Optimized Network Data Storage and Topology Control

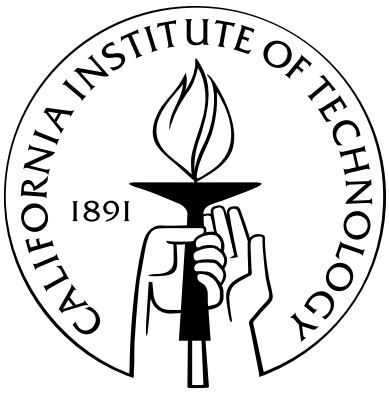
Thesis by

Anxiao Jiang

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy



California Institute of Technology

Pasadena, California

2004

(Submitted May 27, 2004)

Acknowledgements

I am grateful to my advisor, Prof. Jehoshua (Shuki) Bruck, for his enormous help over the years. He has offered valuable advice on all aspects of research, and allowed me complete freedom in exploring new fields. I have received endless support from Shuki in everything.

I am grateful to Prof. Robert J. McEliece, Prof. K. Mani Chandy and Prof. Steven Low. I have learned a lot from them, both inside and outside classes. Prof. McEliece and Prof. Low have been indulgent in letting me attend their group meetings, where I listened, talked, and made friends with their students.

I am grateful to Dr. Mario Blaum, from whom I have received lots of help. One of my favorite papers was sparked by his influential work.

I am grateful to Prof. Richard M. Wilson and Prof. Yaser S. Abu-Mostafa, for serving on my candidacy committee despite their very busy schedules, and for providing valuable feedback.

I am grateful to Prof. Chris Umans and Prof. Leonard Schulman. Attending their theoretical computer science reading groups has been a most stimulating and enjoyable experience — a highlight in my final year at Caltech.

I am grateful to Matthew Cook, for the many very helpful discussions on research we have had.

I am grateful to my wonderful friends at Caltech: Vince Koosh, Marc Riedel, and many others. I am also grateful to Caltech itself, where I have spent the happiest years of my life.

Finally, I am grateful to my dear parents, for their endless love and for being strong; and to my dear sister, Dan, for accompanying me while I was growing up.

Contents

List of Figures

Abstract

This thesis addresses two key challenges for network data-storage systems: optimizing data placement for highly efficient and robust data access, and constructing network topologies that facilitate data transmission scalable to both network sizes and network dynamics. It focuses on two new topics — data placement using erasure-correcting codes, and topology control for nodes in normed spaces. The first topic generalizes traditional file-assignment problems, and has the distinct feature of interleavingly placing data in networks. The second topic emphasizes the construction of network topologies that achieve excellent global performance in comprehensive measurements, through purely local decisions on connectivity. The results of the thesis deepen the current understanding on these important and intriguing topics, and follow a mathematically rigorous approach.

Chapter 1

Introduction

1.1 Facing the Challenges of Network Data Storage

Network data storage for large-scale information sharing is becoming a reality. The method of data replication for the access of distributed users has progressed from mirroring file servers, to commercial content-distribution networks (CDNs) (e.g., Akamai), then to peer-to-peer type of networks. The trend has been a shift from the traditional file management model, where a user's or company's files are stored and maintained on its own computers, to jointly storing many individual owners' files in the same distributed network; from the traditional memory usage model, where each piece of memory can only be used by its owner, to the formation of a network of memories owned by many individuals that can be publicly accessed and utilized; from the traditional server-client service model, to the model where every computer is not only a server providing data, but also a client requesting data stored elsewhere, and even a router in the data-storage network (which may be an overlay network, a self-deployed wireless network, etc.). The driving force behind these transitions is the increasingly wide sharing of information, and the very large number of freely available memories of ordinary users. As a result, we are getting a network that removes unnecessary redundancy in data replicas, that is more efficient for data access, and more robust for data loss and network failures.

There are two fundamental challenges facing such emerging data storage networks, for both of which the research and practice are at their young stage.

The first challenge is the *optimized placement of data*. In nearly all systems so far, files are replicated in their entirety in the network — a computer either stores the entire copy of the file or none of it. From an information-theory point of view, the files are stored using a simple repetition code. A more general approach will be to store files using erasure-correcting codes — encode a file with an erasure-correcting code, and store copies of the codeword components distributively in the network. (Here we emphasize the use of erasure-correcting codes, instead of error-correcting codes, because erasures happen more often at the file level in computer networks.) Clearly, by using a strong erasure-correcting code, significantly better performance can be achieved — smaller data-access delay because an erasure-correcting code is a more general way to represent a file and leads to more freedom in determining the placement of data; more balanced memory usage and data flow because a file is now stored in smaller pieces on more computers, and a client can access several computers for different components of the codeword; higher data availability when the network faces loss of data on some computers, topological changes, or regional failures, because of the strong erasure-correcting ability of the code. All the benefits come at no cost of increased memory requirement — in fact, less data might need to be stored. We emphasize that the last point of performance improvement — higher data availability — is not just useful in the case of emergence. In the peer-to-peer type of data-storage networks, each user can freely turn on and off its computer, and a computer may move (physically or logically) to other areas of the network. So loss of some stored data and changes in the network topology happen all the time; and high availability of data needs to be ensured all the time.

The performance improvement of storing files using erasure-correcting codes does come at a cost — the complexity of decoding the codeword. Every time a user retrieves some components of the codeword, one needs to perform decoding to recover the original file. One obvious way to reduce that cost is to develop better codes and decoding algorithms. Another important method to solve this problem is to

find the placement of codeword components that minimizes the requirement on the codeword length while satisfying other performance requirements, therefore allowing the maximum freedom in selecting the erasure-correcting code of the minimum decoding complexity. In this thesis we study the placement of files in networks using erasure-correcting codes; and our results manifest that sometimes the requirement on codeword length can be reduced to be minimal.

The second challenge is the *optimized construction of network topology*. A peer-to-peer type of data-storage network may be an overlay network, or a wireless network that is possibly mobile. The network topology is how the nodes in the network are connected to each other. A good topology not only needs to prevent network partitioning and have short distance between node pairs, but also needs to have localized construction algorithms — meaning that every node determines its connectivity to other nodes using only very local information — and to enable routing methods scalable to both network size and network dynamics. That is because the emerging peer-to-peer type networks have two common features — they tend to be very large, and their nodes can dynamically join, leave or move around. For such networks, geographic routing — the method of routing messages greedily by trying to reduce the geographic distance to the destination with every hop — appears to be *the* scalable routing method.

While geographic routing is natural for wireless networks, how can it be applied to overlay networks? The idea is to embed the overlay network into a normed space. More details on this idea will be introduced in the next section. Current peer-to-peer type of networks (including both overlay networks and wireless networks) either do not support geographic routing (e.g., the Gnutella network), or support geographic routing but have large stretches in the lengths of routing paths. In this thesis, we will show how to construct network topologies to support *efficient* geographic routing.

This thesis studies two new topics — data placement using erasure-correcting codes, and localized topology control that ensures high performance in geographic routing and other comprehensive aspects. It opens new research directions for the two fundamental tasks of a data-storage network — the storage and transmission of

information.

1.2 Background

1.2.1 Prior Work

1.2.1.1 Data Placement

The placement of replicas of data in a network is often known as the *File Assignment Problem* (FAP). There has been extensive work on this topic dating back to the 1970s. In the well known paper [19] published in 1982, Dowdy and Foster summarized many kinds of file assignment problems based on different system models and optimization objectives, and reviewed the numerous techniques used to solve them. Research on FAP continues to be popular [9], [41], [47], while the focus has been nearly exclusively on file replication without using erasure-correcting codes. A lot of FAPs essentially belong to the family of general problems called *Facility Location*, while other FAPs are quite different due to their specialized settings.

Data placement using erasure-correcting codes is a new topic. A major result was derived by Naor and Roth in [63], but other than that research has been very limited. In [63], Naor and Roth studied how to place components of a codeword on a general graph, such that every vertex can reconstruct the codeword by accessing components stored on itself and its direct neighbors. They presented a solution that is asymptotically optimal in minimizing the total number of stored bits, when the original information in the codeword has a length much larger than the logarithm of the graph's degree.

Data placement using erasure-correcting codes generalizes the traditional file assignment problems. Besides the common element that it shares with traditional FAPs — deciding how much data to store on each node of the network — it also has a distinct factor that makes it very different from traditional FAPs. That is, when data are placed in a network using erasure-correcting codes, the codeword components need to be interleaved on the network so that for every node, it can find enough different

codeword components from nearby.

Data storage using erasure-correcting codes has been applied to disks, such as CDs and the RAID disk arrays [67], as well as to server clusters [55]. In recent years erasure-correcting codes have also been applied to data streaming schemes, such as Digital Fountain [17]. The usage of erasure-correcting codes for network data storage is a topic of this thesis.

1.2.1.2 Topology Control

Peer-to-peer networks residing in the Internet often employ a logic structure called *overlay network*, which represents the logic connections among its nodes and is the basis for routing. Some of them do not enable geographic routing, such as Gnutella [25] which uses limited flooding; but many others essentially use geographic routing, including Chord [81], CAN [70], Viceroy [53], Pastry [74], Butterflies [18], etc. As an example, the overlay network of Chord is a ring along which messages are routed; the overlay network of CAN is embedded in a multi-dimensional torus, and the routing path from a source to a destination approximates the straight line between those two nodes in the torus. However those overlay networks are constructed without considering the '*real*' distance among nodes (the *real* distance between two nodes can be measured by the delay of the shortest path between them); and as a result, the routing is scalable in the sense of keeping small routing tables but is *not* scalable in the sense of having short routing paths. If we use k to denote the average number of logic hops a routing path has in the overlay network, then on average the *real* length of a routing path is about k times that of the shortest path that can actually be used.

A good solution to the above problem is to embed the nodes into a normed space, in such a way that the real distance between any two nodes approximately equals the distance between their images in the normed space. An overlay network constructed using the images in the normed space not only enables geographic routing, but also has routing paths whose lengths are close to the real distance between sources and destinations. Embedding graphs into normed spaces has been a very popular topic in the theoretical computer science society and the mathematics society in recent

years [34], [51], [58], [64]; embedding Internet nodes into the Euclidean space has been studied lately [65], [78], and its usage for constructing overlay networks has been suggested. However no overlay network based on such embeddings for efficient geographic routing has been constructed.

For a wireless network, the positions of its nodes can be determined by using GPS. If GPS is not available, then the positions can be recovered by using embedding algorithms [69], [77] with reasonably low overhead in computation and flow.

How to perform geographic routing for nodes in a normed space is quite obvious; however, the performance of the routing depends on the network topology. Geographic routing typically faces two major problems: the existence of 'dead-end' nodes that have no neighbors closer to the destination, which prohibits the greedy progress of routing; and the large deviation of the routing path from the straight line between the source and destination, which results in long routing delays. For wireless networks (which are assumed to be on a two-dimensional plane), the method of routing on a planar subgraph is proposed to solve the first problem [6], [43]; however the second problem remains unsolved. For nodes in higher-dimensional spaces, neither problem has known solutions. It is very important to study how to efficiently construct network topologies for nodes in normed spaces that solve both of the above problems. This is another topic of this thesis.

1.2.2 Our Approach

Both data placement using erasure-correcting codes and network topology control for nodes in normed spaces are new and broad topics. Our approach is to gain a deep theoretical understanding of these topics, through the exploration of original ideas and the discovering of novel solutions. We believe the intriguing findings here not only are valuable from the viewpoint of science, but also lay a solid basis for the real applications.

This thesis studies several data placement problems, with a special focus on the interleaving placement of data — the feature that distinguishes data placement using

erasure-correcting codes in the generalized file assignment problem, from the traditional FAP. It displays, for the first time, a data-placement construction suitable for users that can access a network through multiple access-points. It presents a new idea of topology control that leads to substantially improved system performance, measured by routing efficiency and many other criteria.

The mathematics developed in the thesis often has relations to topics beyond network data-storage systems. For example, the interleaving problem for tori has applications not only to data placement but also to error-burst correction, and is closely related to Lee-metric codes. We do not hesitate to show such relations when they exist, because that is the beauty of mathematics.

In the following we introduce the four problems studied in this thesis. For each problem, we extract and demonstrate its key result, and leave its less critical aspects to later chapters. We hope in this way the essence of the results can be made easier to grasp.

1.3 Problem 1: Maximum Interleaving and Memory Allocation on Trees

Imagine that we are assigned the following task:

"*We are asked to color the vertices of a graph with a set of colors. For every vertex, we would like there to be as many different colors as possible within a distance that is as small as possible from that vertex. How well can we do it?*"

The answer depends on the graph. If the graph is a cycle with 4 vertices and we are given 3 colors, then there will be at least 2 vertices for which there are only 2 (instead of 3) different colors within one hop. However, if the graph is a tree, then we can actually achieve the best possible result — say there are n colors, then for every vertex, the n nearest colors around it are all different.

The above finding is what we call *the maximum interleaving on a tree*. It can be achieved for a more general scenario: when the edges in the tree have different lengths

and the vertices need to be assigned different numbers of colors (i.e., multi-coloring), the tree can still be colored with the best possible result.

Maximum interleaving comes as part of a technique that we use to solve a rather general data storage problem on tree networks. It is a problem of placing the components of a codeword in the network such that every node can retrieve enough different components for recovering the codeword from nearby. Clearly the colors correspond to the codeword components. Besides this application, maximum interleaving itself is a very basic (and intriguing) property of trees.

As mentioned earlier, a data placement problem using erasure-correcting codes always consists of two parts — deciding how many codeword components to store on each vertex (which we call *memory allocation*), and deciding how to map the codeword components to the vertices. Maximum interleaving not only solves the second part, but also enables the first part to become a totally separate problem — because as long as enough codeword components (regardless of whether they are different or not) can be placed around a node, the maximum-interleaving solution can place enough *different* components around it. In Chapter II, we will present the maximum-interleaving algorithm and the memory-allocation algorithm, and study deeper aspects of them.

Traditionally, trees have been adopted by many data-storage networks, such as the Internet caching systems [72], [83], because they can naturally represent a hierarchical structure as well as the shortest paths toward a central node. In recent years, the use of trees for data storage in peer-to-peer type of networks has attracted a lot of interest [42], [46], [68]. For example, in [68], Plaxton et al. showed in a growth-restricted network (a good model for overlay networks in the Internet and for wireless networks), how to store a file (or a set of files) on a randomly chosen tree with nearly optimal data storage and access costs.

1.4 Problem 2. Optimal t-Interleaving on Tori

Imagine that we are assigned a new task:

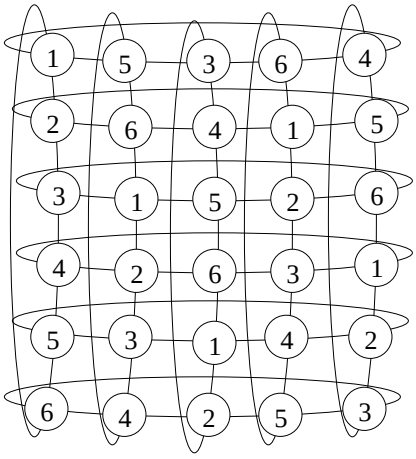


Figure 1.1: 3-Interleaving a 6×5 torus.

"*We are asked to color the vertices of an $l_1 \times l_2$ torus in such a way that for any two vertices of the same color, the shortest path between them contains at least t edges. How can we minimize the number of colors used?*"

The above problem is called *t-interleaving* on a torus. As an example, Fig. 1.1 shows how to color a 6×5 torus with the parameter $t = 3$ — that is, a 3-interleaving on a 6×5 torus. (The integers on vertices represent the colors.)

t-Interleaving generalizes the traditional one-dimensional interleaving used often in telecommunications, and was originally defined by Blaum et al. in [11] for arrays. t-Interleaving on tori has applications on network data storage, error-burst correction, and Lee-metric codes.

This thesis firstly considers those two-dimensional tori that have at least t rows and t-columns. Let us define $|S_t|$ as $|S_t| = \frac{t^2+1}{2}$ if t is odd, and $|S_t| = \frac{t^2}{2}$ if t is even; then it can be shown that $|S_t|$ is a lower bound for the number of colors that need to be used for t-interleaving a torus. In this thesis, the necessary and sufficient condition for tori to meet that lower bound is derived; and it is shown that for tori sufficiently large in both dimensions, *at most* $|S_t| + 1$ colors are needed. For both cases, optimal interleaving algorithms are presented. Then this thesis studies the problem for other cases, thus completing a general picture for t-interleaving on 2-dimensional tori.

The t-interleaving problem has been studied in a number of settings over the

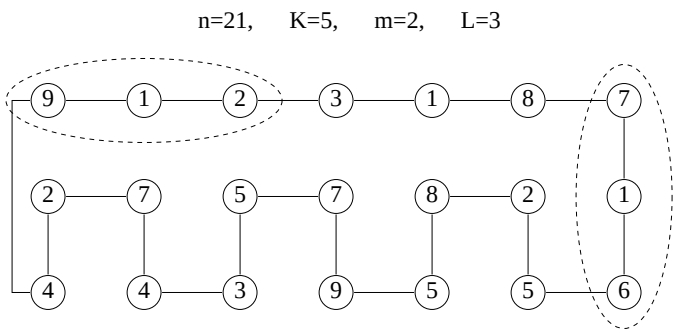


Figure 1.2: An example of multi-cluster interleaving on a cycle.

years [11], [21], [62], [75], [80]. All those works study infinite-array type of graphs. Interleaving infinite-array type of graphs is quite different from interleaving wrapping-around graphs such as tori — for the former, the optimal interleaving can usually be found by studying how to color just a small area of the graph, and then repeating the color-pattern of the small area to cover the whole graph; for the latter, however, the coloring process needs to always take into account the global structure of the graph. Our results show, for the first time, how to t-interleave graphs that have 'wrapping-around' structures.

1.5 Problem 3: Multi-Cluster Interleaving on Paths and Cycles

In a path or a cycle, we call every L consecutive vertices a *cluster*. Assume we are now assigned a third task:

"*We are asked to color a path (or cycle) of n vertices, in such a way that any m non-overlapping clusters are assigned at least K different colors. How can we minimize the number of colors used?*"

We call the above problem *multi-cluster interleaving* (MCI). As an example, Fig. 1.2 shows a multi-cluster interleaving on a cycle of 21 vertices, where every 2 clusters (of size $L = 3$) are assigned at least 5 different colors. (For instance, the two clusters in dashed circles have the colors '9, 1, 2, 7 and 6'.)

The number of colors needed for MCI generally grows with the length of the path or cycle. For example, when $L = 2$ and $K = 3$, a path can be interleaved with N colors if and only if it has no more than $(N-1)[(m-1)N-1]+2$ vertices; and when $K = L+1$, a path that can be interleaved with N colors can have at most $\frac{m-1}{(L-1)!}N^L + O(N^{L-1})$ vertices (here N is seen as the variable).

For a client outside a data-storage network, it can connect to the network through an access point and retrieve the data around that point (a connected subgraph of the network). Multi-cluster interleaving is for clients that have multiple access points. Retrieving data in parallel through multiple access points is a model studied in recent years [16], [73]; and our results demonstrate the first non-trivial data-placement construction for such a data-access model.

1.6 Problem 4: Monotone Percolation and Topology Control

Let us model the nodes in a large wireless network as points that follow a Poisson point process on an infinite 2-dimensional plane. We require every node to select its coverage radius based on only local information — the positions of their nearby neighbors. We ask the question: *when nodes make only such local and autonomous decisions, how good a network can we get?*

The answer may sound surprising — a network with many good global properties can be constructed. In particular, we can construct a network that is strongly connected, has small average degree, has small average coverage radius and power consumption, for which the routing path between any two nodes has a length (whether it is measured by hops or by its Euclidean distance) that is only a constant times the length of the straight line between those two nodes, which enables geographic routing without having any dead-ends, etc. $\cdots\cdots$

This is the first time a topology-control algorithm is proposed that achieves such comprehensive performance guarantees. The key new idea in our algorithm is a

concept that we call *monotone percolation*. In classical percolation theory, we are interested in the emergence of an infinitely large connected component. In contrast, in monotone percolation we are interested in the existence of a relatively short path that makes monotonic progress between any pair of source and destination nodes. To achieve monotone percolation, we require every node to have a set of neighbors that can make effective progress along any direction.

The algorithm can be naturally extended for points in high-dimensional normed spaces. And in fact the points do not have to follow the Poisson point process — unless the points are placed in very ill positions (such as having many big 'holes' in the space with no node inside), similar performance can be achieved. As a result, the algorithm is not only useful for wireless networks, but also for overlay networks.

1.7 Summary

This thesis studies two topics: data placement using erasure-correcting codes, and localized topology control with good global performance. They aim at the two fundamental challenges of the emerging data-storage networks — the efficient and robust storage and transmission of data. The thesis endeavors to deepen our understanding of these new topics, and to open new directions for future research. During that course, novel algorithms have been developed, basic graph properties have been discovered, and a traditional research problem has been generalized.

There are a number of publications associated with the thesis. The work on maximum interleaving and memory allocation for trees has been presented at the 2002 [35] and the 2003 [36] IEEE International Symposiums on Information Theory, and is currently under review for journal publication. The work on t-interleaving for tori will be presented at the 2004 IEEE International Symposium on Information Theory [39], and will be submitted to a journal. The work on multi-cluster interleaving has been presented at the 7th International Symposium on Communication Theory and Applications, and will appear in the journal *IEEE Transactions on Information Theory*. The work on monotone percolation and topology control [40] is under review

by the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004).

Here we have given a glimpse of our results. Now let us begin the real adventure....

Chapter 2

Maximum Interleaving and Memory Allocation in Tree Networks

2.1 The Data Placement Problem

We consider a data placement problem using erasure-correcting codes, which bounds the file-retrieving delays in a heterogeneous data-storage network, under both fault-free and faulty circumstances. Let $G = (V, E)$ be a graph representing a data-storage network, where each edge $e \in E$ has a length $l(e)$. We can understand $l(e)$ as the delay of transmitting a message across the link e. We encode a file into an erasure-correcting codeword that has N codeword components, and store replicas of those components in the network. For any two vertices $u \in V$ and $v \in V$, we define $d(u, v)$ as the length of the shortest path between vertices u and v, and call it the 'distance between u and v'. For any vertex $u \in V$ and any real number r, we define $\mathcal{N}(u, r)$ as the set of vertices within distance r from u — namely, $\mathcal{N}(u, r) = \{v | v \in V, d(u, v) \leq r\}$, and call $\mathcal{N}(u, r)$ the 'neighborhood of u of radius r'. For every vertex $u \in V$, it is associated with a natural number $W_{min}(u)$ (which is called the '*minimum memory size*' of u), a natural number $W_{max}(u)$ (which is called the '*maximum memory size*' of u), and a set $R(u) = \{(r_i(u), k_i(u)) | 1 \leq i \leq n_u\}$ (which is called the '*requirement set*' of u). (Note: each element in a 'requirement set' $R(u)$ is a pair of numbers written in the form as $(r_i(u), k_i(u))$, where $r_i(u)$ is a non-negative real number and $k_i(u)$ is a positive integer.

n_u is the cardinality of the set $R(u)$. We call each element $(r_i(u), k_i(u)) \in R(u)$ a *requirement of* u. As an example, $R(u) = \{(2.0, 5), (3.6, 8), (11.1, 12)\}$ is a requirement set of vertex u that contains 3 requirements.) The data placement problem is defined as follows:

Definition 2.1 Data Placement Problem: *Place replicas of the codeword components on the vertices of* $G = (V, E)$. *We use* $w(u)$ *to denote the number of codeword components placed on each vertex* $u \in V$. *The following two requirements need to be satisfied: (1) for each vertex* $u \in V$, $W_{min}(u) \leq w(u) \leq W_{max}(u)$; *(2) for each vertex* $u \in V$ *and for* $1 \leq i \leq n_u$, *there are at least* $k_i(u)$ *different codeword components stored on vertices in the set* $\mathcal{N}(u, r_i(u))$. *The objective is to minimize the value* $\sum_{u \in V} w(u)$. *(A solution that minimizes* $\sum_{u \in V} w(u)$ *is called an 'optimal data-placement solution'.)* □

We need to explain what the above problem means and why it is formulated this way. With the codeword components stored in the network, when a vertex wants to get the file, it will retrieve enough different codeword components for file-recovery from a neighborhood as small as possible. The radius of the neighborhood is the file-retrieving delay. We allow different vertices to have different requirements on their file-retrieving delays for generality. What's more, it is desirable that the number of distinct codeword components within a certain distance from a vertex grows steadily when that distance increases — so that the file-retrieving delay will degrade gracefully when more and more symbols become inaccessible (e.g., because of data loss or busy processors) — and we allow different vertices to have different specifications on such a relationship. We use he *requirement sets* to represent those specifications. (For example, if a vertex u requires there to be 'at least 5 different codeword components within distance 2.0, at least 8 different codeword components within distance 3.6, and at least 12 different codeword components within distance 11.1', then u's requirement set is $R(u) = \{(2.0, 5), (3.6, 8), (11.1, 12)\}$.) We also allow each vertex to have an upper bound and a lower bound on the numbers of codeword components it can store. (The lower bound can exist for various reasons, e.g., a Web

server typically always stores a copy of each file it generates.) Therefore, the meaning of the data placement problem is to store the codeword components in the network such that all the requirements on file-retrieving delays and the constraints on vertices' storage sizes are respected, with the objective of storing as few codeword components as possible in the network.

The data placement problem is NP-hard for general graphs because the NP-complete dominating set problem [23] can be reduced to it. In this paper we study the data placement problem for trees. The problem for trees has a very special optimal solution due to a basic graph-theoretic property of tree, which we call the *Maximum Interleaving*. Maximum interleaving means that the n codeword components can be stored in the tree network such that for every vertex, the n closest components are all different. (A more rigorous definition of *maximum interleaving* will be presented in Section 2.2.) As we mentioned earlier in the thesis, a solution to a data-placement problem consists of two parts — deciding how many codeword components to store on each vertex (which we call a *memory allocation* problem), and deciding how to map the codeword components to the vertices (an *interleaving* problem). For general graphs these two problems are inseparable; however for trees, the *maximum interleaving* property enables them to be solved separately — because as long as enough codeword components (whether they are different or not) can be placed in a vertex's neighborhood, the maximum-interleaving algorithm will place enough distinct codeword components in the neighborhood.

In the next section, we introduce the maximum interleaving property and two algorithms that construct K-diversity interleaving — a generalized concept of maximum interleaving – on trees. The first algorithm is *general* in the sense that every K-diversity interleaving that exists is a possible output of the algorithm. The second algorithm is more efficient but not general. Then in Section 2.3, we introduce three memory allocation algorithms. By combining the above algorithms, we find an optimal solution to the data placement problem.

2.2 K-Diversity Interleaving and Maximum Interleaving

2.2.1 Definitions

In this section we will study maximum interleaving and its generalized concept — K-diversity interleaving. They are basic properties of trees; and this chapter is a self-containing introduction on this topic. We phrase them as graph coloring problems, and it is easy to see that the colors correspond to the 'codeword components' in the data placement problem.

Let us first define some terms. Let $G = (V, E)$ be a tree, where every edge $e \in E$ has a length $l(e)$. A *point* in G is defined as a point on an edge (including the two vertices incident to the edge). If p is a point on an edge $e \in E$, 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 $\lambda \in [0, 1]$. For any two points p_1 and p_2, $d(p_1, p_2)$ 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, $\mathcal{N}(p, r) = \{v | v \in V, d(v, p) \leq r\}$. Every vertex $v \in V$ is associated with a natural number $w(v)$. We color the tree G with n colors, where each vertex is assigned $w(v)$ colors.

Definition 2.2 *We color the vertices of tree G using n colors, where every vertex $v \in V$ is assigned $w(v)$ colors. Let K be an integer, where $1 \leq K \leq n$. The coloring is called a* **K-Diversity Interleaving** *if and only if the following condition is satisfied: for every point p and every non-negative real number r, some color is assigned more than once to the vertices in $\mathcal{N}(p, r)$ only if at least K different colors are assigned to the vertices in $\mathcal{N}(p, r)$. The K-diversity interleaving is also called* **Maximum Interleaving** *when $K = n$.*

The following is an example.

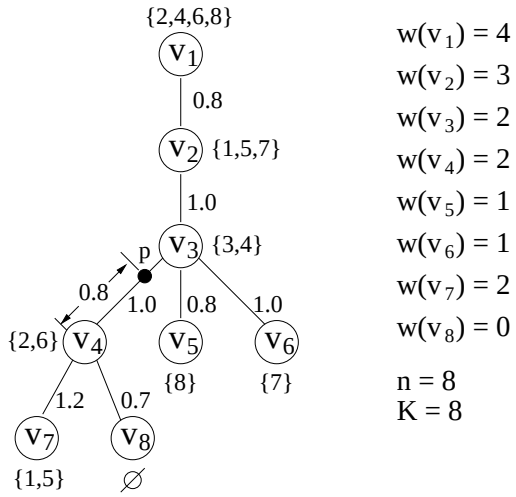


Figure 2.1: An example of K-diversity interleaving and maximum interleaving.

Example 2.1 *Fig. 2.1 shows an example of K-diversity interleaving and maximum interleaving. The parameters $w(v)$ for all vertices v are shown in the figure, and $n = K = 8$. The real number beside each edge e is its length $l(e)$. We denote the 8 colors by the numbers 1, 2, $\cdots$, 8. The colors assigned to every vertex are shown in the set beside that vertex (e.g., vertex v_1 has colors 2, 4, 6 and 8). It can be verified that for every point p and any non-negative real number r, either all the 8 different colors are assigned to the vertices in $\mathcal{N}(p, r)$, or every color is assigned at most once to the vertices in $\mathcal{N}(p, r)$. (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 — as shown in the figure — and let $r = 0.9$, then the colors assigned to $\mathcal{N}(p, r)$ are 3, 4 and 2, 6 with each color assigned only once.) So the coloring is a maximum interleaving (8-diversity interleaving). Clearly it is also a 7-diversity interleaving, 6-diversity interleaving, $\cdots$, 1-diversity interleaving.*

□

2.2.2 A General K-Diversity Interleaving Algorithm

We are going to present a *general* K-diversity interleaving algorithm. The algorithm is called '*general*' in the sense that every K-diversity interleaving that exists for the tree G is a possible output of the algorithm. (The algorithm is allowed to make some

random choices during its execution, so it has more than 1 possible output.) First let us define some new terms.

For any two points p_1 and p_2, we define $\mathcal{C}(p_1, p_2)$ 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. Define $\mathcal{S}(p_1, p_2)$ as the set of vertices within distance $\frac{d(p_1,p_2)}{2}$ from $\mathcal{C}(p_1, p_2)$, namely, $\mathcal{S}(p_1, p_2) = \mathcal{N}(\mathcal{C}(p_1, p_2), \frac{d(p_1,p_2)}{2})$.

We know that each vertex $v \in V$ is to be assigned $w(v)$ colors. We say that each vertex v has $w(v)$ *color-slots*; and assigning $w(v)$ colors to v is understood as 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 also say each of v's color-slots is 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 $d(p, s)$.

We let G be a rooted tree, and denote its root by γ. We define $\mathbb{W}$ as $\mathbb{W} = \sum_{v \in V} w(v)$. G has $\mathbb{W}$ color-slots, which we label by $s_1, s_2, \cdots, s_{\mathbb{W}}$ 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 2.1: *A General K-Diversity Interleaving Algorithm for Tree $G = (V, E)$*

1. Assign K distinct colors to the following K color-slots: $s_1, s_2, \cdots, s_K$, 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 2.1 can be implemented with 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 2.1.

Lemma 2.1 *We use Algorithm 2.1 to color 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, $\cdots$, 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 2.1, 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 2.2 *Assume there is a coloring on the tree $G = (V, E)$, which uses n different colors and assigns $w(v)$ colors to every vertex $v \in V$. The coloring is a K-diversity 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-diversity interleaving, and assume 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 the definition of K-diversity interleaving, 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

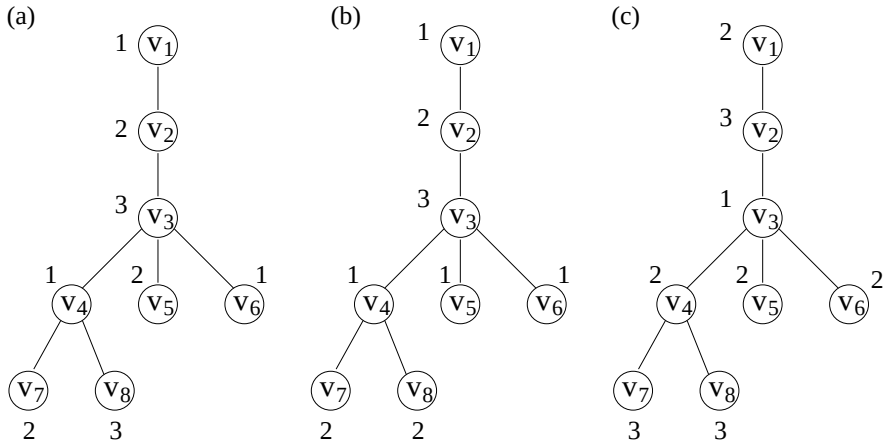


Figure 2.2: Different K-diversity interleavings on the same tree.

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, the coloring is a K-diversity interleaving.

□

Lemma 2.1 and Lemma 2.2 together naturally establish the following conclusion.

Theorem 2.1 *Algorithm 2.1 correctly outputs a K-diversity interleaving for the tree $G = (V, E)$.*

Since Algorithm 2.1 always has an output, we have the following corollary.

Corollary 2.1 *K-diversity interleaving exists for all trees.*

2.2.3 Generality of K-Diversity Interleaving Algorithm 2.1

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

Example 2.2 *Fig. 2.2 (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 is simple to verify that each coloring is a K-diversity interleaving for $K = 3$. 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.* □

When Algorithm 2.1 colors a tree, firstly it needs to label the $\mathbb{W}$ color-slots as s_1, s_2, $\cdots$, $s_{\mathbb{W}}$, 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 2.1 can output different K-diversity interleavings. The following theorem shows that in fact, every K-diversity interleaving that exists is a possible output of Algorithm 2.1, — namely, the set of possible outputs of Algorithm 2.1 is exactly the set of K-diversity interleavings that exist, — 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 2.2 *The set of possible outputs of Algorithm 2.1 is exactly the set of K-diversity interleavings that exist 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 2.1, the set of possible outputs of Algorithm 2.1 is a subset of the set of K-diversity interleavings that exist. So the only thing left to prove is that every K-diversity interleaving is a possible output of Algorithm 2.1, even if the root of the tree is fixed.

Assume the root of the tree is fixed. Assume a fixed K-diversity 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-diversity interleaving must have assigned $\mathbb{W}$ distinct colors to the $\mathbb{W}$ color-slots, which is clearly a possible output of Algorithm 2.1. 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 2.1 can label as s_1, s_2, $\cdots$, s_i and assign the same colors to as the given K-diversity 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 $\sum_{v \in \mathcal{N}(\gamma, r)} w(v) \geq j$. It is simple to see that there exist a set of K color-slots that are assigned distinct colors in the given K-diversity interleaving and are at distance no greater than $R_{min}(\gamma, K)$ from γ, and that every color-slot at distance less than $R_{min}(\gamma, K)$ from γ is contained in that set. Clearly it is feasible for Algorithm 2.1 to label those color-slots as s_1, s_2, $\cdots$, s_K, and assign them the same colors as the given K-diversity 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 2.1 can label as s_1, s_2, $\cdots$, s_i and assign the same colors to as the given K-diversity 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, $\cdots$, $c_{|H|}$. 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-diversity 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-diversity 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-diversity 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-diversity 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-diversity 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}, $\cdots$ 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-diversity interleaving. Then it's simple to see that it is feasible for Algorithm 2.1 to label c_{j_x} as the color-slot s_{I+1} and assign s_{I+1} the same color as the given K-diversity interleaving does, after labelling the color-slots in C as s_1, s_2, $\cdots$, s_I and assigning the same colors to them as the given K-diversity 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 2.1 can assign the same colors to all the $\mathbb{W}$ color-slots as the arbitrarily given K-diversity interleaving does. So any K-diversity interleaving is a possible output of Algorithm 2.1, even if the root of the tree is fixed. Therefore this theorem is proved.

□

2.2.4 An Efficient K-Diversity Interleaving Algorithm

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

Algorithm 2.2: *An Efficient K-Diversity Interleaving Algorithm for Tree $G = (V, E)$*

1. Denote the root of G by γ. Label the vertices in G as $v_1, v_2, \cdots, v_{|V|}$ 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 $s_1, s_2, \cdots, s_{\mathbb{W}}$ in the following way: for $1 \leq i \leq |V|$, label the $w(v_i)$ color-slots of vertex v_i as $s_{W_i+1}, s_{W_i+2}, \cdots, s_{W_i+w(v_i)}$, 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 $C_1, C_2, \cdots, C_K$, such that any two color-slots are assigned the same color if and only if they are in the same set.

□

Example 2.3 *The following example is used to illustrate how Algorithm 2.2 computes. In the example, the tree $G = (V, E)$ is as shown in Fig. 2.3. 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 $v_1, v_2, \cdots, v_{13}$ which is consistent with the way Algorithm 2.2 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. 2.3. Algorithm 2.2 is used to compute a K-diversity interleaving for the tree using n distinct colors, where $n = 6$ and $K = 5$.*

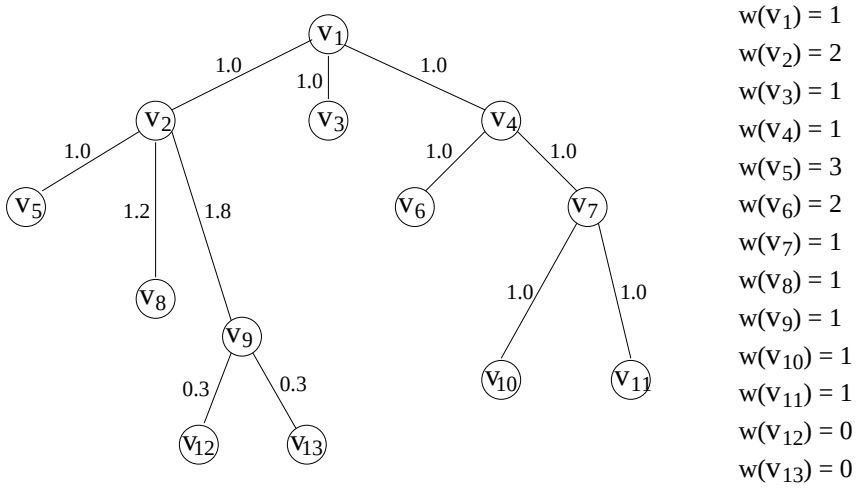


Figure 2.3: An example of K-diversity interleaving using Algorithm 2.2.

There are $\mathbb{W} = \sum_{v \in V} w(v) = 15$ *color-slots in* G*. Algorithm 2.2 labels them as* s_1, s_2, $\cdots$, 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
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}
Vertex v	v_{10}	v_{11}	v_{12}	v_{13}					
Color-slots of vertex v	s_{14}	s_{15}							

Table 1: The labelling of the color-slots

Algorithm 2.2 then assigns values to 5 sets C_1, C_2, $\cdots$, C_5 *as follows:* $C_i = \{s_i\}$ *for* $i = 1, 2, \cdots, 5$.

Next, for $i = 6$ *to 15, Algorithm 2.2 finds the color-slot* s_x *corresponding to* i *(This is the step 5 in Algorithm 2.2. 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 2.2 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 2.2

After the above computing, the values of the 5 sets C_1, C_2, $\cdots$, 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 2.2 arbitrarily selects 5 distinct colors — 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*-diversity interleaving with* K*=5.*

□

It's simple to see that with a K-diversity interleaving output by Algorithm 2.2, the tree $G = (V, E)$ has exactly K distinct colors.

Analysis shows that Algorithm 2.2 can be implemented with time complexity $O(|V|^2 + \mathbb{W})$. And it can be shown that Algorithm 2.2 is *indeed* more efficient than Algorithm 2.1, the general K-diversity interleaving algorithm. The complexity analysis is presented in Appendix III.

The major computation in Algorithm 2.2 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 2.2'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 2.2.

[2]Processing the color-slots one by one is one straightforward way to implement Algorithm 2.2'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.

Lemma 2.3 *Assume Algorithm 2.2 is used to produce a K-diversity interleaving for the tree $G = (V, E)$. Algorithm 2.2 labels the vertices in G as v_1, v_2, $\cdots$, $v_{|V|}$, and labels the color-slots in G as s_1, s_2, $\cdots$, $s_{\mathbb{W}}$. Assume $\mathbb{W} > K$. At the end of Algorithm 2.2, the K sets C_1, C_2, $\cdots$, 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 $s_1, s_2, \cdots, s_{i-1}$ 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, $\cdots$, 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, $h(i) \geq j$.

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, $\cdots$, s_K. s_1, s_2, $\cdots$, s_K are evenly distributed in the K sets C_1, C_2, $\cdots$, C_K. Therefore Lemma 2.3 is true in this case. Now we start proving Lemma 2.3 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, $\cdots$, 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, $\cdots$, 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, $\cdots$, 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 2.2 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 2.2 that $s_{h(i)}$ is also in C_t. By the indiction assumption, the first K color-slots in $L_{h(i)}$ are evenly distributed in the K sets C_1, C_2, $\cdots$, C_K. So the first K color-slots in L_i are also evenly distributed in the K sets C_1, C_2, $\cdots$, 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 2.2 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 indiction assumption, the first K color-slots in $L_{h(i)}$ are evenly distributed in the K sets C_1, C_2, $\cdots$, C_K. So the first K color-slots in L_i are also evenly distributed in the K sets C_1, C_2, $\cdots$, C_K. The analysis of Case 2 ends here.

The proof by induction is complete here. So Lemma 2.3 is proved.

□

Theorem 2.3 *Algorithm 2.2 correctly outputs a K-diversity interleaving for the tree $G = (V, E)$.*

Proof: This theorem can be proved by showing that Algorithm 2.2 is a special case of Algorithm 2.1, namely, all the computation in Algorithm 2.1 is also implemented

in Algorithm 2.2.

Assume Algorithm 2.2 is used to produce a K-diversity interleaving for a tree $G = (V, E)$. Algorithm 2.2 labels the vertices in G as v_1, v_2, $\cdots$, $v_{|V|}$, and labels the color-slots in G as s_1, s_2, $\cdots$, $s_{\mathbb{W}}$. At the end of Algorithm 2.2, the K sets C_1, C_2, $\cdots$, C_K is a partition of all the color-slots in G.

Let γ denote the root of G. By the way Algorithm 2.2 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 2.1 is also followed in Algorithm 2.2.

If $\mathbb{W} \leq K$, then Algorithm 2.2 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 2.1 does, too. So if $\mathbb{W} \leq K$, Algorithm 2.2 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 2.2 assigns K distinct colors to the following K color-slots: s_1, s_2, $\cdots$, s_K, such that no two color-slots among them are assigned the same color, which is what Algorithm 2.1'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 $s_1, s_2, \cdots, s_{y-1}$ 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 2.2 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 2.3, $s_q \notin C_t$. So s_y and s_q are assigned different colors by Algorithm 2.2, which is what Algorithm 2.1's step 2 does, too.

By now it's proven that all the computation in Algorithm 2.1 is also implemented in Algorithm 2.2. Therefore Algorithm 2.2 is a special case of Algorithm 2.1. So

Algorithm 2.2 will correctly output a K-diversity interleaving for the tree $G = (V, E)$, just as Algorithm 2.1 does.

□

2.3 Memory Allocation

2.3.1 Definitions

The result on maximum interleaving (or more generally, the K-diversity interleaving) has essentially solved one part of the data placement problem defined in Section 2.1 — how to map the codeword components to the vertices once how many components to place on each vertex is known. Then clearly, for the data placement problem, the question of how many codeword components should be placed on each vertex for an optimal data-placement solution is reduced to the following problem:

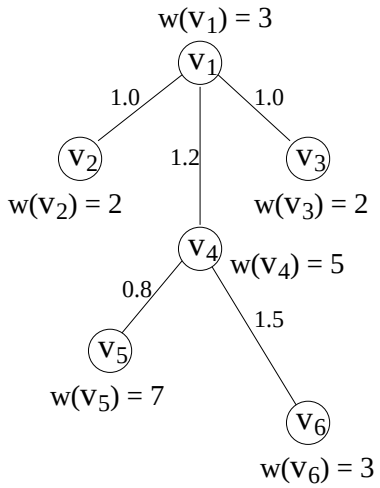
Definition 2.3 Memory Allocation Problem

INSTANCE: A tree $G = (V, E)$. Every edge $e \in E$ has a length $l(e)$. Every vertex $v \in V$ is associated with a set $R(v) = \{(r_i(v), k_i(v)) | 1 \leq i \leq n_v\}$, which is called the 'requirement set' *of v. Each vertex $v \in V$ is also associated with a parameter $W_{min}(v)$, which is called the* 'minimum memory size' *of v, and a parameter $W_{max}(v)$, which is called the* 'maximum memory size' *of v.*

QUESTION: How to assign a natural number $w(v)$ ($W_{min}(v) \leq w(v) \leq W_{max}(v)$) to each vertex $v \in V$, with the value of $\sum_{v \in V} w(v)$ minimized, such that for any vertex $u \in V$ and for $1 \leq i \leq n_u$, $\sum_{v \in \mathcal{N}(u, r_i(u))} w(v) \geq k_i(u)$? (Here $w(v)$ is called the 'memory size *of v'. A solution to this memory allocation problem is called an* optimal memory allocation.*)*

□

Example 2.4 *Fig. 2.4 shows an example of the memory allocation problem. The graph $G = (V, E)$ has 6 vertices; the number beside each edge is its length. A solution is also given: $w(v_1) = 3$, $w(v_2) = 2$, $w(v_3) = 2$, $w(v_4) = 5$, $w(v_5) = 7$, $w(v_6) = 3$,*



$R(v_1) = \{ (1.0, 7), (1.2, 11) \}$
$R(v_2) = \{ (1.2, 5) \}$
$R(v_3) = \{ (1.3, 5) \}$
$R(v_4) = \{ (0.6, 5) \}$
$R(v_5) = \{ (1.6, 12) \}$
$R(v_6) = \{ (1.0, 3) \}$

$W_{min}(v_1) = 0$	$W_{max}(v_1) = 3$
$W_{min}(v_2) = 0$	$W_{max}(v_2) = 5$
$W_{min}(v_3) = 1$	$W_{max}(v_3) = 6$
$W_{min}(v_4) = 0$	$W_{max}(v_4) = 5$
$W_{min}(v_5) = 3$	$W_{max}(v_5) = 10$
$W_{min}(v_6) = 0$	$W_{max}(v_6) = 8$

Figure 2.4: An example of the memory allocation problem.

which is shown in the figure. We claim without proof that no other solution has a smaller value of $\sum_{v \in V} w(v)$*; readers can verify that the claim is true.* □

In the following subsections, we present three polynomial-time algorithms. The first two algorithms respectively solve the memory allocation problem with and without upper bounds on vertices' memory sizes. They have complexities of $O(q|V|^3)$ and $O(q|V|^2)$, where q is the average cardinality of a requirement set. The third algorithm finds a solution that minimizes the greatest memory size of single vertices, among all the optimal solutions. It's complexity is $O(q|V|^3 log(Y - \frac{X}{|V|}))$, where Y is the greatest memory size of single vertices in some memory-allocation solution, and X is the total memory size in that solution.

2.3.2 A Memory Allocation Algorithm

In this subsection, we present an algorithm for the memory allocation problem.

The following proposition is self-evident.

Proposition 2.1 *The memory allocation problem has a solution if and only if for any* $v \in V$ *and for* $1 \leq i \leq n_v$, $\sum_{u \in \mathcal{N}(v, r_i(v))} W_{max}(u) \geq k_i(v)$.

From now on we always assume a solution exists for the memory allocation problem.

For any two vertices v_1 and v_2 in a tree $G = (V, E)$, we say 'v_1 is a descendant of v_2' or 'v_2 is an ancestor of v_1' if $v_2 \neq v_1$ and v_2 is on the shortest path between v_1 and the root. We say 'v_1 is a child of v_2' or 'v_2 is the parent of v_1' if v_1 and v_2 are adjacent and v_1 is a descendant of v_2. For any vertex $v \in V$, we use $Des(v)$ to denote the set of descendants of v.

Definition 2.4 An Optimal Memory Basis: *A set* $\{w(v)|v \in V\}$ *is called an* optimal memory basis *if there exists an optimal memory allocation for the tree* $G = (V, E)$ *which assigns memory size* $w_{opt}(v)$ *to every vertex* $v \in V$*, such that for any* $v \in V$, $W_{min}(v) \leq w(v) \leq w_{opt}(v)$. □

The following lemma shows how one can get a new optimal memory basis from an old optimal memory basis by increasing memory sizes.

Lemma 2.4 *Let* u_1 *be a child of* u_2 *in the tree* $G = (V, E)$*. And let* $\{w_1(v)|v \in V\}$ *be an optimal memory basis. Assume the following conditions are true for the memory allocation problem: "for any vertex* $v \in Des(u_1)$*, the 'requirement set'* $R(v) = \emptyset$*;* $R(u_1)$ *has an element* (r, k)*, namely,* $(r, k) \in R(u_1)$*."*

We define S_1 *as* $S_1 = \mathcal{N}(u_2, r - d(u_1, u_2))$*, and define* S_2 *as* $S_2 = \mathcal{N}(u_1, r) - S_1$*. We compute the elements of a set* $\{w_2(v)|v \in V\}$ *in the following way (step 1 to step 3):*

Step 1: for all $v \in V$*, let* $w_2(v) \leftarrow w_1(v)$*.*

Step 2: Let

$$X \leftarrow \max\{0, k - \sum_{v \in S_1} W_{max}(v) - \sum_{v \in S_2} w_1(v)\},$$

and let $C \leftarrow S_2$*.*

Step 3: Let v_0 *be the vertex in* C *that is the closest to* u_1*—namely,* $d(v_0, u_1) = \min_{v \in C} d(v, u_1)$*. Let* $w_2(v_0) \leftarrow \min\{W_{max}(v_0), w_1(v_0) + X\}$*. Let* $X \leftarrow X - (w_2(v_0) - w_1(v_0))$*, and let* $C \leftarrow C - \{v_0\}$*. Repeat Step 3 until* X *equals 0.*

Then the following conclusion is true: $\{w_2(v)|v \in V\}$ *is also an optimal memory basis.*

Proof: The following three conclusions can be easily seen to be true:

Conclusion (1): $S_1 \cup S_2 = N(u_1, r)$, and $S_1 \cap S_2 = \emptyset$.

Conclusion (2): For any $v \in S_2$, $W_{min}(v) \leq w_1(v) \leq w_2(v) \leq W_{max}(v)$. For any $v \in V - S_2$, $W_{min}(v) \leq w_1(v) = w_2(v) \leq W_{max}(v)$. And

$$\sum_{v \in V} w_2(v) - \sum_{v \in V} w_1(v) = \max\{0, k - \sum_{v \in S_1} W_{max}(v) - \sum_{v \in S_2} w_1(v)\}$$

Conclusion (3):

$$\sum_{v \in S_1} W_{max}(v) + \sum_{v \in S_2} w_2(v) \geq k$$

Now let's use them to prove Lemma 2.4.

$\{w_1(v)|v \in V\}$ is an optimal memory basis. So there exists an optimal memory allocation that assigns memory size $w_{opt}(v)$ to every vertex $v \in V$, such that $w_1(v) \leq w_{opt}(v)$ for any $v \in V$. By Definition 2.3 we know $\sum_{v \in \mathcal{N}(u_1,r)} w_{opt}(v) \geq k$. Since $\sum_{v \in \mathcal{N}(u_1,r)} w_{opt}(v) = \sum_{v \in S_1} w_{opt}(v) + \sum_{v \in S_2} w_{opt}(v) \leq \sum_{v \in S_1} W_{max}(v) + \sum_{v \in S_2} w_{opt}(v)$, we get $\sum_{v \in S_2} w_{opt}(v) \geq k - \sum_{v \in S_1} W_{max}(v)$. And clearly $\sum_{v \in S_2} w_{opt}(v) \geq \sum_{v \in S_2} w_1(v)$. By conclusion (2), $\sum_{v \in S_2} w_2(v) = \sum_{v \in S_2} w_2(v) + \sum_{v \in V - S_2} w_2(v) - \sum_{v \in V - S_2} w_1(v) - \sum_{v \in S_2} w_1(v) + \sum_{v \in S_2} w_1(v) = \sum_{v \in V} w_2(v) - \sum_{v \in V} w_1(v) + \sum_{v \in S_2} w_1(v) = \max\{0, k - \sum_{v \in S_1} W_{max}(v) - \sum_{v \in S_2} w_1(v)\} + \sum_{v \in S_2} w_1(v) = \max\{\sum_{v \in S_2} w_1(v), k - \sum_{v \in S_1} W_{max}(v)\}$. So $\sum_{v \in S_2} w_{opt}(v) \geq \sum_{v \in S_2} w_2(v)$.

We compute the elements of a set $\{w_o(v)|v \in V\}$ in the following way (step 1 to step 3):

Step 1: for all $v \in V - S_2$, let $w_o(v) \leftarrow w_{opt}(v)$. For all $v \in S_2$, let $w_o(v) \leftarrow w_1(v)$.

Step 2: Let $Y \leftarrow \sum_{v \in S_2} w_{opt}(v) - \sum_{v \in S_2} w_1(v)$, and let $C \leftarrow S_2$.

Step 3: Let v_0 be the vertex in C that is the closest to u_1—namely, $d(v_0, u_1) = \min_{v \in C} d(v, u_1)$. Let $w_o(v_0) \leftarrow \min\{W_{max}(v_0), w_1(v_0) + Y\}$. Let $Y \leftarrow Y - (w_o(v_0) - w_1(v_0))$, and let $C \leftarrow C - \{v_0\}$. Repeat Step 3 until X equals 0.

From the above three steps, it's simple to see that the following must be true: for

any $v \in V$, $w_o(v) \geq w_2(v)$; for any $v \in V - S_2$, $w_o(v) = w_{opt}(v)$; for any $v \in S_2$, $W_{min}(v) \leq w_o(v) \leq W_{max}(v)$; $\sum_{v \in V} w_o(v) = \sum_{v \in V} w_{opt}(v)$, and $\sum_{v \in S_2} w_o(v) = \sum_{v \in S_2} w_{opt}(v)$. It's easy to see that the following must also be true: if there exists a vertex $v_1 \in S_2$ such that $w_o(v_1) > w_1(v_1)$, then for any $v \in S_2$ such that $d(v, u_1) < d(v_1, u_1)$, $w_o(v) = W_{max}(v)$; if there exists a vertex $v_2 \in S_2$ such that $w_o(v_2) < W_{max}(v_2)$, then for any $v \in S_2$ such that $d(v, u_1) > d(v_2, u_1)$, $w_o(v) = w_1(v)$. Therefore for any real number L, if we define Q as $Q = \{v | v \in S_2, d(v, u_1) \leq L)\}$, then $\sum_{v \in Q} w_o(v) \geq \sum_{v \in Q} w_{opt}(v)$.

Let $v_0 \in V$ be any vertex such that $R(v_0) \neq \emptyset$, and let (r_0, k_0) be any element in $R(v_0)$. Clearly $v_0 \notin Des(u_1)$. Since $S_2 \subseteq Des(u_1) \cup \{u_1\}$, $\mathcal{N}(v_0, r_0) = \{v | v \in \mathcal{N}(v_0, r_0), v \notin S_2\} \cup \{v | v \in \mathcal{N}(v_0, r_0), v \in S_2\} = \{v | v \in \mathcal{N}(v_0, r_0), v \notin S_2\} \cup \{v | v \in S_2, d(v, v_0) \leq r_0\} = \{v | v \in \mathcal{N}(v_0, r_0), v \notin S_2\} \cup \{v | v \in S_2, d(v, u_1) \leq r_0 - d(u_1, v_0)\}$. So $\sum_{v \in \mathcal{N}(v_0, r_0)} w_o(v) \geq \sum_{v \in \mathcal{N}(v_0, r_0)} w_{opt}(v)$. Clearly $\sum_{v \in \mathcal{N}(v_0, r_0)} w_{opt}(v) \geq k_0$. So $\sum_{v \in \mathcal{N}(v_0, r_0)} w_o(v) \geq k_0$. Since $\sum_{v \in V} w_o(v) = \sum_{v \in V} w_{opt}(v)$, the memory allocation which assigns memory size $w_o(v)$ to every vertex $v \in V$ is also an optimal memory allocation.

For any $v \in V$, $W_{min}(v) \leq w_2(v) \leq w_o(v)$. So $\{w_2(v) | v \in V\}$ is an optimal memory basis.

□

The following lemma shows how one can transform one memory allocation problem into another by modifying the 'requirement sets' and the 'minimum memory sizes'.

Lemma 2.5 *u_1 is a child of u_2 in the tree $G = (V, E)$. And $\{w_0(v) | v \in V\}$ is an optimal memory basis. Assume the following conditions are true for the memory allocation problem: "for any vertex $v \in Des(u_1)$, the 'requirement set' $R(v) = \emptyset$; for any element in $R(u_1)$—say the element is (r, k)—we have $\sum_{u \in \mathcal{N}(u_2, r - d(u_1, u_2))} W_{max}(u) + \sum_{u \in \mathcal{N}(u_1, r) - \mathcal{N}(u_2, r - d(u_1, u_2))} w_0(u) \geq k$."*

We compute the elements of a set $\{\hat{R}(v) | v \in V\}$ in the following way (step 1 and step 2):

Step 1: for all $v \in V$, let $\hat{R}(v) \leftarrow R(v)$.

Step 2: let (r,k) *be an element in* $\hat{R}(u_1)$*. If* $\sum_{v\in\mathcal{N}(u_1,r)} w_0(v) < k$*, then add an element* $(r-d(u_1,u_2), k-\sum_{v\in\mathcal{N}(u_1,r)-\mathcal{N}(u_2,r-d(u_1,u_2))} w_0(v))$ *to the set* $\hat{R}(u_2)$*. Remove the element* (r,k) *from* $\hat{R}(u_1)$*. Repeat Step 2 until* $\hat{R}(u_1)$ *becomes an empty set.*

Let's call the original memory allocation problem, in which the 'requirement set' of each vertex $v \in V$ *is* $R(v)$*, the* 'old problem'*. We derive a new memory allocation problem—which we call the* 'new problem'*—in the following way: in the* 'new problem' *everything is the same as in the* 'old problem'*, except that for each vertex* $v \in V$*, its 'requirement set' is* $\hat{R}(v)$ *instead of* $R(v)$*, and its 'minimum memory size' is* $w_0(v)$ *instead of* $W_{min}(v)$*.*

Then the following conclusions are true:

(1) The 'new problem' *has a solution (an optimal memory allocation).*

(2) An optimal memory allocation for the 'new problem' *is also an optimal memory allocation for the* 'old problem'*.*

Proof: Conclusion (1) can be easily proved by using Proposition 2.1 and the assumption we have here that the *'old problem'* has a solution. Below we give the proof of conclusion (2).

Consider an optimal memory allocation for the *'new problem'* which assigns 'memory size' $\hat{w}_{opt}(v)$ to each vertex $v \in V$. Let $\bar{v} \in V$ be any vertex such that $R(\bar{v}) \neq \emptyset$, and let $(\bar{r},\bar{k})$ be any element in $R(\bar{v})$. Either $(\bar{r},\bar{k}) \in \hat{R}(\bar{v})$ or $(\bar{r},\bar{k}) \notin \hat{R}(\bar{v})$. If $(\bar{r},\bar{k}) \in \hat{R}(\bar{v})$, then clearly $\sum_{u\in\mathcal{N}(\bar{v},\bar{r})} \hat{w}_{opt}(u) \geq \bar{k}$. Now consider the case where $(\bar{r},\bar{k}) \notin \hat{R}(\bar{v})$. Clearly in this case $\bar{v} = u_1$, and either $\sum_{u\in\mathcal{N}(u_1,\bar{r})} w_0(u) \geq \bar{k}$, or $\sum_{u\in\mathcal{N}(u_1,\bar{r})} w_0(u) < \bar{k}$. If $\sum_{u\in\mathcal{N}(u_1,\bar{r})} w_0(u) \geq \bar{k}$, since $\hat{w}_{opt}(u) \geq w_0(u)$ for any $u \in V$, we have $\sum_{u\in\mathcal{N}(\bar{v},\bar{r})} \hat{w}_{opt}(u) \geq \bar{k}$. We define S_1 as $S_1 = \mathcal{N}(u_2, \bar{r} - d(u_1,u_2))$, and define S_2 as $S_2 = \mathcal{N}(u_1,\bar{r}) - S_1$. Then if $\sum_{u\in\mathcal{N}(u_1,\bar{r})} w_0(u) < \bar{k}$, its simple to see that $(\bar{r} - d(u_1,u_2), \bar{k} - \sum_{u\in S_2} w_0(u)) \in \hat{R}(u_2)$. So $\sum_{u\in\mathcal{N}(\bar{v},\bar{r})} \hat{w}_{opt}(u) = \sum_{u\in S_1} \hat{w}_{opt}(u) + \sum_{u\in S_2} \hat{w}_{opt}(u) \geq \bar{k} - \sum_{u\in S_2} w_0(u) + \sum_{u\in S_2} \hat{w}_{opt}(u) \geq \bar{k}$. Therefore $\sum_{u\in\mathcal{N}(\bar{v},\bar{r})} \hat{w}_{opt}(u) \geq \bar{k}$ in all cases.

$\{w_0(v)|v \in V\}$ is an optimal memory basis for the *'old problem'*. So there exists an optimal memory allocation for the '*old problem*' which assigns 'memory size' $w_{opt}(v)$

to each vertex $v \in V$, such that for any $v \in V$, $w_0(v) \leq w_{opt}(v)$.

We compute the elements of four sets — $\{w_1(v)|v \in V\}$, $\{w_2(v)|v \in V\}$, $\{w_3(v)|v \in V\}$, and $\{w_4(v)|v \in V\}$ — in the following way (step 1 to step 5):

Step 1: for any $v \in Des(u_2)$, let $w_1(v) \leftarrow w_0(v)$. For any $v \in V - Des(u_2)$, let $w_1(v) \leftarrow w_{opt}(v)$.

Step 2: for any $v \in Des(u_2)$, let $w_2(v) \leftarrow w_{opt}(v) - w_0(v)$. For any $v \in V - Des(u_2)$, let $w_2(v) \leftarrow 0$.

Step 3: for any $v \in V$, let $w_3(v) \leftarrow 0$. Let $Z \leftarrow \sum_{v \in V} w_2(v)$, and let $C \leftarrow V$.

Step 4: Let v_0 be the vertex in C that is the closest to u_2—namely, $d(v_0, u_2) = \min_{v \in C} d(v, u_2)$. Let $w_3(v_0) \leftarrow \min\{W_{max}(v_0) - w_1(v_0), Z\}$. Let $Z \leftarrow Z - w_3(v_0)$, and let $C \leftarrow C - \{v_0\}$. Repeat Step 4 until Z equals 0.

Step 5: for any $v \in V$, let $w_4(v) \leftarrow w_1(v) + w_3(v)$.

From the above five steps, it's simple to see that the following must be true: "$\sum_{v \in V} w_{opt}(v) = \sum_{v \in V} w_1(v) + \sum_{v \in V} w_2(v) = \sum_{v \in V} w_4(v)$, and $\sum_{v \in V} w_2(v) = \sum_{v \in V} w_3(v)$; for any $v \in V$, $w_0(v) \leq w_4(v) \leq W_{max}(v)$; for any real number L, $\sum_{v \in N(u_2, L)} w_3(v) \geq \sum_{v \in N(u_2, L)} w_2(v)$; for any $v \in V$, if $w_4(v) < W_{max}(v)$, then $\sum_{u \in N(u_2, d(v, u_2))} w_3(u) = \sum_{u \in V} w_2(u)$."

Let $\hat{v} \in V$ be any vertex such that $\hat{R}(\hat{v}) \neq \emptyset$, and let $(\hat{r}, \hat{k})$ be any element in $\hat{R}(\hat{v})$. Clearly $\hat{v} \in V - Des(u_2)$. Either $(\hat{r}, \hat{k}) \in R(\hat{v})$ or $(\hat{r}, \hat{k}) \notin R(\hat{v})$. If $(\hat{r}, \hat{k}) \in R(\hat{v})$, then $\sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_4(v) = \sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_1(v) + \sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_3(v) \geq \sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_1(v) + \sum_{v \in \mathcal{N}(u_2, \hat{r} - d(\hat{v}, u_2))} w_3(v) \geq \sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_1(v) + \sum_{v \in \mathcal{N}(u_2, \hat{r} - d(\hat{v}, u_2))} w_2(v) = \sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_1(v) + \sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_2(v) = \sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_{opt}(v) \geq \hat{k}$.

Now consider the case where $(\hat{r}, \hat{k}) \notin R(\hat{v})$. We define $\tilde{r}$ as $\tilde{r} = \hat{r} + d(u_1, u_2)$, define $\hat{S}_1$ as $\hat{S}_1 = N(u_2, \hat{r})$, define $\hat{S}_2$ as $\hat{S}_2 = \mathcal{N}(u_1, \tilde{r}) - \hat{S}_1$, and define $\tilde{k}$ as $\tilde{k} = \hat{k} + \sum_{v \in \hat{S}_2} w_0(v)$. It's easy to see in this case $\hat{v} = u_2$, and $(\tilde{r}, \tilde{k}) \in R(u_1)$. If $w_4(v) = W_{max}(v)$ for any $v \in \mathcal{N}(u_2, \hat{r})$, then clearly $\sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_4(v) \geq \hat{k}$ because the *new problem* has a solution. If there exists $v_0 \in \mathcal{N}(u_2, \hat{r})$ such that $w_4(v_0) < W_{max}(v_0)$, then $\sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_4(v) = \sum_{v \in \hat{S}_1} w_1(v) + \sum_{v \in V} w_2(v) \geq \sum_{v \in \mathcal{N}(u_1, \tilde{r})} w_1(v) - \sum_{v \in \hat{S}_2} w_1(v) + \sum_{v \in \mathcal{N}(u_1, \tilde{r})} w_2(v) = \sum_{v \in \mathcal{N}(u_1, \tilde{r})} w_{opt}(v) - \sum_{v \in \hat{S}_2} w_1(v) \geq \tilde{k} - \sum_{v \in \hat{S}_2} w_1(v) = \hat{k}$.

So $\sum_{v \in \mathcal{N}(\hat{v}, \hat{r})} w_4(v) \geq \hat{k}$ in all cases. So $\sum_{v \in V} \hat{w}_{opt}(v) \leq \sum_{v \in V} w_4(v)$. Since

we also have $\sum_{v\in V} w_4(v) = \sum_{v\in V} w_{opt}(v)$ and $\sum_{v\in V} w_{opt}(v) \leq \sum_{v\in V} \hat{w}_{opt}(v)$, we get $\sum_{v\in V} \hat{w}_{opt}(v) = \sum_{v\in V} w_{opt}(v)$. So the optimal memory allocation for the *'new problem'*, which assigns 'memory size' $\hat{w}_{opt}(v)$ to every vertex $v \in V$, is also an optimal memory allocation for the *'old problem'*. So conclusion (2) is proved.

□

Based on the above two lemmas, we naturally get the following memory allocation algorithm. The algorithm processes all the vertices one by one. Every time a vertex is processed, it uses the method in Lemma 2.4 to update the memory sizes , and uses the method in Lemma 2.5 to transform the memory allocation problem, until a solution is found.

Algorithm 3.1: *Memory Allocation on Tree $G = (V, E)$*

1. Initially, for every vertex $v \in V$, let $w(v) \leftarrow W_{min}(v)$.

2. Process all the vertices one by one in an order that follows the following rule: "every vertex is processed before any of its ancestors." For each vertex $\tilde{v} \in V$ that is not the root, it is processed with the following two steps:

"Step 1: Treat $\tilde{v}$, the parent of $\tilde{v}$ and the set $\{w(v)|v \in V\}$ as the vertex 'u_1', the vertex 'u_2' and the set '$\{w_1(v)|v \in V\}$' in Lemma 2.4, respectively, and for each element in $R(\tilde{v})$ do the following two things: (1) treat this element in $R(\tilde{v})$ as the element '(r, k)' in Lemma 2.4, and compute the set '$\{w_2(v)|v \in V\}$' as in Lemma 2.4; (2) for every vertex $v \in V$, let $w(v)$ get the value of $w_2(v)$ — namely, $w(v) \leftarrow w_2(v)$.

Step 2: Treat $\tilde{v}$, the parent of $\tilde{v}$ and the set $\{w(v)|v \in V\}$ as the vertex 'u_1', the vertex 'u_2' and the set '$\{w_0(v)|v \in V\}$' in Lemma 2.5, respectively, and do the following two things: (1) compute the set '$\{\hat{R}(v)|v \in V\}$' as in Lemma 2.5; (2) for every vertex $v \in V$, let $R(v) \leftarrow \hat{R}(v)$, and let $W_{min}(v) \leftarrow w(v)$."

Denote the root by v_{root}. The vertex v_{root} is processed in the following way:

"Pretend that the root v_{root} has a parent that is infinitely far away. Treat v_{root}, the parent of v_{root} and the set $\{w(v)|v \in V\}$ as the vertex 'u_1', the vertex 'u_2' and the set '$\{w_1(v)|v \in V\}$' in Lemma 2.4, respectively, and for each element in $R(v_{root})$ do the following two things: (1) treat this element in $R(v_{root})$ as the element '(r, k)'

in Lemma 2.4, and compute the set '$\{w_2(v)|v \in V\}$' as in Lemma 2.4; (2) for every vertex $v \in V$, let $w(v)$ get the value of $w_2(v)$ — namely, $w(v) \leftarrow w_2(v)$."

3. Output the following solution as the solution to the memory allocation problem: for each vertex $v \in V$, assign $w(v)$ to it as its 'memory size'.

□

A pseudo-code of Algorithm 3.1 is presented in Appendix IV for interested readers.

Theorem 2.4 *Algorithm 3.1 is correct.*

Proof: At the beginning of Algorithm 3.1, the value of each $w(v)$ ($v \in V$) is set to be $W_{min}(v)$. Clearly at this moment, $\{w(v)|v \in V\} = \{W_{min}(v)|v \in V\}$ is an 'optimal memory basis.'

Then Algorithm 3.1 processes all the vertices one by one. Each vertex — including the root, in fact — is processed with the following two steps:

Step 1: Modify the value of the set $\{w(v)|v \in V\}$, using the method in Lemma 2.4.

Step 2: Modify the values of $\{R(v)|v \in V\}$ and $\{W_{min}(v)|v \in V\}$, using the method in Lemma 2.5. (Therefore the parameters in the memory allocation problem are changed. Using the terms in Lemma 2.5, the memory allocation problem is changed from an '*old problem*' to a '*new problem*'.)

It's easy to prove by induction that the following two assertions are true:

Assertion 1: Every time a vertex is processed, after step 1, the set $\{w(v)|v \in V\}$ is still an 'optimal memory basis.'

Assertion 2: Every time a vertex is processed, after step 2, the '*new problem*' still has a solution, and any solution of the '*new problem*' is also a solution of the original memory allocation problem.

When all the vertices are processed, all the 'requirement sets' become empty sets, so at this moment the 'optimal memory basis', which is $\{w(v)|v \in V\}$, is also an optimal memory allocation. So Algorithm 1 finds the correct solution.

□

Analysis shows that Algorithm 3.1 has complexity $O(q|V|^3)$, where $|V|$ is the

number of vertices and q is the average cardinality of a requirement set, namely, $q = \frac{1}{|V|}\sum_{v\in V}|R(v)|$.

2.3.3 Memory Allocation for Trees without Upper Bounds on Memory Sizes

In the memory allocation problem, every vertex $v \in V$ has a 'maximum memory size' $W_{max}(v)$. If for every $v \in V$, $W_{max}(v)$ is infinitely large, then we say that there are no upper bounds on memory sizes. For such a special case, some techniques can be used to get a memory allocation algorithm of complexity less than $O(q|V|^3)$, which we will present in this subsection.

The new algorithm is very similar to Algorithm 3.1, except that in this new algorithm, a new notion named '*residual requirement set*' is adopted. The notion is defined as follows. For any vertex $v \in V$, we denote the '*residual requirement set of* v' by $Res(v)$. Say at some moment, each vertex $v \in V$ is temporarily assigned a memory size $w(v)$, and its 'requirement set' is $R(v)$. For every element $(r,k) \in R(v)$, there is a corresponding element $(\bar{r},\bar{k})$ in the 'residual requirement set of v', computed as follows: "$\bar{r} = r$, and $\bar{k} = \max\{k - \sum_{u\in\mathcal{N}(v,r)} w(u), 0\}$." (The meaning of the element $(\bar{r},\bar{k})$ is that the memories of the nodes in $\mathcal{N}(v,r)$ needs to be increased by $\bar{k}$ so that $\sum_{u\in\mathcal{N}(v,r)} w(u)$ will be no less than k.)

Because of the similarity between the new algorithm and Algorithm 3.1, we directly present the pseudo-code of the new algorithm below.

Algorithm 3.2: *Memory Allocation on Tree $G = (V,E)$ without Upper Bounds on Memory Sizes*

1. Label the vertices in V as $v_1, v_2, \cdots, v_{|V|}$ according to the following rule: if v_i is an ancestor of v_j, then $i > j$. Let $w(v_i) \leftarrow W_{min}(v_i)$ for $1 \le i \le |V|$. Let $Res(v_i) \leftarrow \emptyset$ for $1 \le i \le |V|$. For $1 \le i \le |V|$, and for each element $(r,k) \in R(v_i)$, do the following: "if $k - \sum_{v\in\mathcal{N}(v_i,r)} w(v) > 0$, then let $Res(v_i) \leftarrow Res(v_i) \cup \{(r, k - \sum_{v\in\mathcal{N}(v_i,r)} w(v))\}$."

2. For $i = 1$ to $|V| - 1$ do:

{ Let v_P denote the parent of v_i. Let $Q(v_i) \leftarrow Res(v_i)$, and let $x \leftarrow 0$.

While $Q(v_i) \neq \emptyset$ do:

{ Let (r, k) be any element in $Q(v_i)$. If $r < d(v_i, v_P)$, then let $x \leftarrow \max\{x, k\}$ and remove the element (r, k) from the set $Res(v_i)$. Remove the element (r, k) from $Q(v_i)$.

}

Let $w(v_i) \leftarrow w(v_i) + x$.

For $j = i + 1$ to $|V|$, and for every element $(r, k) \in Res(v_j)$, do the following: "if $r \geq d(v_i, v_j)$, then let $(r, k) \leftarrow (r, k - x)$; if $k \leq 0$, then remove the element (r, k) from $Res(v_j)$."

For every element $(r, k) \in Res(v_i)$ do the following: "if $k > x$, then let $Res(v_P) \leftarrow Res(v_P) \cup \{(r - d(v_i, v_P), k - x)\}$."

Let $Res(v_i) \leftarrow \emptyset$.

}

3. Let $x \leftarrow 0$.

While $Res(v_{|V|}) \neq \emptyset$ do:

{ Let (r, k) be any element in $Res(v_{|V|})$. Let $x \leftarrow \max\{x, k\}$. Remove the element (r, k) from $Res(v_{|V|})$.

}

Let $w(v_{|V|}) \leftarrow w(v_{|V|}) + x$.

4. Output $w(v_1)$, $w(v_2)$, $\cdots$, $w(v_{|V|})$ as the solution to the memory allocation problem.

□

The complexity of Algorithm 3.1, which is $O(q|V|^3)$, is dominated by the complexity of updating memory sizes — the memory sizes can be updated up to $O(q|V|^2)$ times, and each time up to $O(|V|)$ memory sizes might change. When there are no upper bounds on the memory sizes, with the help of 'residual requirement sets', each time only one memory size will need to be updated, which has complexity $O(1)$. So the complexity of updating memory sizes is reduced from $O(q|V|^3)$ to $O(q|V|^2)$. Maintaining the 'residual requirement sets' also has a total complexity of $O(q|V|^2)$. So the complexity of Algorithm 3.2 is $O(q|V|^2)$.

2.3.4 Minimizing the Greatest Memory Size of Single Vertices

Minimizing the maximum amount of resource assigned to a single place often has engineering importance in resource assignment problems. In this section, we present an algorithm which finds, among all the solutions to the memory allocation problem, a solution that minimizes the greatest memory size of single vertices. That is, the algorithm finds an 'optimal memory allocation' whose value of $\max_{v \in V} w(v)$ is minimized. The key idea is to use binary search to set an appropriate upper limit on the memory sizes.

Algorithm 3.3 *Memory Allocation on Tree $G = (V, E)$ with Minimized Greatest Memory Size*

1. Use Algorithm 3.1 to find an optimal memory allocation. Say the optimal memory allocation assigns memory size $w_{opt}(v)$ to each vertex $v \in V$. We let $X \leftarrow \sum_{v \in V} w_{opt}(v)$, and let $Y \leftarrow \max_{v \in V} w_{opt}(v)$.

In this algorithm we use 'L' and 'U' to represent the lower limit and the upper limit for the minimized greatest memory size of single nodes; and we use '$M_{min-max}$' to denote the minimized greatest memory size (which is unknown yet). Since every optimal memory allocation's total memory size of all vertices equals X, $M_{min-max}$ must be no less than $\lceil \frac{X}{|V|} \rceil$. Since we've already found an optimal memory allocation whose greatest memory size is Y, $W_{min-max}$ must be no greater than Y. So initially, we let $L \leftarrow \lceil \frac{X}{|V|} \rceil$, and let $U \leftarrow Y$.

2. Use a binary search to find out the exactly value of $W_{min-max}$ in the following way. Let $m \leftarrow \lfloor \frac{L+U}{2} \rfloor$. Use Algorithm 3.1 to find an optimal memory allocation for such a memory allocation problem: everything in this problem is the same as in the original memory allocation problem, except that in this problem, the 'maximum memory size' of each vertex $v \in V$ is $\min\{m, W_{min}(v)\}$ instead of $W_{min}(v)$. If this problem has a solution and in the solution the total memory size equals X, then it means that $M_{min-max}$ is no greater than m, so we let $U \leftarrow m$; otherwise, it means that $M_{min-max}$ is greater than m, so we let $L \leftarrow m + 1$. Repeat this procedure until

L and U become equal.

Now we have $W_{min-max} = L = U$. Use Algorithm 3.1 to find a solution to the following memory allocation problem: in the problem everything is the same as in the original memory allocation problem, except that in this problem, the 'maximum memory size' of each vertex $v \in V$ is $\min\{W_{min-max}, W_{min}(v)\}$ instead of $W_{min}(v)$. The solution found is the solution whose greatest memory size is minimized.

□

The binary search has $O(\log(Y - \lceil \frac{X}{|V|} \rceil))$ steps; in every step Algorithm 3.1 is executed once. So Algorithm 3.3 has complexity $O(q|V|^3 \log(Y - \frac{X}{|V|}))$. (To see how large Y can be, note that Y is never greater than $\max_{v \in V, 1 \leq i \leq n_v} k_i(v)$ or $\max_{v \in V} W_{max}(v)$.)

2.4 Summary

An optimal solution to the data placement problem can be found through the following two steps when the graph $G = (V, E)$ is a tree: firstly, use a memory allocation algorithm in Section 2.2.3 to determine the number of codeword components to place on each vertex; next, use a maximum interleaving algorithm (or a K-diversity interleaving algorithm for $K \geq \max_{u \in V, 1 \leq i \leq n_u} k_i(u)$) to determine which set of codeword components to place on each vertex.

The maximum-interleaving property of trees has three important implications for the data placement practice using erasure-correcting codes. Firstly, it makes memory allocation a problem that is separate from the mapping from codeword components to vertices, thus providing us with the maximum freedom in minimizing the total amount of data to store in the network. Secondly, since by using maximum interleaving, for every vertex the n distinct codeword components can be placed as close as possible around it, that enables us to minimize the file-retrieving delays for all vertices simultaneously. Thirdly, maximum interleaving reduces the requirement on the codeword length, n, to be the minimum. That is because with maximum-interleaving, n only needs to be as large as the maximum number of different codeword components that a vertex would actually ask for.

2.5 Appendix I: Complexity Analysis of Algorithm 2.1

In this appendix we analyze the complexity of Algorithm 2.1.

Complexity Analysis: To implement Algorithm 2.1, it's first necessary to label all the color-slots as s_1, s_2, $\cdots$, $s_{\mathbb{W}}$ 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, $\cdots$, 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 $\{s_1, s_2, \cdots, s_{i-1}\}$ that are at distance less than r_{min} from s_i. (Here r_{min}, as in Algorithm 2.1, is defined as the smallest value of r such that in the set $\{s_1, s_2, \cdots, s_{i-1}\}$, 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 2.1 is found to be $O(|V|^2 + \mathbb{W}|V| + \mathbb{W}K)$.

2.6 Appendix II: Proving the Generality of Algorithm 2.1

In this appendix we present the proof of Theorem 2.2.

Proof: By Theorem 2.1, the set of possible outputs of Algorithm 2.1 is a subset

of the set of K-diversity interleavings. So the only thing left to prove is that every K-diversity interleaving is a possible output of Algorithm 2.1, even if the root of the tree is fixed.

Assume the root of the tree is fixed. Assume a fixed K-diversity 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-diversity interleaving must have assigned $\mathbb{W}$ distinct colors to the $\mathbb{W}$ color-slots, which is clearly a possible output of Algorithm 2.1. 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 2.1 can label as s_1, s_2, $\cdots$, s_i and assign the same colors to as the given K-diversity 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. It is simple to see that all the color-slots in A_0 are assigned distinct colors in the given K-diversity 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-diversity interleaving. Clearly it's feasible for Algorithm 2.1 to label the color-slots in $A_0 \cup B_0$ as s_1, s_2, $\cdots$, s_K, and assign the same colors to them as the given K-diversity 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 2.1 can label as s_1, s_2, $\cdots$, s_i and assign the same colors to as the given K-diversity interleaving does. We will prove that this induction assumption is also true if i is replaced by $i + 1$.

Since Algorithm 2.1 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 $c_1, c_2, \cdots, c_{|H|}$. 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-diversity interleaving. Then it's simple to see that it is feasible for Algorithm 2.1 to label c_{j_1} as the color-slot s_{i+1} and assign s_{i+1} the same color as the given K-diversity interleaving does, after labelling the color-slots in C as $s_1, s_2, \cdots, s_i$ and assigning the same colors to them as the given K-diversity 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-diversity 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-diversity interleaving; for any color-slot $z \in C$ that is assigned the same as c_{j_1} in the given K-diversity 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 2.2, the color-slots in J are assigned at least K distinct colors in the given K-diversity 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-diversity 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 2.1 to label c_{j_2} as the color-slot s_{i+1} and assign s_{i+1} the same color as the given K-diversity 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_2}' 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-diversity 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 $d(\mathcal{C}(u_2, v_2), u_2) = \frac{d(v_2,u_2)}{2} > \frac{d(u_1,v_1)}{2} = d(\mathcal{C}(u_1, v_v), u_2)$, 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} $\cdots$ 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 2.1 to label c_{j_x} as the color-slot s_{i+1} and assign s_{i+1} the same color as the given K-diversity interleaving does, after labelling the color-slots in C as s_1, s_2, $\cdots$, s_i and assigning the same colors to them as the given K-diversity interleaving does.

So by now it has been shown that in all cases, some color-slot $c_{j_x} \in H$ can be found such that it's feasible for Algorithm 2.1 to label the color-slots in $C \cup \{c_{j_x}\}$ as s_1, s_2, $\cdots$, s_{i+1}, and assign the same colors to them as the given K-diversity 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 2.1 can assign the same colors to all the $\mathbb{W}$ color-slots as the arbitrarily given K-diversity interleaving does. So any K-diversity interleaving is a possible output of Algorithm 2.1, even if the root of the tree is fixed. Therefore this theorem is proved.

□

2.7 Appendix III: Complexity Analysis of Algorithm 2.2

In this appendix we analyze the time complexity of Algorithm 2.2, and show that Algorithm 2.2 is more efficient than Algorithm 2.1.

Complexity Analysis: Algorithm 2.2 first labels the vertices as v_1, v_2, $\cdots$, $v_{|V|}$ according to the vertices' distance to the root, and labels the color-slots as s_1, s_2, $\cdots$, $s_{\mathbb{W}}$ 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, $\cdots$, 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 2.2 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 2.2.) 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 $\{s_{K+1}, s_{K+2}, \cdots, s_{\mathbb{W}}\}$, 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 $\{s_{K+1}, s_{K+2}, \cdots, s_{\mathbb{W}}\}$. 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, $\cdots$, 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 $(v_{k_1}, v_{k_2}, \cdots, v_{k_{i-1}})$. (Here $(k_1, k_2, \cdots, k_{i-1})$ is a permutation of $(1, 2, \cdots, i-1)$.) Find four variables a, b, c and d that satisfy the following conditions: $\sum_{1 \le j \le a-1} w(v_{k_j}) + b = K - w(v_i) + 1$, $\sum_{1 \le j \le c-1} w(v_{k_j}) + d = K$, $1 \le a \le i-1$, $1 \le b \le w(v_{k_a})$, $1 \le c \le i-1$, and $1 \le d \le w(v_{k_c})$.

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

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} — $\hat{s}_{k_a,b}, \hat{s}_{k_a,b+1}, \cdots, \hat{s}_{k_a,w(v_{k_a})}$, and all the color-slots of vertices $v_{k_a+1}, v_{k_a+2}, \cdots, v_{k_c-1}$, and these d color-slots of vertex v_{k_c} — $\hat{s}_{k_c,1}, \hat{s}_{k_c,2}, \cdots, \hat{s}_{k_c,d}$; (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 ($1 \le t \le K$) 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, $\cdots$, v_{i-1} into $(v_{k_1}, v_{k_2}, \cdots, v_{k_{i-1}})$ 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 $v_{K+1}, v_{K+2}, \cdots, v_{\mathbb{W}}$, without excluding the complexity of any computation, is $O(|V|^2 + \mathbb{W})$.

By combining the above results, the time complexity of Algorithm 2.2 is found to be $O(|V|^2 + \mathbb{W})$.

Comparison between Algorithm 2.1 and Algorithm 2.2: There are two significant differences between the implementation of Algorithm 2.1 and Algorithm 2.2.

Difference 1: In Algorithm 2.1, 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 2.2, 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 2.1, Algorithm 2.2 reduces the complexity by a factor of $w(v_i)$.

Difference 2: In Algorithm 2.1, 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 2.2, the counterpart operation is simply to insert the color-slot into some set C_t. So for the above computation, compared to Algorithm 2.1, Algorithm 2.2 reduces the complexity by a factor which is approximately K.

The operations in Algorithm 2.1 and the operations in Algorithm 2.2, except for the ones specified in 'Difference 1' and 'Difference 2', are very similar and have the same complexity. Therefore, Algorithm 2.2 is more efficient than Algorithm 2.1.

2.8 Appendix IV: Pseudo-Code of Algorithm 3.1

In this appendix we present the pseudo-code of Algorithm 3.1.

Pseudo-code of Algorithm 3.1: *Memory Allocation on Tree* $G = (V, E)$

1. Label the vertices in V as v_1, v_2, $\cdots$, $v_{|V|}$ according to the following rule: if v_i is an ancestor of v_j, then $i > j$. Let $w(v_i) \leftarrow W_{min}(v_i)$ for $1 \leq i \leq |V|$.

2. For $i = 1$ to $|V| - 1$ do:

{ Let v_P denote the parent of v_i. Let $\tilde{R}(v_i) \leftarrow R(v_i)$.

While $\tilde{R}(v_i) \neq \emptyset$ do:

{ Let (r, k) be any element in $\tilde{R}(v_i)$. Define S_1 as $S_1 = \mathcal{N}(v_P, r - d(v_i, v_P))$, and define S_2 as $S_2 = \mathcal{N}(v_i, r) - S_1$. Update the elements in $\{w(v) | v \in V\}$ in the following way (step 1 and step 2):

Step 1: Let

$$X \leftarrow \max\{0, k - \sum_{v \in S_1} W_{max}(v) - \sum_{v \in S_2} w(v)\}$$

, and let $C \leftarrow S_2$.

Step 2: Let u_0 be the vertex in C that is the closest to v_i—namely, $d(u_0, v_i) = \min_{u \in C} d(u, v_i)$. Let $Temp \leftarrow \min\{W_{max}(u_0), w(u_0) + X\}$. Let $X \leftarrow X - (Temp - w(u_0))$, let $w(u_0) \leftarrow Temp$, and let $C \leftarrow C - \{u_0\}$. Repeat Step 2 until X equals 0.

Remove the element (r, k) from $\tilde{R}(v_i)$.

}

While $R(v_i) \neq \emptyset$ do:

{ Let (r, k) be any element in $R(v_i)$. If $\sum_{u \in \mathcal{N}(v_i, r)} w(u) < k$, then add an element $(r - d(v_i, v_P), k - \sum_{u \in \mathcal{N}(v_i, r) - \mathcal{N}(v_P, r - d(v_i, v_P))} w(u))$ to the set $R(v_P)$. Remove the element (r, k) from $R(v_i)$.

}

}

3. While $R(v_{|V|}) \neq \emptyset$ do:

{ Let (r, k) be any element in $R(v_{|V|})$. Update the elements in $\{w(v) | v \in V\}$ in the following way (step 1 and step 2):

Step 1: Let

$$X \leftarrow \max\{0, k - \sum_{u \in \mathcal{N}(v_{|V|}, r)} w(v)\}$$

, and let $C \leftarrow V$.

Step 2: Let u_0 be the vertex in C that is the closest to $v_{|V|}$—namely, $d(u_0, v_{|V|}) = \min_{u \in C} d(u, v_{|V|})$. Let $Temp \leftarrow \min\{W_{max}(u_0), w(u_0) + X\}$. Let $X \leftarrow X - (Temp - w(u_0))$, let $w(u_0) \leftarrow Temp$, and let $C \leftarrow C - \{u_0\}$. Repeat Step 2 until X equals 0.

Remove the element (r, k) from $R(v_{|V|})$.

}

Output $w(v_1)$, $w(v_2)$, $\cdots$, $w(v_{|V|})$ as the solution to the memory allocation problem.

□

Chapter 3

Optimal t-Interleaving on Tori

3.1 Introduction

Interleaving is an important technique used for network data storage and error burst correction. A most common example is the interleaving of n codewords in the form of '$1-2-3-\cdots-n-1-2-3-\cdots-n-\cdots\cdots$' for combatting one-dimensional error bursts in communication channels [60]. The concept of one-dimensional error burst was generalized to high dimensions by Blaum, Bruck and Vardy in [11], where an error burst of size t is a set of errors confined to a connected subgraph with t vertices in a multi-dimensional linear array. Accordingly, the concept of t-interleaving was defined in [11], which is a scheme to label the vertices of a multi-dimensional linear array with integers such that any subgraph with t vertices in the array are labelled by t distinct integers. t-interleaving schemes on two- and three-dimensional linear arrays were presented in [11], with applications in combatting error bursts in holographic storage systems and optical recording systems. Subsequent work on t-interleaving includes [80], where t-interleaving on circulant graphs with two offsets was studied. Two-dimensional interleaving with repetitions was studied by Etzion and Vardy in [21], where integers were interleaved on a two-dimensional mesh (linear array or its variation) such that in any connected subgraph with t vertices, every integer appears at most r times. Here t and r are given parameters, and the concept of interleaving with repetitions defined in [21] is a generalization of t-interleaving. More work on interleaving with repetitions includes [62] and [75]. Interleaving schemes

on two-dimensional linear arrays achieving the Reiger bound was studied by Abdel-Ghaffar in [1], where error bursts of both rectangular shapes and arbitrary connected shapes were concerned. More examples of interleaving for coping with error bursts include [5], where the error burst is of a 'circular' type, and [15], where linear binary array codes that can correct three-dimensional error bursts were designed based on interleaving. As to interleaving schemes for network data storage, in Chapter II, we have presented a *Maximum-Interleaving* algorithm that interleaves N integers on a tree such that for every point of the tree (including a vertex or a point on an edge), the smallest sphere centered at the point that contains N integers contains all the N distinct integers. That algorithm is useful for distributed data storage in hierarchical or peer-to-peer networks that minimizes data retrieval delay. And in Chapter III, a scheme called *Multi-Cluster Interleaving* will be studied, which is a scheme to interleave integers on a path or cycle such that any m disjoint intervals of length L in the path or cycle together contain at least K distinct integers, where $K > L$. Multi-cluster interleaving can be used for data storage on array-networks, ring-networks or disks where data are accessed through multiple access points.

In this chapter, we study t-interleaving on two-dimensional tori. First we present the necessary definitions. The notion of 't-interleaving' was originally defined in [11] for arrays. We generalize its definition to be for general graphs straightforwardly.

Definition 3.1 *Let G be a graph. We say that G is* interleaved *(or there is an* interleaving *on G) if every vertex of G is labelled by one integer. We say that G is* t-interleaved *(or there is a* t-interleaving *on G) if for every connected subgraph of G that contains t or fewer vertices, the integers on it are all distinct.* □

Definition 3.2 *An $l_1 \times l_2$ torus is a graph containing $l_1 l_2$ vertices and $2l_1 l_2$ edges. We denote its vertices by '(i, j)' for $0 \leq i \leq l_1 - 1$ and $0 \leq j \leq l_2 - 1$, in the way shown in the figure below:*

$(0,0)$	$(0,1)$	$\cdots$	$(0,l_2-1)$
$(1,0)$	$(1,1)$	$\cdots$	$(1,l_2-1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(l_1-1,0)$	$(l_1-1,1)$	$\cdots$	(l_1-1,l_2-1)

Each vertex (i,j) *is incident to four edges, which connect it to its four neighbors* $((i-1) \bmod l_1, j)$, $((i+1) \bmod l_1, j)$, $(i,(j-1) \bmod l_2)$ *and* $(i,(j+1) \bmod l_2)$. □

Definition 3.3 *Given a t-interleaved torus G, the number of distinct integers used to label the vertices of G is called the* degree *of this given t-interleaving scheme. The minimum degree of all the possible t-interleaving schemes for G is called the t-*interleaving number *of G. A t-interleaving on a torus whose degree equals the torus' t-interleaving number is called an* optimal t-interleaving. □

An $l_1 \times l_2$ torus is two-dimensional.

Example 3.1 *The following* 5×5 *torus is 3-interleaved with the degree of 6.*

0	*3*	*1*	*4*	*2*
1	*4*	*2*	*0*	*3*
2	*0*	*3*	*1*	*5*
3	*1*	*5*	*2*	*0*
4	*2*	*0*	*3*	*1*

If we replace the two integers '5' with '4', we will get a 3-interleaving with the degree of 5. Observe the vertex $(1,1)$ *and its four neighbors* $(0,1)$, $(2,1)$, $(1,0)$ *and* $(1,2)$, *and we can see that any two of them are contained in a subgraph of size at most 3 — therefore any 3-interleaving scheme has to label those 5 vertices with 5 distinct integers. So the 3-interleaving number of this torus actually equals 5.* □

Important applications of t-interleaving on tori include both distributed data storage and error-burst correction. Torus has traditionally been a popular network structure for parallel machines, such as the CRAY [66], the iWarp [13], the Tera parallel

computer [84] and the Mosaic [76]. Its simple and regular structure makes it convenient for multi-processor computing and message transmission. The usage of torus networks in wearable computing [57] and ambient intelligent systems [50] looks also promising, where massive interconnected micro processors, memories and sensors are embedded in fabrics of clothes, carpets, etc. We briefly explain the two applications of t-interleaving in these torus networks below.

- *Distributed data storage.* Let G be a torus network, whose vertices are processors with memory. Say a file F is to be distributively stored in G; and there is the requirement that every processor should be able to reconstruct F by accessing the data stored within the distance of r units, where a 'unit' is the distance between any two adjacent processors. We use t-interleaving to solve this problem. Let B_r denote the number of vertices in G that are within the distance of r units from a given vertex (including the given vertex itself). And let n_0 denote the $(2r+1)$-interleaving number of G. Select an erasure-correcting code of length n which can tolerate at least $n - B_r$ erasures, where $n \geq n_0$. Encode F with the code to get a codeword; then see the n components of the codeword as n distinct integers, and assign them to the processors according to a $(2r+1)$-interleaving scheme for G. With a $(2r+1)$-interleaving, given any vertex v, the distance between any two vertices within the distance of r units from v is at most $2r$, so they must be labelled by distinct integers. So every processor can find B_r distinct codeword components within the distance of r units and decode them to recover the file F, satisfying the requirement. Such a data storage method balances the memory usage for all processors well. If an MDS (maximum distance separable) code is used, the total memory of G as well as the maximum single-processor memory used for storing F can be minimized over all the possible methods.

- *Error-burst correction.* For wearable computing systems and ambient intelligent systems, the networks embedded in fabrics are prone to physical damage, such as tearing or punching. It is necessary to achieve reliability through re-

dundancy [57] for such systems. We can see physical damage such as tearing or punching as error-bursts, and use t-interleaving to reliably store files. Here, as in traditional interleaving for error-burst correction, vertices labelled by the same integer store components of the same codeword (which corresponds to a file). Different integers represent different codewords. With a t-interleaving scheme, if a codeword can correct e errors, then any e error-bursts of size up to t can be corrected.

Besides the above applications, t-interleaving on tori is also closely related to a research topic in coding theory called *Lee metric codes* [4], [3], [7], [8], [10], [12], [26], [27], [28], [29], [32], [48], [52], [71]. In a t-interleaved n-dimensional torus, every set of vertices labelled by the same integer is a Lee metric code of length n whose minimum distance is t; and the set of Lee metric codes corresponding to different integers partition the whole code space. Furthermore, if a torus admits a *perfect Lee metric code* of covering radius r, then the torus' $(2r+1)$-interleaving number is no greater than that of any reasonably large torus. (Here a reasonably large torus is defined to be a torus whose size in each dimension is at least $2r+1$.)

A fundamental question on the problem of t-interleaving on tori is: for each integer t, how does the t-interleaving number of an $l_1 \times l_2$ torus depend on the values of l_1 and l_2, and how to construct optimal t-interleaving? To the best of our knowledge, the only related results were covered in [11]. [11] presented, for two-dimensional linear array, one optimal t-interleaving construction for odd t and two optimal t-interleaving constructions for even t, all based on lattice interleavers. Those three constructions all produce interleaving of periodic patterns; and if they are applied to tori, they can, respectively, optimally t-interleave an $l_1 \times l_2$ torus if (1) t is odd, $\frac{t^2+1}{2}|l_1$ and $\frac{t^2+1}{2}|l_2$, or if (2) t is even, $\frac{t^2}{2}|l_1$ and $\frac{t^2}{2}|l_2$, or if (3) t is even, $t|l_1$ and $t|l_2$. However, tori whose sizes satisfy one of those three conditions are very special. And as we will show later, the constructions in [11] are not the only optimal ones.

In this chapter, we address the above fundamental question, and provide a general picture of the answers.

Our main results include:

- Let $|S_t| = \frac{t^2+1}{2}$ if t is odd, and let $|S_t| = \frac{t^2}{2}$ if t is even. $|S_t|$ is a lower bound for the t-interleaving number of any reasonably large $l_1 \times l_2$ torus (which means $l_1 \geq t$ and $l_2 \geq t$). For a reasonably large torus, we say that it can be *perfectly t-interleaved* if its t-interleaving number equals $|S_t|$. We prove that a reasonably large $l_1 \times l_2$ torus can be perfectly t-interleaved if and only if the following condition is satisfied: $|S_t|$ divides both l_1 and l_2 if t is odd, and t divides both l_1 and l_2 if t is even. We reveal the very close relationship between perfect t-interleaving and perfect sphere packing, and present the *complete* set of perfect sphere packing constructions. Based on that, we get a set of efficient perfect t-interleaving constructions, which includes the lattice interleaver (the interleaving method used in [11]) as a special case.

- Define a *post-threshold size* (for a given parameter t) to be a pair (θ_1, θ_2) such that whenever $l_1 \geq \theta_1$ and $l_2 \geq \theta_2$, the t-interleaving number of an $l_1 \times l_2$ torus is either $|S_t| + 1$ or $|S_t|$. We prove that such post-threshold sizes exist for every t. The set of post-threshold sizes we found are shown in Theorem 3.10 and Theorem 3.11. We present optimal t-interleaving constructions for tori whose sizes exceed the found post-threshold sizes. (And we comment that those constructions, as a general interleaving method, can also be used to optimally t-interleave tori of many other sizes.)

- We study upper bounds for t-interleaving numbers. Every $l_1 \times l_2$ torus' t-interleaving number is $|S_t| + O(t^2)$. And that upper bound is tight, even if $l_1 \to +\infty$ or $l_2 \to +\infty$. When both l_1 and l_2 are of the order $\Omega(t^2)$, the t-interleaving number of an $l_1 \times l_2$ torus is $|S_t| + O(t)$.

The results can be illustrated qualitatively as Fig. 3.1. (The figure is not quantitative. The coordinates of points, such as the shape of the curve, are not exact.) Fig. 3.1 shows for any given 't', how the $l_1 \times l_2$ tori can be divided into different classes based on their t-interleaving numbers.

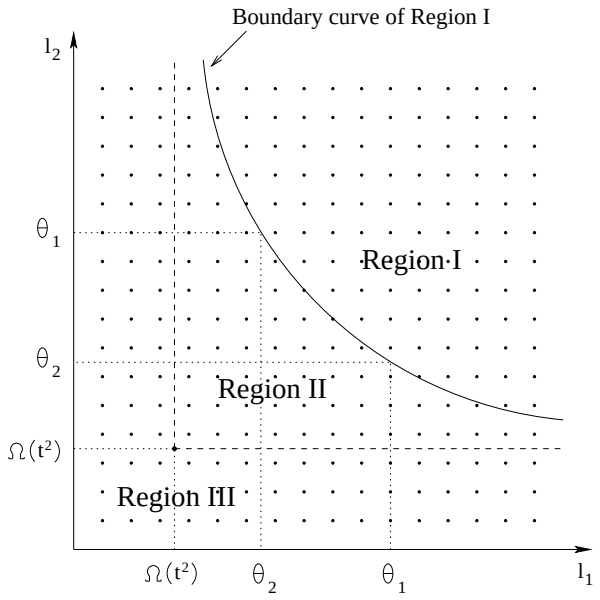


Figure 3.1: A qualitative illustration of the t-interleaving numbers.

The uniform lattice of dots in Fig. 3.1 are the sizes of all the reasonably large tori that can be perfectly t-interleaved. The region labelled as '*Region I* ' consists of all the *post-threshold sizes*. The boundary curve of Region I is non-increasing, and symmetric with respect to the line $l_2 = l_1$. (So if the point (θ_1, θ_2) is on the boundary curve, then so is (θ_2, θ_1).) We note that the area of Region I is $(100 - \delta)\%$ of the total area of the figure with δ approaching 0; and we know the exact t-interleaving number of every torus in this region — $|S_t|$ if it is one of the lattice dots, and $|S_t| + 1$ otherwise. The most important contribution of this paper is to prove the existence of Region I, and present the corresponding optimal interleaving constructions. Region II is the region where $l_1 = \Omega(t^2)$ and $l_2 = \Omega(t^2)$, in which the tori's t-interleaving numbers are upper-bounded by $|S_t| + O(t)$. Region III includes every torus, where the t-interleaving number is upper-bounded by $|S_t| + O(t^2)$. That upper bound for Region III is tight, even if l_1 or l_2 approaches $+\infty$. (So increasing a torus' size in only one dimension does not help reduce the t-interleaving number very effectively in general.)

The engineering importance of Region I can be shown, as an example, with the

t-interleaving's application in distributed data storage. It means for a large torus network (which falls in Region I), we can simply split the file into $|S_t|$ segments, then add one parity-check segment which is the exclusive-OR of the $|S_t|$ segments. The data-storage scheme using such a simple erasure-correcting code can be implemented very efficiently.

The rest of the chapter is organized as follows. In Section 3.2, we show the necessary and sufficient conditions for tori that can be perfectly t-interleaved, and present perfect t-interleaving constructions based on perfect sphere packing. In Section 3.3, we present a t-interleaving method, with which we can t-interleave large tori with a degree within one of the optimal. In Section 3.4, we improve upon the t-interleaving method shown in Section 3.3, and present optimal t-interleaving constructions for tori whose sizes are large in both dimensions. As a parallel result, the existence of Region I is proved. In Section 3.5, we prove some general bounds for the t-interleaving numbers. In Section 3.6, we provide some brief discussions.

3.2 Perfect t-Interleaving

In this section, we formally define the concept of *perfect t-interleaving*, and reveal its close relationship with perfect sphere packing. We show the necessary and sufficient conditions for tori that can be perfectly t-interleaved. After presenting the complete set of perfect sphere packing constructions, we present efficient perfect t-interleaving constructions based on them. The interleaving constructions cover previously known t-interleaving methods as special cases; and they prove that lattice interleavers are not the only method for perfect t-interleaving.

3.2.1 Perfect t-Interleaving and Sphere Packing

Definition 3.4 *The* Lee distance *between two vertices in a torus is the number of edges in the shortest path connecting those two vertices. For two vertices in an $l_1 \times l_2$ torus G, (a_1, b_1) and (a_2, b_2), the* Lee distance *between them is denoted by $d((a_1, b_1), (a_2, b_2))$. (Therefore, $d((a_1, b_1), (a_2, b_2)) = \min\{(a_1 - a_2) \bmod l_1, (a_2 -$

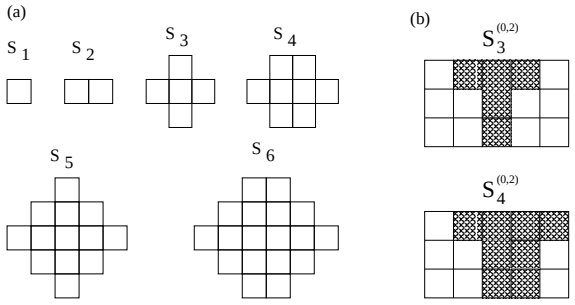


Figure 3.2: Examples of the sphere S_t.

$a_1) \bmod l_1\} + \min\{(b_1 - b_2) \bmod l_2, (b_2 - b_1) \bmod l_2\}$.*) Occasionally, in order to emphasize that the two vertices are in G, we also denote it by* $d_G((a_1, b_1), (a_2, b_2))$. □

Clearly, an interleaving on a torus is a t-interleaving if and only if the Lee distance between any two vertices labelled by the same integer is at least t.

Definition 3.5 *Let G be an $l_1 \times l_2$ torus where $l_1 \geq t$ and $l_2 \geq t$, and let (a, b) be a vertex in G. When t is odd, the* sphere centered at (a, b), $S_t^{(a,b)}$, *is defined to be the set of vertices whose Lee distance to (a, b) is less than or equal to $\frac{t-1}{2}$. When t is even, the* sphere left-centered at (a, b), $S_t^{(a,b)}$, *is defined to be the set of vertices whose Lee distance to either (a, b) or $(a, (b+1) \bmod l_2)$ is less than or equal to $\frac{t}{2} - 1$. (a, b) is called the* center *of $S_t^{(a,b)}$ if t is odd; and (a, b) is called the* left-center *of $S_t^{(a,b)}$ if t is even. If we do not care where the sphere is centered or left-centered, then the sphere is simply denoted by S_t. The number of vertices in the sphere is denoted by $|S_t|$.* □

Example 3.2 *Fig. 3.2 (a) shows the spheres S_1 to S_6. Fig. 3.2 (b) shows two spheres, $S_3(0, 2)$ and $S_4(0, 2)$, in a 3×5 torus.* □

The sphere S_t was originally defined in [11], where it was also shown that $|S_t| = \frac{t^2+1}{2}$ if t is odd, and $|S_t| = \frac{t^2}{2}$ if t is even. For any $l_1 \times l_2$ torus where $l_1 \geq t$ and $l_2 \geq t$, its t-interleaving number is at least $|S_t|$. That is because such a torus contains a complete sphere S_t, and the Lee distance between any two vertices in S_t is less than

t — so any t-interleaving needs to use $|S_t|$ distinct integers to label the vertices in S_t. This lower bound for t-interleaving numbers, $|S_t|$, is called the *sphere packing lower bound*. The relationship between this bound and sphere packing will become clearer soon.

Definition 3.6 *Let G be an $l_1 \times l_2$ torus, where $l_1 \geq t$ and $l_2 \geq t$. If the t-interleaving number of G equals the sphere packing lower bound $|S_t|$, then we say that G can be* perfectly t-interleaved. *A t-interleaving on G that uses exactly $|S_t|$ distinct integers will be called a* perfect t-interleaving. □

Definition 3.7 *A torus G is said to have a* perfect packing of spheres S_t *if spheres S_t are packed in G such that every vertex of G belongs to one sphere, and no two spheres share any common vertex.* □

Lemma 3.1 *(1) Let t be an odd positive integer. An interleaving on an $l_1 \times l_2$ torus ($l_1 \geq t$, $l_2 \geq t$) is a t-interleaving if and only if for any two vertices (a_1, b_1) and (a_2, b_2) that are labelled by the same integer, the two spheres centered at them, $S_t^{(a_1,b_1)}$ and $S_t^{(a_2,b_2)}$, do not share any common vertex.*

(2) Let t be an even positive integer. An interleaving on an $l_1 \times l_2$ torus ($l_1 \geq t-1$, $l_2 \geq t$) is a t-interleaving if and only if for any two vertices (a_1, b_1) and (a_2, b_2) that are labelled by the same integer, the two spheres with them as left-centers, $S_t^{(a_1,b_1)}$ and $S_t^{(a_2,b_2)}$, do not share any common vertex and what's more, $b_1 \neq b_2$ or $(a_1 - a_2) \neq \pm(t-1) \bmod l_1$.

Proof: (1) Let t be odd. Both $S_t^{(a_1,b_1)}$ and $S_t^{(a_2,b_2)}$ are classic spheres with radius $\frac{t-1}{2}$. If the interleaving is a t-interleaving, then the Lee distance between (a_1, b_1) and (a_2, b_2) is at least $t = 2 \cdot \frac{t-1}{2} + 1$, so $S_t^{(a_1,b_1)}$ and $S_t^{(a_2,b_2)}$ must have no intersection. The converse is clearly also true.

(2) Let t be even. We consider two cases — $b_1 = b_2$ and $b_1 \neq b_2$.

First consider the case '$b_1 = b_2$'. In this case, $S_t^{(a_1,b_1)}$ and $S_t^{(a_2,b_2)}$ have no intersection if and only if $d((a_1, b_1), (a_2, b_2)) \geq 2 \cdot (\frac{t}{2} - 1) + 1 = t - 1$. And $d((a_1, b_1), (a_2, b_2)) =$

$t-1$ if and only if $(a_1 - a_2) \equiv \pm(t-1) \bmod l_1$. So the Lee distance between (a_1, b_1) and (a_2, b_2) is at least t if and only if $S_t^{(a_1,b_1)}$ and $S_t^{(a_2,b_2)}$ have no intersection and $(a_1 - a_2) \neq \pm(t-1) \bmod l_1$, which is the conclusion we want.

Now consider the case '$b_1 \neq b_2$'. In this case, the Lee distance between (a_1, b_1) and (a_2, b_2) is at least $t \Longleftrightarrow$ both the Lee distance between $(a_1, (b_1+1) \bmod l_2)$ and (a_2, b_2) and the Lee distance between $(a_2, (b_2+1) \bmod l_2)$ and (a_1, b_1) are at least $t-1$ $\Longleftrightarrow S_{t-1}^{(a_1,(b_1+1) \bmod l_2)}$ does not intersect $S_{t-1}^{(a_2,b_2)}$ and $S_{t-1}^{(a_2,(b_2+1) \bmod l_2)}$ does not intersect $S_{t-1}^{(a_1,b_1)} \Longleftrightarrow S_t^{(a_1,b_1)}$ and $S_t^{(a_2,b_2)}$ have no intersection. (Note that $S_t^{(a_1,b_1)}$ is the union of $S_{t-1}^{(a_1,b_1)}$ and $S_{t-1}^{(a_1,(b_1+1) \bmod l_2)}$, and $S_t^{(a_2,b_2)}$ is the union of $S_{t-1}^{(a_2,b_2)}$ and $S_{t-1}^{(a_2,(b_2+1) \bmod l_2)}$.) So we get the conclusion we want.

□

Theorem 3.1 *When $t \neq 2$, an interleaving on an $l_1 \times l_2$ torus ($l_1 \geq t$, $l_2 \geq t$) is a perfect t-interleaving if and only if for any integer, the spheres S_t centered or left-centered at the vertices labelled by that integer form a perfect sphere packing in the torus.*

When $t = 2$, if an interleaving on an $l_1 \times l_2$ torus ($l_1 \geq t$, $l_2 \geq t$) is a perfect t-interleaving, then for any integer, the spheres S_t left-centered at the vertices labelled by that integer form a perfect sphere packing in the torus.

Proof: We used I to denote the set of distinct integers used by the interleaving on the torus. For any integer $i \in I$, let N_i denote the number of vertices labelled by i.

Firstly, we prove one direction. Assume the interleaving on the $l_1 \times l_2$ torus ($l_1 \geq t$, $l_2 \geq t$) is a perfect t-interleaving. Then $|I| = |S_t|$. By Lemma 3.1, for any $i \in I$, the spheres S_t centered or left-centered at vertices labelled by i do not overlap. By counting the number of vertices in the torus and in each sphere S_t, we get that $N_i \leq \frac{l_1 l_2}{|S_t|}$ for any $i \in I$. Since $\sum_{i \in I} N_i = l_1 l_2$, we get that $N_i = \frac{l_1 l_2}{|S_t|}$ for any $i \in I$. So for any integer $i \in I$, the spheres S_t centered or left-centered at the vertices labelled by i form a perfect sphere packing in the torus.

Next, we prove the other direction. Assume $t \neq 2$, and for any integer, the spheres S_t centered or left-centered at the vertices labelled by that integer form a perfect sphere packing in the torus. Then $N_i = \frac{l_1 l_2}{|S_t|}$ for any $i \in I$. Since $\sum_{i \in I} N_i = l_1 l_2$, we find that $|I|$, the number of distinct integers used by the interleaving, equals $|S_t|$. What is left is to prove that the interleaving is a t-interleaving. From Lemma 3.1, we can see that the interleaving would not be a t-interleaving only if the following situation becomes true: t is even, and there exist two vertices — (a_1, b_1) and (a_2, b_2) — labelled by the same integer such that $b_1 = b_2$ and $(a_1 - a_2) \equiv \delta(t-1) \bmod l_1$, where $\delta = 1$ or -1. We will show such a situation cannot happen.

Suppose that situation happens. WLOG, we assume $(a_1 - a_2) \equiv (t-1) \bmod l_1$. When t is even and $t \neq 2$, it is straightforward to verify that the following four vertices — $(a_1 - (\frac{t}{2} - 1) \bmod l_1, b_1)$, $(a_2 + (\frac{t}{2} - 1) \bmod l_1, b_1)$, $(a_1 - (\frac{t}{2} - 2) \bmod l_1, b_1 - 1 \bmod l_2)$, $(a_2 + (\frac{t}{2} - 2) \bmod l_1, b_1 - 1 \bmod l_2)$ — are contained in either $S_t^{(a_1,b_1)}$ or $S_t^{(a_2,b_2)}$, while the following two vertices — $(a_1 - (\frac{t}{2} - 1) \bmod l_1, b_1 - 1 \bmod l_2)$ and $(a_2 + (\frac{t}{2} - 1) \bmod l_1, b_1 - 1 \bmod l_2)$ — are neither contained in $S_t^{(a_1,b_1)}$ nor in $S_t^{(a_2,b_2)}$. The two vertices, $(a_1 - (\frac{t}{2} - 1) \bmod l_1, b_1 - 1 \bmod l_2)$ and $(a_2 + (\frac{t}{2} - 1) \bmod l_1, b_1 - 1 \bmod l_2)$, cannot both be contained in some spheres S_t that are left-centered at vertices labelled by the same integer which labels (a_1, b_1) and (a_2, b_2), because they are vertically adjacent, and the vertices directly above them, below them or to the right of them are all contained in two spheres that do not contain them. (Observe the shape of a sphere.) That contradicts that fact that all the spheres S_t left-centered at the vertices labelled by the integer which labels (a_1, b_1) form a perfect sphere packing in the torus. So the assumed situation cannot happen. By summarizing the above results, we see that the interleaving must be a perfect t-interleaving.

□

Theorem 3.2 *When $t \neq 2$, an $l_1 \times l_2$ torus ($l_1 \geq t$, $l_2 \geq t$) can be perfectly t-interleaved if and only if the spheres S_t can be perfectly packed in it.*

When $t = 2$, if an $l_1 \times l_2$ torus ($l_1 \geq t$, $l_2 \geq t$) can be perfectly t-interleaved, then the spheres S_t can be perfectly packed in it.

Proof: Let G be an $l_1 \times l_2$ torus. For any t, Theorem 3.1 has shown that if G can be perfectly t-interleaved, then the spheres S_t can be perfectly packed in it. Now we prove the other direction. Assume $t \neq 2$, and the spheres S_t can be perfectly packed in G. Let (x_1, y_1), (x_2, y_2), $\cdots$, (x_n, y_n) be a set of vertices such that the spheres S_t centered or left-centered at them form a perfect packing in G. The proof of Theorem 3.1 has essentially showed that for any i and j $(i \neq j)$, the Lee distance between (x_i, y_i) and (x_j, y_j) is at least t. Now we can interleave G is this way: label each sphere S_t with $|S_t|$ distinct integers such that every integer is used exactly once in every sphere, and make all the spheres to be labelled in the same way (namely, all the spheres have the same 'interleaving pattern'). Clearly, for any two integers a and b, the two sets of vertices respectively labelled by a and b are cosets of each other in the torus. Therefore the Lee distance between any two vertices labelled by the same integer is at least t. So G has a perfect t-interleaving.

□

3.2.2 Perfect t-Interleaving and Its Construction

The following lemma is an important property of perfect sphere packing. It will help us derive the necessary and sufficient conditions for perfect t-interleaving.

Lemma 3.2 *Let t be an even integer and $t \geq 4$. When spheres S_t are perfectly packed in an $l_1 \times l_2$ torus, there exists an integer $a \in \{+1, -1\}$, such that if there is a sphere left-centered at the vertex (x, y), then there are two spheres respectively left-centered at $((x - \frac{t}{2}) \bmod l_1, (y - a \cdot \frac{t}{2}) \bmod l_2)$ and $((x + \frac{t}{2}) \bmod l_1, (y + a \cdot \frac{t}{2}) \bmod l_2)$.*

Proof: Assume spheres S_t are perfectly packed in an $l_1 \times l_2$ torus, where $t \geq 4$ and t is even. First, we need to show that $l_1 \geq t$. When t is even, a sphere S_t spans $t - 1$ rows. So $l_1 \geq t - 1$. Now we show why $l_1 \neq t - 1$. Fig. 3.3 (a) shows two examples — the first example shows a sphere S_4 in a torus of 3 rows, and the second example shows a sphere S_6 in a torus of 5 rows. (The vertices in the two spheres are indicated by relatively large black dots in the figure.) Considering the shapes of the

spheres, we can easily see that the two adjacent vertices in each dashed circle cannot be both contained in non-overlapping spheres. Such a phenomenon always happens when $l_1 = t-1$. Since here spheres S_t are perfectly packed in the torus, we get $l_1 \geq t$.

Clearly, one of the following two cases must be true:

- Case 1: whenever there is a sphere left-centered at a vertex (x, y), there are four spheres respectively left-centered at the four vertices $((x - \frac{t}{2}) \bmod l_1, (y - \frac{t}{2}) \bmod l_2)$, $((x-\frac{t}{2}) \bmod l_1, (y+\frac{t}{2}) \bmod l_2)$, $((x+\frac{t}{2}) \bmod l_1, (y-\frac{t}{2}) \bmod l_2)$ and $((x + \frac{t}{2}) \bmod l_1, (y + \frac{t}{2}) \bmod l_2)$.

- Case 2: there exists a sphere left-centered at a vertex (x_0, y_0), such that there is no sphere left-centered at at least one of the following four vertices — $((x_0 - \frac{t}{2}) \bmod l_1, (y_0 - \frac{t}{2}) \bmod l_2)$, $((x_0 - \frac{t}{2}) \bmod l_1, (y_0 + \frac{t}{2}) \bmod l_2)$, $((x_0 + \frac{t}{2}) \bmod l_1, (y_0 - \frac{t}{2}) \bmod l_2)$ and $((x_0 + \frac{t}{2}) \bmod l_1, (y_0 + \frac{t}{2}) \bmod l_2)$.

If Case 1 is true, then the conclusion of this lemma obviously holds. From now on, let us assume that Case 2 is true. WLOG (without loss of generality), we assume that there is one sphere left-centered at (x_0, y_0), but there is no sphere left-centered at $((x_0 - \frac{t}{2}) \bmod l_1, (y_0 + \frac{t}{2}) \bmod l_2)$. (All the other possible instances can be proved with the same method.)

Since $l_1 \geq t$, the vertex $((x - \frac{t}{2}) \bmod l_1, (y + 1) \bmod l_2)$ — which we shall call 'vertex A' — is not contained in the sphere left-centered at (x_0, y_0). (An example is shown in Fig. 3.3 (b), where the sphere in consideration is an S_8, whose left-center (x_0, y_0) is labelled by 'C'. The vertex A is labelled by 'A'.) The vertex A is contained in one of the perfectly packed spheres, which we shall call 'sphere B'. The relatively position of vertex A in sphere B can only be one of the following two possibilities:

- Possibility 1: the vertex A is the right-most vertex in the bottom row of the sphere B. (See Fig. 3.4 (a).)

- Possibility 2: the vertex A is in the down-left diagonal of the border of the sphere B, but it is not the left-most vertex of the sphere B. (See Fig. 3.4 (b),

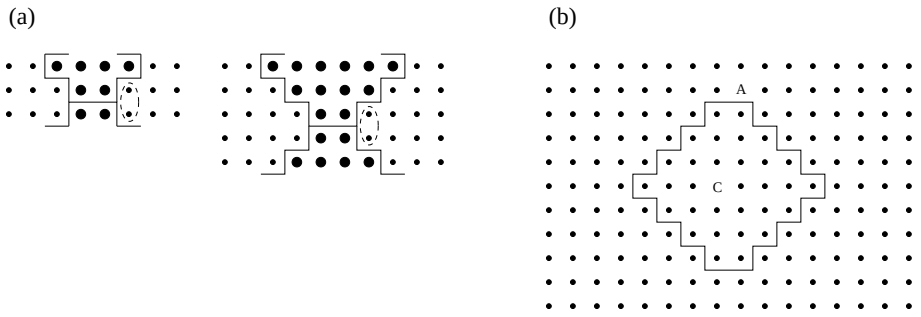


Figure 3.3: A sphere in a torus.

(c) and (d).)

Possibility 1, however, can be easily found to be impossible, since otherwise the neighboring vertex to the right of vertex A and the vertex below it cannot both be contained in non-overlapping spheres. (See the two nodes in the dashed circle in Fig. 3.4 (a).) So only possibility 2 is true. In the following proof we use the example of $t = 8$ for illustration, and assume that the relative position of the sphere B is as shown in Fig. 3.4 (b). We comment that when t takes other values or when the sphere B takes other relative positions, the following argument still holds, which is easy to see.

Let the sphere left-centered at (x_0, y_0) be the sphere denoted by 'L_1' in Fig. 3.5, and let sphere B be the sphere now denoted by 'R_1' in Fig. 3.5. We immediately see that the vertex denoted by 'E' must be the right-most vertex of a sphere, so the sphere containing the vertex 'E' must be the sphere denoted by 'L_2'. Then we immediately see that the vertex denoted by 'F' must be the right-most vertex in the bottom row of a sphere, so the sphere containing the vertex 'F' must be the sphere denoted by 'R_2'. With the same method we can fix the positions of a series of spheres L_1, L_2, L_3, L_4, $\cdots$ and a series of spheres R_1, R_2, R_3, R_4, $\cdots$. Since the torus is finite, we will get a series of spheres L_1, L_2, L_3, L_4, $\cdots$, L_n such that the relative position of L_n to L_1 is the same as the relative position of L_1 to L_2 (see Fig. 3.5 for an illustration) — so such a series of spheres form a 'cycle' in the torus. Since

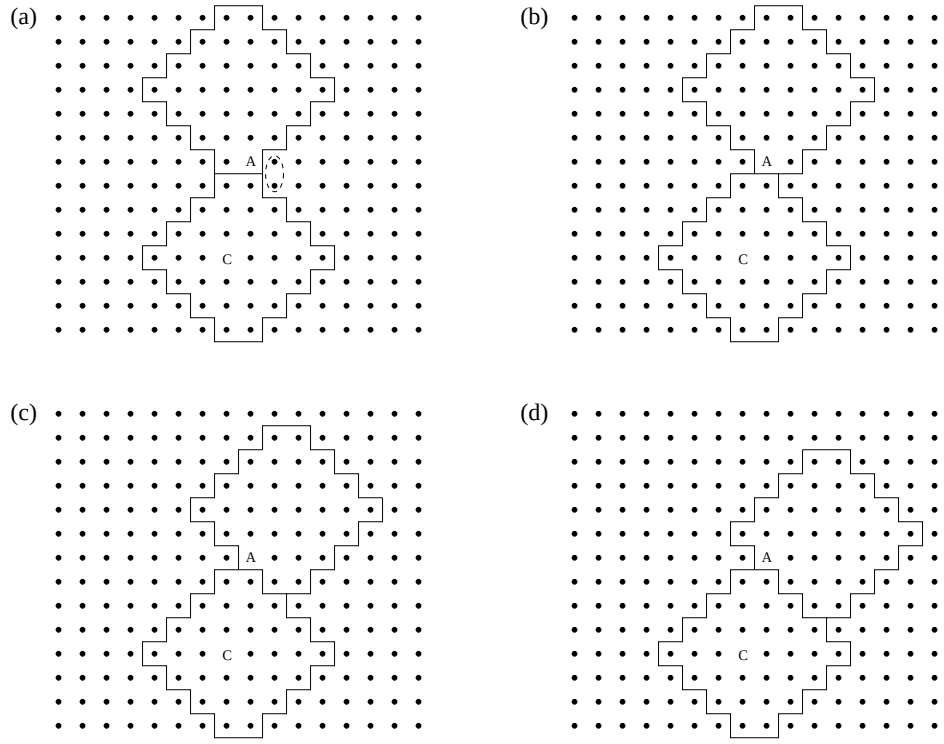


Figure 3.4: Relative positions of spheres and vertices.

the spheres are perfectly packed in the torus, no two spheres in this 'cycle' overlap. Similarly, the spheres R_1, R_2, $\cdots$, R_n also form a 'cycle' in the torus. (Note that we do not make any assumption about whether these two 'cycles' overlap or not.)

If those two 'cycles' contain all the spheres in the torus, then we are already very close to the end of this proof. If those two 'cycles' do not contain all the spheres in the torus, then there must be some spheres outside the two 'cycles' that are directly attached to the down-left side of the 'cycle' formed by L_1, L_2, $\cdots$, L_n. (Consider the very regular way the 'cycle' is formed, and the resulting shape of the 'cycle' which is invariant to horizontal and vertical shifts.) Let D_1 be a sphere directly attached to the 'cycle' formed by L_1, L_2, $\cdots$, L_n, as shown in Fig. 3.5. (Note that we do not care about the exact position of D_1, as long as it is directly attached to the down-left side of the 'cycle'.) Then the node 'I' immediately determines that the sphere containing it must be 'D_2'; similarly the node 'J' determines the position of the sphere 'D_3'; and so on $\cdots$ $\cdots$ So we will get a series of spheres D_1, D_2, D_3, $\cdots$, D_n which will again

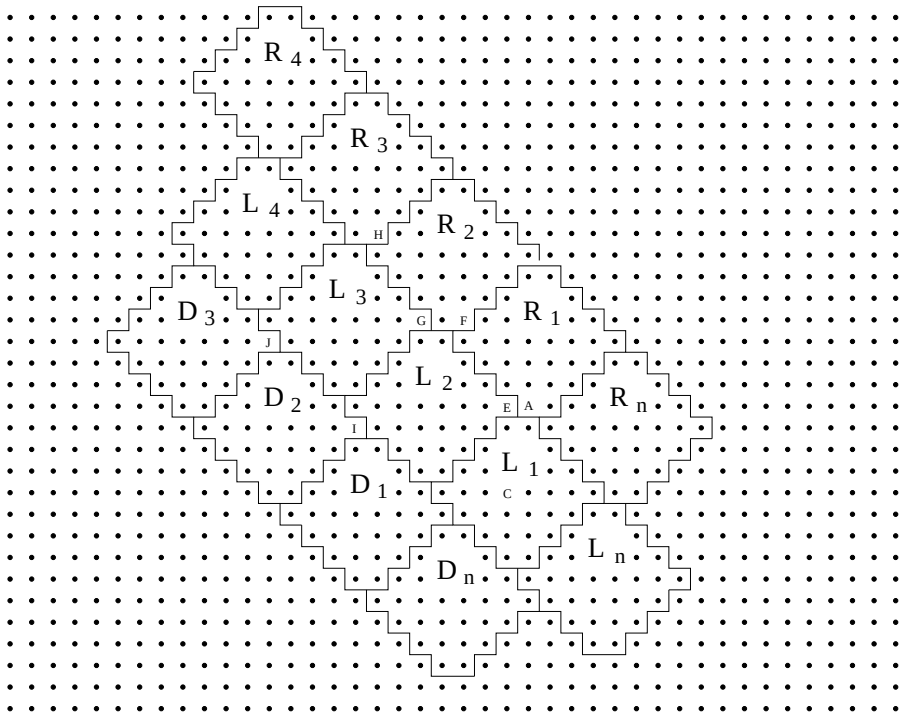


Figure 3.5: The packing of spheres in a torus.

form a 'cycle'. (It is easy to see that this 'cycle' does not overlap the previous two 'cycles'.) With the same method as above, we will find more and more 'cycles', until they together contain all the spheres in the torus.

We can easily see that in each of the 'cycles' here, if there is a sphere left-centered at a vertex (x, y), then there are two spheres respectively left-centered at $((x-\frac{t}{2}) \bmod l_1, (y-\frac{t}{2}) \bmod l_2)$ and $((x+\frac{t}{2}) \bmod l_1, (y+\frac{t}{2}) \bmod l_2)$. When other instances of Case 2 are true (see the definition of 'Case 2' in previous text), it can be shown in the same way that whenever there is a sphere left-centered at a vertex (x, y), there are two spheres respectively left-centered at $((x-\frac{t}{2}) \bmod l_1, (y+\frac{t}{2}) \bmod l_2)$ and $((x+\frac{t}{2}) \bmod l_1, (y-\frac{t}{2}) \bmod l_2)$. By summarizing the above conclusions, we see that this lemma is proved.

□

Definition 3.8 *Let t be an even positive integer, let a be either $+1$ or -1, and let G be an $l_1 \times l_2$ torus. Let (x, y) be an arbitrary vertex in G. We define "the* cycle

containing (x, y) (corresponding to the parameter a)" to be the set of spheres S_t that are respectively left-centered at the vertices (x, y), $((x + \frac{t}{2}) \bmod l_1, (y + a \cdot \frac{t}{2}) \bmod l_2)$, $((x + 2 \cdot \frac{t}{2}) \bmod l_1, (y + 2a \cdot \frac{t}{2}) \bmod l_2)$, $((x + 3 \cdot \frac{t}{2}) \bmod l_1, (y + 3a \cdot \frac{t}{2}) \bmod l_2)$, $\cdots\cdots$
□

The proof of the following lemma is omitted due to its simplicity.

Lemma 3.3 *Let t be an even positive integer, let a be either $+1$ or -1, and let G be an $l_1 \times l_2$ torus. For any vertex (x, y) in G, the* cycle *containing it (corresponding to the parameter a) consists of $\frac{lcm(l_1, l_2, \frac{t}{2})}{\frac{t}{2}}$ distinct spheres S_t.*

The following theorem shows the necessary and sufficient condition for tori that can be perfectly t-interleaved.

Theorem 3.3 *Let G be an $l_1 \times l_2$ torus where $l_1 \geq t$ and $l_2 \geq t$. If t is odd, then G can be perfectly t-interleaved if and only if both l_1 and l_2 are multiples of $\frac{t^2+1}{2}$. If t is even, then G can be perfectly t-interleaved if and only if both l_1 and l_2 are multiples of t.*

Proof: We consider the following three cases one by one:

- Case 1: $t = 2$.
- Case 2: t is even but $t \neq 2$.
- Case 3: t is odd.

Case 1: $t = 2$. In this case, we note that 2-interleaving is equivalent to vertex coloring, so the 2-interleaving number of G equals G's chromatic number $\chi(G)$. Let R_1 and R_2 be two rings which respectively have l_1 and l_2 vertices. Then G is the Cartesian product of those two rings, namely, $G = R_1 \otimes R_2$. It is well known [86] that for any two graphs H_1 and H_2, $\chi(H_1 \otimes H_2) = \max\{\chi(H_1), \chi(H_2)\}$. Since $l_1 \geq t = 2$ (respectively, $l_2 \geq t = 2$), we get that $\chi(R_1) \geq 2$ (respectively, $\chi(R_2) \geq 2$); and $\chi(R_1) = 2$ (respectively, $\chi(R_2) = 2$) if and only if l_1 (respectively, l_2) is a multiple of

2. So $\chi(G) = 2$ if and only if both l_1 and l_2 are multiples of 2. Since $|S_2| = 2$, we get the conclusion in this lemma.

Case 2: t is even but $t \neq 2$. Firstly, we prove one direction. Assume G can be perfectly t-interleaved. Let i be an integer used by a perfect t-interleaving on G. Then by Theorem 3.1, the spheres S_t left-centered at the vertices labelled by i form a perfect sphere packing in G. By Lemma 3.2, there exists an integer $a \in \{+1, -1\}$ such that for any *cycle* containing a vertex labelled by i (corresponding to the parameter a), the spheres S_t in the *cycle* are all left-centered at vertices labelled by i — and therefore they do not overlap. By Lemma 3.3, the *cycle* containing a vertex labelled by i consists of $\frac{lcm(l_1,l_2,\frac{t}{2})}{\frac{t}{2}}$ distinct spheres S_t. So such a *cycle* consists of $\frac{lcm(l_1,l_2,\frac{t}{2})}{\frac{t}{2}} \cdot |S_t| = \frac{lcm(l_1,l_2,\frac{t}{2})}{\frac{t}{2}} \cdot \frac{t^2}{2} = lcm(l_1, l_2, \frac{t}{2}) \cdot t$ vertices. Let (x_1, y_1) and (x_2, y_2) be any two vertices labelled by i. We can see that for the *cycle* containing (x_1, y_1) and the *cycle* containing (x_2, y_2), they either do not overlap, or they are the same *cycle*. Therefore, the vertices in G can be partitioned into several such *cycles* — so $l_1 \cdot l_2$ is a multiple of $lcm(l_1, l_2, \frac{t}{2}) \cdot t$. Since $lcm(l_1, l_2, \frac{t}{2})$ is a multiple of l_1, l_2 must be a multiple of t. Similarly, l_1 must be a multiple of t, too. So if G can be perfectly t-interleaved, then both l_1 and l_2 are multiples of t.

Now we prove the other direction. Assume both l_1 and l_2 are multiples of t. Let W be such a set of vertices in G: $W = \{(x, y) | x \equiv 0 \bmod \frac{t}{2}, y \equiv 0 \bmod \frac{t}{2}, x + y \equiv 0 \bmod t\}$. It is easy to verify that the Lee distance between any two vertices in W is at least t. Now for $i = 0, 1, \cdots, \frac{t}{2} - 1$ and for $j = 0, 1, \cdots, t - 1$, define $W^{i,j}$ to be $W^{i,j} = \{((x + i) \bmod l_1, (y + j) \bmod l_2) | (x, y) \in W\}$. Clearly those $\frac{t}{2} \cdot t = |S_t|$ sets — $W^{0,0}$, $W^{0,1}$, $\cdots$, $W^{\frac{t}{2}-1,t-1}$ — is a partition of the vertices in G. For each $W^{i,j}$, we label the vertices in it with one distinct integer. Clearly such an interleaving is a perfect t-interleaving. So if both l_1 and l_2 are multiples of t, then G can be perfectly t-interleaved.

Case 3: t is odd. Firstly, we prove one direction. Assume both l_1 and l_2 are multiples of $\frac{t^2+1}{2}$. Golomb and Welch have shown in [27] that a $\frac{t^2+1}{2} \times \frac{t^2+1}{2}$ torus can be perfectly packed by the spheres S_t for odd t. Therefore, G can also be perfectly packed

by S_t because a torus has a toroidal topology and G can be 'folded' into a $\frac{t^2+1}{2} \times \frac{t^2+1}{2}$ torus. Let C be a set of vertices in G such that the spheres S_t centered at the vertices in C form a perfect sphere packing. Then the Lee distance between any two vertices in C is at least t. Let (x_0, y_0) be an arbitrary vertex in C. Define M to be such a set of integer-pairs: $M = \{[i,j] | 0 \leq i \leq l_1 - 1, 0 \leq j \leq l_2 - 1, ((x+i) \bmod l_1, (y+j) \bmod l_2)$ *is a vertex in the sphere* $S_t^{(x_0,y_0)}$. $\}$. Clearly $|M| = |S_t|$. For every $[i,j] \in M$, define $C^{i,j}$ to be such a set of vertices in G: $C^{i,j} = \{((x+i) \bmod l_1, (y+j) \bmod l_2) | (x,y) \in C\}$. We see that the sets $C^{i,j}$, for all the elements $[i,j] \in M$, partition the vertices in G; and for every $[i,j] \in M$, the Lee distance between any two vertices in $C^{i,j}$ is at least t. For every $[i,j] \in M$, we label the vertices in $C^{i,j}$ with a distinct integer. Such an interleaving is clearly a perfect t-interleaving. So if both l_1 and l_2 are multiples of $\frac{t^2+1}{2}$, then G can be perfectly t-interleaved.

Now we prove the other direction. Assume G can be perfectly t-interleaved. Let i be an integer used by a perfect t-interleaving on G. Then by Theorem 3.1, the spheres S_t centered at the vertices labelled by i form a perfect sphere packing in G. Golomb and Welch presented in [27] a way to perfectly pack spheres S_t in a torus when t is odd, which can be described as "either of the following two conditions is true: (1) whenever there is a sphere S_t centered at a vertex (x,y), there are two spheres respectively centered at $((x+\frac{t+1}{2}) \bmod l_1, (y+\frac{t-1}{2}) \bmod l_2)$ and $((x-\frac{t-1}{2}) \bmod l_1, (y+\frac{t+1}{2}) \bmod l_2)$; (2) whenever there is a sphere S_t centered at a vertex (x,y), there are two spheres respectively centered at $((x+\frac{t-1}{2}) \bmod l_1, (y+\frac{t+1}{2}) \bmod l_2)$ and $((x-\frac{t+1}{2}) \bmod l_1, (y+\frac{t-1}{2}) \bmod l_2)$". It is well known that that way of packing is in fact the only way to perfectly pack S_t for odd t, whose feasibility requires both l_1 and l_2 to be multiples of $\frac{t^2+1}{2}$. So if G can be perfectly t-interleaved, then both l_1 and l_2 are multiples of $\frac{t^2+1}{2}$.

□

Below we present the complete set of perfect sphere packing constructions. But first let's explain a few concepts. Let G be an $l_1 \times l_2$ torus that is perfectly packed by spheres S_t — there are $\frac{l_1 l_2}{|S_t|}$ such spheres. Define e as $e = \frac{l_1 l_2}{|S_t|}$, and let's say those spheres are centered (or left-centered) at the vertices (x_1, y_1), (x_2, y_2), $\cdots$,

(x_e, y_e). By *vertically (respectively, horizontally) shifting the spheres in* G, we mean to select some integer s, and get a new set of perfectly packed spheres that are centered (or left-centered) at $(x_1 + s \bmod l_1, y_1)$, $(x_2 + s \bmod l_1, y_2)$, $\cdots$, $(x_e + s \bmod l_1, y_e)$ (respectively, at $(x_1, y_1 + s \bmod l_2)$, $(x_2, y_2 + s \bmod l_2)$, $\cdots$, $(x_e, y_e + s \bmod l_2)$). By *vertically reversing the spheres in* G, we mean to get a new set of perfectly packed spheres that are centered (or left-centered) at $(-x_1 \bmod l_1, y_1)$, $(-x_2 \bmod l_1, y_2)$, $\cdots$, $(-x_e \bmod l_1, y_e)$. After such a 'shift' or 'reverse' operation, technically speaking, the way the spheres are perfectly packed in G are changed — however, the 'pattern of the sphere packing' essentially remains the same.

Construction 2.1: *The complete set of perfect sphere packing constructions*
Input: A positive integer t. An $l_1 \times l_2$ torus G, where (1) both l_1 and l_2 are multiples of t if t is even and $t \neq 2$, (2) l_2 is even if $t = 2$, and (3) both l_1 and l_2 are multiples of $\frac{t^2+1}{2}$ if t is odd.
Output: A perfect packing of the spheres S_t in G.
Construction:

1. If t is even and $t \neq 2$, then do the following:

- Let A_1, A_2, $\cdots$, $A_{gcd(\frac{l_1}{t}, \frac{l_2}{t})-1}$ be $gcd(\frac{l_1}{t}, \frac{l_2}{t}) - 1$ integers, where A_i can be any integer in the set $\{0, 1, \cdots, \frac{t}{2} - 1\}$ for $i = 1, 2, \cdots, gcd(\frac{l_1}{t}, \frac{l_2}{t}) - 1$.

- Find the $gcd(\frac{l_1}{t}, \frac{l_2}{t})$ *cycles* in G (corresponding to the parameter 1) respectively containing the vertex $(0, 0)$, $(\sum_{i=1}^{1} A_i, \sum_{i=1}^{1}(t + A_i))$, $(\sum_{i=1}^{2} A_i, \sum_{i=1}^{2}(t + A_i))$, $\cdots$, $(\sum_{i=1}^{gcd(\frac{l_1}{t}, \frac{l_2}{t})-1} A_i, \sum_{i=1}^{gcd(\frac{l_1}{t}, \frac{l_2}{t})-1}(t + A_i))$. The spheres S_t in those $gcd(\frac{l_1}{t}, \frac{l_2}{t})$ *cycles* form a perfect sphere packing in the torus.

2. If $t = 2$, the do the following:

- The $l_1 \times l_2$ torus G has l_1 rows, each of which can be seen as a ring of l_2 vertices. When $t = 2$, the sphere S_t simply consists of two horizontally adjacent vertices. Split each row of G into $\frac{l_2}{2}$ spheres in any way. The resulting $\frac{l_1 l_2}{2}$ spheres form a perfect sphere packing in the torus.

3. If t is odd, then do the following:

- Find such a set of $\frac{l_1 l_2}{|S_t|}$ spheres S_t: each of the spheres is centered at a vertex $(i(m+1)+j\cdot(-m) \bmod l_1, i\cdot m + j(m+1) \bmod l_2)$ for some integers i and j. Those spheres form a perfect sphere packing in the torus.

4. Horizontally shift, vertically shift, and/or vertically reverse the spheres in G in any way.

□

Theorem 3.4 *Construction 2.1 is the* complete *set of perfect sphere packing constructions.*

Proof: We consider the following three cases. For each case, we need to prove two things: firstly, the 'Input' part of Construction 2.1 sets the necessary and sufficient condition for a torus to have perfect sphere packing; secondly, the 'Construction' part of Construction 2.1 generates perfect sphere packing correctly, and every perfect sphere packing that exists is a possible output of it.

Case 1: t is even and $t \neq 2$. In this case, since a sphere S_t occupies $t-1$ rows and t columns, for the $l_1 \times l_2$ torus G to have perfect sphere packing, it must be that $l_1 \geq t-1$ and $l_2 \geq t$. We can show that $l_1 \neq t-1$ in the following way — assume $l_1 = t-1$ and spheres S_t are perfectly packed in G; say a sphere S_t is left-centered at (x, y) in G; then the two vertices, $(x-(\frac{t}{2}-1) \bmod l_1, y-1 \bmod l_2)$ and $(x+(\frac{t}{2}-1) \bmod l_1, y-1 \bmod l_2)$, cannot both be contained in spheres (see the proof of Theorem 3.1 for a very similar argument), and that contradicts the statement that spheres are perfectly packed in G. Therefore, if G can be perfectly packed by spheres, $l_1 \geq t$ and $l_2 \geq t$. Then, from Theorem 3.2 and Theorem 3.3, we see that G can be perfectly packed by spheres if and only if both l_1 and l_2 are multiples of t. So the 'Input' part of Construction 2.1 correctly sets of the necessary and sufficient condition for a torus to have perfect sphere packing.

Lemma 3.2 and its proof have shown that when spheres are perfectly packed in a torus, those spheres can be partitioned into *cycles*. By observing the shape of

the border of a *cycle*, we see that two adjacent *cycles* can freely 'slide' along each other's border — and there are $\frac{t}{2}$ possible relative positions between two adjacent *cycles*. In Construction 2.1, the $\frac{t}{2}$ possible relative positions are determined by A_i, a variable that can take $\frac{t}{2}$ possible values. Now it is easy to see that Step 1 of Construction 2.1 provides a perfect sphere packing (which takes one of many possible forms, depending on the value of the 'A_i's), and its Step 4 changes the positions of the spheres to furthermore cover all the possible cases of perfect sphere packing.

(2) Case 2: $t = 2$. This case is simple, so we skip its analysis.

(3) Case 3: t is odd. In this case, Construction 2.1 re-produces the sphere-packing method presented in [27], which is commonly known as the unique way to pack spheres for odd t (see the final paragraph of the proof of Theorem 3.3 for more detailed introduction).

□

Now we present perfect t-interleaving constructions that are based on perfect sphere packing.

Construction 2.2: *Perfect t-interleaving constructions*

Input: A positive integer t. An $l_1 \times l_2$ torus G, where both l_1 and l_2 are multiples of t if t is even, and both l_1 and l_2 are multiples of $\frac{t^2+1}{2}$ if t is odd.

Output: A perfect t-interleaving on G.

Construction:

(1) If $t \neq 2$, then do the following:

- Use Construction 2.1 to get a perfect sphere packing in G. Label each of those spheres with $|S_t|$ distinct integers, in such a way that all the spheres have the same interleaving pattern, and every integer is used exactly once in each sphere.

(2) If $t = 2$, then do the following:

- For every vertex (i, j) of G ($0 \leq i \leq l_1 - 1$, $0 \leq j \leq l_2 - 1$), if $i + j$ is even, label it with the integer '0', otherwise label it with the integer '1'.

□

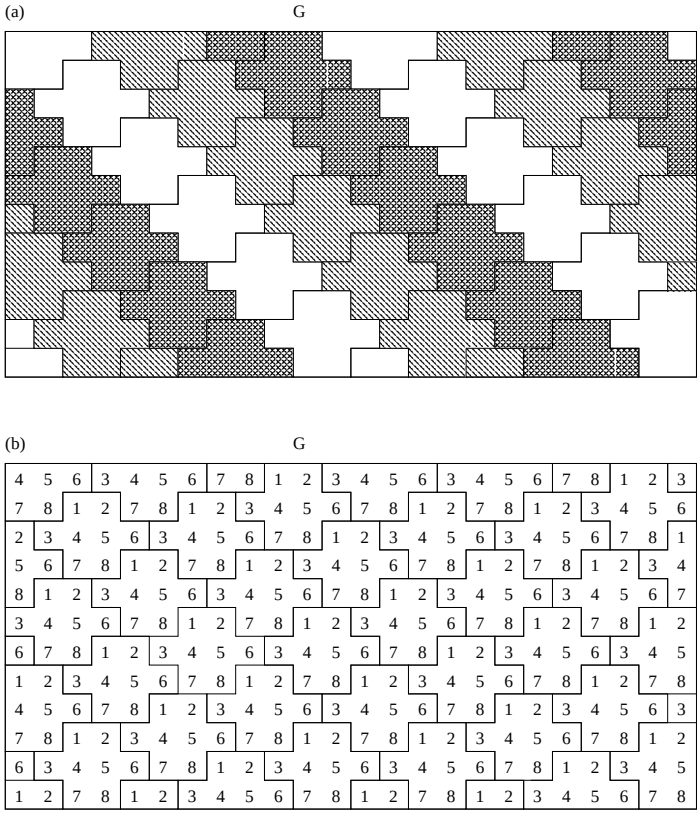


Figure 3.6: Example of perfect t-interleaving using Construction 2.2.

Example 3.3 *Let $t = 4$, and let G be an 12×24 torus. Firstly, we use Construction 2.1 to find a perfect sphere packing in G. Since t is even, the Step 1 of Construction 2.1 is executed. We choose A_1, A_2, $\cdots$, $A_{gcd(\frac{l_1}{t},\frac{l_2}{t})-1}$ to be $A_1 = 0$, $A_2 = 1$. (Note that here $gcd(\frac{l_1}{t},\frac{l_2}{t}) - 1 = 2$.) Then the $gcd(\frac{l_1}{t},\frac{l_2}{t}) = 3$* cycles *in G are as shown in Fig. 3.6 (a), which are three sets of spheres S_t respectively of three different background patterns. The spheres in those 3* cycles *form a perfect packing in G.*

Next, we use Construction 2.2 to perfectly t-interleave G. Let the perfect sphere packing remain as it is; and label all the spheres with the same interleaving pattern, using $|S_t| = 8$ distinct integers. The resulting perfect t-interleaving on G is shown in Fig. 3.6 (b). □

We comment that Construction 2.2 provides the *complete* set of perfect t-interleaving constructions that have the following property: for any two integers, the two sets of

vertices respectively labelled by those two integers are cosets of each other in the torus. What is more, in [11], three t-interleaving constructions were presented, all based on lattice interleavers. Our Construction 2.2 generalizes the results in [11] in two ways: firstly, it covers more constructions based on lattice interleavers, with the results of [11] included as special cases; secondly, when t is even, it also covers constructions that do not use lattice interleavers, which we can make happen by simply letting any A_i and A_j take different values.

3.3 Achieving an Interleaving Degree within One of the Optimal

In this section, we present a t-interleaving construction, with which we can t-interleave any large enough torus with a degree within one of the optimal. The construction presented here will also be used as a building block in Section IV.

3.3.1 Interleaving Construction

Definition 3.9 :

- *Given a positive integer t, if t is odd, then P is defined to be a string of integers 'a_1, a_2, $\cdots$, $a_{\frac{t-1}{2}}$', where $a_{\frac{t-1}{2}} = t+1$ and $a_i = t$ for $1 \leq i < \frac{t-1}{2}$; if t is even, then P is defined to be a string of integers 'a_1, a_2, $\cdots$, $a_{\frac{t}{2}}$', where $a_{\frac{t}{2}} = t$ and $a_i = t-1$ for $1 \leq i < \frac{t}{2}$. (For example, if $t = 3$, then $P =$'4'; if $t = 4$, then $P =$'3,4'; if $t = 5$, then $P =$'5,6'.)*

- *Given a positive integer t, if t is odd, then Q is defined to be a string of integers 'b_1, b_2, $\cdots$, $b_{\frac{t+1}{2}}$', where $b_{\frac{t+1}{2}} = t+1$ and $b_i = t$ for $1 \leq i < \frac{t+1}{2}$; if t is even, then Q is defined to be a string of integers 'b_1, b_2, $\cdots$, $b_{\frac{t}{2}+1}$', where $b_{\frac{t}{2}+1} = t$ and $b_i = t-1$ for $1 \leq i < \frac{t}{2}+1$.*

- *Given a positive integer t, an* offset sequence *is a string of 'P's and 'Q's. (As an example, an offset sequence consisting of 1 'P' and 2 'Q's can be 'PQQ',*

'QPQ' or 'QQP'.) The offset sequence is also naturally seen as a string of integers which is the union of the integers in its 'P's and 'Q's. (For example, when $t = 3$*, if an offset sequence consisting of 1 'P' and 2 'Q's is 'PQQ', then the offset sequence is also seen as '4,3,4,3,4'; when* $t = 4$*, if an offset sequence consisting of 3 'P's and 2 'Q's is 'PQPPQ', then the offset sequence is also seen as '3,4,3,3,4,3,4,3,4,3,3,4'.) The number of integers in an offset sequence is called its* length*.*

□

In this section, we are particularly interested in one kind of t-interleaving on an $l_1 \times l_2$ torus, which has the following features:

- Feature 1: $l_1 = |S_t| + 1$. (In other words, if t is odd, then $l_1 = \frac{t^2+1}{2} + 1$; if t is even, then $l_1 = \frac{t^2}{2} + 1$.)
- Feature 2: The degree of the t-interleaving equals l_1. And in every column of the torus, each of the l_1 integers is assigned to exactly one vertex.
- Feature 3: If the vertex (a_1, b_1) and the vertex (a_2, b_2) are labelled by the same integer, then for $i = 1, 2, \cdots, l_1 - 1$, the vertex $((a_1 + i) \bmod l_1, b_1)$ and the vertex $((a_2 + i) \bmod l_1, b_2)$ are labelled by the same integer.

Example 3.4 *Fig. 3.7 shows a t-interleaving on an* $l_1 \times l_2$ *torus which has the above three features. There* $t = 3$*,* $l_1 = |S_t| + 1 = 6$ *and* $l_2 = 8$*.*

Now let's fixed an integer 'i', where $0 \leq i \leq 5$*, and say the set of vertices labelled by 'i' are '*$(x_0, 0)$*,* $(x_1, 1)$*,* $\cdots$*,* $(x_{l_2-1}, l_2 - 1)$*'. Then the following string of integers: '*$(x_1 - x_0) \bmod l_1, (x_2 - x_1) \bmod l_1, \cdots, (x_7 - x_6) \bmod l_1, (x_0 - x_7) \bmod l_1$*', equals '4,4,4,3,4,4,3,4'. Since when* $t = 3$*, P ='4' and Q ='3,4', the above string of integers actually equals 'PPPQPQ', which is an offset sequence of length* l_2*. We comment that this phenomenon is not a pure coincidence — offset sequences do help us find t-interleavings that have the above three features. In fact, we can prove that in many cases (e.g., when* $t = 5$ *or 7), for* any *t-interleaving on a torus that has the above*

0	2	4	0	3	5	1	4
1	3	5	1	4	0	2	5
2	4	0	2	5	1	3	0
3	5	1	3	0	2	4	1
4	0	2	4	1	3	5	2
5	1	3	5	2	4	0	3

Figure 3.7: An example of t-interleaving of special features.

three features, after horizontally shifting and/or vertically reversing the interleaving pattern, the resulting interleaving will have the same phenomenon as the example shown here.

□

The following construction outputs t-interleaving that has the three features.

Construction 3.1:

Input: A positive integer t. An $l_1 \times l_2$ torus, where $l_1 = |S_t| + 1$. An integer m that equals $\lfloor \frac{t}{2} \rfloor$. Two integers p and q that satisfy the following equation set if t is odd:

$$\begin{cases} pm + q(m+1) = l_2 \\ p(2m^2 + m + 1) + q(2m^2 + 3m + 2) \equiv 0 \bmod (2m^2 + 2m + 2) \\ p \text{ and } q \text{ are non-negative integers, } p + q > 0. \end{cases} \tag{3.1}$$

and satisfy the following equation set if t is even:

$$\begin{cases} pm + q(m+1) = l_2 \\ p(2m^2 - m + 1) + q(2m^2 + m) \equiv 0 \bmod (2m^2 + 1) \\ p \text{ and } q \text{ are non-negative integers, } p + q > 0. \end{cases} \tag{3.2}$$

Output: A t-interleaving on the $l_1 \times l_2$ torus.

Construction: Let $S =$‘$s_0, s_1, \cdots, s_{l_2-1}$’ be an arbitrary offset sequence consisting of p ‘P’s and q ‘Q’s. For $j = 1, 2, \cdots, l_2$ and for $i = 0, 1, \cdots, l_1 - 1$, label the vertex

$((\sum_{k=0}^{j-1} s_k + i) \bmod l_1, j \bmod l_2)$ with the integer 'i'.

□

Example 3.5 *Let $t = 3$, $l_1 = 6$, $l_2 = 8$, $m = 1$, $p = 4$, and $q = 2$. We use Construction 3.1 to t-interleave an $l_1 \times l_2$ torus. Say the offset sequence S is chosen to be 'PPPQPQ'. Then Construction 3.1 outputs the t-interleaving shown in Fig. 3.7.*
□

We explain Construction 3.1 a little bit. The Equation Set (1) (for odd t) and the Equation Set (2) (for even t) ensure that the offset sequence S, which consists of p 'P's and q 'Q's, exists. Furthermore, for any integer j $(0 \leq j \leq l_2 - 1)$, if (a, j) and $(b, (j+1) \bmod l_2)$ are two vertices labelled by the same integer, then $b - a \equiv s_j \bmod l_1$ — namely, the offset sequence S indicates the *vertical offsets* of any two vertices in adjacent columns that are labelled by the same integer. It is simple to verify that the t-interleaving output by Construction 3.1 satisfies all the three features — Feature 1, 2 and 3 — listed earlier in this subsection.

The following lemma will be used to prove the correctness of Construction 3.1 and also in future analysis.

Lemma 3.4 *Let $i \in \{0, 1, \cdots, |S_t|\}$ be any of the integers used by Construction 3.1 to interleave the $l_1 \times l_2$ torus. Let $\{(b_0, 0), (b_1, 1), \cdots, (b_{l_2-1}, l_2 - 1)\}$ be the set of vertices in the torus that are labelled by i. Let m and S have the same meaning as in Construction 3.1 (namely, $m = \lfloor \frac{t}{2} \rfloor$, and $S =$ '$s_0, s_1, \cdots, s_{l_2-1}$' is the offset sequence consisting of p 'P's and q 'Q's utilized by Construction 3.1). For any two integers j_1 and j_2 $(0 \leq j_1 \neq j_2 \leq l_2 - 1)$, we define $L_{j_1 \to j_2}$ as $L_{j_1 \to j_2} = [(j_2 - j_1) \bmod l_2] + \min\{(b_{j_2} - b_{j_1}) \bmod l_1, (b_{j_1} - b_{j_2}) \bmod l_1\}$. Then we have the following conclusions:*

- *Case 1: if t is odd, $j_2 - j_1 \equiv m \bmod l_2$, and $s_{j_1}, s_{(j_1+1) \bmod l_2}, s_{(j_1+2) \bmod l_2}, \cdots, s_{(j_2-1) \bmod l_2}$ do not all equal t, then $b_{j_2} - b_{j_1} \equiv -(m+1) \bmod l_1$ and $L_{j_1 \to j_2} = t$.*
- *Case 2: if t is odd, $j_2 - j_1 \equiv m + 1 \bmod l_2$, and exactly one of $s_{j_1}, s_{(j_1+1) \bmod l_2}, s_{(j_1+2) \bmod l_2}, \cdots, s_{(j_2-1) \bmod l_2}$ equals $t+1$, then $b_{j_2} - b_{j_1} \equiv m \bmod l_1$ and $L_{j_1 \to j_2} = t$.*

- *Case 3: if t is even, $j_2 - j_1 \equiv 1 \bmod l_2$, and $s_{j_1} = t - 1$, then $b_{j_2} - b_{j_1} \equiv t - 1 \bmod l_1$ and $L_{j_1 \to j_2} = t$.*

- *Case 4: if t is even, $j_2 - j_1 \equiv m \bmod l_2$, and $s_{j_1}, s_{(j_1+1) \bmod l_2}, s_{(j_1+2) \bmod l_2}, \cdots, s_{(j_2-1) \bmod l_2}$ do not all equal $t-1$, then $b_{j_2} - b_{j_1} \equiv -m \bmod l_1$ and $L_{j_1 \to j_2} = t$.*

- *Case 5: if t is even, $j_2 - j_1 \equiv m+1 \bmod l_2$, and exactly one of $s_{j_1}, s_{(j_1+1) \bmod l_2}$, $s_{(j_1+2) \bmod l_2}, \cdots, s_{(j_2-1) \bmod l_2}$ equals t, then $b_{j_2} - b_{j_1} \equiv m-1 \bmod l_1$ and $L_{j_1 \to j_2} = t$.*

- *Case 6: if none of the above five cases is true, and $j_2 - j_1 \neq t \bmod l_2$, then $L_{j_1 \to j_2} > t$. If none of the above five cases is true, and $j_2 - j_1 \equiv t \bmod l_2$, then $L_{j_1 \to j_2} \geq t$.*

Proof: Let $\Delta = t+1$ if t is odd, and let $\Delta = t$ if t is even. The offset sequence S consists of 'P's and 'Q's, so it has the following property: for any $i \in \{0, 1, \cdots, l_2 - 1\}$ such that $s_i = \Delta$, the following $m-1$ integers — $s_{(i+1) \bmod l_2}$, $s_{(i+2) \bmod l_2}$, $\cdots$, $s_{(i+m-1) \bmod l_2}$ — all equal $\Delta - 1$, and either $s_{(i+m) \bmod l_2}$ or $s_{(i+m+1) \bmod l_2}$ equals Δ. Also note that $b_{j_2} - b_{j_1} \equiv s_{j_1} + s_{(j_1+1) \bmod l_2} + s_{(j_1+2) \bmod l_2} + \cdots + s_{(j_2-1) \bmod l_2} \bmod l_1$. Based on those two observations, this lemma can be proved with straightforward computation.

□

Theorem 3.5 *Construction 3.1 is correct.*

Proof: Let (b_{j_1}, j_1) and (b_{j_2}, j_2) be any two vertices labelled by the same integer in the $l_1 \times l_2$ torus that was interleaved by Construction 3.1. The Lee distance between them is $d((b_{j_1}, j_1), (b_{j_2}, j_2)) = \min\{(j_2 - j_1) \bmod l_2, (j_1 - j_2) \bmod l_2\} + \min\{(b_{j_2} - b_{j_1}) \bmod l_1, (b_{j_1} - b_{j_2}) \bmod l_1\} = \min\{L_{j_1 \to j_2}, L_{j_2 \to j_1}\}$. From Lemma 3.4, it is clearly that both $L_{j_1 \to j_2}$ and $L_{j_2 \to j_1}$ are no less than t. Therefore $d((b_{j_1}, j_1), (b_{j_2}, j_2)) \geq t$. So Construction 3.1 t-interleaved the torus. And as mentioned before, this t-interleaving satisfies Feature 1, Feature 2 and Feature 3.

□

3.3.2 Existence of Offset Sequences

The feasibility of Construction 3.1 depends only on one thing — whether the two input parameters 'p' and 'q' exist or not. The following theorem shows that when the width of the torus, l_2, exceeds a threshold, 'p' and 'q' are guaranteed to exist.

Theorem 3.6 *Let t be an odd (respectively, even) positive integer. When $l_2 \geq \lfloor \frac{t}{2} \rfloor (\lfloor \frac{t}{2} \rfloor + 1)(|S_t| + 1)$, there exists at least one solution (p, q) to the equation set (1) (respectively, equation set (2)), which is shown in the 'Input' part of Construction 3.1.*

Proof: Firstly, let's assume t is odd. The equation set (1) is as follows:

$$\begin{cases} pm + q(m+1) = l_2 \\ p(2m^2 + m + 1) + q(2m^2 + 3m + 2) \equiv 0 \bmod (2m^2 + 2m + 2) \\ p \text{ and } q \text{ are non-negative integers, } p + q > 0. \end{cases}$$

where $m = \lfloor \frac{t}{2} \rfloor$. We introduce a new variable z, and transform the above equation set equivalently to be:

$$\begin{cases} \begin{pmatrix} m & m+1 \\ 2m^2 + m + 1 & 2m^2 + 3m + 2 \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} l_2 \\ z(2m^2 + 2m + 2) \end{pmatrix} \\ p \text{ and } q \text{ are non-negative integers; } z \text{ is a positive integer.} \end{cases}$$

which is the same as:

$$\begin{cases} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} m & m+1 \\ 2m^2 + m + 1 & 2m^2 + 3m + 2 \end{pmatrix}^{-1} \begin{pmatrix} l_2 \\ z(2m^2 + 2m + 2) \end{pmatrix} \\ p \text{ and } q \text{ are non-negative integers; } z \text{ is a positive integer.} \end{cases}$$

which equals:

$$\begin{cases} p = 2(m+1)(m^2+m+1)z - (2m^2+3m+2)l_2 \\ q = (2m^2+m+1)l_2 - 2m(m^2+m+1)z \\ p \text{ and } q \text{ are non-negative integers; } z \text{ is a positive integer.} \end{cases}$$

There exists a solution for the variables p, q and z in the above equation set if and only if the following conditions can be satisfied:

$$\begin{cases} 2(m+1)(m^2+m+1)z - (2m^2+3m+2)l_2 \geq 0 \\ (2m^2+m+1)l_2 - 2m(m^2+m+1)z \geq 0 \\ z \text{ is a positive integer.} \end{cases}$$

which is equivalent to:

$$\begin{cases} \frac{(2m^2+3m+2)l_2}{2(m+1)(m^2+m+1)} \leq z \leq \frac{(2m^2+m+1)l_2}{2m(m^2+m+1)} \\ z \text{ is a positive integer.} \end{cases}$$

To enable a value for z to exist that satisfies the above conditions, it is sufficient to make $\frac{(2m^2+m+1)l_2}{2m(m^2+m+1)} - \frac{(2m^2+3m+2)l_2}{2(m+1)(m^2+m+1)} \geq 1$ — that is, to make $l_2 \geq 2m(m+1)(m^2+m+1) = \lfloor \frac{t}{2} \rfloor (\lfloor \frac{t}{2} \rfloor + 1)(|S_t|+1)$. Therefore when $l_2 \geq \lfloor \frac{t}{2} \rfloor (\lfloor \frac{t}{2} \rfloor + 1)(|S_t|+1)$, there exists at least one solution (p, q) to the equation set (1).

When t is even, the conclusion can be proved in a very similar way. We skip its details.

□

Corollary 3.1 *When $l_2 \geq \lfloor \frac{t}{2} \rfloor (\lfloor \frac{t}{2} \rfloor + 1)(|S_t|+1)$, Construction 3.1 can be used to output a t-interleaving on an $(|S_t|+1) \times l_2$ torus.*

Proof: When $l_2 \geq \lfloor \frac{t}{2} \rfloor (\lfloor \frac{t}{2} \rfloor + 1)(|S_t|+1)$, all the parameters in the 'Input' part of Construction 3.1 exist, including p and q.

□

(a)

A

0	1	2	3
3	2	1	0

B

0	0	0	0
1	1	1	1
2	2	2	2

C

1	2
3	4

(b)

D

0	1	2	3
3	2	1	0
0	0	0	0
1	1	1	1
2	2	2	2

E

1	2	0	1	2	3	1	2
3	4	3	2	1	0	3	4

Figure 3.8: Examples of *tiling tori*

3.3.3 Interleaving with Degree within One of the Optimal

We define the simple term of *tiling tori* here. By tiling several interleaved tori vertically or horizontally, we get a larger torus, whose interleaving is the straightforward combination of the interleaving on the smaller tori. It is best explained with an example.

Example 3.6 *Three interleaved tori— A, B and C — are shown in Fig.3.8. The torus D is a 5×4 torus, got by* tiling A and B vertically *in the form of* $\begin{bmatrix} A \\ B \end{bmatrix}$. *The torus E is a 2×8 torus, got by* tiling one copy of A and two copies of C horizontally *in the form of* $\begin{bmatrix} C & A & C \end{bmatrix}$.

□

The following construction t-interleaves a large enough torus with at most $|S_t|+2$ distinct integers.

Construction 3.2: *t-interleave an $l_1 \times l_2$ torus G, where $l_1 \geq |S_t|(|S_t|+1)$ and $l_2 \geq \lfloor \frac{t}{2} \rfloor (\lfloor \frac{t}{2} \rfloor + 1)(|S_t|+1)$, using at most $|S_t|+2$ distinct integers.*

1. Let G_1 be an $(|S_t|+1) \times l_2$ torus that is t-interleaved by Construction 3.1, using the integers '0','1', $\cdots$, '$|S_t|$'. Let $\{(c_0, 0), (c_1, 1), \cdots, (c_{l_2-1}, l_2-1)\}$ be the set of vertices in G_1 labelled by the integer '0'.

2. Let G_2 be an $(|S_t|+2) \times l_2$ torus. Label the nodes $\{(c_0, 0), (c_1, 1), \cdots, (c_{l_2-1}, l_2-1)\}$ *in* G_2 with the integer '$|S_t|+1$'.

G_1

0	2	0	2	0	2
1	0	1	0	1	0
2	1	2	1	2	1

G_2

3	2	3	2	3	2
0	3	0	3	0	3
1	0	1	0	1	0
2	1	2	1	2	1

G

0	2	0	2	0	2
1	0	1	0	1	0
2	1	2	1	2	1
3	2	3	2	3	2
0	3	0	3	0	3
1	0	1	0	1	0
2	1	2	1	2	1

Figure 3.9: Examples of Construction 3.2.

3. For $j = 0, 1, \cdots, l_2 - 1$ and for $i = 1, 2, \cdots, |S_t| + 1$, label the node $((c_j + i) \bmod (|S_t| + 2), j)$ with the integer '$i - 1$'.

4. Let x and y be two non-negative integers such that $l_1 = x(|S_t|+1)+y(|S_t|+2)$. Tile x copies of G_1 and y copies of G_2 vertically to get an $l_1 \times l_2$ torus G, which is t-interleaved using at most $|S_t| + 2$ distinct integers.

□

Example 3.7 *We use Construction 3.2 to t-interleave a* 7×6 *torus* G*, where* $t = 2$*. The first step is to use Construction 3.1 to t-interleave a* 3×6 *torus* G_1*. Say the offset sequence selected in Construction 3.1 is* $S =$ '$QQQ' =$ '$1, 2, 1, 2, 1, 2'$*, then* G_1 *is as shown in Fig. 3.9. Then the* 4×6 *torus* G_2 *is as shown in the figure. By tiling one copy of* G_1 *and one copy of* G_2 *vertically, we get the t-interleaved torus* G*.* $|S_t|+2 = 4$ *distinct integers are used to interleave* G*.*

□

Theorem 3.7 *Construction 3.2 is correct.*

Proof: It is a known fact that for any two relatively prime positive integers A and B, any integer C no less than $(A-1)(B-1)$ can be expressed as $C = xA+yB$ where x and y are non-negative integers. Therefore in Construction 3.2, since $l_1 \geq |S_t|(|S_t| + 1)$, l_1 indeed can be expressed as $l_1 = x(|S_t| + 1) + y(|S_t| + 2)$, as shown in the last step of Construction 3.2. So the construction can be executed from beginning to

end successfully. Now we prove that the construction does t-interleave G — that is, for any two nodes (a_1, b_1) and (a_2, b_2) labelled by the same integer i in G, the Lee distance between them is at least t. We consider three cases.

Case 1: $b_1 = b_2$, which means that (a_1, b_1) and (a_2, b_2) are in the same column of G. We see every column of G as a ring of length l_1 (because it is toroidal). Then, observe the integers labelling a column of G, and we can see that on the column, the integers following an integer '$|S_t| + 1$' and before the next integer '$|S_t| + 1$' must be '$0, 1, \cdots, |S_t|, 0, 1, \cdots, |S_t|, \cdots\cdots, 0, 1, \cdots, |S_t|$', where the pattern $0, 1, \cdots, |S_t|$ appears at least once. Therefore since (a_1, b_1) and (a_2, b_2) are labelled by the same integer, the Lee distance between them must be at least $|S_t| + 1 > t$.

Case 2: $b_1 \neq b_2$, and $i \neq |S_t| + 1$. In this case, let's first observe two conclusions:

- The interleaving on G_2 is t-interleaving. (See Construction 3.2 for the definition of G_2.) This can be proved as follows: any two vertices labelled by the same integer in G_2 can be expressed as $((c_{j_1} + i_0) \bmod (|S_t|+2), j_1)$ and $((c_{j_2} + i_0) \bmod (|S_t| + 2), j_2)$ (see the Step 2 and Step 3 of Construction 3.2); then, $d_{G_2}(((c_{j_1} + i_0) \bmod (|S_t| + 2), j_1), ((c_{j_2} + i_0) \bmod (|S_t| + 2), j_2)) = d_{G_2}((c_{j_1}, j_1), (c_{j_2}, j_2)) \geq d_{G_1}((c_{j_1}, j_1), (c_{j_2}, j_2)) \geq t$.

- Let (α, j) and (β, j) be two vertices respectively in G_1 and G_2 both of which are labelled by the same integer. Then it is simple to see that $\beta = \alpha$ or $\beta = \alpha + 1$. Since G_1 has $|S_t| + 1$ rows and G_2 has $|S_t| + 2$ rows, we have $d_{G_2}((\beta, j), (0, j)) \geq d_{G_1}((\alpha, j), (0, j))$ and $d_{G_2}((\beta, j), (|S_t| + 1, j)) \geq d_{G_1}((\alpha, j), (|S_t|, j))$. That is, if u and v are two vertices respectively in G_1 and G_2 both of which are in the j-th column and labelled by the same integer, the vertical distance from v to the two 'borders' of G_2 is no less than the vertical distance from u to the two 'borders' of G_1.

According to Construction 3.2, G is got by vertically tiling x copies of G_1 and y copies of G_2. Let's call each of those $x+y$ tori a *component torus* of G. Now, if (a_1, b_1) and (a_2, b_2) are in the same component torus of G, we know the Lee distance between

them in G is no less than the Lee distance between them in that component torus, which is at least t because that component torus is t-interleaved. If (a_1, b_1) and (a_2, b_2) are not in the same component torus of G, we do the following. We firstly construct a torus G' which is got by vertically tiling $x + y$ copies of G_1. It is simple to see that G' is t-interleaved. We call each of the $x + y$ copies of G_1 in G' a *component torus* of G'. Let's say (a_1, b_1) and (a_2, b_2) are respectively in the k_1-th and k_2-th component torus of G. Let (c_1, b_1) and (c_2, b_2) be the two vertices labelled by the integer i that are respectively in the k_1-th and k_2-th component torus of G'. Observe the shortest path between (a_1, b_1) and (a_2, b_2) in G, and we see that it can be split into such three intervals: from (a_1, b_1) to a border of the k_1-th component torus, from the border of the k_1-th component torus to the border of the k_2-th component torus, and from the border of the k_2-th component torus to (a_2, b_2). There is a corresponding (not necessarily shortest) path connecting (c_1, b_1) and (c_2, b_2) in G', which can be split into such three intervals similarly. And each of the three intervals of the first path is at least as long as the corresponding interval of the second path. G' is t-interleaved, so the second path's length is at least t. So the Lee distance between (a_1, b_1) and (a_2, b_2) in G is at least t.

Case 3: $b_1 \neq b_2$, and $i = |S_t| + 1$. In this case, it is simple to see that the two vertices in G, $(a_1{+}1 \bmod l_1, b_1)$ and $(a_2{+}1 \bmod l_1, b_2)$, are both labelled by the integer 0. Based on the conclusion of Case 2, $d_G((a_1 + 1 \bmod l_1, b_1), (a_2 + 1 \bmod l_1, b_2)) \geq t$. So $d_G((a_1, b_1), (a_2, b_2)) = d_G((a_1 + 1 \bmod l_1, b_1), (a_2 + 1 \bmod l_1, b_2)) \geq t$.

So Construction 3.2 correctly t-interleaved G.

□

As a result of Construction 3.2, we get the following theorem.

Theorem 3.8 *When $l_1 \geq |S_t|(|S_t| + 1)$ and $l_2 \geq \lfloor \frac{t}{2} \rfloor (\lfloor \frac{t}{2} \rfloor + 1)(|S_t| + 1)$, an $l_1 \times l_2$ (or equivalently, $l_2 \times l_1$) torus' t-interleaving number is at most $|S_t| + 2$.*

By combining Construction 2.2 (the construction for perfect t-interleaving) and Construction 3.2, we can t-interleave any sufficiently large torus with a degree within one of the optimal.

3.4 Optimal Interleaving on Large Tori

In the previous section, it is shown that when l_2 is large enough, an $(|S_t|+1) \times l_2$ torus can be t-interleaved using $|S_t|+1$ integers. In this section, we will construct an $[k(|S_t|+1)-1] \times l_2$ torus which is also t-interleaved using $|S_t|+1$ integers, by using an operation we call '*removing a zigzag row*'. ('k' is some integer.) Those two tori have a special property: when they (or multiple copies of them) are tiled vertically to get a larger torus, the larger torus is also t-interleaved with degree $|S_t|+1$. $|S_t|+1$ and $k(|S_t|+1)-1$ are relatively prime, so a large enough l_1 must be a linear combination of those two numbers with non-negative integral coefficients — therefore an $l_1 \times l_2$ torus can be t-interleaved using $|S_t|+1$ integers in this way. We present constructions to optimally t-interleave such tori; and as a parallel result, the existence of Region I (see Section I: Introduction) is proved.

All the results of this section can be split into two parts: one for the case 't is odd', and the other for the case 't is even'. Those two cases can be analyzed with very similar methods; however their analysis and results differ in details. For succinctness, in this section, we only analyze in detail the case 't is odd', which should suffice for illustrating all the ideas. So in the first three subsections here — Subsection A, B, and C, we always assume that t *is odd*. In Subsection D, we present just the final result for the case 't is even'. We list the major intermediate results for the case 't is even' in Appendix II.

3.4.1 Removing a Zigzag Row in a Torus

Definition 3.10 *A* zigzag row *in an $l_1 \times l_2$ torus is a set of l_2 vertices of the torus: $\{(a_0,0),(a_1,1),\cdots,(a_{l_2-1},l_2-1)\}$, where $0 \le a_i \le l_1-1$ for $i=0,1,\cdots,l_2-1$. (For example, $\{(2,0),(3,1),(0,2),(0,3),(3,4)\}$ is a zigzag row in a 4×5 torus.)* □

Definition 3.11 *Let T be an $l_1 \times l_2$ torus. Let $\{(a_0,0),(a_1,1),\cdots,(a_{l_2-1},l_2-1)\}$ be a zigzag row in T. Let there be an interleaving on T, which labels T's vertex (b,c) with the integer $I(b,c)$, for $b=0,1,\cdots,l_1-1$ and $c=0,1,\cdots,l_2-1$. Then a torus*

T

1	3	5	2	4
2	4	(6)	3	(5)
3	(5)	1	4	6
(4)	6	2	(5)	1
5	1	3	6	2
6	2	4	1	3

G

1	3	5	2	4
2	4	1	3	6
3	6	2	4	1
5	1	3	6	2
6	2	4	1	3

Figure 3.10: Removing a zigzag row $\{(3,0),(2,1),(1,2),(3,3),(1,4)\}$ in T.

G is said to be 'got by removing the zigzag row $\{(a_0,0),(a_1,1),\cdots,(a_{l_2-1},l_2-1)\}$ in T' if and only if these two conditions are satisfied:

- *G is an $(l_1-1)\times l_2$ torus.*
- *For $i=0,1,\cdots,l_1-2$ and $j=0,1,\cdots,l_2-1$, the node (i,j) in G is labelled by the integer $I(i,j)$ if $i<a_j$, and by the integer $I(i+1,j)$ if $i\geq a_j$.*

□

Example 3.8 *In Fig. 3.10, a 6×5 torus T is shown. A zigzag row $\{(3,0),(2,1),(1,2),(3,3),(1,4)\}$ in T is circled in the figure. Fig. 3.10 shows a torus G got by removing the zigzag row $\{(3,0),(2,1),(1,2),(3,3),(1,4)\}$ in T.*

It can be readily observed that G can be seen as being derived from T in the following way: firstly, delete the zigzag row in T that is circled in Fig. 3.10; then in each column of T, move the vertices below the circled vertex upward. □

We present three rules to follow for devising a zigzag row. Let B be an $l_0\times l_2$ torus which is t-interleaved by Construction 3.1. (That means $l_0=|S_t|+1$.) Let $S=$'$s_0,s_1,\cdots,s_{l_2-1}$' be the offset sequence utilized by Construction 3.1 when it was t-interleaving B. Let H be an $l_1\times l_2$ torus got by tiling several copies of B vertically. Let $m=\lfloor\frac{t}{2}\rfloor$. Then the three rules for devising a zigzag row <u>in H</u> — $\{(a_0,0),(a_1,1),\cdots,(a_{l_2-1},l_2-1)\}$ — are:

- Rule 1: For any j such that $0 \leq j \leq l_2 - 1$, if the integers $s_j, s_{(j+1) \bmod l_2}, \cdots, s_{(j+m-1) \bmod l_2}$ do not all equal t, then $a_j \geq a_{(j+m) \bmod l_2} + m$.

- Rule 2: For any j such that $0 \leq j \leq l_2 - 1$, if exactly one of the integers $s_j, s_{(j+1) \bmod l_2}, \cdots, s_{(j+m) \bmod l_2}$ equals $t+1$, then $a_j \leq a_{(j+m+1) \bmod l_2} - (m-1)$.

- Rule 3: For any j such that $0 \leq j \leq l_2 - 1$, $m \leq a_j \leq l_1 - m - 1$.

Lemma 3.5 *Let B be a torus t-interleaved by Construction 3.1. Let H be a torus got by tiling copies of B vertically, and let T be a torus got by removing a zigzag row in H, where the zigzag row in H follows the three rules — Rule 1, Rule 2 and Rule 3. Let G be a torus got by tiling copies of B and T vertically. Then, both T and G are t-interleaved.*

Proof: When $t = 1$, the proof is trivial. So we assume $t \geq 3$ in the rest of the proof. It is simple to see that H is t-interleaved, because H is got by tiling B, a t-interleaved torus. We assume B is an $l_0 \times l_2$ torus (where $l_0 = |S_t| + 1$), H is an $l_1 \times l_2$ torus (where l_1 is a multiple of l_0), T is an $l_T \times l_2$ torus (where $l_T = l_1 - 1$), and G is an $l_G \times l_2$ torus. Let $m = \lfloor \frac{t}{2} \rfloor$. Let $S =$'$s_0, s_1, \cdots, s_{l_2-1}$' be the offset sequence utilized by Construction 3.1 when it was t-interleaving B.

(1) In this part, we will prove that T is t-interleaved. Let (x_1, y_1) and (x_2, y_2) be two vertices in T both labelled by some integer 'r'. We need to prove that $d_T((x_1, y_1), (x_2, y_2)) \geq t$.

Let $\{(a_0, 0), (a_1, 1), \cdots, (a_{l_2-1}, l_2 - 1)\}$ denote the zigzag row removed in H to get T. If $a_{y_1} \leq x_1$, then let $z_1 = x_1 + 1$; otherwise let $z_1 = x_1$. Similarly, if $a_{y_2} \leq x_2$, then let $z_2 = x_2 + 1$; otherwise let $z_2 = x_2$. Clearly, the two vertices in H, (z_1, y_1) and (z_2, y_2), are also labelled by 'r'.

We only need to consider the following three cases:

Case 1: $y_1 = y_2$. In this case, $d_H((z_1, y_1), (z_2, y_2))$ is a multiple of $|S_t| + 1$ (the number of rows in B); and $d_T((x_1, y_1), (x_2, y_2)) \geq d_H((z_1, y_1), (z_2, y_2)) - 1 \geq |S_t| = \frac{t^2+1}{2} > t$.

Case 2: $y_1 \neq y_2$ and $d_T((x_1, y_1), (x_2, y_2)) \leq d_H((z_1, y_1), (z_2, y_2)) - 2$. Without loss of generality (WLOG), we assume $x_1 \geq x_2$. Then, based on the definition of the 'removing a zigzag row', it is simple to verify that the following must be true: $d_T((x_1, y_1), (x_2, y_2)) = d_H((z_1, y_1), (z_2, y_2)) - 2$, $a_{y_2} < z_2 < z_1 < a_{y_1}$, $(z_2 - z_1 \bmod l_1) \leq (z_1 - z_2 \bmod l_1)$. By Rule 3, any vertex in the removed zigzag row is neither in the first m rows nor in the last m rows of H, so $(z_2 - z_1 \bmod l_1) \geq 2m + 3$. So $d_T((x_1, y_1), (x_2, y_2)) = d_H((z_1, y_1), (z_2, y_2)) - 2 > (z_2 - z_1 \bmod l_1) - 2 \geq 2m + 1 = t$.

Case 3: $y_1 \neq y_2$ and $d_T((x_1, y_1), (x_2, y_2)) \geq d_H((z_1, y_1), (z_2, y_2)) - 1$. We know that $d_H((z_1, y_1), (z_2, y_2)) \geq t$. So to show that $d_T((x_1, y_1), (x_2, y_2)) \geq t$, we just need to prove that if $d_H((z_1, y_1), (z_2, y_2)) = t$, then $d_T((x_1, y_1), (x_2, y_2)) \geq d_H((z_1, y_1), (z_2, y_2))$. By Lemma 3.4, there are only two non-trivial sub-cases to consider WLOG:

Sub-case 3.1: $y_2 - y_1 \equiv m \bmod l_2$, $z_2 - z_1 \equiv -(m+1) \bmod l_1$, $d_H((z_1, y_1), (z_2, y_2)) = (y_2 - y_1 \bmod l_2) + (z_1 - z_2 \bmod l_1) = t$, and s_{y_1}, $s_{(y_1+1) \bmod l_2}$, $s_{(y_1+2) \bmod l_2}$, $\cdots$, $s_{(y_1+m-1) \bmod l_2}$ do not all equal t. If $z_1 > z_2$ (which means $z_1 = z_2 + (m + 1)$), then from Rule 1, it is simple to see that $x_1 - x_2 = z_1 - z_2$ — so $d_T((x_1, y_1), (x_2, y_2)) = d_H((z_1, y_1), (z_2, y_2)) = t$. If $z_1 < z_2$ (which means that (z_1, y_1) and (z_2, y_2) are respectively in the first and last $m + 1$ rows of H), since the first and last m rows of H and T must be the same, we get that $(x_1 - x_2 \bmod l_T) = (z_1 - z_2 \bmod l_1) = m + 1$ – so $d_T((x_1, y_1), (x_2, y_2)) = d_H((z_1, y_1), (z_2, y_2)) = t$.

Sub-case 3.2: $y_2 - y_1 \equiv m + 1 \bmod l_2$, $z_2 - z_1 \equiv m \bmod l_1$, $d_H((z_1, y_1), (z_2, y_2)) = (y_2 - y_1 \bmod l_2) + (z_2 - z_1 \bmod l_1) = t$, and exactly one of s_{y_1}, $s_{(y_1+1) \bmod l_2}$, $s_{(y_1+2) \bmod l_2}$, $\cdots$, $s_{(y_1+m) \bmod l_2}$ equals $t + 1$. If $z_1 < z_2$ (which means $z_1 = z_2 - m$), then from Rule 2, it is simple to see that $x_2 - x_1 = z_2 - z_1$ — so $d_T((x_1, y_1), (x_2, y_2)) = d_H((z_1, y_1), (z_2, y_2)) = t$. If $z_1 > z_2$ (which means that (z_1, y_1) and (z_2, y_2) are respectively in the last and first m rows of H), since the first and last m rows of H and T must be the same, we get that $(x_2 - x_1 \bmod l_T) = (z_2 - z_1 \bmod l_1) = m$ — so $d_T((x_1, y_1), (x_2, y_2)) = d_H((z_1, y_1), (z_2, y_2)) = t$.

So T is t-interleaved.

(2) In this part, we will prove that G is t-interleaved. First let's have an obser-

vation: when a t-interleaved torus K is tiled with other tori vertically to get a larger torus $\hat{G}$, for any two vertices μ and ν in K (which are now also in $\hat{G}$) labelled by the same integer, the Lee distance between them in $\hat{G}$, $d_{\hat{G}}(\mu, \nu)$, is clearly no less than t. Let's also notice that the torus got by tiling one copy of B and one copy of T vertically is t-interleaved, which can be proved with exactly the same proof as in part (1).

G is got by tiling multiple copies of B and T. Let's call each copy of B or T in G a *component torus*. Let (x_1, y_1) and (x_2, y_2) be two vertices in G labelled by the same integer. Assume $d_G((x_1, y_1), (x_2, y_2)) \leq t$. Then since both B and T have more than t rows, (x_1, y_1) and (x_2, y_2) must be either in the same component torus or in two adjacent component tori. Now if (x_1, y_1) and (x_2, y_2) are in the same component torus, let K denote that component torus; if (x_1, y_1) and (x_2, y_2) are in two adjacent component tori, let K be the torus got by vertically tiling those two component tori; let $\hat{G}$ be the same as G. By using the observation in the previous paragraph, we can readily prove that $d_G((x_1, y_1), (x_2, y_2)) \geq t$. So G is t-interleaved.

□

3.4.2 Constructing the Zigzag Row

We presented three rules on devising a zigzag row in the previous subsection. But specifically, how to construct a zigzag row that follows all those rules? In this subsection, we present such constructions.

Before the formal presentation, let us go over a few concepts. An offset sequence is a string of 'P's and 'Q's, where P and Q are strings of integers depending on t. For example, when $t = 5$, $P =$'$5, 6$' and $Q =$'$5, 5, 6$'. Then an offset sequence 'PPQ' can also be written as '$5, 6, 5, 6, 5, 5, 6$'. Let's also express the offset sequence 'PPQ' as '$s_0, s_1, s_2, s_3, s_4, s_5, s_6$', where $s_0 = 5$, $s_1 = 6$, $\cdots$, $s_6 = 6$. Then for $i = 0, 1, \cdots, 6$, s_i is called the '$(i + 1)$-th element' of the offset sequence. s_2 is also called the 'first element of a P', because it is the first element of the second P in the offset sequence. For the same reason, s_0 is the first element of a P (the first P in the offset sequence),

s_1 is the second (or last) element of a P (the first P in the offset sequence), s_4 is the first element of a Q (the first/last/only Q in the offset sequence), and so on.

Now we begin the formal presentation of the constructions. Let B be an $l_0 \times l_2$ torus that is t-interleaved by Construction 3.1. (Therefore $l_0 = |S_t| + 1$.) Let H be an $l_1 \times l_2$ torus got by tiling z copies of B vertically. (Therefore $l_1 = zl_0 = z(|S_t|+1)$.) Let $S =$'$s_0, s_1, \cdots, s_{l_2-1}$' be the offset sequence utilized by Construction 3.1 when it was t-interleaving B. We say that the offset sequence S consists of p 'P's and q 'Q's, where we require $p > 0$ and $q > 0$. We require that in the offset sequence, the 'P's and 'Q's are interleaved very evenly — to be specific, in the offset sequence, between any two nearby 'P's (including between the last 'P' and the first 'P', because we see the offset sequence as being toroidal, so the last 'P' and the first 'P' are also nearby 'P's), there are either $\lceil \frac{q}{p} \rceil$ or $\lfloor \frac{q}{p} \rfloor$ consecutive 'Q's; and between any two nearby 'Q's (including between the last 'Q' and the first 'Q'), there are either $\lceil \frac{p}{q} \rceil$ or $\lfloor \frac{p}{q} \rfloor$ consecutive 'P's. Also, we require the offset sequence to start with a 'P' and to end with a 'Q'. (For example, an offset sequence consisting of 3 'P's and 5 'Q's that satisfies the above requirements is '$PQQPQQQPQ$'.) Let $m = \frac{t-1}{2}$. Let $L = m + m\lceil \frac{p}{q} \rceil$ if $p \geq q$, and let $L = m + (m-1)\lceil \frac{q}{p} \rceil$ if $p < q$. We require that $l_1 \geq (\lceil \frac{p}{q} \rceil + 1)m^2 + 2m + 1$ if $p \geq q$, and require that $l_1 \geq (\lceil \frac{q}{p} \rceil + 1)m^2 + m + (2 - \lceil \frac{q}{p} \rceil)$ if $p < q$. Below we present two constructions for constructing a zigzag row <u>in H</u>, applicable respectively when $p \geq q$ and when $p < q$. Note that the constructed zigzag row is denoted by $\{(a_0, 0), (a_1, 1), \cdots, (a_{l_2-1}, l_2 - 1)\}$. Also note that both constructions require $t > 3$. (The analysis for the case '$t = 3$', as a *somewhat* special case, will be presented in Appendix I.)

Construction 4.1: *Constructing a zigzag row in H, when t is odd, $t > 3$, and $p \geq q > 0$*

1. Let s_{x_1}, s_{x_2}, $\cdots$, $s_{x_{p+q}}$ be the integers such that $0 = x_1 < x_2 < \cdots < x_{p+q} = l_2 - m - 1$, and each s_{x_i} $(1 \leq i \leq p+q)$ is the first element of a 'P' or 'Q' in the offset sequence S.

Let $a_{x_1} = L$. For $i = 2$ to $p+q$, if $s_{x_{i-1}}$ is the first element of a 'Q', let $a_{x_i} = L$.

For $i = 2$ to $p+q$, if $s_{x_{i-1}}$ is the first element of a 'P', then let $a_{x_i} = a_{x_{i-1}} - m$.

2. For $i = 2$ to m and for $j = 1$ to $p+q$, let $a_{x_j+i-1} = a_{x_j+i-2} + L$.

3. Let s_{y_1}, s_{y_2}, $\cdots$, s_{y_q} be the integers such that $y_1 < y_2 < \cdots < y_q = l_2 - 1$, and each s_{y_i} $(1 \leq i \leq q)$ is the last element of a 'Q' in the offset sequence S.

For $i = 1$ to q, let $a_{y_i} = mL + m$.

Now we have fully determined the zigzag row, $\{(a_0, 0), (a_1, 1), \cdots, (a_{l_2-1}, l_2 - 1)\}$, in the torus H.

□

The zigzag row constructed by Construction 4.1 has a quite regular structure. We show it with an example.

Example 3.9 *We use this example to illustrate Construction 4.1. In this example,* $t = 5$*, and* B *is an* 14×18 *torus as shown in Fig. 3.11(a).* B *is* t*-interleaved by Construction 3.1 by using the offset sequence* $S =$ '$PPPQPPPQ$' $=$ '$5, 6, 5, 6, 5, 6, 5, 5, 6, 5, 6, 5, 6, 5, 6, 5, 5, 6$'*. The torus* H *is shown in Fig. 3.11(b).* H *is an* 28×18 *torus got by tiling 2 copies of* B *vertically. The rest of the parameters used by Construction 4.1 are* $p = 6$*,* $q = 2$*,* $m = 2$ *and* $L = 8$*. It is not difficult to verify that the zigzag row in* H *constructed by Construction 4.1 is* $\{(8, 0), (16, 1), (6, 2), (14, 3), (4, 4), (12, 5), (2, 6), (10, 7), (18, 8), (8, 9), (16, 10), (6, 11), (14, 12), (4, 13), (12, 14), (2, 15), (10, 16), (18, 17)\}$*. In Fig.3.11(b), the vertices in the zigzag row are shown in solid-line circles, solid-line hexagons, or dashed-line circles.*

Now we briefly analyze the structure of the zigzag row in H*. Let us write the offset sequence* S *as* $S =$ '$s_0, s_1, \cdots, s_{17}$'*. Then for* $i = 0, 1, \cdots, 17$*, we can see that* s_i *actually shows the 'offset' between the* i*-th column and the* $(i+1)$*-th column of* H *— in other words, if we shift the integers in the* i*-th column of* H *down (toroidally) by* s_i *units, we get the* $(i+1)$*-th column of* H*. So we can think of* s_i *as 'spanning from the* i*-th column to the* $(i+1)$*-th column of* H*'. And let's say a* P *or* Q *in the offset sequence spans the columns that all its elements span. Then, since the offset sequence here is '*$PPPQPPPQ$*', the ranges each of them spans is as indicated in Fig. 3.11(b).*

Let us observe the vertices in the zigzag row that are in solid-line circles. If we

indicate them by (a_{x_1}, x_1), $(a_{x_2}, x_2), \cdots, (a_{x_{p+q}}, x_{p+q})$, *where* $x_1 < x_2 \cdots < x_{p+q}$, *then we can see that* $s_{x_1}, s_{x_2}, \cdots, s_{x_{p+q}}$ *are the 'first elements' of the 'P's and 'Q's in the offset sequence (namely, each of them is the first element of a 'P' or a 'Q' in the offset sequence). And we can see that the vertices in solid-line circles have a regular structure — basically, it climbs up by* $m = 2$ *units from one vertex to the next, and drops to a base-position if it is between the spanned ranges of a* Q *and a* P. *Now let us observe the vertices in solid-line hexagons. We can see that they correspond to those 'second elements of the 'P's and 'Q's in the offset sequence', and they also have a regular structure. To be specific, the positions of the vertices in solid-line hexagons can be got by shifting the positions of the vertices in solid-line circles horizontally by 1 unit and then down by* $L = 8$ *units. In general, those vertices in a zigzag row that correspond to the* $(i+1)$*-th elements of 'P's and 'Q's can be got by shifting the positions of the vertices that correspond to the* i*-th elements of 'P's and 'Q's horizontally by 1 unit and down by* L *unit (here* $0 \leq i < m$*). As for the vertices in dashed-line circles, they correspond to the 'last elements of the 'Q's in the offset sequence', and they are all in the same row. The above observations can be extended in an obvious way to the general outputs of Construction 4.1.*

□

Now we present the second construction.

Construction 4.2: *Constructing a zigzag row in* H, *when* t *is odd,* $t > 3$, *and* $0 < p < q$

1. Let s_{x_1}, s_{x_2}, $\cdots$, $s_{x_{p+q}}$ be the integers such that $0 = x_1 < x_2 < \cdots < x_{p+q} = l_2 - m - 1$, and each s_{x_i} $(1 \leq i \leq p+q)$ is the first element of a 'P' or 'Q' in the offset sequence S.

   Let $a_{x_1} = L$.

   For $i = 2$ to $p+q$, if s_{x_i} is the first element of a 'P', let $a_{x_i} = L$; if $s_{x_{i-1}}$ is the first element of a 'P', let $a_{x_i} = L - \lceil \frac{q}{p} \rceil (m-1)$; otherwise, let $a_{x_i} = a_{x_{i-1}} + (m-1)$.

2. For $i = 2$ to m and for $j = 1$ to $p+q$, let $a_{x_j+i-1} = a_{x_j+i-2} + L$.

3. Let s_{y_1}, s_{y_2}, $\cdots$, s_{y_q} be the integers such that $y_1 < y_2 < \cdots < y_q = l_2 - 1$, and

(a)

B

0	9	3	12	6	1	9	4	13	7	2	10	5	13	8	2	11	6
1	10	4	13	7	2	10	5	0	8	3	11	6	0	9	3	12	7
2	11	5	0	8	3	11	6	1	9	4	12	7	1	10	4	13	8
3	12	6	1	9	4	12	7	2	10	5	13	8	2	11	5	0	9
4	13	7	2	10	5	13	8	3	11	6	0	9	3	12	6	1	10
5	0	8	3	11	6	0	9	4	12	7	1	10	4	13	7	2	11
6	1	9	4	12	7	1	10	5	13	8	2	11	5	0	8	3	12
7	2	10	5	13	8	2	11	6	0	9	3	12	6	1	9	4	13
8	3	11	6	0	9	3	12	7	1	10	4	13	7	2	10	5	0
9	4	12	7	1	10	4	13	8	2	11	5	0	8	3	11	6	1
10	5	13	8	2	11	5	0	9	3	12	6	1	9	4	12	7	2
11	6	0	9	3	12	6	1	10	4	13	7	2	10	5	13	8	3
12	7	1	10	4	13	7	2	11	5	0	8	3	11	6	0	9	4
13	8	2	11	5	0	8	3	12	6	1	9	4	12	7	1	10	5

(b)

H

P P P Q P P P Q

0	9	3	12	6	1	9	4	13	7	2	10	5	13	8	2	11	6
1	10	4	13	7	2	10	5	0	8	3	11	6	0	9	3	12	7
2	11	5	0	8	3	(11)	6	1	9	4	12	7	1	10	(4)	13	8
3	12	6	1	9	4	12	7	2	10	5	13	8	2	11	5	0	9
4	13	7	2	(10)	5	13	8	3	11	6	0	9	(3)	12	6	1	10
5	0	8	3	11	6	0	9	4	12	7	1	10	4	13	7	2	11
6	1	(9)	4	12	7	1	10	5	13	8	(2)	11	5	0	8	3	12
7	2	10	5	13	8	2	11	6	0	9	3	12	6	1	9	4	13
(8)	3	11	6	0	9	3	12	7	(1)	10	4	13	7	2	10	5	0
9	4	12	7	1	10	4	13	8	2	11	5	0	8	3	11	6	1
10	5	13	8	2	11	5	⟨0⟩	9	3	12	6	1	9	4	12	⟨7⟩	2
11	6	0	9	3	12	6	1	10	4	13	7	2	10	5	13	8	3
12	7	1	10	4	⟨13⟩	7	2	11	5	0	8	3	11	⟨6⟩	0	9	4
13	8	2	11	5	0	8	3	12	6	1	9	4	12	7	1	10	5
0	9	3	⟨12⟩	6	1	9	4	13	7	2	10	⟨5⟩	13	8	2	11	6
1	10	4	13	7	2	10	5	0	8	3	11	6	0	9	3	12	7
2	⟨11⟩	5	0	8	3	11	6	1	9	⟨4⟩	12	7	1	10	4	13	8
3	12	6	1	9	4	12	7	2	10	5	13	8	2	11	5	0	9
4	13	7	2	10	5	13	8	⟨3⟩	11	6	0	9	3	12	6	1	⟨10⟩
5	0	8	3	11	6	0	9	4	12	7	1	10	4	13	7	2	11
6	1	9	4	12	7	1	10	5	13	8	2	11	5	0	8	3	12
7	2	10	5	13	8	2	11	6	0	9	3	12	6	1	9	4	13
8	3	11	6	0	9	3	12	7	1	10	4	13	7	2	10	5	0
9	4	12	7	1	10	4	13	8	2	11	5	0	8	3	11	6	1
10	5	13	8	2	11	5	0	9	3	12	6	1	9	4	12	7	2
11	6	0	9	3	12	6	1	10	4	13	7	2	10	5	13	8	3
12	7	1	10	4	13	7	2	11	5	0	8	3	11	6	0	9	4
13	8	2	11	5	0	8	3	12	6	1	9	4	12	7	1	10	5

Figure 3.11: An example of Construction 4.1.

each s_{y_i} $(1 \leq i \leq q)$ is the last element of a 'Q' in the offset sequence S.

For $i = 1$ to q, let $a_{y_i} = a_{y_i - 1} + L$.

Now we have fully determined the zigzag row, $\{(a_0, 0), (a_1, 1), \cdots, (a_{l_2-1}, l_2 - 1)\}$, in the torus H.

□

Like Construction 4.1, the zigzag row constructed by Construction 4.2 also has a regular (and similar) structure.

Theorem 3.9 *The zigzag rows constructed by Construction 4.1 and Construction 4.2 follow all the three rules — Rule 1, Rule 2 and Rule 3.*

The above theorem can be proved with straightforward verification. So we skip its proof.

3.4.3 Optimal Interleaving When t Is Odd

In this subsection, we prove that when t is odd, for a torus whose size is large enough in both dimensions, its t-interleaving number is at most one more than the sphere packing lower bound, $|S_t|$. We also present the corresponding optimal t-interleaving construction.

Lemma 3.6 *In Equation Set (1) (the equation set in Construction 3.1), let the values of t, m and l_2 be fixed. Let '$p = p_0, q = q_0$' be a solution that satisfies the Equation Set (1). Then, another solution '$p = p_1, q = q_1$' also satisfies the Equation Set (1) if and only if there exists an integer c such that $p_1 = p_0 + c(m+1)(2m^2 + 2m + 2) \geq 0$ and $q_1 = q_0 - cm(2m^2 + 2m + 2) \geq 0$.*

Proof: We can easily prove that "'$p = p_1, q = q_1$' is a solution that satisfies the Equation Set (1) if $p_1 = p_0 + c(m+1)(2m^2 + 2m + 2) \geq 0$ and $q_1 = q_0 - cm(2m^2 + 2m + 2) \geq 0$ for some integer c", by plugging '$p = p_1, q = q_1$' into the Equation Set (1). Now let's prove the other direction.

Assume '$p = p_1, q = q_1$' is a solution that satisfies the Equation Set (1). Let $x = p_1 - p_0$ and $y = q_1 - q_0$. By the first equation in Equation Set (1), $p_1 m + q_1(m+1) = l_2 = p_0 m + q_0(m+1)$ — therefore $(p_1 - p_0)m = -(q_1 - q_0)(m+1)$, which is $xm = -y(m+1)$. So x is a multiple of $m+1$ and y is a multiple of m. So there exists an integer a such that $x = a(m+1)$ and $y = -am$.

Now let us look at the second equation in Equation Set (1), $p_1(2m^2 + m + 1) + q_1(2m^2 + 3m + 2) \equiv 0 \bmod (2m^2 + 2m + 2)$. Note that $2m^2 + m + 1 \equiv -(m+1) \bmod (2m^2 + 2m + 2)$ and $2m^2 + 3m + 2 \equiv m \bmod (2m^2 + 2m + 2)$. So $-p_1(m+1) + q_1 m \equiv 0 \bmod (2m^2 + 2m + 2)$. Since $p_1 = p_0 + x = p_0 + a(m+1)$ and $q_1 = q_0 + y = q_0 - am$, we get $-[p_0 + a(m+1)](m+1) + (q_0 - am)m \equiv [-p_0(m+1) + q_0 m] - [a(m+1)^2 + am^2] \equiv -a(2m^2 + 2m + 1) \equiv 0 \bmod (2m^2 + 2m + 2)$. Since $2m^2 + 2m + 1$ and $2m^2 + 2m + 2$ must be relatively prime, we get $2m^2 + 2m + 2 | a$. So there exist an integer c such that $a = c(2m^2 + 2m + 2)$. Then $p_1 = p_0 + x = p_0 + a(m+1) = p_0 + c(m+1)(2m^2 + 2m + 2) \geq 0$ and $q_1 = q_0 + y = q_0 - am = q_0 - cm(2m^2 + 2m + 2) \geq 0$.(The two inequalities come from the last condition in Equation Set (1).) That completes the proof of the other direction of this lemma.

□

Lemma 3.7 *In Equation Set (1) (the equation set in Construction 3.1), let the values of t, m and l_2 be fixed. Let $\Delta_P = (m+1)(2m^2 + 2m + 2)$ and $\Delta_Q = m(2m^2 + 2m + 2)$. If there exists a solution of p and q that satisfies the Equation Set (1), then there exists a solution '$p = p^*, q = q^*$' that satisfies not only the Equation Set (1) but also one of the following two inequalities:*

$$\frac{l_2}{2m+1} - \frac{\Delta_Q}{2} < q^* \leq p^* < \frac{l_2}{2m+1} + \frac{\Delta_P}{2} \tag{3.3}$$

$$\frac{l_2}{2m+1} - \frac{\Delta_P}{2} \leq p^* < q^* \leq \frac{l_2}{2m+1} + \frac{\Delta_Q}{2} \tag{3.4}$$

Proof: Assume there is a solution '$p = p_0, q = q_0$' that satisfies Equation Set (1). Trivially, either $p_0 \geq q_0$ or $p_0 < q_0$. Firstly, let us assume that $p_0 \geq q_0$. If $p_0 \geq$

$\frac{l_2}{2m+1} + \Delta_P$, then $q_0 = \frac{l_2 - p_0 m}{m+1} \leq \frac{l_2 - [l_2/(2m+1) + \Delta_P]m}{m+1} = \frac{l_2 - [l_2/(2m+1) + (m+1)(2m^2+2m+2)]m}{m+1} = \frac{l_2}{2m+1} - \Delta_Q$ (and vice versa) — so then by Lemma 3.6, '$p = p_0 - \Delta_P, q = q_0 + \Delta_Q$' is also a solution to Equation Set (1), and what's more, $p_0 - \Delta_P \geq \frac{l_1}{2m+1} \geq q_0 + \Delta_Q$. Based on the above observation, we can see that there must exist a solution '$p = p_1, q = q_1$' such that $\frac{l_2}{2m+1} - \Delta_Q < q_1 \leq p_1 < \frac{l_2}{2m+1} + \Delta_P$. If $p_1 < \frac{l_2}{2m+1} + \frac{\Delta_P}{2}$, then $q_1 > \frac{l_2}{2m+1} - \frac{\Delta_Q}{2}$ — then we can simply let $p^* = p_1$ and let $q^* = q_1$. If $p_1 \geq \frac{l_2}{2m+1} + \frac{\Delta_P}{2}$, then $q_1 \leq \frac{l_2}{2m+1} - \frac{\Delta_Q}{2}$ — then we will let $p^* = p_1 - \Delta_P$ and let $q^* = q_1 + \Delta_Q$, in which case we will have $\frac{l_2}{2m+1} - \frac{\Delta_P}{2} \leq p^* < \frac{l_2}{2m+1} < q^* \leq \frac{l_2}{2m+1} + \frac{\Delta_Q}{2}$. So when $p_0 \geq q_0$, this lemma holds. The case that '$p_0 < q_0$' can be analyzed similarly.

□

Theorem 3.10 *Let t be a positive odd integer. Let $m = \frac{t-1}{2}$. Define A as*

$$A = \max\{ \begin{array}{l} (\lceil \frac{l_2 + (m+1)(2m+1)(m^2+m+1)}{l_2 - m(2m+1)(m^2+m+1)} \rceil + 1)m^2 + 2m + 1, \\ (\lceil \frac{l_2 + m(2m+1)(m^2+m+1)}{l_2 - (m+1)(2m+1)(m^2+m+1)} \rceil + 1)m^2 + m + 2 - \lceil \frac{l_2 + m(2m+1)(m^2+m+1)}{l_2 - (m+1)(2m+1)(m^2+m+1)} \rceil \}. \end{array}$$

Then when

$$l_2 \geq (m+1)(2m+1)(m^2+m+1) + 1$$

and

$$l_1 \geq (2m^2+2m+1)\left(\lceil \frac{A}{2m^2+2m+2} \rceil (2m^2+2m+2) - 2 \right),$$

an $l_1 \times l_2$ (or equivalently, $l_2 \times l_1$) torus' t-interleaving number is either $|S_t|$ or $|S_t|+1$.

Proof: This theorem is trivially correct when $t = 1$. When $t = 3$, by the result of Appendix I (Theorem 3.13), we can also easily verify that this theorem is correct. So in the following analysis, we assume that $t > 3$.

Let's first define a few variables for the ease of expression. Let $\Delta_P = (m+1)(2m^2+2m+2)$, $\Delta_Q = m(2m^2+2m+2)$, $B = \frac{l_2 + (m+1)(2m+1)(m^2+m+1)}{l_2 - m(2m+1)(m^2+m+1)}$, $C = \frac{l_2 + m(2m+1)(m^2+m+1)}{l_2 - (m+1)(2m+1)(m^2+m+1)}$, $D = (\lceil B \rceil + 1)m^2 + 2m + 1$, and $E = (\lceil C \rceil + 1)m^2 + m + 2 - \lceil C \rceil$. Then clearly $A = \max\{D, E\}$.

When $l_2 \geq (m+1)(2m+1)(m^2+m+1)+1 = (m+\frac{1}{2})(m+1)(2m^2+2m+2)+1 > m(m+1)(2m^2+2m+2) = \lfloor \frac{t}{2} \rfloor (\lfloor \frac{t}{2} \rfloor + 1)(|S_t|+1)$, by Theorem 3.6, there exists at least

one solution of p and q that satisfies Equation Set (1). Then by Lemma 3.7, there exists a solution '$p = p^*, q = q^*$' to Equation Set (1) that satisfies either the condition $\frac{l_2}{2m+1} - \frac{\Delta_Q}{2} < q^* \leq p^* < \frac{l_2}{2m+1} + \frac{\Delta_P}{2}$ or the condition $\frac{l_2}{2m+1} - \frac{\Delta_P}{2} \leq p^* < q^* \leq \frac{l_2}{2m+1} + \frac{\Delta_Q}{2}$. We analyze the two cases below.

- Case 1: there is a solution '$p = p^*, q = q^*$' to Equation Set (1) that satisfies the condition $\frac{l_2}{2m+1} - \frac{\Delta_Q}{2} < q^* \leq p^* < \frac{l_2}{2m+1} + \frac{\Delta_P}{2}$. We use Construction 3.1 to t-interleave an $(|S_t|+1) \times l_2$ torus G_1. Note that when $l_2 \geq (m+1)(2m+1)(m^2+m+1)+1$, $\frac{l_2}{2m+1} - \frac{\Delta_Q}{2} > 0$, so $q^* > 0$. Also note that $\frac{p^*}{q^*} < \frac{l_2/(2m+1)+\Delta_P/2}{l_2/(2m+1)-\Delta_Q/2} = B$, so $D \geq (\lceil \frac{p^*}{q^*} \rceil + 1)m^2 + 2m + 1$. Let G_2 be an $[\lceil \frac{D}{|S_t|+1} \rceil (|S_t|+1)] \times l_2$ torus got by tiling $\lceil \frac{D}{|S_t|+1} \rceil$ copies of G_1 vertically. We use Construction 4.1 to find a zigzag row in G_2; then by removing the zigzag row in G_2, we get a torus G_3 whose size is $[\lceil \frac{D}{|S_t|+1} \rceil (|S_t|+1) - 1] \times l_2$. Clearly the number of rows in G_1, $|S_t|+1$, and the number of rows in G_3, $\lceil \frac{D}{|S_t|+1} \rceil (|S_t|+1) - 1$, are relatively prime. So for any $l_0 \times l_2$ torus G where $l_0 \geq (|S_t|+1-1)(\lceil \frac{D}{|S_t|+1} \rceil (|S_t|+1) - 1 - 1) = |S_t|(\lceil \frac{D}{|S_t|+1} \rceil (|S_t|+1) - 2)$, it can be got by tiling copies of G_1 and G_3 vertically — and by Lemma 3.5, G is t-interleaved, with the t-interleaving degree of $|S_t|+1$.

- Case 2: there is a solution '$p = p^*, q = q^*$' to Equation Set (1) that satisfies the condition $\frac{l_2}{2m+1} - \frac{\Delta_P}{2} \leq p^* < q^* \leq \frac{l_2}{2m+1} + \frac{\Delta_Q}{2}$. We use Construction 3.1 to t-interleave an $(|S_t|+1) \times l_2$ torus G_1. Note that when $l_2 \geq (m+1)(2m+1)(m^2+m+1)+1$, $\frac{l_2}{2m+1} - \frac{\Delta_P}{2} > 0$, so $p^* > 0$. Also note that $\frac{q^*}{p^*} \leq \frac{l_2/(2m+1)+\Delta_Q/2}{l_2/(2m+1)-\Delta_P/2} = C$, so $E \geq (\lceil \frac{q^*}{p^*} \rceil + 1)m^2 + m + (2 - \lceil \frac{q^*}{p^*} \rceil)$. Let G_2 be an $[\lceil \frac{E}{|S_t|+1} \rceil (|S_t|+1)] \times l_2$ torus got by tiling $\lceil \frac{E}{|S_t|+1} \rceil$ copies of G_1 vertically. We use Construction 4.2 to find a zigzag row in G_2; then by removing the zigzag row in G_2, we get a torus G_3 whose size is $[\lceil \frac{E}{|S_t|+1} \rceil (|S_t|+1) - 1] \times l_2$. Clearly the number of rows in G_1, $|S_t|+1$, and the number of rows in G_3, $\lceil \frac{E}{|S_t|+1} \rceil (|S_t|+1) - 1$, are relatively prime. So for any $l_0 \times l_2$ torus G where $l_0 \geq (|S_t|+1-1)(\lceil \frac{E}{|S_t|+1} \rceil (|S_t|+1) - 1 - 1) = |S_t|(\lceil \frac{E}{|S_t|+1} \rceil (|S_t|+1) - 2)$, it can be got by tiling copies of G_1 and G_3 vertically — and by Lemma 3.5, G is t-interleaved, with the t-interleaving degree of $|S_t|+1$.

Now let G be an $l_1 \times l_2$ torus where $l_2 \geq (m+1)(2m+1)(m^2+m+1)+1$ and $l_1 \geq (2m^2+2m+1)(\lceil \frac{A}{2m^2+2m+2} \rceil (2m^2+2m+2)-2) = |S_t|(\lceil \frac{\max\{D,E\}}{|S_t|+1} \rceil (|S_t|+1)-2)$. Based on the analysis for Case (1) and Case (2), we know that G's t-interleaving number is at most $|S_t|+1$. By the sphere packing lower bound, G's t-interleaving number is at least $|S_t|$. So G's t-interleaving number is either $|S_t|$ or $|S_t|+1$.

□

For easy reference, we show the method for optimally t-interleaving a large torus as a construction below. Note that the construction below is applicable only when $t \geq 5$ (and by default, t is odd). When $t = 1$, any torus can be t-interleaved with 1 integer in a trivial way. When $t = 3$, the torus can be t-interleaved with the construction to be presented in Appendix I.

Construction 4.3: *Optimal t-Interleaving on a Large Torus*

Input: An odd integer t such that $t \geq 5$. An integer m such that $m = \frac{t-1}{2}$. An $l_1 \times l_2$ torus, where

$$l_2 \geq (m+1)(2m+1)(m^2+m+1)+1$$

and

$$l_1 \geq (2m^2+2m+1)\left(\lceil \frac{A}{2m^2+2m+2} \rceil (2m^2+2m+2)-2\right)$$

. (The parameter A is as defined in Theorem 3.10.)

Output: An optimal t-interleaving on the $l_1 \times l_2$ torus.

Construction:

1. If both l_1 and l_2 are multiples of $|S_t|$, then the $l_1 \times l_2$ torus' t-interleaving number is $|S_t|$. In this case, we use Construction 2.2 to t-interleave the $l_1 \times l_2$ torus with $|S_t|$ distinct integers.

2. If either l_1 or l_2 is not a multiple of $|S_t|$, then the $l_1 \times l_2$ torus' t-interleaving number is $|S_t|+1$. In this case, we t-interleave the torus with $|S_t|+1$ integers in the following way: firstly, we t-interleave an $(|S_t|+1) \times l_2$ torus, B, by using Construction 3.1 (note that $|S_t|+1 = 2m^2+2m+2$); secondly, let H be an $[\lceil \frac{A}{|S_t|+1} \rceil (|S_t|+1)] \times l_2$ torus which is got by tiling $\lceil \frac{A}{|S_t|+1} \rceil$ copies of B vertically, and use Construction 4.1

or Construction 4.2 (depending on which is applicable) to find a zigzag row in H; thirdly, remove the zigzag row in H to get a $[\lceil \frac{A}{|S_t|+1} \rceil (|S_t|+1)-1] \times l_2$ torus T; finally, find non-negative integers x and y such that $l_1 = x(|S_t|+1) + y[\lceil \frac{A}{|S_t|+1} \rceil (|S_t|+1)-1]$, and get an $l_1 \times l_2$ torus by tiling x copies of B and y copies of T vertically. The resulting interleaving on the $l_1 \times l_2$ torus is a t-interleaving.

□

3.4.4 Optimal Interleaving When t Is Even

When t is even, the optimal t-interleaving on large tori can be analyzed in a very similar way as in the case of odd t. The main result for even t is shown in the following theorem. For succinctness, we leave the major steps and intermediate results of the corresponding analysis in Appendix II.

Theorem 3.11 *Let t be a positive even integer. Let $m = \frac{t}{2}$. Define A as*

$$\begin{aligned} A \;=\; \max\{\;& (\lceil \tfrac{2l_2+(m+1)(2m+1)(2m^2+1)}{2l_2-m(2m+1)(2m^2+1)} \rceil + 1)m^2 + (3 - \lceil \tfrac{2l_2+(m+1)(2m+1)(2m^2+1)}{2l_2-m(2m+1)(2m^2+1)} \rceil)m - 3, \\ & (\lceil \tfrac{2l_2+m(2m+1)(2m^2+1)}{2l_2-(m+1)(2m+1)(2m^2+1)} \rceil + 1)m^2 + (3 - \lceil \tfrac{2l_2+m(2m+1)(2m^2+1)}{2l_2-(m+1)(2m+1)(2m^2+1)} \rceil)m - 1 \\ & -2\lceil \tfrac{2l_2+m(2m+1)(2m^2+1)}{2l_2-(m+1)(2m+1)(2m^2+1)} \rceil \} \end{aligned}$$

. Then when

$$l_2 > \frac{(m+1)(2m+1)(2m^2+1)}{2}$$

and

$$l_1 \geq 2m^2 \left(\lceil \frac{A}{2m^2+1} \rceil (2m^2+1) - 2 \right)$$

, an $l_1 \times l_2$ (or equivalently, $l_2 \times l_1$) torus' t-interleaving number is either $|S_t|$ or $|S_t|+1$.

3.5 General Bounds on Interleaving Numbers

We have shown that for a torus whose size is large enough in both dimensions (Theorem 3.10 and Theorem 3.11), its t-interleaving number is at most $|S_t|+1$. If the

requirement on the torus' size is loosened to some extent (Theorem 3.8), then its t-interleaving number is at most $|S_t| + 2$. Does that mean for a torus of any size, its t-interleaving number is always at most $|S_t|$ plus a small constant? The answer is no. The following theorem shows bounds on t-interleaving numbers.

Theorem 3.12 *(1) The t-interleaving numbers of two-dimensional tori are $|S_t| + O(t^2)$ in general. And that upper bound is tight, even if the following restriction is enforced on the tori — the number of rows or the number of columns of the torus approaches infinity. (2) When both l_1 and l_2 are of the order $\Omega(t^2)$, the t-interleaving number of an $l_1 \times l_2$ torus is $|S_t| + O(t)$.*

Proof: (1) Firstly, let's show that the t-interleaving numbers of two-dimensional tori are $|S_t| + O(t^2)$ in general. Let G be an $l_1 \times l_2$ torus. First we assume that t is even and $l_1 \geq t$, $l_2 \geq t$. Let $K_1 = \lfloor \frac{l_1}{t} \rfloor$, $K_2 = \lfloor \frac{l_2}{t} \rfloor$. We see G as being tiled by small blocks in the way shown in Fig. 3.12, where the blocks are labelled by 'A' or 'B'. (Note that two blocks both labelled as 'A' are not necessary of the same size. And two blocks both labelled as 'B' are not necessary of the same size, either.) For every block labelled as 'A' (respectively, 'B'), the four blocks around it (to its left, right, up and down) are all labelled as 'B' (respectively, 'A'). Each block consists of either $\lceil \frac{l_1}{2K_1} \rceil$ or $\lfloor \frac{l_1}{2K_1} \rfloor$ rows, and either $\lceil \frac{l_2}{2K_2} \rceil$ or $\lfloor \frac{l_2}{2K_2} \rfloor$ columns. (Note that $\lceil \frac{l_1}{2K_1} \rceil = \lceil \frac{K_1 t + (l_1 \bmod t)}{2K_1} \rceil = \frac{t}{2} + \lceil \frac{l_1 \bmod t}{2K_1} \rceil$, $\lfloor \frac{l_1}{2K_1} \rfloor = \frac{t}{2} + \lfloor \frac{l_1 \bmod t}{2K_1} \rfloor$, $\lceil \frac{l_2}{2K_2} \rceil = \frac{t}{2} + \lceil \frac{l_2 \bmod t}{2K_2} \rceil$, $\lfloor \frac{l_2}{2K_2} \rfloor = \frac{t}{2} + \lfloor \frac{l_2 \bmod t}{2K_2} \rfloor$.) We see each block as a torus of its corresponding size. (So for a block whose size is $\alpha \times \beta$, it vertices are denoted by (i, j) for $i = 0, 1, \cdots, \alpha - 1$ and $j = 0, 1, \cdots, \beta$, in the same way a torus' vertices are normally denoted.) Now we interleave all the blocks following these two rules: (i) only integers in the set $\{1, 2, \cdots, \lceil \frac{l_1}{2K_1} \rceil \cdot \lceil \frac{l_2}{2K_2} \rceil\}$ are used to interleave any block 'A', and only integers in the set $\{\lceil \frac{l_1}{2K_1} \rceil \cdot \lceil \frac{l_2}{2K_2} \rceil + 1, \lceil \frac{l_1}{2K_1} \rceil \cdot \lceil \frac{l_2}{2K_2} \rceil + 2, \cdots, 2 \cdot \lceil \frac{l_1}{2K_1} \rceil \cdot \lceil \frac{l_2}{2K_2} \rceil\}$ are used to interleave any block 'B'; (ii) for all the blocks labelled by 'A' (respectively, 'B') and for any i and j, the vertices denoted by (i, j) in them (provided they exist) are all labelled by the same integer. It is very easy to see that G is t-interleaved in this way, using $2 \cdot \lceil \frac{l_1}{2K_1} \rceil \cdot \lceil \frac{l_2}{2K_2} \rceil = 2(\frac{t}{2} + \lceil \frac{l_1 \bmod t}{2K_1} \rceil)(\frac{t}{2} + \lceil \frac{l_2 \bmod t}{2K_2} \rceil) \leq 2(\frac{t}{2} + \lceil \frac{t-1}{2} \rceil)(\frac{t}{2} + \lceil \frac{t-1}{2} \rceil) = 2t^2 =$

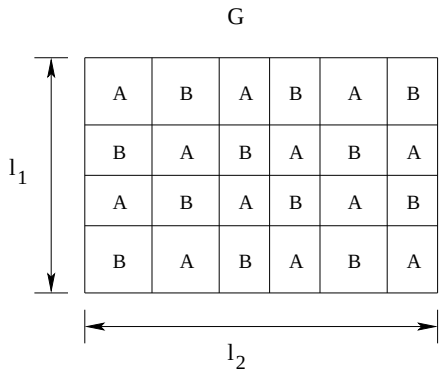


Figure 3.12: See G as being tiled by small blocks.

$|S_t| + \frac{3}{2}t^2$ distinct integers. So G's t-interleaving number is $|S_t| + O(t^2)$.

Now we assume t is even, and $l_1 < t$ or $l_2 < t$. Without loss of generality, let's say $l_1 < t$. Then we see G as being tiled horizontally by smaller tori A_1, A_2, $\cdots$, A_n, where each A_i — for $i = 1, 2, \cdots, n-1$ — is an $l_1 \times t$ torus, and A_n is an $l_1 \times (l_2 \bmod t)$ torus. We interleave A_1, A_2, $\cdots$, A_{n-1} in exactly the same way, and assign $l_1 \times t$ distinct integers to each of them. We interleave A_n with a disjoint set of $l_1 \times (l_2 \bmod t)$ integers. Clearly G is t-interleaved in this way, using $l_1 \cdot t + l_1 \cdot (l_2 \bmod t) = |S_t| + O(t^2)$ distinct integers. So again, G's t-interleaving number is $|S_t| + O(t^2)$.

Finally we assume t is odd. We can $(t+1)$-interleave G using $|S_{t+1}| + O((t+1)^2) = \frac{(t+1)^2}{2} + O((t+1)^2) = \frac{t^2+1}{2} + O(t^2) = |S_t| + O(t^2)$ distinct integers. $t+1$ is even, and a $(t+1)$-interleaving is also a t-interleaving. So G's t-interleaving number is still $|S_t| + O(t^2)$.

Now let's show that the above bound on t-interleaving numbers, $|S_t| + O(t^2)$, is tight, no matter if t is even or odd. Consider an $l_1 \times l_2$ torus where l_1 is the largest even integer that is no greater than $\lfloor \frac{3}{2}t \rfloor$, and l_2 is any integer greater than or equal to $\lfloor \frac{3}{4}t \rfloor$. We are firstly going to show that a t-interleaving can place an integer at most twice in any $\lfloor \frac{3}{4}t \rfloor$ consecutive columns of the torus.

Assume a t-interleaving places an integer on three vertices in $\lfloor \frac{3}{4}t \rfloor$ consecutive columns of the torus. Without loss of generality, let's say those three nodes are $v_{0,0}$, v_{i_1,j_1} and v_{i_2,j_2}, where $0 \le j_1 \le \lfloor \frac{3}{4}t \rfloor - 1$ and $0 \le j_2 \le \lfloor \frac{3}{4}t \rfloor - 1$. Since the interleaving

is a t-interleaving, the Lee distance between any two of those three vertices is at least t. Let $a = \frac{l_1}{2}$ and $b = \lfloor \frac{3}{4}t \rfloor - 1$. It is not difficult to see that the Lee distance between v_{i_1,j_1} and $v_{a,b}$ is at most $\min\{(a-i_1) \bmod l_1, (i_1-a) \bmod l_1\} + (b-j_1) = \frac{l_1}{2} - \min\{(0-i_1) \bmod l_1, (i_1-0) \bmod l_1\} + (b-j_1) = \frac{l_1}{2} + b - [\min\{(0-i_1) \bmod l_1, (i_1-0) \bmod l_1\} + j_1]$. Since the Lee distance between $v_{0,0}$ and v_{i_1,j_1} is at most $\min\{(0-i_1) \bmod l_1, (i_1-0) \bmod l_1\} + j_1$, we know that $\min\{(0-i_1) \bmod l_1, (i_1-0) \bmod l_1\} + j_1 \geq t$. Therefore the Lee distance between v_{i_1,j_1} and $v_{a,b}$ is at most $\frac{l_1}{2} + b - t \leq \lfloor \frac{3}{2}t \rfloor/2 + \lfloor \frac{3}{4}t \rfloor - 1 - t < \frac{t}{2}$. Similarly, the Lee distance between v_{i_2,j_2} and $v_{a,b}$ is also less than $\frac{t}{2}$. Therefore the Lee distance between v_{i_1,j_1} and v_{i_2,j_2} is less than t, which is a contradiction. So a t-interleaving cannot place an integer on more than two vertices in $\lfloor \frac{3}{4}t \rfloor$ consecutive columns of the torus.

Any $\lfloor \frac{3}{4}t \rfloor$ consecutive columns of the $l_1 \times l_2$ torus contain $l_1 \times \lfloor \frac{3}{4}t \rfloor \geq (\frac{3}{2}t-2) \times (\frac{3}{4}t-1) = \frac{9}{8}t^2 - 3t + 2$ vertices, where each integer can be placed on at most two vertices by a t-interleaving. Therefore the t-interleaving number of the torus is at least $\frac{\frac{9}{8}t^2-3t+2}{2} = \frac{9}{16}t^2 - \frac{3}{2}t + 1 = \frac{t^2+1}{2} + \frac{1}{16}t^2 - \frac{3}{2}t + \frac{1}{2} \geq |S_t| + \frac{1}{16}t^2 - \frac{3}{2}t + \frac{1}{2} = |S_t| + \Theta(t^2)$, which matches the upper bound $|S_t| + O(t^2)$. Since here l_2 can be *any* integer that is no less than $\lfloor \frac{3}{4}t \rfloor$, the upper bound is tight even if the number of columns (or equivalently, the number of rows) of the torus approaches infinity. The first part of this theorem has been proved by now.

(2) Let's prove the second part of this theorem. In the previous part of this proof, a method for t-interleaving an $l_1 \times l_2$ torus has been proposed for the case 't is even and $l_1 \geq t$, $l_2 \geq t$'. That method uses $2(\frac{t}{2} + \lceil \frac{l_1 \bmod t}{2K_1} \rceil)(\frac{t}{2} + \lceil \frac{l_2 \bmod t}{2K_2} \rceil)$ distinct integers. (Note that $K_1 = \lfloor \frac{l_1}{t} \rfloor$ and $K_2 = \lfloor \frac{l_2}{t} \rfloor$.) When both l_1 and l_2 are of the order $\Omega(t^2)$, both K_1 and K_2 are of the order of $\Omega(t)$ — and then $2(\frac{t}{2} + \lceil \frac{l_1 \bmod t}{2K_1} \rceil)(\frac{t}{2} + \lceil \frac{l_2 \bmod t}{2K_2} \rceil) = 2(\frac{t}{2} + O(1))(\frac{t}{2} + O(1)) = \frac{t^2}{2} + O(t) = |S_t| + O(t)$. When t is odd, we can t-interleave an $l_1 \times l_2$ torus, where $l_1 = \Omega(t^2) = \Omega((t+1)^2)$ and $l_2 = \Omega(t^2) = \Omega((t+1)^2)$, by $(t+1)$-interleaving it using $|S_{t+1}| + O(t+1) = \frac{(t+1)^2}{2} + O(t) = \frac{t^2+1}{2} + O(t) = |S_t| + O(t)$ distinct integers. So no matter if t is even or odd, when both l_1 and l_2 are of the order $\Omega(t^2)$, the t-interleaving number of an $l_1 \times l_2$ torus is $|S_t| + O(t)$.

□

3.6 Brief Discussions

In this chapter, we study the t-interleaving problem for two-dimensional tori. The necessary and sufficient conditions for tori that can be perfectly t-interleaved are proven, and the corresponding perfect t-interleaving construction is presented, based on the method of sphere packing. The most important contribution of this paper is to prove that for tori whose sizes are large in both dimensions, which constitute by far the majority of all existing cases, their t-interleaving numbers are at most one more than the sphere packing lower bound. Optimal t-interleaving constructions for such tori are presented, based on the method of removing-a-zigzag-row and tori-tiling. Then, some bounds on the t-interleaving numbers are shown. Those results together give a general picture for the t-interleaving problem for two-dimensional tori.

The importance of the t-interleaving method based on removing-a-zigzag-row and tori-tiling is not limited to the results in Theorem 3.10 and Theorem 3.11. Those two theorems should be seen as a lower bound for the performance of the t-interleaving method. By analyzing the performance of the corresponding t-interleaving constructions more carefully, and furthermore, by keeping the main idea of the t-interleaving method but tuning its specific parameters on a case-by-case basis, we can improve the bounds derived in Theorem 3.10 and Theorem 3.11. The content of Appendix I can serve as an example in this aspect. What's more, the t-interleaving method can be used to optimally t-interleave some tori whose sizes do not fall within the derived bounds.

3.7 Appendix I

The optimal t-interleaving construction for odd t, Construction 4.3, if applicable only when $t \geq 5$. In this appendix, we present the optimal t-interleaving construction when $t = 3$, thus completing the result for t-interleaving on large tori while t being

odd. We also use this case, $t = 3$, as an example to show how previous results can be improved if the t-interleaving problem is analyzed case by case and more carefully.

We will show that when $l_1 \geq 20$ and $l_2 \geq 15$ (or equivalently, when $l_1 \geq 15$ and $l_2 \geq 20$), an $l_1 \times l_2$ torus' 3-interleaving number is either 5 or 6. (Note that $|S_3| = 5$.) Below we present an construction that can optimally 3-interleaves any $l_1 \times l_2$ torus where $l_1 \geq 20$ and $l_2 \geq 15$, except when $l_2 = 19$.

Construction 4.4: *Optimally 3-Interleave an $l_1 \times l_2$ torus, where $l_1 \geq 20$, $l_2 \geq 15$, and $l_2 \neq 19$.*

1. If both l_1 and l_2 are multiples of 5, then the $l_1 \times l_2$ torus' 3-interleaving number is $|S_t| = 5$. In this case, 3-interleave the $l_1 \times l_2$ torus with 5 integers by using Construction 2.2.

If l_1 or l_2 is not a multiple of 5, then use the following 3 steps to 3-interleave the $l_1 \times l_2$ torus with 6 integers.

2. Find non-negative integers x_1 and x_2 such that $l_1 = 5x_1 + 6x_2$. Find non-negative integers y_1, y_2 and y_3 such that $l_2 = 5y_1 + 8y_2 + 12y_3$.

3. There are 6 tori shown in Fig. 3.13(a)— an 5×5 torus 'A', an 5×8 torus 'B', an 5×12 torus 'C', an 6×5 torus 'A'', an 6×8 torus 'B'' and an 6×12 torus 'C''.

Get a $5 \times l_2$ torus M_1 by tiling horizontally y_1 copies of 'A', y_2 copies of 'B' and y_3 copies of 'C' (whose order can be arbitrary).

Get a $6 \times l_2$ torus M_2 by tiling horizontally y_1 copies of 'A'', y_2 copies of 'B'' and y_3 copies of 'C'', whose order needs to satisfy this rule: for $i = 1$ to $y_1 + y_2 + y_3$, if the i-th module-torus in M_1 is an 'A' (respectively, a 'B' or a 'C'), then the i-th module in M_2 is an 'A'' (respectively, a 'B'' or a 'C'').

4. Get an $l_1 \times l_2$ torus by tiling x_1 copies of M_1 and x_2 copies of M_2 vertically (whose order can be arbitrary). The interleaving on the $l_1 \times l_2$ torus is a 3-interleaving.

□

Example 3.10 *We use Construction 4.4 to 3-interleave an $l_1 \times l_2$ torus where $l_1 = 11$ and $l_2 = 25$. l_1 is not a multiple of $|S_t|$, so the torus' 3-interleaving number is greater*

(a) Modules

A

0	2	4	1	3
1	3	0	2	5
2	5	1	3	0
4	0	2	5	1
5	1	3	0	2

B

0	2	4	0	3	5	1	3
1	3	5	2	4	0	2	5
2	4	1	3	5	1	4	0
3	0	2	4	0	3	5	1
5	1	3	5	2	4	0	2

C

0	2	5	1	4	0	3	5	2	4	1	3
1	3	0	2	5	1	4	0	3	5	2	4
2	5	1	4	0	3	5	2	4	1	3	0
4	0	3	5	2	4	1	3	0	2	5	1
5	1	4	0	3	5	2	4	1	3	0	2

A'

0	2	4	1	3
1	3	⑤	2	④
2	④	0	3	5
③	5	1	④	0
4	0	2	5	1
5	1	3	0	2

B'

0	2	4	0	3	5	1	3
1	3	5	①	4	0	2	④
2	4	⓪	2	5	1	③	5
3	⑤	1	3	0	②	4	0
④	0	2	4	①	3	5	1
5	1	3	5	2	4	0	2

C'

0	2	5	1	4	0	3	5	2	4	1	3
1	3	0	2	5	1	4	0	3	5	2	4
2	④	1	③	0	②	5	①	4	⓪	3	⑤
③	5	②	4	①	3	⓪	2	⑤	1	④	0
4	0	3	5	2	4	1	3	0	2	5	1
5	1	4	0	3	5	2	4	1	3	0	2

(b) Tiling of modules

0	2	4	1	3	0	2	4	0	3	5	1	3	0	2	5	1	4	0	3	5	2	4	1	3
1	3	0	2	5	1	3	5	2	4	0	2	5	1	3	0	2	5	1	4	0	3	5	2	4
2	5	1	3	0	2	4	1	3	5	1	4	0	2	5	1	4	0	3	5	2	4	1	3	0
4	0	2	5	1	3	0	2	4	0	3	5	1	4	0	3	5	2	4	1	3	0	2	5	1
5	1	3	0	2	5	1	3	5	2	4	0	2	5	1	4	0	3	5	2	4	1	3	0	2
0	2	4	1	3	0	2	4	0	3	5	1	3	0	2	5	1	4	0	3	5	2	4	1	3
1	3	5	2	4	1	3	5	1	4	0	2	4	1	3	0	2	5	1	4	0	3	5	2	4
2	4	0	3	5	2	4	0	2	5	1	3	5	2	4	1	3	0	2	5	1	4	0	3	5
3	5	1	4	0	3	5	1	3	0	2	4	0	3	5	2	4	1	3	0	2	5	1	4	0
4	0	2	5	1	4	0	2	4	1	3	5	1	4	0	3	5	2	4	1	3	0	2	5	1
5	1	3	0	2	5	1	3	5	2	4	0	2	5	1	4	0	3	5	2	4	1	3	0	2

Figure 3.13: Using modules for 3-interleaving. (a) The 6 modules; (b) Tiling the modules.

than 5. Since $l_1 = 5+6$ *and* $l_2 = 5+8+12$*, the variables in Construction 4.4 can be set as follows:* $x_1 = 1$*,* $x_2 = 1$*,* $y_1 = 1$*,* $y_2 = 1$ *and* $y_3 = 1$*. And we can let the torus* M_1 *have the form of* $[ABC]$*, and let the torus* M_2 *have the form of* $[A'B'C']$*. We then tile* M_1 *and* M_2 *to get the* $l_1 \times l_2$ *torus, which is of the form* $\begin{bmatrix} A & B & C \\ A' & B' & C' \end{bmatrix}$*. This 3-interleaved torus is shown in Fig. 3.13(b). The interleaving used* $6 = |S_3| + 1$ *integers.*

Clearly, since $25 = 5 \times 5 + 8 \times 0 + 12 \times 0$*, another choice to tile the* 11×25 *torus is* $\begin{bmatrix} A & A & A & A & A \\ A' & A' & A' & A' & A' \end{bmatrix}$*.*
□

Construction 4.4 constructs a 3-interleaved $l_1 \times l_2$ torus by tiling copies of 6 module-tori — the 6 tori shown in Fig. 3.13(a). It can be readily verified that when those 6 tori are tiled following the rule in Construction 4.4, the resulting interleaving on the $l_1 \times l_2$ torus is indeed a 3-interleaving. There are only a limited number of cases to analyze for the verification, so we skip the details. We comment that Construction 4.4 does not work for the case $l_2 = 19$, because 19 cannot be written as a linear combination of 5, 8 and 12 with non-negative coefficients — therefore an $l_1 \times 19$ torus cannot be got by tiling the module-tori. We present the construction for the case $l_2 = 19$ below.

Construction 4.5: *Optimally 3-Interleave an* $l_1 \times 19$ *torus, where* $l_1 \geq 20$*.*
Construction: Find non-negative integers x_1 and x_2 such that $l_1 = 5x_1 + 6x_2$. There are 2 tori shown in Fig. 3.14 — a 5×19 torus F and a 6×19 torus F'. Get an $l_1 \times 19$ torus by tiling x_1 copies of F and x_2 copies of F' vertically (whose order can be arbitrary). The resulting interleaving on the $l_1 \times 19$ torus is a 3-interleaving. □

The correctness of Construction 4.5 can be easily verified, so we skip the details. Based on the previous two constructions, we readily get the following conclusion for 3-interleaving.

Theorem 3.13 *When* $l_1 \geq 20$ *and* $l_2 \geq 15$*, or when* $l_1 \geq 15$ *and* $l_2 \geq 20$*, an* $l_1 \times l_2$

F

0	2	4	1	3	5	1	3	0	2	4	0	2	5	1	3	5	1	4
1	3	0	2	4	0	2	5	1	3	5	1	4	0	2	4	0	3	5
2	5	1	3	5	1	4	0	2	4	0	3	5	1	3	5	2	4	0
4	0	2	4	0	3	5	1	3	5	2	4	0	2	4	1	3	5	1
5	1	3	5	2	4	0	2	4	1	3	5	1	3	0	2	4	0	3

F'

0	2	4	⓪	3	5	1	3	⑤	2	4	0	2	④	1	3	5	1	4
1	3	⑤	1	4	0	2	④	0	3	5	1	③	5	2	4	0	②	5
2	④	0	2	5	1	③	5	1	4	0	②	4	0	3	5	①	3	0
③	5	1	3	0	②	4	0	2	5	①	3	5	1	4	⓪	2	4	1
4	0	2	4	①	3	5	1	3	⓪	2	4	0	2	⑤	1	3	5	②
5	1	3	5	2	4	0	2	4	1	3	5	1	3	0	2	4	0	3

Figure 3.14: Two modules used for 3-interleaving an $l_1 \times 19$ torus, where $l_1 \geq 20$.

torus' 3-interleaving number is either $|S_3|$ *or* $|S_3| + 1$.

We comment that the result we got here is comparatively better than the result derived in Section IV. (For example, if Theorem 3.10 is applied for the case $t = 3$, then the bound for l_2 would be 19. However here our bound for l_2 is 15.) However, we should notice that the t-interleaving method used here is the same as the method used for $t > 3$ *per se*. (We can see that the module-tori 'A', 'B', 'C' in Fig. 3.13(a) and 'F' in Fig. 3.14 are got by removing a zigzag row from 'A'', 'B'', 'C'' and 'F''. The zigzag rows are shown in circles in those two figures. Both the interleaving method here and the method in Section IV are based on torus tiling.) The improvement is made by better tuning of construction parameters and more careful analysis of the bounds. The construction used for $t = 3$ does not follow all the requirements used in Section IV. (For example, the zigzag row in Fig. 3.14 does not follow Rule 3.) In Section IV, while endeavoring to optimally tune all the parameters, we also need to ensure that the construction will work for all the cases of $t > 3$. If the interleaving problem is analyzed case by case (specifically, for each value of t, l_1 and l_2), the interleaving construction has room for further optimization.

3.8 Appendix II

In this appendix, we show how to optimally t-interleave large tori when t is even. The process is similar to the case where t is odd, differing only in details. For this reason, we just present a succinct description of the process and results. This appendix's content is parallel to that of the first three subsections of Section IV, so comparative reading should help the understanding greatly.

We assume t is even throughout the remainder of this appendix. The definitions of '*a zigzag row*' and '*removing a zigzag row*' are the same as in Definition 4.1 and 4.2.

Let B be an $l_0 \times l_2$ torus which is t-interleaved by Construction 3.1 utilizing the offset sequence $S = `s_0, s_1, \cdots, s_{l_2-1}$'. Let H be an $l_1 \times l_2$ torus got by tiling several copies of B vertically. Let $m = \frac{t}{2}$. There are four rules to follow for devising a zigzag row — denoted by $\{(a_0, 0), (a_1, 1), \cdots, (a_{l_2-1}, l_2 - 1)\}$ — in H:

- Rule 1: For any j such that $0 \leq j \leq l_2 - 1$, if the integers $s_j, s_{(j+1) \bmod l_2}, \cdots, s_{(j+m-1) \bmod l_2}$ do not all equal $t - 1$, then $a_j \geq a_{(j+m) \bmod l_2} + m - 1$.
- Rule 2: For any j such that $0 \leq j \leq l_2 - 1$, if exactly one of the integers $s_j, s_{(j+1) \bmod l_2}, \cdots, s_{(j+m) \bmod l_2}$ equals t, then $a_j \leq a_{(j+m+1) \bmod l_2} - (m - 2)$.
- Rule 3: For any j such that $0 \leq j \leq l_2 - 1$, if $s_j = t - 1$, then $a_j \leq a_{(j+1) \bmod l_2} - (2m - 2)$.
- Rule 4: For any j such that $0 \leq j \leq l_2 - 1$, $2m - 2 \leq a_j \leq l_1 - 1 - (2m - 2)$.

Lemma 3.8 *Let B be a torus t-interleaved by Construction 3.1. Let H be a torus got by tiling copies of B vertically, and let T be a torus got by removing a zigzag row in H, where the zigzag row in H follows the four rules — Rule 1, Rule 2, Rule 3 and Rule 4. Let G be a torus got by tiling copies of B and T vertically. Then, both T and G are t-interleaved.*

Now we present two constructions for finding a zigzag row, which are the counterparts of Construction 4.1 and 4.2. Let B be an $l_0 \times l_2$ torus which is t-interleaved

by Construction 3.1 utilizing the offset sequence $S = `s_0, s_1, \cdots, s_{l_2-1}$'. Let H be an $l_1 \times l_2$ torus got by tiling z copies of G vertically. We say the offset sequence S consists of p 'P's and q 'Q's, where $p > 0$ and $q > 0$. We require that in S, the 'P's and 'Q's are interleaved very evenly, and that S starts with a P and ends with a Q. Let $m = \frac{t}{2}$. Let $L = (2m-2)+(m-1)\lceil \frac{p}{q} \rceil$ if $p \geq q$, and let $L = (2m-2)+(m-2)\lceil \frac{q}{p} \rceil + 1$ if $p < q$. We require that $l_1 \geq (\lceil \frac{p}{q} \rceil + 1)m^2 + (3 - \lceil \frac{p}{q} \rceil)m - 3$ if $p \geq q$, and require that $l_1 \geq (\lceil \frac{q}{p} \rceil + 1)m^2 + (3 - \lceil \frac{q}{p} \rceil)m - (2\lceil \frac{q}{p} \rceil + 1)$ if $p < q$. Below we present two constructions for constructing a zigzag row, which is denoted by $\{(a_0, 0), (a_1, 1), \cdots, (a_{l_2-1}, l_2 - 1)\}$, in H, applicable respectively when $p \geq q$ and $p < q$.

Construction 4.6: *Constructing a zigzag row in H, when t is even, $t > 2$, and $p \geq q > 0$*

1. Let s_{x_1}, s_{x_2}, $\cdots$, $s_{x_{p+q}}$ be the integers such that $0 = x_1 < x_2 < \cdots < x_{p+q} = l_2 - m - 1$, and each s_{x_i} $(1 \leq i \leq p+q)$ is the first element of a 'P' or 'Q' in the offset sequence S.

   Let $a_{x_1} = L$. For $i = 2$ to $p+q$, if $s_{x_{i-1}}$ is the first element of a 'Q', let $a_{x_i} = L$.

   For $i = 2$ to $p+q$, if $s_{x_{i-1}}$ is the first element of a 'P', then let $a_{x_i} = a_{x_{i-1}} - (m-1)$.

2. For $i = 2$ to m and for $j = 1$ to $p+q$, let $a_{x_j+i-1} = a_{x_j+i-2} + L - m + 1$.

3. Let s_{y_1}, s_{y_2}, $\cdots$, s_{y_q} be the integers such that $y_1 < y_2 < \cdots < y_q = l_2 - 1$, and each s_{y_i} $(1 \leq i \leq q)$ is the last element of a 'Q' in the offset sequence S.

   For $i = 1$ to q, $a_{y_i} = L + (m-1)(L - m + 1) + (m-1)$.

   Now we have fully determined the zigzag row, $\{(a_0, 0), (a_1, 1), \cdots, (a_{l_2-1}, l_2 - 1)\}$, in the torus H.

□

Construction 4.7: *Constructing a zigzag row in H, when t is even, $t > 2$, and $0 < p < q$*

1. Let s_{x_1}, s_{x_2}, $\cdots$, $s_{x_{p+q}}$ be the integers such that $0 = x_1 < x_2 < \cdots < x_{p+q} = l_2 - m - 1$, and each s_{x_i} $(1 \leq i \leq p+q)$ is the first element of a 'P' or 'Q' in the offset sequence S.

Let $a_{x_1} = L$. For $i = 2$ to $p + q$, if s_{x_i} is the first element of a 'P', then let $a_{x_i} = L$; if $s_{x_{i-1}}$ is the first element of a 'P', then let $a_{x_i} = L - \lceil \frac{q}{p} \rceil (m-2) - 1$; otherwise, let $a_{x_i} = a_{x_{i-1}} + (m-2)$.

2. For $i = 2$ to m and for $j = 1$ to $p + q$, let $a_{x_j+i-1} = a_{x_j+i-2} + L - m + 1$.

3. Let $s_{y_1}, s_{y_2}, \cdots, s_{y_q}$ be the integers such that $y_1 < y_2 < \cdots < y_q = l_2 - 1$, and each s_{y_i} is the last element of a 'Q' in the offset sequence S.

For $i = 1$ to q, $a_{y_i} = a_{y_i-1} + L - m + 1$.

Now we have fully determined the zigzag row, $\{(a_0, 0), (a_1, 1), \cdots, (a_{l_2-1}, l_2 - 1)\}$, in the torus H.

□

Theorem 3.14 *The zigzag rows constructed by Construction 4.6 and Construction 4.7 follow all the four rules — Rule 1, Rule 2, Rule 3 and Rule 4.*

Lemma 3.9 *In Equation Set (2) (which is in Construction 3.1), let the values of t, m and l_2 be fixed. Let '$p = p_0, q = q_0$' be a solution that satisfies the Equation Set (2). Then, another solution '$p = p_1, q = q_1$' also satisfies the Equation Set (2) if and only if there exists an integer c such that $p_1 = p_0 + c(m+1)(2m^2+1) \geq 0$ and $q_1 = q_0 - cm(2m^2+1) \geq 0$.*

Lemma 3.10 *In Equation Set (2) (which is in Construction 3.1), let the values of t, m and l_2 be fixed. Let $\Delta_P = (m+1)(2m^2+1)$ and $\Delta_Q = m(2m^2+1)$. If there exists a solution of p and q that satisfies the Equation Set (2), then there exists a solution '$p = p^*, q = q^*$' that satisfies not only the Equation Set (2) but also one of the following two inequalities:*

$$\frac{l_2}{2m+1} - \frac{\Delta_Q}{2} < q^* \leq p^* < \frac{l_2}{2m+1} + \frac{\Delta_P}{2} \tag{3.5}$$

$$\frac{l_2}{2m+1} - \frac{\Delta_P}{2} \leq p^* < q^* \leq \frac{l_2}{2m+1} + \frac{\Delta_Q}{2} \tag{3.6}$$

Theorem 3.11 *Let t be a positive even integer. Let $m = \frac{t}{2}$. Define A as*

$$\begin{aligned} A \;=\; \max\{\; & (\lceil \tfrac{2l_2+(m+1)(2m+1)(2m^2+1)}{2l_2-m(2m+1)(2m^2+1)} \rceil + 1)m^2 + (3 - \lceil \tfrac{2l_2+(m+1)(2m+1)(2m^2+1)}{2l_2-m(2m+1)(2m^2+1)} \rceil)m - 3, \\ & (\lceil \tfrac{2l_2+m(2m+1)(2m^2+1)}{2l_2-(m+1)(2m+1)(2m^2+1)} \rceil + 1)m^2 + (3 - \lceil \tfrac{2l_2+m(2m+1)(2m^2+1)}{2l_2-(m+1)(2m+1)(2m^2+1)} \rceil)m - 1 \\ & -2\lceil \tfrac{2l_2+m(2m+1)(2m^2+1)}{2l_2-(m+1)(2m+1)(2m^2+1)} \rceil \} \end{aligned}$$

. Then when

$$l_2 > \frac{(m+1)(2m+1)(2m^2+1)}{2}$$

and

$$l_1 \geq 2m^2 \left(\lceil \frac{A}{2m^2+1} \rceil (2m^2+1) - 2 \right)$$

, an $l_1 \times l_2$ (or equivalently, $l_2 \times l_1$) torus' t-interleaving number is either $|S_t|$ or $|S_t| + 1$.

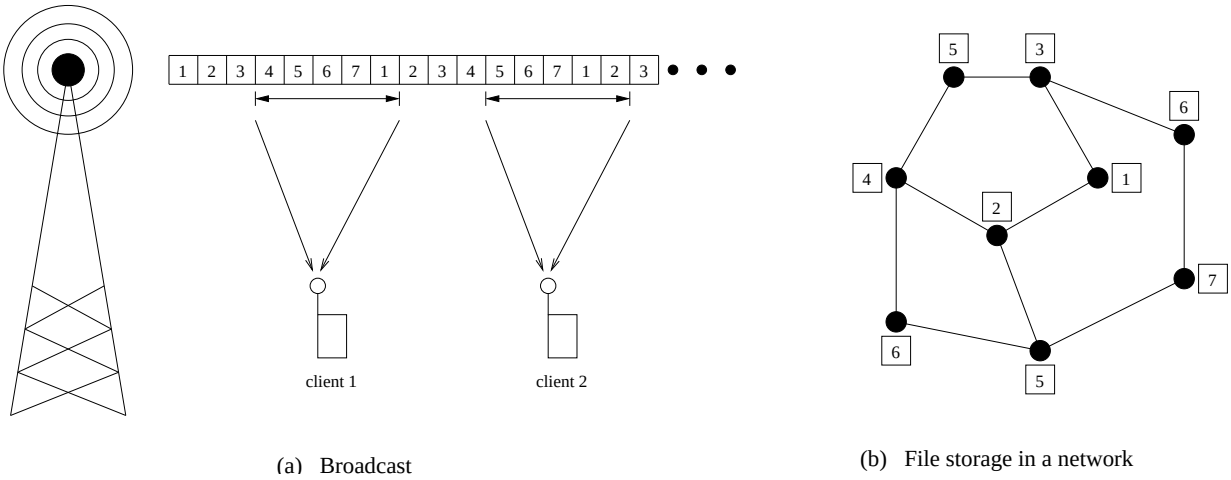
We skip the specific construction of optimally t-interleaving large tori here, because of its similarity to Construction 4.3. But we present its sketch. Basically, if the torus can be perfectly t-interleaved, then it can be optimally t-interleaved using Construction 2.2; if the torus cannot be perfectly t-interleaved and $t \geq 4$, then it can be optimally t-interleaved using the tori-tiling method. The only remaining case is 'the torus cannot be perfectly t-interleaved and $t = 2$'. In that case, we can optimally t-interleave the torus (say it is an $l_1 \times l_2$ torus) using $|S_t| + 1 = 3$ distinct integers in the following way: interleave a ring of l_1 vertices and a ring of l_2 vertices using 3 integers — 0, 1 and 2 — such that no two adjacent vertices in those two rings are assigned the same integer; for $i = 1, 2, \cdots, l_1$ (respectively, for $i = 1, 2, \cdots, l_2$), use $I(i)$ (respectively, use $J(i)$) to denote the integer assigned to the i-th vertex in the ring of l_1 (respectively, l_2) vertices; for $i = 0, 1, \cdots, l_1 - 1$ and $j = 0, 1, \cdots, l_2 - 1$, label the vertex (i, j) in the $l_1 \times l_2$ torus with the integer $(I(i+1) + J(j+1)) \bmod 3$ — and then the torus is optimally 2-interleaved.

Chapter 4

Multi-Cluster Interleaving on Paths and Cycles

4.1 Introduction

As we have discussed before, interleaving codewords is an important method not only for error-correction, but also for data storage and retrieval. For instance, data-streaming and broadcast schemes using erasure-correcting codes have received extensive interest in both academia and industry, where interleaved components of a codeword are transmitted in sequence, and every client can listen to this data flow for a while until enough codeword components are received for recovering the information [17], [54]. (An example is shown in Fig. 4.1 (a), where a codeword of 7 components is broadcast repeatedly. We assume that the codeword can tolerate 2 erasures. Therefore every client only needs to receive 5 different components.) Another instance is file storage in networks, where a file is encoded into a codeword, and components of the codeword are interleavingly placed on a network, such that every node in the network can retrieve enough distinct codeword components from its proximity for recovering the file [35], [63]. (An example is shown in Fig. 4.1 (b), where the codeword again has length 7 and can tolerate 2 erasures. We assume that all edges have length 1. Then every network node can retrieve 5 distinct codeword components from its proximity of radius 2 for recovering the file.) In all the above cases, the codeword components are interleaved on some graph structure. For exam-



(a) Broadcast

(b) File storage in a network

Figure 4.1: Examples of interleaving for data retrieving

ple, in the data streaming and broadcast case, the codeword components can be seen as interleaved on a path, because they are sequentially transmitted along the time axis. (If the sequence of data are transmitted repeatedly — e.g., using a broadcast disk — then they can be seen as interleaved on a cycle.) For the network file storage case, the codeword components are interleaved on more general graphs. We see that a client (or a network node) usually retrieves data from a connected subgraph of the graph. We call every such connected subgraph a *cluster*.

By using interleaving, the above schemes all enable 'flexible' data-retrieving, in the sense that the information contained in the interleaved data can be recovered by accessing *any* reasonably large cluster. The data-retrieving performance can be further improved if multiple clusters can be accessed in parallel. Accessing data placed in different parts of a graph in parallel has the benefits of balancing load and reducing access time, and has been studied to some extent [16], [73]. Then, it is natural to ask the following question: what is the appropriate form of interleaving for parallel data-retrieving?

If it is required that for any m ($m \geq 2$) non-overlapping clusters, the interleaved codeword components on them are all distinct, then each codeword component can be placed only once on the graph, even if m is as small as 2. Such an interleaving scheme, although minimizing the total size of the clusters that each client needs to access, is

not scalable because it requires the number of components in the codeword to equal the size of the graph, which would imply very high encoding/decoding complexity or even non-existence of the code if the graph is huge. So in an interleaving scheme for parallel data-retrieving, a tradeoff is needed between the scheme's scalability and the amount of overlapping among codeword components on different clusters.

In this chapter, we only study interleaving for parallel data-retrieving on paths and cycles, which seems to have natural applications in data-streaming and broadcasting. Imagine that codeword components interleaved the same way are transmitted asynchronously in several channels. Then a client can simultaneously listen to multiple channels in order to get data faster, which is equivalent to retrieving data from multiple clusters. Another possible application is data storage on disks, where we assume multiple heads can read different parts of a disk in parallel to accelerate I/O speed.

Now assume the codeword in consideration contains N components, and assume this codeword can be decoded by using any K ($K \leq N$) distinct codeword components. Assume every client can access m ($m \geq 2$) clusters in the path or cycle in parallel, where a cluster is defined to be a connected subgraph of the path or cycle containing L vertices. The only constraint on the m clusters a client can access is that those m clusters don't overlap each other. That constraint is for guaranteeing that each client can access no less than mL vertices. Then to enable clients to decode the codeword, there need to be at least K distinct codeword components placed on any m non-overlapping clusters. Therefore we define the following general interleaving problem for parallel data-retrieving:

Definition 4.1 *Let $G = (V, E)$ be a path (or cycle) of n vertices. Let N, K, m and L be positive integers such that $N \geq K > L$ and $m \geq 2$. A* cluster *is defined to be a connected subgraph of the path (or cycle) containing L vertices. Assign one integer in the set $\{1, 2, \cdots, N\}$ to each vertex. Such an assignment is called a* Multi-Cluster Interleaving *(MCI) if and only if any m clusters that are non-overlapping are assigned no less than K distinct integers.*

From the above definition, it can be seen that an MCI problem is fully charac-

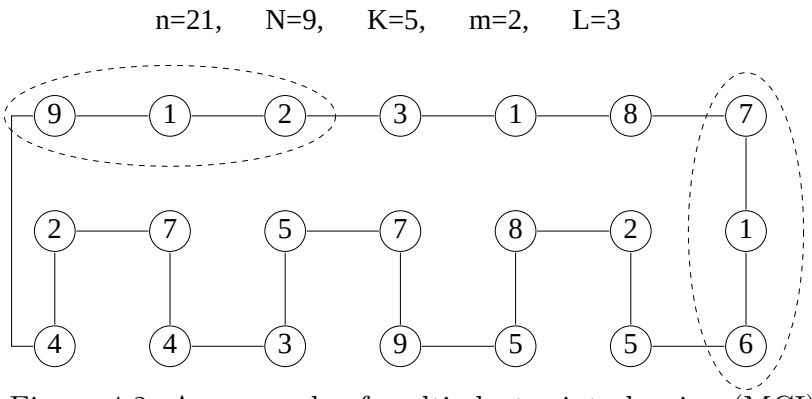


Figure 4.2: An example of multi-cluster interleaving (MCI)

terized by the five parameters n, N, K, m, L and the graph $G = (V, E)$. For the ease of explanation later on, we note that throughout this chapter, the parameters n, N, K, m, L and the graph $G = (V, E)$ will always have the meanings as defined in Definition 4.1.

Clearly if we let $m = 1$ in Definition 4.1 (and then let $K = L$), then it becomes the traditional 1-dimensional interleaving. And if an interleaving on a path (or cycle) is an MCI for some given value of m, then it is an MCI for larger values of m as well.

The following is an example of MCI.

Example 4.1 *A cycle $G = (V, E)$ of $n = 21$ vertices is shown in Fig. 4.2. The parameters are $N = 9$, $K = 5$, $m = 2$ and $L = 3$. An interleaving is shown in the figure, where the integer on every vertex is the integer assigned to it. It can be verified that any 2 clusters of size 3 that don't overlap have at least 5 distinct integers. For example, the two clusters in circle in Fig. 4.2 have integers '9, 1, 2' and '7, 1, 6' respectively, so they together have no less than 5 distinct integers. So the interleaving is a multi-cluster interleaving on the cycle G.*

If we remove an edge in the cycle, then G will become a path. Clearly if all other parameters remain the same, the interleaving shown in Fig. 4.2 will be a multi-cluster interleaving on the path.

□

The general MCI problem can be divided into smaller problems according to the

values of the parameters. The key results in this chapter are:

- The family of problems with constraints that $L = 2$ and $K = 3$ are solved for both paths and cycles. We show that when $L = 2$ and $K = 3$, an MCI exists on a path if and only if the number of vertices in the path is no greater than $(N-1)[(m-1)N-1]+2$, and an MCI exists on a cycle if and only if the number of vertices in the cycle is no greater than $(N-1)[(m-1)N-1]$. Structural properties of MCIs in this case are analyzed, and algorithms are presented which output MCIs on paths or cycles as long as the MCIs exist.

- The family of problems with the constraint that $K = L+1$ are studied for paths. A scheme using a 'hierarchical-chain' structure is presented for constructing MCI on paths. It is shown that the scheme solves the MCI problem for paths that are asymptotically as long as the longest path on which MCIs exist, and clearly, for shorter paths as well.

The rest of the chapter is organized as follows. In Section 4.2, we firstly derive an upper bound for the lengths of paths and cycles on which MCIs exist. We then prove a tighter upper bound for paths for the case of $L = 2$ and $K = 3$. In Section 4.3, we present an optimal construction for MCI on paths for the case of $L = 2$ and $K = 3$, which meets the upper bound presented in Section 4.2. In Section 4.4, we study the MCI problem for paths when $K = L + 1$. In Section 4.5 we generalize our results on MCI from paths to cycles.

4.2 Upper Bounds

While the traditional interleaving exists on infinitely long paths, that is no longer true for MCI. The following proposition derives a very simple upper bound for the lengths of paths and cycles on which MCIs exist.

Proposition 4.1 *If an MCI exists on a path (or cycle) of n vertices, then $n \leq (m-1)L\binom{N}{L} + (L-1)$.*

Proof: Let $G = (V, E)$ be a path (or cycle) of n vertices with an MCI on it. We can find at most $\lfloor \frac{n}{L} \rfloor$ non-overlapping clusters in G. Let $S \subseteq \{1, 2, \cdots, N\}$ be an arbitrary set of L distinct integers. Then since the interleaving on G is an MCI, among those $\lfloor \frac{n}{L} \rfloor$ non-overlapping clusters, at most $m-1$ of them are assigned only (all or part of the) integers in S and no integers in $\{1, 2, \cdots, N\} - S$. S can be one of $\binom{N}{L}$ possible sets. So $\lfloor \frac{n}{L} \rfloor \leq (m-1)\binom{N}{L}$. So we get $n \leq (m-1)L\binom{N}{L} + (L-1)$.

□

We comment that for the same set of parameters N, K, m and L, if an MCI exists on a path of $n = n_0$ vertices, then an MCI also exists on any path of $n < n_0$ vertices. That is because given an MCI on a path, the interleaving on the path's connected subgraph (which is a shorter path) is also an MCI for the same set of parameters N, K, m and L. However, such an argument does not necessarily hold for cycles.

The upper bound of Proposition 4.1 is in fact loose. In the remainder of this section, we shall prove a tighter upper bound for MCI on paths for the case of $L = 2$ and $K = 3$. We present it as the following theorem. Later study will show that this upper bound is exact.

Theorem 4.1 *When $L = 2$ and $K = 3$, if there exists an MCI on a path of n vertices, then $n \leq (N-1)[(m-1)N - 1] + 2$.*

Before proving the above theorem, we firstly define some notations that will be used throughout this chapter as follows. Let $G = (V, E)$ be a path. We denote the n vertices in the path G by v_1, v_2, $\cdots$, v_n. For $2 \leq i \leq n-1$, the two vertices adjacent to v_i are v_{i-1} and v_{i+1}. A connected subgraph of G induced by vertices $v_i, v_{i+1}, \cdots, v_j$ $(j \geq i)$ is denoted by $(v_i, v_{i+1}, \cdots, v_j)$. If a set of integers are interleaved on G, then $c(v_i)$ denotes the integer assigned to vertex v_i. The integers assigned to a connected subgraph of G, $(v_i, v_{i+1}, \cdots, v_j)$, are denoted by $[c(v_i) - c(v_{i+1}) - \cdots - c(v_j)]$.

The following lemma reveals a certain relationship between the structure of MCIs and the lengths of paths.

Lemma 4.1 *Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on*

a path of n vertices. Then in any MCI on a path of n_{max} vertices, no two adjacent vertices are assigned the same integer.

Proof: Let $G = (V, E)$ be a path of n_{max} vertices. Assume that there is an MCI on G. We'll prove this lemma by showing that in this MCI, if two adjacent vertices of G are assigned the same integer, then there exists a longer path that also has an MCI, which would clearly contradict the definition of n_{max}.

Assume that in the MCI on G, there are two adjacent vertices that are assigned the same integer. Then without loss of generality (WLOG), one of the following four cases must be true (because we can always get one of the four cases by permuting the names of the integers and by reversing the indices of the vertices):

Case 1: There exist 4 consecutive vertices in G — v_i, v_{i+1}, v_{i+2}, v_{i+3} — such that $c(v_i) = 1$, $c(v_{i+1}) = c(v_{i+2}) = 2$, and $c(v_{i+3}) = 1$ or 3.

Case 2: There exist $x+2 \geq 5$ consecutive vertices in G — v_i, v_{i+1}, $\cdots$, v_{i+x}, v_{i+x+1} — such that $c(v_i) = 1$, $c(v_{i+1}) = c(v_{i+2}) = \cdots = c(v_{i+x}) = 2$, and $c(v_{i+x+1}) = 1$ or 3.

Case 3: $c(v_1) = c(v_2) = 1$, and $c(v_3) = 2$.

Case 4: $c(v_1) = c(v_2) = \cdots = c(v_x) = 1$ and $c(v_{x+1}) = 2$, where $x \geq 3$.

We analyze the four cases one by one.

Case 1: In this case, we insert a vertex v' between v_{i+1} and v_{i+2}, and get a new path of $n_{max} + 1$ vertices. Call this new path H, and assign the integer '4' to v'. Consider any m non-overlapping clusters in H. If none of those m clusters contains v', then clearly they are also m non-overlapping clusters in the path G, and therefore have been assigned at least $K = 3$ distinct integers. If the m clusters contain all the three vertices v_{i+1}, v' and v_{i+2}, then either v_i or v_{i+3} is also contained in the m clusters because each cluster contains $L = 2$ vertices, and therefore the m clusters have been assigned at least $K = 3$ distinct integers: '1,2,4' or '2,3,4'. WLOG, the only remaining possibility is that one of the m clusters contains v_{i+1} and v' while none of them contains v_{i+2}. Note that among the m clusters, the $m - 1$ of them which don't contain v' are also $m - 1$ clusters in the path G, and they together with (v_{i+1}, v_{i+2}) are m non-overlapping clusters in G and therefore are assigned at least

$K = 3$ distinct integers. Since $c(v_{i+1}) = c(v_{i+2})$, the original m clusters including (v_{i+1}, v') must also have been assigned at least $K = 3$ distinct integers. Now we can conclude that the interleaving on H is also an MCI. But H's length is greater than n_{max}, which contradicts the definition of n_{max}.

Case 2: In this case, we insert a vertex v' between v_{i+1} and v_{i+2}, and insert a vertex v'' between v_{i+x-1} and v_{i+x}, and get a new path of $n_{max} + 2$ vertices. Call this new path H, assign the integer '4' to v', and assign the integer '3' to v''. Consider any m non-overlapping clusters in H. If neither v' nor v'' is contained in the m clusters, then clearly those m clusters are also m non-overlapping clusters in the path G, and therefore are assigned at least $K = 3$ distinct integers. If both v' and v'' are contained in the m clusters, then at least one vertex in the set $\{v_{i+1}, v_{i+2}, \cdots, v_{i+x-1}, v_{i+x}\}$ is also in the m clusters, and therefore the m clusters have at least these 3 integers: '2', '3' and '4'. WLOG, the only remaining possibility is that the m clusters contain v' but not v''. (Note that the cluster containing v' is assigned integers '2' and '4'.) When that possibility is true, if the m clusters contain v_{i+x+1}, then they are assigned at least 3 distinct integers — '1,2,4' or '2,3,4'; if the m clusters don't contain v_{i+x+1}, then they don't contain v_{i+x} either — then we divide the m clusters into two groups A and B, where A is the set of clusters none of which contains any vertex in $\{v', v_{i+2}, v_{i+3}, \cdots, v_{i+x-1}\}$, and B is the set of clusters none of which is in A. Say there are y clusters in B. Then, if the cluster containing v' also contains v_{i+1} (respectively, v_{i+2}), there exist a set C of y clusters in the path G that only contain vertices in $\{v_{i+1}, v_{i+2}, \cdots, v_{i+x-1}, v_{i+x}\}$ (respectively, $\{v_{i+2}, v_{i+3}, \cdots, v_{i+x-1}, v_{i+x}\}$), such that (in both cases) the m clusters in $A \cup C$ are non-overlapping in G. Those m clusters in $A \cup C$ are assigned at least $K = 3$ distinct integers since the interleaving on G is an MCI; and they are assigned no more distinct integers than the original m clusters in $A \cup B$ are, because $c(v_{i+1}) = c(v_{i+2}) = \cdots = c(v_{i+x})$ and either v_{i+1} or v_{i+2} is in the same clusters as v'. So the m clusters in $A \cup B$ are assigned at least $K = 3$ distinct integers. Now we can conclude that the interleaving on H is also an MCI. And that again contradicts the definition of n_{max}.

Case 3: In this case, we insert a vertex v' between v_1 and v_2, and assign the integer '3' to v'. The rest of the analysis is very similar to that for Case 1.

Case 4: In this case, we insert a vertex v' between v_1 and v_2, and insert a vertex v'' between v_{x-1} and v_x, assign the integer '3' to v', and assign the integer '2' to v''. The rest of the analysis is very similar to that for Case 2.

So a contradiction exists in all four cases. Therefore, this lemma is proved.

□

The next two lemmas derive upper bounds on the lengths of paths, respectively for the case of $N \geq 4$ and $N = 3$.

Lemma 4.2 *Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a path of n vertices. Then $n_{max} \leq (N-1)[(m-1)N - 1] + 2$.*

Proof: Let $G = (V, E)$ be a path of n_{max} vertices. Assume there is an MCI on G. By Lemma 4.1, no two adjacent vertices in G are assigned the same integer. We color the vertices in G with three colors — 'red', 'yellow' and 'green' — through the following three steps:

Step 1, for $2 \leq i \leq n_{max} - 1$, if $c(v_{i-1}) = c(v_{i+1})$, then color v_i with the 'red' color;

Step 2, for $2 \leq i \leq n_{max}$, color v_i with the 'yellow' color if v_i is not colored 'red' and there exists j such that these four conditions are satisfied: (1) $1 \leq j < i$, (2) v_j is not colored 'red', (3) $c(v_j) = c(v_i)$, (4) the vertices between v_j and v_i — that is, v_{j+1}, v_{j+2}, $\cdots$, v_{i-1} — are all colored 'red';

Step 3, for $1 \leq i \leq n_{max}$, if v_i is neither colored 'red' nor colored 'yellow', then color v_i with the 'green' color.

Clearly, each vertex of G is assigned exactly one of the three colors.

If we arbitrarily pick two different integers — say 'i' and 'j' — from the set $\{1, 2, \cdots, N\}$, then we get a *pair* $[i, j]$ (or $[j, i]$, equivalently). There are totally $\binom{N}{2}$ such un-ordered *pairs*. We partition those $\binom{N}{2}$ *pairs* into four groups 'A', 'B', 'C' and 'D' in the following way:

(1) A *pair* $[i, j]$ belongs to group A if and only if the following two conditions are satisfied: (i) at least one 'green' vertex is assigned the integer 'i' and at least one 'green' vertex is assigned the integer 'j', (ii) for any two 'green' vertices that are assigned integers 'i' and 'j' respectively, there is at least one 'green' vertex between them.

(2) A *pair* $[i, j]$ belongs to group B if and only if the following two conditions are satisfied: (i) at least one 'green' vertex is assigned the integer 'i' and at least one 'green' vertex is assigned the integer 'j', (ii) there exist two 'green' vertices that are assigned integers 'i' and 'j', respectively, such that there is no 'green' vertex between them.

(3) A *pair* $[i, j]$ belongs to group C if and only if one of the following two conditions is satisfied: (i) at least one 'green' vertex is assigned the integer 'i' and no 'green' vertex is assigned the integer 'j', (ii) at least one 'green' vertex is assigned the integer 'j' and no 'green' vertex is assigned the integer 'i'.

(4) A *pair* $[i, j]$ belongs to group D if and only if no 'green' vertex is assigned the integer 'i' or 'j'.

E is the set of edges in $G = (V, E)$. For any $1 \leq i \neq j \leq N$, let $E(i, j) \subseteq E$ denote such a subset of edges of G: an edge of G is in $E(i, j)$ if and only if one endpoint of the edge is assigned the integer 'i' and the other endpoint of the edge is assigned the integer 'j'. Let $z(i, j)$ denote the number of edges in $E(i, j)$. Upper bounds for $z(i, j)$ are derived below.

For any *pair* $[i, j]$ in group A or group C, $z(i, j) \leq 2m - 2$. That's because otherwise there would exist m non-overlapping clusters in G — note that a cluster contains exactly one edge — each of which is assigned only integers 'i' and 'j', which would contradict the assumption that the interleaving on G is an MCI.

Now consider a *pair* $[i, j]$ in group B. $z(i, j) \leq 2m - 2$ for the same reason as in the previous case. Assume $z(i, j) = 2m - 2$. Then in order to avoid the existence of m non-overlapping clusters in G each of which is assigned only integers 'i' and 'j', the $z(i, j) = 2m - 2$ edges in $E(i, j)$ must be consecutive in the path G, which means, WLOG, that there are $2m - 1$ consecutive vertices v_{y+1}, v_{y+2}, $\cdots$, v_{y+2m-1} $(y \geq 0)$

whose assigned integers are in the form of $[c(v_{y+1}) - c(v_{y+2}) - \cdots - c(v_{y+2m-1})] = [i - j - i - j - \cdots - i - j - i]$. According to the definition of 'group B', there exist a 'green' vertex v_{k_1} and a 'green' vertex v_{k_2}, such that v_{k_1} is assigned the integer 'i', v_{k_2} is assigned the integer 'j', and there is no 'green' vertex between them. Therefore every vertex between v_{k_1} and v_{k_2} is either 'red' or 'yellow'. By the way the vertices are colored, it's not difficult to see that either '$k_1 < k_2$ and v_{k_2-1} is assigned the integer $c(v_{k_1}) = i$' (let's call this 'case 1'), or '$k_2 < k_1$ and v_{k_1-1} is assigned the integer $c(v_{k_2}) = j$' (let's call this 'case 2'). If 'case 1' is true, then since there is an edge between v_{k_2-1} (which is assigned the integer 'i') and v_{k_2} (which is assigned the integer 'j'), v_{k_2} is in the set $\{v_{y+1}, v_{y+2}, \cdots, v_{y+2m-1}\}$. However, it's simple to see that every vertex in the set $\{v_{y+1}, v_{y+2}, \cdots, v_{y+2m-1}\}$ that is assigned the integer 'j' must have the color 'red' — so v_{k_2} must be 'red' instead of 'green' — therefore a contradiction exists. Now if 'case 2' is true, since there is an edge between v_{k_1-1} (which is assigned the integer 'j') and v_{k_1} (which is assigned the integer 'i'), both v_{k_1-1} and v_{k_1} are in the set $\{v_{y+2}, v_{y+3}, \cdots, v_{y+2m-1}\}$. Then again since every vertex in the set $\{v_{y+1}, v_{y+2}, \cdots, v_{y+2m-1}\}$ that is assigned the integer 'j' must have the color 'red', and since the color of v_{k_1} is 'green', it is simple to see that all the vertices in the set $\{v_{y+1}, v_{y+2}, \cdots, v_{k_1-1}\}$ that is assigned the integer 'i' must have the color 'red'. Then since the color of v_{y+1} is 'red', the vertex v_y exists and it must have been assigned the integer 'j' — and that contradicts the statement that all the edges in $E(i, j)$ are in the subgraph $(v_{y+1}, v_{y+2}, \cdots, v_{y+2m-1})$. Therefore a contradiction always exists when $z(i, j) = 2m - 2$. So $z(i, j) \neq 2m - 2$. So for any *pair* $[i, j]$ in group B, $z(i, j) \leq 2m - 3$.

Now consider a *pair* $[i, j]$ in group D. By the definition of 'group D', no 'green' vertex is assigned the integer 'i' or 'j'. Let $\{v_{k_1}, v_{k_2}, \cdots, v_{k_t}\}$ denote the set of vertices that are assigned the integer 'i', where $k_1 < k_2 < \cdots < k_t$. If $\{v_{k_1}, v_{k_2}, \cdots, v_{k_t}\} \neq \emptyset$, by the way vertices are colored, it's simple to see that v_{k_1} cannot be 'yellow' — so v_{k_1} must be 'red'. Then similarly, v_{k_2}, v_{k_3}, $\cdots$, v_{k_t} must be 'red', too. Therefore all the vertices that are assigned the integer 'i' are of the color 'red'. Similarly, all the vertices that are assigned the integer 'j' are of the color 'red'. Assume there is an

edge whose two endpoints are assigned the integer 'i' and the integer 'j', respectively. Then since the two vertices adjacent to a 'red' vertex must be assigned the same integer, there exists an infinitely long segment (subgraph) of the path G to which the assigned integers are in the form of '$\cdots - i - j - i - j - i - j - \cdots$', which is certainly impossible. Therefore a contradiction exists. So for any *pair* $[i, j]$ in group D, $z(i, j) = 0$.

Let 'x' denote the number of *distinct* integers assigned to 'green' vertices, and let 'X' denote the set of those x distinct integers. It's simple to see that exactly $\binom{x}{2}$ *pairs* $[i, j]$ are in group A or group B, where $i \in X$ and $j \in X$ — and among them at least $x - 1$ pairs are in group B. It's also simple to see that exactly $x(N - x)$ pairs are in group C and exactly $\binom{N-x}{2}$ pairs are in group D. By using the upper bounds we've derived for $z(i, j)$, we see that the number of edges in G is at most $[\binom{x}{2} - (x - 1)] \cdot (2m - 2) + (x - 1) \cdot (2m - 3) + x(N - x) \cdot (2m - 2) + \binom{N-x}{2} \cdot 0 = (1 - m)x^2 + (2mN - 2N - m)x + 1$, whose maximum value (at integer solutions) is achieved when $x = N - 1$ — and that maximum value is $(N - 1)[(m - 1)N - 1] + 1$. So n_{max}, the number of vertices in G, is at most $(N - 1)[(m - 1)N - 1] + 2$.

□

Lemma 4.3 *Let the values of N, K, m and L be fixed, where $N = 3$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a path of n vertices. Then $n_{max} \leq (N - 1)[(m - 1)N - 1] + 2$.*

Proof: Let $G = (V, E)$ be a path of n vertices that has an MCI on it. We need to show that $n \leq (N - 1)[(m - 1)N - 1] + 2$.

If in the MCI on G, no two adjacent vertices are assigned the same integer, then with the same argument as in the proof of Lemma 4.2, it can be shown that $n \leq (N - 1)[(m - 1)N - 1] + 2$.

Now assume there are two adjacent vertices in G that are assigned the same integer. Clearly we can find t non-overlapping clusters in G, such that $n \leq 2t + 2$ and at least one of the t clusters contains two vertices that are assigned the same integer. Among those t non-overlapping clusters, let x, y, z, a, b and c respectively

denote the number of clusters that are assigned only the integer '1', only the integer '2', only the integer '3', both the integers '1' and '2', both the integers '2' and '3', and both the integers '1' and '3'. Since the interleaving is an MCI, any m non-overlapping clusters are assigned at least $K = 3$ distinct integers. Therefore $x + y + a \leq m - 1$, $y + z + b \leq m - 1$, $z + x + c \leq m - 1$. So $2x + 2y + 2z + a + b + c \leq 3m - 3$. So $x+y+z+a+b+c \leq 3m-3-(x+y+z)$. Since $x+y+z \geq 1$, $t = x+y+z+a+b+c$, and $n \leq 2t+2$, we get $n \leq 2(x+y+z+a+b+c+1) \leq 2[3m-3-(x+y+z)+1] \leq 6m - 6 = (N-1)[(m-1)N-1]+2$.

Therefore this lemma is proved.

□

So with Lemma 4.2 and Lemma 4.3 proved, we see that Theorem 4.1 becomes a natural conclusion.

4.3 Optimal Construction for MCI on Paths with Constraints $L = 2$ and $K = 3$

In this section, we present a construction for MCIs on paths whose lengths attain the upper bound of Theorem 4.1, therefore proving the exactness of that bound. The construction is shown as the following algorithm.

Algorithm 1: *MCI on the longest path with constraints $L = 2$ and $K = 3$*
Input: Parameters N, K, m and L, where $N \geq 3$, $K = 3$, $m \geq 2$ and $L = 2$. A path $G = (V, E)$ of $n = (N-1)[(m-1)N-1]+2$ vertices.
Output: An MCI on G.
Algorithm:

Let $H = (V_H, E_H)$ be a multi-graph. The vertex set of H, V_H, is $\{u_1, u_2, \cdots, u_N\}$. For any two vertices u_i and u_j $(i \neq j)$, there are $2m-3$ edges between them if $2 \leq i = j+1 \leq N-1$ or $2 \leq j = i+1 \leq N-1$, and there are $2m-2$ edges between them otherwise. There is no loop in H. (Therefore H has exactly $n-1$ edges.)

Find a walk in H, $u_{k_1} \rightarrow u_{k_2} \rightarrow \cdots \rightarrow u_{k_n}$, that satisfies the following two

requirements: (1) the walk starts with u_1 and ends with u_{N-1} — namely, $u_{k_1} = u_1$ and $u_{k_n} = u_{N-1}$ — and passes every edge in H exactly once; (2) for any two vertices of H, the walk passes all the edges between them *consecutively*.

For $i = 1, 2, \cdots, n$, assign the integer 'k_i' to the vertex v_i in G, and we get an MCI on G.

□

Here is an example of the above algorithm.

Example 4.2 *Assume $G = (V, E)$ is a path of $n = 11$ vertices, and the parameters are $N = 4$, $K = 3$, $m = 2$ and $L = 2$. Therefore $n = (N-1)[(m-1)N - 1] + 2$. Algorithm 1 constructs a graph $H = (V_H, E_H)$, which is shown in Fig. 4.3(a). The walk in H, $u_{k_1} \rightarrow u_{k_2} \rightarrow \cdots \rightarrow u_{k_n}$, can be easily found. For example, we can let the walk be $u_1 \rightarrow u_3 \rightarrow u_1 \rightarrow u_4 \rightarrow u_1 \rightarrow u_2 \rightarrow u_4 \rightarrow u_2 \rightarrow u_3 \rightarrow u_4 \rightarrow u_3$. Corresponding to that walk, we get the interleaving on G as shown in Fig. 4.3(b). It can be easily verified that the interleaving is indeed an MCI.*

□

Theorem 4.2 *Algorithm 1 correctly outputs an MCI on the path G.*

Proof: The interleaving on G that Algorithm 1 outputs corresponds to a walk in the graph $H = (V_H, E_H)$. The N vertices of H correspond to the N integers interleaved on G. First let us assume such a walk exists and show that the interleaving on G is indeed an MCI. Every cluster in G is assigned two different integers since there is no loop in H. For any two vertices in H, there are at most $2m - 2$ edges between them, and those edges are passed consecutively in the walk. So there do not exist m or more than m non-overlapping clusters in G that are assigned the same two integers. So every m non-overlapping clusters in G are assigned at least $K = 3$ different integers. So the interleaving on G is an MCI.

Now we show that the walk in H exists. The following is a simple way to find a valid walk. Note that the graph H has the following property: only the number of edges between u_1 and u_2, or u_2 and u_3, $\cdots$, or u_{N-2} and u_{N-1}, is odd; the number

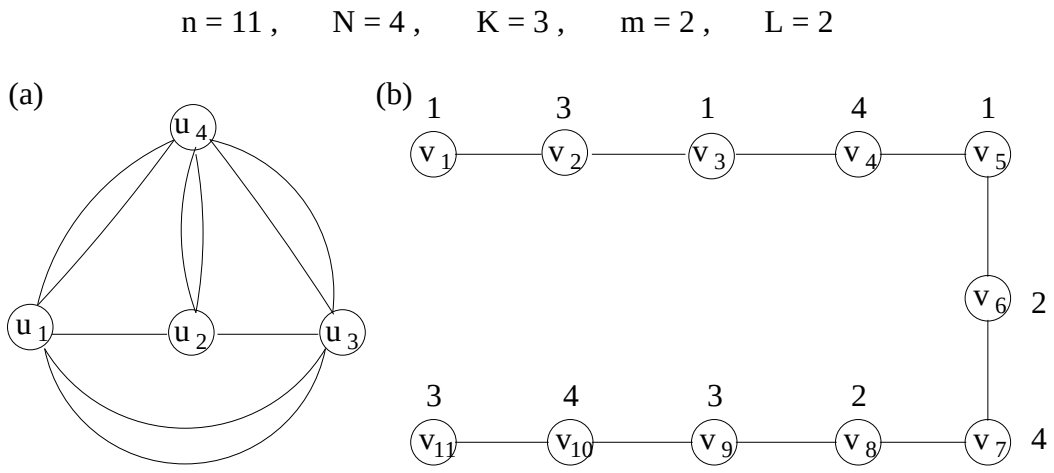


Figure 4.3: (a) The graph $H = (V_H, E_H)$ (b) MCI on the path $G = (V, E)$

of edges between any two other vertices is even. So we first get the following walk: $u_1 \rightarrow u_2 \rightarrow u_3 \rightarrow \cdots \rightarrow u_{N-2} \rightarrow u_{N-1}$. Then, for $i = 1, 2, \cdots, N-2$, replace the segment '$u_i \rightarrow u_{i+1}$' in the walk with the segment '$u_i \rightarrow u_{i+1} \rightarrow u_i \rightarrow u_{i+1} \rightarrow \cdots \rightarrow u_i \rightarrow u_{i+1}$', where in the latter segment, each of the $2m-3$ edges between u_i and u_{i+1} is passed exactly once. Next, for any vertex u_i and vertex u_j such that the edges between them have not been passed by the walk, we make the walk pass the edges between them by replacing a node 'u_i' in the walk with a segment '$u_i \rightarrow u_j \rightarrow u_i \rightarrow u_j \rightarrow \cdots \rightarrow u_i \rightarrow u_j \rightarrow u_i$', where in the segment, each of the $2m-2$ edges between u_i and u_j is passed exactly once. When doing the replacement, ensure that the edges between any other two vertices are still passed consecutively. It is simple to verify that the final walk we get satisfies all the requirements in Algorithm 1. And that completes our proof.

□

The graph $H = (V_H, E_H)$ in Algorithm 1 has either $2m-2$ or $2m-3$ edges between any two of its vertices. H has the same number of edges as the path for which the MCI is computed. If we reduce the number of edges between the vertices of H, then the walk in H will correspond to an MCI on a shorter path (since the walk will pass fewer edges). Based on this idea, we get an algorithm that can find MCIs for any path

on which MCIs exist, instead of just for the longest path. We present it as Algorithm 4, and leave it in Appendix I for simplicity. Obviously, an MCI on a short path can be got by simply taking a segment of the interleaving on the longest path computed by Algorithm 1. However such a method has some unnecessary computation and becomes inefficient when the path is substantially shorter than the longest path.

The following theorem presents the necessary and sufficient condition for there to exist an MCI on a path, when $L = 2$ and $K = 3$. Equivalently, it shows that the upper bound given in Theorem 4.1 is exact.

Theorem 4.3 *When $L = 2$ and $K = 3$, there exists an MCI on a path of n vertices if and only if $n \leq (N-1)[(m-1)N-1]+2$.*

Proof: By Theorem 4.1 and Theorem 4.2. □

4.4 MCI on Paths with Constraint $K = L + 1$

In this section we study the MCI problem on paths with the constraint that $K = L+1$. It covers the MCI problem with constraints that $L = 2$ and $K = 3$, which is studied in the previous sections, as a special case.

We define three operations on paths — 'remove a vertex,' 'insert a vertex' and 'combine two paths.' Let G be a path of n vertices: $(v_1, v_2, \cdots, v_n)$. By 'removing the vertex v_i' from G $(1 \leq i \leq n)$, we get a new path '$(v_1, v_2, \cdots, v_{i-1}, v_{i+1}, \cdots, v_n)$'. By 'inserting a vertex $\hat{v}$' in front of the vertex v_i in G $(1 \leq i \leq n)$, we get a new path '$(v_1, v_2, \cdots v_{i-1}, \hat{v}, v_i, \cdots, v_n)$'. (Similarly we can define 'inserting a vertex $\hat{v}$ behind the vertex v_i in G' and 'inserting a vertex $\hat{v}$ between the vertices v_i and v_{i+1} in G'.) Let H be a path of n' vertices: $(u_1, u_2, \cdots, u_{n'})$. Assume for $1 \leq i \leq n$, v_i is assigned the integer $c(v_i)$; and assume for $1 \leq i \leq n'$, u_i is assigned the integer $c(u_i)$. Also, let l be a positive integer between 1 and $min(n, n')$, and assume for $1 \leq i \leq l$, $c(v_i) = c(u_{n'-l+i})$. Then by saying 'combining H with G such that the last l vertices of H overlap the first l vertices of G', we mean to construct a path of $n'+n-l$ vertices whose assigned integers are $[c(u_1) - c(u_2) - \cdots - c(u_{n'}) - c(v_{l+1}) - c(v_{l+2}) - \cdots - c(v_n)]$

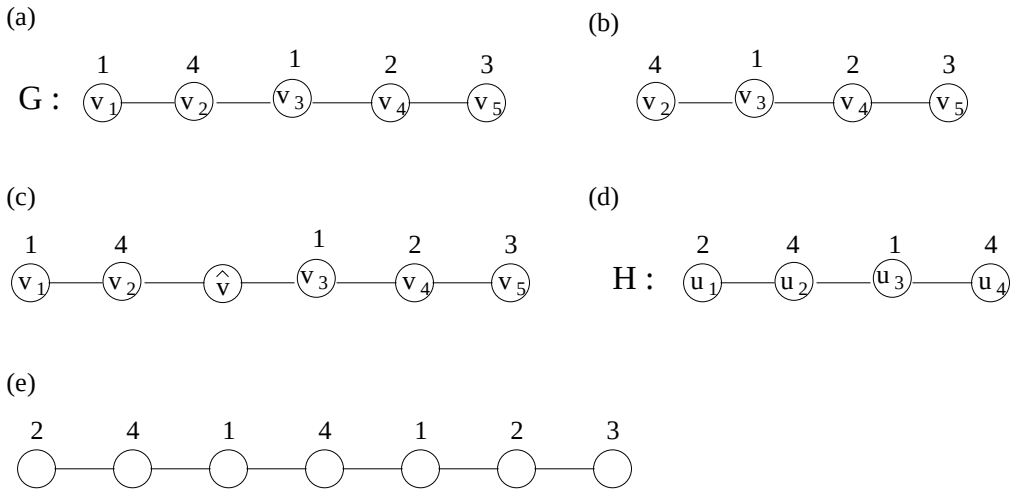


Figure 4.4: Illustrations of three operations on paths.

(which is the same as $[c(u_1) - c(u_2) - \cdots - c(u_{n'-l}) - c(v_1) - c(v_2) - \cdots - c(v_n)]$).

Examples of those operations are shown below.

Example 4.3 *Let G be a path as shown in Fig. 4.4(a), where the integer above each vertex is the integer assigned to it. (So there is an interleaving on G.) By removing the vertex v_1 from G, we get the path shown in Fig. 4.4(b). By inserting a vertex $\hat{v}$ in front of the vertex v_3 in G (or equivalently, behind the vertex v_2 in G, or between the vertex v_2 and v_3 in G), we get the path shown in Fig. 4.4(c).*

Let H be a path as shown in Fig. 4.4(d), where the integer above each vertex is the integer assigned to it. (So there is an interleaving on H, too.) By combining H with G such that the last 2 vertices of H overlap the first 2 vertices of G, we get the path shown in Fig. 4.4(e).

□

Now we present an algorithm which computes an MCI on a path. Being different from Algorithm 1, in this algorithm the length of the path is unknown. The algorithm tries to find the longest path that has an MCI, and computes an MCI for it. Thus the output of this algorithm not only provides an MCI solution, but also gives a lower bound on the maximum length of the path on which MCIs exist.

Algorithm 2: *MCI on a path with the constraint* $K = L + 1$

Input: Parameters N, K, m and L, where $N \geq K = L + 1 \geq 3$ and $m \geq 2$.

Output: An MCI on a path $G = (V, E)$ of n vertices, with the value of n as large as possible.

Algorithm:

1. If $L = 2$, then let $G = (V, E)$ be a path of $n = (N-1)[(m-1)N - 1] + 2$ vertices, and use Algorithm 1 to find an MCI on G. Output G and the MCI on it, then exit. (So step 2 to step 4 will be executed only if $L \geq 3$.)

2. Find a path B_{L+1} as long as possible that satisfies the following two conditions:

(1) Each vertex of B_{L+1} is assigned an integer in $\{1, 2, \cdots, L\}$, namely, there is an interleaving of the integers in $\{1, 2, \cdots, L\}$ on B_{L+1};

(2) Define a *segment* of a path to be a connected subgraph of the path. Then any m non-overlapping *segments* of B_{L+1} each of which contains $L - 1$ vertices are assigned at least L distinct integers.

To find the path B_{L+1}, (recursively) call Algorithm 2 in the following way: when calling Algorithm 2, replace the inputs of the algorithm — N, K, m and L — respectively with L, L, m and $L - 1$; then let the output of Algorithm 2 be the path B_{L+1} with an interleaving on it.

Scan the vertices in B_{L+1} backward (from the last vertex to the first vertex), and insert a new vertex after every $L - 1$ vertices in B_{L+1}. (In other words, if the vertices in B_{L+1} are v_1, v_2, $\cdots$, $v_{\hat{n}}$, then after inserting vertices into B_{L+1} in the way described above, we get a new path of $\hat{n} + \lfloor \frac{\hat{n}}{L-1} \rfloor$ vertices; and if we look at the new path in the reverse order — from the last vertex to the first vertex — then the path is of the form ($v_{\hat{n}}$, $v_{\hat{n}-1}$, $\cdots$, $v_{\hat{n}+1-(L-1)}$, a new vertex, $v_{\hat{n}-(L-1)}$, $v_{\hat{n}-(L-1)-1}$, $\cdots$, $v_{\hat{n}+1-2(L-1)}$, a new vertex, $v_{\hat{n}-2(L-1)}$, $v_{\hat{n}-2(L-1)-1}$, $\cdots$, $v_{\hat{n}+1-3(L-1)}$, a new vertex, $\cdots\cdots$). In this new path, every connected subgraph of L vertices contains exactly one newly inserted vertex.) Assign the integer '$L + 1$' to every newly inserted vertex in the new path, and denote this new path by 'A_{L+1}'.

3. for $i = L + 2$ to N do

{ Find a path B_i as long as possible that satisfies the following three conditions:

(1) Each vertex of B_i is assigned an integer in $\{1, 2, \cdots, i-1\}$, namely, there is an interleaving of the integers in $\{1, 2, \cdots, i-1\}$ on B_i;

(2) Define a *segment* of a path to be a connected subgraph of the path. Then any m non-overlapping *segments* of B_i each of which contains $L-1$ vertices are assigned at least L distinct integers;

(3) for $j = 1$ to $L-1$, the j-th last vertex of B_i is assigned the same integer as the $(L-j)$-th vertex of A_{i-1}.

To find the path B_i, (recursively) call Algorithm 2 in the following way: when calling Algorithm 2, replace the inputs of the algorithm — N, K, m and L — respectively with $i-1$, L, m and $L-1$; then let the output of Algorithm 2 be the path B_i with an interleaving on it.

Scan the vertices in B_i backward (from the last vertex to the first vertex), and insert a new vertex after every $L-1$ vertices in B_i. Assign the integer 'i' to every newly inserted vertex in the new path, and denote this new path by 'A_i'.

}

4. Get a new path by combining the paths A_N, A_{N-1}, $\cdots$, A_{L+1} in the following way: combine A_N with A_{N-1}, combine A_{N-1} with A_{N-2}, $\cdots$, and combine A_{L+2} with A_{L+1} such that the last $L-1$ vertices of A_N overlap the first $L-1$ vertices of A_{N-1}, the last $L-1$ vertices of A_{N-1} overlap the first $L-1$ vertices of A_{N-2}, $\cdots$, and the last $L-1$ vertices of A_{L+2} overlap the first $L-1$ vertices of A_{L+1}. (In other words, if we denote the number of vertices in A_i by l_i, for $L+1 \leq i \leq N$, then the new path we get has $\sum_{i=L+1}^{N} l_i - (L-1)(N-L-1)$ vertices.) Let this new path be $G = (V, E)$. Output G and the interleaving (which is an MCI) on it, then exit.

□

The following is an example of Algorithm 2.

Example 4.4 *In this example, the input parameters for Algorithm 2 are* $N = 6$, $K = 4$, $m = 2$ *and* $L = 3$. *That is, we use Algorithm 2 to compute an path that is as long as possible and interleave 6 integers on it, such that in the path, any 2 non-overlapping clusters of length 3 are assigned at least 4 distinct integers.*

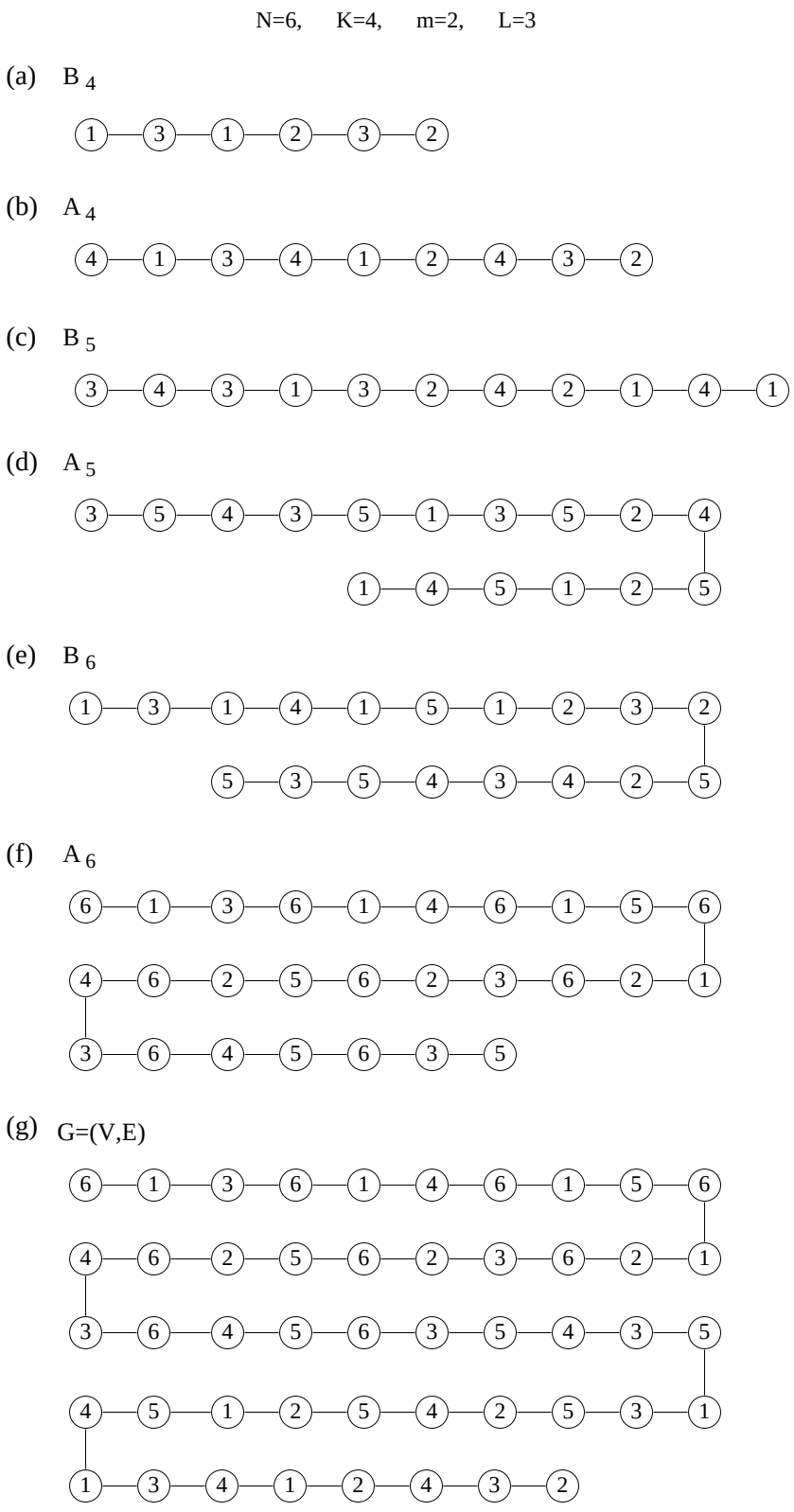


Figure 4.5: An example of Algorithm 2.

Algorithm 2 firstly computes a linear path B_4 as long as possible that satisfies the following two conditions: (1) each vertex of B_4 is assigned an integer in $\{1, 2, 3\}$; (2) any $m = 2$ non-overlapping segments *of B_4 each of which contains $L - 1 = 2$ vertices are assigned at least $L = 3$ distinct integers. To compute B_4, Algorithm 2 calls itself in a recursive way, by setting the inputs of the algorithm — N, K, m and L — to be 3, 3, 2 and 2; during that call, it uses Algorithm 1 to compute B_4. There are more than 1 possible outcomes of Algorithm 1; without loss of generality (WLOG), let us assume the output here is that B_4 is assigned integers in the form of $[1-3-1-2-3-2]$. The path B_4 is shown in Figure 4.5 (a).*

Algorithm 2 then scans B_4 backward, inserts a new vertex into B_4 after every $L - 1 = 2$ vertices, and assigns the integer '4' to every newly inserted vertex. As a result, we get a path whose assigned integers are in the form of $[4-1-3-4-1-2-4-3-2]$. We call this new path A_4. A_4 is shown in Figure 4.5 (b).

Algorithm 2 then computes a path B_5 as long as possible that satisfies the following three conditions: (1) each vertex of B_5 is assigned an integer in $\{1, 2, 3, 4\}$; (2) any $m = 2$ non-overlapping segments *of B_5 each of which contains $L - 1 = 2$ vertices are assigned at least $L = 3$ distinct integers; (3) the last vertex of B_5 is assigned the same integer as the 2nd vertex of A_4 (which is the integer '1'), and the 2nd last vertex of B_5 is assigned the same integer as the 1st vertex of A_4 (which is the integer '4').*

Algorithm 2 computes B_5 by once again calling itself. Algorithm 2 can use the following method to find a path that satisfies all the above 3 conditions. Firstly, use Algorithm 1 to find a path that satisfies the first 2 conditions, which is easy, and call this path C_5. All the integers assigned to C_5 are in the set $\{1, 2, 3, 4\}$; and from Algorithm 1, it is simple to see that the last two vertices in C_5 are assigned two different integers. (Note that the first two vertices in A_4 are also assigned two different integers.) So by permuting the names of the integers assigned to C_5, we can get a path that satisfies not only the first 2 conditions but also the 3rd condition. Call this path B_5. There are more than 1 possible result of B_5. WLOG, we assume the integers assigned to B_5 are in the form of $[3-4-3-1-3-2-4-2-1-4-1]$. B_5 is shown in Figure 4.5 (c). Then Algorithm 2 inserts vertices into B_5 and gets a

new path A_5, whose assigned integers are in the form of $[3-5-4-3-5-1-3-5-2-4-5-2-1-5-4-1]$. *$A_5$ is shown in Figure 4.5 (d).*

Next, Algorithm 2 computes a path B_6, by calling itself again. WLOG, we assume the integers assigned to B_6 are in the form of $[1-3-1-4-1-5-1-2-3-2-5-2-4-3-4-5-3-5]$. *$B_6$ is shown in Figure 4.5 (e). Then Algorithm 2 inserts vertices into B_6 and gets a new path A_6, whose assigned integers are in the form of* $[6-1-3-6-1-4-6-1-5-6-1-2-6-3-2-6-5-2-6-4-3-6-4-5-6-3-5]$. *$A_6$ is shown in Figure 4.5 (f).*

Finally, Algorithm 2 combines A_6, A_5 and A_4 such that the last $L-1=2$ vertices of A_6 overlap the first 2 vertices of A_5, and the last $L-1=2$ vertices of A_5 overlap the first 2 vertices of A_4. As a result, we get a path $G=(V,E)$ of 48 vertices which is assigned the integers $[6-1-3-6-1-4-6-1-5-6-1-2-6-3-2-6-5-2-6-4-3-6-4-5-6-3-5-4-3-5-1-3-5-2-4-5-2-1-5-4-1-3-4-1-2-4-3-2]$. *$G$ is shown in Figure 4.5 (g). This is the output of Algorithm 2. It can be verified that the interleaving on G is indeed an MCI.*

□

Algorithm 2 outputs a path G, which is as long as the algorithm can find, and an MCI on G. The MCI on G has a 'hierarchical-chain' structure, because G is a chain of the sub-paths A_{L+1}, A_{L+2}, $\cdots$, A_N, and these sub-paths form the horizontal hierarchy because on them more and more integers are interleaved and they have increasing lengths. Each A_i $(L+1 \leq i \leq N)$ is derived from a path B_i. Since B_i is got by recursively calling Algorithm 2, it also is a chain of its own sub-paths — and the same analysis can be performed for the sub-paths in B_i $\cdots\cdots$ So such sub-paths form the vertical hierarchy in the MCI. G's length, n, is unknown before Algorithm 2 ends. But if we can use n to evaluate the complexity of Algorithm 2, then Algorithm 2 can be easily seen to have complexity $O(n)$. Algorithm 2 constructs the path G piece by piece. So it is simple to see that the algorithm can be easily modified to efficiently compute MCIs on any path of less than n vertices.

Below we prove the correctness of Algorithm 2.

Theorem 4.4 *Algorithm 2 is correct.*

Proof: We prove this theorem by induction. If $L = 2$, then Algorithm 2 uses Algorithm 1 to compute the MCI — so the result is clearly correct. Also, we notice that for any MCI output by Algorithm 1, any two adjacent vertices are assigned different integers. We use those as the base case.

Let I be an integer such that $2 < I \leq L$. Let's assume the following statement is true: if we replace the inputs of Algorithm 2 — parameters N, K, m and L — with any other set of valid inputs $\hat{N}, \hat{K} = i + 1, \hat{m}$ and i such that $2 \leq i < I$, Algorithm 2 will correctly output an MCI on a path; and in that MCI, any i consecutive vertices are assigned i different integers. (This is our induction assumption.)

Now let's replace the inputs of Algorithm 2 — parameters N, K, m and L — with a set of valid inputs $N', K' = I + 1, m'$ and I. Then Algorithm 2 needs to compute (in its step 2 and step 3) $N' - I$ paths: B_{I+1}, B_{I+2}, $\cdots$, $B_{N'}$. For $I + 1 \leq j \leq N'$, B_j is (recursively) computed by calling Algorithm 2. The interleaving on B_j is in fact an MCI where the size of each cluster is $I - 1$ — so by the induction assumption, Algorithm 2 will correctly output the interleaving on B_j. B_j is assigned the integers in $\{1, 2, \cdots, j-1\}$; and by the induction assumption, any $I-1$ consecutive vertices in B_j are assigned $I-1$ different integers. The path A_{I+1} is constructed by inserting vertices into B_{I+1} such that every I consecutive vertices in A_{I+1} contains exactly one newly inserted vertex, and all the newly inserted vertices are assigned the integer '$I + 1$'. So any I consecutive vertices in A_{I+1} are assigned I different integers. Therefore it is always feasible to adjust the interleaving on B_{I+2} to make the last $I - 1$ vertices of B_{I+2} be assigned the same integers as the first $I - 1$ vertices of A_{I+1}. Noticing that the last $I - 1$ vertices of B_{I+2} are assigned the same integers as the last $I - 1$ vertices of A_{I+2}, we see that A_{I+2} and A_{I+1} can be successfully combined with $I - 1$ overlapping vertices by Algorithm 2. Similarly, for $I + 3 \leq t \leq N'$, A_t and A_{t-1} can be successfully combined by Algorithm 2; and for $I + 2 \leq t \leq N'$, any I consecutive vertices in A_t are assigned I different integers. Algorithm 2 uses G to denote the path got by combining A_{L+1}, A_{L+2}, $\cdots$, A_N. Clearly any I consecutive vertices in

G are also I consecutive vertices in A_j for some j $(I+1 \leq j \leq N')$, therefore are assigned I different integers. And for any m' non-overlapping clusters in G — each cluster here contains I vertices — either they are all contained in A_j for some j $(I+1 \leq j \leq N')$, or at least one cluster is contained in $A_{j'}$ for some j' and one other cluster is contained in $A_{j''}$ for some $j'' \neq j'$ $(I+1 \leq j', j'' \leq N')$. In the former case, by removing those vertices that are assigned the integer 'j' in those m' clusters, we get m' non-overlapping connected subgraphs in B_j each of which contains $I-1$ vertices, which are assigned at least I different integers not including 'j' — so the m' clusters in G (which are also in A_j) are assigned at least $I+1$ different integers. In the latter case, without loss of generality, let's say $j' < j''$. Then the cluster in $A_{j'}$ are assigned I different integers not including 'j''', and the cluster in $A_{j''}$ is assigned an integer 'j''' — so the m' clusters in G are assigned at least $I+1$ different integers. Therefore the interleaving on G is an MCI (with parameters N', K', m' and I). So the induction assumption also holds when $i = I$.

Algorithm 2 computes the result for the original problem by recursively calling itself. By the above induction, every intermediate time Algorithm 2 is called, the output is correct. So the final output of Algorithm 2 is also correct.

□

The length of the longest path on which an MCI exists increases when N, the number of integers that are interleaved, increases. The performance of Algorithm 2 can be evaluated by the difference between the length of the path constructed by Algorithm 2 and the length of the longest path on which an MCI exists. We're interested in studying how the difference goes when N increases. The following theorem shows the result.

Theorem 4.5 *Fix the values of the parameters K, m and L, where $K = L+1 \geq 3$ and $m \geq 2$, and let N be a variable ($N \geq K$). Then the longest path on which an MCI exists has $\frac{m-1}{(L-1)!}N^L + O(N^{L-1})$ vertices. And the path output by Algorithm 2 also has $\frac{m-1}{(L-1)!}N^L + O(N^{L-1})$ vertices.*

Proof: Let $G = (V, E)$ be a path of n vertices with an MCI on it. Then by Proposi-

tion 4.1, $n \leq (m-1)L\binom{N}{L} + (L-1)$. So $n \leq \frac{m-1}{(L-1)!}N^L + O(N^{L-1})$.

When $L = 2$, Algorithm 2 outputs a path of $(N-1)[(m-1)N-1]+2$ vertices. When $L \geq 3$, to get the output, Algorithm 2 needs to construct the paths A_{L+1}, A_{L+2}, $\cdots$, A_N; and for $L+1 \leq i \leq N$, A_i is got by inserting vertices into the path B_i. B_i is again an output of Algorithm 2, which is assigned $i-1$ distinct integers, and in which an corresponding 'cluster' has $L-1$ vertices. Let's use $F(N,m,L)$ to denote the number of vertices in the path output by Algorithm 2, and use $A(i,m,L)$ to denote the number of vertices in the path A_i. Then based on the above observed relations, we get the following 3 equations:

(1) $F(N,m,2) = (N-1)[(m-1)N-1]+2$;

(2) when $L \geq 3$, $F(N,m,L) = \sum_{i=L+1}^{N} A(i,m,L) - (N-L-1)(L-1)$;

(3) when $i \geq L+1 \geq 4$, $A(i,m,L) = \lfloor \frac{L}{L-1} \cdot F(i-1,m,L-1) \rfloor$. (Note that $F(i-1,m,L-1)$ is the number of vertices in the path B_i.)

By solving the above equations, we get $F(N,m,L) = \frac{m-1}{(L-1)!}N^L + O(N^{L-1})$, which meets the upper bound on the path's length we've derived. So the longest path on which an MCI exists has $\frac{m-1}{(L-1)!}N^L + O(N^{L-1})$ vertices; and the path output by Algorithm 2 also has $\frac{m-1}{(L-1)!}N^L + O(N^{L-1})$ vertices.

□

Theorem 4.5 shows that the path output by Algorithm 2 is asymptotically as long as the longest path on which an MCI exists. What's more, the lengths of those two paths have the same highest-degree term (in N).

We end this part of discussion with some numerical results which are representative. In Table 1, the length of the path output by Algorithm 2 (denoted by 'Output of Algorithm 2' in the table) is compared with the upper bound on path lengths of Theorem 4.1 (denoted by 'Upper bound' in the table), for four different sets of parameters — $m = 2$ and $L = 3$, $m = 2$ and $L = 5$, $m = 5$ and $L = 3$, and $m = 5$ and $L = 5$. (Note that here $K = L+1$, so the value of parameter K is determined by L. The length of a path is defined to be the number of vertices in it.) If we use n to denote the length of the path output by Algorithm 2, and use U_{bound} to denote the upper bound of Theorem 4.1, then the 'relative difference' in the table is defined to

be $\frac{U_{bound}-n}{U_{bound}} = 1 - \frac{n}{U_{bound}}$. Table 1 shows how the 'relative difference' decreases when N increases. With Theorem 4.5, we can prove that as N approaches $+\infty$, the 'relative difference' becomes arbitrarily close to 0 — the optimal value. We comment that U_{bound} is a quite loose upper bound, so the relative difference between the length of the path output by Algorithm 2 and the true length of the longest path on which an MCI exists is even smaller than that shown in the table.

	$m = 2$ and $L = 3$			$m = 2$ and $L = 5$		
N	Output of Algorithm 2	Upper bound	Relative difference	Output of Algorithm 2	Upper bound	Relative difference
10	312	362	0.1381	930	1264	0.2642
20	3177	3422	0.0716	68265	77524	0.1194
50	57072	58802	0.0294	10081020	10593804	0.0484
100	477897	485102	0.0149	367196445	376437604	0.0245
150	1637472	1653902	0.0099	2.9093×10^9	2.9580×10^9	0.0165
200	3910797	3940202	0.0075	1.2521×10^{10}	1.2678×10^{10}	0.0124

	$m = 5$ and $L = 3$			$m = 5$ and $L = 5$		
N	Output of Algorithm 2	Upper bound	Relative difference	Output of Algorithm 2	Upper bound	Relative difference
10	1383	1442	0.0409	4395	5044	0.1287
20	13428	13682	0.0186	298785	310084	0.0364
50	233463	235202	0.0074	41846205	42375204	0.0125
100	1933188	1940402	0.0037	1.4964×10^9	1.5058×10^9	0.0062
150	6599163	6615602	0.0025	1.1783×10^{10}	1.1832×10^{10}	0.0041
200	15731388	15760802	0.0019	5.0556×10^{10}	5.0713×10^{10}	0.0031

Table 1: Comparison between the length of the path output by Algorithm 2 and an upper bound, and their relative difference.

4.5 MCI on Cycles

In this section, we generalize our results on MCI from paths to cycles, for the case of $L = 2$ and $K = 3$. The analysis for the two kinds of graphs bears plenty of similarity, even though the 'circular' structure of the cycle leads to certain difference sometimes.

Let $G = (V, E)$ be a cycle. The following notations will be used in this section and the appendices of this chapter. We denote the n vertices in the cycle $G = (V, E)$ by v_1, $v_2, \cdots, v_n$. For $2 \leq i \leq n-1$, the two vertices adjacent to v_i are v_{i-1} and v_{i+1}. Vertex v_1 and v_n are adjacent to each other. A connected subgraph of G induced by vertices $v_i, v_{i+1}, \cdots, v_j$ is denoted by $(v_i, v_{i+1}, \cdots, v_j)$. If a set of integers are interleaved on G, then $c(v_i)$ denotes the integer assigned to vertex v_i. The integers assigned to a connected subgraph of G, $(v_i, v_{i+1}, \cdots, v_j)$, are denoted by $[c(v_i)-c(v_{i+1})-\cdots-c(v_j)]$.

The following three lemmas reveal some structural properties of MCI on cycles and present an upper bound on the cycles' lengths.

Lemma 4.4 *Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a cycle of n vertices. Then in any MCI on a cycle $G = (V, E)$ of n_{max} vertices, no two adjacent vertices are assigned the same integer.*

Lemma 4.5 *Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a cycle of n vertices. Then $n_{max} \leq (N-1)[(m-1)N-1]$.*

Lemma 4.6 *Let the values of N, K, m and L be fixed, where $N = 3$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a cycle of n vertices. Then $n_{max} \leq (N-1)[(m-1)N-1]$.*

The techniques used to prove the above three lemmas are similar to the proofs of Lemma 4.1, 4.2 and 4.3 (but also have difference, especially for Lemma 4.5 and Lemma 4.6). For simplicity, we present the proofs of the above three lemmas in Appendix II.

Below we present an algorithm for finding MCIs on cycles. Note that a Eulerian walk in a graph is a closed walk that passes every edge of the graph exactly once.

Algorithm 3: *MCI on a cycle with constraints* $L = 2$ *and* $K = 3$

Input: A cycle $G = (V, E)$ of n vertices. Parameters N, K, m and L, where $N \geq 3$, $K = 3$, $m \geq 2$ and $L = 2$.

Output: An MCI on G.

Algorithm:

1. If $n > (N-1)[(m-1)N-1]$, there does not exist an MCI on G, so exit the algorithm.

2. If $n \leq N$, arbitrarily select n integers in the set $\{1, 2, \cdots, N\}$, and assign one distinct integer to each vertex, then exit the algorithm.

3. If $N < n \leq (N-1)[(m-1)N-1]$ and $n - \{(N-1)[(m-1)N-1]\}$ is even, then let $H = (V_H, E_H)$ be such a multi-graph: its vertex set $V_H = \{u_1, u_2, \cdots, u_N\}$; for any $1 \leq i < j \leq N$, there are $x_{i,j}$ undirected edges between u_i and u_j; there is no loop in H. The integers $x_{i,j}$ $(1 \leq i < j \leq N)$ satisfy the following four requirements:

(1) for $1 \leq i \leq N-1$, $x_{i,N}$ is even and $0 \leq x_{i,N} \leq 2m-2$;

(2) for $1 \leq i \leq N-2$ and $j = i+1$, $x_{i,j}$ is odd and $1 \leq x_{i,j} \leq 2m-3$; also, $x_{1,N-1}$ is odd and $1 \leq x_{1,N-1} \leq 2m-3$;

(3) for $1 \leq i \leq N-4$ and $i+2 \leq j \leq N-2$, $x_{i,j}$ is even and $0 \leq x_{i,j} \leq 2m-2$; also, for $2 \leq i \leq N-3$ and $j = N-1$, $x_{i,j}$ is even and $0 \leq x_{i,j} \leq 2m-2$;

(4) if we define S as $S = \{x_{i,j} | i = 1, 2, \cdots, N; j = 1, 2, \cdots, N; i < j\}$, then $\sum_{x \in S} x = n$.

Find a Eulerian walk in H, $u_{k_1} \to u_{k_2} \to \cdots \to u_{k_n}$ (and finally back to u_{k_1}), that satisfies the following condition: for any $1 \leq i < j \leq N$, the walk passes all the $x_{i,j}$ edges between u_i and u_j *consecutively*.

For $i = 1, 2, \cdots, n$, assign the integer 'k_i' to the vertex v_i in G, then exit the algorithm.

4. If $N < n \leq (N-1)[(m-1)N-1]$ and $n - \{(N-1)[(m-1)N-1]\}$ is odd, then let $H = (V_H, E_H)$ be such a multi-graph: its vertex set $V_H = \{u_1, u_2, \cdots, u_N\}$;

for any $1 \leq i < j \leq N$, there are $x_{i,j}$ undirected edges between u_i and u_j; there is no loop in H. The integers $x_{i,j}$ $(1 \leq i < j \leq N)$ satisfy the following three requirements:

(1) for $1 \leq i \leq N-1$ and $j = i+1$, $x_{i,j}$ is odd and $1 \leq x_{i,j} \leq 2m-3$; also, $x_{1,N}$ is odd and $1 \leq x_{1,N} \leq 2m-3$;

(2) for $1 \leq i \leq N-3$ and $i+2 \leq j \leq N-1$, $x_{i,j}$ is even and $0 \leq x_{i,j} \leq 2m-2$; also, for $2 \leq i \leq N-2$ and $j = N$, $x_{i,j}$ is even and $0 \leq x_{i,j} \leq 2m-2$;

(3) if we define S as $S = \{x_{i,j} | i = 1, 2, \cdots, N; j = 1, 2, \cdots, N; i < j\}$, then $\sum_{x \in S} x = n$.

Find a Eulerian walk in H, $u_{k_1} \rightarrow u_{k_2} \rightarrow \cdots \rightarrow u_{k_n}$ (and finally back to u_{k_1}), that satisfies the following condition: for any $1 \leq i < j \leq N$, the walk passes all the $x_{i,j}$ edges between u_i and u_j *consecutively*.

For $i = 1, 2, \cdots, n$, assign the integer 'k_i' to the vertex v_i in G, then exit the algorithm.

□

The following is an example of Algorithm 3.

Example 4.5 *Assume $G = (V, E)$ is a cycle of $n = 12$ vertices, and the parameters are $N = 4$, $K = 3$, $m = 3$ and $L = 2$. Therefore $N < n \leq (N-1)[(m-1)N - 1]$ and $n - \{(N-1)[(m-1)N - 1]\} = -9$ is odd. So Algorithm 3's step 4 is used to compute the interleaving. We can (easily) choose the following values for $x_{i,j}$: $x_{1,2} = 3$, $x_{1,3} = 2$, $x_{1,4} = x_{2,3} = x_{3,4} = 1$, $x_{2,4} = 4$. Then the graph $H = (V_H, E_H)$ is as shown in Fig. 4.6(a). We can then (easily) find the following Eulerian walk in H: $u_1 \rightarrow u_3 \rightarrow u_1 \rightarrow u_2 \rightarrow u_1 \rightarrow u_2 \rightarrow u_4 \rightarrow u_2 \rightarrow u_4 \rightarrow u_2 \rightarrow u_3 \rightarrow u_4$ (then back to u_1). Corresponding to that walk, we get the MCI as shown in Fig. 4.6(b).*

□

Theorem 4.6 *Algorithm 3 correctly outputs an MCI on the cycle G.*

Theorem 4.7 *When $L = 2$ and $K = 3$, there exists an MCI on a cycle of n vertices if and only if $n \leq (N-1)[(m-1)N - 1]$.*

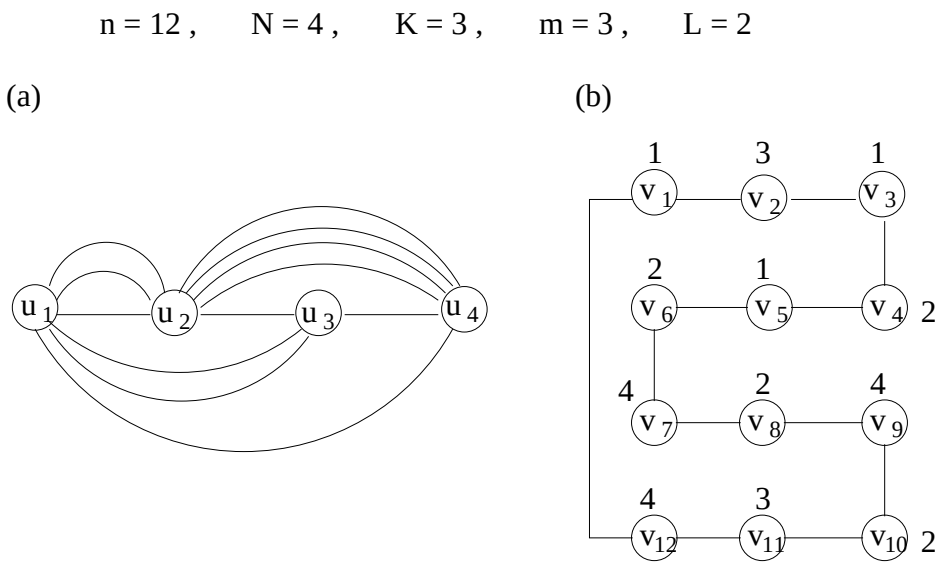


Figure 4.6: (a) The graph $H = (V_H, E_H)$ (b) MCI on the cycle $G = (V, E)$

For simplicity, we skip the proofs of the above two theorems. (The correctness of these two theorems should be very clear once the proofs for Theorem 4.2 and Theorem 4.3 are understood.)

4.6 Appendix I

In this appendix, we present Algorithm 4, which computes MCIs for linear paths when $L = 2$ and $K = 3$.

Algorithm 4: *MCI on linear path with constraints* $L = 2$ *and* $K = 3$

Input: A path $G = (V, E)$ of n vertices. Parameters N, K, m and L, where $N \geq 3$, $K = 3$, $m \geq 2$ and $L = 2$.

Output: An MCI on G.

Algorithm:

1. If $n > (N-1)[(m-1)N - 1] + 2$, there does not exist an MCI on G, so exit the algorithm.

2. If $n \leq N$, arbitrarily select n integers in the set $\{1, 2, \cdots, N\}$, and assign one distinct integer to each vertex, then exit the algorithm.

3. If $N < n \leq (N-1)[(m-1)N-1]+2$ and $n-\{(N-1)[(m-1)N-1]+2\}$ is even, then let $H = (V_H, E_H)$ be such a multi-graph: its vertex set $V_H = \{u_1, u_2, \cdots, u_N\}$; for any $1 \leq i < j \leq N$, there are $x_{i,j}$ undirected edges between u_i and u_j; there is no loop in H. The integers $x_{i,j}$ $(1 \leq i < j \leq N)$ satisfy the following four requirements:

(1) for $1 \leq i \leq N-1$, $x_{i,N}$ is even and $0 \leq x_{i,N} \leq 2m-2$;

(2) for $1 \leq i \leq N-2$ and $j = i+1$, $x_{i,j}$ is odd and $1 \leq x_{i,j} \leq 2m-3$;

(3) for $1 \leq i \leq N-3$ and $i+2 \leq j \leq N-1$, $x_{i,j}$ is even and $0 \leq x_{i,j} \leq 2m-2$;

(4) if we define S as $S = \{x_{i,j} | i = 1, 2, \cdots, N; j = 1, 2, \cdots, N; i < j\}$, then $\sum_{x \in S} x = n-1$.

Find a walk in H, $u_{k_1} \rightarrow u_{k_2} \rightarrow \cdots \rightarrow u_{k_n}$, that satisfies the following two requirements:

(1) the walk starts with u_1 and ends with u_{N-1} — namely, $u_{k_1} = u_1$ and $u_{k_n} = u_{N-1}$ — and passes every edge in H exactly once;

(2) for any $1 \leq i < j \leq N$, the walk passes all the $x_{i,j}$ edges between u_i and u_j *consecutively*.

For $i = 1, 2, \cdots, n$, assign the integer 'k_i' to the vertex v_i in G, then exit the algorithm.

4. If $N < n \leq (N-1)[(m-1)N-1]+2$ and $n-\{(N-1)[(m-1)N-1]+2\}$ is odd, then use step 3 to find an interleaving on a path of $n+1$ vertices. Remove one vertex from an end of that path, and we get an interleaving on G. Then exit the algorithm.

□

We comment that in step 3 of the above algorithm, the numbers of edges between the vertices of H, $x_{i,j}$ $(1 \leq i < j \leq N)$, can be easily determined using just elementary methods, since the constraints there are very simple.

4.7 Appendix II

In this appendix, we present the proofs of Lemma 4.4, 4.5 and 4.6.

Lemma 4.4 *Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a cycle of n vertices. Then in any MCI on a cycle $G = (V, E)$ of n_{max} vertices, no two adjacent vertices are assigned the same integer.*

Proof: The proof is by contradiction. Let $G = (V, E)$ be a cycle of n_{max} vertices. Assume there is an MCI on G; and assume two adjacent vertices of G are assigned the same integer. Then WLOG, one of the following two cases must be true (because we can always get one of the two cases by permuting the names of the integers and by shifting the indices of the vertices):

Case 1: There exist 4 consecutive vertices in G — v_i, v_{i+1}, v_{i+2}, v_{i+3} — such that $c(v_i) = 1$, $c(v_{i+1}) = c(v_{i+2}) = 2$, and $c(v_{i+3}) = 1$ or 3.

Case 2: There exist $x+2 \geq 5$ consecutive vertices in G — v_i, v_{i+1}, $\cdots$, v_{i+x}, v_{i+x+1} — such that $c(v_i) = 1$, $c(v_{i+1}) = c(v_{i+2}) = \cdots = c(v_{i+x}) = 2$, and $c(v_{i+x+1}) = 1$ or 3.

With the same argument as for the case 1 and case 2 in Lemma 4.1's proof, we can show that for both cases here, there is a cycle of more than n_{max} vertices on which an MCI exists. And that contradicts the definition of n_{max}. So this lemma is proved.

□

Lemma 4.5 *Let the values of N, K, m and L be fixed, where $N \geq 4$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a cycle of n vertices. Then $n_{max} \leq (N-1)[(m-1)N - 1]$.*

Proof: Let $G = (V, E)$ be a cycle of n_{max} vertices. Assume there is an MCI on G. By Lemma 4.4, no two adjacent vertices in G are assigned the same integer. And clearly, each one of the $N \geq 4$ integers is assigned to at least one vertex in G. We color the vertices in G with three colors — 'red', 'yellow' and 'green' — through the following three steps:

Step 1, for $1 \leq i \leq n_{max}$, if the two vertices adjacent to v_i are assigned the same integer, then we color v_i with the 'red' color;

Step 2, for $1 \leq i \leq n_{max}$, we color v_i with the 'yellow' color if v_i is not colored 'red' and there exists j such that these four conditions are satisfied: (1) $j \neq i$, (2)

v_j is not colored 'red', (3) $c(v_j) = c(v_i)$, (3) the following vertices between v_j and v_i — v_{j+1}, v_{j+2}, $\cdots$, v_{i-1} (note that if a lower index exceeds n_{max}, it is subtracted by n_{max}, so that the lower index is always between 1 and n_{max}) — are all colored 'red';

Step 3, for $1 \leq i \leq n_{max}$, if v_i is neither colored 'red' nor colored 'yellow', then we color v_i with the 'green' color.

Clearly, each vertex in G is assigned exactly one of the three colors.

If we arbitrarily pick two different integers — say 'i' and 'j' — from the set $\{1, 2, \cdots, N\}$, then we get a *pair* $[i, j]$ (or $[j, i]$, equivalently). There are totally $\binom{N}{2}$ such un-ordered *pairs*. We partition those $\binom{N}{2}$ *pairs* into four groups 'A', 'B', 'C' and 'D' in the following way:

(1) A *pair* $[i, j]$ is placed in group A if and only if the following two conditions are satisfied: (i) at least one 'green' vertex is assigned the integer 'i' and at least one 'green' vertex is assigned the integer 'j', (ii) there doesn't exist a connected subgraph of G such that the subgraph contains exactly two green vertices (and possibly also vertices of other colors), where one of the green vertices is assigned the integer 'i' and the other green vertex is assigned the integer 'j'.

(2) A *pair* $[i, j]$ is placed in group B if and only if the following two conditions are satisfied: (i) at least one 'green' vertex is assigned the integer 'i' and at least one 'green' vertex is assigned the integer 'j', (ii) there exists a connected subgraph of G such that the subgraph contains exactly two green vertices (and possible also vertices of other colors), where one of the green vertices is assigned the integer 'i' and the other green vertex is assigned the integer 'j'.

(3) A *pair* $[i, j]$ is placed in group C if and only if one of the following two conditions is satisfied: (i) at least one 'green' vertex is assigned the integer 'i' and no 'green' vertex is assigned the integer 'j', (ii) at least one 'green' vertex is assigned the integer 'j' and no 'green' vertex is assigned the integer 'i'.

(4) A *pair* $[i, j]$ is placed in group D if and only if no 'green' vertex is assigned the integer 'i' or 'j'.

E is the set of edges in $G = (V, E)$. For any $1 \leq i \neq j \leq N$, let $E(i, j) \subseteq E$ denote such a subset of edges of G: an edge of G is in $E(i, j)$ if and only if the two

endpoints of the edge are assigned the integer 'i' and the integer 'j' respectively. Let $z(i,j)$ denote the number of edges in $E(i,j)$. Upper bounds for $z(i,j)$ are derived below.

For any *pair* $[i,j]$ in group A or group C, $z(i,j) \leq 2m-2$. That is because otherwise there would exist m non-overlapping clusters in G — note that a cluster contains exactly one edge — each of which is assigned only integers 'i' and 'j', which would contradict the assumption that the interleaving on G is an MCI.

Now consider a *pair* $[i,j]$ in group B. $z(i,j) \leq 2m-2$ for the same reason as in the previous case. Assume $z(i,j) = 2m-2$. Then in order to avoid the existence of m non-overlapping clusters in G each of which is assigned only integers 'i' and 'j', the $z(i,j) = 2m-2$ edges in $E(i,j)$ must be consecutive in the cycle G, which means, WLOG, that there are $2m-1$ consecutive vertices v_{y+1}, v_{y+2}, $\cdots$, v_{y+2m-1} $(y \geq 0)$ whose assigned integers are in the form of $[c(v_{y+1})-c(v_{y+2})-\cdots-c(v_{y+2m-1})] = [i-j-i-j-\cdots-i-j-i]$. According to the definition of 'group B', there exist a 'green' vertex v_{k_1} and a 'green' vertex v_{k_2}, such that v_{k_1} is assigned the integer 'i', v_{k_2} is assigned the integer 'j', and either all the vertices in the subgraph $(v_{k_1+1}, v_{k_1+2}, \cdots, v_{k_2-1})$ (note that if an index is greater than n_{max}, then it is subtracted by n_{max} so that the index for a vertex is always between 1 and n_{max}; the same applies to the following contexts) are 'yellow' or 'red', or all the vertices in the subgraph $(v_{k_2+1}, v_{k_2+2}, \cdots, v_{k_1-1})$ are 'yellow' or 'red'. By the way the vertices are colored, it's not difficult to see that either 'v_{k_2-1} is assigned the integer $c(v_{k_1}) = i$' (let's call this 'case 1'), or 'v_{k_1-1} is assigned the integer $c(v_{k_2}) = j$' (let's call this 'case 2'). If 'case 1' is true, then since there is an edge between v_{k_2-1} (which is assigned the integer 'i') and v_{k_2} (which is assigned the integer 'j'), v_{k_2} is in the set $\{v_{y+1}, v_{y+2}, \cdots, v_{y+2m-1}\}$. However, it's simple to see that every vertex in the set $\{v_{y+1}, v_{y+2}, \cdots, v_{y+2m-1}\}$ that is assigned the integer 'j' must have the color 'red' — so v_{k_2} must be 'red' instead of 'green' — therefore a contradiction exists. Now if 'case 2' is true, since there is an edge between v_{k_1-1} (which is assigned the integer 'j') and v_{k_1} (which is assigned the integer 'i'), both v_{k_1-1} and v_{k_1} are in the set $\{v_{y+2}, v_{y+3}, \cdots, v_{y+2m-1}\}$. Then again since every vertex in the set $\{v_{y+1}, v_{y+2}, \cdots, v_{y+2m-1}\}$ that is assigned the integer 'j' must have the

color 'red', and since the color of v_{k_1} is 'green', it is simple to see that all the vertices in the set $\{v_{y+1}, v_{y+2}, \cdots, v_{k_1-1}\}$ that is assigned the integer 'i' must have the color 'red'. Then since the color of v_{y+1} is 'red', the vertex v_y must have been assigned the integer 'j' — and that contradicts the statement that all the edges in $E(i,j)$ are in the subgraph $(v_{y+1}, v_{y+2}, \cdots, v_{y+2m-1})$. Therefore a contradiction always exists when $z(i,j) = 2m-2$. So $z(i,j) \neq 2m-2$. So for any *pair* $[i,j]$ in group B, $z(i,j) \leq 2m-3$.

Now consider a *pair* $[i,j]$ in group D. By the definition of 'group D', no 'green' vertex is assigned the integer 'i' or 'j'. Let $\{v_{k_1}, v_{k_2}, \cdots, v_{k_t}\}$ denote the set of 'yellow' vertices that are assigned the integer 'i', where $k_1 < k_2 < \cdots < k_t$. If $\{v_{k_1}, v_{k_2}, \cdots, v_{k_t}\} \neq \emptyset$, by the way vertices are colored, it's simple to see that every vertex of G that is not in the set $\{v_{k_1}, v_{k_2}, \cdots, v_{k_t}\}$ is 'red'. And if $\{v_{k_1}, v_{k_2}, \cdots, v_{k_t}\} = \emptyset$, then all the vertices that are assigned the integer 'i' are 'red'. Similarly, we can show that either all the vertices that are assigned the integer 'j' are 'red', or there is no 'green' vertex in G and every 'yellow' vertex is assigned the integer 'j'. Together, there are three possible cases in total — case 1, every vertex that is assigned the integer 'i' or 'j' is 'red'; case 2, there is no 'green' vertex in G and every 'yellow' vertex is assigned the integer 'i'; case 3, there is no 'green' vertex in G and every 'yellow' vertex is assigned the integer 'j'. If case 1 is true, assume there is an edge whose two endpoints are assigned the integer 'i' and the integer 'j', respectively. Then since the two vertices adjacent to a 'red' vertex must be assigned the same integer, every vertex in G is assigned either the integer 'i' or the integer 'j', which contradicts the fact that all the $N \geq 4$ integers must have been assigned to the vertices in G. So for any *pair* $[i,j]$ in group D, if case 1 is true, then $z(i,j) = 0$. If case 2 is true, then for any $k \neq i$, there are at most $2m-2$ edges in $E(i,k)$; and for any $k_1 \neq i$ and $k_2 \neq i$, since every vertex that is assigned the integer 'k_1' or 'k_2' is 'red', there is no edge in $E(k_1, k_2)$ — therefore, there are at most $(N-1) \cdot (2m-2) < (N-1)[(m-1)N-1]$ edges in G. So when case 2 is true, there are less than $(N-1)[(m-1)N-1]$ vertices in G, and this lemma is proved. Similarly, when case 3 is true, there are also less than $(N-1)[(m-1)N-1]$ vertices in G, and this lemma is also proved. For this reason, in the following paragraph, we'll simply assume that case 1 is always true —

and therefore $z(i,j) = 0$ is true for any *pair* $[i,j]$ in group D.

Let the number of *distinct* integers assigned to 'green' vertices be denoted by 'x', and let 'X' denote the set of those x distinct integers. It's simple to see that exactly $\binom{x}{2}$ *pairs* $[i,j]$ are in group A and group B, where $i \in X$ and $j \in X$ — and among them at least x *pairs* are in group B. It's also simple to see that exactly $x(N-x)$ pairs are in group C and exactly $\binom{N-x}{2}$ *pairs* are in group D. By using the upper bounds we've derived on $z(i,j)$, we see that the number of edges in G is at most $[\binom{x}{2} - x] \cdot (2m-2) + x \cdot (2m-3) + x(N-x) \cdot (2m-2) + \binom{N-x}{2} \cdot 0 = (1-m)x^2 + (2mN - 2N - m)x$, whose maximum value (at integer solutions) is achieved when $x = N-1$ — and that maximum value is $(N-1)[(m-1)N-1]$. So n_{max}, the number of vertices in G, is at most $(N-1)[(m-1)N-1]$.

□

Lemma 4.6 *Let the values of N, K, m and L be fixed, where $N = 3$, $K = 3$, $m \geq 2$ and $L = 2$. Let n_{max} denote the maximum value of n such that an MCI exists on a cycle of n vertices. Then $n_{max} \leq (N-1)[(m-1)N-1]$.*

Proof: Let $G = (V, E)$ be a cycle of n_{max} vertices that has an MCI on it. We need to show that $n_{max} \leq (N-1)[(m-1)N-1]$. It's simple to see that G is assigned $N = 3$ distinct integers. If in the MCI on G, no two adjacent vertices are assigned the same integer, then with the same argument as in the proof of Lemma 4.5, it can be shown that $n_{max} \leq (N-1)[(m-1)N-1]$. Now assume there are two adjacent vertices in G that are assigned the same integer. Then there are three possible cases.

Case 1: n_{max} is even.

Case 2: n_{max} is odd, and there are at least 2 non-overlapping clusters in G each of which is assigned only 1 distinct integer.

Case 3: n_{max} is odd, and there don't exist 2 non-overlapping clusters in G each of which is assigned only 1 distinct integer.

We consider the three cases one by one.

Case 1: n_{max} is even. In this case, clearly we can find $\frac{n_{max}}{2}$ non-overlapping clusters (of size $L = 2$) such that at least one of them contains two vertices that are assigned

the same integer. Among those $\frac{n_{max}}{2}$ non-overlapping clusters, let x, y, z, a, b and c respectively denote the number of clusters that are assigned only integer '1', only integer '2', only integer '3', both integers '1' and '2', both integers '2' and '3', and both integers '1' and '3'. Since the interleaving is an MCI, clearly $x+y+a \leq m-1$, $y+z+b \leq m-1$, $z+x+c \leq m-1$. So $2x+2y+2z+a+b+c \leq 3m-3$. So $x+y+z+a+b+c \leq 3m-3-(x+y+z)$. Since $x+y+z \geq 1$ and $n_{max} = 2(x+y+z+a+b+c)$, we get $n_{max} \leq 2[3m-3-(x+y+z)] \leq 6m-8 = (N-1)[(m-1)N-1]$.

Case 2: n_{max} is odd, and there are at least 2 non-overlapping clusters in G each of which is assigned only 1 integer. In this case, clearly we can find $\frac{n_{max}-1}{2}$ non-overlapping clusters (of size $L=2$) among which there are at least two clusters each of which is assigned only one integer. Among those $\frac{n_{max}-1}{2}$ non-overlapping clusters, let x, y, z, a, b and c respectively denote the number of clusters that are assigned only integer '1', only integer '2', only integer '3', both integers '1' and '2', both integers '2' and '3', and both integers '1' and '3'. Since the interleaving is an MCI, clearly $x+y+a \leq m-1$, $y+z+b \leq m-1$, $z+x+c \leq m-1$. So $2x+2y+2z+a+b+c \leq 3m-3$. So $x+y+z+a+b+c \leq 3m-3-(x+y+z)$. Since $x+y+z \geq 2$ and $n_{max} = 2(x+y+z+a+b+c)+1$, we get $n_{max} \leq 2[3m-3-(x+y+z)]+1 \leq 6m-9 < (N-1)[(m-1)N-1]$.

Case 3: n_{max} is odd, and there don't exist 2 non-overlapping clusters in G each of which is assigned only 1 integer. Let x', y', z', a', b' and c' respectively denote the number of edges in G whose two endpoints are both assigned integer '1', are both assigned integer '2', are both assigned integer '3', are assigned integers '1' and '2', are assigned integers '2' and '3', are assigned integers '1' and '3'. (Then $x'+y'+z'+a'+b'+c' = n_{max}$.) It's simple to see that among x', y' and z', two of them equal 0, and the other one is either 1 or 2. So without loss of generality, we consider the following two sub-cases.

Sub-case 1: $x'=1$, and $y'=z'=0$. In this case, $a' \leq 2m-3$, because otherwise there will be m non-overlapping clusters in G that are assigned only integers '1' and '2'. Similarly, $c' \leq 2m-3$. Also clearly, $b' \leq 2m-2$. If $a'=2m-3$ and $c'=2m-3$,

then since there don't exist m non-overlapping clusters in G that are assigned only 1 or 2 distinct integers, the MCI on G can only take the following form: in G, there are $a' = 2m - 3$ *consecutive* edges each of which has integers '1' and '2' assigned to its endpoints (the segment of the cycle G consisting of these edges begins with a vertex that is assigned the integer '2' and ends with a vertex that is assigned the integer '1'), followed by an edge whose two endpoints both are assigned the integer '1', then followed by $c' = 2m - 3$ *consecutive* edges each of which has the integers '1' and '3' assigned to its endpoints (the segment of the cycle G consisting of these edges begins with a vertex that is assigned the integer '1' and ends with a vertex that is assigned the integer '3'), then followed by b' *consecutive* edges each of which has the integers '2' and '3' assigned to its endpoints (the segment of the cycle G consisting of these edges begins with a vertex that is assigned the integer '3' and ends with a vertex that is assigned the integer '2') — then it's simple to see that b' can't be even, which implies that $b' < 2m - 2$ here. So in any case, we have $a' + b' + c' < (2m-3)+(2m-2)+(2m-3) = 6m-8$. So $n_{max} = x'+y'+z'+a'+b'+c' < 6m-7$. So $n_{max} \leq 6m - 8 = (N-1)[(m-1)N-1]$.

Sub-case 2: $x' = 2$, and $y' = z' = 0$. In this case, with arguments similar to those in sub-case 1, we get $a' \leq 2m - 4$, $c' \leq 2m - 4$, and $b' \leq 2m - 2$. So $n_{max} = x' + y' + z' + a' + b' + c' \leq 2 + (2m-4) + (2m-2) + (2m-4) = 6m - 8 = (N-1)[(m-1)N-1]$.

So it's proved that in any case, $n_{max} \leq (N-1)[(m-1)N-1]$. And this lemma is proved.

□

Chapter 5

Monotone Percolation and Topology Control

5.1 Preface

As discussed at the beginning of the thesis, the topological construction is a most fundamental challenge for peer-to-peer type of networks, including overlay networks and wireless networks. The topology-control method should be localized, in order to be scalable to the size and dynamics of the network; however, at the same time, the derived topology needs to achieve good global performance, including enabling geographic routing and many other performance requirements.

In this chapter, we present a novel topology-control algorithm for wireless networks, which accomplish the above objectives. The nodes in the wireless networks are modelled by a Poisson point process on a 2-dimensional Euclidean plane. For nodes in higher-dimensional normed spaces, our algorithm can be naturally extended to achieve similar performance — unless the nodes follow some very ill point process (such as having many big 'holes' in the normed space with no nodes inside). Therefore, the algorithm provides a very promising solution for the topology control of overlay networks as well.

5.2 Introduction

The topology of a wireless network is the basis for its performance. Nearly all the important properties — e.g., routing efficiency, capacity — are relying on it. The topology control of a wireless network is for every node to determine its connectivity to other nodes by selecting its coverage radius (transmission range). In this chapter, we always consider the wireless nodes to be a Poisson point process on an infinite 2-dimensional Euclidean plane; and we consider all the nodes to have omnidirectional coverage — namely, if a node v selects its coverage radius to be r, then there is a directed link (edge) from v to every other node within distance r.

Many topology control algorithms have been proposed, and the study on their performance has been extensive [20], [24], [31], [33], [56], [79], [82], [87]. They are generally using different tradeoffs among the design parameters, including coverage radius, node degree, connectivity, etc. Here we briefly introduce some most representative algorithms — the unit-disk graph scheme, the nearest-neighbors scheme, the Bluetooth scheme, and the nearest-with-forward-progress scheme:

- Unit-disk graph (UDG) scheme [24]: in this scheme, all the nodes have the same coverage radius. The network has no strong connectivity with probability 1. However, if the coverage radius is big enough, the network percolates — namely, there exists an infinite strongly-connected component in the network whose size is a positive (instead of 0) percentage of the network.

- Nearest-neighbors (NN) scheme [87]: in this scheme, all the nodes connect themselves to the same number of nearest neighbors. (Then the edges are made bidirectional.) It is shown that if we use n to denote the total number of nodes in the network (where $n \to \infty$), in order to make the network asymptotically connected, it is sufficient and necessary for every node to have $\Theta(\log n)$ neighbors.

- Bluetooth (BT) scheme [20]: in this scheme, the network has n nodes (where $n \to \infty$) and is confined in a $[0, 1]^2$ square. Every node connects itself to c other nodes chosen randomly within distance r, where c and r are constants. (Then

the edges are made bidirectional.) It is proven that when $r > 0$ and $c \geq 2$, the network is asymptotically connected.

- Nearest-with-forward-progress (NFP) scheme [33]: every node v connects itself to enough neighbors such that for every infinitely faraway point, the nearest node that is closer to the point than v is is included as a neighbor of v. (In the original definition of this scheme, the coverage radius of a node was assume to be dynamically changing. Here we'll regard a node's coverage radius as the maximum value it can be.)

In wireless networking, routing is probably the most extensively studied topic. For a large-scale wireless (and possibly mobile) network, the control messages for finding and maintaining routing paths can easily consume most of the network capacity. Currently geographic routing [14], [43], [45], [44], [59], or at least compact routing schemes similar to geographic routing, seem to be *the* way to make routing scalable. In geographic routing, when a node needs to forward a message to a destination, it tries to select a neighbor that is geographically closer to the destination as the next hop. The potential of geographic routing depends on the network's topology. In a general topology, geographic routing will meet many dead-ends (a dead-end is a node whose Euclidean distance to the destination is smaller than all its neighbors'); and to counter that problem, path-recovery mechanisms need to be used, where a well known method is to route on a planar sub-graph [6], [14], [43] . Such methods ensure correct delivery at the cost of extra routing length, under-utilization of communication links and possibly load un-balancing. Topologies more appropriate for geographic routing can be got by using schemes such as NFP [33], where dead-ends are eliminated with high probability for faraway destinations. The ideal topology will be one that eliminates dead-ends for all destinations.

The great value of the topology control algorithms proposed so far is evident, and some of their results are very deep. Their performance, however, is not yet satisfactory. The objectives that a good topology control algorithm should try to achieve include:

- **Localized construction**: every node determines its own coverage radius and connectivity to other nodes using only the information of its nearby nodes.
- **Strong connectivity**: a network is strongly connected if there is a directed path from any node to any other node.
- **Good expansion**: a wireless network has good expansion if for any connected area in the 2-dimensional plane, when the size of the area increases, the probability that *all* the nodes can reach the area (through directed paths) approaches 1.
- **Small node degree**: the degree of a node v is the number of other nodes within the coverage radius (transmission range) of v. (Namely, here 'degree' means out-degree.) A small degree of v means that when v transmits a message, its signal interferes with only a small number of other nodes.
- **Small node degree for utilization**: in some schemes, a node does not use all its links to forward messages. Instead, it uses only a subset of its links, whose cardinality we call the 'degree for utilization'. Such an action is sometimes useful, e.g., for simplifying link-usage control mechanisms.
- **Small coverage radius**: it leads to small power consumption.
- **Small hop distortion**: for any two nodes u and v, we define the hop distortion for them to be the ratio between the minimum number of hops (edges) it takes to travel from u to v and the Euclidean distance between them. (But we only consider the cases where the number of hops is at least 2, because otherwise it's neither necessary nor possible to bound this ratio.) The hop distortion of a network is the supreme value of the hop distortions for *all* the node pairs.
- **Small length distortion**: for any two nodes u and v, we define the length distortion for them to be the ratio between the length of the shortest path (in length) from u to v and the Euclidean distance between them. The length distortion of a network is the supreme value of the length distortions for *all* the node pairs.

The hop distortion and the length distortion together reflect the optimality of transmission delay and power usage.

- **Enabling geographic routing with no dead-ends**: such a property will greatly facilitate scalable, compact and efficient routing.

In this chapter, we present an adaptive topology control algorithm. More specifically, it includes a family of algorithms that incrementally achieve many or *all* of the above objectives. These are the first algorithms that achieve such comprehensive performance guarantees, to the best of our knowledge. We call the algorithms we present *Monotone Percolation algorithms*, because they all share these common features: for every node, it can reach infinitely many other nodes, which span every (big enough) area of the 2-dimensional plane; and for any pair of source and destination nodes, there exists a path that makes monotonic progress in the direction of the destination and maybe also in reducing the distance to the destination. (When we say 'one node can reach another node', it means there exists a directed path from the first node to the second one.) Such a percolation model bears both clear similarity and distinction when compared to classic percolation processes; in particular, it should be contrasted with the *oriented percolation* studied previously [30], [61].

We present 5 related algorithms, which we shall call *Algorithms I*, *II*, *III*, *IV* and *V*. We summarize and compare their performance with that of the four representative algorithms that we introduced — UDG, NN, BT and NFP — in the Table 1 below. (In Table 1, '*A-I*', '*A-II*', $\cdots$, '*A-V*' mean the 5 algorithms we present. Both the 'Hop distortion' and the 'Length distortion' are for the whole network. We assume the nodes in the network follow a Poisson point process on an infinite 2-dimensional plane, whose density is 1 node per unit area. To better explain the asymptotic performance of the previous algorithms, we use n to denote the total number of nodes in the network — and of course, $n \to \infty$. For the BT algorithm, in its original definition all the nodes are confined in a $[0, 1]^2$ square; since now we are enforcing the density to be 1, we scale up the lengths in the square at a rate of $\sqrt{n}$.)

From the table, we can clearly see that the algorithms we present make substantial improvements.

	UDG	NN	BT	NFP	*A-I*	*A-II*	*A-III*	*A-IV*	*A-V*
Localized construction	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
Strong connectivity	No	Yes	Yes	—	—	Yes	Yes	Yes	Yes
Good expansion	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Average degree	*Note**	$\Omega(\log(n)) \to \infty$	$\Theta(n) \to \infty$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Average degree for utilization	*Note**	$\Omega(\log(n)) \to \infty$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Average coverage radius	*Note**	$\Omega(\sqrt{\log(n)}) \to \infty$	$\Omega(\sqrt{n}) \to \infty$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Hop distortion	∞	—	∞	∞	∞	∞	∞	$\Theta(1)$	$\Theta(1)$
Length distortion	∞	—	∞	∞	∞	—	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Enabling geographic routing with no dead-ends	No	—	No	*Far only**	*Far only**	*Far only**	Yes	Yes	Yes

*Note**: for UDG, its average node degree (which is the same as average node degree for utilization) depends on the uniform coverage radius. For percolation to happen, the average degree needs to be at least 4.51258 — or equivalently, the coverage radius needs to be at least 1.1985.

*Far only**: geographic routing is enabled with no dead-ends with high probability only for faraway destinations.

Table 1: Summary and comparison of topology control algorithms.

Our contribution is in presenting these novel algorithms and their performance analysis, which uses some new techniques. And these algorithms demonstrate how good global connectivity-, expansion- and distortion-objectives can be achieved using purely local constructions.

The rest of the chapter is organized as follows. In Section 5.3, we define some basic terms. Section 5.4 through Section 5.8 respectively introduce the 5 related topology control algorithms and analyze their performance.

5.3 Basic Terms

The wireless network we consider consists of nodes on an infinite 2-dimensional Euclidean plane that follow a Poisson point process, whose density is 1 node per unit area. (All the results here can be modified very easily for a more general density λ.) We assume the nodes know their coordinates on the plane. Every node determines its coverage radius using the given topology control algorithm. If a node v chooses its coverage radius to be r, then there is a directed edge from v to every other node whose distance to v is less than or equal to r (and such a node is called a *neighbor* of v). (See Fig. 5.1.) As a result, we get a network that is a directed graph. The number of outgoing-edges of v is called the *degree* of v.

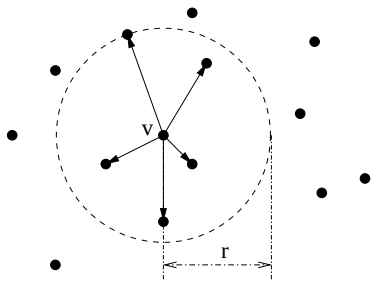


Figure 5.1: Coverage radius and outgoing edges.

Given a network, for any two nodes u and v, $d(u, v)$ denotes the Euclidean distance between them. Given a directed path from u to v, we use $H(u, v)$ to denote the number of edges (namely, *hops*) *in that path.* $h(u, v)$ denotes the *minimum* number of edges in

a directed path from u to v — that is, $h(u,v) = \min_{\text{paths from } u \text{ to } v} H(u,v)$. Given a directed path from u to v that consists of nodes 'u, w_1, w_2, $\cdots$, w_k, v', we use $L(u,v)$ to denote the *length of that path*, where $L(u,v) = d(u,w_1)+d(w_1,w_2)+\cdots+d(w_k,v)$. $l(u,v)$ denotes the *minimum* length of a directed path from u to v.

Given a directed path from node u to node v, we call $\frac{H(u,v)}{d(u,v)}$ the *hop distortion of that path*. In this chapter, we only consider those cases where the number of edges in the path is at least 2, because if it is 1, then the hop distortion of the path is simply $\frac{1}{d(u,v)}$ — then if $d(u,v) \to 0$, it becomes neither necessary nor possible to bound the hop distortion for any topology control method. We define *the hop distortion of the network* to be $\max_{u,v} \frac{h(u,v)}{d(u,v)}$ — that is, the *worst* hop distortion we may suffer even if the shortest-path routing (measured in hops) is always used.

Similarly, given a directed path from u to v, we define $\frac{L(u,v)}{d(u,v)}$ to be the *length distortion of that path*. We define *the length distortion of the network* to be $\max_{u,v} \frac{l(u,v)}{d(u,v)}$.

Now we define *cones*. Assume a node v has m neighbors. Then the rays starting at v and respectively crossing those neighbors cut the plane into m disjoint areas, which we call the *cones of* v. (For example, in Fig. 5.2, node v has 5 cones, whose cone angles are A_1, A_2, $\cdots$, A_5. Certainly $A_1 + A_2 + \cdots + A_5 = 2\pi$.) We say that those cones (or cone angles) of v are *caused* by those neighbors.

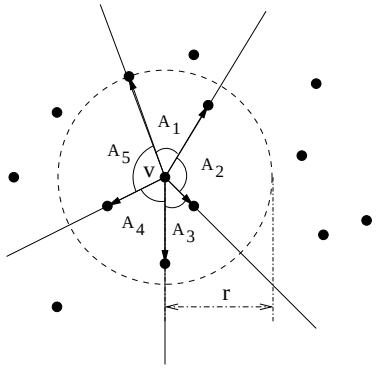


Figure 5.2: The cones and cone angles of v.

We would like to mention the difference between our work and two previously published algorithms, the cone-based algorithm [85], [49] and NFP [33]. The cone-

based algorithm and our algorithms both use the concept of 'cones', but other than that they have very different components and constructions; NFP bears some limited similarity with our Algorithm I, but the similarity quickly diminishes as we move on to Algorithms II, III, IV and IV — and even for NFP and Algorithm I, they have totally different node-connection and routing methods. In both the cone-based algorithm and NFP, the nodes' coverage radii change over time, while our algorithms use a static-coverage model (where the coverage radii are fixed once they are determined). The studied performance aspects are also distinct — for the cone-based algorithm, connectivity preservation, power-usage competitive analysis and power minimization were studied; for NFP, signal interference, throughput and forward progress were studied; while for our algorithms, the performance aspects we study are as listed in Section 5.2. So as a result, the algorithms, analysis and results are all very different.

5.4 Algorithm I

5.4.1 Definition

Algorithm I is defined as follows: "every node chooses its coverage radius to be the minimum value such that its cone angles are all smaller than π."

5.4.2 Routing Property

Say a node u needs to forward a message to a destination node v. With the network topology constructed using Algorithm I, u must have a neighbor w such that the angle $\angle wuv < \frac{\pi}{2}$. If $d(u, v) = \infty$, then it is not difficult to see that $d(w, v) < d(u, v)$ — so by relaying the message to w, u makes the message geographically closer to the destination. Now for destinations not infinitely far away, the farther away the destination is, the more likely u will have a neighbor that gets closer to the destination. So the network enables geographic routing without dead-ends with high probability for faraway destinations.

5.4.3 Node Degree

It is natural to think that when a node u uses Algorithm I to determine its coverage radius r, u does it in the following way: u firstly lets $r = 0$; then u gradually increases r, and as a result, u covers (connects itself to) more and more neighbors; when r reaches a particular value — let's call it r_0, — one more new neighbor is covered and suddenly all u's cone angles become smaller than π, and u knows that its coverage radius should be equal to r_0.

For the ease of analysis, let's do a little imagination: let's imagine that node u keeps increasing r for ever — even after r has exceeded r_0. (u will always remember that r_0 is what its coverage radius should be, though. So such an imagination won't hurt our analysis.)

Let's now understand the above process in another way. Let's imagine there is a circle centered at u whose circumference has length 1, which we shall call the 'Unit Circle'. We denote the neighbors of u by v_0, v_1, v_2 $\cdots$, where $d(u, v_0) < d(u, v_1) < d(u, v_2) < \cdots$. (In this chapter we always only consider the nodes to be in *general positions*, because in the probabilistic analysis, the probability that several nodes are not in general positions equals 0. *Please keep this in mind also for analysis elsewhere.*) For each v_i, the ray starting at u and going through v_i intersects the Unit Circle, and we denote the intersection point by w_i. (See Fig. 5.3 for an example.) We assign a *coordinate* to each point on the Unit Circle in the following way: for point p, its coordinate equals the distance one needs to cover when one moves from w_0 to p clockwise on the Unit Circle. (So the coordinate is in the range $[0, 1)$.)

When u is increasing its coverage radius r to cover more neighbors, every time a new neighbor — say it's v_i — is covered, v_i can be anywhere on the circle of radius r with the same probability; so w_i's coordinate has a uniform distribution in the range $[0, 1)$. We would like to think that the points w_0, w_1, w_2 $\cdots$ *cut* the Unit Circle into pieces. (For example, in Fig. 5.3, when u has 4 neighbors, the Unit Circle is cut into 4 pieces, whose endpoints are respectively w_0 and w_1, w_1 and w_3, w_3 and w_2, w_2 and w_0.) **Clearly, all u's cone angles are smaller than π if and only if all the pieces**

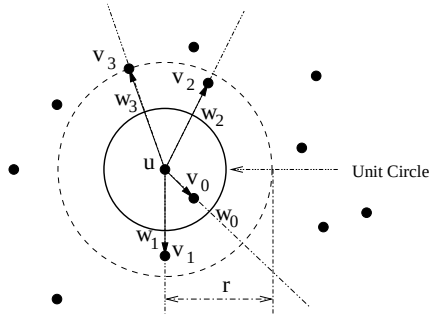


Figure 5.3: Projecting u's neighbors onto the Unit Circle.

of the Unit Circle have lengths smaller than 0.5.

Theorem 5.1 *Let Z denote the degree of a node in a network constructed using Algorithm I. Then $Pr\{Z \leq n\} = 1 - \frac{n}{2^{n-1}}$ for $n = 1, 2, 3, \cdots$; and $Pr\{Z = n\} = \frac{n-2}{2^{n-1}}$ for $n = 2, 3, 4, \cdots$.*

Proof: We consider the node u in the previous discussion, and let Z be its degree. The probability that '$Z \leq n$' equals the probability of the following event — "n uniformly distributed points w_0, w_1, $\cdots$, w_{n-1} on the Unit Circle cut the Unit Circle into n pieces whose lengths are all smaller than 0.5", which then equals the probability of this event — "among those n uniformly distributed points w_0, w_1, $\cdots$, w_{n-1}, there exist two points w_i and w_j $(1 \leq i \neq j \leq n-1)$ such that the following three conditions are satisfied: (i) w_i's coordinate x is in the range $(0, 0.5)$, (ii) w_j's coordinate y is in the range $(0.5, x + 0.5)$, (iii) for each of the $n - 3$ points excluding w_0, w_i and w_j, its coordinate is either in the range $(0, x)$ or in the range $(y, 1)$."

There are $(n-1)(n-2)$ ways to select the two points x_i and x_j. The probability that 'a point w_k $(k \neq 0, i, j)$ is in the range $(0, x)$ or $(y, 1)$' equals $(x - 0) + (1 - y) = 1 + x - y$. So $Pr\{Z \leq n\} = \int_0^{\frac{1}{2}} \int_{\frac{1}{2}}^{x+\frac{1}{2}} (n-1)(n-2)(1+x-y)^{n-3} dy dx = 1 - \frac{n}{2^{n-1}}$. (Here $n \geq 3$.) The rest of the proof is straightforward. □

Next we compute the expectation and variance of the node degree and show that they are small. In principle they can, certainly, be computed by using the distribution of node degree derived in Theorem 5.1. However we present here an

alternative solution for the following reasons: the solution is intriguing and makes the degree's expectation and variance significantly easier to compute; it provides a more 'combinatorial' analysis that can be extended for many generalized forms of Algorithm I.

Theorem 5.2 *Let Z denote the degree of a node in a network constructed using Algorithm I. Then $E(Z) = 5$, $Var(Z) = 4$.*

Proof: Let's once more consider the generic node u in the previous analysis, and let Z be its degree. Assume at some moment, the Unit Circle is cut into n pieces by n points — w_0, w_1, $\cdots$, w_{n-1}. For $i = 1, \cdots, n-1$, let's use C_i to denote the coordinate of w_i; and we write C_i in the binary form: $C_i = (0.c_{i1}c_{i2}c_{i3}\cdots)_2$. (*For example, if $C_i = 0.75$, then its binary form is $C_i = (0.c_{i1}c_{i2}c_{i3}\cdots)_2 = (0.110\cdots)_2$.*) Let's re-order C_1, C_2, $\cdots$, C_{n-1} according to this rule: "C_i is placed in front of C_j if $(0.0c_{i2}c_{i3}\cdots)_2 < (0.0c_{j2}c_{j3}\cdots)_2$." Let's call those re-ordered numbers D_1, D_2, $\cdots$, D_{n-1}. (*For example, if $n = 4$ and $C_1 = (0.010\cdots)_2$, $C_2 = (0.111\cdots)_2$, $C_3 = (0.101\cdots)_2$, then we re-order them as: $D_1 = (0.101\cdots)_2$, $D_2 = (0.010\cdots)_2$, $D_3 = (0.111\cdots)_2$.*)

For $i = 1, 2, \cdots, n-1$, we denote the binary form of D_i by $(0.d_{i1}d_{i2}d_{i3}\cdots)_2$. Now let's observe the following sequence of bits: d_{11}, d_{21}, $\cdots$, $d_{(n-1)1}$. It's not difficult to realize that "the n points — w_0, w_1, $\cdots$, w_{n-1} — cut the Unit Circle into n pieces all shorter than 0.5" if and only if "in the sequence of bits — d_{11}, d_{21}, $\cdots$, $d_{(n-1)1}$ — there is a 0 that appears *behind* a 1". We can also see that even if the permutation (re-ordering) from 'C_1, C_2, $\cdots$, C_{n-1}' to 'D_1, D_2, $\cdots$, D_{n-1}' is fixed, 'd_{11}, d_{21}, $\cdots$, $d_{(n-1)1}$' are still $n-1$ i.i.d. random variables with the equal probability to be 0 or 1 (because the permutation has nothing to do with the sequence 'd_{11}, d_{21}, $\cdots$, $d_{(n-1)1}$').

We define the following game: "we throw a fair coin at discrete times 1, 2, 3 $\cdots$; if at time i, the coin shows side 1 (or, '*HEADS*') *for the first time*, then we let $x = i$; if at time j ($j > i$), the coin shows side 0 (or, '*TAILS*') *for the first time after time i*, then we let $y = j - i$." From the analysis, we can see that $Pr\{Z \le n\} = Pr\{x + y \le n - 1\}$. So for any m, $Pr\{Z = m\} = Pr\{x + y + 1 = m\}$. So $E(Z) = E(x + y + 1)$ and $Var(Z) = Var(x + y + 1)$. x and y both have the geometric distribution and they

are clearly independent. So $E(x) = E(y) = Var(x) = Var(y) = 1/(\frac{1}{2}) = 2$. So $E(Z) = E(x) + E(y) + 1 = 5$ and $Var(Z) = Var(x) + Var(y) = 4$. □

5.4.4 Coverage Radius

Theorem 5.3 *Let R denote the coverage radius of a node in a network constructed using Algorithm I. Then the cumulative distribution function of R is $F_R(x) = Pr\{R \leq x\} = 1 - e^{-\pi x^2} - \pi x^2 e^{-\frac{\pi x^2}{2}}$ for $x \geq 0$. And the probability density function of R is $f_R(x) = \frac{dF_R(x)}{dx} = 2\pi x e^{-\pi x^2} + \pi^2 x^3 e^{-\frac{\pi x^2}{2}} - 2\pi x e^{-\frac{\pi x^2}{2}}$ for $x \geq 0$; $E(R) = \frac{\sqrt{2}+1}{2} \approx 1.207$; and $Var(R) = \frac{5}{\pi} - \frac{3+2\sqrt{2}}{4} \approx 0.1344$.*

Proof: Let u be the node whose coverage radius we are studying here, and let Z denote its degree. Let n denote the number of nodes (except u itself) whose distance to u is less than or equal to x. Then, $F_R(x) = Pr\{R \leq x\} = Pr\{Z \leq n\} = Pr\{$ the n cone angles caused by the n nearest neighbors of u are all smaller than $\pi\} = \sum_{i=3}^{\infty} Pr\{n = i\} Pr\{$ the i cone angles caused by the i nearest neighbors of u are all smaller than $\pi\}$. n is a Poisson variable; and it's not hard to see that the event "the n cone angles caused by the n nearest neighbors of u are all smaller than π" is independent of x, even though n itself depends on x, because the nodes are distributed uniformly with respect to directions — so that event's probability simply equals $Pr\{Z \leq n\}$ (whose expression is shown in Theorem 5.1). Therefore $F_R(x) = \sum_{i=3}^{\infty} \frac{(\pi x^2)^i}{i!} e^{-\pi x^2} Pr\{Z \leq i\} = \sum_{i=3}^{\infty} \frac{(\pi x^2)^i}{i!} e^{-\pi x^2} (1 - \frac{i}{2^{i-1}}) = 1 - e^{-\pi x^2} - \pi x^2 e^{-\frac{\pi x^2}{2}}$. The rest of the proof is straightforward. □

5.4.5 Expansion and Connectivity

Theorem 5.4 *A network constructed using Algorithm I has* good expansion. *What's more, it contains a strongly-connected subgraph that* every *node can reach. Such a subgraph is infinitely large and unique.*

Proof: Let u be an arbitrary node in a network constructed using Algorithm I. Let $E(u)$ denote the set of edges of the network that u can reach. The edges in $E(u)$ cut

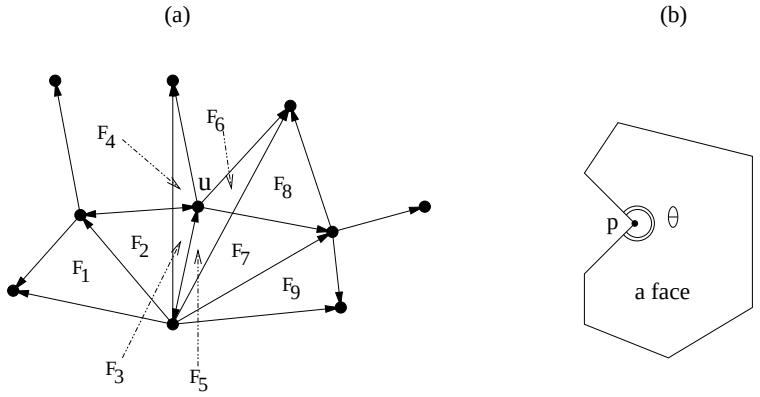


Figure 5.4: (a) The *faces* caused by the edges reachable from node u. (b) A *face* that cannot possibly exist.

the 2-dimensional plane into disjoint areas, which we shall call *faces*. (See Fig. 5.4 (a) for an example, where the faces are labelled as F_1, F_2, $\cdots$.) Each of the faces must have the shape of a convex polygon. To see that, let's assume there is a face which is a concave polygon, as shown in Fig. 5.4 (b). Then the face has an inner angle θ that is greater than π. It is easy to see that there must be a node located at the point p, and the cone-angles of that node are not all smaller than π — and that contradicts the fact that the network is constructed using Algorithm I. Each of the faces must also be a closed polygon — because otherwise, the boundary of the face would contain an infinitely long edge, which is impossible. So all the faces are closed convex polygons. With the fact that the nodes in the network follow a Poisson point process, it is not difficult to see that the network has *good expansion*.

Let's call a maximal strongly-connected subgraph a *component*, and call a component that has no directed edges going into other components a *sink component*. The network constructed using Algorithm I must have one or more sink components. Assume there exist two sink components A and B, which respectively contain a node u and a node v. It is simple to see that the set of nodes and edges in A (respectively, B) are exactly the set of nodes and edges in the network that u (respectively, v) can reach. Both the edges in A and the edges in B cut the plane into faces that are closed convex polygons of finite areas. So the edges in A and in B must intersect each other

somewhere. Without loss of generality (WLOG), we assume there is an edge in A, which starts at node x_1 and ends at node x_2, and an edge in B, which starts at node y_1 and ends at node y_2, which intersect each other at a point s. Also WLOG, we assume $d(s, x_2) \geq d(s, y_2)$. Then $d(x_1, x_2) = d(x_1, s)+d(s, x_2) \geq d(x_1, s)+d(s, y_2) \geq d(x_1, y_2)$. Since x_2 is within the coverage radius of x_1, so is y_2. So there is a directed edge starting at x_1 and ending at y_2, which contradicts the facts that component A is a sink component. So there is only one sink component. It is simple to see that all the nodes of the network can reach the unique sink component, and the sink component is infinitely large.

□

The strongly-connected subgraph that all nodes can reach spans the plane well. That makes long-range communication feasible.

5.5 Algorithm II

5.5.1 Definition

Algorithm II is an enhanced version of Algorithm I. Compared to Algorithm I, Algorithm II further increases nodes' coverage radii so that for any two nodes, if Algorithm I would create a directed edge between them, then Algorithm II will create a bi-directional edge between them. The reason for doing this is that in engineering, it's often desirable to use bi-directional links for the ease of medium-access control and 2-way communication (although in principal, having short loops instead may also work).

Algorithm II has two steps, and it is defined as follows: "step 1, use Algorithm I to determine the coverage radius of every node; step 2, for every node v, if its coverage radius was r and the maximum length of its incoming edges was t in step 1, then v makes its coverage radius to be $\max\{r, t\}$."

5.5.2 Connectivity and Routing Property

All the good routing and connectivity properties that Algorithm I can achieve is also achieved by Algorithm II. Moreover, Algorithm II guarantees the strong connectivity of the network:

Theorem 5.5 *A network constructed using Algorithm II is strongly connected.*

Proof: For a fixed set of nodes on the 2-dimensional plane, let's use Algorithm I (Algorithm II) to construct a network which we shall call Network I (Network II). Clearly for any two nodes p and q, if there is a directed path from p to q or from q to p in Network I, then in Network II, all the edges in that path become bi-directional — so p can q can reach *each other*. Now consider two arbitrary nodes u_1 and u_2. By Theorem 5.4, in Network I there is a strongly-connected subgraph that u_1 and u_2 can both reach. Then it's not hard to see that in Network II, u_1 and u_2 can reach each other. So Network II is strongly connected. □

5.5.3 Coverage Radius and Node Degree

The distribution of a node's coverage radius is hard to compute for Algorithm II. Nevertheless, we can show that its expectation and variance are upper bounded by small constants.

Lemma 5.1 *In a network constructed using Algorithm I, randomly and uniformly select an edge, and denote the edge's length by L. Then the probability density function of L is $f_L(x) = \frac{2\pi}{5}x(e^{-\pi x^2} + \pi x^2 e^{-\frac{\pi x^2}{2}})$, for $x \geq 0$.*

Proof: Without loss of generality, we assume there is a node u at the origin point of the 2-dimensional plane, and consider its outgoing edges. Let's use $\Delta(x)$ to denote the probability that "u has an outgoing edge whose length is between x and $x+dx$", where $dx \to 0$. Then $\Delta(x) = Pr\{$the cone-angles of u caused by the nodes within distance x are not all smaller than $\pi\}Pr\{$there exists a node whose distance to u is between x and $x+dx\} = Pr\{$the cone-angles of u caused by the nodes within distance x are not

all smaller than $\pi\}[\pi(x+dx)^2-\pi x^2][1+o(1)] = Pr\{$the cone-angles of u caused by the nodes within distance x are not all smaller than $\pi\}[2\pi x dx+o(dx)]$. From Theorem 5.3, we can see that $Pr\{$the cone-angles of u caused by the nodes within distance x are not all smaller than $\pi\} = e^{-\pi x^2}+\pi x^2 e^{-\frac{\pi x^2}{2}}$. So $\Delta(x) = (e^{-\pi x^2}+\pi x^2 e^{-\frac{\pi x^2}{2}})[2\pi x dx+o(dx)]$. It is not hard to see that $f_L(x) = [\lim_{dx\to 0}\frac{\Delta(x)}{dx}]\cdot C = (e^{-\pi x^2}+\pi x^2 e^{-\frac{\pi x^2}{2}})\cdot 2\pi x\cdot C$, where C is some appropriate scaling factor. From $\int_0^\infty f_L(x)dx = 1$, we find that $C = \frac{1}{5}$. So $f_L(x) = \frac{2\pi}{5}x(e^{-\pi x^2}+\pi x^2 e^{-\frac{\pi x^2}{2}})$, for $x \geq 0$.

□

Theorem 5.6 *Let R denote the coverage radius of a node in a network constructed using Algorithm II. Then $E(R) \leq 1.6585$, and $Var(R) \leq 1.338$.*

Proof: Imagine we have a very large bag, and let's play the following game: "generate the nodes following the Poisson point process and use Algorithm I to construct a network; corresponding to each node u of the network, we put a red stick whose length equals the coverage radius of u into the bag; corresponding to each directed edge of the network, we put a green stick whose length equals the length of the edge into the bag; repeat all the above steps."

Clearly there will be infinitely many sticks in the bag, because each network has infinite nodes and we generate the network infinitely many times. Nevertheless, if we use L_R to denote the length of a red stick and use L_G to denote the length of a green stick, we know the probability density functions of L_R and L_G: $f_{L_R}(x) = 2\pi x e^{-\pi x^2}+\pi^2 x^3 e^{-\frac{\pi x^2}{2}}-2\pi x e^{-\frac{\pi x^2}{2}}$ (by Theorem 5.3), and $f_{L_G}(x) = \frac{2\pi}{5}x(e^{-\pi x^2}+\pi x^2 e^{-\frac{\pi x^2}{2}})$ (by Lemma 5.1). Also, Theorem 5.2 tells us that the average degree of a node is 5, so the number of green sticks is 5 times the number of red sticks. Therefore, if we use L to denote the length of a stick in the bag (regardless of its color), then the probability density function of L is $f_L(x) = [f_{L_R}(x)+5f_{L_G}(x)]\cdot\frac{1}{6} = \frac{2\pi}{3}xe^{-\pi x^2}+\frac{\pi^2}{2}x^3 e^{-\frac{\pi x^2}{2}}-\frac{\pi}{3}xe^{-\frac{\pi x^2}{2}}$.

For Algorithm II, we see (from its definition) that a node determines its coverage radius through these two steps: "firstly, Algorithm I is used to construct a network; next, every node sets its coverage radius to be the larger value of its old coverage

radius and the length of its longest old incoming edge." Equivalently, we can also see the nodes as using the following method to determine their coverage radii: "every node picks a red stick and a green stick (not necessarily uniformly) from the bag, and sets its coverage radius to be the maximum length of these two sticks.". Clearly, no two nodes need to, or should, pick the same red stick or green stick.

We can see that the average coverage radius for Algorithm II cannot exceed the average coverage radius for the following scheme: "pick just one stick out of the bag for each node (one by one), and set the node's coverage radius to be equal to that stick's length; for every node, we always pick the longest stick that is still available (regardless of its color)." What is the average length of the sticks picked out in the above scheme? The number of sticks in the bag is 6 times the number of nodes. So if we are picking out exactly those sticks longer than a value z, then $\int_z^\infty f_L(x)dx = \frac{1}{6}$ — so we find $z \approx 1.4115$. Then the average length of the sticks picked out in the above scheme is $[\int_z^\infty x f_L(x)dx] \cdot 6 = 1.6585$. So $E(R) \leq 1.6585$.

With the same method we can prove that $E(R^2) \leq 2.7948$. The average coverage radius for Algorithm II is no less than that of algorithm I, so $E(R) \geq \frac{\sqrt{2}+1}{2} \approx 1.207$ (by Theorem 5.3). So $Var(R) = E(R^2) - E^2(R) \leq 2.7948 - 1.207^2 = 1.338$.

□

From the proof of Theorem 5.6, it is easy to realize that, in fact, all the moments of the coverage radius (not just its expectation and second moment) in a network constructed using Algorithm II are finite constants (instead of ∞). As to the node degree, we can see that in such a network, the neighbors within a node's coverage range do not follow a Poisson point process, because the node's coverage radius is dependent on the positions of the neighbors. Nevertheless, it is not hard to see that the expectation of the node degree is also a finite constant.

5.6 Algorithm III

5.6.1 Definition

Algorithm III is defined as follows: "every node chooses its coverage radius to be the minimum value such that its cone angles are all smaller than or equal to θ, where θ is a fixed number and $0 < \theta < \frac{2\pi}{3}$."

5.6.2 Connectivity and Routing Property

Lemma 5.2 *In a network constructed using Algorithm III, let u and v be any two nodes. Then u has a neighbor w such that $d(w, v) < d(u, v)$.*

Proof: Let r be the coverage radius of u. We just need to consider the case where $d(u, v) > r$. Since all u's cone angles are smaller than or equal to θ, u must have a neighbor w such that the angle $\angle wuv \leq \frac{\theta}{2} < \frac{\pi}{3}$ — then since $d(u, w) \leq r < d(u, v)$, we get $d(w, v) < d(u, v)$.

□

If in Algorithm III we make $\theta > \frac{2\pi}{3}$, then the statement in Lemma 5.2 no longer holds (however it holds if $\theta = \frac{2\pi}{3}$). So in this sense $\frac{2\pi}{3}$ is a threshold for θ. From the above lemma we can also see that for any source and destination, we can use geographic routing to find a path between them without meeting any dead-end (because every intermediate node can use a neighbor closer to the destination as the next node in the routing path). So we get:

Theorem 5.7 *A network constructed using Algorithm III is strongly connected. And it enables geographic routing with no dead-ends.*

5.6.3 Length Distortion and Routing Property

Lemma 5.3 *Let O and P be two nodes, and let OAE be a sector centered at O as shown in Fig. 5.5, where $\angle AOP \leq \frac{\pi}{3}$ and $d(O, P) > d(O, A)$. For every point t in the sector (including the boundary of the sector), we define $\eta(t)$ as $\eta(t) =$*

$\frac{d(O,t)}{d(O,P)-d(t,P)}$, *and call* $\eta(t)$ *the* 'competitive ratio of t'. *Then, in the sector,* A *is the unique point that maximizes the competitive ratio — and that value is* $\eta(A) = \frac{d(O,A)}{d(O,P)-\sqrt{d^2(O,P)+d^2(O,A)-2d(O,P)d(O,A)\cos\angle AOP}}$. *What's more,* $\eta(A)$ *is a strictly decreasing function in* $d(O,P)$.

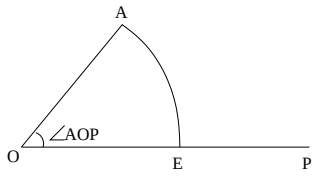


Figure 5.5: Maximum competitive ratio in a sector.

Proof: Draw a circle C centered at P that intersects the sector. If we move a point c on C, the value $d(O,P) - d(c,P)$ clearly remains constant. If C intersects the arc $\widehat{AE}$, then clearly for points on C that are also in the sector, the intersection of C and $\widehat{AE}$ is the only point that maximizes the competitive ratio. If C does not intersect the arc $\widehat{AE}$, then it intersects the line segment $\overline{OA}$; then it's also clear that for points on C that are also in the sector, the intersection of C and $\overline{OA}$ is the only point that maximizes the competitive ratio. So the point (or points) in the sector that maximizes the competitive ratio is either on the arc $\widehat{AE}$ or on the line segment $\overline{OA}$. But then since all the points on the arc $\widehat{AE}$ have the same distance to O, it is simple to see that A has a greater competitive ratio than any other point on the arc $\widehat{AE}$. So the point (or points) in the sector that maximizes the competitive ratio is on the line segment $\overline{OA}$.

Now let's find out which point on the line segment $\overline{OA}$ maximizes the competitive ratio. We draw a circle centered at point P whose radius is $d(P,A)$, which is shown as the solid circle in Fig. 5.6 — let's denote the intersection of the circle and the line segment $\overline{OA}$ by 'D'. Let B be an arbitrary point on the line segment $\overline{DA}$. Draw a circle centered at point P whose radius is $d(P,B)$, which intersects the arc $\widehat{AE}$ at point C, as shown in Fig. 5.6. It is simple to see that the competitive ratio of B is smaller than that of C, which is in turn smaller than that of A. So the competitive

ratio of A is greater than any other point on the line segment $\overline{DA}$.

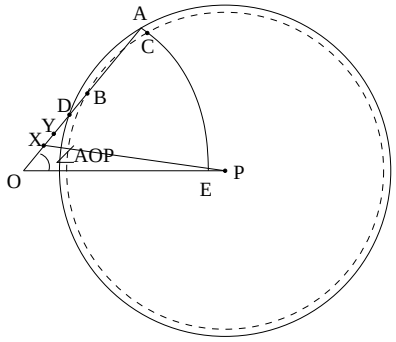


Figure 5.6: The maximum competitive ratio of points on the line segment $\overline{OA}$.

Now consider points on the line segment $\overline{OD}$. Let X and Y be two points on $\overline{OD}$, and let the distance between X and Y be extremely small — namely, $d(X,Y) \to 0$. Y is closer to P than X is by $d(X,Y)\cos\angle YXO$ — and that value is a strictly decreasing function of $\angle YXO$. So it is not difficult to see that for a point on the line segment $\overline{OD}$, the further away it is from O, the greater its competitive ratio is. So the competitive ratio of point D is greater than any other point on the line segment $\overline{OD}$.

By combining the above results, we can see that in the sector OAE, A is the unique point that maximizes the competitive ratio. Clearly, A's competitive ratio is $\eta(A) = \frac{d(O,A)}{d(O,P)-\sqrt{d^2(O,P)+d^2(O,A)-2d(O,P)d(O,A)\cos\angle AOP}}$. Let's define f as $f = d(O,P) - \sqrt{d^2(O,P)+d^2(O,A)-2d(O,P)d(O,A)\cos\angle AOP}$. We take the derivative of $\eta(A)$, and find that $\frac{\partial\eta(A)}{\partial d(O,P)} = -\frac{d(O,A)}{f^2}\cdot\frac{\partial f}{\partial d(O,P)}$. Since $\frac{\partial f}{\partial d(O,P)} =$
$1 - \frac{1}{2}\cdot\frac{1}{\sqrt{d^2(O,P)+d^2(O,A)-2d(O,P)d(O,A)\cos\angle AOP}}\cdot(2d(O,P)-2d(O,A)\cos\angle AOP) =$
$1 - \frac{d(O,P)-d(O,A)\cos\angle AOP}{\sqrt{(d(O,P)-d(O,A)\cos\angle AOP)^2+(d(O,A)\sin\angle AOP)^2}} > 0$, we find that $\frac{\partial\eta(A)}{\partial d(O,P)} < 0$. So $\eta(A)$ is a strictly decreasing function in $d(O,P)$.

□

We derive a routing method, which we call *Routing Method I* as follows: "when a node u needs to forward a message to a destination node v, u sends the message to a neighbor w such that $\angle wuv \leq \frac{\theta}{2} < \frac{\pi}{3}$." (With Algorithm III, such a neighbor w is

guaranteed to exist.)

Theorem 5.8 *In a network constructed by using Algorithm III, the length distortion of the network is at most* $\frac{1}{1-\sqrt{2-2\cos\frac{\theta}{2}}}$.

Proof: First let's take a look at Lemma 5.3. In that lemma, it is said that $\eta(A)$ is strictly decreasing in $d(O,P)$, so $\eta(A) < \frac{d(O,A)}{d(O,A)-\sqrt{d^2(O,A)+d^2(O,A)-2d(O,A)d(O,A)\cos\angle AOP}} = \frac{1}{1-\sqrt{2-2\cos\angle AOP}}$.

For any two nodes u and v, we use Routing Method I to get a path from u to v. Let's say the path consists of nodes '$u = w_1$, w_2, w_3, $\cdots$, $w_{k+1} = v$'. Then the length distortion of this path is $\frac{\sum_{i=1}^{k} d(w_i,w_{i+1})}{d(w_1,w_{k+1})} = \frac{\sum_{i=1}^{k} d(w_i,w_{i+1})}{\sum_{i=1}^{k}[d(w_i,w_{k+1})-d(w_{i+1},w_{k+1})]}$. From Lemma 5.3 and our analysis at the beginning of this proof, we can see that for every i ($1 \le i \le k$), $\frac{d(w_i,w_{i+1})}{d(w_i,w_{k+1})-d(w_{i+1},w_{k+1})} < \frac{1}{1-\sqrt{2-2\cos\frac{\theta}{2}}}$. So both the length distortion of this path, and the length distortion of the network, is at most $\frac{1}{1-\sqrt{2-2\cos\frac{\theta}{2}}}$.

□

The proof of Theorem 5.8 shows that even by using Routing Method I, a geographic routing scheme, the length distortions of the routing paths can still be upper bounded by $\frac{1}{1-\sqrt{2-2\cos\frac{\theta}{2}}}$.

5.6.4 Hop Distortion, Node Degree and Coverage Radius

The hop distortion for networks constructed using Algorithm III is ∞. That's because even though the node density is 1 node per unit area, the following phenomenon is still certainly going to happen somewhere in the network: 'for two nodes u and v, the nodes between and around them are so dense that they all have very small coverage radii; as a result, every routing path from u to v consists of a very large number of edges (which is impossible to bound from above).'

The distributions of the node degree and the coverage radius can be computed in similar ways as for Algorithm I. Clearly, both of them are functions with an exponentially decreasing tail. Therefore both the expectations and variances of the node degree and the coverage radius are constants (instead of ∞) (that depend on θ).

5.7 Algorithm IV

5.7.1 Definition

Algorithm IV consists of two steps, and it is defined as follows: "step 1, every node chooses its coverage radius to be the minimum value such that its cone angles are all smaller than or equal to θ, where θ is a fixed number and $0 < \theta < \frac{2\pi}{3}$; step 2, for those nodes whose coverage radii in step 1 were small than δ, they set their coverage radii to be δ. Here δ is a fixed number and $\delta > 0$."

5.7.2 Hop Distortion

Algorithm IV is an enhanced version of Algorithm III, so all the good connectivity, length-distortion and routing properties that Algorithm III can achieve can also be achieved by Algorithm IV. Now we show that Algorithm III can achieve more: for a network constructed using Algorithm IV, its hop distortion is upper-bounded by a constant.

Firstly we present a new routing method, which we shall call *Routing Method II*: "when a node u needs to forward a message to a destination node v, u sends the message to its neighbor w that satisfies the following two conditions: (1) if we define S as $S = \{p \mid p \text{ is a neighbor of } u, \text{ and } \angle puv \leq \frac{\theta}{2} < \frac{\pi}{3}\}$, then $w \in S$; (2) for every node $p \in S$, $d(w, v) \leq d(p, v)$." Routing method II is a special case of Routing Method I.

Lemma 5.4 *In a network constructed using Algorithm IV, if we use Routing Method II to do routing, then for any two consecutive edges in any routing path, the total length of those two edges is greater than δ.*

Proof: We assume Routing Method II is used for routing. Now consider the following scenario: "node O needs to forward a message to destination node P, and it sends the message to its neighbor C; node C sends the message to its neighbor D." Then to prove this lemma, we need to prove that $d(O, C) + d(C, D) > \delta$.

We assume that the coverage radius of node O is R. See Fig. 5.7. Clearly, node C must be in the sector OAB; since $d(D, P) < d(C, P)$ and C is the closest to P

than all the the other nodes in the sector OAB, D must be out of the sector OAB. Then there are three possibilities: Possibility 1, the line segment $\overline{CD}$ intersects $\overline{OA}$; Possibility 2, $\overline{CD}$ intersects the arc $\widehat{AB}$; Possibility 3, $\overline{CD}$ intersects $\overline{BO}$. We consider them one by one. Without loss of generality, from now on we assume node C is in the sector OAE.

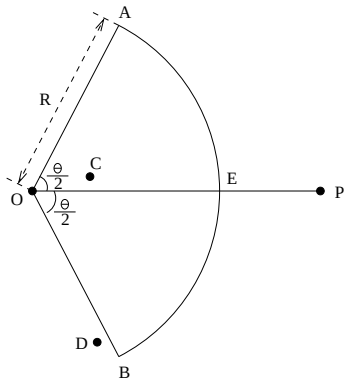


Figure 5.7: Two consecutive edges in a routing path.

(1) Firstly, assume $\overline{CD}$ intersects $\overline{OA}$. Then it is as shown in Fig. 5.8.

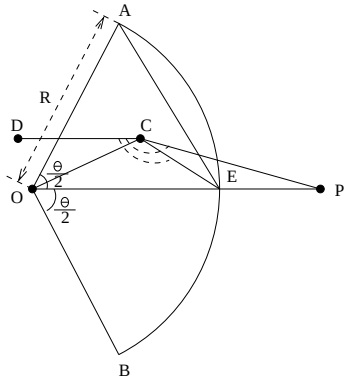


Figure 5.8: Two consecutive edges in a routing path, when $\overline{CD}$ intersects $\overline{OA}$.

Imagine there is a circle that passes through these three points — O, A and E — which we shall denote by 'circle OAE'. All the three inner angles in the triangle ΔOAE are smaller than $\frac{\pi}{2}$, so the center of the 'circle OAE' is inside the triangle ΔOAE — so the sector OAE is totally contained inside the 'circle OAE'; then since

node C lies in the sector OAE, C is in the 'circle OAE' — so $\angle OCE \geq OAE$. Then $\angle DCP \geq \angle OCE \geq \angle OAE = \angle OEA = \frac{\pi - \angle AOE}{2} = \frac{\pi - \frac{\theta}{2}}{2} > \frac{\pi - \frac{2\pi/3}{2}}{2} = \frac{\pi}{3} > \frac{\theta}{2}$. Also, $2\pi - \angle DCP \geq \angle ACP > \angle AOP = \frac{\theta}{2}$. That contradicts the fact that C decided to send the message to D based on the Routing Method II. So the case we are considering here is impossible.

(2) Secondly, assume $\overline{CD}$ intersects the arc $\widehat{AB}$, as shown in Fig. 5.9. In this case, clearly $d(O, C) + d(C, D) > R \geq \delta$, which is the conclusion we need.

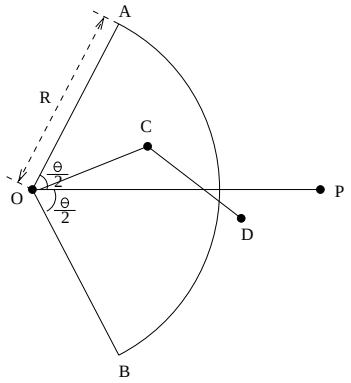


Figure 5.9: Two consecutive edges in a routing path, when $\overline{CD}$ intersects the arc $\widehat{AB}$.

(3) Thirdly, assume $\overline{CD}$ intersects $\overline{OB}$. Then it is as shown in Fig. 5.10.

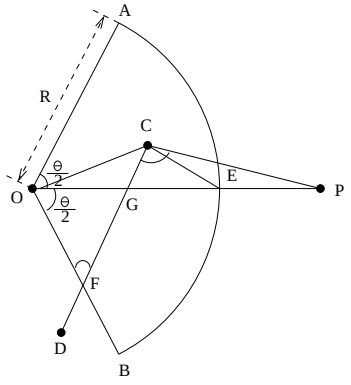


Figure 5.10: Two consecutive edges in a routing path, when $\overline{CD}$ intersects $\overline{OB}$.

Let's see how we can adjust the positions of all the nodes to minimize the value of $d(O, C) + d(C, D)$. The following are the 2 steps for that adjustment: "(i) We make

node D to be infinitely close to the point F. (ii) We know $\angle PCF = \angle PCD \leq \frac{\theta}{2} < \frac{\pi}{3}$, so $\angle OFC = \pi - \angle FOP - \angle OGF \leq \pi - \angle PCF - \angle CGP = \angle CPO < \angle CEO < \frac{\pi}{2}$. So to minimize the value of $d(C, F)$ (which now equals $d(C, D)$), we need to make $\angle PCF = \frac{\theta}{2}$, and make node P be infinitely close to the point E." After the above two adjustment steps, the situation becomes as shown in Fig. 5.11.

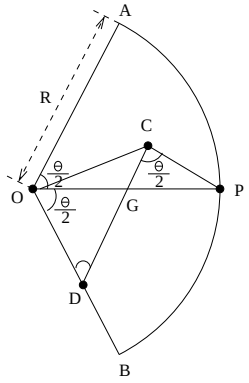


Figure 5.11: Two consecutive edges in a routing path, when $\overline{CD}$ intersects $\overline{OB}$.

Now let's temporarily assume that the position of node D is fixed, and let's see where node C needs to be in order to minimize $d(O, C) + d(C, D)$. Imagine there is a circle that passes through these three nodes — O, P and D — which we shall call 'circle OPD'. (Circle OPD is the dashed circle in Fig. 5.12.) Since $\angle POD = \angle PCD = \frac{\theta}{2}$, C must be on the circle OPD. C is also in the sector OAP, so node C can only lie on the arc $\widehat{HP}$. (The arc $\widehat{HP}$ is part of the circle OPD, and H is the intersection point of $\overline{OA}$ and the circle OPD.) But at which position on the arc $\widehat{HP}$ should node C be in order to minimize $d(O, C) + d(C, D)$?

On the two-dimensional plane, given any number s such that $s > d(O, D)$, if we find out those points p such that $d(O, p) + d(p, D) = s$, then we'll see that those points form an ellipse with O and D as its foci. (Such an ellipse is shown in dotted line in Fig. 5.12.) Let's gradually increase the value of s, and as a result the ellipse grows. Once the ellipse intersects the arc $\widehat{HP}$, the first intersection point is where node C should be in order to minimize $d(O, C) + d(C, D)$. Clearly this intersection point is either point H or point P. If the intersection point is P, then we already

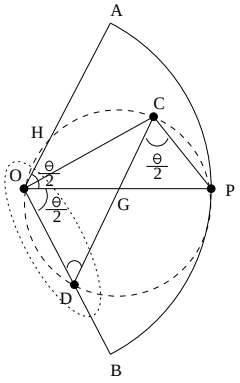


Figure 5.12: Two consecutive edges in a routing path, when $\overline{CD}$ intersects $\overline{OB}$.

have $d(O,C) + d(C,D) = d(O,P) + d(C,D) > d(O,P) \geq \delta$. So in the rest of the proof, we assume that node C lies where point H is, and we'll prove that even in this case, we still have $d(O,C) + d(C,D) > \delta$.

Since now node C lies where H is, the situation becomes as shown in Fig. 5.13. From now on we no longer assume that node D's position is fixed. Instead, we let node C take different positions on the line segment $\overline{OA}$; and the position of D becomes uniquely determined by C.

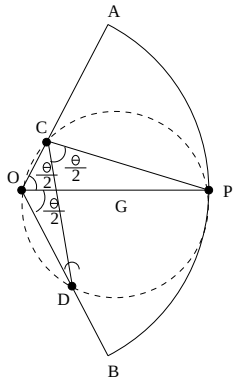


Figure 5.13: Two consecutive edges in a routing path, when $\overline{CD}$ intersects $\overline{OB}$.

Let's define t as $t = d(O,C)$, and define $f(t)$ as $f(t) = d(O,C) + d(C,D)$. (Here $0 < t \leq d(O,A) = R$.) Then $f(t) = d(O,C) + d(C,D) = t + \frac{d(C,D)\sin\angle CDO}{\sin\angle CDO} = t + \frac{d(O,C)\sin\angle COD}{\sin\angle CPO} = t + \frac{t\sin\theta}{\sin\angle CPO} = t + \frac{d(C,P)\cdot t\sin\theta}{d(C,P)\sin\angle CPO} = t + \frac{d(C,P)\cdot t\sin\theta}{d(O,C)\sin\angle COP} = t + \frac{d(C,P)\cdot t\sin\theta}{t\sin\frac{\theta}{2}} =$

$t + 2\cos\frac{\theta}{2}d(C,P) = t + 2\cos\frac{\theta}{2}\sqrt{d^2(O,C) + d^2(O,P) - 2d(O,C)d(O,P)\cos\angle COP} = t + 2\cos\frac{\theta}{2}\sqrt{t^2 + R^2 - 2tR\cos\frac{\theta}{2}}$.

Let's see when $f(t)$ takes its minimum value. Assume that there is a value t_0 ($0 < t_0 \leq R$) such that the derivative of $f(t)$ at $t = t_0$ — namely, $\frac{df(t)}{dt}|_{t=t_0}$ — equals 0. By solving the equation $\frac{df(t)}{dt} = 0$, we find that $t_0 = R\cos\frac{\theta}{2} - \frac{R\sin\frac{\theta}{2}}{\sqrt{4\cos^2\frac{\theta}{2}-1}}$. By using the fact that $t_0 > 0$ and the above formula, we find that $\theta < \frac{\pi}{2}$. Then we find that $f(t_0) = R\cos\frac{\theta}{2} + R\sin\frac{\theta}{2}\sqrt{4\cos^2\frac{\theta}{2} - 1} > R\cos\frac{\theta}{2} + R\sin\frac{\theta}{2} = R\sqrt{\cos^2\frac{\theta}{2} + \sin^2\frac{\theta}{2} + 2\cos\frac{\theta}{2}\sin\frac{\theta}{2}} = R\sqrt{1 + \sin\theta} > R \geq \delta$. (Note that $0 < \theta < \frac{2\pi}{3}$.) So $f(t_0) > \delta$. Now let's check the boundary values. When $t = d(O,A) = R$, clearly $f(t) > R \geq \delta$. When t approaches 0, the value of $f(t)$ approaches $2R\cos\frac{\theta}{2} > 2R\cos\frac{\pi}{3} = R \geq \delta$. Then it's not difficult to see that $f(t) > \delta$ for any $0 < t \leq R$. So we have proved the conclusion in this case.

□

Theorem 5.9 *In a network constructed using Algorithm IV, we use Routing Method II to do routing. For any two nodes O and P, let's use $H(O,P)$ to denote the number of edges in the routing path from O to P. Assume $H(O,P) \geq 2$. Then the hop distortion for that routing path, $\frac{H(O,P)}{d(O,P)}$, is upper bounded as follows: $\frac{H(O,P)}{d(O,P)} < \frac{2}{d(O,P)} + \frac{2}{\delta(1-\sqrt{2-2\cos\frac{\theta}{2}})}$.*

Proof: According to Theorem 5.8, the length of the routing path from O to P is at most $d(O,P) \cdot \frac{1}{1-\sqrt{2-2\cos\frac{\theta}{2}}}$. The total length of any two consecutive edges in the routing path is more than δ (by Lemma 5.4), so the first $2\lfloor\frac{H(O,P)-1}{2}\rfloor$ edges in the routing path have a total length of more than $\delta\lfloor\frac{H(O,P)-1}{2}\rfloor$ — but that total length is less than the length of the routing path. So $\delta\lfloor\frac{H(O,P)-1}{2}\rfloor < d(O,P) \cdot \frac{1}{1-\sqrt{2-2\cos\frac{\theta}{2}}}$. As a result, $\frac{H(O,P)}{d(O,P)} \leq \frac{2\lfloor\frac{H(O,P)-1}{2}\rfloor+2}{d(O,P)} < \frac{2}{d(O,P)} + \frac{2}{\delta(1-\sqrt{2-2\cos\frac{\theta}{2}})}$.

□

In Theorem 5.9, clearly $d(O,P) > \delta$. So we get the following conclusion:

Theorem 5.10 *The hop distortion for a network constructed using Algorithm IV is less than $\frac{2}{\delta}(1 + \frac{1}{1-\sqrt{2-2\cos\frac{\theta}{2}}})$.*

We see that even when Routing Method II — a geographic routing method — is used, the hop distortion and length distortion for the routing paths are still upper-bounded by constants.

5.7.3 Node Degree and Coverage Radius

The distributions of node degree and coverage radius for Algorithm IV can be computed by using the corresponding distributions for Algorithm III. They are again functions with exponential decreasing tails, so both the expectations and the variances of the node degree and the coverage radius are constants (instead of ∞) that depend on θ and δ.

5.8 Algorithm V

5.8.1 Definition

Algorithm V is defined as follows: "for every node u, it *disregards* those nodes whose distance to u is less than Δ, and chooses its coverage radius to be the minimum value such that its cone angles (which are caused only by those neighbors whose distance to u is at least Δ) are all smaller than or equal to θ. Here θ and Δ are fixed numbers, $0 < \theta < \frac{2\pi}{3}$, and $\Delta > 0$."

Please note that although a node disregards its neighbors within distance Δ when determining its coverage radius, it still connects itself to all the nodes within its coverage radius.

5.8.2 Hop Distortion and Other Performance Measurements

We can easily see that the upper bound for length distortion shown in Theorem 5.8 also holds for Algorithm V. In a network constructed using Algorithm V, we present the following routing method which we shall call *Routing Method III*: "when u needs to forward a message to a destination node v, if v is not a neighbor of u, then u sends the message to a neighbor w such that $d(u, w) \geq \Delta$ and $\angle wuv \leq \frac{\theta}{2}$; otherwise, u

directly sends the message to v." Routing Method III is a special case of Routing Method I. The routing path from u to v based on Routing Method III has a length that is at most $d(u,v) \cdot \frac{1}{1-\sqrt{2-2\cos\frac{\theta}{2}}}$ (see the proof of Theorem 5.8); and in that routing path the length of every edge except the last one is at least Δ. So similar to the proofs of Theorem 5.9 and Theorem 5.10, we can prove the following:

Theorem 5.11 *In a network constructed using Algorithm V, we use Routing Method III to do routing. For any two nodes O and P, let's use $H(O,P)$ to denote the number of edges in the routing path from O to P. Assume $H(O,P) \geq 2$. Then the hop distortion for that routing path, $\frac{H(O,P)}{d(O,P)}$, is upper bounded as follows: $\frac{H(O,P)}{d(O,P)} \leq \frac{1}{d(O,P)} + \frac{1}{\Delta(1-\sqrt{2-2\cos\frac{\theta}{2}})}$.*

Theorem 5.12 *The hop distortion for a network constructed using Algorithm V is at most $\frac{1}{\Delta}(1 + \frac{1}{1-\sqrt{2-2\cos\frac{\theta}{2}}})$.*

The distributions of node degree and coverage radius for Algorithm V can be computed by using the corresponding distributions for Algorithm III (and the methods here are simpler than those for Algorithm IV). The expectations and variances of those two random variables are also constants (instead of ∞) that depend on θ and Δ. We comment that although the proofs for Algorithm V are simpler than their counterparts for Algorithm IV, it is at the cost of its performance — when a node is determining its coverage radius, it disregards its closest neighbors, and that may make its coverage radius a little larger than necessary.

5.9 Appendix: The One-Hop Progress in a Network Constructed Using Algorithm I

In this appendix, we show how to compute the distribution of the one-hop progress for a network constructed using Algorithm I. When a node u needs to forward a message to a destination node v, u sends the message to one of its neighbors w — then $d(u,v) - d(w,v)$ is called the one-hop progress. One-hop progress shows how

effectively messages can be routed in a network; and it is a topic studied frequently [33], [56], [82], [88]. The solution we present here uses novel analysis techniques that have not appeared in previous publications; and it can be generalized to compute the one-hop progress for extended versions of Algorithm I (e.g., Algorithm III).

We assume here that a node always forwards the message to the neighbor that makes the most one-hop progress. And we assume the destination is infinitely far away. This is a basic model used in many works [33], [82]. And for very large networks it is a good approximation — two randomly sampled source and destination nodes are most likely to be very far away from each other, and the distance from most of the intermediate nodes on the routing path to the destination will be large.

We impose an $x - y$ coordinate system on the 2-dimensional plane. And without loss of generality, we assume there is a node O at the original point, which needs to forward a message to the destination node whose coordinates are $(+\infty, 0)$. Then among the neighbors of O, the one that makes the most progress is the one with the greatest x-coordinate. Let's use ζ to denote the maximum x-coordinate of the neighbors of O. Then ζ is the one-hop progress for O.

Let's use v_L to denote the farthest neighbor of O. Then $d(O, v_L)$ is the coverage radius of O. We define C as the disk centered at O whose radius is $d(O, v_L)$. From the definition of Algorithm I, we know that if we disregard v_L, then the cone-angles caused by the rest of the neighbors of O would not be all smaller than π — then let's use v_A and v_B to denote the two neighbors whose corresponding cone-angle is no less than π. (See Fig. 5.14 for an illustration. The node at the origin is O.) We define M as the sector in C with O, E, F as its three extreme points (see Fig. 5.14 for the illustration), whose angle $\angle EOF = \angle v_A O v_B \leq \pi$. Clearly, $v_L \notin M$, and all the other neighbors of O are in M.

The one-hop progress for O — that is, ζ — has to be in the range of $(0, d(O, v_L)]$. Let x be a number where $0 < x \leq d(O, v_L)$. Our goal in this appendix is to compute $Pr\{\zeta \leq x\}$, namely, the cumulative distribution function of ζ.

For the sake of explanation, let's define some more notations that will be used later on. Please see Fig. 5.14 for the illustration. X is a point whose x-coordinate

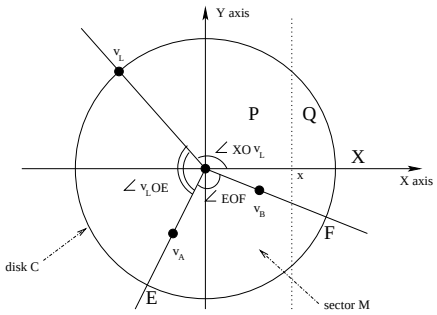


Figure 5.14: O's neighbors

is positive and whose y-coordinate is 0. We can think of X as the point whose coordinates are $(+\infty, 0)$. We partition the disk C into two parts — P and Q. P contains all those points in C whose x-coordinates are no greater than x; and Q contains all those points in C whose x-coordinates are greater than x. So if $\zeta \leq x$, then all the neighbors of O are in P.

How large can the three angles — $\angle XOv_L$, $\angle v_LOE$ and $\angle EOF$ — be, if $\zeta \leq x$? Since v_L is on the circumference of C, we get $\arccos \frac{x}{d(O,v_L)} \leq \angle XOv_L \leq 2\pi - \arccos \frac{x}{d(O,v_L)}$. From the definition of Algorithm I, we see that $0 < \angle v_LOE < \pi$, $\angle EOF \leq \pi$, and $\angle v_LOE + \angle EOF > \pi$. Therefore $\pi - \angle v_LOE < \angle EOF \leq \pi$. So in summary, there exist three constraints: (1) $\arccos \frac{x}{d(O,v_L)} \leq \angle XOv_L \leq 2\pi - \arccos \frac{x}{d(O,v_L)}$; (2) $0 < \angle v_LOE < \pi$; (3) $\pi - \angle v_LOE < \angle EOF \leq \pi$.

Let's use ε to denote the following event: "$\zeta \leq x$; $r \leq d(O, v_L) < r + dr$, $\beta \leq \angle XOv_L < \beta + d\beta$, $\theta \leq \angle v_LOE < \theta + d\theta$, $\varphi \leq \angle EOF < \varphi + d\varphi$." (Here r, β, θ and φ are positive numbers, and $dr \to 0$, $d\beta \to 0$, $d\theta \to 0$, $d\varphi \to 0$.) Also, let's define f_ε as $f_\varepsilon = \frac{Pr\{\varepsilon\}}{d\varphi d\theta d\beta dr}$. Then the cumulative distribution function of ζ is:

$$\begin{aligned} & F_\zeta(x) = Pr\{\zeta \leq x\} \\ = & \int_x^\infty \int_{\arccos \frac{x}{r}}^{2\pi - \arccos \frac{x}{r}} \int_0^\pi \int_{\pi-\theta}^\pi f_\varepsilon d\varphi d\theta d\beta dr \end{aligned}$$

Everything we are considering here is symmetric in respect to the x-axis, so we

can simplify the above formula to be:

$$\begin{aligned} & F_\zeta(x) = Pr\{\zeta \le x\} \\ = \; & 2\int_x^\infty \int_{\arccos\frac{x}{r}}^{\pi} \int_0^\pi \int_{\pi-\theta}^{\pi} f_\varepsilon d\varphi d\theta d\beta dr \\ & \qquad\qquad \cdots\cdots \text{Formula(1)} \end{aligned}$$

All we need to know now is how to compute $Pr\{\varepsilon\}$ (which equals $f_\varepsilon d\varphi d\theta d\beta dr$).

When the event ε is true, node v_L needs to exist in such a tiny space: "when v_L is in that space, $d(O, v_L) \in [r, r + dr)$, and $\angle XOv_L \in [\beta, \beta + d\beta)$." Let's use T_1 to denote the area of that tiny space. Then $T_1 = rd\beta dr$. So the probability that a node (which is v_L) exists in that space is $rd\beta dr$. (Remember that the nodes follow a Poisson point process.)

When the event ε is true, node v_A needs to exist in such a tiny space: "firstly, that tiny space is inside P; secondly, when v_A is in that tiny space, $\angle v_L O v_A = \angle v_L OE \in [\theta, \theta + d\theta)$." Let's use T_2 to denote the area of that tiny space. It is not difficult to find out that T_2's value is as follows:

- If $\arccos\frac{x}{r} \le \beta + \theta < 2\pi - \arccos\frac{x}{r}$, then $T_2 = \frac{1}{2}r^2 d\theta$.
- If $2\pi - \arccos\frac{x}{r} \le \beta + \theta < 2\pi$, then $T_2 = \frac{1}{2}[\frac{x}{\cos(\beta+\theta)}]^2 d\theta$.

The probability that a node (which is v_A) exists in that space of area T_2 is just T_2.

When the event ε is true, node v_B needs to exist in such a tiny space: "firstly, that tiny space is inside P; secondly, when v_B is in that tiny space, $\angle v_A O v_B = \angle EOF \in [\varphi, \varphi + d\varphi)$." Let's use T_3 to denote the area of that tiny space. It is not difficult to find out that T_3's value is as follows:

- If $2\pi - \arccos\frac{x}{r} \le \beta + \theta + \varphi < 2\pi + \arccos\frac{x}{r}$, then $T_3 = \frac{1}{2}[\frac{x}{\cos(\beta+\theta+\varphi)}]^2 d\varphi$; otherwise, $T_3 = \frac{1}{2}r^2 d\varphi$.

The probability that a node (which is v_B) exists in that space of area T_3 is just T_3.

Let's define S as the intersection of P and M — namely, $S = P \cap M$. Let's use $|S|$ to denote the area of S. Then, a straightforward geometric analysis tells us that when ε is true, the value of $|S|$ is:

- If $\beta + \theta < 2\pi - \arccos\frac{x}{r}$ and $\beta + \theta + \varphi < 2\pi - \arccos\frac{x}{r}$, then $|S| = \frac{1}{2}r^2\varphi$.
- If $\beta + \theta < 2\pi - \arccos\frac{x}{r}$ and $2\pi - \arccos\frac{x}{r} \leq \beta + \theta + \varphi < 2\pi + \arccos\frac{x}{r}$, then $|S| = \frac{1}{2}r^2(2\pi - \arccos\frac{x}{r} - \beta - \theta) + \frac{1}{2}x\sqrt{r^2 - x^2} + \frac{1}{2}x^2\tan(\beta + \theta + \varphi)$.
- If $\beta + \theta < 2\pi - \arccos\frac{x}{r}$ and $2\pi + \arccos\frac{x}{r} \leq \beta + \theta + \varphi$, then $|S| = \frac{1}{2}r^2\varphi - r^2\arccos\frac{x}{r} + x\sqrt{r^2 - x^2}$.
- If $2\pi - \arccos\frac{x}{r} \leq \beta + \theta < 2\pi$ and $2\pi - \arccos\frac{x}{r} \leq \beta + \theta + \varphi < 2\pi + \arccos\frac{x}{r}$, then $|S| = \frac{1}{2}x^2\tan(2\pi - \beta - \theta) + \frac{1}{2}x^2\tan(\beta + \theta + \varphi)$.
- If $2\pi - \arccos\frac{x}{r} \leq \beta + \theta < 2\pi$ and $2\pi + \arccos\frac{x}{r} \leq \beta + \theta + \varphi$, then $|S| = \frac{1}{2}x^2\tan(2\pi - \beta - \theta) + \frac{1}{2}x\sqrt{r^2 - x^2} + \frac{1}{2}r^2(\beta + \theta + \varphi - 2\pi)$.

When the event ε is true, all the neighbors of O excluding v_L, v_A and v_B are inside S. When we exclude S from the disk C, the area of the remaining space is $\pi r^2 - |S|$, and the probability that 'the Poisson point process places no node inside that remaining space' is $e^{-\pi r^2 + |S|}$. So the probability that 'all the neighbors of O excluding v_L, v_A and v_B are inside S' equals $e^{-\pi r^2 + |S|}$.

The event ε happens if and only if the following events happen: "the nodes v_L, v_A and v_B exist in those tiny spaces we discussed above, and all the neighbors of O excluding v_L, v_A and v_B are inside S." Since the nodes on the 2-dimensional plane follow the Poisson point process, the above events are all independent. Therefore, we get:

$$Pr\{\varepsilon\} = T_1T_2T_3e^{-\pi r^2 + |S|}$$

Again, note that $Pr\{\varepsilon\} = f_\varepsilon d\varphi d\theta d\beta dr$. So by plugging the above equation into Formula (1), we get the integral formula that computes the cumulative distribution

function of the one-hop progress.

Chapter 6

Future Directions

This thesis has studied two new topics — data placement using erasure-correcting codes and localized topology control for nodes in normed spaces — and opened several new research directions (such as multi-cluster interleaving). All those are very broad fields and much more related work is still left to be done. The numerous areas very worth exploring include:

- *Algorithms for data placement on general graphs and graphs with special features.* Data placement problems for general graphs are often NP-hard to optimize. How to find approximation solutions to data placement using erasure-correcting codes needs considerably more study. Also, for graphs with special features (such as graphs with power-law distributions of degrees and unit-disk graphs), it is interesting to know how the features will influence the data placement performance and the complexity of the corresponding algorithms.

- *Multi-cluster interleaving on general graphs, and multi-cluster interleaving on paths and cycles for* $K \geq L + 2$.

- *Cooperative data access.* Having users cooperatively download data saves bandwidth. One example is using a multicast tree for users accessing the same data. When data are stored distributively in the network using erasure-correcting codes, cooperative data access becomes even more practical, because now users have the freedom to join any set of the many multicast trees corresponding

to different codeword components, at the time of their choosing. How to efficiently implement such cooperative data access schemes, however, still needs to be known.

- *Incremental embedding algorithms for peer-to-peer type networks.* Currently proposed methods for embedding peer-to-peer type networks into normed spaces usually assume that all the nodes are embedded at the same time. However, peer-to-peer type networks typically have dynamic membership; so how to incrementally embed newly joined nodes is an intriguing open problem.

- *The topology control for overlay networks.* The topology-control algorithm presented in this thesis can be extended for overlay networks in the Internet. And the topology constructed using the algorithm can be further optimized by adding some long edges for each node. How to select the long edges to shorten the routing paths is very related to the topic of random graph constructions.

- *Relationship between topology-control algorithms and the evolution of networks.* There has been work on studying the evolution of networks under certain topology control models, such as preferential attachment or random attachment. Comparatively, the topology control algorithm presented in this thesis allows much fewer random or greedy decisions on connectivity. Study on the relationship between different types of topology control algorithms and the evolution of networks will be influential for the network design theory.

Research on the above areas concerns both theories and practical systems. And that is where the author of this thesis stores his never ending interest.

Bibliography

[1] K. A. S. Abdel-Ghaffar, "Achieving the Reiger Bound for Burst Errors Using Two-dimensional Interleaving Schemes", in *Proceedings of the IEEE International Symposium on Information Theory*, pp. 425, Germany, 1997.

[2] Akamai, http://www.akamai.com.

[3] B. F. AlBdaiwi and B. Bose, "Quasi-Perfect Lee Distance Codes", *IEEE Transactions on Information Theory*, vol. 49, no. 6, pp. 1535-1539, 2003.

[4] B. F. AlBdaiwi and B. Bose, "On Resource Placements in 3D Tori", *Journal of Parallel and Distributed Computing*, vol. 63, pp. 838-845, 2003.

[5] C. Almeida and R. Palazzo, "Two-dimensional Interleaving Using the Set Partition Technique", in *Proceedings of the IEEE International Symposium on Information Theory*, pp. 505, Trondheim, Norway, 1994.

[6] K. Alzoubi, X. Li, Y. Wang, P. Wan and O Frieder, "Geometric Spanners for Wireless Ad Hoc Networks", *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, pp. 408–421, 2003.

[7] J. Astola, "A Note on Perfect Lee-codes over Small Alphabets", *Discrete Applied Mathematics*, vol. 4, pp. 227-228, 1982.

[8] J. Astola, "An Elias-Type Bound for Lee Codes over Large Alphabets and Its Applications to Perfect Codes", *IEEE Transactions on Information Theory*, vol. IT-28, no. 1, pp. 111-113, 1982.

[9] B. Awerbuch, Y. Bartal and A. Fiat, "Competitive Distributed File Allocation", in *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing*, pp. 164–173, 1993.

[10] M. M. Bae and B. Bose, "Resource Placement in Torus-based Networks", *IEEE Transactions on Computers*, vol. 46, no. 10, pp. 1083–1092, 1997.

[11] M. Blaum, J. Bruck and A. Vardy, "Interleaving Schemes for Multidimensional Cluster Errors", *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 730–743, 1998.

[12] E. R. Berlekamp, *Algebraic Coding Theory*, Aegean Park Press, 1984.

[13] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Perterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski and J. Webb, "iWarp: An Integrated Solution to High-Speed Parallel Computing", in *Proceedings of IEEE Supercomputing'88*, pp. 330–339, 1988.

[14] P. Bose, P. Morin, I. Stojmenovic and J. Urrutia, "Routing with Guaranteed Delivery in Ad Hoc Wireless Networks", in *Proceedings of the 3rd ACM Intl. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications DIAL M99*, pp. 48–55, 1999.

[15] I. M. Boyarinov, "Three-dimensional Cluster Error-Correcting Array Codes", in *Proceedings of the IEEE International Symposium on Information Theory*, pp. 118, Lausanne, Switzerland, 2002.

[16] J. W. Byers, M. Luby and M. Mitzenmacher, "Accessing Multiple Mirror Sites in Parallel: Using Tornado Codes to Speed Up Downloads", in *Proceedings of IEEE Infocom'99*, pp. 275-283, Boston Univ., MA, USA, 1999.

[17] J. W. Byers, M. Luby, M. Mitzenmacher and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data", in *Proceedings of ACM SIGCOMM'98*, Vancouver, Canada, Sep., 1998.

[18] Mayur Datar, "Butterflies and Peer-to-Peer Networks", in *Proceedings of European Symposium of Algorithms*, 2002.

[19] L. W. Dowdy and D. V. Foster, "Comparative Models of the File Assignment Problem", *Computing Surveys*, vol. 14, no. 2, pp. 287–313, 1982.

[20] D. Dubhashi, O. Haggstrom and A. Panconesi, "Connectivity Properties of Bluetooth Wireless Networks", manuscript, 2003.

[21] T. Etzion and A. Vardy, "Two-dimensional Interleaving Schemes with Repetitions: Constructions and Bounds", *IEEE Transactions on Information Theory*, vol. 48, no. 2, pp. 428–457, 2002.

[22] P. Ganesan, Q. Sun and H. Garcia-Molina, "YAPPERS: A Peer-to-Peer Lookup Service over Arbitrary Topology", in *Proceedings of IEEE Infocom*, pp. 1250–1260, 2003.

[23] 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.

[24] E. N. Gilbert, "Random Plane Networks", *Journal of SIAM*, vol. 9, pp. 533–543, 1961.

[25] Gnutella, http://www.gnutella.com.

[26] S. W. Golomb and E. C. Posner, "Rook Domains, Latin Squares, Affine Planes, and Error-Distributing Codes", *IEEE Transactions on Information Theory*, vol. 10, pp. 196-208, July, 1964.

[27] S. W. Golomb and L. R. Welch, "Perfect Codes in the Lee Metric and the Packing of Polyominoes", *SIAM J. Appl. Math.*, vol. 18, no. 2, pp. 302–317, 1970.

[28] S. Gravier, M. Mollard and C. Payan, "On the Non-existence of 3-Dimensional Tiling in the Lee Metric", *Europ. J. Combinatorics*, vol. 19, pp. 567-572, 1998.

[29] S. Gravier, M. Mollard and C. Payan, "Variations on Tilings in the Manhattan Metric", *Geometriae Dedicata*, vol. 79, pp. 265-273, 1999.

[30] G. Grimmett, *Percolation*, Springer-Verlag, Berlin Heidelberg, 1999.

[31] O. Haggstrom and R. Meester, "Nearest Neighbor and Hard Sphere Models in Continuum Percolation", *Random Structures and Algorithms*, vol. 9, no. 3, pp. 295–315, 1996

[32] D. B. Holdridge and J. A. Davis, "Comment on 'A Note on Perfect Lee-Codes' ", *Discrete Applied Mathematics*, vol. 3, pp. 221, 1981.

[33] T.-C. Hou and V. O. K. Li, "Transmission Range Control in Multihop Packet Radio Networks", *IEEE Transactions on Communications*, vol. COM-34, no. 1, pp. 38–44, 1986.

[34] P. Indyk, "Algorithmic Applications of Low-distortion Geometric Embeddings", in *Proceedings of the Annual Symposium on Foundations of Computer Science*, pp. 10–33, 2001.

[35] A. Jiang and J. Bruck, "Diversity Coloring for Information Storage in Networks", in *Proceedings of the 2002 IEEE International Symposium on Information Theory*, pp. 381, Lausanne, Switzerland, 2002.

[36] A. Jiang and J. Bruck, "Memory Allocation in Information Storage Networks", in *Proceedings of the 2003 IEEE International Symposium on Information Theory*, pp. 453, Yokohama, Japan, 2003.

[37] A. Jiang and J. Bruck, "Diversity Coloring for Distributed Data Storage in Networks", submitted to *IEEE Transactions on Information Theory*.

[38] A. Jiang and J. Bruck, "Multi-cluster Interleaving on Linear Arrays and Rings", in *Proceedings of the Seventh International Symposium on Communication Theory and Applications (ISCTA'03)*, pp. 112-117, Ambleside, Lake District, UK, July, 2003.

[39] A. Jiang, M. Cook and J. Bruck, "Optimal t-Interleaving on Tori", to appear in *Proceedings of the IEEE International Symposium on Information Theory*, Chicago, USA, 2004.

[40] A. Jiang and J. Bruck, "Monotone Percolation and the Topology Control of Wireless Networks", submitted to the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004).

[41] 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*, vol. 12, no. 6, pp. 628–637, 2001.

[42] D. Karger and M. Ruhl, "Finding Nearest Neighbors in Growth-restricted Metrics", in *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC'02)*, pp. 741–750, 2002.

[43] B. Karp and H. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks", in *Proceedings of Mobicom'00*, Aug., 2000.

[44] Y. Ko and N. Vaidya, "Location-aided Routing LAR in Mobile Ad Hoc Networks", in *Proceedings of Mobicom'98*, Oct., 1998.

[45] E. Kranakis, H. Singh and J. Urrutia, "Compass Routing on Geogmetric Networks", in *Proceedings of 11th Canadian Conference on Computational Geometry*, Vancouver, Canada, Aug., 1999.

[46] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells and B. Zhao, "OceanStore: An Architecture for Global-scale Persistent Storage", in *Proceedings of the 9th Internatinal Conference Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*, pp. 190–201, 2000.

[47] J. F. Kurose and R. Simha, "A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems", *IEEE Transactions on Computers*, vol. 38, no. 5, pp. 705–717, 1989.

[48] T. Lepisto, "A Note on Perfect Lee-codes over Small Alphabets", *Discrete Applied Mathematics*, vol. 3, pp. 73–74, 1981.

[49] L. Li, J. Y. Halpern, P. Bahl, Y.-M. Wang and R. Wattenhofer, "Analysis of a Cone-based Distributed Topology Control Algorithm for Wireless Multi-hop Networks", in *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pp. 264-273, New Port, Rhode Island, USA, Aug., 2001.

[50] M. Lindwer, D. Marculescu, T. Basten, R. Zimmermann, R. Marculescu, S. Jung and E. Cantatore, "Ambient Intelligence Visions and Achievements: Linking Abstract Ideas to Real-World Concepts", in *Proceedings of the IEEE Design, Automation and Test in Europe Conf. (DATE)*, Munich, Germany, Mar., 2003.

[51] N. Linial, E. London and Y. Rabinovich, "The Geometry of Graphs and Some of Its Algorithmic Applications", *Combinatorica*, vol. 15, pp. 215–245, 1995.

[52] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Elsevier Science B. V., 1977.

[53] D. Malkhi, M. Naor and D. Ratajczak, "Viceroy: A Scalable and Dynamic Emulation of the Butterfly", in *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC '02)*, pp. 183-192, 2002.

[54] A. Mahanti, D. L. Eager, M. K. Vernon and D. Sundaram-Stukel, "Scalable On-demand Media Streaming with Packet Loss Recovery", in *Proceedings of ACM SIGCOMM 2001*, pp. 97-108, San Diego, Aug., 2001.

[55] Q. M. Malluhi and W. E. Johnston, "Coding for High Availability of a Distributed-Parallel Storage System", *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 12, pp. 1237–1252, 1998.

[56] A. Mann and J. Ruckert, "Transmission Range Control for Packet Radio Networks or Why Magic Numbers Are Distance Dependent", *Lecture Notes in Control and Information Sciences*, vol. 143, pp. 818-830, 1990.

[57] D. Marculescu, R. Marculescu, S. Park and S. Jayaraman, "Ready to Ware", *IEEE Spectrum*, pp. 28–32, Oct., 2003.

[58] J. Matousek, "Embedding Finite Metric Spaces into Euclidean Spaces", Chapter 15 of *Lectures on Discrete Geometry*, Springer, 2002.

[59] M. Mauve, J. Widmer and H. Hartenstein, "A Survey on Position-based Routing in Mobile Ad-hoc Networks", *IEEE Network*, vol. 15, no. 6, pp. 30–39, 2001.

[60] R. J. McEliece, *The Theory of Information and Coding*, Cambridge University Press, 2002.

[61] R. Meester and R. Roy, *Continuum Percolation*, Cambridge University Press, 1996.

[62] Y. Merksamer and T. Etzion, "On the Optimality of Coloring with a Lattice", manuscript.

[63] M. Naor and R. M. Roth, "Optimal File Sharing in Distributed Networks", *SIAM J. Comput.*, vol. 24, no. 1, pp. 158–183, 1995.

[64] I. Newman and Y. Rabinovich, "A Lower Bound on the Distortion of Embedding Planar Metrics into Euclidean Space", in *Proceedings of the Annual Symposium on Computational Geometry*, pp. 94–96, Barcelona, Spain, 2002.

[65] T. Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-based Approaches", in *Proceedings of the IEEE Infocom*, pp. 170-179, June, 2002.

[66] W. Oed, "Massively Parallel Processor System CRAY T3D", Technical Report, Cray Research GmbH, Nov., 1993.

[67] D. A. Patterson, G. A. Gibson and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks", in *Proceedings of SIGMOD Int. Conf. Data Management*, pp. 109–116, 1988.

[68] C. G. Plaxton, R. Rajaraman and A. W. Richa, "Accessing Nearby Copies of Replicated Objects in a Distributed Environment", in *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'97)*, pp. 311–320, New York, NY, USA, June, 1997.

[69] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker and I. Stoica, "Geographic Routing without Location Information", in *Proceedings of Mobicom 2003*, pp. 96–108, 2003.

[70] S. Ratnasamy and P. Francis and M. Handley and R. Karp and S. Shenker, "A Scalable Content-Addressable Network", in *Proceedings of ACM SIGCOMM*, pp. 161–172, Aug., 2001.

[71] A. I. Riihonen, "A Note on Perfect Lee-Codes", *Discrete Applied Mathematics*, vol. 2, pp. 259-260, 1980.

[72] P. Rodriguez, C. Spanner and E. W. Biersack, "Analysis of Web Caching Architectures: Hierarchical and Distributed Caching", *IEEE/ACM Transactions on Networking*, vol. 9, no. 4, pp. 404–418, 2001.

[73] P. Rodriguez and E. W. Biersack, "Dynamic Parallel Access to Replicated Content in The Internet", *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 455–465, 2002.

[74] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems", in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, Nov. 2001.

[75] M. Schwartz and T. Etzion, "Optimal 2-Dimensional 3-Dispersion Lattices", *Lecture notes in Computer Science 2643*, pp. 216–225, 2003.

[76] C. L. Seitz et al., "Submicron Systems Architecture Project Semi-Annual Technical Report", Technical Report Caltech-CS-TR-88-18, California Institute of Technology, Nov., 1988.

[77] Y. Shang, W. Ruml, Y. Zhang and M. Fromherz, "Localization from Mere Connectivity", in *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pp. 201–212, 2003.

[78] Y. Shavitt and T. Tankel, "Big-bang Simulation for Embedding Network Distances in Euclidean Space", in *Proceedings of IEEE Infocom*, pp. 1922-1932, San Francisco, CA, USA, 2003.

[79] C.-C. Shen, C. Srisathapornphat, R. Liu, Z. Huang, C. Jaikaeo and E. L. Lloyd, "CLTC: a Cluster-based Topology Control Framework for Ad Hoc Networks", *IEEE Transactions on Mobile Computing*, vol. 3, no. 1, pp. 18–32, 2004.

[80] A. Slivkins and J. Bruck, "Interleaving Schemes on Circulant Graphs with Two Offsets", to appear in *IEEE Transactions on Information Theory*.

[81] Ion Stoica, Robert Morris, David Liben-Nowell, David Karger, M. Frans Kaashoek, Frank Dabek and Hari Balakrishnan, "Chord: A Scalable Peer-to-peer Look-up Protocol for Internet Applications", *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[82] H. Takagi and L. Kleinrock, "Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals", *IEEE Transactions on Communications*, vol. COM-32, no. 3, pp. 246–257, 1984.

[83] X. Tang and S. T. Chanson, "Coordinated En-route Web Caching", *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 595–607, 2002.

[84] Tera Computer Systems, "Overview of the Tera Parallel Computer", 1993.

[85] R. Wattenhofer, L. Li, P. Bahl and Y.-M. Wang, "Distributed Topology Control for Power Efficient Operation in Multihop Wireless Ad Hoc Networks", in *Proceedings of IEEE Infocom 2001*, pp. 1388-1397, Apr., 2001.

[86] D. B. West, *Introduction to Graph Theory*, Englewood Cliffs, NJ: Prentice-Hall, 1996.

[87] F. Xue and P. R. Kumar, "The Number of Neighbors Needed for Connectivity of Wireless Networks", *Wireless Networks*, vol. 10, no. 2, pp. 169–181, 2004.

[88] M. Zorzi and R. R. Rao, "Geographic Random Forwarding (GeRaF) for Ad Hoc and Sensor Networks: Multihop Performance", *IEEE Transactions on Mobile Computing*, vol. 2, no. 4, pp. 337–348, 2003.