}\}$ that are at distance less than r_{min} from s_i . (Here r_{min} , as in Algorithm 1, is defined as the smallest value of r such that in the set $\{s_1, s_2, \dots, s_{i-1}\}$, there are no less than K color-slots at distance no greater than r from s_i . Note that the value of r_{min} changes if the value of i changes.) To do that, the algorithm needs to inspect the color-slots of the tree in the order of their distance to s_i — which requires the ordering of the vertices in the tree based on their distance to s_i — until K color-slots are inspected and the value of r_{min} is determined. Making a list for every vertex in the tree which orders vertices according to their distance to that vertex has a total complexity of $O(|V|^2)$. Then to color s_i , up to $O(|V|)$ vertices and $O(K)$ color-slots need to be inspected. Therefore coloring the remaining $\mathbb{W} - K$ color-slots has a total complexity of $O(|V|^2 + \mathbb{W} \cdot |V| + \mathbb{W} \cdot K)$.

By combining the above results, the time complexity of Algorithm 1 is found to be $O(|V|^2 + \mathbb{W}|V| + \mathbb{W}K)$.

APPENDIX II

In this appendix we present the proof of Theorem 4.

Proof: By Theorem 3, the set of possible outputs of Algorithm 1 is a subset of the set of K-interleavings. So the only thing left to prove is that every K-interleaving is a possible output of Algorithm 1, even if the root of the tree is fixed.

Assume the root of the tree is fixed. Assume a fixed K-interleaving for the tree $G = (V, E)$ is given. There are totally $\mathbb{W} = \sum_{v \in V} w(v)$ color-slots in the tree. If $\sum_{v \in V} w(v) \leq K$, it's simple to see that the K-interleaving must have assigned \mathbb{W} distinct colors to the \mathbb{W} color-slots, which is clearly a possible output of Algorithm 1. So from now on we only consider the case where $\mathbb{W} > K$. We'll prove by induction that for $1 \leq i \leq \mathbb{W}$, there is a set of i color-slots that Algorithm 1 can label as s_1, s_2, \dots, s_i and assign the same colors to as the given K-interleaving does.

Let γ denote the fixed root of the tree. For $1 \leq j \leq \mathbb{W}$, let $R_{min}(\gamma, j)$ denote the minimum value of r such that $\sum_{v \in \mathcal{N}(\gamma, r)} w(v) \geq j$. Let A_0 denote the set of color-slots at distance less than $R_{min}(\gamma, K)$ from γ . By Lemma 1, all the color-slots in A_0 are assigned distinct colors in the given K-interleaving, and there exists a set B_0 of $K - |A_0|$ color-slots at distance $R_{min}(\gamma, K)$ from γ such that all the K color-slots in $A_0 \cup B_0$ are assigned distinct colors in the given K-interleaving. Clearly it's feasible for Algorithm 1 to label the color-slots in $A_0 \cup B_0$ as s_1, s_2, \dots, s_K , and assign the same colors to them as the given K-interleaving does. Therefore the induction is true for $1 \leq i \leq K$. We'll use this as the base case.

Assume the following induction assumption is true: for some fixed i such that $K \leq i < \mathbb{W}$, there is a set C of i color-slots which Algorithm 1 can label as s_1, s_2, \dots, s_i and assign the same colors to as the given K-interleaving does. We will prove that this induction assumption is also true if i is replaced by $i + 1$.

Since Algorithm 1 always assigns colors to color-slots in the order of their increasing distance to the root γ , it's simple to see that no color-slot in C is at distance greater than $R_{min}(\gamma, i)$ from γ ; and for any color-slot not in C , it is at distance no less than $R_{min}(\gamma, i + 1)$ from γ .

Let D denote the set of all the \mathbb{W} color-slots in the tree. Let H denote the set of color-slots in $D - C$ at distance $R_{min}(\gamma, i + 1)$ from γ . (Clearly H must be non-empty.) Denote the color-slots in H as $c_1, c_2, \dots, c_{|H|}$. Randomly pick a color-slot c_{j_1} from H . (Here $1 \leq j_1 \leq |H|$.) Let $r_{min}(j_1)$ denote the smallest value of r such that there are no less than K color-slots in C at distance no greater than r from c_{j_1} . We consider the following two cases.

Case 1: In this case, no color-slot in C at distance less than $r_{min}(j_1)$ from c_{j_1} is assigned the same color as c_{j_1} in the given K-interleaving. Then it's simple to see that it is feasible for Algorithm 1 to label c_{j_1} as the color-slot s_{i+1} and assign s_{i+1} the same color as the given K-interleaving does, after labelling the color-slots in C as s_1, s_2, \dots, s_i and assigning the same colors to them as the given K-interleaving does.

Case 2: In this case, at least one color-slot in C at distance less than $r_{min}(j_1)$ from c_{j_1} is assigned the same color as c_{j_1} in the given K-interleaving. Let t_{k_1} denote such a color-slot: $t_{k_1} \in C$; t_{k_1} and c_{j_1} are assigned the same color in the given K-interleaving; for any color-slot $z \in C$ that is assigned the same as c_{j_1} in the given K-interleaving, $d(t_{k_1}, c_{j_1}) \leq d(z, c_{j_1})$. Clearly, $d(t_{k_1}, c_{j_1}) < r_{min}(j_1)$. Say c_{j_1} is a color-slot of vertex u_1 , and t_{k_1} is a color-slot of vertex v_1 . $d(u_1, \gamma) = d(c_{j_1}, \gamma) = R_{min}(\gamma, i+1) \geq R_{min}(\gamma, i) \geq d(t_{k_1}, \gamma) = d(v_1, \gamma)$, so $\mathcal{C}(u_1, v_1)$ is on the path between u_1 and γ .

Let J denote the set of color-slots of the vertices in $\mathcal{S}(u_1, v_1)$. For any color-slot $z \in J$, $z \in C \cup H$ because otherwise $d(z, \gamma) > R_{min}(\gamma, i+1)$ and therefore $d(z, \mathcal{C}(u_1, v_1)) > d(u_1, \mathcal{C}(u_1, v_1)) = \frac{d(u_1, v_1)}{2}$, which implies $z \notin J$. So $J \subseteq C \cup H$. Therefore, $J = (J \cap C) \cup (J \cap H)$, and $(J \cap C) \cap (J \cap H) = \emptyset$.

By Lemma 3, the color-slots in J are assigned at least K distinct colors in the given K-interleaving. For any color-slot $z \in J \cap C$, $d(u_1, z) \leq d(u_1, \mathcal{C}(u_1, v_1)) + d(\mathcal{C}(u_1, v_1), z) \leq \frac{d(u_1, v_1)}{2} + \frac{d(u_1, v_1)}{2} = d(u_1, v_1) = d(c_{j_1}, t_{k_1}) < r_{min}(j_1)$. There are fewer than K color-slots in C at distance less than $r_{min}(j_1)$ from c_{j_1} , so there are fewer than K color-slots in $J \cap C$. Therefore there is a color-slot in $J \cap H$ (which we denote by c_{j_2}) such that in the given K-interleaving, neither c_{j_1} nor any color-slot in $J \cap C$ is assigned the same color as c_{j_2} .

We can replace c_{j_1} with c_{j_2} and consider (go through) the above two cases again. (All the parameters need to change accordingly when we replace c_{j_1} with c_{j_2} .) If case 1 becomes true when we replace c_{j_1} with c_{j_2} , then it's feasible for Algorithm 1 to label c_{j_2} as the color-slot s_{i+1} and assign s_{i+1} the same color as the given K-interleaving does. Otherwise case 2 is true when we replace c_{j_1} with c_{j_2} , and then we can find such parameters ' t_{k_2}, u_2, v_2 and c_{j_3} ': ' t_{k_2}, u_2, v_2 and c_{j_3} ' are to c_{j_2} just as ' t_{k_1}, u_1, v_1 and c_{j_1} ' are to c_{j_1} . Then we can replace c_{j_1} with c_{j_3} and consider (go through) the above two cases again, and so on Notice that $\mathcal{C}(u_1, v_1)$ lies on the path between γ and u_2 , because otherwise $d(\mathcal{C}(u_1, v_1), u_2) > d(\mathcal{C}(u_1, v_1), u_1) = \frac{d(u_1, v_1)}{2}$ and that would imply that $c_{j_2} \notin J$, which is false. Then it's simple to see that a color-slot in C is within distance $d(u_1, v_1)$ from c_{j_2} if and only if this color-slot is in $J \cap C$. No color-slot in $J \cap C$ is assigned the same color as c_{j_2} in the given K-interleaving, but t_{k_2} and c_{j_2} are assigned the same color, and $t_{k_2} \in C$. So $d(v_2, u_2) = d(t_{k_2}, c_{j_2}) > d(u_1, v_1)$. $\mathcal{C}(u_2, v_2)$ lies on the path between γ and u_2 just as $\mathcal{C}(u_1, v_1)$ lies on the path between γ and u_1 . So both $\mathcal{C}(u_2, v_2)$ and $\mathcal{C}(u_1, v_1)$ lie on the path between γ and u_2 . Since $d(\mathcal{C}(u_2, v_2), u_2) = \frac{d(v_2, u_2)}{2} > \frac{d(u_1, v_1)}{2} = d(\mathcal{C}(u_1, v_1), u_2)$, the point $\mathcal{C}(u_2, v_2)$ must be lying on the path between γ and $\mathcal{C}(u_1, v_1)$, and $\mathcal{C}(u_2, v_2)$ and $\mathcal{C}(u_1, v_1)$ can't be the same point. Similarly, let ' u_3 and v_3 ' be the parameters which are to c_{j_3} just as ' u_2 and v_2 ' are to c_{j_2} and just as ' u_1 and v_1 ' are to c_{j_1} ; then the point $\mathcal{C}(u_3, v_3)$ must be lying on the path between γ and $\mathcal{C}(u_2, v_2)$, and $\mathcal{C}(u_3, v_3)$, $\mathcal{C}(u_2, v_2)$ and $\mathcal{C}(u_1, v_1)$ must all be distinct points. And so on So $c_{j_1}, c_{j_2}, c_{j_3} \dots$ are all distinct color-slots. There are a limited number of color-slots in H . So eventually we'll find some $c_{j_x} \in H$ ($1 \leq x \leq |H|$) such that it's feasible for Algorithm 1 to label c_{j_x} as the color-slot s_{i+1} and assign s_{i+1} the same color as the given K-interleaving does, after labelling the color-slots in C as s_1, s_2, \dots, s_i and assigning the same colors to them as the given K-interleaving does.

So by now it been shown that in all cases, some color-slot $c_{j_x} \in H$ can be found such that it's feasible for Algorithm 1 to label the color-slots in $C \cup \{c_{j_x}\}$ as s_1, s_2, \dots, s_{i+1} , and assign the same colors to them as the given K-interleaving does. Since $C \cup \{c_{j_x}\}$ contains $i+1$ color-slots, the induction assumption is true when i is replaced by $i+1$.

The proof by induction is complete here. Now it's clear that Algorithm 1 can assign the same colors to all the W color-slots as the arbitrarily given K-interleaving does. So any K-interleaving is a possible output of Algorithm 1, even if the root of the tree is fixed. Therefore this theorem is proved.

□

APPENDIX III

In this appendix we analyze the time complexity of Algorithm 2, and show that Algorithm 2 is more efficient than Algorithm 1.

Complexity Analysis: Algorithm 2 first labels the vertices as $v_1, v_2, \dots, v_{|V|}$ according to the vertices' distance to the root, and labels the color-slots as $s_1, s_2, \dots, s_{\mathbb{W}}$ based on the labelling of the vertices. This has complexity $O(|V| \log |V| + \mathbb{W})$.

Initializing the values of the K sets C_1, C_2, \dots, C_K — namely, to let $C_i = \{s_i\}$ for $1 \leq i \leq K$ — has complexity $O(K)$. Then for $i = K + 1$ to \mathbb{W} , the step 5 in Algorithm 2 searches for the color-slot s_x and the set C_t corresponding to i , and inserts s_i into C_t . (For the meaning of s_x and C_t , see the step 5 in the Algorithm 2.) To do that, we can process the vertices one by one, with the method described below. For simplicity, we only describe the process of processing one vertex v_i . (Also for simplicity, we assume all v_i 's color-slots are in $\{s_{K+1}, s_{K+2}, \dots, s_{\mathbb{W}}\}$, and $1 \leq w(v_i) \leq K$. It's simple to see that the following method only needs to be modified slightly when $w(v_i) > K$ or some of v_i 's color-slots are not in $\{s_{K+1}, s_{K+2}, \dots, s_{\mathbb{W}}\}$. And when $w(v_i) = 0$, there is no need to process v_i .)

The process of processing v_i is as follows. Make a list which sorts the vertices v_1, v_2, \dots, v_{i-1} according to the following two rules: (1) for any $1 \leq j_1 \neq j_2 \leq i - 1$, if $d(v_i, v_{j_1}) < d(v_i, v_{j_2})$, then v_{j_1} is placed before v_{j_2} in the list; (2) for any $1 \leq j_1 \neq j_2 \leq i - 1$, if $d(v_i, v_{j_1}) = d(v_i, v_{j_2})$ and $j_1 < j_2$, then v_{j_1} is placed before v_{j_2} in the list. (Making $|V|$ such lists for all the $|V|$ vertices in the tree has a total complexity of $O(|V|^2)$.) Say in the list, the vertices are ordered as $(v_{k_1}, v_{k_2}, \dots, v_{k_{i-1}})$. (Here $(k_1, k_2, \dots, k_{i-1})$ is a permutation of $(1, 2, \dots, i - 1)$.) Find four variables a, b, c and d that satisfy the following conditions: $\sum_{1 \leq j \leq a-1} w(v_{k_j}) + b = K - w(v_i) + 1$, $\sum_{1 \leq j \leq c-1} w(v_{k_j}) + d = K$, $1 \leq a \leq i - 1$, $1 \leq b \leq w(v_{k_a})$, $1 \leq c \leq i - 1$, and $1 \leq d \leq w(v_{k_c})$.

For $p = 1, 2, \dots, |V|$, define W_p as $W_p = \sum_{1 \leq q \leq p-1} w(v_q)$. (By convention, $W_1 = 0$.) For $p = 1, 2, \dots, |V|$ and $q = 1, 2, \dots, w(v_p)$, define $\hat{s}_{p,q}$ as the color-slot s_{W_p+q} . (So for any $1 \leq p \leq |V|$, the $w(v_p)$ color-slots of vertex v_p are $\hat{s}_{p,1}, \hat{s}_{p,2}, \dots, \hat{s}_{p,w(v_p)}$.)

Define \hat{S} as an *ordered* set of color-slots that satisfies the following two conditions: (1) the only color-slots in \hat{S} are these $w(v_{k_a}) - b + 1$ color-slots of vertex v_{k_a} — $\hat{s}_{k_a,b}, \hat{s}_{k_a,b+1}, \dots, \hat{s}_{k_a,w(v_{k_a})}$, and all the color-slots of vertices $v_{k_{a+1}}, v_{k_{a+2}}, \dots, v_{k_{c-1}}$, and these d color-slots of vertex v_{k_c} — $\hat{s}_{k_c,1}, \hat{s}_{k_c,2}, \dots, \hat{s}_{k_c,d}$; (2) for any two color-slots $\hat{s}_{k_x,y}$ and $\hat{s}_{k_z,u}$ in \hat{S} , $\hat{s}_{k_x,y}$ is placed before $\hat{s}_{k_z,u}$ in \hat{S} if and only if ' $x > z$ ' or ' $x = z$ and $y > u$ '. (It can be verified that \hat{S} contains $w(v_i)$ color-slots.)

Now for $j = 1$ to $w(v_i)$, insert the color-slot $\hat{s}_{i,j}$ into the set C_t , where C_t ($1 \leq t \leq K$) is the set that the j -th color-slot of \hat{S} is in. (Note that t changes when j changes.) And the process of processing vertex v_i ends here.

If the complexity of making the list which sorts the vertices v_1, v_2, \dots, v_{i-1} into $(v_{k_1}, v_{k_2}, \dots, v_{k_{i-1}})$ is not counted, then the process of processing vertex v_i simply has complexity $O(|V| + w(v_i))$. Therefore the complexity of processing all the $\mathbb{W} - K$ vertices $v_{K+1}, v_{K+2}, \dots, v_{\mathbb{W}}$, without excluding the complexity of any computation, is $O(|V|^2 + \mathbb{W})$.

By combining the above results, the time complexity of Algorithm 2 is found to be $O(|V|^2 + \mathbb{W})$.

Comparison between Algorithm 1 and Algorithm 2: There are two significant differences between the implementation of Algorithm 1 and Algorithm 2.

Difference 1: In Algorithm 1, the color-slots are colored one by one. Except for the first K color-slots, to color a color-slot, some vertices that K other color-slots belong to need to be inspected. However in Algorithm 2, as described in the previous complexity analysis, the vertices — instead of the color-slots — are processed one by one; every time a vertex v_i is processed (here $w(i) > 0$), the vertices that K other color-slots belong to are inspected for all the $w(v_i)$ color-slots of v_i — instead of for just one color-slot — in a very similar manner. So when the processing of all the color-slots of a vertex v_i is considered, for the above computation, compared to Algorithm 1, Algorithm 2 reduces the complexity by a factor of $w(v_i)$.

Difference 2: In Algorithm 1, except for the first K color-slots, to color a color-slot, approximately K colors needs to be checked to determine the set of colors that can possibly be assigned to that color-slot. However, in Algorithm 2,

the counterpart operation is simply to insert the color-slot into some set C_t . So for the above computation, compared to Algorithm 1, Algorithm 2 reduces the complexity by a factor which is approximately K .

The operations in Algorithm 1 and the operations in Algorithm 2, except for the ones specified in ‘Difference 1’ and ‘Difference 2’, are very similar and have the same complexity. Therefore, Algorithm 2 is more efficient than Algorithm 1.

References

- [1] K. A. S. Abdel-Ghaffar, R. J. McEliece, and H. C. A. van Tilborg. Two-dimensional burst identification codes and their use in burst correction. *IEEE Transactions on Information Theory*, 34:494–504, 1988.
- [2] C. Almeida and R. Palazzo. Two-dimensional interleaving using the set partition technique. In *Proc. IEEE International Symposium on Information Theory*, page 505, Trondheim, Norway, 1994.
- [3] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pages 164–173, 1993.
- [4] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computers*, 44(2):192–202, 1995.
- [5] M. Blaum, J. Bruck, and A. Vardy. Interleaving schemes for multidimensional cluster errors. *IEEE Transactions on Information Theory*, 44(2):730–743, 1998.
- [6] V. Bohossian, C. C. Fan, P. S. LeMahieu, M. D. Riedel, L. Xu, and J. Bruck. Computing in the RAIN: a reliable array of independent nodes. *IEEE Transactions on Parallel and Distributed Systems*, 12(2):99–114, 2001.
- [7] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. Constant-factor approximation algorithm for the k-median problem. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 1–10, 1999.
- [8] P. Delsarte. Bilinear forms over a finite field, with applications to coding theory. *J. Combin. Theory*, 25-A:226–241, 1978.
- [9] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *Computing Surveys*, 14(2):287–313, 1982.
- [10] T. Etzion and A. Vardy. Two-dimensional interleaving schemes with repetitions: constructions and bounds. *IEEE Transactions on Information Theory*, 48(2):428–457, 2002.
- [11] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.
- [12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York and San Francisco, 1979.
- [13] S. W. Golomb and L. R. Welch. Perfect codes in the Lee metric and the packing of polyominoes. *SIAM J. Appl. Math.*, 18(2):302–317, 1970.
- [14] K. Kalpakis, K. Dasgupta, and O. Wolfson. Optimal placement of replicas in trees with read, write, and storage costs. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):628–637, 2001.
- [15] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC’97)*, pages 654–663, El Paso, Texas, May 1997.

- [16] D. Karger, A. Sherman, A. Berkhemier, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins, and Y. Yerushalmi. Web caching with consistent hashing. In *Proc. 8th Int. World Wide Web Conf.*, May 1999.
- [17] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. II: the p-medians. *SIAM J. Appl. Math.*, 37(3):539–560, 1979.
- [18] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. I: the p-centers. *SIAM J. Appl. Math.*, 37(3):513–538, 1979.
- [19] S. Khuller and Y. J. Sussmann. The capacitated k-center problem. *SIAM J. Discrete Math.*, 13(3):403–418, 2000.
- [20] M. R. Korupolu and M. Dahlin. Coordinated placement and replacement for large-scale distributed caches. *IEEE Transactions on Knowledge and Data Engineering*, 14(6):1317–1329, 2002.
- [21] J. Kubiawicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weather- spoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *Proc. 9th International Conference Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, pages 190–201, Cambridge, MA, U.S.A., 2000.
- [22] A. Kumar, R. Rastogi, A. Silberschatz, and B. Yener. Algorithms for provisioning virtual private networks in the hose model. *IEEE/ACM Transactions on Networking*, 10(4):565–578, 2002.
- [23] J. F. Kurose and R. Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers*, 38(5):705–717, 1989.
- [24] Q. M. Malluhi and W. E. Johnston. Coding for high availability of a distributed-parallel storage system. *IEEE Transactions on Parallel and Distributed Systems*, 9(12):1237–1252, 1998.
- [25] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.
- [26] M. Naor and R. M. Roth. Optimal file sharing in distributed networks. *SIAM J. Comput.*, 24(1):158–183, 1995.
- [27] National Lab of Applied Network Research (NLNR). [Online]. Available: <http://ircache.nlnr.net/>.
- [28] C. Park and J. J. Metzner. Efficient location of discrepancies in multiple replicated large files. *IEEE Transactions on Parallel and Distributed Systems*, 13(6):597–610, 2002.
- [29] D. A. Patterson, G. A. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks. In *Proc. SIGMOD Int. Conf. Data Management*, pages 109–116, 1988.
- [30] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '97)*, pages 311–320, New York, NY, USA, 1997.
- [31] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.
- [32] P. Rodriguez and E. W. Biersack. Dynamic parallel access to replicated content in the Internet. *IEEE/ACM Transactions on Networking*, 10(4):455–465, 2002.
- [33] P. Rodriguez, C. Spanner, and E. W. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *IEEE/ACM Transactions on Networking*, 9(4):404–418, 2001.

- [34] R. M. Roth. Maximum-rank array codes and their application to crisscross error correction. *IEEE Transactions on Information Theory*, 37:328–336, 1991.
- [35] C. Shahabi and F. Banaei-Kashani. Decentralized resource management for a distributed continuous media server. *IEEE Transactions on Parallel and Distributed Systems*, 13(11):1183–1200, 2002.
- [36] D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC'97)*, pages 265–274, El Paso, Texas, May 1997.
- [37] N. G. Smith. The UK national web cache — the state of the art. In *Proc. Computer Networks and ISDN Systems*, volume 28, pages 1407–1414, 1996.
- [38] J. Wang. A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Comm. Rev.*, 29(5):36–46, 1999.