Scalable Constructions of Fractional Repetition Codes in Distributed Storage Systems

Joseph C. Koo and John T. Gill, III
Department of Electrical Engineering, Stanford University
Email: {`jckoo,gill`}`@stanford.edu`

***Abstract*—In distributed storage systems built using commodity hardware, it is necessary to have data redundancy in order to ensure system reliability. In such systems, it is also often desirable to be able to quickly repair storage nodes that fail. We consider a scheme—introduced by El Rouayheb and Ramchandran—which uses combinatorial block design in order to design storage systems that enable efficient (and exact) node repair. In this work, we investigate systems where node sizes may be much larger than replication degrees, and explicitly provide algorithms for constructing these storage designs. Our designs, which are related to projective geometries, are based on the construction of bipartite cage graphs (with girth 6) and the concept of mutually-orthogonal Latin squares. Via these constructions, we can guarantee that the resulting designs require the fewest number of storage nodes for the given parameters, and can further show that these systems can be easily expanded without need for frequent reconfiguration.**

I. Introduction

Recent trends in distributed storage systems have been toward the use of commodity hardware as storage nodes, where nodes may be individually unreliable. Such systems can still be feasible for large-scale storage as long as there is overall reliability of the entire storage system. Recent research in distributed storage systems has focused on using techniques from coding theory to increase storage efficiency, without sacrificing system reliability and node repairability [1].

In this work, we consider 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* [1]—where replacement nodes 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 [2], who propose a storage system allowing for exact repair. Using the idea of Steiner systems [3], 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 [2] stores data in an uncoded manner, for computing applications the storage nodes may also serve as processing nodes.

A Steiner system $S(t, k, v)$ specifies a distribution of v elements into blocks of size k such that the maximum number of overlapping elements between any two blocks is $t-1$ (so if $t = 2$, then no two blocks can share any pairs of elements[1]). For instance, Example 1 shows a Steiner system and the resulting distribution of data chunks to storage nodes.

Example 1. *Consider a distributed storage system to store* 9 *total data chunks, where each chunk is stored within storage nodes that can hold* 3 *chunks each. Then it is possible to distribute the chunks across* 12 *nodes, where every chunk has exactly* 4 *replicas and any two distinct nodes share at most only one overlapping chunk. This is shown in Figure 1.*

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

Fig. 1. Storage design from Steiner system $S(2, 3, 9)$; same as [2, Fig. 6(a)].

In most practical distributed storage systems, however, 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 [4]—which stores data in chunks of as small as 64 MB each—has a replication degree on the order of three replicas but may store thousands of chunks on each storage node. Thus in this work, we consider a graph-based construction of Steiner systems where the replication degree and node size are significantly asymmetric.

Specifically, we construct storage systems where the replication degree of each data chunk is $q+1$, whereas each node may store up to $q^n + q^{n-1} + \cdots + q^2 + q + 1$ chunks (for any given integer n). Although it is known from the theory of projective geometries [3] that systems with these parameters can be designed, by using our graph-based method we are able to give a systematic construction that is highly scalable;

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

[2]For brevity, we refer to the number of chunks per node as the *node size*.

for a system constructed according to the methods in this paper, it is always possible to increase the storage system without moving any existing data chunks—and still be able to preserve the property that no pairs of chunks recur in more than one storage node.

Our construction is based on relating Steiner system problems with the problem of shortest cycles on bipartite graphs. More specifically, our systems arise from the construction of *cage graphs* [5], 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 constructions are the *smallest* possible systems (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.

A. Related Work

The problem of distributed storage with efficient repair is discussed in [1]. Using network coding, the authors propose a scheme for storing data where node repair is functional. Dimakis et al. [1] also define the idea of a storage-bandwidth tradeoff, and discuss ways to implement either minimum storage or minimum bandwidth systems. Even though exact repair of storage nodes is sometimes necessary, the storage-bandwidth tradeoff under exact repair is not yet fully understood. Building upon the network coding constructions of [1], Rashmi et al. [6] give a scheme for achieving the minimum bandwidth operating point under exact repair, finding a point on the storage-bandwidth tradeoff curve.

El Rouayheb and Ramchandran [2] introduce a related scheme, termed *fractional repetition codes*, which can perform exact repair for the minimum bandwidth regime. They then derive information theoretic bounds on the storage capacity of such systems with the given repair requirements. Although their repair model is table-based (instead of random access as in [1]), the scheme of [2] has the favorable characteristics of exact repair and the uncoded storage of data chunks. Randomized constructions of such schemes are investigated in [7].

Uncoded storage has numerous advantages for distributed storage systems. For instance, uncoded data at nodes allows for distributed computing (e.g., for cloud computing), by spreading out computation to the node(s) that contain the data to be processed. Upfal and Widgerson [8] consider a method for parallel computation by randomly distributing data chunks among multiple memory devices, and derive some asymptotic performance results. In contrast, our designs are deterministic, and we are also able to guarantee the smallest possible size for our storage system. Furthermore, if uncoded data chunks are distributed among the nodes according to Steiner systems, then load-balancing of computations is always possible.

Steiner systems are an example of balanced incomplete block design (BIBD), within the field of combinatorial design theory [9]. Some parameters for which Steiner systems can be designed are given in [10], [2]. In this work, we consider Steiner systems similar to those from finite projective planes. Specifically, designs in which the replication degree is $q+1$ and with each storage node storing up to $q^n + q^{n-1} + \cdots + q^2 + q + 1$ data chunks can also be found from the projective geometry $PG(n+1, q)$—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.[3] This alternate 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 designs are uniquely determined given knowledge of the base set of mutually-orthogonal Latin squares (which we discuss later).

In addition to [2], the use of BIBDs for guaranteeing load-balanced disk repair in distributed storage systems is also considered in [11], [12], for application to RAID-based disk arrays. In [12], the authors discuss how block designs may be used to lay out parity stripes in declustered parity RAID disk arrays. The block designs from our work may be helpful for distributing parity blocks in this scenario, in order to build disk arrays with good repair properties.

Certain block designs may also be applicable to the design of error-correcting codes, particularly in the construction of geometrical codes [13, Sections 2.5 and 13.8]. Graphs without short cycles have been considered in the context of Tanner graphs [14], and finite geometries in particular have been considered in the context of LDPC codes [15]. Block designs and their related bipartite graphs are also considered in code design for magnetic recording applications in [16].

B. Outline of Paper

In the next section, we provide necessary background. Section III illustrates how our constructions work, through the construction of regular bipartite cage graphs; this construction provides a base upon which the larger construction of Section IV is built. In Section IV we give the main contribution, which is the design of scalable storage systems that can be expanded readily. Finally, Section V concludes.

II. Preliminaries

A. Notation

When describing parameters for constructible graphs we let $p_n(q) = q^n + q^{n-1} + \cdots + q^2 + q + 1 = \frac{q^{n+1}-1}{q-1}$, for $n \in \mathbb{Z}_{++}$. In the rest of this paper, q will always denote either a prime number or a power of a prime.

[3]We do not describe this in detail, but very similar graph-based methods can also be used to construct designs related to the affine geometry $AG(n+1, q)$. These constructions are just as expandable as the projective geometry–based designs. A brief note on these constructions is given in Section IV-C.

In this work, we consider simple undirected bipartite graphs $G = (X, Y, E)$. Cardinality is denoted by $| \cdot |$. For a vertex x, $\deg(x)$ gives the number of incident edges. We only consider graphs where all of the vertices in a vertex set have the same degree, so we can write $\deg(X) = \deg(x)$ for some $x \in X$. The symbol $\sim$ is used to denote an edge; for vertices x and y, we say that $x \sim y$ if and only if $(x, y) \in E$.

B. Graph Interpretation of Steiner Systems

A Steiner system is a collection of elements, $\mathcal{V}$, into blocks, $\mathcal{B}$, where any subset of elements only occurs once in the block collection. We reinterpret the Steiner system requirements by considering its incidence graph [17]. In this work, we will consider bipartite graphs $G = (X, Y, E)$, where there are u vertices in X, each of degree k, and v vertices in Y, each of degree l. We call such a graph to be *biregular* when $k \neq l$. Clearly, $lv = uk$.

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 $b_h = \{y_g \in Y \mid y_g \sim x_h\}$. Then the collection of blocks, $\mathcal{B} = \{b_h \mid h = 0, 1, \ldots, u-1\}$, satisfies the following:

1) Each element $y_g \in \mathcal{V}$ occurs in exactly l blocks of $\mathcal{B}$.
2) Each block $b_h \in \mathcal{B}$ contains k elements.

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 the 4-cycle $x_h \sim y_g \sim x_{h'} \sim y_{g'} \sim x_h$. Thus the nonexistence of such 4-cycles is equivalent to the nonexistence of shared pairs of elements between blocks. In Figure 2, we show the bipartite graph associated with Example 1.

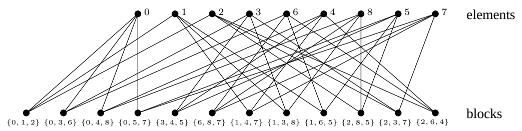


Fig. 2. Bipartite graph of Steiner system corresponding to $k = 3$, $l = 4$.

In the above, we construct Steiner systems where l is the repetition degree, k is the block size, v is the total number of elements, and u is the total number of blocks. In the rest of this paper, we shall always assume that $l \geq k$.[4]

Since X and Y are interchangeable, we could instead let X be the elements and Y be the blocks of another block system,[5] resulting in the *transpose codes* of [2]. 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. 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 number of elements and v is the total number of blocks.

C. 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. 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 minimum number of vertices for a particular desired degree distribution [5], [17]. The goal of this work is to construct biregular cages of girth 6 (so that no 4-cycles are present)—in order to construct the smallest possible Steiner system with the desired parameters. 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. Such systems will meet the lower bound of Lemma 1.[6]

Lemma 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*

$$v \geq 1 + l(k-1) \tag{1}$$
$$u \geq l + l(l-1)(k-1)/k. \tag{2}$$

Proof: We sketch the proof for (1) here; a more detailed proof of (1) as well as for (2) is provided in Appendix B-A.

One method for constructing the bipartite graph is by starting with a single vertex $y \in Y$ (called the *layer* 0 vertex) and connecting it to l vertices of X (called the *layer* 1 vertices). These vertices of X must be connected to $k-1$ distinct other vertices of Y (the *layer* 2 vertices). Note that any remaining vertices of X (the *layer* 3 vertices) would then need to be connected back to the layer 2 vertices of Y in such a way as to preserve the nonexistence of 4-cycles. ■

Any bipartite cage achieving the lower bounds of Lemma 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 $\binom{v}{2} = u\binom{k}{2}$,[7] we know that every pair of elements occurs in at least one block—and therefore occurs in only one block.

The proof of Lemma 1 gives us clues on how to construct bipartite graphs that achieve the lower bounds—which must necessarily be cage graphs. We will show how to avoid introducing 4-cycles between the layer 2 and layer 3 vertices, by considering the use of mutually-orthogonal Latin squares (see Appendix A or [20]). Specifically, in order to construct the cage graphs, we will require the existence of a set of q mutually-orthogonal $q \times q$ squares, $\{L^{(0)}, L^{(1)}, \ldots, L^{(q-1)}\}$, 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 or a prime power. We give an example for $q = 3$:

[4]We can construct systems where $k > l$ by swapping the two vertex sets.

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

[6]The result of Lemma 1 is sometimes known as a Moore-type bound [18], although we note that the bound in (2) is tighter than the corresponding bound in [19] for our case, when $l > k$.

[7]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.

Example 2. *A set of* 3 *mutually-orthogonal* 3×3 *squares is*

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

III. Regular Cage Graphs

We now show how to construct girth-6 bipartite cage graphs where the degrees of both vertex sets are equal. 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. The resulting graphs will have $|X| = |Y| = q^2+q+1$.

A. Construction of Regular Cage Graph

The construction of regular bipartite cage graphs of girth 6 is inspired from the construction in Wong [5], and is given in Algorithm 1. Bipartiteness arises from the construction.

Algorithm 1 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 vertex $\hat{x}_{m,i}$, where $m \in \{0, 1, \ldots, k-2\}$, $i \in \{0, 1, \ldots, l-2\}$. Connect $\hat{x}_{m,i}$ to vertices $\hat{y}_{j+1, L^{(m)}_{i,j}}$, where $j = 0, 1, \ldots, l-2$.
```

The q^2+q layer 2 vertices $\hat{y}_{j,m}$, $j = 0, 1, \ldots, q$ and $m = 0, 1, \ldots, q-1$, coincide with the vertices $y_1, y_2, \ldots, y_{q^2+q}$, and can be mapped using $y_{jq+m+1} = \hat{y}_{j,m}$. Similarly, the q^2 layer 3 vertices $\hat{x}_{m,i}$, $m = 0, 1, \ldots, q-1$ and $i = 0, 1, \ldots, q-1$, coincide with 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 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.

We first show an example of Algorithm 1 with $k = 4$ and $l = 4$ (so $q = 3$), before proving that this indeed results in the desired cage graph. This graph will have $|X| = |Y| = 13$.

The first three steps are straightforward, as they involve connecting the vertices of layers 0, 1, and 2 in a tree. Step 4 connects all the vertices associated with $\hat{y}_{0,m}$ with the $l-1 = q$ vertices $\hat{x}_{m,i}$, $i = 0, 1, \ldots, q-1$. This gives Figure 3a.

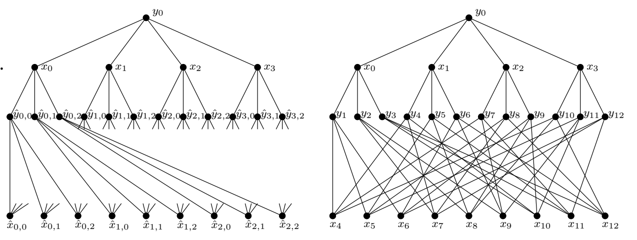
Now we consider connecting the other outgoing edges of each $\hat{x}_{m,i}$ vertex to the remaining $\hat{y}_{j,\mu}$ vertices, $j \neq 0$. The set of mutually-orthogonal squares of order $q = 3$, given in Example 2, guarantees that 4-cycles do not get introduced in step 5. Figure 3b shows the resulting bipartite cage graph.

B. Properties of Graph Constructed from Algorithm 1

We show that the graph constructed from Algorithm 1 is indeed a cage graph, as well as discuss additional properties.

Lemma 2. *In the bipartite graph constructed from Algorithm 1, the shortest cycle consists of at least* 6 *vertices.*

Proof: See Appendix B-B or [5, Section 4]. ■


y_0
x_0
x_1
x_2
x_3
$\hat{y}_{0,0}$ $\hat{y}_{0,1}$ $\hat{y}_{0,2}$ $\hat{y}_{1,0}$ $\hat{y}_{1,1}$ $\hat{y}_{1,2}$ $\hat{y}_{2,0}$ $\hat{y}_{2,1}$ $\hat{y}_{2,2}$ $\hat{y}_{3,0}$ $\hat{y}_{3,1}$ $\hat{y}_{3,2}$
$\hat{x}_{0,0}$ $\hat{x}_{0,1}$ $\hat{x}_{0,2}$ $\hat{x}_{1,0}$ $\hat{x}_{1,1}$ $\hat{x}_{1,2}$ $\hat{x}_{2,0}$ $\hat{x}_{2,1}$ $\hat{x}_{2,2}$
y_1 y_2 y_3 y_4 y_5 y_6 y_7 y_8 y_9 y_{10} y_{11} y_{12}
x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12}

(a) Conclusion of step 4. (b) Final bipartite graph.

Fig. 3. Construction of bipartite cage where $k = l = 4$, using Algorithm 1.

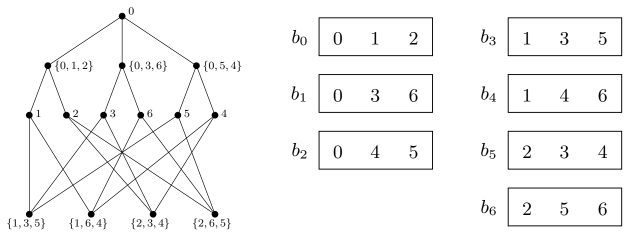
Theorem 3 (see also [5, Section 4])**.** *The regular bipartite graph constructed from Algorithm 1 is a bipartite cage graph of girth at least* 6*, with degree* $q+1$ *at all vertices.*

Proof: Algorithm 1 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—where every vertex has degree $q+1$. Thus $v = |Y|$ and $u = |X|$ achieve the lower bounds of Lemma 1 for the required degree distributions. By Lemma 2, the shortest cycle has at least 6 vertices, so the result is shown. ■

By interpreting Y as the elements and X as the blocks, 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. Thus lookup tables for the entire block design need not be stored, since the 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 $q+1$ [17]. The regular cage graph with $k = l = 3$ is the Heawood graph; see Figure 4, which also shows the associated Steiner system. This construction of the Heawood graph is analogous to the Skolem construction [9] of Steiner triple systems for $v = 9$.



(a) Bipartite graph. (b) Block design.

Fig. 4. Steiner system corresponding to $k = 3$, $l = 3$. Figure 4a visualizes the system as a bipartite graph, and Figure 4b shows the block design. This gives the same Steiner system as in [2, Fig. 3 (Example 2)].

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 (e.g., see [21], where $q = 6$).

IV. Scalable Designs

In Section IV-A, we construct cage graphs where the vertex degrees of the two vertex sets are highly unbalanced, i.e., where $\deg(X) = k = q+1$ but $\deg(Y) = l = p_n(q)$. We discuss some favorable scalability properties in Section IV-B.

A. Construction of Designs with $k = q + 1$, $l = p_n(q)$

The construction here is recursive; thus we call $l[n] = p_n(q)$ as the degree of the vertices in Y, at iteration n.[8] (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 constructed cages have $|X| = \frac{p_{n+1}(q)p_n(q)}{q+1}$ and $|Y| = p_{n+1}(q)$. We show such a graph in Figure 5.

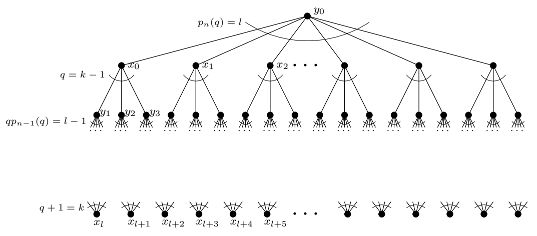


Fig. 5. Construction of bipartite cage graph with $k = q + 1$, $l = p_n(q)$.

We will inductively construct bipartite cages with $k = q+1$ and $l[n] = p_n(q)$ using a layered method similar to 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 Y as the elements and X as the blocks, this gives a Steiner system with block size $k = q+1$ and with $v[n-1] = p_n(q)$ 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 $u[n-1] = \frac{p_n(q)p_{n-1}(q)}{q+1}$ blocks.) This system can then be used to construct the $k = q + 1$, $l = p_n(q)$ cage—as given in Algorithm 2.

Algorithm 2 Construction of bipartite cage when $k = q+1$, $l = p_n(q)$

```
Require: A set of $v[n-1] = p_n(q)$ elements, and a collection $\mathcal{B} = \{b_h \mid h = 0, 1, \dots, u[n-1]-1\}$ 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 $x_0, x_1, \dots, x_{p_n(q)-1}$.
3: [Layer 2] For each vertex $x_j$, $j = 0, 1, \dots, l-1$, connect $x_j$ to $k - 1 = q$ vertices of $Y$. Let $\hat{y}_{j,m}$, $m = 0, 1, \dots, k-2$, 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 $b_h = \{g_0, g_1, g_2, \dots, g_q\}$.
6:     [Layer 3] Connect each vertex $\hat{y}_{g_0,m}$ $(m = 0, 1, \dots, k-2)$ to $q$ distinct vertices of $X$, called $\hat{x}^{(h)}_{m,i}$, $i = 0, 1, \dots, q-1$. 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 vertex $\hat{x}^{(h)}_{m,i}$, where $m \in \{0, 1, \dots, k-2\}$, $i \in \{0, 1, \dots, q-1\}$. Connect $\hat{x}^{(h)}_{m,i}$ to $\hat{y}_{g_{j+1}, L^{(m)}_{i,j}}$, $j = 0, 1, \dots, q-1$.
8: end for
Ensure: Bipartite cage with degrees $k = q+1$, $l[n] = p_n(q)$, and number of vertices $|Y| = v[n] = p_{n+1}(q)$, $|X| = u[n] = \frac{p_{n+1}(q)p_n(q)}{q+1}$
```

Algorithm 2 differs from Algorithm 1 in steps 6 and 7 because we only connect via $\hat{y}_{g_j,m}$ (where $j = 0, 1, \dots, q$) instead of $\hat{y}_{j,m}$ for all $j = 0, 1, \dots, l-1$. This is due to only considering $(q+1)$-element *subsets* instead of the entire set of $x_0, \dots, x_{l[n]}$ vertices when constructing each smaller subcage.

[8]We let $u[n]$, $v[n]$ denote the respective quantities at iteration n. Since $k[n] = q + 1$ for all n, we do not qualify k with the iteration number n.

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

$$\{y_0\} \cup \{x_j \mid j \in b_h\} \cup \{\hat{y}_{j,m} \mid j \in b_h,\ m = 0, 1, \dots, k-2\} \cup \{\hat{x}^{(h)}_{m,i} \mid m = 0, 1, \dots, k-2,\ i = 0, 1, \dots, q-1\}.$$

Lemma 4. *The graph of Algorithm 2 has the desired number of vertices, $|X|$ and $|Y|$, and satisfies the degree requirements.*

Proof: This can be shown via careful accounting. We provide the complete proof in Appendix B-C. ■

Lemma 5. *In the constructed bipartite graph of Algorithm 2, the shortest cycle has length of at least 6 vertices.*

Proof: As there are neither odd cycles nor length-2 cycles, we only need to check that there are no length-4 cycles. Since each selection b_h of layer 1 vertices induces a subgraph which is isomorphic to the $k = l = q + 1$ bipartite regular cage graph, any properties from the regular graph also hold for the subgraph. Thus 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 layer 2 vertices are connected to different x_j vertices of layer 1. 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 $\hat{x}^{(h')}_{m',i'}$.[9] 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 two 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. ■

Lemma 6. *Supposing that 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 constructs a bipartite cage (of girth 6) with parameters $k = q+1$, $l[n] = p_n(q)$ (and $v[n] = p_{n+1}(q)$, $u[n] = \frac{p_{n+1}(q)p_n(q)}{q+1}$).*

Proof: Follows from Lemmas 4 and 5. ■

Theorem 7. *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 $k = l = q + 1$ cage graph from Algorithm 1, and so is constructible. The conclusion follows by induction, using Lemma 6. ■

In Figure 6, we show the resulting storage system design after iteration $n = 2$, for the case $q = 2$ (i.e., $k = 3$). This system is in fact an extension of the $k = 3$, $l = 3$ block design of Figure 4; for storage nodes $b_0, b_1, \dots, b_6$, the first 3 data chunks in each node are exactly the same between Figures 4b and 6. This scalability will be explained in Section IV-B.

These cage graphs form a family of designs where $k =$

[9]We know $h \neq h'$, or else the 4-cycle is entirely within the b_h-subgraph.

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

Fig. 6. Block design for distributed storage system corresponding to $k = 3$ and $l = 7$. Each data chunk has 3 replicas and each storage node stores 7 chunks. In total there are 35 distinct data chunks and 15 storage nodes.

$q+1$, $l = p_n(q)$, $v = p_{n+1}(q)$, and $u = \frac{p_{n+1}(q)p_n(q)}{q+1}$, and thus are coincident with the Steiner systems $S(2, q+1, p_{n+1}(q))$.

B. Advantages of Scaled Constructions

The construction of Algorithm 2 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 these methods. That is, if an extant system has $l[n-1] = p_{n-1}(q)$, it is relatively simple to increase the size of the system so that the degree of Y has $l[n] = p_n(q)$. This is because the following holds:

Theorem 8. *Consider a cage graph, $G[n]$, with parameters $k = q+1$, $l[n] = p_n(q)$ constructed in iteration n of Algorithm 2. 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, and called $G[n-1]$) is a subgraph of $G[n]$.*

Theorem 8 will be proved with the help of Lemma 9.

Lemma 9. *Consider a cage graph with $k = q+1$, $l[n] = p_n(q)$, to be constructed in the n-th iteration of Algorithm 2. 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 $n = 2$. The cage graph with parameters $k = q+1$, $l[1] = q+1$ is the graph from Algorithm 1. 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 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). 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 8.

Proof of Theorem 8: From Lemma 9, one 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 $l = p_n(q)$ (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 8, we see that because $G[n-1]$ is a subgraph of $G[n]$—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) 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 4b results in the appended storage system of Figure 6.

It is similarly simple to construct a storage system which 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 $u[n]$ elements (i.e., $k = q+1$ and $l[n] = p_n(q)$) and then leave empty slots in the blocks which are supposed to store 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.

C. Other Scalable Constructions

Due to space constraints, we do not discuss the construction of a related class of block designs, which are those that coincide with affine geometries [3]. A similar construction to Algorithm 1 can be used to construct cage graphs where $k = q$ and $l = q+1$—leading to the graph of Figure 2 when $q = 3$. From this base case, similar scalability results can be derived for storage system designs with $k = q$ and $l = p_n(q)$.

V. Conclusion

In this paper, 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.

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 (DHTs)—such as CAN [22], Chord [23], Pastry [24], and Tapestry [25]—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.

References

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

[2] 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.

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

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

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

[6] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit and optimal exact-regenerating codes for the minimum-bandwidth point in distributed storage," in *Proceedings of the IEEE Intl. Symp. on Information Theory*, Austin, TX, Jun. 13 – 18, 2010, pp. 1938–1942.

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

[8] 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.

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

[10] 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.

[11] R. R. Muntz and J. C. S. Lui, "Performance analysis of disk arrays under failure," in *Proceedings of the 16th VLDB Conference*, Brisbane, Australia, Aug. 13 – 16, 1990, pp. 162–173.

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

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

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

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

[16] 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.

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

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

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

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] B. Y. Zhao *et al.*, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 1, pp. 41–53, Jan. 2004.

Appendix A
Latin Squares

We discuss Latin squares and mutually-orthogonal Latin squares—which will aid in the construction of bipartite cage graphs of girth 6. A comprehensive treatment of Latin squares can be found in the text by Dénes and Keedwell [20].

Definition 1. *Consider a $q \times q$ matrix L where the entries take on values from $\mathcal{Q} = \{0, 1, 2, \ldots, q-1\}$. Then L is a* Latin square *if for every row i, the entries satisfy $L_{i,j} \neq L_{i,j'}$ whenever $j \neq j'$; and for every column j, the entries satisfy $L_{i,j} \neq L_{i',j}$ whenever $i \neq i'$.*

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

In fact—given any Latin square—by labeling symbols and permuting columns appropriately, we can establish a Latin square with a specified column in natural order. Next we define the concept of orthogonality for Latin squares.

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

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

When q is a prime or prime power, sets of mutually-orthogonal Latin squares can be derived by first identifying the generator of the multiplicative group associated with the finite field of characteristic q. That is, consider a Galois field $GF(q)$ with primitive element α, so that the elements are

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

Then the Latin squares, $L^{(1)}, L^{(2)}, \ldots, L^{(q-1)}$, with entries

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

are mutually-orthogonal with natural order zeroth column.[10]

[10] 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 we let the $q \times q$ matrix $L^{(0)}$ consist of $L^{(0)}_{i,j} = i$ 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 10. *For $\{L^{(0)}, \ldots, L^{(q-1)}\}$ 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 column has overlapping 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 $a, b \in \mathcal{Q}$), by the definition of mutually-orthogonal squares, we know that no other [non-zeroth] column will have overlapping entries. ■

APPENDIX B
PROOFS OF SELECTED LEMMAS

A. *Proof of Lemma 1*

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 $x_1, x_2 \in \tilde{X}$ were also both connected to some other vertex $\tilde{y} \neq y$. Then the graph would have a cycle of length 4, consisting of vertices $y \sim x_1 \sim \tilde{y} \sim x_2 \sim y$. Thus each vertex in $\tilde{X}$ must be connected to $k-1$ unique vertices of Y; we let these vertices be $\tilde{Y}$, where $|\tilde{Y}| = l(k-1)$. 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 $x_1 \in \tilde{X}$ and $x_2 \in \tilde{X}$ would form the 4-cycle $\tilde{y} \sim x_1 \sim y \sim x_2 \sim \tilde{y}$ (similar to above). Therefore for any $\tilde{y} \in \tilde{Y}$, the vertex must connect to at least $l-1$ vertices of $X \backslash \tilde{X}$. Let $\hat{X}$ consist of vertices in $X \backslash \tilde{X}$ 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 $u = |X| \geq |\tilde{X}| + |\hat{X}| \geq l + l(l-1)(k-1)/k$.

B. *Proof of Lemma 2*

We show that there are no cycles of lengths 2 or 4 (since bipartite graphs have no odd cycles). Clearly, there are no 2-cycles, since the graph is simple (i.e., no multiple edges).

To show that there are no cycles of length 4, we consider vertices from each particular layer, and show that the construction results in no 4-cycle involving the vertices at that layer. For layer 0, there are no 4-cycles which include vertex y_0, as layers 0, 1, and 2 form a tree of depth 3. Now consider any 4-cycles which 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 $\hat{y}_{j,m'}$ can not 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 only connected 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, $\hat{y}_{0,m}$ is not part 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 overlapping entries between the two squares can be found; this contradicts Lemma 10, since only the zeroth column has overlapping entries. Thus no 4-cycles exist which involve layer 2 vertices.

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

C. *Proof of Lemma 4*

We want to show that all the vertices in Y have exactly $l[n] = p_n(q)$ outgoing edges, and all the vertices in X have exactly $k = q+1$ outgoing edges. Furthermore, $|Y| = v[n] = p_{n+1}(q)$ and $|X| = u[n] = \frac{p_{n+1}(q)p_n(q)}{q+1}$.

First we verify that we have the correct number of vertices. For Y, there is 1 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 $q^2 u[n-1] = \frac{q^2 p_n(q) p_{n-1}(q)}{q+1}$ vertices. Therefore, $u[n] = |X| = p_n(q) + \frac{q^2 p_n(q) p_{n-1}(q)}{q+1} = \frac{p_{n+1}(q) p_n(q)}{q+1}$.

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 in the collection of subsets, $\mathcal{B}$, that 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 1 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, 1 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.