COMBINATORIAL APPROACHES TO SCALABILITY PROBLEMS IN STORAGE, NETWORKING, AND COMMUNICATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT
OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Joseph Clarence Koo
December 2011

This dissertation is online at: http://purl.stanford.edu/cs712sv8343

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

John Gill, III, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Stephen Boyd

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Amin Saberi

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

As computing systems have grown larger and more interconnected, every single layer in the underlying technologies has had to scale correspondingly in order to continue to support these systems. However, this has caused significant new challenges in system design, since smaller systems cannot simply be scaled-up and still work as a larger system. In this dissertation, we take a critical look at the scaling challenges in the storage, networking, and communications layers, by considering the combinatorial aspects of existing approaches. By recognizing the need for understanding the combinatorics of these problems, we are then able to provide efficient solutions that perform well even as problem domains get significantly larger.

With the ease of data creation and collection, it is now common to see data centers with thousands or even tens of thousands of storage nodes. Within such large distributed storage systems, multiple node failures are the norm rather than the exception. In the first part of this dissertation, we present a systematic method for designing storage systems where node repair is efficient even as the system scales to large numbers of nodes. In this scheme, data chunk replication is used to guarantee storage reliability. By distributing data chunks and their replicas through the storage system according to a Steiner system design, node repair can be heavily parallelized, thus resulting in higher data availability. We show that the design of such storage systems based on Steiner systems can be extended to scenarios where node sizes are much larger than replication degrees. Via these constructions, which are based on bipartite cage graphs and mutually orthogonal Latin squares, the resulting designs are guaranteed to require the fewest number of storage nodes for the given parameters. Furthermore, the scheme is scalable, since these storage systems can be easily

expanded without need for frequent reconfiguration.

Traffic on networks has also experienced significant growth in recent years, leading to growth in network sizes as well. However, existing methods for network throughput, based on routing algorithms, do not fully utilize the available network resources. Via coding within the network, it has been shown that it is possible to significantly increase data rates within a given network, compared to routing-based approaches. Oftentimes, though, determining the appropriate network code for large networks can be quite computationally complex. In the second part of this dissertation, we focus on the class of network coding problems known as the non-uniform demand problem, and show how network codes for these problems can be efficiently constructed even as the networks become larger. The non-uniform demand network coding problem is a single-source, multiple-sink network transmission problem where the sinks may have heterogeneous demands, and is generally an NP-complete problem. In this work, we present non-uniform network demand scenarios under which network coding solutions can be found in polynomial time. This is accomplished by relating the demand problem with the graph coloring problem and then applying results from the strong perfect graph theorem.

Even within the link layer, the proliferation in connected devices has also caused its own scalability challenges. Within a wireless channel shared by multiple transmitters and receivers, interference among the many users can cause severe degradations in connection quality, which only gets worse as more users share the channel. A technique known as interference alignment, however, has been shown to enable high data rates in an interference channel, even as the number of interfering users gets large. In an ergodic setting, interference alignment can be achieved by pairing complementary channel realizations in order to amplify signals and cancel interference. Unfortunately, such a scheme is prone to large delays in decoding message symbols when there are many users. Thus in the third part of this dissertation we take an in-depth look at the causes of such delays and propose better schemes that mitigate these issues. We show that delay can be mitigated by using outputs from potentially more than two channel realizations, at the possible cost of reduced data rate. We further demonstrate the tradeoff between rate and delay via a timesharing strategy.

Acknowledgements

The long road toward completing the Ph.D. has been sometimes bumpy and sometimes smooth, but fortunately for me it has not been a solitary experience. Although my name appears as the author on this dissertation, the finished product would have been impossible without the guidance and support of so many others. To them, I owe many debts of gratitude for much encouragement and well-placed counsel along the way.

I have been fortunate to have had John Gill for my advisor, as the work in this thesis has benefited greatly from his depth of knowledge and insight. There have been countless times in which a well-placed question or seemingly random suggestion has pushed the work in a fruitful direction. I'm constantly amazed at his curiosity and attention to detail in a variety of topics. As an advisor he has always given me the freedom to pursue the research questions that I found interesting, which made the entire process much more enjoyable as a result. I feel privileged to be a member of the small cadre of students who have called him advisor.

I'd also like to thank my associate advisor Amin Saberi, whose encouragement and enthusiasm for my work enabled me to continue in my research. He has been a source of good pointers when they have been necessary, and his many suggestions on related topics have been immensely helpful. The approaches in this dissertation have also benefited directly from his well-taught classes on algorithms, combinatorial optimization, and discrete mathematics.

I'm extremely grateful that I have been able to work more directly with Stephen Boyd the last few years at Stanford. I have benefited a lot not just from taking his classes, but also from being on the other end and TAing his classes. Many people

know Stephen Boyd as a phenomenal teacher, but I have been very lucky to know him also as a phenomenal mentor. I'd like to thank him for much good advice and support over the years when I have needed them the most. Much of the work in this dissertation bears his fingerprints: from him, I have learned a lot about how to think through challenging problems and how to be more precise in thinking and writing—abilities that I hope to but may never be able to master to the same degree. I'd also like to thank him for agreeing to serve on the reading committee for this dissertation.

Thanks to Hector Garcia-Molina for chairing my oral examination committee, and also for some very thoughtful discussions on the systems design aspects of my work. Also a special thanks to Mohsen Bayati for agreeing to serve on my orals committee—especially on extremely short notice.

I would also like to thank two very important mentors from my undergraduate days at Caltech: Babak Hassibi gave me one of my first tastes of research, and always believed that I was capable of pursuing the Ph.D. He was responsible for "gentle nudges" toward the Electrical Engineering Ph.D. program at Stanford—for which I am grateful. Payman Arabshahi, with his cheerful attitude, has always been available for good counsel on my pursuits, and has provided encouraging advice at various points after I left Caltech.

I've appreciated the hard work of Denise Murphy and Katt Clark to make sure logistical issues were never a problem—especially Denise for smoothing out every administrative issue I've ever had and for always making sure that everything was going well for me.

While in the office, I have benefited from fruitful discussions about research, life, and everything from the residents of Packard 268 and 243. A special shout-out goes to William Wu, who has been my officemate the longest; I'm glad our numerous research discussions eventually led to a paper co-authored together. I'd also like to acknowledge Aditya Siripuram, Arezou Keshavarz, Argyris Zymnis, Borja Peleato, Brendan O'Donoghue, Ekine Akuiyibo, Eric Chu, Fernando Gómez Pancorbo, Jacob Mattingley, Kwangmoo Koh, Matt Kraning, and Yang Wang; long days in Packard wouldn't have been the same without your company and commiseration.

Outside the office, I have been fortunate to have made many good friends who

have kept me sane these many years. I'm especially grateful for the roommates: Gareth Yeo, Chand John, Ali Vafai, and Glen Gibb. I'm still not going to say which one of you has been my favorite roommate, but it has been a pleasure living with each of you; I will never forget the many food runs and other random activities that we've shared over the years. Even more importantly, I've appreciated the many moments when you've provided sympathetic ears as the inevitable bumps along the road have occurred. I've also been lucky to have met many other good friends while at Stanford, including—but not exclusively—Adam Lee, Alan Asbeck, Andrew Poon, Andrew Reid, Chi Cao Minh, Dawson Wong, Ed Choi, Elaine Kim, Emmalynne Hu Roy, Genny Pang, Jenny Chen, Johnny Pan, Kaushik Roy, Kevin Bin Wu, Laura Nowell, Matt DeLio, Nelly Lau, and Vincent Chen. Thanks for sticking by me and encouraging me to finish this whole thing.[1] Each of you has made a positive impact on my experience in graduate school, and my time at Stanford would have been much less fun without you—dare I say, not fun at all without you.

Of course, my earliest supporters were my parents, Joseph and Helen, and my sister, Selene—from whom I have benefited from their unwavering support and unconditional love. It has never mattered to them whether I completed my degree or not, as long as I was happy with what I was doing. To my sister, I must say that although we've had many moments of commiseration over the past several years as we both went through our respective programs (for which I owe you many hours of your time that you're not getting back!), it is your constant encouragement and good advice that I've treasured the most. I'm glad that we were both able to finish what we started. I may still be behind, but I wouldn't have it any other way. And to my parents especially: I hope I have made you proud.

Although this dissertation is the destination for the Ph.D. journey, the process has been more about the journey than the destination itself. To those I have mentioned and the countless others who have supported and helped me along the way, I offer a heartfelt thank you.

[1] Dumpling threat notwithstanding.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, as computing and communication systems have become larger and more interconnected, systems design must increasingly consider scale as an important consideration. Existing approaches that worked for small systems may fail in dramatic fashion once more users are introduced into the system. Such scalability issues can be caused by various factors that do not scale well when, for example, system size is increased or resource contention is amplified. In many cases, it becomes necessary to rethink the design of such systems in order to account for these issues. Thus in this dissertation, we will reconsider problems in storage, networking, and communications in light of potential scalability issues—and propose new ways of tackling these problems that are not as prone to the scalability problems that arise.

In the problems that are considered here, the main issue affecting scalability is that the existing solutions are combinatorial in nature—and thus such solutions can become intractable if one does not have a good understanding of the combinatorial aspects of these problems. The main theme of this dissertation is that by taking a critical look at the combinatorial aspects of these problems, it is possible to come up with solutions that perform well even as the problem scenarios significantly increase in size.

In the remainder of the introduction, we will survey some of the combinatorial challenges for the problems we solve in this dissertation, and allude to the solution

methods that encompass the work herein. Such problems span an array of applications, namely, distributed storage, network coding, and communications, but the underlying message is the same—that careful consideration of the combinatorics is essential for arriving at solutions that will scale gracefully as the problem spaces increase.

1.1 Distributed Storage

With the current explosive growth of the internet and other media, the amount of data being created and collected in a given year has become quite staggering—leading to the need for much larger storage systems to store and serve this data. Because storage technologies (e.g., hard drive sizes) have not kept up with storage system sizes, it is no longer uncommon to encounter data centers consisting of thousands or tens of thousands of storage nodes—and this number is projected to continue to increase [40].

On the other hand, existing designs for reliability were not designed to handle storage systems of such large magnitudes. For instance, many existing storage systems use some variant of RAID (Redundant Arrays of Inexpensive Disks) [66] in order to guarantee reliability—which unfortunately does not scale well as the number of storage nodes in the system increases. As an example, the parity striping utilized in RAID levels 5 and 6 occurs across all storage nodes—meaning that all nodes need to be accessed in order to rebuild any parity stripe. Clearly, this becomes an untenable situation as storage systems grow ever larger, as storage node reconstruction methods become bogged down in the read phase. Moreover, as storage systems increase in size, the possibility for multiple simultaneous node failures increases—again emphasizing the need for efficient node reconstruction in the storage system.[1]

These downsides of RAID have not gone unnoticed, and so ad hoc methods such

[1]Compounded upon the fact that large storage systems are more prone to failure is that most distributed storage are now built using commodity hardware and drives, for cost minimization reasons, which means that node failures in the system are yet more likely. In the Hadoop Distributed File System (HDFS) architecture guide [3], for example, it is noted that such systems have to be designed explicitly for scenarios where "Hardware failure is the norm rather than the exception."

as nested RAID (e.g., with secondary RAID levels [78]) have been introduced for scaling. Other ideas, including parity declustering [42], have also been introduced as bolt-ons to RAID—although the performance of these fixes under storage system scaling is poorly understood. Some other approaches, such as distributing data chunks according to distributed hash tables [47] or random distribution schemes, are also deployed in distributed storage systems, but these approaches are in fact not optimal for efficient node repair. A more careful look at how to design storage systems for scalability and reliability is required, which we will discuss here.

In Chapter 2 of this dissertation, we look at storage system reliability from the replication standpoint, and present a method for distributing data chunks and their replicas that enables fast reconstruction—and yet the designs are scalable to very large storage systems.

Fast reconstruction is achieved because these designs guarantee that storage node reconstruction can always be performed via reading multiple nodes in parallel. As an example, consider a storage system storing 12 total data chunks, where each chunk has 3 replicas, and data chunks and replicas are distributed to the storage nodes according to the scheme in Figure 1.1. Then we see that for any node that fails, the replicas of the 4 chunks on that node can always be obtained by reading a single data chunk from each of 4 separate nodes that hold the replicas. In fact, maximal parallelism can be achieved, as these designs guarantee that exactly one data chunk needs to be read from each of the nodes containing the failed node's replicas.

b_0	0	1	2	3	b_3	1	4	7	10	b_6	1	6	8	11
b_1	0	4	5	6	b_4	2	5	8	10	b_7	3	5	7	11
b_2	0	7	8	9	b_5	3	6	9	10	b_8	2	4	9	11

Figure 1.1: Distribution of data chunks to storage nodes, where any node's replicas are always on distinct other nodes. This is the smallest such design where each node stores 4 data chunks and each chunk has 3 replicas; it requires 9 storage nodes to store 12 total data chunks.

Using ideas from graph theory [10] and finite geometries [15], we show in this dissertation that it is possible to scale up these designs to more realistic parameters than in Figure 1.1—and in fact, to scenarios that comprise the many thousands of storage nodes in a storage system alluded to earlier. These designs are able to handle storage systems where chunk replication has some small number of replicas (usually around three copies), but storage nodes individually store thousands of data chunks. These schemes will also have further scalability properties that we discuss later.

A portion of this work appears in the following publication[2]:

> J. C. Koo and J. T. Gill, "Scalable constructions of fractional repetition codes in distributed storage systems," in *Proceedings of the 49th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, September 28–30, 2011.

1.2 Network Coding

In Chapter 3 of this dissertation, we consider a scalability problem related to networking. Traditional networking has considered the use of routing in order to transmit data across the network—and network data rates have correspondingly been analyzed using this paradigm. However, as data traffic on networks has significantly increased over the years, the most common methods for increasing data transmission rates across the network have been by adding more links and routes to the network—i.e., by expanding the network—or by routing more intelligently. These strategies are potentially quite costly and are often not an efficient use of resources. It has recently been shown, however, that data transmission rates in some existing networks can be significantly increased by using *network coding* within the network [1]. For these network scenarios, routing is not the most efficient strategy—that is, routing is unable to achieve the full data rates in the network—but by using network coding, the maximum possible data rates can be achieved. Thus for these network scenarios,

[2]

network coding is provably optimal for achieving network capacity [1]. As a simple example, consider the *butterfly network* shown in Figure 1.2.

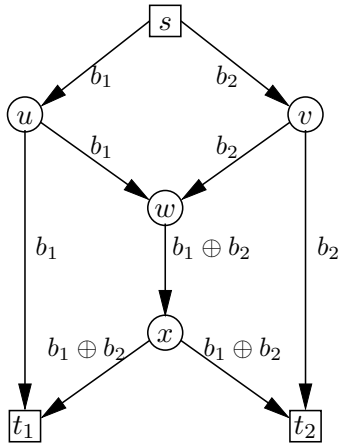


Figure 1.2: The butterfly network (adapted from [1]) shows how network coding increases possible data rates. Source s is transmitting data streams b_1 and b_2 to both sinks t_1 and t_2. The network code has node w sending $b_1 \oplus b_2$ on its outgoing link. As a result, s can transmit both streams to both sinks in one time period. (For example, sink t_1 receives stream b_1, and can recover stream b_2 by taking $b_1 \oplus (b_1 \oplus b_2) = b_2$.) With routing, only one stream at a time can traverse the $w \to x$ link, so it takes two time periods to transmit the requisite data to both sinks.

It is clear that in the butterfly network, the $w \to x$ link is a bottleneck with routing only, because the link can only transmit one data stream at a time; in order for both sinks t_1 and t_2 to receive both streams b_1 and b_2, two time periods are required.[3] However, with network coding, the coding opportunity allows for the bottleneck link to participate in transmitting both streams simultaneously—albeit in a coded fashion. This enables both sinks to receive both streams within a single time period.

In a more general multicast scenario of transmitting the same data from a single source across the network to multiple destinations, it has has been shown that network coding is provably optimal for achieving the maximum possible data rates in the network [1]—potentially enabling huge gains in data rates for large networks [64]. However, the full utility of network coding for general networks is still not known.

[3]Using routing, if the network is used continuously, it may be possible to use as few as an average of 4/3 time periods to transmit two data streams simultaneously to both sinks, but that is the minimum time possible under routing.

We consider the problem of transmitting data across a network from a single source to multiple destinations, where each destination may request data at a different rate—the so-called non-uniform demand problem [16]. The problem of constructing network codes in these scenarios has been shown to be NP-hard; thus it remains difficult to construct such network codes as network sizes get larger, as such algorithms would likely require time that is superpolynomial in the network size in order to find the network code (assuming that P $\neq$ NP [37]). In this dissertation, we provide an algorithm for constructing network codes in the non-uniform demand scenario that is based on the problem of graph coloring [37]—which is also NP-hard. However, using our algorithm and a result from graph theory known as the strong perfect graph theorem [20], we are able to show that there exist non-uniform demand network coding problems where network codes may be efficiently constructed in polynomial time. Thus we show how the combinatorial problem of constructing network codes may be simplified in these particular networks, enabling the benefits of network coding to be applied to these potentially large networks.

This work appears in the following publication[4]:

> J. C. Koo and J. T. Gill, "Low-complexity non-uniform demand multicast network coding problems," in *Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, September 30 – October 2, 2009, pp. 228–235.

1.3 Communication over Interference Channels

In Chapter 4 of this dissertation, we look at the problem of communicating across a wireless channel where each communication link may be prone to multiple interferers. In such interference channels, significant challenges arise in providing high data transmission rates to every wireless device, as each device's transmission is essentially interference to all other devices. As the number of wireless devices sharing the same

[4]Copyright © 2009 IEEE. Reprinted with permission.

channel increases, the problem of mitigating interference in the channel is exacerbated due to the increase in the number of potential interferers.

Even in scenarios where each transmitter is communicating only with a distinct receiver, as in Figure 1.3, effective strategies for simultaneous data transmission are poorly understood. For many strategies, there is an inherent conflict between high data rates for some transmitter-receiver pairs and low interference at other transmitter-receiver pairs. In these interference channels, techniques that treat interference as noise do not perform well [33]. Furthermore, orthogonalization techniques such as time or frequency division multiplexing address the interference issue, but are unable to achieve the maximum data rates in the channel [33, 14].

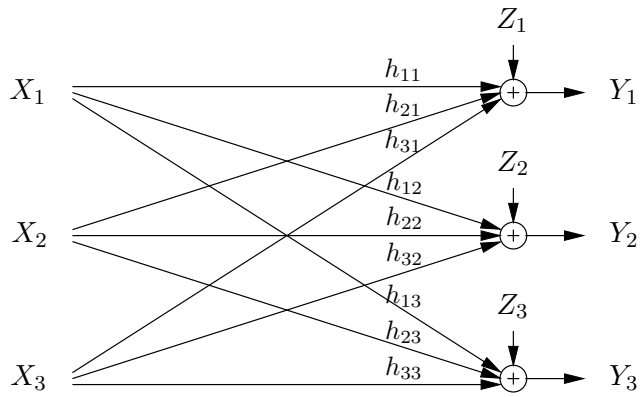


Figure 1.3: Interference channel with distinct transmitter-receiver pairs. Transmitter X_k wants to send only to receiver Y_k, for $k = 1, 2, 3$. Desired signals have channel gain of h_{kk} from transmitter X_k to receiver Y_k, whereas interference terms have gains of h_{kl}, $k \neq l$, from transmitter X_k to receiver Y_l. Z_k denotes additive noise.

However, a clever technique known as interference alignment [56, 12] allows the channel to be more fully utilized, in effect granting half the interference-free rate to each transmitter-receiver pair—no matter the total number of transmitters and receivers using the channel—even while they are simultaneously communicating. Moreover, it has been shown that such data rates are sum capacity optimal when the signal-to-noise ratios of the direct channels are high [56, 12]. Another interference alignment approach, known as ergodic interference alignment [61], has also been shown to be able to achieve the same high data rates. Unfortunately, the ergodic interference

alignment scheme suffers from huge average delay, especially as the number of devices sharing the same channel scales much larger, since the scheme relies on the occurrence of very specific circumstances among *all* the transmitter-receiver pairs. In this dissertation, we propose a method for improving the delay in the ergodic interference alignment scheme, by considering additional ways channel realizations may be aligned. The resulting delay (and per-user data rate) of our method is analyzed via Markov chain analysis, by considering the combinatorics involved in aligning such channel realizations.

This work appears in the following publication[5]:

> J. C. Koo, W. Wu, and J. T. Gill, "Delay-rate tradeoff for ergodic interference alignment in the Gaussian case," in *Proceedings of the 48th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, September 29 – October 1, 2010, pp. 1069–1075.

J. C. Koo was responsible for formulating the idea of larger alignment sets for interference alignment and devising the first-to-complete alignment scheme—and for performing the Markov chain analysis of rate and delay. W. Wu was responsible for making the connections between the first-to-complete alignment analysis, and the Markov chain and coupon-collecting problems—and for additional Markov chain analysis.

[5]

Chapter 2

Efficient Repair and Scalable Construction for Distributed Storage Systems

2.1 Introduction

Recent trends in distributed storage systems have been toward the use of commodity hardware as storage nodes, and thus the systems consist of storage nodes that may be individually unreliable. Such systems can still be feasible for large-scale storage as long as there is reliability of the entire system—for example, via the incorporation of data redundancy. Recent research in distributed storage systems has focused on using techniques from error-control coding theory to increase storage efficiency, without sacrificing system reliability and node repairability [27].

In this work, we look at the problem of storage systems where failed storage nodes must be quickly replaced by replacement nodes. To achieve short downtimes, we consider techniques where the repair of a particular node (i.e., by obtaining replacement data) is via contacting multiple non-failed nodes in parallel—where each contacted node contributes only a small portion of the replacement data. Such replacement strategies have been studied in the context of both *functional repair* [27]—where

replacement nodes are not copies of the failed node, but instead serve functionally for overall data recovery—and *exact repair*—where replacement nodes must be exact copies of the failed node.

We build upon the work of El Rouayheb and Ramchandran [31], who propose a storage system allowing for exact repair. Using the idea of Steiner systems [15, 21], the authors design distributed storage systems with the desired redundancy and repairability properties—where even though each storage node is responsible for storing multiple data chunks, replacement of any failed node is always possible by obtaining only a single data chunk from each of several non-failed nodes. In systems where multiple nodes can be read in parallel, then such a scheme ensures high availability, even in the presence of node failures. Moreover, since the scheme described in [31] stores data in an uncoded manner (where chunks are stored in their original form), such storage systems may additionally be used for cloud computing services. That is, if each storage node is also a processing node, then chunks can be processed directly at the storage node without needing to decode data at a data aggregator—which would usually require accessing data from multiple storage nodes—before performing any data processing operations.

The storage system of El Rouayheb and Ramchandran [31] involves the use of Steiner systems, in order to achieve the desired redundancy and repairability properties. In short, a Steiner system $S(t, k, v)$ specifies a distribution of v elements into blocks of size k such that the maximum intersection between any two blocks is $t - 1$ elements (so if $t = 2$, then no two blocks can share any pairs of elements). For example, consider the following Steiner system and resulting distribution of chunks to storage nodes:

Example 2.1. *Consider a distributed storage system to store a total of* 9 *data chunks, where each data chunk is stored within storage nodes that can hold* 3 *data chunks each. Then it is possible to distribute the chunks across* 12 *nodes, where every data chunk has exactly* 4 *replicas and any two distinct nodes share at most only one overlapping chunk. Such a distribution of chunks into nodes is shown in Figure 2.1. This distribution of data chunks coincides with a Steiner system of the form* $S(2, 3, 9)$.

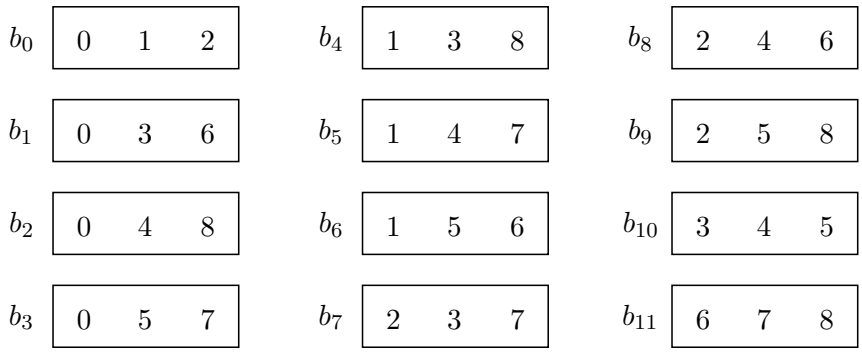


Figure 2.1: Example of Steiner system $S(2, 3, 9)$. This also shows the distribution of the 9 data chunks $\{0, 1, 2, \ldots, 8\}$ to the storage nodes $b_0, b_1, \ldots, b_{11}$, where each node is of size 3 and there are no repeats of chunk pairs among the nodes. Each data chunk has 4 replicas in the system. This storage system is the same as [31, Figure 6(a)].

El Rouayheb and Ramchandran [31] also discuss using Steiner systems to construct what they call *transpose codes*, where the Steiner parameter k now represents the number of replicas of each chunk and the parameter v represents the total number of nodes. Using transpose codes and knowledge of an underlying Steiner system, distributed storage systems with the desired replication degree and total number of storage nodes can be designed and constructed[1]; this contrasts with the original Steiner system codes, which specify node size and total number of chunks. Since the transpose design can also be specified by the resulting node size and total number of chunks, we use $S^T(t, l, u)$ to denote a system designed using a transpose code, and where l is the corresponding node size and u is the corresponding total number of data chunks. These transpose designs are related to the original Steiner system $S(t, k, v)$ that resulted in the transpose codes; the relationship between k, v, l, and u will be discussed later.

By constructing the distributed storage system from the Steiner system $S(t, k, v)$, it is possible to construct storage systems by considering k as the node size and v as the total number of data chunks. Separately, using transpose codes, k could mean the number of replicas of each chunk and v the total number of storage nodes. However, for practical storage system constructions, it is more useful to specify the node size

[1]In fact, the distributed storage system of Figure 1.1 was designed from the transpose code of the Steiner system of Figure 2.1.

and the number of replicas for each chunk—and then determine the other parameters of the system from these requirements. In this work, we shall take this approach of parameterizing systems using node size and replication degree.

We base our work on the distributed storage system constructions of El Rouayheb and Ramchandran [31]—specifically the use of uncoded storage at nodes, and where any particular pair of data chunks may not occur at more than one storage node. In most practical distributed storage systems, it is often desirable for the number of data chunks per node[2] to be much greater than the replication degree of each chunk. For example, the Google File System (GFS) [38] and the Hadoop Distributed File System (HDFS) [3], which both store data in chunks of as small as 64 MB each, have replication degrees on the order of three replicas but may store thousands of chunks on each storage node. Thus in this work, we devise a graph-based construction of Steiner systems where the replication degree and node size are significantly asymmetric—specifically, where replication degree is much smaller than node size. We will also describe how our constructions allow for easy scalability of the system, where a storage system may be increased in size without disturbing the previous smaller system. Since our methods for constructing larger Steiner systems are systematic, the algorithms we provide lead to ready scalability and implementability. Moreover, our constructions provide an advantage over other Steiner system deployments (or even deployments where data chunks are randomly distributed), which would usually require storing potentially large lookup tables; this is because our systems can be generated quickly, requiring the knowledge of a single generator.[3]

An explicit listing of the families of Steiner systems that our methods construct is given in Table 2.1. From the Steiner system, we can directly construct storage systems where each data chunk has l replicas and each storage node stores k data chunks; in this case, there are a total of u storage nodes and v distinct data chunks. More practically, using transpose codes applied to the constructions given here, we can construct storage systems where each data chunk has k replicas and each storage

[2]In the rest of this work, we refer to the number of chunks per node as the *node size*.

[3]That is, the generator is one of the primitive elements of the particular Galois field. We discuss this requirement in more detail later, and also in Appendix A.

node stores l data chunks; in this case, there are a total of v storage nodes and u distinct data chunks.

Vertex degrees		Number of vertices		Storage design		Construction	
k	l	v	u	Steiner system	Transpose code	Section	Algorithm
$q+1$	$q+1$	q^2+q+1	q^2+q+1	$S(2, q+1, q^2+q+1)$	$S^T(2, q+1, q^2+q+1)$	Sec. 2.3	Alg. 2.2
q	$q+1$	q^2	q^2+q	$S(2, q, q^2)$	$S^T(2, q+1, q(q+1))$	Sec. 2.4	Alg. 2.3
$q+1$	$p_n(q)$	$p_{n+1}(q)$	$\frac{p_{n+1}(q)p_n(q)}{q+1}$	$S(2, q+1, p_{n+1}(q))$	$S^T(2, p_n(q), \frac{p_{n+1}(q)p_n(q)}{q+1})$	Sec. 2.5.1	Alg. 2.4
q	$p_n(q)$	q^{n+1}	$q^n p_n(q)$	$S(2, q, q^{n+1})$	$S^T(2, p_n(q), q^n p_n(q))$	Sec. 2.5.2	Alg. 2.5

Table 2.1: Summary of constructible designs. Here, q is any prime number or any power of a prime number. For brevity, we denote $p_n(q) = q^n + q^{n-1} + q^{n-2} + \cdots + q^2 + q + 1 = \frac{q^{n+1}-1}{q-1}$. Whenever a design is parameterized by n, then the design is valid for any $n = 1, 2, 3, \ldots$. The columns titled *Vertex degrees* and *Number of vertices* refer to the parameters of the bipartite graph associated with the Steiner system, and is discussed in Section 2.2.2. Under the *Storage design* column, the constructible storage systems are given as both the original Steiner system, as well as the designs corresponding to the transpose code. Under *Construction*, the *Section* column refers to the particular section where the design is discussed, and the *Algorithm* column refers to the specific algorithm that constructs the desired design.

Our constructions are based on relating Steiner system problems with the problem of shortest cycles on bipartite graphs, and use concepts related to Latin squares. More specifically, our systems arise from the construction of *cage graphs* [88], which are graphs with the minimum number of vertices for a given allowable shortest cycle length and other specified conditions on the vertex degrees. Because we are constructing cage graphs, we further know that for a given desired node size and replication degree, our constructed systems are the *smallest* possible (in terms of total number of storage nodes and total number of data chunks stored). This is useful for the practical application of such constructions, as it immediately translates into least hardware cost for the desired system requirements.

In this work, we will show an additional advantage of designing storage systems via the construction of cage graphs—that is, that the design of storage systems using this approach lends itself to systems that are easily expandable. For a system constructed according to the methods in this work, we provide a method for increasing the size of the storage system so that existing data chunks do not need to be moved—and still preserve the property that no pairs of chunks occur in more than one storage node.

Furthermore, the bipartite graph interpretation allows for easy visualization of

the symmetries inherent in the Steiner system, and also presents a different method for arriving at the transpose codes of [31].

2.1.1 Related Work

The problem of constructing distributed storage systems with efficient (and reliable) repair is discussed in [28] and [27]. Using ideas from network coding [1, 89], the authors of these works propose a scheme for storing data such that linear combinations of the original data chunks are stored at each node. Data recovery of the entire data set is achieved by collecting different linear combinations of data chunks, from a sufficient number of storage nodes, so that the linear combinations satisfy a full rank property. In this scheme, the node repair method is termed *functional repair*: when a storage node fails, its replacement node does not store data identical to that of the failed node; the replacement node merely has to be able to participate in data recovery of the entire data set. Furthermore, [28] and [27] define the idea of a storage-bandwidth tradeoff for distributed storage systems, and discuss ways to implement either minimum storage or minimum bandwidth systems.

Although the work of Dimakis et al. [27] shows how to achieve the storage-bandwidth tradeoff, their scheme suffers from the shortcoming that repair is only functional. An important problem is that of *exact repair*, where a failed node must be replaced by an exact copy of the failed node. The storage-bandwidth tradeoff under the scenario of exact repair is not yet fully understood. However, building upon the network coding–based distributed storage system constructions of [27], Rashmi et al. [70, 71] give a scheme for achieving the minimum bandwidth operating point under exact repair, finding one point on the storage-bandwidth tradeoff curve.

El Rouayheb and Ramchandran [31] introduce a related distributed storage scheme, termed *fractional repetition codes*, which can perform exact repair for the minimum bandwidth regime; they use Steiner systems (for example, see Example 2.1 from above). The authors of [31] show that the schemes of Rashmi et al. [70, 71] are a special case of their method, and further derive information theoretic bounds on the storage capacity of such systems with the given repair requirements. In [31], node

repair requires a replacement node to contact specific subsets of nodes (say, a set of l nodes), rather than arbitrary subsets of l nodes; thus repair is not as flexible as the network coding–based scheme of Dimakis et al. [27]—where repair is possible even if the remaining nonfailed nodes are randomly selected and accessed for data chunks. Although the repair model of [31] is table-based (instead of random access as in [27]), their scheme has the favorable characteristics of exact repair and the uncoded storage of data chunks. Randomized constructions of such schemes are investigated in [67].

Uncoded storage and repair (where data chunks—instead of linear combinations of data chunks—are stored) has numerous advantages for distributed storage systems. First of all, for a storage system storing a total of u data chunks, recovery of the original data is immediate once an entire set of u distinct chunks is retrieved. There is no need for inverting linear combinations in order to perform data recovery, as would occur in network coding–based schemes such as in [27]. Moreover, uncoded data at nodes allows for distributed computing (e.g., for use in cloud computing applications), by spreading out computation to the node(s) that contain the data to be processed; this is possible since the original data chunks are available directly at the nodes. For example, Upfal and Widgerson [82] consider a method for parallel computation by randomly distributing data chunks among multiple memory devices. Although [82] considers distributing chunks to multiple nodes and uses a similar bipartite graph interpretation for the storage system as in our work, their work investigates the random distribution of data chunks among the nodes—and thus all performance results are asymptotic. In contrast, our designs are deterministic, and we can also guarantee the smallest possible size for our storage system. Furthermore, if the uncoded data chunks are distributed among the nodes according to Steiner systems, as proposed by [31] and elaborated upon in this work, then load-balancing of computations is easier to achieve, as for a set of l data chunks stored at a node it will always be possible to find some set of l other nodes such that each node processes a single distinct data chunk out of the data set. Although we do not go into more detail about the benefits here, such a property may be extremely useful in massive distributed computing frameworks such as MapReduce [25] and Hadoop [2].

In the field of combinatorial design theory [5, 6, 29, 53], Steiner systems are an example of balanced incomplete block design (BIBD). In this work we consider Steiner systems $S(t, k, v)$ only for $t = 2$, i.e., where pairs of elements may not occur in more than one block. Some Steiner systems are known for when $k = 2, 3, 4, 5$, and with the specific block designs found in tables such as those in [21]. It is also known that for a given k, there exists a Steiner system when v is sufficiently large and certain integrality feasibility conditions are satisfied (see [85, 86, 87] or [31] for the exact statement and conditions)—although in such cases, "sufficiently large v" is difficult to control. In this work, we consider Steiner systems similar to those from finite projective geometries [15]. Specifically, designs in which the replication degree is $q+1$ and with each storage node storing up to $q^n+q^{n-1}+\cdots+q+1$ data chunks can also be found from the projective geometry $\mathrm{PG}(n+1, q)$, where the data chunks are the lines and the storage nodes are the points of the corresponding space. However, in this work we show that via our recursive graph construction method, it is possible to initially deploy small storage systems without needing to know *a priori* the future maximum extent of the storage system—while still being able to preserve the Steiner property in subsequent expanded systems. This alternative approach for constructing projective geometries has tremendous benefits for practical storage system designs, as otherwise the connection between system design and the construction and extension of such geometries is not immediately obvious. Furthermore, our graph-based construction is simple to implement, and the resulting designs are uniquely determined given the base set of mutually orthogonal Latin squares used in the construction.

We will also construct Steiner systems arising from affine geometries [15]. These are designs in which the replication degree is q and with each storage node storing up to $q^n + q^{n-1} + \cdots + q + 1$ data chunks, and can be found from the affine geometry $\mathrm{AG}(n+1, q)$. These constructions turn out to be just as expandable as the projective geometry–based designs.

We now discuss the Steiner system approach of our work (and of [31]) in relation to other possible distributed storage system designs.

Clearly, a storage system where nodes are replicated in full (e.g., mirrored disks

such as in RAID 1 [66]) will always be able to perform exact repair; however, such a system does not allow for parallelism during repair. Whenever a storage node fails, a node that is contacted for repair will have to serially copy its entire contents to the replacement node, potentially incurring a large downtime of both the contacted node and the replacement node. Because we consider scenarios where the size of a storage node (in terms of number of chunks stored at the node) is much larger than the replication degree, even if all of the replicas of the failed node are contacted simultaneously to contribute partial recoveries, each contacted node will still need to contribute numerous data chunks toward repair. Using Steiner systems, we instead have a scenario where each contacted node needs to contribute only a single chunk toward recovery of the failed node. Moreover, in systems with available bandwidth, the receipt of data for recovery can be quite rapid if the data can be received in parallel.

Furthermore, under worst-case scenarios, multiple node failures are potentially more disastrous in storage systems with full node replication than systems designed using Steiner system–based block designs. For a replication degree of k, under full node replication if all k replicas of a storage node were to fail, then the entire data set of a storage node would be lost—which could be a significant number of data chunks. However, under the fractional replication scheme of [31], any loss of k nodes would result in the loss of at most one data chunk—namely the loss of the possible single chunk that is shared among all k failed nodes. Thus, at least in comparison to the naïve strategy of full node replication, fractional repetition codes have an advantage in combating multiple node failures. For any other strategy where chunk replicas may occur together in more than one node, using Steiner systems will still have an advantage over these chunk distribution strategies, since with Steiner systems no more than one chunk will be shared in common among any k nodes that fail.

Some other distributed storage implementations that consider repair strategies for high availability and reliable storage include OceanStore [73], TotalRecall [7], and DHash++ [24]. Many of these systems utilize distributed hash tables (DHTs) [47],

in order to guarantee rapid node repair and replacement. The use of DHTs for distributed storage systems is advantageous because DHTs allow data to be stored in such a way that small amounts of new data and new nodes can be added easily to the system, via use of a special hashing method (see, for example, the early implementations in CAN [72], Chord [77], Pastry [75], and Tapestry [90]—which are surveyed in [4]). Our proposed storage method has a similar application to a distributed storage system utilizing a DHT with multiple hashes; the storage scheme we propose is a way of generating multiple hashes so as to satisfy the property that no data chunk pairs occur in more than one node. Although our equivalent hash is not a consistent hash [47], our method has a different scalability benefit—as discussed in Section 2.5.3—which is the ability to easily introduce large numbers of new nodes without shifting or moving any data already in the system, while still preserving the appropriate separation of chunk pairs of old and/or new chunks.

The use of block designs and Steiner systems for distributed storage systems has also been briefly considered with respect to quorum systems. For example, Maekawa in [57] describes a method for obtaining mutual exclusion by using projective plane–based designs to design coteries [36] when the number of sets and the set size are approximately the same. Although the larger Steiner systems that we consider in this work may not be able to establish coteries, it is clear, however, that sets that are constructed as Steiner systems cannot be dominated. This is because the no-repeating-pairs property of Steiner systems guarantees that no set can be a superset of any group of other sets in the system. However, Steiner systems—and specifically our constructions of Steiner systems that are highly unbalanced between replication degree and block size—may be able to find additional application in the design of quorum systems, although we do not pursue this avenue further in this work. We do note that several distributed storage systems implementing voting mechanisms for data guarantees—such as OceanStore [73], RAMBO [39], and FAB [76]—can potentially benefit from any application of our Steiner system constructions to the design of mutual exclusion sets.

In addition to [31], the use of balanced incomplete block designs for guaranteeing

load-balanced disk repair in distributed storage systems was also introduced by Muntz and Lui [60] and then further investigated by Holland et al. [42], for application to RAID-based disk arrays. Notably, in [42], the authors discuss how block designs may be used to lay out parity stripes in declustered parity RAID disk arrays. The block designs that we construct in our work may be helpful for distributing parity blocks in this scenario, in order to build disk arrays with good repair properties. Other methods for using block design in storage applications may also be found in the survey by Colbourn and van Oorschot [22].

We briefly mention that some block designs may also be applicable to the design of error-correcting codes, particularly in the construction of geometrical codes [55, Sections 2.5 and 13.8]. Graphs without short cycles have been considered in the context of Tanner graphs [80] for error-correcting code design, as applied to situations where all the variable node degrees are the same and where all the check node degrees are the same. Furthermore, finite geometries in particular have been considered in the context of LDPC codes [50]. Our work is applicable only to algorithms that allow at least cycles of length 6, due to our girth restriction on cycle lengths—and thus may find only limited applicability. It is worth noting that Tanner's original paper does give a recursive construction for generating bipartite graphs of particular girth, although with the intent of constructing graphs with much larger girth (than 6) and by using only projective plane methods. Block designs and their related bipartite graphs are also considered in the context of code design for magnetic recording applications in [83].

2.1.2 Outline of Chapter

In the next section, we provide some necessary background for our work. In each of Sections 2.3, 2.4, and 2.5, we discuss algorithms for each of our constructions, as well as provide examples for some of the smaller cases. The specific systems that are discussed in each section are shown in Table 2.1. Sections 2.3 and 2.4 serve to illustrate how our constructions work, and to provide a base upon which the larger constructions of Section 2.5 are built. Then in Section 2.5 we give the main contribution of

our work, which is the design of scalable storage systems that can be expanded readily. Section 2.6 shows some results from simulations of a distributed storage system designed using our constructions. Finally, Section 2.7 presents conclusions.

2.2 Preliminaries

In this section, we provide background on the various concepts that will be used in the rest of this chapter. Although some of the topics may seem disparate, we hope to provide in this section an understanding of the relationships between the various topics. First we introduce some notation for the graphs that we consider. Then we discuss existing work on fractional repetition codes [31] and their relations to Steiner systems and bipartite graphs. Since our techniques will consider the construction of graphs that are cages, we will additionally provide some useful properties of cage graphs. Following that, we give a brief discussion of mutually orthogonal Latin squares, since this concept will aid in the construction of cage graphs.

2.2.1 Notation

When describing parameters for constructible graphs we will use $p_n(q)$ to mean

$$p_n(q) = q^n + q^{n-1} + q^{n-2} + \cdots + q^2 + q + 1 = \frac{q^{n+1} - 1}{q - 1}, \tag{2.1}$$

where n is any positive integer. In the rest of this chapter, q will always denote either a prime number or a power of a prime.

In this work, we consider simple undirected bipartite graphs, where the two vertex sets are denoted by X and Y, respectively. The edge set is denoted by E, so a particular graph is defined as $G = (X, Y, E)$. We use $|\cdot|$ to denote cardinality, so for example, $|X|$ means the number of vertices in the vertex set X. Also, the degree of a vertex is the number of edges incident to the vertex, and we let $\deg(x)$ represent the degree of vertex x. Since we consider only graphs where all of the vertices in a particular vertex set have the same degree, we denote by $\deg(X)$ the degree of any

vertex in X; that is, $\deg(X) = \deg(x)$ for every $x \in X$. These definitions clearly also hold for the vertex set Y.

We use the symbol $\sim$ to denote vertices that are connected by an edge. Thus for vertices x and y, we say that $x \sim y$ if and only if $(x, y) \in E$. For sets of vertices $\tilde{X} \subseteq X$ and $\tilde{Y} \subseteq Y$, we say that $\tilde{X} \sim \tilde{Y}$ if and only if every $x \in \tilde{X}$ and every $y \in \tilde{Y}$ satisfy $x \sim y$ (i.e., $\tilde{X} \sim \tilde{Y}$ if and only if the induced subgraph of $(\tilde{X}, \tilde{Y})$ is a complete bipartite graph). As a special case, for vertex x and subset of vertices $\tilde{Y}$, we say that $x \sim \tilde{Y}$ if and only if $x \sim y$ for all $y \in \tilde{Y}$.

2.2.2 Graph Interpretation of Steiner Systems

In [31], El Rouayheb and Ramchandran introduce the idea of fractional repetition codes, based on the construction of Steiner systems (see Example 2.1 for a simple example). Generally, a Steiner system is a collection of elements into blocks satisfying certain properties, and is denoted by $S(t, k, v)$.

Definition 2.1 ([21, Chapter II.5])**.** *A* Steiner system $S(t, k, v)$ *(where* $2 \leq t < k < v$*) is a set of* v elements*,* $\mathcal{V}$*, along with a collection* $\mathcal{B}$ *of subsets (called* blocks*) of* $\mathcal{V}$*, where each block is of size* k *and any* t*-subset of* $\mathcal{V}$ *is contained in exactly one block.*

We focus on Steiner systems where $t = 2$; therefore any particular pair of elements from $\mathcal{V}$ (i.e., any 2-subset of $\mathcal{V}$) is contained in exactly one block.[4] In fact, what we are interested in are systems where any particular pair of elements from $\mathcal{V}$ occurs in no more than one block; it turns out that our constructions necessarily result in each pair occurring in *exactly* one block.

We will reinterpret the Steiner system requirements by considering the properties of its *incidence graph* [10]. Consider a bipartite graph $G = (X, Y, E)$, where there are u vertices in X, each of degree k, and v vertices in Y, each of degree l. For convenience, these parameters are enumerated in Table 2.2. We call such a graph to be *biregular* when $k \neq l$. Clearly, $lv = uk$.

[4] In the rest of this work, whenever we use the term Steiner system, we are referring to Steiner systems with $t = 2$.

Vertex set	Vertex degrees	Number of vertices
X	$k = \deg(X)$	$u = \lvert X \rvert$
Y	$l = \deg(Y)$	$v = \lvert Y \rvert$

Table 2.2: Relationship between parameters for bipartite graph $G = (X, Y, E)$. X and Y denote the two respective vertex sets. We adhere to $l \geq k$ throughout the rest of this chapter.

Now we label the vertices of Y as the elements of $\mathcal{V}$ (i.e., $\mathcal{V} = \{y_g \mid g = 0, 1, \ldots, v-1\}$), and the vertices of X as the blocks of $\mathcal{B}$. Consider a particular vertex x_h (where $h \in \{0, 1, \ldots, u-1\}$), and define block $b_h = \{y_g \in Y \mid y_g \sim x_h\}$. That is, b_h denotes all the vertices in Y that are connected to the particular vertex x_h. Then the collection of blocks, $\mathcal{B} = \{b_h \mid h = 0, 1, \ldots, u-1\}$, satisfies the following properties:

1. Each element $y_g \in \mathcal{V}$ occurs in exactly l blocks of $\mathcal{B}$ (since y_g is connected to l vertices of X). There are exactly v such elements y_g.

2. Each block $b_h \in \mathcal{B}$ contains k elements (since x_h is connected to k vertices of Y). There are exactly u such blocks b_h.

In order to verify that a block system is a Steiner system, we need to make sure that any two blocks b_h and $b_{h'}$ $(h \neq h')$ do not share any pair of elements. In fact, this condition is equivalent to making sure that no cycles of 4 vertices exist in the bipartite graph G. It is clear that whenever two blocks b_h and $b_{h'}$ share some pair of elements y_g and $y_{g'}$, then this is equivalent to a 4-cycle in G consisting of $x_h \sim y_g \sim x_{h'} \sim y_{g'} \sim x_h$. Thus the nonexistence of such 4-cycles is equivalent to the nonexistence of shared pairs of elements between blocks.

Thus in the above description, we construct Steiner systems where l is the repetition degree, k is the block size (number of elements in each block), v is the total number of elements, and u is the total number of blocks. We have the set of elements $Y = \{y_0, y_1, \ldots, y_{v-1}\}$ (so $|Y| = v$)—each element with repetition degree l—and the collection of blocks $X = \{x_0, x_1, \ldots, x_{u-1}\}$ (so $|X| = u$)—each block with block size k.

We call the resulting distributed storage system arising from this Steiner system as the *original code*. In the rest of this chapter, we shall always assume that $l \geq k$.[5]

For visualization purposes, we show the bipartite graph interpretation of the Steiner system of Example 2.1 (also Figure 2.1) in Figure 2.2.

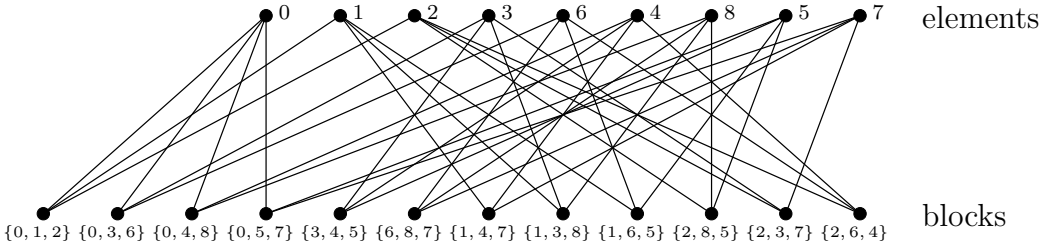


Figure 2.2: Steiner system corresponding to $k = 3$ and $l = 4$. Single numbers refer to elements, and triplets represent specific blocks. This gives the same Steiner system as in Figure 2.1.

Although in the above description, we denote Y as the elements and X as the blocks of the equivalent Steiner system, since the X and Y are interchangeable, it is clear that we could just as easily denote X as the elements and Y as the blocks of another Steiner system.[6] With this in mind, it is evident that for a particular distributed storage system constructed using a Steiner system that is equivalent to a particular bipartite graph, one can obtain the transpose codes of [31] by interchanging the roles of X and Y. Thus, our constructions of distributed storage systems (as well as those of [31]) may be specified either using known Steiner systems or using their equivalents where the roles of blocks and elements are reversed. In fact, since for practical cases we wish to construct distributed storage systems where the repetition degree is smaller than the block size, we will more often employ the *transpose code* of the Steiner system in order to construct the storage system. To stay consistent with $l \geq k$, in these cases we let k be the repetition degree and l be the block size. Under this interpretation, u is the total number of elements and v is the total number of blocks. We then have the set of elements $X = \{x_0, x_1, \ldots, x_{u-1}\}$ (so $|X| = u$)—each element with repetition degree k—and the collection of blocks $Y = \{y_0, y_1, \ldots, y_{v-1}\}$

[5] We can construct systems where $k > l$ by swapping the two vertex sets of the bipartite graph.

[6] In the language of finite geometries, interchanging the roles of elements and blocks is the same as interchanging *points* and *lines*.

(so $|Y| = v$)—each block with block size l. As mentioned previously, for a given Steiner system $S(t, k, v)$, we define the system resulting from transposing elements and blocks to be $S^T(t, l, u)$. This definition is consistent with the interpretation of l being the block size and u being the total number of elements for the transpose system.

We see from the preceding discussion that in order to construct our distributed storage systems using Steiner systems, we want to construct graphs that contain shortest cycle of a particular length. Such graphs that are specified by the shortest cycle in the graph are known as cage graphs, which we discuss next.

2.2.3 Cage Graphs

In an undirected graph $G = (V, E)$, a cycle of length d is a set of d vertices connected in a closed path, i.e., a set of vertices $\{v^{(0)}, v^{(1)}, v^{(2)}, \ldots, v^{(d-1)}\} \subseteq V$ such that the edge connections $v^{(0)} \sim v^{(1)} \sim v^{(2)} \sim \cdots \sim v^{(d-1)} \sim v^{(0)}$ exist. In the bipartite graph $G = (X, Y, E)$, such a cycle must necessarily alternate between vertices of X and vertices of Y; thus any cycles must have even length. The *girth* of a graph is defined as the length of the shortest cycle in the graph.

Then, a *d-cage* is a girth-d graph with the minimum number of vertices for a particular desired degree distribution [88, 10]. We are interested in bipartite graphs $G = (X, Y, E)$ that are biregular, where all of the vertices in X have degree k and all of the vertices in Y have degree l. The main goal of this work is to construct biregular cage graphs of girth 6 (so that no 4-cycles are present), in order to construct the smallest possible Steiner system with the desired degree distribution. Using transpose codes we can then construct systems requiring the fewest possible storage nodes (i.e., smallest v) and the least number of total distinct chunks (i.e., smallest u), while still having the desired repetition degree k and block size l for the system. Such systems will meet the lower bound of Lemma 2.1.[7]

[7]The result of Lemma 2.1 is sometimes known as a Moore-type bound [59], although we note that the bound in (2.3) is tighter than the corresponding bound in [43] for our case, when $l > k$.

Lemma 2.1. *Consider a simple biregular bipartite graph (X, Y, E) that does not have any cycle of 4 or fewer vertices. If $\deg(X) = k$ and $\deg(Y) = l$ (where $l \geq k$), then the number of vertices, $v = |Y|$ and $u = |X|$, has lower bounds*

$$v \geq 1 + l(k-1) \tag{2.2}$$

$$u \geq l + l(l-1)(k-1)/k. \tag{2.3}$$

Proof. The lower bound on v can be seen by considering an arbitrary vertex $y \in Y$. The vertex y must be connected to l distinct vertices of X; call this subset of vertices $\tilde{X} \subseteq X$. Now suppose that two vertices $\tilde{x}_1, \tilde{x}_2 \in \tilde{X}$ were both connected to some other vertex $\tilde{y} \neq y$. Then the graph would have a cycle of length 4, consisting of vertices $y \sim \tilde{x}_1 \sim \tilde{y} \sim \tilde{x}_2 \sim y$. Now, for each $\tilde{x}_j \in \tilde{X}$, let $\tilde{Y}_j = \{\tilde{y} \mid \tilde{y} \sim \tilde{x}_j,\ \tilde{y} \neq y\}$ (so $|\tilde{Y}_j| = k - 1$ for every j). Since the graph has no cycles of length 4, the sets $\tilde{Y}_j$ are disjoint; therefore $\tilde{Y} = \bigcup_{j=1}^{l} \tilde{Y}_j$ has cardinality $|\tilde{Y}| = l(k-1)$. Because $|\{y\} \cup \tilde{Y}| = 1 + l(k-1)$ (since $y \notin \tilde{Y}$), we establish the lower bound on $v = |Y|$.

Now consider the $l(k-1)$ vertices of $\tilde{Y}$. These vertices must each be connected to only one vertex of $\tilde{X}$. Otherwise, a vertex $\tilde{y} \in \tilde{Y}$ connected to both $\tilde{x}_1 \in \tilde{X}$ and $\tilde{x}_2 \in \tilde{X}$ would form the 4-cycle $\tilde{y} \sim \tilde{x}_1 \sim y \sim \tilde{x}_2 \sim \tilde{y}$ (similar to above). Therefore for any $\tilde{y} \in \tilde{Y}$, the vertex must connect to at least $l - 1$ vertices of $X \setminus \tilde{X}$. Let $\hat{X}$ consist of vertices in $X \setminus \tilde{X}$ such that all $x \in \hat{X}$ are connected to some vertex in $\tilde{Y}$. Since there are at least $l(k-1)(l-1)$ edges between $\tilde{Y}$ and $\hat{X}$, and any vertex $x \in \hat{X}$ has degree k, then $|\hat{X}| \geq l(k-1)(l-1)/k$. As $\tilde{X} \cap \hat{X} = \emptyset$, so $u = |X| \geq |\tilde{X}| + |\hat{X}| \geq l + l(l-1)(k-1)/k$. □

Any bipartite cage achieving the lower bounds of Lemma 2.1 satisfies the Steiner system property that each pair of elements occurs in *exactly* one block. We already know that every pair of elements occurs in at most one block. Since $v = 1 + l(k-1)$ and $u = l + l(l-1)(k-1)/k$ also satisfies $\binom{v}{2} = u\binom{k}{2}$,[8] we know that every pair of elements occurs in at least one block—and therefore occurs in exactly one block.

[8]The condition $\binom{v}{2} = u\binom{k}{2}$ comes from the fact that there are a total of $\binom{v}{2}$ pairs of elements, which should correspond exactly to the sets of $\binom{k}{2}$ pairs of elements in each of the u blocks.

The proof of Lemma 2.1 also gives us clues on how to construct bipartite graphs that achieve the lower bounds; these must necessarily be cage graphs. If our constructions achieve the lower bounds (2.2) and (2.3), then the resulting graphs are cage graphs. The bound of (2.2) arises by expanding the initial element to l blocks, where each block must have $k-1$ distinct (non-overlapping) elements in addition to the initial element. Thus, one possible method for constructing bipartite graphs achieving the lower bounds would be by starting with a single vertex $y \in Y$ and connecting it to l vertices of X. These vertices of X must then be connected to $k-1$ distinct other vertices of Y. The remaining vertices of X would then need to be connected to the extant vertices of Y in such a way as to preserve the nonexistence of 4-cycles. We formalize this intuition into Algorithm 2.1 as it will introduce additional terminology and also form the skeleton for all of our later graph constructions.

Algorithm 2.1 Construction of bipartite graph achieving $|Y| = 1 + l(k-1)$ and $|X| = l + l(l-1)(k-1)/k$

1: [**Layer** 0] Start with a single vertex $y_0 \in Y$. Vertex y_0 is called the *layer* 0 *vertex*.
2: [**Layer** 1] Connect y_0 to l vertices of X. Without loss of generality, call these vertices $x_0, x_1, \ldots, x_{l-1}$; these are called the *layer* 1 *vertices*.
3: [**Layer** 2] For each vertex x_j, $j = 0, 1, \ldots, l-1$, connect x_j to $k-1$ vertices of Y, so that $l(k-1)$ vertices of Y are newly connected. These vertices of Y will be labeled $y_1, y_2, \ldots, y_{l(k-1)}$, and are called the *layer* 2 *vertices*.
4: [**Layer** 3] Create $l(l-1)(k-1)/k$ vertices $x_l, x_{l+1}, \ldots, x_{l-1+l(l-1)(k-1)/k}$, which we call the *layer* 3 *vertices*. Each of the layer 3 vertices has k outgoing edges, to be connected back to the layer 2 vertices.
5: Connect the layer 2 vertices $\{y_i \mid i = 1, 2, \ldots, l(k-1)\}$ to the layer 3 vertices $\{x_{l-1+j} \mid j = 1, 2, \ldots, l(l-1)(k-1)/k\}$ so that no 4-cycles are formed (if possible).

At the conclusion of step 4 of Algorithm 2.1, we obtain the graph shown in Figure 2.3. Notice that steps 1 through 3 avoid creating 4-cycles among the vertices in layer 0 through layer 2, as these layers form a tree subgraph.

Thus, constructing the cage graph requires introducing edges appropriately between the layer 2 and layer 3 vertices so that no 4-cycles are produced by step 5. We will show how to achieve this property for certain graph parameters, by using mutually orthogonal Latin squares (see Appendix A or [26]). Specifically, in order

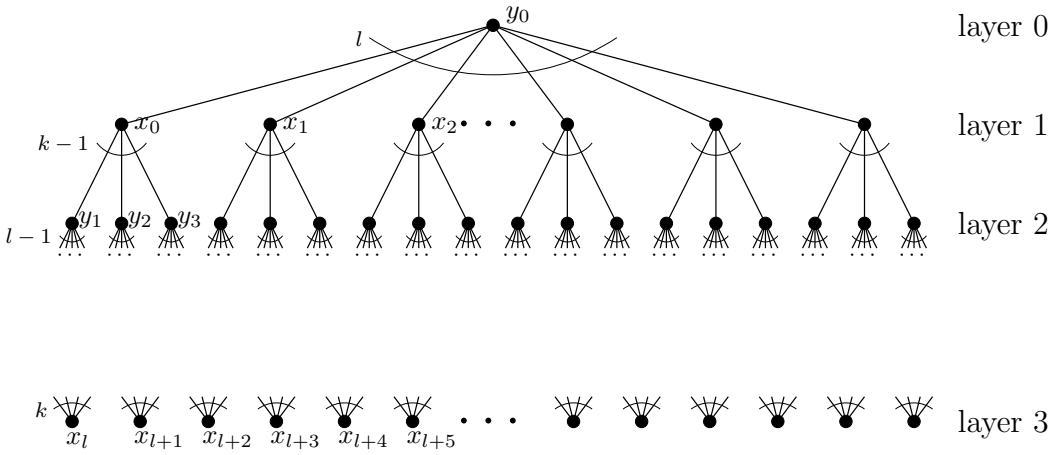


Figure 2.3: Generic construction of bipartite cage, at the conclusion of step 4 of Algorithm 2.1.

to construct the cage graphs, we will require the existence of a set of q mutually orthogonal $q \times q$ squares, $\{L^{(0)}, L^{(1)}, L^{(2)}, \ldots, L^{(q-1)}\}$, where $L^{(0)}$ is a square with every column in natural order, and $L^{(1)}, L^{(2)}, \ldots, L^{(q-1)}$ are mutually orthogonal Latin squares where each square has its zeroth column in natural order. Such a set always exists when q is a prime number or a power of a prime number. As an example, we have the following set of mutually orthogonal squares of order $q = 3$:

Example 2.2. *A set of* 3 *mutually orthogonal* 3×3 *squares consists of the squares*

$$L^{(0)} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}, \quad L^{(1)} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix}, \quad L^{(2)} = \begin{bmatrix} 0 & 2 & 1 \\ 1 & 0 & 2 \\ 2 & 1 & 0 \end{bmatrix}. \tag{2.4}$$

2.3 Regular Cage Graphs

In this section, we give a method for constructing girth-6 bipartite cage graphs where the degrees of both vertex sets are equal—and thus the graphs are regular. More specifically, the vertex degrees will satisfy $\deg(X) = \deg(Y) = q + 1$ (i.e., $k = l = q+1$), where q is any prime or power of a prime. We give an algorithm for constructing such graphs, then provide an example of the construction algorithm, followed by a

proof that the constructed graphs are indeed cage graphs of girth 6. The resulting graphs will have $|X| = |Y| = q^2 + q + 1$.

2.3.1 Construction of Regular Cage Graphs

The construction of regular bipartite cage graphs of girth 6 is inspired from the construction in Wong [88], and is given in Algorithm 2.2. Bipartiteness of the graph arises from the construction.

Algorithm 2.2 Construction of bipartite cage when $k = l = q + 1$

1: [**Layer** 0] Start with a single vertex $y_0 \in Y$.
2: [**Layer** 1] Connect y_0 to $l = q + 1$ vertices of X. Without loss of generality, call these vertices $x_0, x_1, \ldots, x_{l-1}$.
3: [**Layer** 2] For each vertex x_j, $j = 0, 1, \ldots, l-1$, connect x_j to $k - 1 = q$ vertices of Y. Let $\hat{y}_{j,m}$, $m = 0, 1, \ldots, k-2$, denote the vertices of this step that are connected to vertex x_j.
4: [**Layer** 3] Connect each vertex $\hat{y}_{0,m}$ $(m = 0, 1, \ldots, k-2)$ to $l - 1 = q$ distinct vertices of X, called $\hat{x}_{m,i}$, $i = 0, 1, \ldots, l-2$. Therefore, $\hat{x}_{m,i} \neq \hat{x}_{m',i'}$ unless $m = m'$ and $i = i'$. There will be $(k-1)(l-1) = q^2$ such vertices $\hat{x}_{m,i}$.
5: Consider a particular vertex $\hat{x}_{m,i}$, where $m \in \{0, 1, \ldots, k-2\}$ and $i \in \{0, 1, \ldots, l-2\}$. Connect $\hat{x}_{m,i}$ to vertices $\hat{y}_{j+1,L_{i,j}^{(m)}}$, where $j = 0, 1, \ldots, l-2$.

The $q^2 + q$ layer 2 vertices $\hat{y}_{j,m}$, $j = 0, 1, \ldots, q$ and $m = 0, 1, \ldots, q-1$, that are introduced in step 3 are the same as the vertices $y_1, y_2, \ldots, y_{q^2+q}$. For example, we could use the mapping $y_{jq+m+1} = \hat{y}_{j,m}$. Similarly, the q^2 layer 3 vertices $\hat{x}_{m,i}$, $m = 0, 1, \ldots, q-1$ and $i = 0, 1, \ldots, q-1$, that are introduced in step 4 are the same as the vertices $x_{q+1}, x_{q+2}, \ldots, x_{q+q^2}$, and can be mapped using $x_{q+mq+i+1} = \hat{x}_{m,i}$.

Notice that the resulting bipartite graph consists of the layer 0 and layer 2 vertices on one side of the graph, connected only to layer 1 and layer 3 vertices on the other side of the graph. It is clear that the graph constructed from Algorithm 2.2 is bipartite, where the two sets of vertices are Y and X, respectively.

We first show an example of the construction of Algorithm 2.2 with $k = 4$ and $l = 4$ (so $q = 3$), before proving that this indeed results in the desired cage graph.

2.3.2 Example of Regular Cage Graph Construction: $k = 4$, $l = 4$

Here we show the construction of Algorithm 2.2, where $q = 3$, so that the vertices of X have degree $k = 4$ and the vertices of Y also have degree $l = 4$. It turns out that the constructed bipartite graph will have number of vertices $|X| = u = 13$ and $|Y| = v = 13$.

The first three steps of the algorithm are straightforward, as they involve connecting the vertices of layers 0, 1, and 2 in a tree. The result is shown in Figure 2.4.

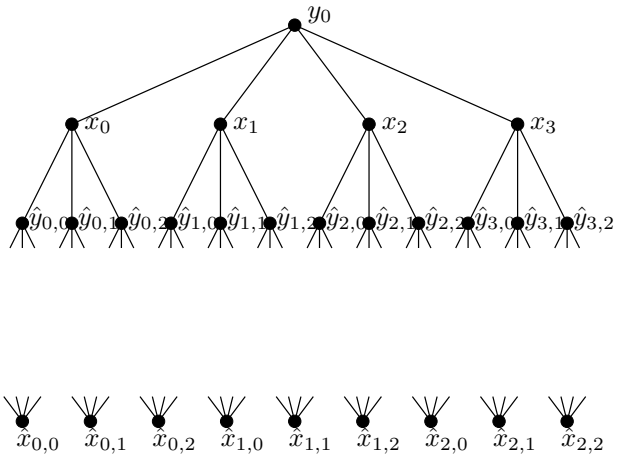


Figure 2.4: Construction of bipartite cage graph with $k = 4$ and $l = 4$, at the conclusion of step 3 of Algorithm 2.2.

Step 4 involves connecting all the vertices associated with $\hat{y}_{0,m}$ with the appropriate vertices $\hat{x}_{m,i}$. That is, for each $m \in \{0, 1, \ldots, q-1\}$, connect $\hat{y}_{0,m}$ with $l - 1 = q$ distinct vertices $\hat{x}_{m,i}$, $i = 0, 1, \ldots, q-1$. This gives Figure 2.5.

Now we consider connecting the other outgoing edges of each $\hat{x}_{m,i}$ vertex to the remaining $\hat{y}_{j,\mu}$ vertices, $j \neq 0$. The set of mutually orthogonal squares of order $q = 3$, given in Example 2.2, guarantees that 4-cycles do not get introduced in step 5. For vertex $\hat{x}_{m,i}$, each row of Table 2.3 shows the initial connection from vertex $\hat{y}_{0,m}$, as well as the additional $k-1 = q$ connections to the vertices $\hat{y}_{j+1,L_{i,j}^{(m)}}$, $j = 0, 1, \ldots, q-1$. For example, after the first 3 rows (corresponding to layer 3 vertices $\hat{x}_{0,0}, \hat{x}_{0,1}, \hat{x}_{0,2}$), we get the connections shown in Figure 2.6.

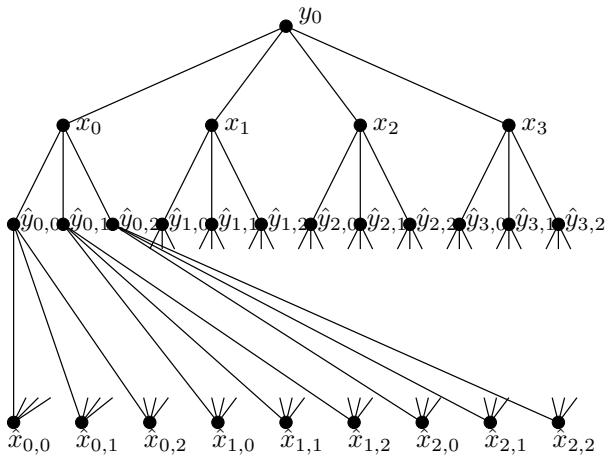


Figure 2.5: Construction of bipartite cage graph with $k = 4$ and $l = 4$, at the conclusion of step 4 of Algorithm 2.2.

$\hat{x}_{m,i}$	$\hat{y}_{j,\mu}$			
$(m,\ i)$	$(0,\ m)$	$(1,\ L_{i,0}^{(m)})$	$(2,\ L_{i,1}^{(m)})$	$(3,\ L_{i,2}^{(m)})$
$(0,\ 0)$	$(0,\ 0)$	$(1,\ 0)$	$(2,\ 0)$	$(3,\ 0)$
$(0,\ 1)$	$(0,\ 0)$	$(1,\ 1)$	$(2,\ 1)$	$(3,\ 1)$
$(0,\ 2)$	$(0,\ 0)$	$(1,\ 2)$	$(2,\ 2)$	$(3,\ 2)$
$(1,\ 0)$	$(0,\ 1)$	$(1,\ 0)$	$(2,\ 1)$	$(3,\ 2)$
$(1,\ 1)$	$(0,\ 1)$	$(1,\ 1)$	$(2,\ 2)$	$(3,\ 0)$
$(1,\ 2)$	$(0,\ 1)$	$(1,\ 2)$	$(2,\ 0)$	$(3,\ 1)$
$(2,\ 0)$	$(0,\ 2)$	$(1,\ 0)$	$(2,\ 2)$	$(3,\ 1)$
$(2,\ 1)$	$(0,\ 2)$	$(1,\ 1)$	$(2,\ 0)$	$(3,\ 2)$
$(2,\ 2)$	$(0,\ 2)$	$(1,\ 2)$	$(2,\ 1)$	$(3,\ 0)$

Table 2.3: Connections between vertices $\hat{x}_{m,i}$ and $\hat{y}_{j,\mu}$ for $k = 4$, $l = 4$. The column underneath $\hat{x}_{m,i}$ gives the (m, i) index of the particular layer 3 vertex, and the columns underneath $\hat{y}_{j,\mu}$ give the (j, μ) indices of the connected layer 2 vertices.

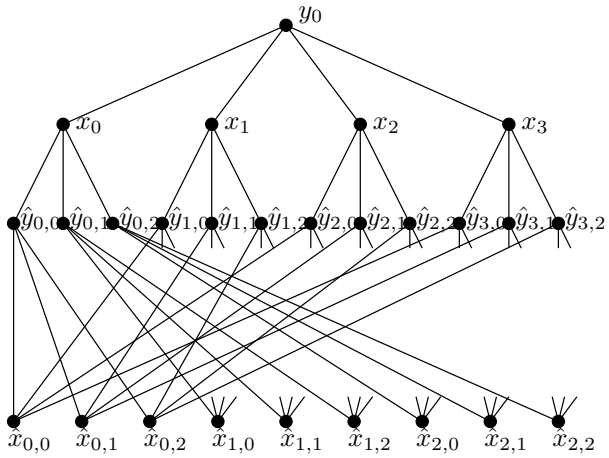


Figure 2.6: Construction of bipartite cage graph with $k = 4$ and $l = 4$, after partial completion of step 5 of Algorithm 2.2, where only the $m = 0$ vertices of $\hat{x}_{m,i}$ have made the appropriate connections.

After all of the connections of Algorithm 2.2 are made, this results in the bipartite cage graph given in Figure 2.7 (where the $\hat{x}$ and $\hat{y}$ vertices have been relabeled to be indexed sequentially).

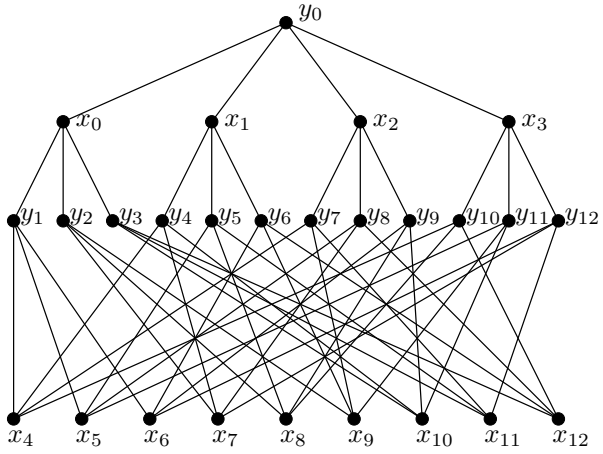


Figure 2.7: Bipartite cage graph corresponding to $k = 4$ and $l = 4$.

2.3.3 Properties of Regular Graph Constructed from Algorithm 2.2

We now show that the graph constructed from Algorithm 2.2 is indeed a cage graph, and discuss additional properties of the resulting graph.

Lemma 2.2. *In the bipartite graph constructed from Algorithm 2.2, the shortest cycle consists of at least* 6 *vertices.*

Proof. We need only to show that there are no cycles of length 2 and length 4, as odd cycles cannot occur in a bipartite graph. Clearly, there are no cycles of length 2, since the graph is simple (i.e., no multiple edges between vertices).

To show that there are no cycles of length 4, we consider vertices from each particular layer, and show that the construction of Algorithm 2.2 results in no 4-cycle involving the vertices at that layer. For layer 0, we first note that there are no 4-cycles that include vertex y_0, as layers 0, 1, and 2 form a tree of depth 3. Now consider any 4-cycles that include some vertex x_j from layer 1. Such a 4-cycle must also include $\hat{y}_{j,m}$ and $\hat{y}_{j,m'}$ for some $m \neq m'$ (and $m, m' \in \{0, 1, \ldots, k-2\}$). If $j = 0$, then step 4 of the algorithm guarantees that $\hat{y}_{j,m}$ and $\hat{y}_{j,m'}$ do not connect to any layer 3 vertices in common. For $j \neq 0$, since any layer 3 vertex $\hat{x}_{m,i}$ is connected to at most one vertex of $\{\hat{y}_{j,\mu} \mid \mu = 0, 1, \ldots, k-2\}$, so the vertices $\hat{y}_{j,m}$ and $\hat{y}_{j,m'}$ cannot be connected to the same layer 3 vertex for $m \neq m'$.

This leaves 4-cycles consisting only of layer 2 and layer 3 vertices. Suppose that a vertex $\hat{y}_{0,m}$ is a member of a 4-cycle (for any $m \in \{0, 1, \ldots, k-2\}$). Note that $\hat{y}_{0,m}$ is connected only to the $l-1$ vertices $\hat{x}_{m,i}$, $i = 0, 1, \ldots, l-2$. Because $L^{(m)}$ has Latin columns (even for $L^{(0)}$), we see that the layer 3 vertices $\hat{x}_{m,i}$ and $\hat{x}_{m,i'}$, where $i \neq i'$, will never connect to the same layer 2 vertex, i.e., $\hat{y}_{j+1,L^{(m)}_{i,j}} \neq \hat{y}_{j+1,L^{(m)}_{i',j}}$ for any $j = 0, 1, \ldots, l-2$. (Of course, $\hat{x}_{m,i}$ and $\hat{x}_{m,i'}$ are both connected to $\hat{y}_{0,m}$, but they are connected to no other common vertex.) Thus, $\hat{y}_{0,m}$ cannot be a member of a 4-cycle.

Now consider a potential 4-cycle consisting of vertices $\hat{y}_{j,\mu}$ and $\hat{y}_{j',\mu'}$, where $j, j' \neq 0$ and $j \neq j'$. Then there will be two layer 3 vertices $\hat{x}_{m,i}$ and $\hat{x}_{m',i'}$ such that $L^{(m)}_{i,j-1} = \mu = L^{(m')}_{i',j-1}$ and $L^{(m)}_{i,j'-1} = \mu' = L^{(m')}_{i',j'-1}$. However, this would imply that

the two squares $L^{(m)}$ and $L^{(m')}$ have two separate columns, $j-1$ and $j'-1$, where equal corresponding entries between the two squares can be found; this contradicts Lemma A.5, since only the zeroth column has equal corresponding entries. Thus no 4-cycles exist that involve layer 2 vertices.

Since layer 3 vertices must connect to layer 2 vertices, the sequence of contradictions implies that the shortest cycle consists of at least 6 vertices. □

Theorem 2.3 (see also [88, Section 4])**.** *The regular bipartite graph constructed from Algorithm 2.2 is a bipartite cage graph of girth at least* 6*, with degree* $q+1$ *at all vertices.*

Proof. Algorithm 2.2 is constructed with the same number of vertices as Algorithm 2.1, so the constructed graph has the minimum number of vertices for the required degree distributions for X and Y (see also Lemma 2.1). That is, Algorithm 2.2 results in $1+l(k-1) = q^2+q+1$ vertices for Y and $l+l(l-1)(k-1)/k = q^2+q+1$ vertices for X in the bipartite graph, where every vertex has degree $q+1$. Thus $v = |Y|$ and $u = |X|$ achieve the lower bounds of Lemma 2.1 for the required degree distributions. By Lemma 2.2, the shortest cycle has at least 6 vertices, so the result is shown. □

The method of proof of Theorem 2.3 can be used to prove that constructions similar to Algorithm 2.2 (including the skeleton algorithm given in Algorithm 2.1) result in cage graphs. This is because any such constructions will have the minimum number of vertices for the degree parameters (k, l), and then one only needs to check for the absence of cycles of length 4.

We now interpret the vertices of Y as the elements, and the vertices of X as the blocks of a Steiner system. That is, since each vertex $y \in Y$ is connected to the $l = q+1$ vertices $\tilde{X} = \{\tilde{x}_j \mid j = 0, 1, \ldots, l-1\} \subseteq X$, then element y is contained within each of the l blocks $\tilde{x}_j$, $j = 0, 1, \ldots, l-1$ (and thus has l replicas). Moreover, since each $x \in X$ is connected to $k = q+1$ vertices $\tilde{Y} = \{\tilde{y}_i \mid i = 0, 1, \ldots, k-1\} \subseteq Y$, thus each block x contains k elements. As mentioned previously, no two blocks $x_1, x_2 \in X$ share any pairs of elements, as there are no cycles of length 4 in the constructed graph.

Thus we have constructed a $S(2, k, v) = S(2, q + 1, q^2 + q + 1)$ Steiner system—and also a corresponding storage system design.

We see that in order to generate the cage graph and associated block system, the only required information is the generator element used to generate the multiplicative group for the finite field—as the set of mutually orthogonal squares can then be uniquely determined (for example, via the method in Appendix A.2). Thus lookup tables for the entire block design need not be stored, since such tables can always be generated easily.

In fact, the constructibility of a regular cage graph with $q^2 + q + 1$ vertices in each vertex set is equivalent to the constructibility of a projective plane of order $q+1$ [10]. For the regular cage graph with $k = l = 3$, we get the Heawood graph; see Figure 2.8. The associated Steiner system is shown in Figure 2.9. We comment that this construction of the Heawood graph is analogous to the Skolem construction [53] of Steiner triple systems for $v = 9$.

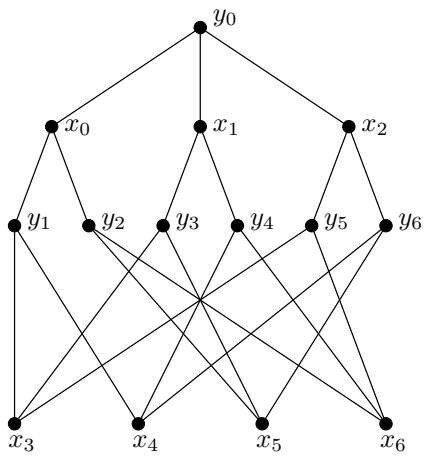
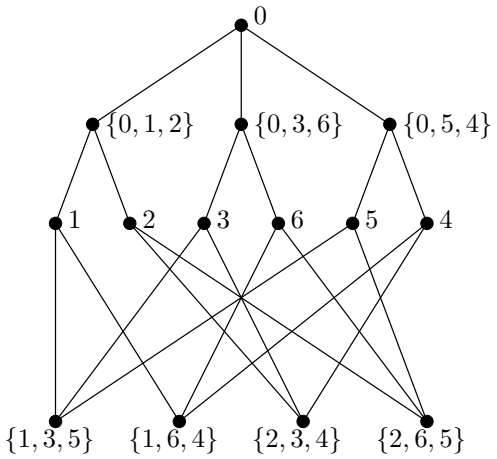


Figure 2.8: Bipartite cage graph corresponding to $k = 3$ and $l = 3$.

We also mention that similar methods can be used to construct regular graphs (i.e., $k = l = q + 1$) of girth 6 when q is not a prime power. For example in [65], the authors construct the girth-6 cage graph where $k = l = 7$ (so $q = 6$), which has a total of 90 vertices (45 vertices in each vertex set). However, here the number of vertices in each vertex set is strictly larger than the lower bounds of 43 vertices in



(a) Bipartite graph.

b_0	0	1	2
b_1	0	3	6
b_2	0	4	5
b_3	1	3	5
b_4	1	4	6
b_5	2	3	4
b_6	2	5	6

(b) Block design.

Figure 2.9: Steiner system corresponding to $k = 3$ and $l = 3$. Single numbers refer to elements, and triplets represent specific blocks. Figure 2.9(a) visualizes the system as a bipartite graph, and Figure 2.9(b) shows the corresponding block design. This gives the same Steiner system as in [31, Figure 3 (Example 2)], except for some permutations of blocks.

each vertex set derived from Lemma 2.1. We do not pursue such graphs with $q = 6$ any further in this work, as our later scalable constructions depend on cage graphs where the Lemma 2.1 lower bounds are achieved with equality.

2.4 Pruned Cage Graphs

In this section, we give a method for constructing girth-6 bipartite cage graphs where the degrees of the two vertex sets are slightly unequal, and thus the graphs are biregular. More specifically, the vertex degrees will satisfy $\deg(X) = k = q$ and $\deg(Y) = l = q + 1$, where q is any prime or power of a prime. We give an algorithm for constructing such graphs, then provide an example of the construction algorithm, followed by a proof that the constructed graphs are indeed cage graphs of girth 6. The resulting graphs will have $|X| = q^2 + q$ and $|Y| = q^2$. We call these graphs *pruned*, as they are constructed by starting with the regular cage graphs and then pruning certain vertices and edges.

2.4.1 Construction of Pruned Cage Graphs

For these constructions, we interpret the 0 symbol as an erasure symbol, that is, symbolizing a lack of connection—or a connection to a fictitious vertex. The use of an erasure symbol is analogous to the construction of an affine plane from a projective plane, where a line and all the points along that line are removed from the projective plane in order to arrive at an affine plane [5].

We first define the set of $q \times (q+1)$ matrices $\{\Lambda^{(m)} \mid m = 0, 1, \ldots, q-1\}$, which are derived from the set of squares $\{L^{(m)} \mid m = 0, 1, \ldots, q-1\}$. The (i, j) entry of $\Lambda^{(m)}$ is given by

$$\Lambda_{i,j}^{(m)} = \begin{cases} m & \text{if } j = 0 \\ L_{i,j-1}^{(m)} & \text{if } j \neq 0 \end{cases}, \tag{2.5}$$

for $i = 0, 1, \ldots, q-1$ and $j = 0, 1, \ldots, q$. In fact, the matrices $\Lambda^{(m)}$ are the same as the connection table of Algorithm 2.2 (i.e., Table 2.3).

The following observations can be made about the set of $\Lambda^{(m)}$ matrices:

O1. For $m = 0$, the $i = 0$ row of $\Lambda^{(0)}$ consists of all 0's, since $m = 0$ and also since all entries in the zeroth row of $L^{(0)}$ are 0.

O2. For $m = 0$ and row $i \neq 0$, the first column of $\Lambda^{(0)}$ consists of all 0's, and the rest of the row consists of q non-0 symbols (since all non-zeroth rows of $L^{(0)}$ consist entirely of non-0 symbols).

O3. For $m \neq 0$, the i-th row of $\Lambda^{(m)}$ consists of q non-0 symbols. This is because the zeroth column of $\Lambda^{(m)}$ has no 0 symbols and because the i-th row of $L^{(m)}$ is Latin, so has $q-1$ non-0 symbols.

We delete the $i = 0$ row of $\Lambda^{(0)}$, as it consists entirely of the erasure symbol 0 (from Observation O1). Then from Observations O2 and O3, we see that all the other rows have exactly q non-0 (i.e., non-erasure) symbols. There are exactly $q^2 - 1 = l(k-1)$ such rows that consist of q non-0 symbols.

Now we show how to construct the bipartite cage graph when $k = q$ and $l = q+1$,

in Algorithm 2.3. It is worth noting that the first three steps of Algorithm 2.3 are exactly the same as in Algorithm 2.2, except with the appropriate k and l.

Algorithm 2.3 Construction of bipartite cage when $k = q$ and $l = q + 1$

1: [**Layer** 0] Start with a single vertex $y_0 \in Y$.
2: [**Layer** 1] Connect y_0 to $l = q + 1$ vertices of X. Without loss of generality, call these vertices $x_0, x_1, \ldots, x_{l-1}$.
3: [**Layer** 2] For each vertex x_j, $j = 0, 1, \ldots, l - 1$, connect x_j to $k - 1 = q - 1$ vertices of Y. Let $\hat{y}_{j,m}$, $m = 0, 1, \ldots, k - 2$, denote the vertices of this step that are connected to vertex x_j.
4: [**Layer** 3] Call a set of $q^2 - 1$ distinct vertices (from vertex set X) as $\hat{x}_{m,i}$, where $m = 0, 1, \ldots, k - 1$, $i = 0, 1, \ldots, l - 2$, and excluding the case $(m, i) = (0, 0)$.
5: Consider a particular vertex $\hat{x}_{m,i}$, where $m \in \{0, 1, \ldots, k-1\}$, $i \in \{0, 1, \ldots, l-2\}$, and $(m, i) \neq (0, 0)$. Connect $\hat{x}_{m,i}$ to the vertices $\hat{y}_{j,\Lambda_{i,j}^{(m)}-1}$, $j = 0, 1, \ldots, l-1$, where a connection is made if and only if $\Lambda_{i,j}^{(m)} \neq 0$.

The $l(k-1) = q^2 - 1$ layer 2 vertices $\hat{y}_{j,m}$, $j = 0, 1, \ldots, q$ and $m = 0, 1, \ldots, q - 2$, introduced in step 3 are the same as the vertices $y_1, y_2, \ldots, y_{q^2-1}$. For example, we could use the mapping $y_{j(q-1)+m+1} = \hat{y}_{j,m}$. Similarly, the $q^2 - 1$ layer 3 vertices $\hat{x}_{m,i}$, $m = 0, 1, \ldots, q-1$ and $i = 0, 1, \ldots, q-1$, $(m, i) \neq (0, 0)$, introduced in step 4 are the same as the vertices $x_{q+1}, x_{q+2}, \ldots, x_{q+q^2-1}$, and can be mapped using $x_{q+mq+i} = \hat{x}_{m,i}$.

Notice that the resulting bipartite graph consists of the layer 0 and layer 2 vertices on one side of the graph, connected only to layer 1 and layer 3 vertices on the other side of the graph. It is clear that the graph constructed from Algorithm 2.2 is bipartite, where the two sets of vertices are Y and X, respectively.

We first show an example of the construction of Algorithm 2.3 with $k = 3$ and $l = 4$ (so $q = 3$), before proving that this indeed results in the desired cage graph.

2.4.2 Example of Pruned Cage Graph Construction: $k = 3$, $l = 4$

Here we show the construction of Algorithm 2.3, where $q = 3$, so that the vertices of X have degree $k = 3$ and the vertices of Y have degree $l = 4$. It turns out

that the constructed bipartite graph will have number of vertices $|X| = u = 12$ and $|Y| = v = 9$.

The first three steps of the algorithm are straightforward, as they involve connecting the vertices of layers 0, 1, and 2 in a tree. The result is shown in Figure 2.10.

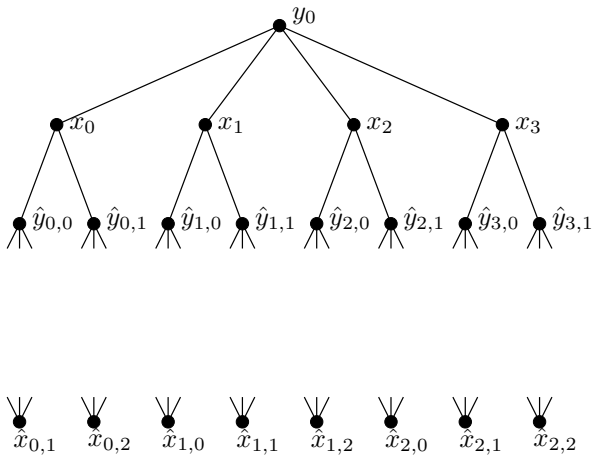


Figure 2.10: Construction of bipartite cage graph with $k = 3$ and $l = 4$, at the conclusion of step 3 of Algorithm 2.3.

Step 4 involves connecting all the vertices associated with $\hat{y}_{0,m}$ with the appropriate vertices $\hat{x}_{m,i}$. That is, for each $m \in \{0, 1, \ldots, q-2\}$, connect $\hat{y}_{0,m}$ with $l - 1 = q$ distinct vertices $\hat{x}_{m,i}$, $i = 0, 1, \ldots, q-1$.

Now we consider connecting the outgoing edges of each $\hat{x}_{m,i}$ vertex back to the $\hat{y}_{j,\mu}$ vertices. The set of mutually orthogonal squares of order $q = 3$ are the same squares as in Example 2.2. This gives the following $\Lambda^{(m)}$ matrices:

$$\Lambda^{(0)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 2 & 2 & 2 \end{bmatrix}, \quad \Lambda^{(1)} = \begin{bmatrix} 1 & 0 & 1 & 2 \\ 1 & 1 & 2 & 0 \\ 1 & 2 & 0 & 1 \end{bmatrix}, \quad \Lambda^{(2)} = \begin{bmatrix} 2 & 0 & 2 & 1 \\ 2 & 1 & 0 & 2 \\ 2 & 2 & 1 & 0 \end{bmatrix}. \quad (2.6)$$

Therefore, in step 5, the outgoing edges of $\hat{x}_{m,i}$ to $\hat{y}_{j,\mu}$ are connected according to Table 2.4, where $*$ denotes an erased symbol (i.e., no edge connection) and where the columns corresponding to $\hat{y}_{j,\mu}$ actually show $(j, \mu + 1)$. Notice that Table 2.4 is identical to Table 2.3, except that any 0 elements are now erased, and Table 2.4 is missing the $(m, j) = (0, 0)$ row of Table 2.3. This makes sense since the two tables

are constructed in the same way, except that there is a missing row, and that any 0 elements are now erased.

$\hat{x}_{m,i}$	$\hat{y}_{j,\mu}$			
$(m,\ i)$	$(0,\ m)$	$(1,\ L_{i,0}^{(m)})$	$(2,\ L_{i,1}^{(m)})$	$(3,\ L_{i,2}^{(m)})$
$(0,\ 1)$	$(0,\ *)$	$(1,\ 1)$	$(2,\ 1)$	$(3,\ 1)$
$(0,\ 2)$	$(0,\ *)$	$(1,\ 2)$	$(2,\ 2)$	$(3,\ 2)$
$(1,\ 0)$	$(0,\ 1)$	$(1,\ *)$	$(2,\ 1)$	$(3,\ 2)$
$(1,\ 1)$	$(0,\ 1)$	$(1,\ 1)$	$(2,\ 2)$	$(3,\ *)$
$(1,\ 2)$	$(0,\ 1)$	$(1,\ 2)$	$(2,\ *)$	$(3,\ 1)$
$(2,\ 0)$	$(0,\ 2)$	$(1,\ *)$	$(2,\ 2)$	$(3,\ 1)$
$(2,\ 1)$	$(0,\ 2)$	$(1,\ 1)$	$(2,\ *)$	$(3,\ 2)$
$(2,\ 2)$	$(0,\ 2)$	$(1,\ 2)$	$(2,\ 1)$	$(3,\ *)$

Table 2.4: Connections between vertices $\hat{x}_{m,i}$ and $\hat{y}_{j,\mu}$ for $k = 3$, $l = 4$. The column underneath $\hat{x}_{m,i}$ gives the (m, i) index of the particular layer 3 vertex, and the columns underneath $\hat{y}_{j,\mu}$ give the $(j, \mu + 1)$ indices of the connected layer 2 vertices—except for erased symbols, denoted by $*$. Note here that to preserve consistency with previous notation, the columns underneath $\hat{y}_{j,\mu}$ report $(j, \mu + 1)$ instead of (j, μ).

After all of the connections of Algorithm 2.3 are made, this results in the bipartite cage graph given in Figure 2.11 (where the $\hat{x}$ and $\hat{y}$ vertices have been relabeled to be indexed sequentially).

If we consider the vertices Y as elements and the vertices X as blocks containing the elements, then we get the blocks shown in Figure 2.12.

2.4.3 Properties of Pruned Graph Constructed from Algorithm 2.3

We now show that the graph constructed from Algorithm 2.3 is indeed a cage graph, and discuss additional properties of the resulting graph.

Lemma 2.4. *In the bipartite graph constructed from Algorithm 2.3, the shortest cycle consists of at least* 6 *vertices.*

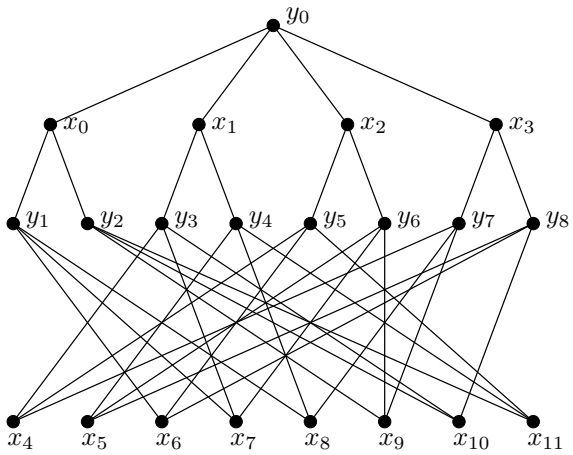


Figure 2.11: Bipartite cage graph corresponding to $k = 3$ and $l = 4$.

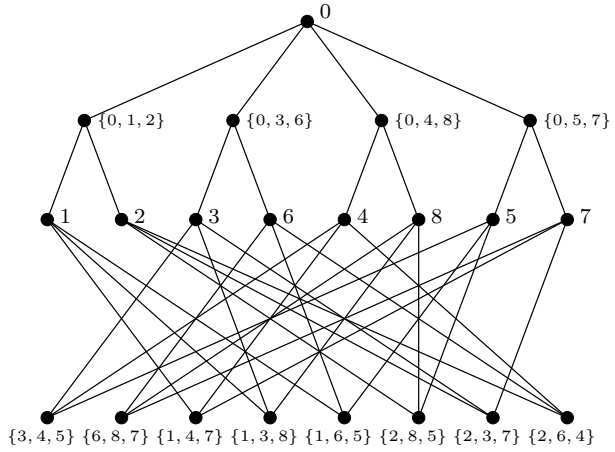


Figure 2.12: Steiner system corresponding to $k = 3$ and $l = 4$. Single numbers refer to elements, and triplets represent specific blocks. This gives the same Steiner system as in [31, Figure 6(a)]. The corresponding block design is the one given in Figure 2.1.

Proof. Since the graph of Algorithm 2.3 is pruned (i.e., vertices and edges removed, with no new vertices or edges added) from the graph of Algorithm 2.2, the proof is immediate from Lemma 2.2. We shall elucidate some of the steps of the proof again, for completeness.

We need only to show that there are no cycles of length 2 and length 4, as odd cycles cannot occur in a bipartite graph. Clearly, there are no cycles of length 2, since the graph is simple (i.e., no multiple edges between vertices).

Similar to Lemma 2.2, there are no cycles of length 4 involving any vertices from layers 0 and 1.

This leaves 4-cycles consisting only of layer 2 and layer 3 vertices. Suppose that a vertex $\hat{y}_{0,m}$ is a member of a 4-cycle (for any $m \in \{0, 1, \ldots, k-2\}$). Note that $\hat{y}_{0,m}$ is connected only to the $l-1$ vertices $\hat{x}_{m,i}$, $i = 0, 1, \ldots, l-2$. Because $L^{(m)}$ has Latin columns (even for $L^{(0)}$), we see that the layer 3 vertices $\hat{x}_{m,i}$ and $\hat{x}_{m,i'}$, where $i \neq i'$, will never connect to the same layer 2 vertex, i.e., $\hat{y}_{j+1,L^{(m)}_{i,j}} \neq \hat{y}_{j+1,L^{(m)}_{i',j}}$ for any $j = 0, 1, \ldots, l-2$. (Of course, $\hat{x}_{m,i}$ and $\hat{x}_{m,i'}$ are both connected to $\hat{y}_{0,m}$, but they are connected to no other common vertex.) Thus, $\hat{y}_{0,m}$ cannot be a member of a 4-cycle.

Now consider a potential 4-cycle consisting of vertices $\hat{y}_{j,\mu}$ and $\hat{y}_{j',\mu'}$, where $j, j' \neq 0$ and $j \neq j'$. Then there will be two layer 3 vertices $\hat{x}_{m,i}$ and $\hat{x}_{m',i'}$ such that $L^{(m)}_{i,j-1} = \mu = L^{(m')}_{i',j-1}$ and $L^{(m)}_{i,j'-1} = \mu' = L^{(m')}_{i',j'-1}$. However, this would imply that the two squares $L^{(m)}$ and $L^{(m')}$ have two separate columns, $j-1$ and $j'-1$, where equal corresponding entries can be found; this contradicts Lemma A.5. Thus no 4-cycles exist that involve layer 2 vertices.

Since layer 3 vertices must connect to layer 2 vertices, the sequence of contradictions implies that the shortest cycle consists of at least 6 vertices. □

Theorem 2.5. *The regular bipartite graph constructed from Algorithm 2.3 is a bipartite cage graph of girth at least* 6*, with degree q at vertices X and degree $q+1$ at vertices Y.*

Proof. Algorithm 2.3 is constructed with the same number of vertices as Algorithm 2.1, so the constructed graph has the minimum number of vertices for the required degree distributions for X and Y (see also Lemma 2.1). That is, Algorithm 2.3 results in

$1 + l(k-1) = q^2$ vertices for Y, each of degree $q+1$, and $l + l(l-1)(k-1)/k = q^2 + q = q(q+1)$ vertices for X, each of degree q. By Lemma 2.4, the shortest cycle has at least 6 vertices, so the result is shown. □

We now interpret the vertices of Y as the elements, and the vertices of X as the blocks of a Steiner system. That is, since each vertex $y \in Y$ is connected to the $l = q+1$ vertices $\tilde{X} = \{\tilde{x}_j \mid j = 0, 1, \ldots, l-1\} \subseteq X$, then element y is contained within each of the l blocks $\tilde{x}_j$, $j = 0, 1, \ldots, l-1$ (and thus has l replicas). Moreover, since each $x \in X$ is connected to $k = q$ vertices $\tilde{Y} = \{\tilde{y}_i \mid i = 0, 1, \ldots, k-1\} \subseteq Y$, thus each block x contains k elements. As mentioned previously, no two blocks $x_1, x_2 \in X$ share any pairs of elements, as there are no cycles of length 4 in the constructed graph. Thus we have constructed a $S(2, k, v) = S(2, q, q^2)$ Steiner system. Furthermore, we can also interpret the vertices with Y being the blocks and X being the elements of a different system. This gives a $S^T(2, l, u) = S^T(2, q+1, q(q+1))$ transpose system.

In fact, the constructibility of a biregular cage graph with $q(q+1)$ vertices in one vertex set and q^2 vertices in the other vertex set (where the vertex degrees are q and $q+1$, respectively) is equivalent to the constructibility of an affine plane of order $q+1$. The associated block design can also be constructed algebraically for certain cases without resorting to the graph-based approach; this is shown in Appendix B. We comment that this construction of the bipartite cage graph when $k = 3$ and $l = 4$ is analogous to the Bose construction [53] of Steiner triple systems for $v = 9$.

2.5 Scalable Designs

Starting either from the regular cage graphs of Section 2.3 or from the pruned cage graphs of Section 2.4, it is possible to construct cage graphs where the vertex degrees of the two vertex sets are highly unbalanced. These scenarios are more practical for the design of distributed storage systems. In Section 2.5.1, we construct cage graphs where the vertex degrees are $\deg(X) = k = q+1$ but $\deg(Y) = l = q^n + q^{n-1} + \cdots + q+1$, which gives an extension to the construction of the regular cage graphs. Then in Section 2.5.2, we construct cage graphs where the vertex degrees are $\deg(X) = k = q$

but $\deg(Y) = l = q^n + q^{n-1} + \cdots + q + 1$, which gives an extension to the construction of the pruned cage graphs. For both of these constructions, we give example parameter values of possible constructions. The scaled constructions are iterative in nature, and also have favorable scalability properties, which we discuss in Section 2.5.3.

2.5.1 Construction of Designs with $k = q + 1$, $l = q^n + \cdots + q + 1$

In this exposition, we have $l = p_n(q) = q^n + \cdots + q + 1$. Later we will see that the construction is recursive; thus we call $l[n] = p_n(q)$ as the degree of the vertices in the vertex set Y, at iteration n.[9] (For notational simplicity, if no iteration number is specified, then it is assumed that we are referring to the quantity for the n-th iteration.) The construction will construct Steiner systems of the form $S(2, q + 1, q^{n+1} + q^n + \cdots + q^2 + q + 1)$. That is, for $\deg(X) = k = q + 1$ and $\deg(Y) = l = p_n(q)$, we construct cage graphs where $|X| = \frac{p_{n+1}(q)p_n(q)}{q+1}$ and $|Y| = p_{n+1}(q)$. We show such a graph in Figure 2.13.

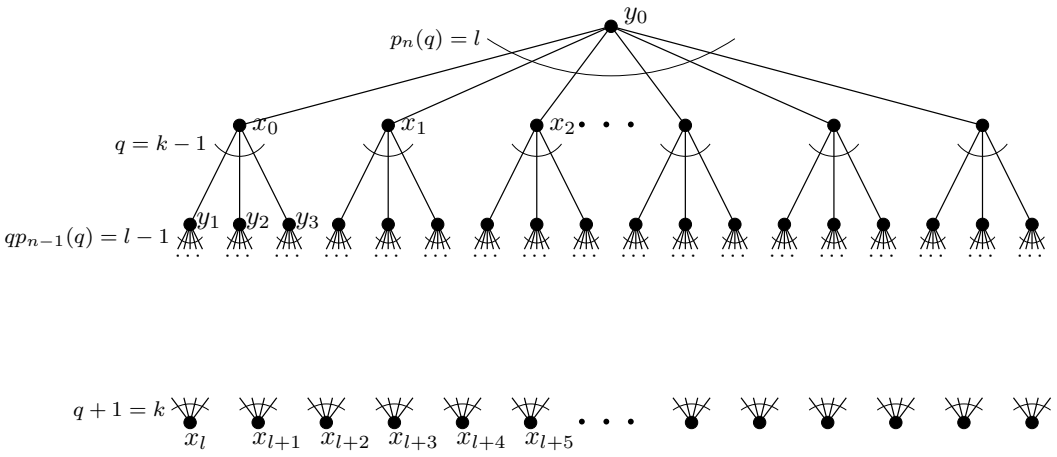


Figure 2.13: Construction of bipartite cage graph with $k = q + 1$ and $l = p_n(q) = q^n + q^{n-1} + \cdots + q + 1$.

We will inductively construct bipartite cage graphs with $k = q+1$ and $l[n] = p_n(q)$

[9] We also let $u[n]$ and $v[n]$ denote the respective quantities at iteration n. In contrast, $k[n] = q+1$ for all n, so we do not qualify k with the iteration number n.

using a layered method similar to the algorithms from before. Notice that for $n = 1$ the graph is the $k = l = q + 1$ cage.

Thus suppose that a cage graph with parameters $k = q+1$ and $l[n-1] = p_{n-1}(q)$ exists. For this graph, $v[n-1] = p_n(q)$ and $u[n-1] = \frac{p_n(q)p_{n-1}(q)}{q+1}$. By taking the vertices of Y as the elements and the vertices of X as the blocks, this gives a Steiner system with block size $k = q + 1$ and $v[n-1] = p_n(q)$ total elements, i.e., $S(2, q+1, p_n(q))$. (Here, each element is repeated $l[n-1] = p_{n-1}(q)$ times, and there are a total of $u[n-1] = \frac{p_n(q)p_{n-1}(q)}{q+1}$ blocks.)

Once we have the $S(2, q + 1, p_n(q))$ Steiner system (where $l = p_{n-1}(q)$), we can use this to construct the $k = q + 1$, $l = p_n(q)$ cage, as given in Algorithm 2.4.

Algorithm 2.4 Construction of bipartite cage when $k = q + 1$ and $l = p_n(q)$

Require: A set of $v[n-1] = p_n(q)$ elements, and a collection $\mathcal{B} = \{b_h \mid h = 0, 1, \ldots, u[n-1]-1\}$ of $(q+1)$-element blocks b_h, such that each element has exactly $l[n-1] = p_{n-1}(q)$ replicas and no particular pair of elements occurs in more than one block.

1: **[Layer** 0] Start with a single vertex $y_0 \in Y$.
2: **[Layer** 1] Connect y_0 to $l = p_n(q)$ vertices of X. Without loss of generality, call these vertices $x_0, x_1, \ldots, x_{p_n(q)-1}$.
3: **[Layer** 2] For each vertex x_j, $j = 0, 1, \ldots, l-1$, connect x_j to $k-1 = q$ vertices of Y. Let $\hat{y}_{j,m}$, $m = 0, 1, \ldots, k-2$, denote the vertices of this step that are connected to vertex x_j.
4: **for** $h = 0$ to $u[n-1]-1$ **do**
5: Let the block b_h consist of elements $b_h = \{g_0, g_1, g_2, \ldots, g_q\}$.
6: **[Layer** 3] Connect each vertex $\hat{y}_{g_0,m}$ $(m = 0, 1, \ldots, k-2)$ to q distinct vertices of X, called $\hat{x}^{(h)}_{m,i}$, $i = 0, 1, \ldots, q-1$. Therefore, $\hat{x}^{(h)}_{m,i} \neq \hat{x}^{(h)}_{m',i'}$ unless $m = m'$ and $i = i'$. (For the h-th iteration, there will be a total of $(k-1)q = q^2$ such vertices $\hat{x}^{(h)}_{m,i}$.)
7: Consider a particular vertex $\hat{x}^{(h)}_{m,i}$, where $m \in \{0, 1, \ldots, k-2\}$ and $i \in \{0, 1, \ldots, q-1\}$. Connect $\hat{x}^{(h)}_{m,i}$ to vertices $\hat{y}_{g_{j+1}, L^{(m)}_{i,j}}$, where $j = 0, 1, \ldots, q-1$.
8: **end for**

Ensure: Bipartite cage with degrees $k = q + 1$, $l[n] = p_n(q)$, and number of vertices $|Y| = v[n] = p_{n+1}(q)$, $|X| = u[n] = \frac{p_{n+1}(q)p_n(q)}{q+1}$

It can be seen that Algorithm 2.4 is quite similar to Algorithm 2.2 for constructing

regular cage graphs. The methods of construction for layers 0 through 2 (in steps 1, 2, and 3) have almost the same flavor for the two algorithms, except for the vertex degree l being different. However, Algorithm 2.4 is different in steps 6 and 7, because we only connect via $\hat{y}_{g_j,m}$ (where $j = 0, 1, \ldots, q$) instead of $\hat{y}_{j,m}$ for all $j = 0, 1, \ldots, l-1$. This is due to only considering $(q+1)$-element *subsets* instead of the entire set of $x_0, \ldots, x_{l[n]}$ vertices when constructing each smaller subcage.

For each iteration h where we select the subset of layer 1 vertices denoted by $b_h = \{g_0, g_1, g_2, \ldots, g_q\}$, let us denote the subgraph induced by the subset of vertices

$$\begin{aligned} &\{y_0\} \cup \{x_j \mid j \in b_h\} \cup \{\hat{y}_{j,m} \mid j \in b_h,\ m = 0, 1, \ldots, k-2\} \\ &\cup \{\hat{x}^{(h)}_{m,i} \mid m = 0, 1, \ldots, k-2,\ i = 0, 1, \ldots, q-1\} \end{aligned} \tag{2.7}$$

as the b_h*-subgraph*.

Lemma 2.6. *The constructed bipartite graph of Algorithm 2.4 has the desired number of vertices in X and Y, and satisfies the degree requirements. That is, all vertices in Y have exactly $l[n] = p_n(q)$ outgoing edges, and all vertices in X have exactly $k = q+1$ outgoing edges. Furthermore, $|Y| = v[n] = p_{n+1}(q)$ and $|X| = u[n] = \frac{p_{n+1}(q)p_n(q)}{q+1}$.*

Proof. First we verify that we have the correct number of vertices. For Y, there is one layer 0 vertex. In layer 2, we will have $l(k-1) = p_n(q)q = p_{n+1}(q) - 1$ vertices, since each of the l vertices x_j, $j = 0, 1, \ldots, l-1$, is connected to $k-1$ different layer 2 vertices of Y. Thus $v[n] = |Y| = p_{n+1}(q)$. For X, there are $l[n] = p_n(q)$ layer 1 vertices. For layer 3, in each of the $u[n-1]$ iterations of step 6, there are $(k-1)q = q^2$ distinct vertices of X involved. Thus, layer 3 consists of $q^2u[n-1] = \frac{q^2p_n(q)p_{n-1}(q)}{q+1}$ vertices. Therefore, $u[n] = |X| = p_n(q) + \frac{q^2p_n(q)p_{n-1}(q)}{q+1} = \frac{p_{n+1}(q)p_n(q)}{q+1}$.

Now we count the number of edges from each vertex. For layer 0, step 2 results in degree of $l[n] = p_n(q)$ for vertex y_0. For layer 1, each vertex x_j, $j = 0, 1, \ldots, l-1$, is connected to exactly $q+1$ vertices (one edge to y_0 and then q edges to the layer 2 vertices), as can be seen from step 3.

Now consider a particular layer 2 vertex $\hat{y}_{j,m}$. We know that in the collection of subsets, $\mathcal{B}$, each element j is selected exactly $l[n-1] = p_{n-1}(q)$ times; thus, $\hat{y}_{j,m}$

occurs in exactly $l[n-1] = p_{n-1}(q)$ iterations. Moreover, in each iteration that $\hat{y}_{j,m}$ occurs, it has exactly q edges to the layer 3 vertices (whether or not j is the g_0 or some $g_{j'+1}$ of the current subset b_h). Therefore, each layer 2 vertex has one edge to its corresponding layer 1 vertex, and $qp_{n-1}(q)$ edges to the layer 3 vertices, for a total of $1 + qp_{n-1}(q) = p_n(q)$ edges—which is the desired degree for that vertex.

By construction, every layer 3 vertex $\hat{x}^{(h)}_{m,i}$ has degree $q+1$, that is, one edge from the associated $\hat{y}_{g_0,m}$ and q edges to the vertices $\hat{y}_{g_{j+1},L^{(m)}_{i,j}}$, $j = 0, 1, \ldots, q-1$, connected via the Latin squares method. □

Lemma 2.7. *In the constructed bipartite graph of Algorithm 2.4, the shortest cycle has length at least* 6*.*

Proof. As before, we need only check that there are no cycles of length 4. Since each selection b_h of layer 1 vertices induces a subgraph that is isomorphic to the $k = q + 1$, $l = q + 1$ bipartite regular cage graph, all properties from the regular graph—including Lemma 2.2—also hold for the subgraph. Specifically, this tells us that within any b_h-subgraph, there are no 4-cycles.

Consequently, any potential 4-cycle must involve only edges from layer 2 to layer 3 vertices, where the involved layer 2 vertices are connected to different x_j vertices of layer 1.[10] That is, suppose that the layer 2 vertices $\hat{y}_{j,\mu}$ and $\hat{y}_{j',\mu'}$, where $j \neq j'$, are involved in a 4-cycle with the layer 3 vertices $\hat{x}^{(h)}_{m,i}$ and $\hat{x}^{(h')}_{m',i'}$.[11] If a 4-cycle existed, then this would mean that the subsets b_h and $b_{h'}$ both contain the elements j and j'. That is, such a cycle implies that the b_h-subgraph must include the edge between $\hat{y}_{j,\mu}$ and $\hat{x}^{(h)}_{m,i}$, as well as the edge between $\hat{y}_{j',\mu'}$ and $\hat{x}^{(h)}_{m,i}$; also, the $b_{h'}$-subgraph must include the edge between $\hat{y}_{j,\mu}$ and $\hat{x}^{(h')}_{m',i'}$, as well as the edge between $\hat{y}_{j',\mu'}$ and $\hat{x}^{(h')}_{m',i'}$. This means that the subsets b_h and $b_{h'}$ both contain the elements j and j'. However, since b_h and $b_{h'}$ are subsets that do not share any pair of elements, the fact that $j, j' \in b_h$ and $j, j' \in b_{h'}$ is a contradiction. □

[10] For completeness, we note that if the two layer 2 vertices were connected to the same layer 1 vertex, so that $j = j'$, then there could be no 4-cycle, as a resulting 4-cycle would entirely occur within some b_h-subgraph.

[11] We know that $h \neq h'$, or else the 4-cycle would be entirely within the b_h-subgraph.

Lemma 2.8. *If a bipartite cage (of girth 6) with parameters $k = q + 1$, $l[n-1] = p_{n-1}(q)$, $v[n-1] = p_n(q)$, and $u[n-1] = \frac{p_n(q)p_{n-1}(q)}{q+1}$ exists, then Algorithm 2.4 constructs a bipartite cage of girth 6 with parameters $k = q + 1$, $l[n] = p_n(q)$—hence $v[n] = p_{n+1}(q)$, $u[n] = \frac{p_{n+1}(q)p_n(q)}{q+1}$.*

Proof. Follows from Lemmas 2.6 and 2.7. □

Theorem 2.9. *A bipartite cage of girth 6 with parameters $k = q+1$ and $l[n] = p_n(q)$ exists and is constructible. This graph has $v[n] = p_{n+1}(q)$ and $u[n] = \frac{p_{n+1}(q)p_n(q)}{q+1}$.*

Proof. The base case where $n = 1$ is the $k = q + 1$, $l = q + 1$ cage graph constructed from Algorithm 2.2, and so is constructible. The conclusion follows by induction, using Lemma 2.8. □

In Figure 2.14, we show an example of the resulting storage system design after iteration $n = 2$ of Algorithm 2.4, for the case where $q = 2$ (i.e., $k = 3$). This system is in fact an extension of the $k = 3$, $l = 3$ block design of Figure 2.9; for storage nodes $b_0, b_1, \ldots, b_6$, the first 3 data chunks in each node are exactly the same between Figures 2.9(b) and 2.14. We will show how this happens when we discuss the scalability of our constructed storage systems in Section 2.5.3.

b_0	0	1	2	7	8	9	10	b_5	2	3	4	27	30	31	34	b_{10}	8	13	14	24	26	32	34
b_1	0	3	6	11	14	15	18	b_6	2	5	6	28	29	32	33	b_{11}	9	15	17	19	20	31	32
b_2	0	4	5	12	13	16	17	b_7	7	11	13	19	21	27	29	b_{12}	9	16	18	21	22	33	34
b_3	1	3	5	19	22	23	26	b_8	7	12	14	20	22	28	30	b_{13}	10	15	16	23	24	27	28
b_4	1	4	6	20	21	24	25	b_9	8	11	12	23	25	31	33	b_{14}	10	17	18	25	26	29	30

Figure 2.14: Block design for distributed storage system corresponding to $k = 3$ and $l = 7$. That is, each data chunk has 3 replicas and each storage node stores 7 chunks each. In total there are 35 distinct data chunks and 15 storage nodes. This design is an extension of the $k = 3$, $l = 3$ design of Figure 2.9.

The constructed cage graphs form a family of designs where $k = q + 1$ and $l = p_n(q)$, so $v = p_{n+1}(q)$ and $u = \frac{p_{n+1}(q)p_n(q)}{q+1}$. Using transpose code designs, we can design distributed storage systems with the parameters listed in Table 2.5 for $q = 2$ and in Table 2.6 for $q = 3$; other parameters are possible using other valid values for q. We note that the constructed cage graphs correspond to the Steiner systems $S(2, q + 1, p_{n+1}(q))$, which are the projective geometry designs when q is a prime power.

replicas	node size	storage nodes	total chunks
k	l	v	u
3	3	7	7
3	7	15	35
3	15	31	155
3	31	63	651
3	63	127	2667
3	127	255	10795
3	255	511	43435
3	511	1023	174251

Table 2.5: Designs from projective geometries with $q = 2$, using transpose code designs.[12]

2.5.2 Construction of Designs with $k = q$, $l = q^n + \cdots + q + 1$

We can also construct Steiner systems of the form $S(2, q, q^{n+1})$. These systems arise from the cage graphs where $\deg(X) = k = q$ and $\deg(Y) = l = p_n(q)$, so $|X| = q^n p_n(q)$ and $|Y| = q^{n+1}$. The main difference between these cage graphs and the $k = q + 1$, $l = p_n(q)$ cages constructed previously is that now we require each vertex in X to have one fewer outgoing edge than in Section 2.5.1. As a result, we can get away with only having $|Y| = q^{n+1}$ instead of $|Y| = q^{n+1} + q^n + \cdots + q + 1$.

To construct the $S(2, q, q^{n+1})$ design, the starting condition for the algorithm is the Steiner system $S(2, q + 1, q^n + q^{n-1} + \cdots + q + 1)$ of Section 2.5.1. Then we can

[12] We could also have the labels signifying the original Steiner system design, where k is the node size, l is the number of replicas, v is the total number of distinct data chunks, and u is the number of storage nodes.

replicas	node size	storage nodes	total chunks
k	l	v	u
4	4	13	13
4	13	40	130
4	40	121	1210
4	121	364	11011
4	364	1093	99463
4	1093	3280	896260
4	3280	9841	8069620
4	9841	29524	72636421

Table 2.6: Designs from projective geometries with $q = 3$, using transpose code designs.[12]

run Algorithm 2.5 for one iteration to achieve the desired Steiner system. Thus the quantities $l[n']$, $u[n']$, and $v[n']$ for the early iterations $n' = 1, 2, \ldots, n-1$ are from Algorithm 2.4 of Section 2.5.1; and only the quantities $l[n]$, $u[n]$, and $v[n]$ of the n-th iteration are unlike the previous case. Notice also that only steps 3, 6, and 7 are different from the corresponding steps in Algorithm 2.4.

In this scenario, for each iteration h where we select the subset of layer 1 vertices denoted by $b_h = \{g_0, g_1, g_2, \ldots, g_q\}$, let us denote the subgraph induced by the subset of vertices

$$\begin{aligned} &\{y_0\} \cup \{x_j \mid j \in b_h\} \cup \{\hat{y}_{j,m} \mid j \in b_h,\ m = 0, 1, \ldots, k-1\} \\ &\quad \cup \{\hat{x}^{(h)}_{m,i} \mid m = 0, 1, \ldots, k-1,\ i = 0, 1, \ldots, q-1,\ (m,i) \neq (0,0)\} \end{aligned} \tag{2.8}$$

as the *b_h-subgraph*.

Lemma 2.10. *The constructed bipartite graph of Algorithm 2.5 has the desired number of vertices in X and Y, and satisfies the degree requirements. That is, all vertices in Y have exactly $l[n] = p_n(q)$ outgoing edges, and all vertices in X have exactly $k = q$ outgoing edges. Furthermore, $|Y| = v[n] = q^{n+1}$ and $|X| = u[n] = q^n p_n(q)$.*

Proof. First we verify that we have the correct number of vertices. For Y, there is one layer 0 vertex. In layer 2, we will have $l(k-1) = q^{n+1} - 1$ vertices, since each of the l vertices x_j, $j = 0, 1, \ldots, l-1$, is connected to $k-1$ different layer 2 vertices

Algorithm 2.5 Construction of bipartite cage when $k = q$ and $l = p_n(q)$

Require: A set of $v[n-1] = p_n(q)$ elements, and a collection $\mathcal{B} = \{b_h \mid h = 0, 1, \ldots, u[n-1]-1\}$ of $(q+1)$-element blocks b_h, such that each element has exactly $l[n-1] = p_{n-1}(q)$ replicas and no particular pair of elements occurs in more than one block.

1: **[Layer** 0] Start with a single vertex $y_0 \in Y$.
2: **[Layer** 1] Connect y_0 to $l = p_n(q)$ vertices of X. Without loss of generality, call these vertices $x_0, x_1, \ldots, x_{p_n(q)-1}$.
3: **[Layer** 2] For each vertex x_j, $j = 0, 1, \ldots, l-1$, connect x_j to $k-1 = q-1$ vertices of Y. Let $\hat{y}_{j,m}$, $m = 0, 1, \ldots, k-2$, denote the vertices of this step that are connected to vertex x_j.
4: **for** $h = 0$ to $u[n-1]-1$ **do**
5: Let the block b_h consist of elements $b_h = \{g_0, g_1, g_2, \ldots, g_q\}$.
6: **[Layer** 3] Call a set of q^2-1 distinct vertices (from vertex set X) as $\hat{x}^{(h)}_{m,i}$, where $m = 0, 1, \ldots, k-1$, $i = 0, 1, \ldots, q-1$, and excluding the case $(m, i) = (0, 0)$.
7: Consider a particular vertex $\hat{x}^{(h)}_{m,i}$, where $m \in \{0, 1, \ldots, k-1\}$, $i \in \{0, 1, \ldots, q-1\}$, and $(m, i) \neq (0, 0)$. Connect $\hat{x}^{(h)}_{m,i}$ to the vertices $\hat{y}_{g_j, \Lambda^{(m)}_{i,j}-1}$, $j = 0, 1, \ldots, q$, where a connection is made if and only if $\Lambda^{(m)}_{i,j} \neq 0$. (Recall that $\Lambda^{(m)}$ is defined in (2.5).)
8: **end for**

Ensure: Bipartite cage with degrees $k = q$, $l[n] = p_n(q)$, and number of vertices $|Y| = v[n] = q^{n+1}$, $|X| = u[n] = q^n p_n(q)$

of Y. Thus $v[n] = |Y| = q^{n+1}$. For X, there are $l[n] = p_n(q)$ layer 1 vertices. For layer 3, in each of the $u[n-1]$ iterations of step 6, there are $kq - 1 = q^2 - 1$ distinct vertices of X involved. Thus, layer 3 consists of $(q^2-1)u[n-1] = (q-1)p_n(q)p_{n-1}(q)$ vertices. Therefore, $u[n] = |X| = p_n(q) + (q-1)p_n(q)p_{n-1}(q) = q^n p_n(q)$.

Now we count the number of edges from each vertex. For layer 0, step 2 results in degree of $l[n] = p_n(q)$ for vertex y_0. For layer 1, each vertex x_j, $j = 0, 1, \ldots, l[n]-1$, is connected to exactly $q+1$ vertices (one edge to y_0 and then q edges to the layer 2 vertices), as can be seen from step 3.

Now consider a particular layer 2 vertex $\hat{y}_{j,m}$. We know that in the collection of subsets, $\mathcal{B}$, each element j is selected exactly $l[n-1]$ times; thus, $\hat{y}_{j,m}$ occurs in exactly $l[n-1] = p_{n-1}(q)$ iterations. Moreover, in each iteration that $\hat{y}_{j,m}$ occurs,

it has exactly q edges to the layer 3 vertices. Therefore, each layer 2 vertex has one edge to its corresponding layer 1 vertex, and $qp_{n-1}(q)$ edges to the layer 3 vertices, for a total of $1 + qp_{n-1}(q) = p_n(q)$ edges—which is the desired degree for that vertex.

By construction, every layer 3 vertex $\hat{x}^{(h)}_{m,i}$ has degree q, since the vertex has q edges to the vertices $\hat{y}_{g_j,\Lambda^{(m)}_{i,j}-1}$, $j = 0, 1, \ldots, q$, connected via the Latin squares method (and with symbol 0 being erased). □

Lemma 2.11. *In the constructed bipartite graph of Algorithm 2.5, the shortest cycle has length at least* 6*.*

Proof. As before, we need only check that there are no cycles of length 4. Since each selection b_h of layer 1 vertices induces a subgraph that is isomorphic to the $k = q$, $l = q + 1$ bipartite cage graph, all properties from that graph—including Lemma 2.4—also hold for the subgraph. Specifically, this tells us that within any b_h-subgraph, there are no 4-cycles.

Consequently, any potential 4-cycle must involve only edges from layer 2 to layer 3 vertices, where the involved layer 2 vertices are connected to different x_j vertices of layer 1.[13] That is, suppose that the layer 2 vertices $\hat{y}_{j,\mu}$ and $\hat{y}_{j',\mu'}$, where $j \neq j'$, are involved in a 4-cycle with the layer 3 vertices $\hat{x}^{(h)}_{m,i}$ and $\hat{x}^{(h')}_{m',i'}$.[14] If a 4-cycle existed, then this would mean that the subsets b_h and $b_{h'}$ both contain the elements j and j'. That is, such a cycle implies that the b_h-subgraph must include the edge between $\hat{y}_{j,\mu}$ and $\hat{x}^{(h)}_{m,i}$, as well as the edge between $\hat{y}_{j',\mu'}$ and $\hat{x}^{(h)}_{m,i}$; also, the $b_{h'}$-subgraph must include the edge between $\hat{y}_{j,\mu}$ and $\hat{x}^{(h')}_{m',i'}$, as well as the edge between $\hat{y}_{j',\mu'}$ and $\hat{x}^{(h')}_{m',i'}$. This means that the subsets b_h and $b_{h'}$ both contain the elements j and j'. However, since b_h and $b_{h'}$ are subsets that do not share any pair of elements, the fact that $j, j' \in b_h$ and $j, j' \in b_{h'}$ is a contradiction. □

Theorem 2.12. *A bipartite cage of girth* 6 *with parameters $k = q$ and $l[n] = p_n(q)$ exists and is constructible. This graph has $v[n] = q^{n+1}$ and $u[n] = q^n p_n(q)$.*

[13] For completeness, we note that if the two layer 2 vertices were connected to the same layer 1 vertex, so that $j = j'$, then there could be no 4-cycle, as a resulting 4-cycle would entirely occur within some b_h-subgraph.

[14] We know that $h \neq h'$, or else the 4-cycle would be entirely within the b_h-subgraph.

Proof. By Theorem 2.9, we know that there exists a Steiner system $S(2, q+1, p_n(q))$.[15] Thus the conclusion follows from Lemmas 2.10 and 2.11. □

The constructed cage graphs form a family of designs where $k = q$ and $l = p_n(q)$, so $v = q^{n+1}$ and $u = q^n p_n(q)$. Using transpose code designs, we can design distributed storage systems with the parameters listed in Table 2.7 for $q = 3$; other parameters are possible using other valid values for q. We note that the constructed cage graphs correspond to the Steiner systems $S(2, q, q^{n+1})$, which are the affine geometry designs when q is a prime power.

replicas	node size	storage nodes	total chunks
k	l	v	u
3	4	9	12
3	13	27	117
3	40	81	1080
3	121	243	9801
3	364	729	88452
3	1093	2187	796797
3	3280	6561	7173360
3	9841	19683	64566801

Table 2.7: Designs from affine geometries with $q = 3$, using transpose code designs.[16]

2.5.3 Advantages of Scaled Constructions

The construction of Algorithm 2.4 is not merely a construction for a cage graph with large degree $l[n]$ for the vertices of Y. This particular construction also allows for the easy expansion of storage systems built using the methods in this work. That is, if an existing system has $l[n-1] = p_{n-1}(q)$, it is relatively simple to increase the size

[15] More importantly, we know that there exists a set of $v[n-1] = p_n(q)$ elements, and a collection $\mathcal{B}$ of $(q+1)$-element blocks such that each element has exactly $l[n-1] = p_{n-1}(q)$ replicas and no particular pair of elements occurs in more than one block.

[16] We could also have the labels signifying the original Steiner system design, where k is the node size, l is the number of replicas, v is the total number of distinct data chunks, and u is the number of storage nodes.

of the system so that the degree of Y has $l[n] = p_n(q)$. This is because the following holds:

Theorem 2.13. *Consider a cage graph, $G[n]$, with parameters $k = q+1$, $l[n] = p_n(q)$ constructed in iteration n of Algorithm 2.4. The cage graph with parameters $k = q+1$, $l[n-1] = p_{n-1}(q)$ (i.e., constructed in the previous iteration of Algorithm 2.4, and called $G[n-1]$) is a subgraph of $G[n]$.*

Theorem 2.13 will be proved with the help of Lemma 2.14.

Lemma 2.14. *Consider a cage graph with $k = q+1$, $l[n] = p_n(q)$, to be constructed in the n-th iteration of Algorithm 2.4. From the set of $p_n(q)$ elements and the collection of blocks $\mathcal{B}[n]$, it is possible to select a subset of $p_{n-1}(q)$ elements, called $\mathcal{S}[n-1]$, such that the subcollection of blocks from $\mathcal{B}[n]$ that contain only elements from $\mathcal{S}[n-1]$ is [isomorphic to] the entire collection of blocks $\mathcal{B}[n-1]$ required in the $(n-1)$-th iteration of the algorithm.*

Proof. Here, the elements are Y and the blocks are X. We now prove by induction.

The base case is where $n = 2$. The cage graph with parameters $k = q+1$, $l[1] = q+1$ is the graph constructed from Algorithm 2.2. In order to construct the cage graph with parameters $k = q+1$, $l[2] = q^2+q+1$ during iteration 2, we choose elements from the collection of blocks $\mathcal{B}[2] = X[1]$ (i.e., the block collection $\mathcal{B}$ at iteration 2 corresponds to the vertex set X at iteration 1). By construction, the block $b_0 \in \mathcal{B}[2]$ contains $q+1$ elements, so the subgraph associated with b_0 is isomorphic to the cage graph with parameters $k = q+1$, $l[1] = q+1$.

Now consider an arbitrary iteration n. From the $(n-1)$-th iteration, we know that $\mathcal{B}[n-2] \subset \mathcal{B}[n-1]$ (up to isomorphism with appropriate indexing of elements). Since $\mathcal{B}[n-1]$ is used in iteration n of Algorithm 2.4 for choosing subsets of layer 1 vertices to construct $G[n]$, and $\mathcal{B}[n-2]$ was used in iteration $n-1$ for choosing subsets of layer 1 vertices to construct $G[n-1]$, then the fact that $\mathcal{B}[n-2] \subset \mathcal{B}[n-1]$ results in $G[n-1]$ being a subgraph of $G[n]$. The block systems corresponding to $G[n-1]$ and $G[n]$ thus satisfy $\mathcal{B}[n-1] \subset \mathcal{B}[n]$ (again, up to isomorphism with appropriate

indexing of elements). Because the blocks of $\mathcal{B}[n-1]$ contain a total of $p_{n-1}(q)$ elements (i.e., $|\{y \in b \mid b \in \mathcal{B}[n-1]\}| = p_{n-1}(q)$), the result is shown. □

Now we can prove Theorem 2.13.

Proof of Theorem 2.13. From Lemma 2.14, we can select $p_{n-1}(q)$ layer 1 vertices such that the block system consisting of only these vertices is isomorphic to $\mathcal{B}[n-1]$. The subgraph constructed through these layer 1 vertices is thus isomorphic to the graph of the previous iteration, $G[n-1]$. □

For the distributed storage system, we take Y as the blocks and X as the elements. Thus each element has $k = q+1$ repetitions and each block has size $l = p_n(q) = q^n + q^{n-1} + \cdots + q + 1$ (such a system requires a total of $v = p_{n+1}(q)$ storage nodes and stores a total of $u = \frac{p_{n+1}(q)p_n(q)}{q+1}$ distinct data chunks). From Theorem 2.13, we see that because the $G[n-1]$ cage graph is a subgraph of the $G[n]$ cage, where the subgraph is a truncation of outgoing edges from each Y vertex, this means that the blocks of size $l[n-1] = p_{n-1}(q)$ are truncations of the blocks of size $l[n] = p_n(q)$. Equivalently, if we have constructed (using Algorithm 2.4) the storage system with block size $l[n-1] = p_{n-1}(q)$, then expanding to a storage system with block size $l[n] = p_n(q)$ can be accomplished by appending the remaining outgoing edges from each Y vertex. No elements need to be moved from the existing system, and yet the Steiner property (of no repeating pairs of elements) will still hold—one need only append new elements to the appropriate blocks. For instance, the expansion of the system of Figure 2.9(b) results in the appended storage system of Figure 2.14.

It is similarly simple to construct a storage system that has total number of elements, $\tilde{u}$, that is between the valid quantities $u[n-1]$ and $u[n]$ (i.e., $u[n-1] < \tilde{u} < u[n]$). One should construct the system for $u[n]$ number of elements (i.e., $k = q+1$ and $l[n] = p_n(q)$) and then leave empty slots in the blocks that are supposed to store the elements $x_{\tilde{u}}, x_{\tilde{u}+1}, x_{\tilde{u}+2}, \ldots, x_{u[n]-1}$. This will preserve the Steiner property and also allow expansion of the storage system until $u[n]$ elements arrive.

Although we have discussed only the scalability of the $k = q+1$, $l = p_n(q)$ storage system constructed from Algorithm 2.4, in fact the $k = q$, $l = p_n(q)$ storage system

of Algorithm 2.5 also scales just as easily. Although we do not provide the proofs here, we comment that because Algorithm 2.5 uses the exact same initial condition as Algorithm 2.4, therefore an analog to Theorem 2.13 (with the appropriate parameters) still holds true by appealing again to Lemma 2.14.

2.6 Simulation Results

Although it is clear that there are theoretical improvements in repair speed due to performing parallel disk reads, we also want to empirically characterize the potential benefits for disk arrays in production—that is, disk arrays that are also undergoing some other workload in addition to disk repair. Toward that end, we have created an implementation of the scalable data chunk distribution method in the RAIDframe disk array simulator [23]. For any given disk array, data chunks and their replicas can be placed within the array according to the scalable cage graph algorithms given in Algorithms 2.4 and 2.5.

As mentioned previously, by distributing data chunks according to the Steiner system design, in the worst case under multiple disk failures, less data will be lost compared to schemes where data chunks that are co-located on one disk might also be co-located on some other disk. Here we want to compare the fraction of data loss in the system when using disk mirroring (i.e., using some variant of RAID 1) versus when using a Steiner system design.

We tested our scheme on an array of 1023 Seagate Barracuda 160 GB disks that each have an average seek time of 11.0 ms for reads and 12.0 ms for writes. We use triple redundancy (i.e., replication degree of 3); thus according to Table 2.5, the Steiner system design for the 1023-disk array has each disk being divided into 511 chunks.[17] For disk mirroring with triple redundancy, every disk is replicated 3 times, so in the entire disk array there is in fact only 341 disks' worth of data being stored. The disk array stores a total of 52 TB of distinct data in either configuration.

The disk array was run for 24 hours in simulation, where regular disk accesses are

[17]Note that this means that each data chunk has about 300 MB.

made according to the read and write operations of an e-mail and research workload server. More specifically, the disk array is undergoing the reads and writes of the workload taken from a single day of the DEAS03 trace [32] (which can be downloaded as the `deasna2` trace from [79]). On top of this read and write workload, disk failures within the storage array are introduced in the simulation, where disk failure occurs according to Murphy's Law [9]. That is, whenever a disk is supposed to fail, a disk is chosen to fail such that the maximum possible data loss for the given data distribution scheme occurs. Reconstruction of failed disks is then performed to recover any data for which some of the replicas are still available.

Figure 2.15 shows the fraction of data lost in the storage system, compared between the two data chunk distribution schemes. Here, we vary the disk mean-time-between-failures (MTBFs) of the disks,[18] in order to see the system performance when disks are less reliable (i.e., with low MTBF).

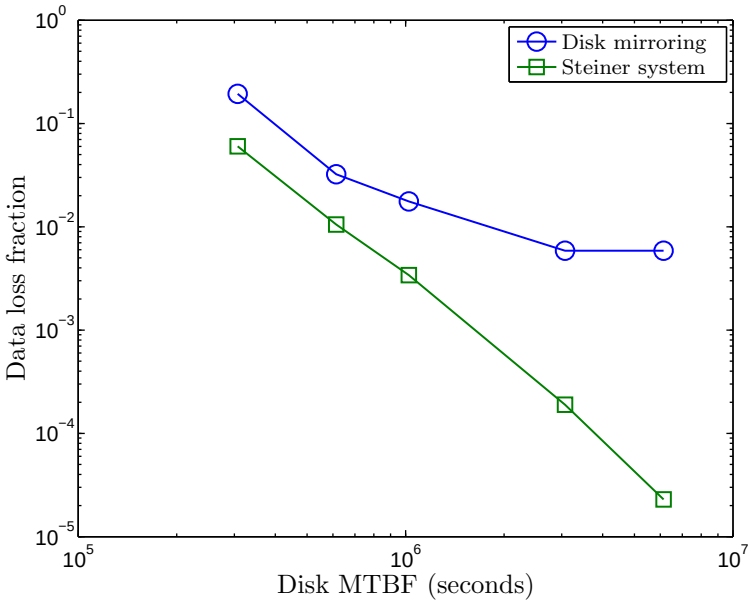


Figure 2.15: Fraction of data loss comparisons between disk mirroring and Steiner system chunk distribution.

[18]The time between potential disk failures is governed by an exponential distribution with the particular disk MTBF.

From the simulation results, we see that by using the Steiner system chunk distribution scheme, there is significantly less data loss as compared to disk mirroring. This is due to two advantages of the Steiner system design: first, the Steiner system design has faster reconstruction times since all reconstruction reads can always be performed in parallel; and second, with multiple disk failures, by not having co-located replicas potential data losses from any additional disk failure is at most one chunk—whereas in any other scheme the potential data loss is related to the number of chunk replicas that may be co-located. Thus, these advantages will also be evident when compared against any chunk distribution scheme with co-located replicas, such as most random chunk distribution schemes—even if such schemes have replicas that are more spread out than in disk mirroring.

Although the simulation times are short and the disks quite unreliable, it is clear that similar trends for data loss would hold even as the parameters are increased to more realistic scenarios (i.e., longer simulation times coupled with larger disk MTBFs).

2.7 Conclusion

In this chapter, we give practical, scalable, and implementable constructions of bipartite cage graphs where the vertex degrees are highly asymmetric. This allows for the design of distributed storage systems based on Steiner systems, where the number of replicas of each data chunk may be much smaller than the storage node size. Using our constructions, a system designer can guarantee that a system consuming the least amount of resources (e.g., fewest number of storage nodes) has been deployed, and also be able to easily expand the storage system when necessary.

By designing our storage system using Steiner systems, we can guarantee that the distributed storage system is robust to node failure, and that node replacement is efficient. Any failed node can have its data replaced by accessing as many nodes as data chunks stored at the node, so that repair is highly parallelized—allowing for short downtimes of both the failed node as well as any nodes contacted for repair

data. Determining which nodes to contact for replacement data can be achieved by searching the bipartite graph for connections to the requested data chunks.

We further comment that the chunk distribution schemes given by our cage graph construction method can also be used to guarantee collision resistance in existing storage system implementations. As an example, for storage systems implementing distributed hash tables, when the desired replication degree and number of storage nodes are known, then the chunk and replica locations from the appropriate block design may be used as the hashing function.

Chapter 3

Low-Complexity Non-Uniform Demand Network Coding

3.1 Introduction

Network coding is a technique that has been shown to enable higher transmission rates across communication networks, when compared against routing methods. This occurs because network coding allows for the sharing of links by data that is flowing toward different sinks, and—through appropriate coding of symbols—have the sinks still be able to decode out these disparate flows. In the butterfly network example first proposed by Ahlswede et al. [1] (see Figure 1.2), if the input data at node w are coded together, it is possible to multicast two streams of information b_1 and b_2 from the source s to both sinks t_1 and t_2 within a single time period. The benefits of allowing coding at nodes are evident; under routing, multiple time periods would be required to send both streams to both sinks. The authors show that in any network with a single source and multiple sinks, the information rate can achieve the minimum (over all sinks) of the maximum flow to the sink nodes. In subsequent work, Li et al. [52] prove that linear network codes are sufficient for multicast, and Jaggi et al. [44] give a polynomial-time algorithm for constructing such linear codes.

Following the quick successes of characterizing and developing algorithms for multicast network coding problems, there has been much work concerning the construction of network codes for more general scenarios—although this has proven to be a much more difficult problem. Koetter and Médard [49] give an algebraic characterization for achievable linear network codes, but prove that checking for the existence of such codes requires running time that is not polynomially bounded. Then, Rasala Lehman and Lehman [69] prove that for most network coding scenarios, finding linear network codes to satisfy arbitrary source and demand requirements is NP-hard. Of relevance to the current work is the problem of constructing network codes to send data from a single source to multiple sinks with arbitrary demands (potentially with different demands by different sinks).

In this work, we study networks where the single source may send data to multiple sinks at unequal rates. The motivation for this can be seen in the extended butterfly network of Figure 3.1. Here, the traditional butterfly network is augmented with an additional path between the source s and sink t_2. Within a single time period, at most two streams can be transmitted to sink t_1, but it is possible to transmit more than two streams to sink t_2. If sink t_2 is constrained to only receive two streams, then available capacity is wasted.

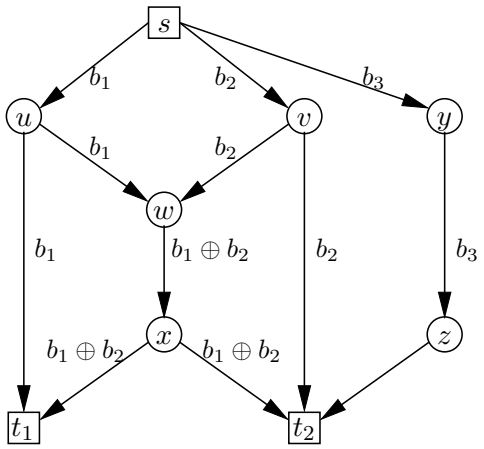


Figure 3.1: The extended butterfly network, which has an additional path to sink t_2 that can be utilized for sending additional data. Using network coding and taking advantage of the extra path through nodes y and z, source s can transmit three streams (b_1, b_2, and b_3) to sink t_2 simultaneously with two streams (b_1 and b_2) to sink t_1.

The problem of sending unequal-rate data from a single source to multiple sinks has two flavors: the multiple multicast connections problem [54] and the non-uniform demand problem [16]. In the multiple multicast connections problem, the sinks are allowed to receive data at different rates, but the subset of information demanded by each particular sink—while arbitrary—is identified in advance. On the other hand, in the non-uniform demand problem, the amount of information a particular sink must be able to receive is specified in advance, but it does not matter which specific pieces of information are received. This is a scenario where the source has a large set of messages that it wishes to send to the sinks, and each particular sink wishes to receive a subset of the source's messages; however, the requirement at each sink is only that the messages it receives is a subset of a particular size rather than a requirement of receiving some specific subset of messages. This problem can be understood as a relaxed version of the multiple multicast connections network coding problem, since if it is known that a sink is unable to receive a subset of size n, then any specific demand for a subset of size n can automatically be rejected as impossible. Conversely, if it is known that a sink is able to receive some subset of size n, then it may be possible to find a network coding solution with a specific demand for that sink that is of size n. The non-uniform demand problem makes it possible to determine the maximum possible data rates that can be received by the sinks.

The non-uniform demand problem was originally investigated by Cassuto and Bruck [16]. For demand scenarios where all sinks require the same high rate except for two sinks demanding some lower rate, the authors prove that it is possible to satisfy these demands in all cases, and that linear codes are sufficient. They also give limited conditions for the achievability of non-uniform demands when there are more than two lower-demand rate sinks along with any number of higher rate equal-rate sinks. In our work, we more specifically consider the case of non-uniform demands where each [possibly heterogeneous] sink demands a data rate of its own maximum-flow (i.e., maximum point-to-point rate from the source). From this seemingly more restrictive setting, however, we are able to describe a larger class of networks for which the non-uniform demand problem is solvable (and solvable in polynomial time), thus

enabling non-uniform demand network coding to be more widely applicable.

Although the class of network demand scenarios for which we give polynomial-time solutions is not exhaustive, it may be difficult to enumerate the conditions more generally. This is because the non-uniform demand network coding problem is NP-hard [16]. We give an alternate proof of the NP-hardness of the non-uniform demand network coding problem (see Appendix C), which proves this result for slightly different demand scenarios than those addressed in [16].

3.1.1 Related Work

A technique that we use is that of transmitting data along paths, or through flows. This approach has been widely used in the network coding literature, and has enabled many significant results. In Jaggi et al. [44], the polynomial-time algorithms for multicast problems rely on the concept of sending data down [perhaps overlapping] paths. In [35], Fragouli and Soljanin give a decomposition of networks into flows, in order to model data transmission in a network more simply. Using this decomposition and a graph coloring formulation, alphabet size bounds for any network code are then proven. Although the flow-based and path-based approaches are similar in many ways, the two techniques differ in that the flow-based approach creates a new flow every time a piece of data is transformed by coding, whereas the path-based approach keeps track of each piece of data as it is sent individually down a path, even if any transformations get applied to the data. We shall use a path-based approach.

The path-based approach is also used in a few papers that consider network coding for the multiple unicast problem. The multiple unicast scenario is where there are multiple sources and multiple sinks, but sources and sinks only connect pairwise and each pair is transmitting data at the same rate. The multiple unicast scenario is beyond the scope of this paper, but it is worthwhile to consider how path-based approaches can be used. In [51], the authors use a path-based approach for wireless scenarios, and give conditions for stream decodability at sinks—although these conditions do not allow for contamination from upstream overlaps (i.e., when contamination from one path to a second path subsequently spills over onto a third path).

However, the authors were able to achieve good experimental results, showing significantly increased throughput using their network coding protocol. In another work, [84] uses a path-based approach to completely characterize the set of all multiple unicast situations with two sources and two sinks for which network coding solutions can be found, by looking at all possible ways in which the two source-sink paths can overlap. This is similar to the strategy taken in our paper, except that we consider more than two sinks, but with only a single source.

We briefly mention some results regarding the multiple multicast connections problem, since achievable solutions for such problems are also achievable for the non-uniform demand problem with the same demanded rates. (Of course, the reverse is not always true.) Thus, an understanding of the multiple connections multicast problem can give us insight into the non-uniform demand problem, by showing some possible characterizations of solutions. Many of these results consider the case of two sinks. In [63], after enumeration of all possible scenarios, the authors conclude that in the case of two sinks with differing rates, linear coding is sufficient. The same conclusion is made in [68], although the authors use a different approach that considers a path-based enumeration. A characterization of the achievable data rate region using network coding is given for the two sink case. For more than two sinks, conditions under which solutions exist for the multiple connections problem have not been enumerated.

Separately from the multiple multicast connections problem, the non-uniform demand problem itself has also been the subject of study. In [16], Cassuto and Bruck introduce the problem and give some results concerning the achievability of the individual max-flow rates to each sink. In our work, we address similar guarantees but for a wider class of network demands. The authors in [16] also prove that the non-uniform demand problem is NP-hard, using a reduction from a 3-CNF problem, although the demand problems for which their result holds have network coding solutions that do not fully utilize the available data rates (in our terminology, the solutions are not saturating). We supplement their proof by considering whether or not networks in which the network coding solutions use all possible paths are still difficult to solve.

We take a similar approach by considering contamination amongst data; however, we do not allow intermediate decoding as they do. The non-uniform demand problem is also studied in [18], which considers the case where demands are allowed to be relaxed in the solution. For general networks, the authors give bounds on the fraction of max-flow rate that is achievable, and show networks for which the bounds are tight. We take a different approach and instead characterize specific network demand scenarios for which the max-flow rate can be achieved.

3.1.2 Outline of Chapter

In this chapter, we will investigate the case of non-uniform demand network coding in which each sink receives data at its individual point-to-point (i.e., max-flow min-cut) capacity rate. In Section 3.2, we discuss useful notation. In Section 3.3, we define our approach to the problem, and analyze some of its characteristics. Following that, in Section 3.4, we give an algorithm for determining if a non-uniform demand solution exists, and discuss some of its performance issues. Using this algorithm, we characterize in Section 3.5 a class of networks for which the non-uniform demand network coding problem can be solved in polynomial time. Of course, not all non-uniform demand network coding problems can be solved efficiently; in Appendix C, we give an alternate proof of the NP-completeness of the non-uniform demand problem that accounts for the demand scenarios we are considering.

3.2 Notation and Definitions

We will consider a directed acyclic network graph $G = (V, E)$, where each edge has unit capacity. Each sink will be indexed as $j \in \{1, 2, \ldots, t\}$, where t is the total number of sink nodes. (Recall that there is only a single source node s.) Because the graph is acyclic, there exists a partial ordering of the nodes starting from the source s. A partial ordering of the edges can be constructed based on the ordering of the nodes from which the edges originate; for edges $e = (v, w)$ and $e' = (v', w')$, we denote $e \preceq e'$ if and only if $v \preceq v'$ in the partial ordering of nodes.

For a particular sink j, we define $\mathcal{P}_j$ as a set of paths associated with sink j. These are unit-capacity edge-disjoint paths from the source s to sink j and can be determined from maximum-flow algorithms such as the Ford-Fulkerson augmenting path algorithm [34]. Paths $p \in \mathcal{P}_j$ are given as the elements of $\mathcal{P}_j = \{p_{j,1}, p_{j,2}, \ldots, p_{j,n_j}\}$, where $n_j = |\mathcal{P}_j|$ is the number of paths to sink j. Call the set of all paths $\mathcal{P} = \bigcup_{j=1}^{t} \mathcal{P}_j$. In contrast to much of the literature on network coding for multicast, we will consider the *maximum* of the max-flows (instead of the minimum of the max-flows) and denote this quantity n, so $n = \max_j |\mathcal{P}_j| = \max_j n_j$.

In order to keep track of data that overlaps onto paths with other sinks, we introduce the concepts of contamination and contaminating set. We say that *contamination* from path p_{jk} onto path $p_{j'k'}$ occurs when data transmitted on p_{jk} gets combined into the data that is supposed to be transmitted on $p_{j'k'}$. This can occur, for example, when data on paths p_{jk} and $p_{j'k'}$ are coded together in order to be transmitted across an edge where the two paths overlap. Then the *contamination set* of p_{jk} is the set of all paths that experience contamination due to data from p_{jk}. If we call $\mathcal{D}_{jk}(e)$ as the set of paths that are contaminated by path p_{jk} downstream of edge e, then $\mathcal{D}_{jk}(e)$ can be defined recursively as follows:

$$\mathcal{D}_{jk}(e) = \bigcup_{p_{j'k'}} \left\{ p_{j'k'} \cup \left(\bigcup_{e' \succ e} \mathcal{D}_{j'k'}(e') \right) \right\},$$

where the union over $p_{j'k'}$ is over all paths $p_{j'k'}$ that overlap path p_{jk} at edge e (and $j' \neq j$). Then $\mathcal{D}_{jk} = \bigcup_{e \in E} \mathcal{D}_{jk}(e)$ gives the contamination set of p_{jk}. This definition accounts for a data stream on a path to contaminate onto paths that it does not explicitly overlap, due to contamination being spread from path to overlapping path.

As an example, we shall consider the construction of the contamination set $\mathcal{D}_{1,2}$, using the path decomposition of the extended butterfly network given in Figure 3.2. Note that $\mathcal{D}_{2,2}(e') = \emptyset$ for all $e' \succ (s, v)$ because path $p_{2,2}$ has no overlaps downstream of edge (s, v) with any other path, and $\mathcal{D}_{2,1}((x, t_2)) = \emptyset$ because no other paths

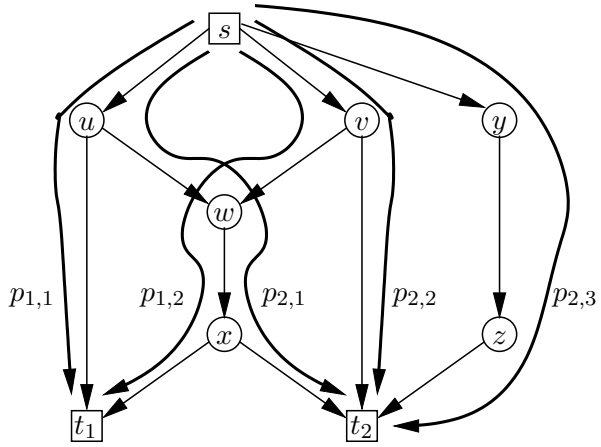


Figure 3.2: Path decomposition for extended butterfly network.

overlap $p_{2,1}$ at the terminating edge (x, t_2). Then we see that

$$\begin{aligned}
\mathcal{D}_{1,2}((s,v)) &= \{p_{2,2}\} \cup \left(\bigcup_{e' \succ (s,v)} \mathcal{D}_{2,2}(e')\right) = \{p_{2,2}\} \\
\mathcal{D}_{1,2}((v,w)) &= \emptyset \\
\mathcal{D}_{1,2}((w,x)) &= \{p_{2,1}\} \cup \mathcal{D}_{2,1}((x,t_2)) = \{p_{2,1}\} \\
\mathcal{D}_{1,2}((x,t_1)) &= \emptyset.
\end{aligned}$$

Therefore, $\mathcal{D}_{1,2} = \bigcup_{e \in E} \mathcal{D}_{1,2}(e) = \{p_{2,1}\} \cup \{p_{2,2}\} = \{p_{2,1}, p_{2,2}\}$.

We also wish to keep track of the particular data streams sent to each of the sinks. A stream is defined as the identifier of the data that is being transmitted down a particular path, as opposed to the identifier of the path itself. We identify a particular stream with the index $i \in \{1, 2, \ldots, n\}$, which means that a path is allowed to transmit a stream only from the set $\{1, 2, \ldots, n\}$. Each stream represents one information unit, of which only n unique information units are allowed to be transmitted. This restriction is not prohibitive, since n is the maximum flow. Each sink receives a subset of the same n streams, so different sinks will likely receive many of the same streams. The number of distinct streams a particular sink receives is its data rate, since each unit-capacity edge-disjoint path can transmit at most only a single data stream.

We also define decodable and saturating solutions.

Definition 3.1. *A* decodable solution *to a network coding problem is one in which every sink is able to decode all of the information that is intended to be sent to it. In the example of streams assigned to paths, a decodable solution is one in which every sink can recover all of the streams that are assigned on paths to that sink.*

Definition 3.2. *A* saturating solution *is an assignment* $f : \mathcal{P} \to \{1, 2, \ldots, n\}$ *from paths* $p_{jk} \in \mathcal{P}$ *to streams* $\{1, 2, \ldots, n\}$*, such that for each* $j \in \{1, 2, \ldots, t\}$*,*

$$f(p_{jk}) \neq \emptyset, \quad k \in \{1, 2, \ldots, n_j\}$$

and

$$f(p_{jk}) \neq f(p_{jk'}), \quad k \neq k'.$$

That is, a saturating stream assignment is a stream assignment in which all paths to every sink are assigned some data stream; no path is left unassigned. Moreover, any streams assigned to different paths to the same sink must be distinct. Otherwise, if two paths carried the same stream, one of the paths is redundant and does not carry additional information. Thus, a saturating stream assignment is one in which each sink j achieves its maximum possible data rate of n_j.

We briefly mention the concept of intermediate decoding. Specifically, for the network codes we are considering, we do not allow intermediates nodes (i.e., nodes that are neither source nor sink) to decode data and retransmit only a part of the data on its outgoing links. In other words, intermediate nodes are not allowed to remove any contamination that it might receive on its incoming links, even if it possesses enough information to decode out the contamination. Although this condition may prevent certain network codes from being considered, it is still general enough that except for certain cases, we should be able to find the appropriate network coding solution if it exists.

Definition 3.3. *The* non-uniform demand problem *is the following solvability problem (adapted from [16]): Given a directed acyclic network graph* $G = (V, E)$ *(where*

each edge has capacity 1*), source* s*, sinks* $j \in \{1, 2, \ldots, t\}$*, and demand function* $d : \{1, 2, \ldots, t\} \to \{1, 2, \ldots, n\}$ *(where* $d(j)$ *is the demanded rate of sink* j*), is there a network coding solution such that for all* j*, sink* j *receives information at a rate* $d(j)$*?*

Note that for the case when saturating solutions are required, then $d(j) = n_j$ for all j.

3.3 The Non-Uniform Demand Stream Assignment Problem

The goal is to determine whether or not an assignment of data streams to paths can give a decodable network solution.

Definition 3.4. *The* non-uniform demand stream assignment problem *is the following: Given a network graph* $G = (V, E)$ *and a decomposition into paths, is there an assignment of streams to paths, such that no intermediate decoding occurs, and the solution is both saturating and decodable?*

We establish necessary and sufficient conditions for a network to have a saturating and decodable solution.

Theorem 3.1. *Given a set of paths between the source and the sinks, if no intermediate decoding is allowed, then there exists a saturating and decodable solution if and only if all streams that contaminate onto paths to a particular sink have also been assigned to some other path to the same sink.*

Proof. Necessity follows from the fact that if the solution is decodable, then each sink can separate out all streams and all contamination sent to it. For a particular sink, each path carries an assigned stream mixed with the contamination from along that path. Because no intermediate decoding is allowed, all contamination arrives at the sink, but arrives mixed in with the assigned streams on the respective paths. If there are n_j paths to the sink j and the solution is saturating, then there are n_j unique streams assigned on paths to the sink. Now, if a contaminant is not also assigned

to some other path to that sink, then that means that there is data from at least $n_j + 1$ streams on inputs to the sink (the assigned n_j streams of data plus at least one more data stream from the contamination). However, because there are only n_j paths from the source to that sink, where each path supports a data rate of 1, that means that no more than n_j unique data streams can be received by the sink (or else the max-flow condition would be violated). Thus, any situation where more than n_j data streams (perhaps mixed) can be seen by the sink is a situation where fewer than n_j data streams can be decoded successfully by the sink, and the solution is either not decodable or not saturating.

For each sink, it is straightforward to show sufficiency of assigning the contamination onto a path to that sink as the primary stream on another of its paths, in order to guarantee decodability. If no intermediate decoding is allowed, all contamination arrives at the sink. If no other path has been assigned the same data stream, then there is no way to determine (or even have partial knowledge) of the data due to the contamination in order to either utilize or remove it. Thus, there must be an assigned stream on some other path to that sink that provides this information. □

The concept of saturation is important, so that it is possible to state (using max-flow theorems) that contamination without a corresponding assigned stream cannot be removed, as there will not be enough flow to support this additional data. Saturation is also useful because it enables us to determine whether or not the maximum data rate actually utilizes all paths, without repetitive data streams. In fact, if data is repeated (i.e., multiple paths to the same sink are assigned the same stream), then one of the multiple paths could be shut off with no harm to the data rate toward that sink, and possibly even increasing data rates across the entire network due to less contamination onto paths to other sinks.

If the sufficient conditions of Theorem 3.1 are met, then a linear network code, that is also saturating and decodable, can be constructed using the algebraic approach of [49]. Or, by using the path-based interpretation for the network, an appropriate linear network code for the network can be efficiently constructed using the deterministic method of [44] or the random linear network coding method of [41].

3.4 An Algorithm for Assigning Streams

We now give an algorithm for solving the non-uniform demand stream assignment problem. Again, we restrict ourselves to the case where intermediate decoding does not occur within the network; that is, even though nodes within the network may encode information, there is no preliminary decoding (and removal of streams) except at the sink nodes that are receiving information. Equivalently, the output of any node is at least as contaminated (but not less contaminated) than any of the inputs to that node. Admittedly, precluding the removal of streams within the network does limit the solution space. However, since the goal is to maximize the data rate to each sink, solutions that do remove streams must be able to compensate for the loss of data rate due to the removal of streams with some other benefit, such as less contamination further down the network.

Our goal is to be able to assign streams to paths—with guarantees that the contamination will be decodable—while maximizing the number of streams transmitted. We give a method that guarantees saturation and decodability, by utilizing a polynomial transformation to graph coloring. We first give the transformation of the network graph into a coloring graph in Algorithm 3.1, and then give the algorithm for finding the saturating and decodable solution in Algorithm 3.2.

The coloring graph $\hat{G}$ can be interpreted with respect to the solution found by Algorithm 3.2. The vertices v_{jk}, $k = 1, \ldots, n_j$, correspond to paths in the original network, and so are called regular vertices. The additional vertices w_{jk}, where $k = 1, \ldots, n - n_j$, correspond to fictitious paths, indicating the streams that are *not* assigned to paths leading to sink j—hence the name fictitious vertices. The edges in $\hat{E}_j^{\text{complete}}$ form a complete subgraph among all n_j regular vertices associated with sink j, guaranteeing saturation. (In fact, the entire induced subgraph of $\hat{V}_j$ is a clique.) For $\hat{E}_j^{\text{overlaps}}$, the edges $(v_{jk}, w_{j'k'})$ connect the vertices from sink j to the vertices of sink j' if there is some overlap on paths toward these two sinks; these edges force a relationship between the streams assigned on paths to one sink and streams assigned on paths to other sinks, providing decodability.

Because $n = \max_j n_j$, there must exist at least one clique of size n (associated with

Algorithm 3.1 Transformation of network graph into coloring graph

Require: Original network graph $G = (V, E)$, decomposed into edge-disjoint paths $\mathcal{P}_j = \{p_{jk} \,|\, k = 1, \ldots, n_j\}$ for each sink j

1: Create vertices $v_{jk} \in \hat{V}$, for $k = 1, \ldots, n_j$, which are associated with paths $p_{jk} \in \mathcal{P}_j$. For each j, introduce $n - n_j$ additional vertices $w_{jk} \in \hat{V}$. (v_{jk} are *regular vertices* and w_{jk} are *fictitious vertices.*) Let

$$\hat{V}_j = \{v_{jk} \,|\, 1 \le k \le n_j\} \cup \{w_{jk} \,|\, 1 \le k \le n - n_j\}.$$

Call $\hat{V}_j$ the *sink subgraph* associated with sink j. Then $\hat{V} = \bigcup_{j=1}^{t} \hat{V}_j$.

2: For each j, connect all vertices v_{jk} and vertices w_{jk} together into a clique. Specifically, for each j, let

$$\begin{aligned}\hat{E}_j^{\text{complete}} = &\{(v_{jk}, v_{jk'}) \,|\, 1 \le k < k' \le n_j\} \cup \\ &\{(w_{jk}, w_{jk'}) \,|\, 1 \le k < k' \le n - n_j\} \cup \\ &\{(v_{jk}, w_{jk'}) \,|\, 1 \le k \le n_j, 1 \le k' \le n - n_j\}.\end{aligned}$$

3: For each vertex v_{jk}, connect v_{jk} to vertices $w_{j'k'}$ for all $k' = 1, \ldots, n - n_{j'}$, if path p_{jk} contaminates onto some path to sink $j' \neq j$. For each j, call

$$\hat{E}_j^{\text{overlaps}} = \bigcup_{k=1}^{n_j} \{(v_{jk}, w_{j'k'}) \,|\, 1 \le k' \le n - n_{j'} \text{ if } \exists \tilde{k} \text{ s.t. } p_{j'\tilde{k}} \in \mathcal{D}_{jk}\}.$$

That is, if path p_{jk} to sink j contaminates onto some path $p_{j'\tilde{k}}$ to sink j', then node v_{jk} must connect to all $n - n_{j'}$ fictitious vertices associated with sink j'.

4: Let $\hat{E} = \bigcup_{j=1}^{t} (\hat{E}_j^{\text{complete}} \cup \hat{E}_j^{\text{overlaps}})$.

5: **return** coloring graph $\hat{G} = (\hat{V}, \hat{E})$

the induced subgraph of $\hat{V}_{j^\star}$, where $j^\star = \arg\max_j n_j$). Thus, $\hat{G}$ cannot be colored with fewer than n colors, so step 3 of Algorithm 3.2 is equivalent to coloring $\hat{G}$ with *at most* n colors. If the minimum coloring solution of $\hat{G}$ requires more than n colors, then the following lemma tells us that decodability has been violated.

Lemma 3.2. *In the equivalent coloring graph constructed from Algorithm 3.1, if the chromatic number $\chi(\hat{G}) > n$, then the original network is not decodable.*

Proof. If $\chi(\hat{G}) > n$ and every vertex is a member of at least one induced clique of size n, then there must exist two cliques of size n such that there is at least one edge

Algorithm 3.2 Non-uniform demand stream assignment

Require: Directed acyclic network graph $G = (V, E)$, source s, and sinks $j \in \{1, 2, \ldots, t\}$

1: For each sink j, find a set of edge-disjoint paths from s to j. Call the set of these paths $\mathcal{P}_j = \{p_{jk} \mid k = 1, \ldots, n_j\}$, where there are n_j such paths. Let $n = \max_j n_j$.
2: Using Algorithm 3.1, construct coloring graph $\hat{G}$ from $\mathcal{P} = \bigcup_{j=1}^{t} \mathcal{P}_j$.
3: Color $\hat{G}$ using exactly n colors. Let c_{jk} be the color of v_{jk} in $\hat{G}$.
4: For each path $p_{jk} \in \mathcal{P}$, assign stream c_{jk} to that path. (Each path $p_{jk} \in \mathcal{P}$ in the network graph G is assigned the stream given by the color of its associated vertex $v_{jk} \in \hat{V}$ in the coloring graph $\hat{G}$.)
5: In the network graph, at each node where the inputs to the node are different streams, send as the output of the node a combination of the data from the input streams (e.g., linear combination, or some other combining method)—taking care that no input streams are nullified in the node output.

connecting these two cliques. Moreover, because the coloring graph is colored with more than n colors, some pair (j, j') of connected cliques (each clique associated with a different sink) must satisfy the following condition: If clique j does not have color c within it, then its fictitious vertices must be connected to a regular vertex with color c in clique j'. In this case, sink j cannot decode color c even though some path to j has contamination c from a path to j', and so decodability is violated. □

Theorem 3.3. *Algorithm 3.2 succeeds in coloring the equivalent coloring graph with exactly n colors if and only if the original network graph has a decodable and saturating solution with no intermediate decoding.*

Proof. It is clear that $\chi(\hat{G}) = n$ is necessary for the solution to be decodable and saturating. From Lemma 3.2, we know that if the network is decodable, then the equivalent coloring graph must have $\chi(\hat{G}) \leq n$. Now consider sink $j^\star$, where $j^\star = \arg\max_j n_j$. If the network is saturating, then sink $j^\star$ must be able to receive $n = n_{j^\star}$ distinct streams. That is, the clique associated with sink $j^\star$ must be colored with at least n colors. This gives us $\chi(\hat{G}) \geq n$. Thus, $\chi(\hat{G}) = n$.

Next we prove sufficiency of $\chi(\hat{G}) = n$ for a decodable and saturating solution. For the coloring to be valid, if path p_{jk} contaminates *any* path $p_{j'\tilde{k}}$ to sink j', then by construction of $\hat{E}^{\text{overlaps}}$, none of the fictitious vertices $w_{j'k'}$ associated with sink j'

may have the same color as vertex v_{jk} (call this color c_{jk}). Because $\chi(\hat{G}) = n$ and each sink subgraph is an induced clique of size n, some regular vertex $v_{j'k'}$ to sink j' must be colored c_{jk}. Equivalently, contamination due to path p_{jk} onto path $p_{j'\tilde{k}}$ has been assigned to some path $p_{j'k'}$ to sink j' (and this is true for all possible contaminations), so by Theorem 3.1 the solution is decodable. Moreover, if $\chi(\hat{G}) = n$, then by construction of $\hat{E}^{\text{complete}}$, all paths have assigned streams, and the assigned streams are distinct for different paths to the same sink. Thus, the solution is saturating. □

Theorem 3.3 gives necessary and sufficient conditions for a saturating and decodable solution to exist. Moreover, the stream assignment algorithm tells us how to allocate streams in order to construct this solution.

Example 3.1 (Extended Butterfly Network)**.** *Figures 3.3 and 3.4 show the result of Algorithm 3.2 on the extended butterfly network.*

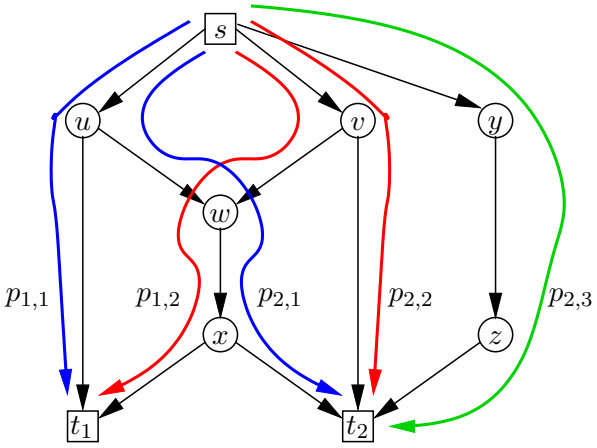


Figure 3.3: Stream assignment for extended butterfly network. The color of each path indicates the stream that should be assigned to that path.

3.4.1 Shortcomings

Assuming that the correct set of edge-disjoint paths are chosen in the first step of the algorithm, then if the solution exists it will be found. However, we do not address the proper selection of edge-disjoint paths, even though there may be multiple path

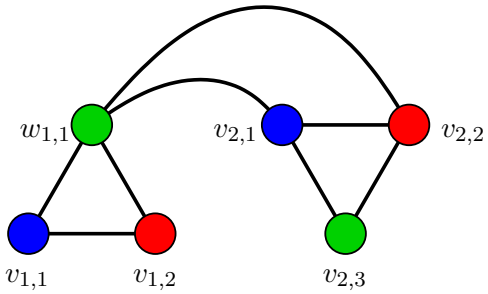


Figure 3.4: Equivalent coloring graph for extended butterfly network. The graph can be colored with 3 colors, so it gives a saturating and decodable solution to the original network. The colors correspond to the streams that should be assigned to the paths.

decompositions, where some decompositions lead to sub-optimal assignments. For example, it is possible to construct a counterexample network with a given path decomposition, where switching a single edge for one path greatly increases allowed throughput.

The algorithm determines, for a given set of paths, whether or not n streams can be assigned. But to determine if some $\bar{n} > n$ streams can be assigned, additional fictitious vertices need to be introduced. For each sink, an additional $\bar{n} - n$ fictitious vertices must be introduced in order to get sink subgraphs of size $\bar{n}$. The relationship between these larger graphs and the original coloring graph is unknown, and it is possible that no matter how large $\bar{n}$ is chosen, it will still be impossible to find a saturating and decodable assignment with $\bar{n}$ streams. If one wishes to determine how many additional available streams will guarantee saturation and decodability, $\bar{n}$ could be increased without bound while searching for a possible stream assignment.

In fact, it is possible to have networks where there does not exist a saturating solution that is also decodable, no matter how large the available stream set is. This can occur when there is too much overlap but not enough available paths to remove the contamination. For example, consider a two sink case, where $n_1 = 1$ and $n_2 = 2$. Suppose both paths of sink 2 overlap the path to sink 1 at some link[s]. No matter what the stream assignment (or how large the space of possible streams) for sink 1, it will never be able to decode out both contaminants if saturation for both sinks is required.

Thus, our algorithm works only for the restrictive case where no intermediate decoding is allowed, yet all paths to sinks must be saturating. Loosening either the saturation or the no intermediate decoding restrictions would be beneficial, but at the moment, the algorithm relies on both conditions.

Another issue to keep in mind is that because graph coloring is an NP-complete problem [37], there are no known polynomial time algorithms that will perform the coloring step (unless P = NP). Additionally, there are no good approximation algorithms known for the graph coloring problem (see [48] for algorithms that can color n-colorable graphs with number of colors logarithmic in the number of vertices of the graph, but with no guarantees based on the actual chromatic number n). Even if there were good approximation algorithms, an approximation algorithm might not be enough to answer the question of whether or not a saturating solution exists (i.e., whether or not $\chi(\hat{G}) = n$) since we require finding the chromatic number exactly. One might conjecture that because the coloring graph $\hat{G}$ is carefully constructed, it might have some special structure that could allow for a polynomial-time coloring algorithm. In the next section, we give some structural properties of the coloring graph that can lead to a polynomial-time coloring, but we also show a counterexample network where this particular structural analysis is not sufficient to prove polynomial-time solvability.

3.5 Efficiently Solvable Non-Uniform Demand Problems

The coloring step in the stream assignment problem is problematic, as the graph coloring problem is NP-complete and so in general no known polynomial-time algorithm can solve the problem. However, if we restrict our class of demand problems to only those for which the corresponding coloring graph is polynomial-time solvable, then such demand problems will also be polynomial-time solvable. Specifically, we consider the class of graphs known as Berge graphs, which are graphs characterized by the absences of both odd holes (induced cycles of odd length at least 5) and odd antiholes (complements of odd holes). Then by the strong perfect graph theorem [20],

a Berge graph is also a perfect graph, so the chromatic number of a Berge graph is equal to the size of its maximum clique.

This fact is useful in finding solutions to the non-uniform demand problem because in our formulation, the maximum clique is easily found, so if the coloring graph is Berge, then the chromatic number is also readily determined. The main result of this section is that we can find the maximum clique for the coloring graph of Algorithm 3.2 in polynomial time and hence also its chromatic number if the coloring graph is Berge. We first prove some preliminary results.

Lemma 3.4. *Any induced clique consisting of vertices from different sink subgraphs can only consist of vertices from at most two sink subgraphs. That is, it is impossible to induce a complete subgraph consisting of at least one vertex from each of $\hat{V}_j$, $\hat{V}_{j'}$, and $\hat{V}_{j''}$.*

Proof. For vertices belonging to different sink subgraphs, i.e., $v \in \hat{V}_j$ and $v' \in \hat{V}_{j'}$ where $j \neq j'$, either v is regular and v' is fictitious, or v is fictitious and v' is regular. Regular vertices are not connected to regular vertices, nor are fictitious vertices connected to fictitious vertices, unless they belong to the same sink subgraph. Any complete subgraph consisting of at least one vertex from each of $\hat{V}_j$, $\hat{V}_{j'}$, and $\hat{V}_{j''}$ must contain at least two vertices of the same type from different sink subgraph (e.g., two regular vertices and one fictitious vertex, where each vertex is from a different sink subgraph). However, such a scenario cannot exist, as that implies that two vertices of the same type but from different sink subgraphs are connected. □

The preceding lemma tells us that in order to find the maximum clique in the coloring graph, it is sufficient to search for induced cliques pairwise between sink subgraphs. We can select sink subgraphs two at a time and determine the largest induced complete subgraph consisting only of vertices from these two sink subgraphs. This procedure requires solving $\binom{t}{2}$ subproblems, where each subproblem can be performed in time that is polynomial in n.

Lemma 3.5. *For a pair of sink subgraphs $\hat{V}_j$ and $\hat{V}_{j'}$, finding the maximum induced complete subgraph of $\hat{V}_j \cup \hat{V}_{j'}$ takes time polynomial in the sink subgraph size, n.*

Proof. If sinks j and j' do not have any overlapping paths, then because $\hat{V}_j$ and $\hat{V}_{j'}$ are disjoint, the maximum induced complete subgraph of $\hat{V}_j \cup \hat{V}_{j'}$ is $\hat{V}_j$ (or $\hat{V}_{j'}$), which has size n. Disjointness of $\hat{V}_j$ and $\hat{V}_{j'}$ can be checked by considering each vertex $v_{jk} \in \hat{V}_j$ and seeing if it has an edge to any vertex in $\hat{V}_{j'}$. This requires n steps.

If a path p_{jk} to sink j contaminates onto some path to sink j', then the regular vertex v_{jk} is connected to all of the fictitious vertices of $\hat{V}_{j'}$. Thus, the largest induced complete subgraph consisting of both regular vertices from $\hat{V}_j$ and fictitious vertices from $\hat{V}_{j'}$ has size $m_{j,j'} + (n - n_{j'})$, where $m_{j,j'}$ is the number of regular vertices of $\hat{V}_j$ that are connected to the fictitious vertices of $\hat{V}_{j'}$. (Recall that $n - n_{j'}$ is the number of fictitious vertices of $\hat{V}_{j'}$.) Computing $m_{j,j'}$ takes $O(n)$ time, as it merely requires counting up the number of regular vertices of $\hat{V}_j$ that are connected to the fictitious vertices of $\hat{V}_{j'}$. Equivalently, $m_{j,j'}$ can be computed by counting the number of paths to sink j that contaminate onto some path to sink j'. Of course, the largest induced complete subgraph of $\hat{V}_j \cup \hat{V}_{j'}$ may instead consist of regular vertices from $\hat{V}_{j'}$ and fictitious vertices from $\hat{V}_j$; by a similar argument, finding such a subgraph also takes polynomial time. Then we can find the maximum induced complete subgraph of $\hat{V}_j \cup \hat{V}_{j'}$, and the size of this induced subgraph is $n + \max(0, m_{j,j'} - n_{j'}, m_{j',j} - n_j)$. □

From this, it can be readily shown that finding the maximum clique in the coloring graph is polynomial-time. We can then conclude that it is possible to determine the existence of a saturating and decodable solution (again, disregarding intermediate decoding) in polynomial time.

Theorem 3.6. *For a particular non-uniform demand scenario, if the associated coloring graph is a Berge graph, then it is a polynomial-time operation to determine whether or not there exists a saturating and decodable solution that does not require intermediate decoding. Moreover, if the solution exists, it can be found using the non-uniform demand stream assignment algorithm (Algorithm 3.2).*

Proof. From Lemma 3.5, we know that finding the maximum induced clique between two sink subgraphs is a polynomial time operation. Thus, finding the maximum clique of the coloring graph takes polynomial time, as it consists of solving $\binom{t}{2} = \frac{t(t-1)}{2}$ such

subproblems. Because this coloring graph is a Berge graph, then its chromatic number can be found in polynomial time, since the chromatic number is equal to the maximum clique size. From Theorem 3.3, we can then determine if the original network graph has a saturating and decodable solution. Not only that, but if the coloring requires no more than n colors, then the coloring found from the stream assignment algorithm immediately gives the non-uniform demand solution. □

The interpretation of Theorem 3.6 is that for coloring graphs that are Berge graphs, if we find that the maximum clique has size n, then we can conclude that the coloring graph can be colored with n colors, and so the original non-uniform demand network coding problem has a saturating and decodable solution. If, however, the maximum clique has size greater than n, then we can also conclude that no saturating and decodable solution exists—at least no solution that does not require intermediate decoding. This result is particularly promising, as we can then quickly enumerate a sufficient condition under which a saturating and decodable solution can be found in polynomial time—specifically, if the associated coloring graph is Berge.

One might ask if the step of determining whether or not a graph is Berge might be a difficult task, as any difficulties in doing so would outweigh any benefits gained by solving the demand problem efficiently. However, a polynomial-time algorithm for recognizing Berge graphs does exist [19]. Consequently, if it is determined that the coloring graph is a Berge graph, then the stream assignment algorithm can be used to find the saturating and decodable solution to the non-uniform demand problem in polynomial time. Or, if it is determined that the coloring graph is not a Berge graph, then some other, perhaps superpolynomial time, algorithm will be needed to perform the coloring step.

However, non-uniform demand scenarios that lead to non-Berge coloring graphs do exist. We give an example.

Example 3.2 (Network with non-Berge coloring graph)**.** *Consider the network given in Figure 3.5. Its corresponding coloring graph (Figure 3.6) has an odd hole of length* 5*, induced by the vertices* $v_{1,1}$*,* $w_{1,1}$*,* $v_{2,1}$*,* $v_{2,2}$*, and* $w_{3,1}$*, so it is not Berge. However, a valid coloring of size* $n = 3$ *does exist.*

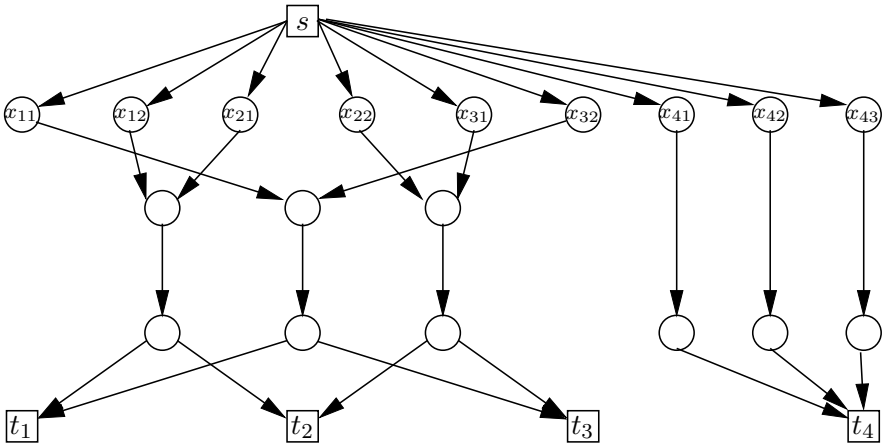


Figure 3.5: Network with non-Berge coloring graph. For each j and k, path $p_{j,k}$ is the path that passes through node x_{jk} on its way to sink t_j.

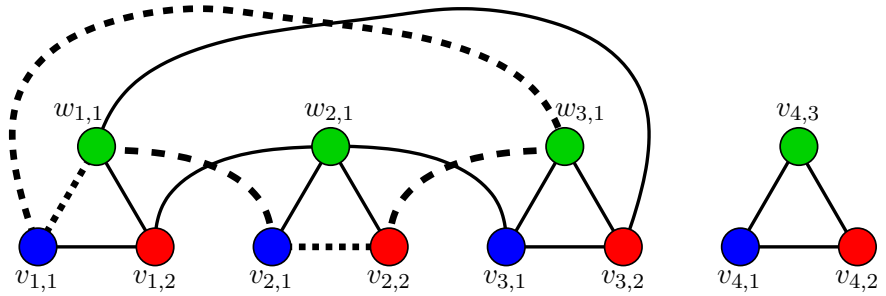


Figure 3.6: Coloring graph for non-Berge network. The dotted lines indicate the induced subgraph of vertices $v_{1,1}$, $w_{1,1}$, $v_{2,1}$, $v_{2,2}$, and $w_{3,1}$; the induced subgraph is an odd hole of length 5, so this coloring graph is not a Berge graph. It should be noted, however, that a coloring using $n = 3$ colors does exist for this graph, and is as shown.

3.6 Conclusion

In this chapter, we have considered the non-uniform demand network coding problem, where the sinks are allowed to receive data at unequal rates. We give an algorithm for finding network coding solutions that satisfy the decodability and saturation properties. Additionally, we show that for certain types of networks, i.e., those that can be transformed into equivalent Berge graphs, our algorithm can find the solution in polynomial time. Moreover, it will be difficult to do much better for the general case, since the non-uniform demand problem is NP-hard, even when demands are restricted to only those that are saturating.

Our results can be interpreted relative to the case of equal-rate multicast, where network coding solutions can always be found in polynomial time [44]. Our algorithm relies on introducing fictitious vertices in the coloring graph, which corresponds to introducing fictitious paths in the original network. These fictitious paths do not overlap anywhere with any other paths, but serve only to bring the total number of paths to each sink up to n. We can perform multicast on the expanded network consisting of the original network plus the network induced by the fictitious paths; in this expanded network, each sink is guaranteed to receive n streams of information. The main contribution of our work is that using our algorithm, we can directly specify that the subset of paths in the expanded network corresponding to paths in the original network is assigned mutually decodable streams, without needing any of the information transmitted on the fictitious paths. From the perspective of the equal-rate multicast problem on the expanded graph, the interpretation of our algorithm is that it provides a partitioning of information between data transmitted on the original network paths and data transmitted on the fictitious paths.

A few issues require further study. We have not considered the optimal selection of paths; our assumption is that the set of paths we use are the ones that give the demand solution if it exists. The optimal selection of paths is a challenging problem in itself, as it requires knowing what types of demand solutions may arise from the particular choice of paths. When implementing this algorithm, heuristics, such as minimizing the number of overlapping links on paths to different sinks, will most

likely be sufficient. We also mention that although our conditions guarantee that a network code will exist if our algorithm finds a solution, the actual construction of the network code is only briefly mentioned. In the case of no intermediate decoding, linear codes will be sufficient. However, if intermediate decoding were to be allowed, then we must be more careful, as it has been shown that sometimes nonlinear codes are required to solve certain other network coding problems [30].

Our approach considers network coding scenarios that are scalar, where the same code is employed during every time period. Although this allows for a wide variety of codes and is also practically implementable, there are certain network coding problems where vector solutions (i.e., solutions where the network code may be different at each time period) exist, but scalar solutions do not [58]. One avenue of inquiry would be the adaptation of our algorithms to find vector solutions in the cases where scalar solutions do not exist; this might be possible by augmenting our network graphs to also include a time dimension. However, characterizing the set of networks with polynomial-time-solvable vector solutions (but no scalar solutions) will require more work.

Chapter 4

Delay-Rate Tradeoff for Ergodic Interference Alignment

4.1 Introduction

The technique of interference alignment has expanded what is known about achievable rates for wireless interference channels. First proposed by Maddah-Ali et al. [56] and then applied to wireless interference channels by Cadambe and Jafar [12], interference alignment employs a transmission strategy that compensates for the interference channel between transmitters and receivers. At each receiver, the interference components can then be consolidated into a part of the channel that is orthogonal to the signal component. In fact, the interference is isolated to half of the received signal space, while the desired signal is located in the other half—leading to the statement that every receiver can have "half the cake." This is a significant improvement over every receiver receiving only $1/K$ of the cake, which is the case if standard orthogonalization techniques are used (where K is the number of transmitter-receiver pairs).[1]

Interference alignment in an ergodic setting is studied in Nazer et al. [61] and provides the basis for our analysis. Using their Gaussian achievable scheme, we delve

[1] Another interpretation of the boon due to interference alignment is an affirmative answer to the question "Can 100 speakers talk for 30-minutes each in one room within one hour and with zero interference to each other's audience?" [11]

deeper into the associated decoding delays and consider how delays can be reduced, although at the cost of decreased rate. Even though the analysis in [61] additionally considers a scheme for finite field channels (also similar to the method in [45]), we defer to the reader the extension of our analysis to the finite field case.

Our approach for reducing delays is to consider interference alignment where alignment may require more than one additional instance of channel fading. In [61], interference is aligned by transmitting the same message symbol during complementary channel realizations. In contrast, our approach will utilize multiple channel realizations (potentially more than two), which when summed together yield cancelled interference (and amplified signal). We call such a set of channel matrices an *alignment set*; this will be defined more formally later. Using multiple channel realizations to align interference has also been studied in [62] for different cases of receiver message requirements; however, we instead consider how to utilize these many channel realizations to reduce the delay of individual messages at each receiver. At first glance, it may seem that using alignment sets of larger sizes will only increase the delay; but if we allow alignment using alignment sets of multiple sizes simultaneously, then we can decrease the time required for a message symbol to be decoded.

We now give a simple example of an alignment set and show the concept of ergodic interference alignment.

Example 4.1. *Consider a* 3*-user Gaussian interference channel with channel response given by* $\boldsymbol{Y} = \boldsymbol{H}\boldsymbol{X} + \boldsymbol{Z}$*, where* $\boldsymbol{X}$ *is the [perhaps complex] transmitted symbol vector (with power constraint* $E[|X_k|^2] \leq P$ *for each user* $k = 1, 2, 3$*),* $\boldsymbol{H}$ *is the channel matrix,* $\boldsymbol{Z}$ *is an independently and identically distributed zero-mean unit-variance additive white Gaussian noise vector, and* $\boldsymbol{Y}$ *is the received symbol vector. Suppose*

the following channel matrices occur at time steps t_0, t_1, t_2, and t_3, respectively:

$$\boldsymbol{H}^{(0)} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{H}^{(1)} = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

$$\boldsymbol{H}^{(2)} = \begin{bmatrix} 1 & 1 & -1 \\ -1 & 1 & 1 \\ -1 & -1 & 1 \end{bmatrix}, \quad \boldsymbol{H}^{(3)} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}.$$

If the same vector $\boldsymbol{X}$ is sent at all these times, then the sum of the non-noise terms is given by $\sum_{i=0}^{3} \boldsymbol{H}^{(i)}\boldsymbol{X} = 4[X_1, X_2, X_3]^T$ because $\sum_{i=0}^{3} \boldsymbol{H}^{(i)} = 4\boldsymbol{I}$. By utilizing all four channel realizations together, the signals (diagonal entries) are amplified, while the interference terms (off-diagonal entries) are cancelled, so this collection of matrices is an alignment set. As long as a receiver knows when an alignment set occurs, then in order to decode its own message, it does not need to know the channel fades to the other receivers.

Inferring from [13] or [14], the astute reader may notice that in the example, the sum capacity when sending across each channel matrix separately is actually greater than the alignment rate—a capacity of $4\log(1+3P)$ for separate coding, compared to a rate of $3\log(1+4P)$ by using the indicated interference alignment scheme. However, when the number of transmitters (and receivers) exceeds the number of alignment channel realizations, then the rate benefits of using alignment sets start to become evident. Aligning across 4 channel realizations with K transmitter-receiver pairs, a rate of $K\log(1+4P)$ is achievable, which can quickly eclipse the separate-coding sum capacity of $4\log(1+KP)$. Moreover, as we will discuss, the benefit of using larger alignment sets is not in the rate, but rather in the reduction of decoding delay.

In the next section, we will formally describe the interference alignment setup, and define our notions of rate and delay. In Section 4.3, we will take a brief look at the conventional ergodic interference alignment scheme, by considering the rate and delay inherent in aligning interference using complementary channel realizations.

Section 4.4 will give the main result of this work, which is the analysis of rate and delay when aligning interference by utilizing multiple channel realizations. We will also give a scheme for trading off the rate and the delay. We conclude in Section 4.5.

4.2 Preliminaries

The setup is the same as the K-user interference channel of [61] and [62], where there are K transmitter-receiver pairs. The number of channel uses is n. For the k-th transmitter, $k = 1, \ldots, K$, each message w_k is chosen independently and uniformly from the set $\{1, 2, \ldots, 2^{n\tilde{R}_k}\}$ for some $\tilde{R}_k \geq 0$. Only transmitter k knows message w_k. Let $\mathcal{X}$ be the channel input and output alphabet. The message w_k is encoded into the n channel uses using the encoder $\mathcal{E}_k : \{1, 2, \ldots, 2^{n\tilde{R}_k}\} \to \mathcal{X}^n$. The output of the encoding function is the transmitted symbol $X_k(t) = [\mathcal{E}_k(w_k)]_t$ at time t, for $t = 1, \ldots, n$.

The communication channel undergoes fast fading, so the channel fades change at every time step. At time t, the channel matrix $\boldsymbol{H}(t)$ has complex entries $[\boldsymbol{H}(t)]_{kl} = h_{kl}(t)$ for $k, l = 1, \ldots, K$. In this model, all transmitters and receivers are given perfect knowledge of $\boldsymbol{H}(t)$ at all times t. Let $\mathcal{H}$ be the set of all possible channel fading matrices.

The message symbol $X_k(t)$ is transmitted at time t. We assume zero delay across the channel, so the channel output seen by receiver k at time t is the received symbol

$$Y_k(t) = \sum_{l=1}^{K} h_{kl}(t) X_l(t) + Z_k(t), \tag{4.1}$$

where $Z_k(t)$ is an additive noise term. Each receiver k then decodes the received message symbols according to $\mathcal{D}_k : \mathcal{X}^n \to \{1, 2, \ldots, 2^{n\tilde{R}_k}\}$ to arrive at an estimate $\hat{w}_k$ for w_k.

Definition 4.1. *The ergodic rate tuple* $(R_1, R_2, \ldots, R_K)$ *is* achievable *if for all* $\epsilon > 0$ *and large enough* n*, there exist channel encoding and decoding functions* $\mathcal{E}_1, \ldots, \mathcal{E}_K$*,* $\mathcal{D}_1, \ldots, \mathcal{D}_K$ *such that* $\tilde{R}_k > R_k - \epsilon$ *for* $k = 1, 2, \ldots, K$ *and* $\Pr\left(\bigcup_{k=1}^{K} \{\hat{w}_k \neq w_k\}\right) < \epsilon$.

We will consider a Gaussian channel with complex channel inputs and outputs, so $\mathcal{X} = \mathbb{C}$. Each transmitter k has power constraint

$$E[|X_k(t)|^2] \leq \mathrm{SNR}_k,$$

where $\mathrm{SNR}_k \geq 0$ is the signal-to-noise ratio. The channel coefficients $h_{kl}(t)$, $k, l = 1, \ldots, K$, are independently and identically distributed both in space and time. We require also that h_{kl} be drawn from a distribution that is symmetric about zero, so $\Pr(h_{kl}) = \Pr(-h_{kl})$. The noise terms $Z_k(t)$ are drawn independently and identically from a circularly symmetric complex Gaussian distribution; thus, $Z_k(t) \sim \mathcal{CN}(0, 1)$.

4.2.1 Channel Quantization

In this exposition, we consider quantized versions of the channel matrix. For some quantization parameter $\gamma > 0$, let $Q_\gamma(h_{kl})$ be the closest point in $(\mathbb{Z} + j\mathbb{Z})\gamma$ to h_{kl} in Euclidean distance. The γ-quantized version of the channel matrix $\boldsymbol{H} \in \mathbb{C}^{K \times K}$ is given by the entries $[\boldsymbol{H}_\gamma]_{kl} = Q_\gamma(h_{kl})$.

Our scheme uses typical realizations of the channel matrices. For any $\epsilon > 0$, choose the maximum magnitude $\tau > 0$ such that $\Pr(\bigcup_{k,l}\{|h_{kl}| > \tau\}) < \frac{\epsilon}{3}$. Throw out all time indices with any channel coefficient magnitude larger than τ. Let γ and δ be small positive constants. Then choose n large enough so that the typical set of sequences A_δ^n of channel matrices has probability $\Pr(A_\delta^n) \geq 1 - \frac{\epsilon}{3}$ (see [61] for details). Because this sequence of γ-quantized channel matrices is δ-typical, the corresponding rate decrease is no more than a fraction of δ.

In the remainder of this chapter, we will deal only with the γ-quantized channel matrices $\boldsymbol{H}_\gamma$, so we drop the subscript γ; all further occurrences of $\boldsymbol{H}$ refer to the quantized channel realization $\boldsymbol{H}_\gamma$. We also redefine the channel alphabet $\mathcal{H}$ to include only the typical set of quantized channel matrices, which has cardinality $|\mathcal{H}| = (2\tau/\gamma)^{2K^2}$.

4.2.2 Aligning Interference

In the standard interference alignment approach, the interference is aligned by considering the channel matrix $\boldsymbol{H}$ in tandem with its complementary matrix $\boldsymbol{H}^c$, where

$$\boldsymbol{H}^c = \begin{bmatrix} h_{11} & -h_{12} & \cdots & -h_{1K} \\ -h_{21} & h_{22} & \cdots & -h_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ -h_{K1} & -h_{K2} & \cdots & h_{KK} \end{bmatrix}.$$

That is, $\boldsymbol{H}^c$ has entries h_{kl} for $k = l$ and $-h_{kl}$ for $k \neq l$.

For alignment using more channel realizations, we define the concept of an alignment set.

Definition 4.2. *An* alignment set *of size $m \in 2\mathbb{Z}^+$ is a collection of matrices $\mathcal{A} = \{\boldsymbol{H}^{(0)}, \boldsymbol{H}^{(1)}, \ldots, \boldsymbol{H}^{(m-1)}\}$ such that the diagonal entries (signal terms) are the same:*

$$h_{kk}^{(0)} = h_{kk}^{(1)} = \cdots = h_{kk}^{(m-1)} \tag{4.2}$$

for $k = 1, \ldots, K$, and the sum of interference terms cancel:

$$\left|h_{kl}^{(0)}\right| = \left|h_{kl}^{(1)}\right| = \cdots = \left|h_{kl}^{(m-1)}\right| \tag{4.3}$$

and

$$\left|\{h_{kl}^{(i)} = h_{kl}^{(0)} \mid i = 1, \ldots, m-1\}\right| = \tfrac{m}{2} - 1 \tag{4.4}$$

$$\left|\{h_{kl}^{(i)} = -h_{kl}^{(0)} \mid i = 1, \ldots, m-1\}\right| = \tfrac{m}{2} \tag{4.5}$$

for $k = 1, \ldots, K$, $l = 1, \ldots, K$, $k \neq l$. Within an alignment set, the sum of channel matrices, $\boldsymbol{B} = \sum_{i=0}^{m-1} \boldsymbol{H}^{(i)}$, will have entries $b_{kk} = m h_{kk}^{(0)}$ and $b_{kl} = 0$, for $k, l = 1, \ldots, K$, $k \neq l$. We denote $\mathcal{A}_{\boldsymbol{H}}$ to be an alignment set of which $\boldsymbol{H}$ is a member.

We have seen some examples of alignment sets already. Any channel realization $\boldsymbol{H}$

and its complement $\boldsymbol{H}^c$ together form an alignment set of size 2. Additionally, the set of matrices given in Example 4.1 is an alignment set of size 4.

Since channel transmission is instantaneous, the only delay considered is due to waiting for the appropriate channel realizations before a message symbol can be decoded.

Definition 4.3. *The* average delay *of an ergodic interference alignment scheme is the expected number of time steps between the first instance a message symbol $\boldsymbol{X}$ is sent and the time until $\boldsymbol{X}$ is recovered at the receiver.*

If $\boldsymbol{X}(t_0)$ is sent at time t_0 but cannot be decoded until the appropriate interference alignment occurs at time t_1, then the delay is $t_1 - t_0$. Note that the delay does not consider the decoding of the entire message w_k, just the symbols transmitted at each individual time, $X_k(t)$, $k = 1, \ldots, K$.

4.3 Interference Alignment using Complementary Channel Realizations

The method of interference alignment via sending the same channel input vector when a complementary channel realization occurs is given in [61]. Call $R_k^{(2)}$ the achievable rate for interference alignment using complements, i.e., requiring two channel realizations before decoding each message symbol.

Lemma 4.1 ([61, Theorem 3])**.** *An achievable rate tuple by aligning using complementary channel realizations is*

$$R_k^{(2)} = \tfrac{1}{2} E[\log(1 + 2|h_{kk}|^2 \mathrm{SNR}_k)]$$

for $k = 1, \ldots, K$, where the expectation is over the distribution of channel fades h_{kk} drawn from the matrices in $\mathcal{H}$.

When a channel realization $\boldsymbol{H}$ occurs, then the sent message symbol is decoded

when the complementary channel realization $\boldsymbol{H}^c$ occurs. Let $d^{(2)}$ denote the average delay between channel realizations $\boldsymbol{H}$ and $\boldsymbol{H}^c$.

Lemma 4.2. *When all channel realizations are equally likely, the average delay incurred by performing interference alignment with complementary channel realizations is* $d^{(2)} = |\mathcal{H}|$.

Proof. At each time, each channel realization is equally likely. The time until $\boldsymbol{H}^c$ occurs is a geometric random variable with parameter $\Pr(\boldsymbol{H}^c) = 1/|\mathcal{H}|$, hence the average delay is $|\mathcal{H}|$. □

Note that the delay $d^{(2)}$ can be quite large. Using our quantization scheme, the delay is $d^{(2)} = |\mathcal{H}| = (2\tau/\gamma)^{2K^2}$.

4.4 Interference Alignment using Multiple Channel Realizations

This section will focus on using alignment sets of sizes $m = 2$ and $m = 4$. Extensions for larger alignment sets will be discussed in Section 4.4.3.

For ease of analysis, we assume that each channel realization $\boldsymbol{H}$ is equally likely, although the ideas presented may be readily extended to the cases where the distribution of channel realizations is non-uniform. However, for this particular interference alignment scheme to work, all channel realizations within the same alignment set must be equiprobable: for an alignment set $\mathcal{A}_{\boldsymbol{H}} = \{\boldsymbol{H}, \boldsymbol{H}^{(1)}, \boldsymbol{H}^{(2)}, \ldots, \boldsymbol{H}^{(m-1)}\}$, we require that $\Pr(\boldsymbol{H}) = \Pr(\boldsymbol{H}^{(1)}) = \Pr(\boldsymbol{H}^{(2)}) = \cdots = \Pr(\boldsymbol{H}^{(m-1)})$. Fortunately, this condition is satisfied since we assume that channel entries are drawn from distributions that are symmetric about zero.

4.4.1 First-to-Complete Alignment

We call the following scheme for achieving lower delay the *first-to-complete* scheme. This is essentially a coupon-collecting race between an alignment set of size 2 and an

alignment set of size 4. Since the entire future of channel realizations is known, for the channel realization $\boldsymbol{H} \in \mathcal{H}$ occurring at time t_0 we can collect the realizations occurring at future times $t > t_0$. We say that an alignment set $\mathcal{A}_{\boldsymbol{H}}$ of size 4 has been *completed* once all matrices $\tilde{\boldsymbol{H}} \in \mathcal{A}_{\boldsymbol{H}}$ have been realized. If $\boldsymbol{H}^c$ occurs before $\mathcal{A}_{\boldsymbol{H}}$ is completed, then pair up $\boldsymbol{H}$ with that realization of $\boldsymbol{H}^c$. Otherwise, group together $\boldsymbol{H}$ with the other members of the alignment set $\mathcal{A}_{\boldsymbol{H}}$.

We derive the achievable rate by separately finding the rates when decoding using alignment sets of different sizes, and then weighting these rates by the probabilities that a particular size set is completed before the other. From [61], if $\boldsymbol{H}$ at time t_0 is paired with $\boldsymbol{H}^c$ at time t_1, then the same symbol vector $\boldsymbol{X}(t_0)$ is transmitted at both times t_0 and t_1. Since this is alignment with channel complements, the rate $R_k = \frac{1}{2}E[\log(1 + 2|h_{kk}|^2\mathrm{SNR}_k)] - \epsilon$ is achievable with probability $1 - \epsilon$.

Now we find the rate when $\boldsymbol{H}$ at time $\hat{t}_0 = t_0$ is instead grouped with the members of its size-4 alignment set $\mathcal{A}_{\boldsymbol{H}}$. Assume that the channel realizations of the other members of the alignment set occur at times $\hat{t}_1$, $\hat{t}_2$, and $\hat{t}_3$, respectively. In the scheme, we send the same message symbol $X_k(\hat{t}_0)$ at times $\hat{t}_0$, $\hat{t}_1$, $\hat{t}_2$, and $\hat{t}_3$. The channel outputs are

$$Y_k(t) = h_{kk}(t)X_k(\hat{t}_0) + \sum_{l \neq k} h_{kl}(t)X_l(\hat{t}_0) + Z_k(t) \tag{4.6}$$

for $t = \hat{t}_0, \hat{t}_1, \hat{t}_2, \hat{t}_3$. From the alignment set definition, we know $h_{kk}(\hat{t}_0) = h_{kk}(\hat{t}_1) = h_{kk}(\hat{t}_2) = h_{kk}(\hat{t}_3)$ and $h_{kl}(\hat{t}_0) + h_{kl}(\hat{t}_1) + h_{kl}(\hat{t}_2) + h_{kl}(\hat{t}_3) = 0$ for $k = 1, \ldots, K$ and $l \neq k$. Thus, the signal-to-interference-plus-noise ratio of the channel from $X_k(\hat{t}_0)$ to $Y_k(\hat{t}_0) + Y_k(\hat{t}_1) + Y_k(\hat{t}_2) + Y_k(\hat{t}_3)$ is at least

$$\frac{\mathrm{SNR}_k((4|\Re(h_{kk})| - 2\gamma)^2 + (4|\Im(h_{kk})| - 2\gamma)^2)}{4 + (2\gamma)^2 \sum_{l \neq k} \mathrm{SNR}_l}.$$

Taking the channel quantization parameter $\gamma \to 0$, the SINR is $4|h_{kk}|^2\mathrm{SNR}_k$, which

gives the rate (as $\tau \to \infty$):

$$R_k = \tfrac{1}{4} E[\log(1 + 4|h_{kk}|^2 \mathrm{SNR}_k)] - \tfrac{2}{3}\epsilon. \tag{4.7}$$

Thus there exist γ and τ such that we achieve $R_k > \frac{1}{4} E[\log(1 + 4|h_{kk}|^2 \mathrm{SNR}_k)] - \epsilon$ with probability $1 - \epsilon$ when aligning using an alignment set of size 4.[2]

Recall that $\boldsymbol{H}$ at time t_0 is grouped only with the channel realizations of the alignment set that completes first, so that the realizations corresponding to the other alignment sets are *not* associated with $\boldsymbol{H}$ and can be used for some other transmissions. For example, if $\boldsymbol{H}^c$ occurs between times $\hat{t}_1$ and $\hat{t}_2$—i.e., at time t_1, where $t_0 < \hat{t}_1 < t_1 < \hat{t}_2 < \hat{t}_3$—then since the transmitter knows the sequence of channel realizations in advance, it may avoid utilizing $\boldsymbol{H}(\hat{t}_1)$ to send $\boldsymbol{X}(t_0)$, which would become a wasted transmission when $\boldsymbol{H}^c$ occurs at time t_1. In this example, decoding is via channel complements, so $\boldsymbol{X}(t_0)$ is sent during times t_0 and t_1, but never during times $\hat{t}_1$, $\hat{t}_2$, and $\hat{t}_3$.

We now determine the probability that the first-to-complete scheme decodes using the alignment set of size 4 rather than the alignment set of size 2. This can be computed by considering a Markov chain with the following states:

- s_{-1}: Decode using $\boldsymbol{H}$ and its complement, $\boldsymbol{H}^c$
- s_0: No matches yet to any alignment set
- s_1: First match with size-4 alignment set
- s_2: Second match with size-4 alignment set
- s_3: Third match with size-4 alignment set, so decode using $\mathcal{A}_{\boldsymbol{H}}$

The Markov chain is shown in Figure 4.1. States s_{-1} and s_3 are absorbing. Because this is a success runs Markov chain [81], its absorption probabilities and hitting times are known. The probability of decoding via the alignment set of size 4 is the probability of absorption at state s_3 starting from state s_0, and is computed to be $\beta_4 = 1/4$. Note that β_4 does not depend on the number of possible channel realizations, $|\mathcal{H}|$.

[2] Higher rates may be possible by optimizing power allocations, for example via waterfilling. Here we only consider rates achievable using equal-power allocations.

This is intuitively clear since matrices not belonging to an alignment set do not affect the probability that one set completes before another.

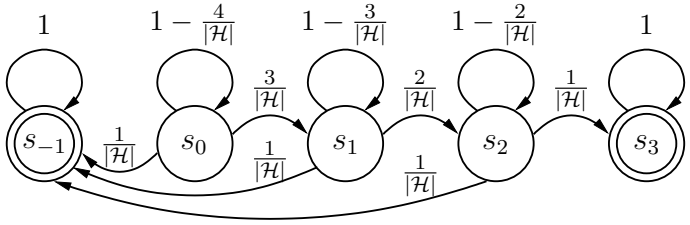


Figure 4.1: Success runs Markov chain associated with first-to-complete alignment. States indicate progress towards completion of the alignment sets. Quantities above the arrows indicate transition probabilities.

Lemma 4.3. *An achievable rate tuple for the first-to-complete scheme has rates (for all $k = 1, \ldots, K$):*

$$R_k^{(2,4)} = \tfrac{3}{8}E[\log(1 + 2|h_{kk}|^2\mathrm{SNR}_k)] + \tfrac{1}{16}E[\log(1 + 4|h_{kk}|^2\mathrm{SNR}_k)].$$

Proof. Because decoding via the size-2 alignment set occurs $1 - \beta_4$ of the time, and decoding via the size-4 alignment set occurs β_4 of the time, an achievable rate is $R_k^{(2,4)} = \frac{1}{2}(1 - \beta_4)E[\log(1 + 2|h_{kk}|^2\mathrm{SNR}_k)] + \frac{1}{4}\beta_4 E[\log(1 + 4|h_{kk}|^2\mathrm{SNR}_k)]$. Plugging in $\beta_4 = 1/4$ gives the result. □

Lemma 4.4. *For the first-to-complete scheme, the average decoding delay is $d^{(2,4)} = (3/4)|\mathcal{H}| = (3/4)d^{(2)}$.*

Proof. The delay until either alignment set is completed is the mean hitting time until one of the corresponding absorption states is reached in the Markov chain of Figure 4.1. A simple computation for the hitting time yields $d^{(2,4)} = (3/4)|\mathcal{H}|$. □

4.4.2 Delay-Rate Tradeoff

Although the first-to-complete scheme achieves lower delay than interference alignment using only complements, it has the drawback of having lower rate. By using

timesharing, we can achieve any delay d such that $(3/4)|\mathcal{H}| = d^{(2,4)} \leq d \leq d^{(2)} = |\mathcal{H}|$, and every user $k \in \{1, \ldots, K\}$ will still have increased data rate over that of $R_k^{(2,4)}$.

In the timesharing scheme, with probability $1 - \alpha$ where $0 \leq \alpha \leq 1$, pair up $\boldsymbol{H}$ with the first instance of $\boldsymbol{H}^c$ that occurs later in time; this is alignment using only complements. With probability α, however, perform the first-to-complete scheme: pair up $\boldsymbol{H}$ with $\boldsymbol{H}^c$ only if $\boldsymbol{H}^c$ occurs before any alignment set of size 4 is completed; otherwise, group $\boldsymbol{H}$ with the size-4 alignment set that completes first.

Theorem 4.5. *The achievable rate when time sharing with probability α of using the first-to-complete scheme is*

$$\begin{aligned} R_k(\alpha) &= (1-\alpha)R_k^{(2)} + \alpha R_k^{(2,4)} \\ &= \tfrac{1}{2}\left(1 - \tfrac{1}{4}\alpha\right) E[\log(1 + 2|h_{kk}|^2 \mathrm{SNR}_k)] + \tfrac{1}{16}\alpha E[\log(1 + 4|h_{kk}|^2 \mathrm{SNR}_k)]. \end{aligned}$$

Proof. Evident. □

Theorem 4.6. *The average delay when time sharing is*

$$d(\alpha) = (1-\alpha)d^{(2)} + \alpha d^{(2,4)} = (1 - \tfrac{1}{4}\alpha)|\mathcal{H}|.$$

Proof. Evident. □

Corollary 4.7. *The average delay, when time sharing between the first-to-complete scheme (using alignment sets of both sizes 2 and 4) and channel-complement alignment, is lower than the average delay when using only complements.*

Proof. By choosing any $\alpha > 0$, we get delay $d(\alpha)$ strictly less than $|\mathcal{H}| = d^{(2)}$. □

The reduced delay is an intuitive result since the first-to-complete scheme allows additional opportunities to align, without disallowing existing opportunities.

4.4.3 Extension to Larger Alignment Sets

We now extend our analysis to more general collections of alignment sets. Consider a finite tuple of positive even numbers $I = (m_1, m_2, \ldots, m_{|I|})$, possibly with repetitions.

We generalize first-to-complete alignment by using non-overlapping alignment sets with sizes dictated by the entries of I. As soon as all members of any particular alignment set have been seen, we say that that alignment set has been completed; we transmit and decode using the particular alignment set. As an example, the first-to-complete alignment scheme given in the first part of this section corresponds to $I = (2, 4)$. For the case of a general tuple I, the process is identical to the multiple subset coupon collecting problem of Chang and Ross [17], in which coupons are repeatedly drawn with replacement until any one of several predetermined subsets of coupons have been collected.

To compute the achievable rates $(R_1^I, R_2^I, \ldots, R_K^I)$ and delay d^I associated with running first-to-complete alignment among I-sized alignment sets, we construct the associated Markov chain. The state vector $\boldsymbol{s} = (s_1, s_2, \ldots, s_{|I|})$ is defined so that element s_i counts how many members of the i-th alignment set have already occurred, excluding the initial matrix $\boldsymbol{H}$. Initially, the Markov chain is at state $\boldsymbol{s} = \boldsymbol{0}$, since no alignment set member aside from $\boldsymbol{H}$ has yet been realized. At each time t, if $\boldsymbol{H}(t)$ is a member of the i'-th alignment set and has not yet been realized, then increment $s_{i'} := s_{i'} + 1$. When $s_{i'} = m_{i'} - 1$ for some i', this means that the i'-th alignment set (of size $m_{i'}$) has been completed. The Markov chain enters an absorbing state, and the receiver decodes. Let V denote the set of absorbing states. The state transition probabilities are

$$P_{\boldsymbol{s},\tilde{\boldsymbol{s}}} = \begin{cases} \dfrac{m_{i'} - 1 - s_{i'}}{|\mathcal{H}|} & \tilde{s}_{i'} = s_{i'} + 1 \text{ for some } i', \ldots \\ & \quad \tilde{s}_i = s_i \text{ for all } i \neq i', \ \boldsymbol{s} \notin V \\ 1 - \sum_i \dfrac{m_i - 1 - s_i}{|\mathcal{H}|} & \tilde{\boldsymbol{s}} = \boldsymbol{s}, \ \boldsymbol{s} \notin V \\ 1 & \tilde{\boldsymbol{s}} = \boldsymbol{s}, \ \boldsymbol{s} \in V \text{ (absorption)} \\ 0 & \text{otherwise} \end{cases} .$$

Let β_m^I be the probability that the first completed alignment set is the alignment set of size $m \in I$. Equivalently, β_m^I is the probability that the Markov chain reaches the absorption state corresponding to the completion of a specific size-m alignment set.

These absorption probabilities can be computed via matrix inversion (see Appendix D or Taylor and Karlin [81] for more details). Table 4.1 gives example values for β_m^I.

Set sizes	Absorption probability			Delay
I	$\beta_{m_1}^I$	$\beta_{m_2}^I$	$\beta_{m_3}^I$	d^I
$(2,4)$	0.75	0.25		$0.75\lvert\mathcal{H}\rvert$
$(2,6)$	0.8333	0.1667		$0.8333\lvert\mathcal{H}\rvert$
$(2,4,4)$	0.6429	0.1786	0.1786	$0.6429\lvert\mathcal{H}\rvert$
$(2,4,6)$	0.6944	0.2083	0.0972	$0.6944\lvert\mathcal{H}\rvert$
$(4,4)$	0.5	0.5		$1.2167\lvert\mathcal{H}\rvert$
$(4,6)$	0.625	0.375		$1.3988\lvert\mathcal{H}\rvert$
$(4,8)$	0.7	0.3		$1.4972\lvert\mathcal{H}\rvert$
$(4,4,4)$	0.3333	0.3333	0.3333	$0.9790\lvert\mathcal{H}\rvert$
$(6,10)$	0.6429	0.3571		$1.8607\lvert\mathcal{H}\rvert$

Table 4.1: Absorption probabilities and delays. For values to be valid, $|\mathcal{H}| \geq 1 + \sum_{i=1}^{|I|}(m_i - 1)$ must hold.

Following a similar argument as in Lemma 4.3, the rate for receiver $k \in \{1, \ldots, K\}$ by using a first-to-complete scheme with specific alignment sets of sizes drawn from I is

$$R_k^I = \sum_{m \in I} \frac{1}{m} \beta_m^I E[\log(1 + m|h_{kk}|^2 \mathrm{SNR}_k)].$$

We now incorporate timesharing and describe the delay-rate tradeoff. Let $\mathcal{I}$ be a finite collection of these tuples I; that is, $\mathcal{I} \subseteq \{I = (m_1, \ldots, m_{|I|}) \mid m_i \in 2\mathbb{Z}^+\}$. We can do timesharing between first-to-complete schemes, with sizes drawn from $I \in \mathcal{I}$, according to the vector $\boldsymbol{\alpha} = (\alpha_{I_1}, \alpha_{I_2}, \ldots, \alpha_{I_{|\mathcal{I}|}})$ where $\sum_{I \in \mathcal{I}} \alpha_I = 1$ and $\alpha_I \geq 0$ for all $I \in \mathcal{I}$. The rate will be

$$R_k(\boldsymbol{\alpha}) = \sum_{I \in \mathcal{I}} \alpha_I R_k^I. \tag{4.8}$$

Alternatively, to be explicit about the rates due to alignment sets of particular sizes, the rate can also be written as

$$R_k(\boldsymbol{\alpha}) = \sum_{m \in 2\mathbb{Z}^+} \left(\sum_{I \in \mathcal{I} \,:\, m \in I} \alpha_I \beta_m^I \right) \frac{1}{m} E[\log(1 + m|h_{kk}|^2 \mathrm{SNR}_k)].$$

The average delay using alignment sets of sizes $I = (m_1, m_2, \ldots, m_{|I|})$ is equal to the mean absorption time for the Markov chain. From [17], by using poissonization [74] this delay can be computed as[3]

$$d^I = |\mathcal{H}| \int_0^1 \frac{1}{1-u} \prod_{i=1}^{|I|} (1 - u^{m_i - 1})\, du. \tag{4.9}$$

Table 4.1 gives average delays for some representative collections of alignment sets. Then the delay using timesharing is

$$d(\boldsymbol{\alpha}) = \sum_{I \in \mathcal{I}} \alpha_I d^I, \tag{4.10}$$

which is linear in the number of possible channel realizations, $|\mathcal{H}|$.

From Table 4.1, we can make an observation regarding the computed absorption probabilities and associated delays. When the first alignment set has size 2, notice that $d^I = \beta_2^I |\mathcal{H}|$. This holds for any tuple I that contains an alignment set of size 2 (see Appendix D).

[3]This evaluates to an inclusion-exclusion sum of harmonic numbers H_n:

$$d^I = |\mathcal{H}| \left[\sum_{U \subseteq I, U \neq \emptyset} (-1)^{1-|U|} H_{\left(-|U| + \sum_{m \in U} m\right)} \right].$$

The delay can also be expressed analytically using the digamma function Ψ, giving

$$\begin{aligned} d^I &= |\mathcal{H}| \sum_{U \subseteq I} (-1)^{1-|U|} \Psi\left(1 - |U| + \sum_{m \in U} m\right) \\ &= |\mathcal{H}| \left[\gamma + \sum_{m_1 \in I} \Psi(m_1) - \sum_{m_1, m_2 \in I} \Psi(-1 + m_1 + m_2) + \sum_{m_1, m_2, m_3 \in I} \Psi(-2 + m_1 + m_2 + m_3) - \cdots \right], \end{aligned}$$

where γ is the Euler-Mascheroni constant. Also, from [17], we can find the variance of this delay, as well as the average delay when alignment sets overlap.

4.4.4 Further Considerations

In this analysis, we consider only alignment sets that do not share any common matrices. However, as the number of allowable sizes, $|I|$, grows larger, this condition will become harder to fulfill since there will be greater potential for collisions. Finding tuples of alignment sets such that there are no overlapping channels is an avenue for future work. One thing to note is that because only $2^{K(K-1)}$ matrices satisfy $h_{kk}^{(i)} = h_{kk}$ and $|h_{kl}^{(i)}| = |h_{kl}|$ for $k = 1, \ldots, K$ and $l \neq k$, an alignment set of size $m = 2^{K(K-1)}$ would consist of all possible channel matrices that might align with $\boldsymbol{H}$, and so necessarily must collide with any other alignment set for $\boldsymbol{H}$.

A related issue is that of allowing decoding using *all* alignment sets of a particular size m, of which there are $\binom{m-1}{m/2}^{K(K-1)}$ such alignment sets. For example, a system could choose to perform first-to-complete alignment among *any* alignment set of sizes 2 and 4. Because non-intersection between different alignment sets may no longer be guaranteed, the analysis will be more complicated.

From Table 4.1, we can start to notice the potential for delay reduction via using multiple alignment sets of the same size. Although the delay will still scale linearly in $|\mathcal{H}|$, it is possible to significantly reduce the delay below $d^{(2)} = |\mathcal{H}|$. As an example, from Figure 4.2 we can observe the behavior of the linear scaling factor, in the case of allowing alignment using more and more size-4 alignment sets.[4] Thus a deeper consideration of alignment with multiple same-size alignment sets may be a fruitful area for further inquiry.

There are myriad other ways in which alignment may occur; i.e., there is more than one way to align channel matrices. Definition 4.2 gives one set of sufficient conditions for channel realizations to align, in order to keep the analysis tractable, and the benefits that arise by considering larger alignment sets are already evident. An obvious extension to this would be to consider alignment sets in which arbitrary linear combinations add up to multiples of the identity, and to only consider alignment among subsets of users. Subsequent work by [46] takes a step in this direction.

[4]Of course, the trend shown in the Figure 4.2 only holds for scenarios where the number of users K is large enough that there exists enough distinct alignment sets of size 4 for alignment.

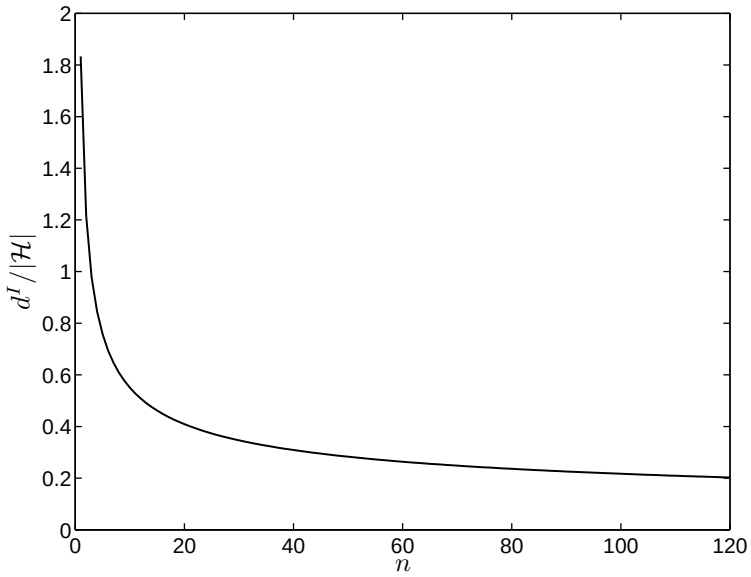


Figure 4.2: Plot showing the decrease of the delay linear scaling factor, for multiple disjoint alignment sets of size 4. The number of alignment sets of size 4 is n. Thus each point represents the delay associated with the tuple $I = (4, 4, 4, \ldots)$, where the tuple has n elements.

The moral of this story, however, is that delay can always be reduced by allowing alignment using a greater number of possible choices of alignment sets. The data rate may decrease correspondingly, so the tradeoff needs to be appropriately chosen according to the needs of the communication system.

4.5 Conclusion

In our analysis, we have not considered the delays between when a message symbol is available and when it is first transmitted. We have only defined delay as the time between when the symbol is first transmitted and when it is able to be recovered by the receiver. We believe this is a reasonable metric of delay, as long as message symbols are not all generated at one time. However, an analysis using queueing theory may be necessary to verify this claim.

In this work, we have proposed an interference alignment scheme that reduces

delay, although with potentially decreased data rate. Delay is mitigated by allowing more ways to align interference, through the utilization of larger alignment sets. We have also introduced a scheme to trade off the delay and rate. In the end, even though the rate may be reduced, we can still say, in the parlance of interference aligners, that each person gets κ of the cake, where $1/K \leq \kappa \leq 1/2$. Thus our scheme can still be an improvement over non-aligning channel-sharing strategies in terms of data rate.

Chapter 5

Conclusion

In this dissertation, we have used a variety of tools ranging from cage graph construction to graph coloring to Markov chain analysis, in order to understand and tackle the scalability issues that arise in various technological realms. Here, we have shown the necessity for considering the combinatorial nature of these problems, by explicitly describing some basic scalability challenges that arise in various storage, networking, and communications problems. By so doing, we hope to elucidate the issues that need to be addressed in order to derive solutions that will scale gracefully as computing systems grow ever larger. Furthermore, the specific solutions we have given in this work should provide a good theoretical foundation for solving the problems we have described, and give a good starting point when thinking about improving existing systems for scalability. By thinking about the combinatorics explicitly, we can understand the issues that arise when problem domains get much larger, and are also able to best pursue the directions that enable these challenges to be overcome.

Appendix A

Latin Squares

In this appendix, we discuss Latin squares and mutually orthogonal Latin squares. These are used in the construction of bipartite cage graphs of girth 6. Here, we only consider Latin squares of size $q \times q$ where q is a prime or a prime power. The material here comes from the comprehensive text by Dénes and Keedwell [26], with a few minor modifications.

A.1 Definitions

We first provide some definitions regarding Latin squares.

Definition A.1. *Consider a $q \times q$ matrix L whose entries belong to the set $\mathcal{Q} = \{0, 1, 2, \ldots, q-1\}$. Then L is a* Latin square *if for every row $i \in \{0, 1, \ldots, q-1\}$, the entries satisfy $L_{i,j} \neq L_{i,j'}$ whenever $j \neq j'$; and for every column $j \in \{0, 1, \ldots, q-1\}$, the entries satisfy $L_{i,j} \neq L_{i',j}$ whenever $i \neq i'$.*

Thus we see that in a Latin square, every symbol from $\mathcal{Q}$ occurs exactly once in each row; and every symbol occurs exactly once in each column. We call a row in which every symbol occurs exactly once to be *Latin*; a similar definition holds for a column to be called Latin.

Definition A.2. *A row i of a square L is in* natural order *if the symbols $\mathcal{Q} = \{0, 1, \ldots, q-1\}$ occur in sequential order, i.e., $L_{i,j} = j$ for $j = 0, 1, \ldots, q-1$.*

Similarly, a column j is in natural order if the symbols occur in sequential order in the column. A Latin square is in standard form *if its zeroth row and zeroth column are both in natural order.*

Example A.1. *A simple construction of Latin squares in standard form can be accomplished by setting $L_{i,j} = i + j \pmod q$; that is, by cyclically permuting a standard form zeroth row.*

In fact—given any Latin square—by labeling symbols and permuting columns appropriately, we can establish a Latin square in standard form.

We next define the concept of orthogonality for Latin squares.

Definition A.3. *A pair of $q \times q$ squares $L^{(m)}$ and $L^{(m')}$ is considered* orthogonal *if the set of ordered pairs of elements satisfies $\{(L^{(m)}_{i,j}, L^{(m')}_{i,j}) \mid i, j = 0, 1, \ldots, q-1\} = \{(a, b) \mid a, b = 0, 1, \ldots, q-1\}$. Thus the squares $L^{(m)}$ and $L^{(m')}$ are orthogonal if the pairwise concatenation of the two squares takes on all q^2 pairs of symbols chosen from $\{0, 1, \ldots, q-1\}$.*

Definition A.4. *A set of r squares $\{L^{(1)}, L^{(2)}, \ldots, L^{(r)}\}$, each of size $q \times q$, is* mutually orthogonal *if every pair of squares, $L^{(m)}$, $L^{(m')}$, where $m, m' = 1, 2, \ldots, r$ and $m \neq m'$, is orthogonal.*

A.2 Construction of Mutually Orthogonal Latin Squares

Now we discuss how to construct mutually orthogonal Latin squares. Here, we consider only Latin squares of size $q \times q$ where q is a prime or a prime power. To start, we note the following upper bound on the number of mutually orthogonal Latin squares.

Lemma A.1 ([26, Theorem 5.1.5])**.** *A set of mutually orthogonal Latin squares of size $q \times q$ can have at most $q - 1$ squares.*

Thus if we can construct a set of $q-1$ mutually orthogonal Latin squares, then it is not possible to construct any additional Latin squares that are orthogonal to all of the squares already present in the existing set.

For q a prime or power of a prime, we can construct the full set of $q-1$ mutually orthogonal Latin squares, which we name $\{L^{(1)}, L^{(2)}, \ldots, L^{(q-1)}\}$. This construction comes from [26, Theorem 5.2.4], and will involve the cyclic permutation of columns. In order to construct the mutually orthogonal Latin squares, we require the finite, or Galois, field GF(q), with elements

$$e_0 = 0, \quad e_1 = 1, \quad e_2 = \alpha, \quad e_3 = \alpha^2, \quad \ldots, \quad e_{q-1} = \alpha^{q-2}, \tag{A.1}$$

where α is a primitive element of the multiplicative group of GF(q) [55, 8]. We call the sequence $e_0, e_1, e_2, e_3, \ldots, e_{q-1}$ to be the natural order, i.e., we consider element e_i to be equivalent to symbol i.[1]

We construct the set of Latin squares $\{L^{(1)}, L^{(2)}, \ldots, L^{(q-1)}\}$ by taking

$$L^{(m)}_{i,j} = e_i + e_m e_j, \quad m = 1, 2, \ldots, q-1, \ i, j = 0, 1, \ldots, q-1. \tag{A.2}$$

For example, the first square $L^{(1)}$ has entries $L^{(1)}_{i,j} = e_i + e_j$.

Lemma A.2. *For each $m = 1, 2, \ldots, q-1$, the square $L^{(m)}$ defined with entries as in* (A.2) *is Latin and has zeroth column in natural order. Furthermore, the first Latin square, $L^{(1)}$, has zeroth row also in natural order.*

Proof. For any square $L^{(m)}$, the zeroth column consists of $L^{(m)}_{i,0} = e_i$ for all $i = 0, 1, \ldots, q-1$ (since $e_0 = 0$), and so is in natural order. For $L^{(1)}$, the zeroth row has entries $L^{(1)}_{0,j} = e_j$ for all $j = 0, 1, \ldots, q-1$ (since $e_1 = 1$ is the multiplicative identity element), and so the row is in natural order.

For a square $L^{(m)}$, consider an arbitrary column, $j \in \{0, 1, \ldots, q-1\}$. If we define the fixed element $\beta = e_m e_j$, then the entries of column j consist of $L^{(m)}_{i,j} = e_i + \beta$, $i = 0, 1, \ldots, q-1$. Because the elements belong to a field, so $L^{(m)}_{i,j} = L^{(m)}_{i',j}$ if and only

[1] If $e_i \neq i$, then we can always reorder the rows of $L^{(m)}$ so that the zeroth column consists of the symbols $\{0, 1, \ldots, q-1\}$ in sequential order.

if $i = i'$; thus the q elements in the column take on every value of e_i, $i = 0, 1, \ldots, q-1$. Since this holds for any column, then all the columns are Latin.

Now consider an arbitrary row, $i \in \{0, 1, \ldots, q-1\}$. Then the row consists of entries $L_{i,j}^{(m)} = \delta + \gamma e_j$, $j = 0, 1, \ldots, q-1$, where the $\gamma = e_m$ and $\delta = e_i$ are fixed elements. Because the elements belong to a field, so $L_{i,j}^{(m)} = L_{i,j'}^{(m)}$ if and only if $\gamma e_j = \gamma e_{j'}$, which occurs if and only if $e_j = e_{j'}$ (since $\gamma \neq 0$). Therefore, the elements in the row take on every value of $e_0, e_1, \ldots, e_{q-1}$ exactly once. Thus every row is Latin. □

Lemma A.3. *The squares $L^{(m)}$ and $L^{(m')}$, defined as in* (A.2)*, are orthogonal.*

Proof. Suppose that $L^{(m)}$ and $L^{(m')}$ were not orthogonal. Then there exist two distinct locations (i, j) and (i', j') such that

$$e_i + e_m e_j = e_{i'} + e_m e_{j'} \tag{A.3}$$

$$e_i + e_{m'} e_j = e_{i'} + e_{m'} e_{j'}. \tag{A.4}$$

Subtracting (A.4) from (A.3) gives $(e_m - e_{m'})e_j = (e_m - e_{m'})e_{j'}$. Since $m \neq m'$ (and so $e_m \neq e_{m'}$), this implies $e_j = e_{j'}$, or that $j = j'$. Then from (A.3), we get that $e_i = e_{i'}$, or that $i = i'$. However, by assumption, $(i, j) \neq (i', j')$, so we have a contradiction. Thus our assertion is proved. □

Theorem A.4. *The set of squares $L^{(m)}$ defined by* (A.2) *gives a set of $q-1$ mutually orthogonal Latin squares, each with zeroth column in natural order.*

Proof. Follows from Lemmas A.2 and A.3. □

By Lemma A.1, the construction of $L^{(m)}$ from (A.2) gives a largest possible set of mutually orthogonal Latin squares of order q.

Additionally, if we let the $q \times q$ matrix $L^{(0)}$ consist of $L_{i,j}^{(0)} = i$ for all $i, j \in \{0, 1, \ldots, q-1\}$ (i.e., each column of $L^{(0)}$ is the same, and consists of symbols numbered sequentially), then the set of squares $\{L^{(0)}, L^{(1)}, L^{(2)}, \ldots, L^{(q-1)}\}$ is a set of q mutually orthogonal squares, where only $L^{(0)}$ is not Latin.

Lemma A.5. *For the set of squares* $\{L^{(0)}, L^{(1)}, \ldots, L^{(q-1)}\}$ *as defined above, we have* $L^{(m)}_{i,j} = L^{(m')}_{i,j}$ *if and only if* $j = 0$. *That is, for any pair of squares, only the zeroth columns have equal corresponding entries.*

Proof. Because all of the squares have the zeroth column in natural order, sufficiency is immediate. Now, since there are q entries in the zeroth column and there are only q pairs of elements (a, b) such that $a = b$ (where $a, b \in \{0, 1, \ldots, q-1\}$), by the definition of mutually orthogonal squares, we know that no other column will have equal corresponding entries. □

Appendix B

Algebraic Construction of Affine Planes

In this appendix, we will provide an algebraic method for constructing block designs with $k = q$ elements per block and $l = q + 1$ replicas per element, where q is a prime number. The smallest such designs will have $v = q^2$ total elements and $u = q(q + 1)$ blocks. These designs correspond to Steiner systems of the form $S(2, q, q^2)$, and produce the affine plane designs (i.e., pruned cage graphs) of Section 2.4. For instance, Example 2.1 is such a design for when $q = 3$; we reproduce the associated block design in Figure B.1, which is the same as Figure 2.1 except that the blocks and elements are reordered so that the solution of the algebraic method is clear to see.

For this construction, the q^2 elements are $e_0, e_1, \ldots, e_{q^2-1}$; these may arise, for example, as the elements of GF(q^2) (see Appendix A or [55, 8]). When $\tilde{a} < 0$ or $\tilde{a} > q^2 - 1$, we take $e_{\tilde{a}} = e_a$, where $a = \tilde{a} \bmod q^2$. Furthermore, the $q(q + 1)$ blocks $b_0, b_1, \ldots, b_{q(q+1)-1}$ are grouped as follows:

$$\mathcal{B}_i = \begin{cases} \bigcup_{m=0}^{q-1} b_{iq+m}, & i = 0, 1, \ldots, q-1 \\ \bigcup_{m=0}^{q-1} b_{q^2+m}, & i = q \end{cases}$$

b_0	0	1	2	b_1	3	4	5	b_2	6	7	8
b_3	0	4	8	b_4	3	7	2	b_5	6	1	5
b_6	0	7	5	b_7	3	1	8	b_8	6	4	2
b_9	0	3	6	b_{10}	1	4	7	b_{11}	2	5	8

Figure B.1: Example of Steiner system $S(2,3,9)$, where blocks and elements are labeled according to the solution of the algebraic design method. This is the same as Figure 2.1, except for some reordering of blocks and elements.

The set of all blocks is $\mathcal{B} = \bigcup_{i=0}^{q} \mathcal{B}_i$.

With this definition, the algebraic method for allocating elements to blocks is straightforward and is given by the following:

- $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{q-1}$: For $i = 0, 1, \ldots, q-1$ and $m = 0, 1, \ldots, q-1$,

$$b_{iq+m} \leftarrow \{e_{(mq+j)(iq+1)} \mid j = 0, 1, \ldots, q-1\}. \tag{B.1}$$

- $\mathcal{B}_q$: For $m = 0, 1, \ldots, q-1$,

$$b_{q^2+m} \leftarrow \{e_{m+jq} \mid j = 0, 1, \ldots, q-1\}. \tag{B.2}$$

Notice that within a particular block, each of the elements in that block (i.e., for $j = 0, 1, \ldots, q-1$) are allocated according to some stride length (e.g., separated by strides of $iq+1$ in the case of (B.1), or separated by strides of q in the case of (B.2)).

The rest of this appendix will be devoted to proving that the algebraic allocation of elements to blocks given by (B.1) and (B.2) corresponds to an affine plane design.

To start, we will need the following result from number theory.

Lemma B.1. *Let n and r be coprime positive integers. The sets $\mathbb{Z}_n = \{0, 1, 2, 3, \ldots, n-1\}$ and $\mathbb{Z}_n r = \{0, r, 2r, 3r, \ldots, (n-1)r\}$ are equivalent modulo-n. That is,*

$$\mathbb{Z}_n \equiv \mathbb{Z}_n r \pmod{n}.$$

Proof. We know that $0r \equiv 0 \pmod{n}$. Now suppose we have $z, z' \in \mathbb{Z}_n$ with $z \neq z'$ that satisfy $zr \equiv z'r \pmod{n}$, i.e., $(z - z')r \equiv 0 \pmod{n}$. This implies $(z - z') \equiv 0 \pmod{n}$ since n and r are coprime—so $z \equiv z' \pmod{n}$. Furthermore, since both z and z' only take on values between 0 and $n - 1$ (inclusive), this implies $z = z'$, which is a contradiction. Thus if $z \neq z'$, we know that $zr \not\equiv z'r \pmod{n}$. By the pigeonhole principle the set $\mathbb{Z}_n r$, modulo-n, must take on all values in $\mathbb{Z}_n$. Thus the result is proven. □

Furthermore, we note that the blocks allocated in (B.1) have the following interesting properties:

Lemma B.2. *Consider the blocks in $\bigcup_{i=0}^{q-1} \mathcal{B}_i$, i.e., the blocks of the form b_{iq+m} for $i = 0, 1, \ldots, q - 1$. Take a particular $m \in \{0, 1, \ldots, q - 1\}$. Then the $j = 0$ entry of block b_{iq+m} is the same element for all b_{iq+m}, $i = 0, 1, \ldots, q - 1$.*

Proof. For a particular $m \in \{0, 1, \ldots, q - 1\}$, the $j = 0$ entry of block b_{iq+m} is $e_{mq(iq+1)} = e_{mq}$ for all i. □

Lemma B.3. *Consider the blocks in $\bigcup_{i=0}^{q-1} \mathcal{B}_i$, i.e., the blocks of the form b_{iq+m} for $i = 0, 1, \ldots, q - 1$, and choose one block from each of $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{q-1}$. Call the block from $\mathcal{B}_i$ as b_{iq+m_i}. Except for the $j = 0$ entry, the set of all stride lengths to get every other element in the blocks $\bigcup_{i=0}^{q-1} b_{iq+m_i}$ is $\mathbb{Z}_{q^2} \setminus \{tq \mid t = 0, 1, \ldots, q - 1\}$; that is, all possible integers except multiples of q.*

Proof. For a given $j \neq 0$, we know from Lemma B.1 that $\{ji \mid i = 0, 1, \ldots, q-1\} \equiv \mathbb{Z}_q \pmod{q}$. For block b_{iq+m_i}, over $j \neq 0$ we have the elements $\{e_{(m_i q+j)(iq+1)} \mid j \neq 0\}$; this results in the strides $\{jiq + j \pmod{q^2} \mid j \neq 0\}$ from the $j = 0$ entry, in block b_{iq+m_i} (note that $e_{(m_i q+j)(iq+1)} = e_{m_i q+jiq+j}$ and that the $j = 0$ entry is $e_{m_i q}$). Over all the chosen blocks (by considering $i = 0, 1, \ldots, q - 1$), we have the strides

$$\bigcup_{i=0}^{q-1} \{jiq + j \mid j \neq 0\} \equiv \mathbb{Z}_{q^2} \setminus \{tq \mid t = 0, 1, \ldots, q - 1\} \pmod{q^2}.$$

Because there are $q(q-1)$ possible strides within these blocks and also q^2-q possible values in $\mathbb{Z}_{q^2} \setminus \{tq \mid t = 0, 1, \ldots, q-1\}$, so the set of strides in $\bigcup_{i=0}^{q-1} b_{iq+m_i}$ is unique (by the pigeonhole principle). □

Now, the following set of results shows that the element allocation yields an affine plane design.

Lemma B.4. *By allocating elements to blocks according to* (B.1) *and* (B.2)*, each block will have $k = q$ elements, and each element will have $l = q + 1$ replicas.*

Proof. It is clear that each block has exactly $k = q$ elements. Next we want to show that each element will have $l = q + 1$ replicas.

Consider a given $i \in \{0, 1, \ldots, q-1\}$. The blocks $\mathcal{B}_i = \bigcup_{m=0}^{q-1} b_{iq+m}$ collectively contain the elements

$$\bigcup_{m=0}^{q-1} \{e_{(mq+j)(iq+1)} \mid j = 0, 1, \ldots, q-1\} = \{e_{a(iq+1)} \mid a = 0, 1, \ldots, q^2-1\}.$$

Because $iq + 1$ is coprime with q^2 when q is prime, by Lemma B.1,

$$\{a(iq+1) \mid a = 0, 1, \ldots, q^2-1\} \equiv \{a \mid a = 0, 1, \ldots, q^2-1\} \pmod{q^2}.$$

Thus each element e_a, $a = 0, 1, \ldots, q^2-1$, occurs exactly once within the blocks of $\mathcal{B}_i$. Over all $i \in \{0, 1, \ldots, q-1\}$, each element has exactly q replicas, that is, each element has exactly q replicas within the blocks $\bigcup_{i=0}^{q-1} \mathcal{B}_i$.

Now consider the blocks $\mathcal{B}_q = \{b_{q^2+m} \mid m = 0, 1, \ldots, q-1\}$. Because

$$\bigcup_{m=0}^{q-1} \{e_{m+jq} \mid j = 0, 1, \ldots, q-1\} = \{e_a \mid a = 0, 1, \ldots, q^2-1\},$$

each element e_a, $a = 0, 1, \ldots, q^2-1$, occurs exactly once within $\mathcal{B}_q$.

Therefore, in total across all the blocks $\mathcal{B}$, each element e_a has exactly $l = q + 1$ replicas. □

Lemma B.5. *By allocating elements to blocks according to* (B.1) *and* (B.2)*, no pair of elements occurs in more than one block.*

Proof. Suppose two blocks b_h and $b_{h'}$ contain the same two elements e_a and $e_{a'}$. From the proof of Lemma B.4, we know that e_a and $e_{a'}$ each occur only once within any $\mathcal{B}_i$; thus we know that $b_h \in \mathcal{B}_i$ and $b_{h'} \in \mathcal{B}_{i'}$, where $i \neq i'$. This tells us that b_h and $b_{h'}$ are blocks where the elements are allocated according to different stride lengths. Let s be the stride length within block b_h, and s' be the stride length within block $b_{h'}$ (so $s \neq s'$). We want to consider the various scenarios for s and s'.

Suppose $i, i' \in \{0, 1, \ldots, q-1\}$; then $s = iq + 1$ and $s' = i'q + 1$. Because $i \neq i'$, by Lemma B.3 if e_a and $e_{a'}$ are separated by some stride length in block b_h, then that stride length does not also occur in block $b_{h'}$. Thus e_a and $e_{a'}$ cannot occur together in any two blocks from $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_{q-1}$, and so there can be no overlapping pairs of elements among any of the b_{iq+m} ($i \in \{0, 1, \ldots, q-1\}$, $m \in \{0, 1, \ldots, q-1\}$) blocks.

Now suppose $i \in \{0, 1, \ldots, q-1\}$ and $i' = q$, so we want to consider any overlapping pairs of elements between some block b_{iq+m} and $b_{q^2+m'}$. From Lemma B.3, we know that the stride lengths in b_{iq+m} are from $\mathbb{Z}_{q^2} \setminus \{tq \mid t = 0, 1, \ldots, q-1\}$, whereas the stride lengths in $b_{q^2+m'}$ are multiples of q, i.e., taken from $\{tq \mid t = 0, 1, \ldots, q-1\}$. Thus if $b_{iq+m} \in \mathcal{B}_i$ and $b_{q^2+m'} \in \mathcal{B}_q$ were to share some element e_a, then the two blocks would not share any other element in common.[1]

Therefore, no pair of elements can occur in more than one block. □

Theorem B.6. *By allocating elements to blocks according to* (B.1) *and* (B.2)*, each block will have $k = q$ elements, each element will have $l = q + 1$ replicas, and no pair of elements occurs in more than one block*

Proof. Follows from Lemmas B.4 and B.5. □

[1] This can also be seen more succinctly because the stride lengths within b_{iq+m} take on the form $t(iq+1)$, $t \in \{-(q-1), \ldots, -1, 0, 1, \ldots, q-1\}$, while those within $b_{q^2+m'}$ take on the form tq, $t \in \{-(q-1), \ldots, -1, 0, 1, \ldots, q-1\}$. Then

$$\{tq \mid t = -(q-1), \ldots, q-1\} \cap \{t(iq+1) \mid t = -(q-1), \ldots, q-1\} = \{0\}$$

since $iq + 1$ is relatively prime to q for q prime, so b_{iq+m} and $b_{q^2+m'}$ could not share more than one element in common.

From Theorem B.6, we know that the rules (B.1) and (B.2) provide an algebraic method for allocating elements to blocks that results in the affine plane designs with $k = q$ elements per block and $l = q + 1$ replicas per block (where q is prime).

Appendix C

NP-Completeness of Non-Uniform Demand Problem

We give an alternative proof of NP-completeness of the non-uniform demand network coding problem, via a polynomial reduction from a general graph coloring problem. Unlike [16], in which the network demand problems shown to be NP-hard do not have fully saturated demands, our proof considers sink demands in which saturation must occur. The coloring problem that we consider is the following: Given an undirected graph $\hat{G} = (\hat{V}, \hat{E})$, is there a coloring using n [or fewer] colors? We first give the reduction in Algorithm C.1, and then follow with a proof that the reduction leads to an equivalent problem.

Step 1 of Algorithm C.1 sets up most of the network graph. Overlaps between paths reflect the fact that a vertex cannot have two different colors. The addition of $|V|$ sinks in step 2 forces the stream assignment algorithm to assign the same color to all paths crossing through the same link v; otherwise, in the sinks with two paths, it may be possible that the corresponding stream will be assigned to the path in the pair that does not overlap at the considered link. Furthermore, the single sink with n paths in step 3 guarantees that at least n different streams will be assigned.

Lemma C.1. *Performing non-uniform demand stream assignment on the network*

Algorithm C.1 A reduction from general graph coloring to non-uniform demand stream assignment

Require: Undirected graph $\hat{G} = (\hat{V}, \hat{E})$ to be colored

1: For each edge $e_j = (v_j, w_j) \in \hat{E}$, construct a sink j in the network graph consisting of two paths $p_{j,1}$ and $p_{j,2}$. If two edges $e_j = (v, w_j) \in \hat{E}$ and $e_i = (v, w_i) \in \hat{E}$ share a vertex v, then force the paths $p_{j,1}$ and $p_{i,1}$ to overlap at some link. Call the overlapping link in the network graph by v. (If the shared vertex is w such that $e_j = (v_j, w) \in \hat{E}$ and $e_i = (v_i, w) \in \hat{E}$, then force paths $p_{j,2}$ and $p_{i,2}$ to overlap at some link w.) Thus, vertices in the coloring graph determine the intersections of paths in the network graph—where the link of intersection occurs according to the vertex in the coloring graph.

2: Introduce another $|V|$ sinks, with only a single path to each sink. Label these sinks $1, 2, \ldots, |V|$. For a particular sink v, call the single path p_v, and make path p_v intersect with all other paths that cross through link v in the network graph.

3: Introduce one additional sink, with n paths. These n paths do not intersect any paths defined in prior steps.

4: Solve the non-uniform demand stream assignment problem on the resulting network graph.

digraph G formed from Algorithm C.1 is equivalent to coloring the original undirected graph $\hat{G}$.

Proof. First we show that if there exists a coloring solution for $\hat{G}$ using n colors, then there will also be a non-uniform demand stream assignment on the constructed network graph with n streams. To do so, start with a coloring solution. For a particular vertex v in the coloring graph, assign the stream corresponding to the color of vertex v to all paths that intersect at the associated link v in the network graph. Because no path has more than one link with overlap, then there is no ambiguity about the stream that is assigned to that path. Because graph coloring guarantees that the vertices connected by an edge will have different colors, each sink from step 1 will receive two paths that have different stream assignments. Thus, the solution is saturating. Not only that, in the network graph, any intersecting paths intersect only at one link, so contamination is mitigated by assigning the same stream to all paths that intersect at the same edge. Thus, the solution to the graph coloring with n

colors gives a non-uniform demand stream assignment for the constructed network graph using exactly n distinct streams. This stream assignment is both saturating and decodable.

Conversely, assume that there exists a saturating and decodable stream assignment to the non-uniform demand stream assignment problem on the constructed network graph, which uses exactly n streams. Then this solution can be used to determine a graph coloring of the original graph, with n colors. To prove this, first consider the sinks that have single paths. From these sinks, suppose path p_v to sink v (associated with vertex v in the coloring graph) is assigned stream c. By decodability, any other paths that intersect path p_v must be carrying stream c. That is, for a path p'_j that intersects p_v, then its pair path p''_j (i.e., toward the same sink) is not the path of the pair that is carrying stream c. Otherwise, sink v would need to decode out the stream on path p'_j (which would be some $c' \neq c$), but sink v cannot, since it is only assigned to receive stream c from the single path p_v. Thus, all paths associated with the same vertex v in the original coloring graph must have the same stream assignment; this is the color assigned to vertex v. Now, consider the sinks with paired paths. Because the non-uniform demand stream assignment solution is saturating, that means that the two paths are assigned different streams. This is equivalent to the requirement that vertices that are connected by an edge in the original coloring graph be assigned different colors. Thus, the stream assignment on G using n streams gives a coloring on $\hat{G}$ using n colors. □

From the preceding construction, it is straightforward to determine the complexity of the stream assignment problem.

Theorem C.2. *The non-uniform demand stream assignment problem with saturated demands is* NP*-complete.*

Proof. First we show that the problem is in NP, by showing that it takes polynomial time to check if a given solution to the non-uniform demand stream assignment problem is feasible. For every overlap link in the network graph G, we want to check if the sinks receiving the paths crossing through the link have additional streams assigned

to them that are the same as all the contaminations from that link. This takes at most $O(n|E|)$ time, where $|E|$ is the number of links in the corresponding network graph (and $|E|$ can be as small as $4|\hat{E}| + 3|\hat{V}| + n$).

Next, we show that the transformation from the coloring graph to the related network graph given in Algorithm C.1 is polynomial. The network graph has $|\hat{E}| + |\hat{V}|+1$ sinks (more precisely, it has $2|\hat{E}|+|\hat{V}|+n$ paths), so the reduction is polynomial. Because any instance of graph coloring can be polynomially reduced to a non-uniform demand stream assignment problem, and solutions can be checked in polynomial time, the non-uniform demand stream assignment problem is NP-complete. □

From the above arguments, we also conclude that non-uniform demand stream assignment can be solved in polynomial time when $n = 2$ (i.e., when the maximum data rate to any sink is upper bounded by 2). This is because graph coloring is polynomial time (i.e., by searching for bipartiteness in the graph) when only 2 colors are allowed [37].

Appendix D

Markov Chain Analysis of First-to-Complete Alignment

We provide more details on computing the absorption probabilities and hitting times from the Markov chain constructions of Section 4.4, using techniques from [81]. Assume there are a total of n states in the Markov chain, with k transient states and $n-k$ absorbing states. In the rest of this appendix, let $\boldsymbol{e}_i$ denote a vector consisting of all 0's except for a 1 in the i-th position (i.e., $\boldsymbol{e}_i$ is the canonical basis vector in the i-th direction). We let state $i = 0$ be the initial state of the Markov chain—with no alignment sets completed—so $\boldsymbol{e}_0$ is the initial probability distribution. Also, let $\mathbf{1}$ be the all-ones vector (of appropriate length).

Consider the $n \times n$ probability transition matrix $\boldsymbol{P}$, with the P_{ij} entry denoting the probability of transitioning from state i to state j. Without loss of generality, we may re-order the states so that the transient states are indexed first, and then followed by the absorbing states. Equivalently, we permute the rows and columns of $\boldsymbol{P}$ to have the block upper-triangular form $\boldsymbol{P} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{R} \\ \mathbf{0} & \boldsymbol{I} \end{bmatrix}$, where the block $\boldsymbol{Q}$ (of size $k \times k$) corresponds to transition probabilities between transient states and the block $\boldsymbol{R}$ (of size $k \times (n-k)$) corresponds to transition probabilities from transient states to absorbing states. (The lower-right block is the identity matrix since an absorbing state can transition only to itself, and obviously the lower-left block is all zeros since

absorbing states cannot transition to transient states.) As an example, if we consider the Markov chain of Figure 4.1 with re-ordered state vector $\boldsymbol{s} = (s_0, s_1, s_2, s_3, s_{-1})$, then the permuted probability transition matrix is

$$\boldsymbol{P} = \begin{bmatrix} 1 - \frac{4}{|\mathcal{H}|} & \frac{3}{|\mathcal{H}|} & 0 & 0 & \frac{1}{|\mathcal{H}|} \\ 0 & 1 - \frac{3}{|\mathcal{H}|} & \frac{2}{|\mathcal{H}|} & 0 & \frac{1}{|\mathcal{H}|} \\ 0 & 0 & 1 - \frac{2}{|\mathcal{H}|} & \frac{1}{|\mathcal{H}|} & \frac{1}{|\mathcal{H}|} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

which evidently has the appropriate structure.

Expressions for the absorption probabilities and hitting times can be derived using the various blocks of the probability transition matrix.

Lemma D.1. *Define the length-$(n-k)$ absorption probability vector $\boldsymbol{\beta}$, where the β_j entry is the probability of becoming absorbed in state j. Then*

$$\boldsymbol{\beta} = (\boldsymbol{e}_0^T(\boldsymbol{I} - \boldsymbol{Q})^{-1}\boldsymbol{R})^T = \boldsymbol{R}^T(\boldsymbol{I} - \boldsymbol{Q}^T)^{-1}\boldsymbol{e}_0.$$

Proof. We consider the $k \times (n-k)$ transient-to-absorbing matrix $\boldsymbol{U}$, where the U_{ij} entry denotes the probability of starting in transient state i and ultimately becoming absorbed in absorbing state j. By first-step analysis, $\boldsymbol{U}$ satisfies the recursion $\boldsymbol{U} = \boldsymbol{Q}\boldsymbol{U} + \boldsymbol{R}$, so $\boldsymbol{U} = (\boldsymbol{I} - \boldsymbol{Q})^{-1}\boldsymbol{R}$. Because $\boldsymbol{Q}$ represents the probabilities of transitioning between transient states, $(\boldsymbol{I} - \boldsymbol{Q})^{-1}$ is the fundamental matrix and is well-defined. Then β_j is the probability of starting in state 0 and eventually becoming absorbed in state j, so $\boldsymbol{\beta} = (\boldsymbol{e}_0^T\boldsymbol{U})^T = (\boldsymbol{e}_0^T(\boldsymbol{I} - \boldsymbol{Q})^{-1}\boldsymbol{R})^T$. □

Lemma D.2. *The hitting time (i.e., the time until absorption in* any *absorption state) is given by*

$$d = \boldsymbol{e}_0^T(\boldsymbol{I} - \boldsymbol{Q})^{-1}\boldsymbol{1}.$$

Proof. Let $\boldsymbol{D}$ be the length-k vector where the D_i entry is the hitting time when starting in transient state i. Then first-step analysis gives the recursion $\boldsymbol{D} = \boldsymbol{Q}\boldsymbol{D} + \boldsymbol{1}$, so $\boldsymbol{D} = (\boldsymbol{I} - \boldsymbol{Q})^{-1}\boldsymbol{1}$. The overall hitting time is then $d = \boldsymbol{e}_0^T\boldsymbol{D} = \boldsymbol{e}_0^T(\boldsymbol{I} - \boldsymbol{Q})^{-1}\boldsymbol{1}$. □

Suppose one employs the first-to-complete alignment scheme with alignment sets of sizes $I = (m_1, m_2, \ldots, m_{|I|})$, and where the first alignment set has size $m_1 = 2$. Here we prove that the mean time to absorption of the Markov chain is equal to the number of possible channel fading matrices multiplied by the probability of completion using the set of size 2.

Theorem D.3. *If* $2 \in I$*, then* $d^I = \beta_2^I |\mathcal{H}|$.

Proof. Let $\hat{\jmath}$ be the state associated with the realization of the channel complement (i.e., the state associated with completing the size-2 alignment set). We assume that each channel realization is equally likely with probability $1/|\mathcal{H}|$, so the probability of transitioning into state $\hat{\jmath}$ is $1/|\mathcal{H}|$ starting from any [transient] state. From Lemma D.1 and since the $\hat{\jmath}$-th column of $\boldsymbol{R}$ is $(1/|\mathcal{H}|)\mathbf{1}$, we see that $\beta_2^I = \beta_{\hat{\jmath}} = (1/|\mathcal{H}|)\boldsymbol{e}_0^T(\boldsymbol{I} - \boldsymbol{Q})^{-1}\mathbf{1}$. Since $d^I = \boldsymbol{e}_0^T(\boldsymbol{I} - \boldsymbol{Q})^{-1}\mathbf{1}$, the result follows. □

Bibliography

[1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[2] Apache Software Foundation. (2011, Oct. 18) Hadoop MapReduce. [Online]. Available: http://hadoop.apache.org/mapreduce/

[3] ——. (2011, Aug. 25) HDFS architecture guide. [Online]. Available: http://hadoop.apache.org/common/docs/current/hdfs_design.html

[4] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Looking up data in P2P systems," *Communications of the ACM*, vol. 46, no. 2, pp. 43–48, Feb. 2003.

[5] T. Beth, D. Jungnickel, and H. Lenz, *Design Theory*, 2nd ed., ser. Encyclopedia of Mathematics and its Applications. Cambridge: Cambridge University Press, 1999, vol. 1.

[6] ——, *Design Theory*, 2nd ed., ser. Encyclopedia of Mathematics and its Applications. Cambridge: Cambridge University Press, 1999, vol. 2.

[7] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total Recall: System support for automated availability management," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, Mar. 29 – 31, 2004, pp. 337–350.

[8] R. E. Blahut, *Algebraic Codes for Data Transmission*. Cambridge: Cambridge University Press, 2003.

[9] A. Bloch, *Murphy's Law and Other Reasons Why Things Go Wrong*. Los Angeles: Price/Stern/Sloan Publishers, 1977.

[10] J. A. Bondy and U. S. R. Murty, *Graph Theory*, ser. Graduate Texts in Mathematics. New York: Springer, 2008.

[11] V. R. Cadambe and S. A. Jafar, "Can 100 speakers talk for 30-minutes each in one room within one hour and with zero interference to each other's audience?" in *Proceedings of the 45th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Sep. 26 – 28, 2007.

[12] ——, "Interference alignment and degrees of freedom of the K-user interference channel," *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3425–3441, Aug. 2008.

[13] ——, "Multiple access outerbounds and the inseparability of parallel interference channels," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, New Orleans, LA, Nov. 30 – Dec. 4, 2008.

[14] ——, "Parallel Gaussian interference channels are not always separable," *IEEE Transactions on Information Theory*, vol. 55, no. 9, pp. 3983–3990, Sep. 2009.

[15] P. J. Cameron, *Combinatorics: Topics, Techniques, Algorithms*. Cambridge: Cambridge University Press, 1994.

[16] Y. Cassuto and J. Bruck, "Network coding for non-uniform demands," in *Proceedings of the IEEE International Symposium on Information Theory*, Adelaide, Australia, Sep. 4 – 9, 2005, pp. 1720–1724.

[17] K.-C. Chang and S. M. Ross, "The multiple subset coupon collecting problem," *Probability in the Engineering and Informational Sciences*, vol. 21, no. 3, pp. 435–440, Jul. 2007.

[18] C. Chekuri, C. Fragouli, and E. Soljanin, "On achievable information rates in single-source non-uniform demand networks," in *Proceedings of the IEEE International Symposium on Information Theory*, Seattle, Jul. 9 – 14, 2006, pp. 773–777.

[19] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour, and K. Vušković, "Recognizing Berge graphs," *Combinatorica*, vol. 25, no. 2, pp. 143–186, Mar. 2005.

[20] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas, "The strong perfect graph theorem," *Annals of Mathematics*, vol. 164, no. 1, pp. 51–229, 2006.

[21] C. J. Colbourn and J. H. Dinitz, Eds., *The Handbook of Combinatorial Designs*, 2nd ed., ser. Discrete Mathematics and its Applications. Boca Raton: Chapman & Hall/CRC Press, 2007.

[22] C. J. Colbourn and P. C. van Oorschot, "Applications of combinatorial designs in computer science," *ACM Computing Surveys*, vol. 21, no. 2, pp. 223–250, Jun. 1989.

[23] W. V. Courtright II, G. Gibson, M. Holland, L. Neal Reilly, and J. Zelenka, "RAIDframe: A rapid prototyping tool for RAID systems," Carnegie Mellon University, Tech. Rep. CMU-CS-97-142, Jun. 4 1997. [Online]. Available: http://www.pdl.cmu.edu/RAIDframe/

[24] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, Mar. 29 – 31, 2004, pp. 85–98.

[25] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, San Francisco, Dec. 6 – 8, 2004, pp. 137–150.

[26] J. Dénes and A. D. Keedwell, *Latin Squares and their Applications*. New York: Academic Press, 1974.

[27] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[28] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Ubiquitous access to distributed data in large-scale sensor networks through decentralized erasure codes," in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN)*, Los Angeles, Apr. 25 – 27, 2005, pp. 111–117.

[29] J. H. Dinitz and D. R. Stinson, Eds., *Contemporary Design Theory: A Collection of Surveys*, ser. Wiley-Interscience Series in Discrete Mathematics and Optimization. New York: John Wiley and Sons, 1992.

[30] R. Dougherty, C. Freiling, and K. Zeger, "Insufficiency of linear coding in network information flow," *IEEE Transactions on Information Theory*, vol. 51, no. 8, pp. 2745–2759, Aug. 2005.

[31] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *Proceedings of the 48th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Sep. 29 – Oct. 1, 2010, pp. 1510–1517.

[32] D. J. Ellard, "Trace-based analyses and optimizations for network storage servers," Ph.D. dissertation, Harvard University, Cambridge, MA, May 2004. [Online]. Available: http://www.eecs.harvard.edu/sos/papers/dje-thesis.pdf

[33] R. H. Etkin, D. N. C. Tse, and H. Wang, "Gaussian interference channel capacity to within one bit," *IEEE Transactions on Information Theory*, vol. 54, no. 12, pp. 5534–5562, Dec. 2008.

[34] L. R. Ford, Jr. and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[35] C. Fragouli and E. Soljanin, "Information flow decomposition for network coding," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 829–848, Mar. 2006.

[36] H. Garcia-Molina and D. Barbara, "How to assign votes in a distributed system," *Journal of the ACM*, vol. 32, no. 4, pp. 841–860, Oct. 1985.

[37] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman, 1979.

[38] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, NY, Oct. 19 – 22, 2003, pp. 29–43.

[39] S. Gilbert, N. A. Lynch, and A. A. Shvartsman, "RAMBO: A robust, reconfigurable atomic memory service for dynamic networks," *Distributed Computing*, vol. 23, no. 4, pp. 225–272, Dec. 2010.

[40] G. Grider, "Exa-yotta-yotta-yotta...For checkpoint only," in *Supercomputing 2008 Panel: Exa and Yotta Scale Data — Are We Ready?*, Austin, TX, Nov. 21, 2008. [Online]. Available: http://www.pdsi-scidac.org/events/PDSW08/resources/slides/Grider-Exa-Yotta-yotta-yotta.pdf

[41] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "Polynomial time algorithms for multicast network code construction," *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[42] M. Holland, G. A. Gibson, and D. P. Siewiorek, "Architectures and algorithms for on-line failure recovery in redundant disk arrays," *Distributed and Parallel Databases*, vol. 2, no. 3, pp. 295–335, Jul. 1994.

[43] S. Hoory, "The size of bipartite graphs with a given girth," *Journal of Combinatorial Theory, Series B*, vol. 86, no. 2, pp. 215–220, Nov. 2002.

[44] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. M. G. M. Tolhuizen, "Polynomial time algorithms for multicast network code construction," *IEEE Transactions on Information Theory*, vol. 51, no. 6, pp. 1973–1982, Jun. 2005.

[45] S.-W. Jeon and S.-Y. Chung, "Capacity of a class of linear binary field multi-source relay networks," *IEEE Transactions on Information Theory*, submitted for publication. [Online]. Available: http://arxiv.org/abs/0907.2510

[46] O. Johnson, M. Aldridge, and R. Piechocki, "Delay-rate tradeoff in ergodic interference alignment," 2010. [Online]. Available: http://arxiv.org/abs/1004.0208

[47] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web," in *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, El Paso, TX, May 4 – 6, 1997, pp. 654–663.

[48] D. Karger, R. Motwani, and M. Sudan, "Approximate graph coloring by semidefinite programming," *Journal of the ACM*, vol. 45, no. 2, pp. 246–265, Mar. 1998.

[49] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[50] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[51] J. Le, J. C. S. Lui, and D. M. Chiu, "DCAR: Distributed coding-aware routing in wireless networks," in *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS)*, Beijing, Jun. 17 – 20, 2008, pp. 462–469.

[52] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *IEEE Transactions on Information Theory*, vol. 49, no. 2, pp. 371–381, Feb. 2003.

[53] C. C. Lindner and C. A. Rodger, *Design Theory*, 2nd ed. Boca Raton: Chapman & Hall/CRC Press, 2009.

[54] D. Lun, M. Médard, T. Ho, and R. Koetter, "Network coding with a cost criterion," in *Proceedings of the International Symposium on Information Theory and its Applications*, Parma, Italy, Oct. 10 – 13, 2004, pp. 1232–1237.

[55] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam: North-Holland Publishing Company, 1977.

[56] M. A. Maddah-Ali, A. S. Motahari, and A. K. Khandani, "Communication over MIMO X channels: Interference alignment, decomposition, and performance analysis," *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3457–3470, Aug. 2008.

[57] M. Maekawa, "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," *ACM Transactions on Computer Systems*, vol. 3, no. 2, pp. 145–159, May 1985.

[58] M. Médard, M. Effros, D. Karger, and T. Ho, "On coding for non-multicast networks," in *Proceedings of the 41st Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Oct. 1 – 3, 2003, pp. 21–29.

[59] O. Milenkovic, N. Kashyap, and D. Leyba, "Shortened array codes of large girth," *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3707–3722, Aug. 2006.

[60] R. R. Muntz and J. C. S. Lui, "Performance analysis of disk arrays under failure," in *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB)*, Brisbane, Australia, Aug. 13 – 16, 1990, pp. 162–173.

[61] B. Nazer, M. Gastpar, S. A. Jafar, and S. Vishwanath, "Ergodic interference alignment," in *Proceedings of the IEEE International Symposium on Information Theory*, Seoul, Korea, Jun. 28 – Jul. 3, 2009, pp. 1769–1773.

[62] ——, "Interference alignment at finite SNR: General message sets," in *Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Sep. 30 – Oct. 2, 2009, pp. 843–848.

[63] C. K. Ngai and R. W. Yeung, "Multisource network coding with two sinks," in *Proceedings of the International Conference on Communications, Circuits and Systems (ICCCAS)*, vol. 1, Chengdu, Jun. 27 – 29, 2004, pp. 34–37.

[64] ——, "Network coding gain of combination networks," in *Proceedings of the 2004 Information Theory Workshop (ITW)*, San Antonio, TX, Oct. 24 – 29, 2004, pp. 283–287.

[65] M. O'Keefe and P. K. Wong, "The smallest graph of girth 6 and valency 7," *Journal of Graph Theory*, vol. 5, no. 1, pp. 79–85, Spring 1981.

[66] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Chicago, Jun. 1 – 3, 1988, pp. 109–116.

[67] S. Pawar, N. Noorshams, S. El Rouayheb, and K. Ramchandran, "DRESS codes for the storage cloud: Simple randomized constructions," in *Proceedings of the IEEE International Symposium on Information Theory*, St. Petersburg, Russia, Jul. 31 – Aug. 5, 2011, pp. 2338–2342.

[68] A. Ramamoorthy and R. D. Wesel, "The single source two terminal network with network coding," in *Proceedings of the 9th Canadian Workshop on Information Theory*, Montreal, Jun. 2005.

[69] A. Rasala Lehman and E. Lehman, "Complexity classification of network information flow problems," in *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, New Orleans, LA, Jan. 11 – 14, 2004, pp. 142–150.

[70] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proceedings of the 47th Annual Allerton Conference on Communication, Control, and Computing*, Monticello, IL, Sep. 30 – Oct. 2, 2009, pp. 1243–1249.

[71] ——, "Explicit and optimal exact-regenerating codes for the minimum-bandwidth point in distributed storage," in *Proceedings of the IEEE International Symposium on Information Theory*, Austin, TX, Jun. 13 – 18, 2010, pp. 1938–1942.

[72] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, San Diego, CA, Aug. 27 – 31, 2001, pp. 161–172.

[73] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *IEEE Internet Computing*, vol. 5, no. 5, pp. 40–49, Sep./Oct. 2001.

[74] S. M. Ross, *Introduction to Probability Models*, 10th ed. Boston: Academic Press, 2010.

[75] 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. 12 – 16, 2001, pp. 329–350.

[76] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence, "FAB: Building distributed enterprise disk arrays from commodity components," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI)*, Boston, Oct. 7 – 13, 2004, pp. 48–58.

[77] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, San Diego, CA, Aug. 27 – 31, 2001, pp. 149–160.

[78] Storage Networking Industry Association. (2009, Mar. 27) Common RAID Disk Data Format (DDF) specification. [Online]. Available: http://www.snia.org/tech_activities/standards/curr_standards/ddf

[79] ——. (2011) Network File System traces. IOTTA Repository. [Online]. Available: http://iotta.snia.org/tracetypes/2

[80] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[81] H. M. Taylor and S. Karlin, *An Introduction to Stochastic Modeling*, 3rd ed. San Diego: Academic Press, 1998.

[82] E. Upfal and A. Widgerson, "How to share memory in a distributed system," *Journal of the ACM*, vol. 34, no. 1, pp. 116–127, Jan. 1987.

[83] B. Vasic, "Structured iteratively decodable codes based on Steiner systems and their application in magnetic recording," in *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, San Antonio, TX, Nov. 25 – 29, 2001, pp. 2954–2960.

[84] C.-C. Wang and N. B. Shroff, "Beyond the butterfly: A graph-theoretic characterization of the feasibility of network coding with two simple unicast sessions," in *Proceedings of the IEEE International Symposium on Information Theory*, Nice, France, Jun. 24 – 29, 2007, pp. 121–125.

[85] R. M. Wilson, "An existence theory for pairwise balanced designs, I: Composition theorems and morphisms," *Journal of Combinatorial Theory, Series A*, vol. 13, no. 2, pp. 220–245, Sep. 1972.

[86] ——, "An existence theory for pairwise balanced designs, II: The structure of PBD-closed sets and the existence conjectures," *Journal of Combinatorial Theory, Series A*, vol. 13, no. 2, pp. 246–273, Sep. 1972.

[87] ——, "An existence theory for pairwise balanced designs, III: Proof of the existence conjectures," *Journal of Combinatorial Theory, Series A*, vol. 18, no. 1, pp. 71–79, Jan. 1975.

[88] P.-K. Wong, "Cages—A survey," *Journal of Graph Theory*, vol. 6, no. 1, pp. 1–22, Spring 1982.

[89] R. W. Yeung, *Information Theory and Network Coding*. New York, NY: Springer, 2008.

[90] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, Jan. 2004.