

A New Minimum Density RAID-6 Code with a Word Size of Eight

James S. Plank

plank@cs.utk.edu

Technical Report CS-08-612

Department of Electrical Engineering and Computer Science
University of Tennessee

April, 2008.

This paper has been submitted for publication.

See the web link below for current publication status.

<http://www.cs.utk.edu/~plank/plank/papers/CS-08-612.html>

A New Minimum Density RAID-6 Code with a Word Size of Eight

James S. Plank

Department of Electrical Engineering and Computer Science
University of Tennessee

Abstract

RAID-6 storage systems protect k disks of data with two parity disks so that the system of $k + 2$ disks may tolerate the failure of any two disks. Coding techniques for RAID-6 systems are varied, but an important class of techniques are those with *minimum density*, with an optimal combination of encoding, decoding and modification complexity. The *word size* of a code impacts both how the code is laid out on each disk's sectors and how large k can be. Word sizes which are powers of two are especially important, since they fit precisely into file system blocks. Minimum density codes exist for many word sizes, with the notable exception of eight. This paper fills that gap by describing new codes for this important word size. The description includes performance properties as well as details of the discovery process.

1 Introduction

Disk and network storage systems have grown to the point where the fault-tolerance of RAID-5 is no longer enough. RAID-5 systems protect k data disks with a single parity disk so that the loss of any single data or parity disk may be tolerated without data loss. Large network storage systems such as Oceanstore [19], LoCI [1], Data Domain [25], Pergamum [20] and Panasas [21] have grown to such an extent that combinations of disk, sector and network

node failures require a higher level of fault tolerance. RAID-6 fits this requirement.

With RAID-6, a second parity disk is added to the system in such a way that the loss of any two disks may be tolerated without data loss. However, unlike RAID-5, which provides a precise method for encoding the data, RAID-6 leaves the method unspecified. Methods for implementing RAID-6 are varied, and include classic erasure coding techniques like Reed-Solomon coding [4, 17, 12, 18] and more specialized codes such as EVENODD [2], RDP [5], Blaum-Roth [3], X-Code[24], B-Codes [23] and Liberation codes [15]. These codes have varied performance characteristics with no one code being ideal for all applications. As such it is important for RAID-6 storage implementors to have at their disposal a variety of RAID-6 codes from which to choose [15].

In this paper, we focus on a class of codes called *Minimum Density* RAID-6 codes. These codes are based on binary matrices which satisfy a lower-bound on the number of non-zero entries. They exhibit excellent properties as RAID-6 codes, including near-optimal encoding and decoding performance combined with optimal modification performance. Additionally, they allow for flexibility in the number of data drives that they support without sacrificing encoding or decoding performance. The only known minimum density codes for RAID-6 are the Blaum-Roth [3] and Liberation codes [15].

One parameter of these codes is the *word size* which constrains how data is laid out on the disk. Specifically, these codes require data to be partitioned into blocks whose size must be multiples of the word size. For convenient manipulation in file systems, word sizes that are powers of two are extremely convenient. Blaum-Roth codes are defined for word sizes which are one less than a prime number, and Liberation codes are defined for word sizes which are equal to a prime number. As such, the very important word size of eight has no minimum density code defined for it.

In this paper, we define multiple minimum density codes whose word size equals eight, and in doing so, fill an important hole in the field. We provide their specification, describe their performance, and give details on how we discovered them. Additionally, we make their implementation available as part of the **Jerasure** general purpose erasure coding library [13].

2 Nomenclature and the RAID-6 Specification

It is an unfortunate consequence of the history of erasure coding research that there is no unified nomenclature for erasure coding. We borrow terminology mostly from Hafner *et al* [8], but try to conform to more classic erasure coding terminology (e.g. [3, 11]) when appropriate.

Our storage system is composed of an **array** of n disks, each of which is the same size. Of these n disks, k of them hold **data** and the remaining m hold coding information, often termed **parity**. In a RAID-6 system, $m = 2$.

Disks are composed of **sectors**, which are the smallest units of I/O to and from a disk. Sector sizes typically range from 512 to 4096 bytes, and are always powers of two. Most erasure codes are defined such that disks are logically organized to hold bits; however, when they are implemented, all the bits on one sector are considered as one single entity called an **element**. When a code specifies that bit on drive P equals the exclusive-or of bits on drives D_0 and D_1 , that really means that a sector on drive P will be calculated to be the parity (bitwise exclusive-or) of corresponding sectors on drives D_0 and D_1 . We will thus use the terms *bit*, *sector* and *element* equivalently in this paper.

A code typically has a **word size** w , which means that for the purposes of coding, each disk is partitioned into **strips** of w sectors each. Each of these strips is encoded and decoded independently from the other strips on the same disk. The collection of strips from all disks on the array that encode and decode together is called a **stripe**.

Thus, to define an erasure code, one must specify the word size w , and then one must describe the way in which sectors on the parity strips are calculated from the sectors on the data strips in the same stripe.

The RAID-6 specification calls for two parity drives, P and Q . The contents of the P drive are calculated as the parity of the data drives, just as in RAID-5. The contents of the Q drive are defined by the particular code, but must be a *Maximum Distance Separable (MDS)* code. This means that any combination of two-disk failures may be tolerated without data loss. While there are a variety of codes that can tolerate two-disk failures (e.g [24, 23, 9, 22, 6, 7, 16]), either the requirement of a specific P and Q drive or the MDS requirement

renders them inapplicable to this RAID-6 specification.

2.1 Bit Matrix Coding

One convenient way to describe an erasure code is to employ bit matrices. In this description, there are k data disks D_0, \dots, D_{k-1} and m parity disks C_0, \dots, C_{m-1} , each of which logically holds exactly w bits (as described above, in reality each is partitioned into strips of w sectors each). We label these bits $d_{i,0}, \dots, d_{i,w-1}$ for data disk D_i , and $c_{j,0}, \dots, c_{j,w-1}$ for parity disk C_j .

The system uses a $w(k+m) \times wk$ matrix over $GF(2)$ to perform encoding. This means that every element of the matrix is either zero or one, and arithmetic is equivalent to arithmetic modulo two. We call the matrix a *binary distribution matrix*, or BDM; however, it relates to classic error correcting codes by being the transpose of the code's *generator matrix* [11]. The state of a bit matrix coding system is described by the matrix-vector product depicted in Figure 1.

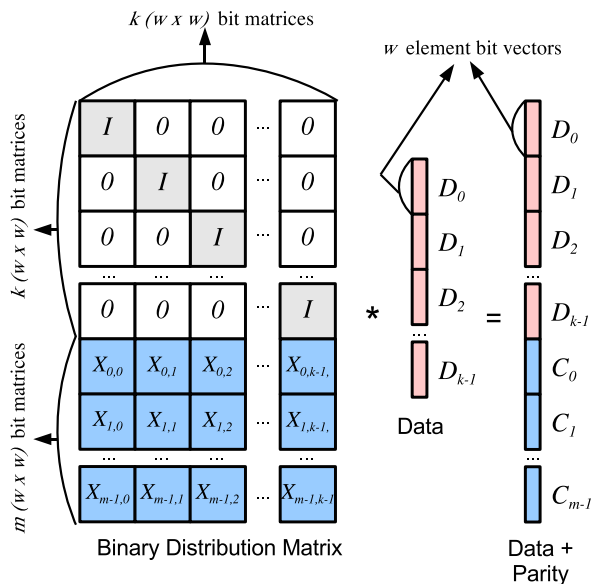


Figure 1: A bit matrix coding system.

The BDM has a specific format. Its first wk rows compose a $wk \times wk$ identity matrix, pictured in Figure 1 as a $k \times k$ matrix whose elements are each $w \times w$ bit matrices. The

next mw rows are composed of mk matrices, each of which is a $w \times w$ bit matrix $X_{i,j}$.

We multiply the BDM by a vector composed of the wk bits of data. We depict that in Figure 1 as k bit vectors with w elements each. The product vector contains the $(k + m)w$ bits of the entire system. The first wk elements are equal to the data vector, and the last wm elements contain the parity bits, held in the m parity disks.

Note that each disk corresponds to a row of $w \times w$ matrices in the BDM, and that each bit of each disk corresponds to one of the $w(k + m)$ rows of the BDM. The act of encoding is to calculate each bit of each C_i as the dot product of that bit's row in the BDM and the data. Since each element of the system is a bit, this dot product may be calculated as the XOR of each data bit that has a one in the parity bit's row. Therefore, the performance of encoding is directly related to the number of ones in the BDM.

To decode, suppose some of the disks fail. As long as there are k surviving disks, we decode by creating a new $wk \times wk$ matrix BDM' from the wk rows corresponding to k of the surviving disks. The product of that matrix and the original data is equal to these k surviving disks. To decode, we therefore invert BDM' and multiply it by the survivors – that allows us to calculate any lost data. Once we have the data, we may use the original BDM to calculate any lost parity disks.

For a coding system to be MDS, it must tolerate the loss of any m disks. Therefore, every possible BDM' matrix must be invertible.

Since the first wk rows of the BDM compose an identity matrix, we may precisely specify a BDM with a *Coding Distribution Matrix (CDM)* composed of the last wm rows of the BDM. It is these rows that define how the parity disks are calculated.

Again, relating the above text to classic coding theory, suppose we label a code's CDM to be matrix A . Then the code's *generator matrix* is equal to $(I|A^T)$, and its *parity check matrix* is equal to $(A|I)$ [11], although sometimes it is written $(A|I)^T$ [8].

2.2 RAID-6 Bit Matrix Encoding

When this methodology is applied to RAID-6, the BDM is much more restricted. First, $m = 2$, and the two parity disks are named $P = C_0$ and $Q = C_1$. Since the P disks must be the parity of the data disks, each matrix $X_{0,i}$ is equal to a $w \times w$ identity matrix. Thus, the

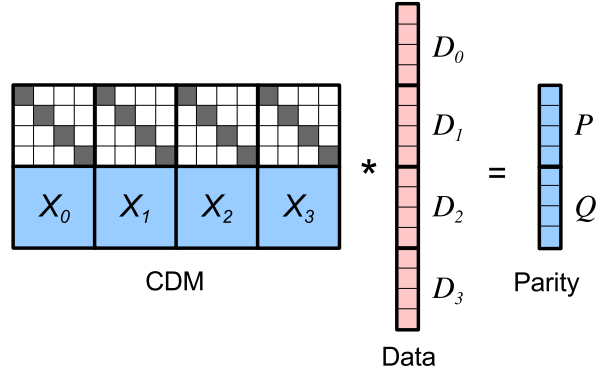


Figure 2: Bit matrix representation of RAID-6 coding when $k = 4$ and $w = 4$.

only degree of freedom is in the definition of the $X_{1,i}$ matrices that encode the Q disks. For simplicity of notation, we remove the first subscript and call these matrices X_0, \dots, X_{k-1} . A RAID-6 system is depicted in Figure 2 for $k = 4$ and $w = 4$.

To calculate the contents of a coding bit, we simply look at the bit's row of the CDM and calculate the XOR of each data bit that has a one in its corresponding column. For example, in Figure 2, it is easy to see that $p_0 = d_{0,0} \oplus d_{1,0} \oplus d_{2,0} \oplus d_{3,0}$.

Each data bit $d_{i,j}$ corresponds to column $wi + j$ in the CDM. Therefore, when bit $d_{i,j}$ is modified, one must update each parity bit whose row contains a one in column wi_j with the XOR of the data bit's old and new values.

2.3 Minimal Density Bit Matrices

A *minimal density bit matrix* is a CDM one that achieves a lower bound of $2kw + k - 1$ non-zero entries [3], yet still defines an MDS RAID-6 code. This is achieved when one of the X_i matrices has exactly w ones and the remaining $k - 1$ matrices have exactly $w + 1$ ones. These matrices define codes with an excellent blend of properties:

- Their encoding performance is equal to $k - 1 + \frac{k-1}{2w}$ XOR operations per encoded element. Optimal encoding is $k - 1$ XOR operations per encoded element, so the performance penalty is a factor of $1 + \frac{1}{2w}$. One nice feature of this penalty is that it is independent of k , which means that the codes lend themselves well to applications when data disks need to be added or subtracted dynamically [15].

- Their modification performance is optimal [3].
- Their decoding performance is near optimal as well, although it requires a technique that augments the standard matrix inversion to use intermediate results in the decoding. This technique is first presented by Hafner [8] and is later applied to minimal density bit matrices by Plank [15].
- They lend themselves well to general algorithms for uncorrelated sector failure reconstruction [8].

Minimal density bit matrices have been described when $w + 1$ is prime [3] and when w is prime [15]. The most convenient values of w from an implementation standpoint are powers of two, since they allow for data and coding strips to fit into standard file system blocks. While $w = 16$ has a known minimal density bit matrix (since $16+1$ is prime), $w = 8$ does not. Thus there is a very important gap in the field that needs to be filled: the discovery of minimal density bit matrices for $w = 8$. This paper describes the search for these codes, their discovery, their properties and their performance.

3 Summary of Codes Found

Our approach toward code discovery was not elegant, but pragmatic. Any minimal density bit matrix for $w = 8$ will be equivalent to a code where X_0 is the identity matrix and the other seven X_i will each be composed of a permutation matrix (a matrix with exactly one one in each row and column) plus one extra one bit. There are $56(8!)$ such matrices, which means that there are

$$\binom{56(8!)}{7} \approx 5.94 \times 10^{40}$$

possible settings of X_i that can lead to MDS matrices.

Our approach was to perform an enumeration of these matrices, yet to prune the enumeration so that the search would be tractable. This pruning was successful, and we discovered 48 distinct sets of minimal distance MDS matrices, where two matrices belong to a set if they are permutations of one another (Section 5.2 defines these permutations). Given a matrix,

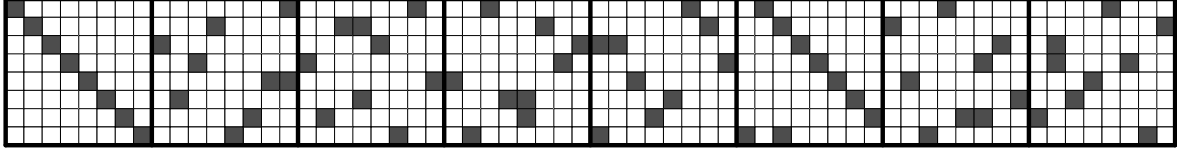


Figure 3: The eight X_i for the best performing minimal density MDS matrix for $w = 8$.

there are $(8!)(7!)$ permutations, which means that of the 5.94×10^{40} possible minimal density matrices, there are only 9.754×10^9 MDS ones.

In Figure 3, we display the matrix from the best of the 48 sets. The 48 sets are equivalent from an encoding and modification standpoint. They differ, though, in decoding performance. A matrix from the best decoding of the 48 sets is pictured in Figure 3. All 48 sets are enumerated in [14].

4 Performance

In this section, we compare the performance of these codes to other RAID-6 codes. The only other code that exists for $w = 8$ is the Cauchy Reed-Solomon (CRS) code [4]. For evaluation, we use the CRS implementation in the **Jerasure** library [13], which optimizes the encoding performance. To compare with other RAID-6 codes, we selected the smallest values of w that are greater than eight. For RDP, EVENODD and Blaum-Roth codes, $w + 1$ must be prime, so $w = 10$. For Liberation codes, w must be prime, so $w = 11$. All codes were implemented using the **Jerasure** library either directly (CRS, Blaum-Roth, Liberation) or as a base for the other codes (RDP, EVENODD).

We display the performance of encoding for k ranging from three to eight in Figure 4. The values on the Y axis are the factor of overhead over optimal encoding, where Optimal encoding is $k - 1$ XOR operations per coding element.

Of the two codes for $w = 8$, the minimal density code vastly outperforms Cauchy Reed-Solomon coding. When $w = 8$, the primitive polynomial that generates Cauchy coding matrices has three non-zero terms, which means that the number of ones in the eight X_i are 8, 11, 11, 14, 14, 17, 18 and 18 respectively. The minimal density X_0 has eight ones, and

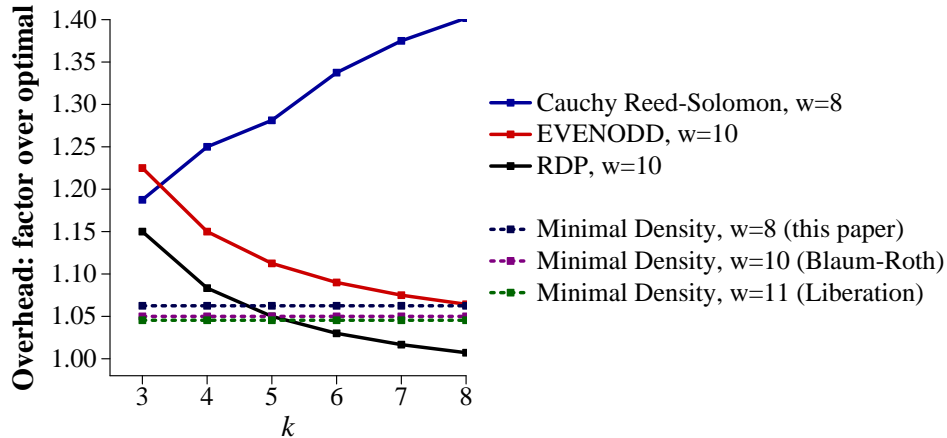


Figure 4: Encoding performance of RAID-6 codes when w is the smallest legal value ≥ 8 .

the remaining X_i have nine.

With respect to the others, the performance of the three minimal density codes (Blaum-Roth, Liberation and the codes of this paper) exhibit the same performance as noted in [15]. Their overhead is independent of k . They are always superior to EVENODD coding and outperform RDP when k is small. Since the performance of these codes improves as w grows, the Liberation code for $w = 11$ performs better than the Blaum-Roth code for $w = 10$, which performs better than the $w = 8$ code.

The performance of decoding is displayed in Figure 5. The results mirror the results of [15]: the decoding performance of minimal density codes is worse than their encoding performance, but not by a large margin. As with encoding, decoding outperforms CRS coding significantly. With all the minimal density codes, decoding is optimized with the the “Code-specific Hybrid Reconstruction” algorithm of [8], (which is equivalent to the “Bit Scheduling” algorithm of [15]).

Figure 6 shows the decoding performance of each of the 48 sets of minimal density MDS matrices when $k = w = 8$. Although they have equivalent encoding and modification performance, their inverses have different properties, which means that their decoding performance ranges from a factor 1.1845 to the best to 1.1962. In the figure, the sets are ordered by their decoding performance.

Finally, the performance of modification is displayed in Figure 7. The values on the Y

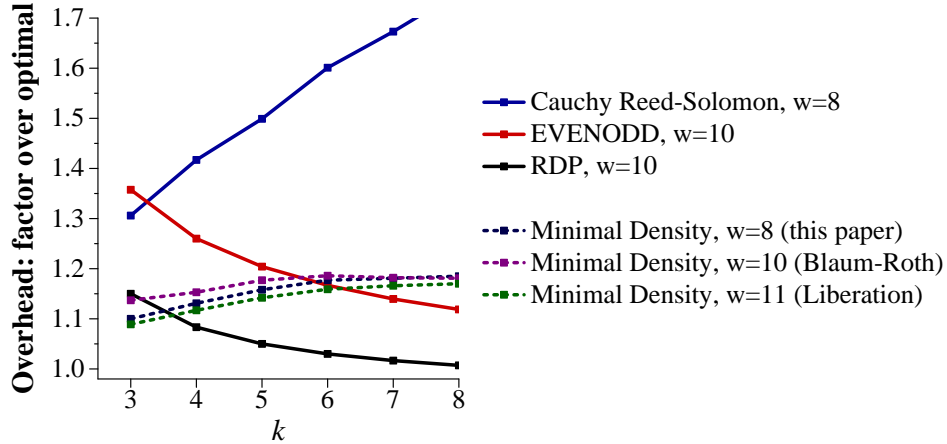


Figure 5: Decoding performance of RAID-6 codes when w is the smallest legal value ≥ 8 .

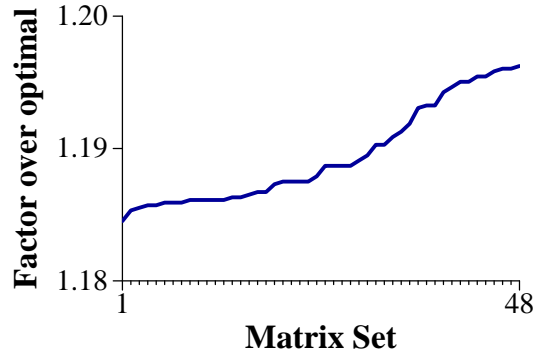


Figure 6: Decoding performance for $k = 8$ of the 48 minimal distance MDS codes for $w = 8$.

axis are the average number of coding elements that must be updated when a data element is modified. For the bit matrix coding techniques, this is the average number of ones in each column of the CDM.

The results again mirror those in [15]. The minimal density codes achieve optimality for their respective values of w [3] and clearly outperform the others.

5 Details of the Search

Brute-force enumeration is typically as subtle hitting a rock with a sledge hammer. water. However, in our quest to discover minimal density MDS codes for $w = 8$, the number of

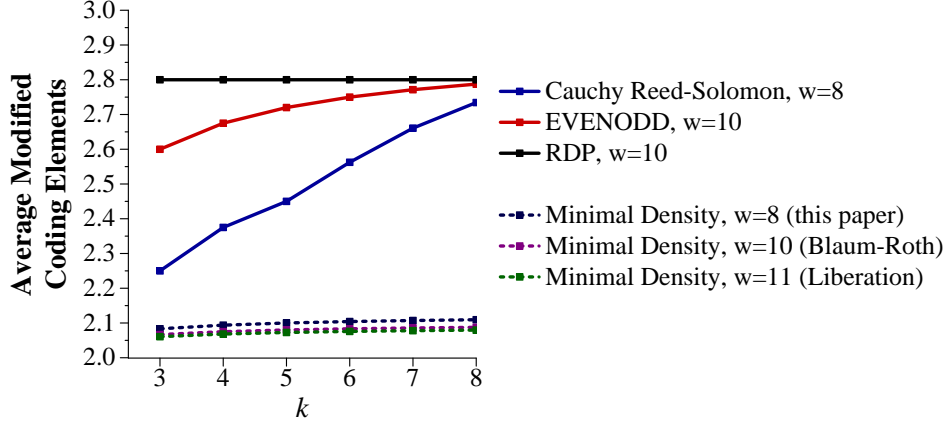


Figure 7: Modification performance of RAID-6 codes when w is the smallest legal value ≥ 8 .

matrices is too large to exhaustively search without thousands of machines and months of time. We employed some interesting techniques, described in this section, to limit the search space so that our enumeration of MDS matrices took four days, zero hours and 34 minutes on a MacBook Pro with a 2.16 GHz Intel Core Duo processor.

To perform the enumeration, we represent each X_i ($i > 0$) with a permutation matrix and an extra one. The permutation matrix may be represented by a vector Π_i which has w integer elements $\pi_{i,0}, \dots, \pi_{i,w-1}$. $\pi_{i,j}$ is the column which contains the location of the one in row j . In order to be a valid permutation matrix, Π_i must be such that $0 \leq \pi_{i,j} < w$ and if $j \neq j'$ then $\pi_{i,j} \neq \pi_{i,j'}$. When generating a permutation matrix Π_i , we have w choices for $\pi_{i,0}$, $w - 1$ choices for $\pi_{i,1}$, etc. Thus, there are $w!$ possible permutation matrices.

We can represent an X_i with its permutation matrix Π_i plus a row and column identifying the extra one. We will use the following notation to represent X_i :

$$X_i = \{\Pi_i, r_i, c_i\}$$

For example, X_1 in Figure 3 may be represented as $\{(7, 3, 0, 2, 6, 1, 5, 4), 4, 7\}$. Given a permutation matrix Π_i , there are $w^2 - w$ possible locations for the extra one, since it cannot coincide with any of the existing ones in Π_i . Thus, there are $(w!)(w^2 - w)$ possible matrices for each X_i .

5.1 MDS Testing and Enumeration

We will always start with $X_0 = I$. For $i > 0$, $X_i = \{\Pi_i, r_i, c_i\}$ must be such that $\pi_{i,r_i} \neq c_i$. In order for a RAID-6 code to be MDS, the X_i matrices must satisfy two properties [3, 15]:

1. They must each be invertible.
2. For all $i \neq j$, the sum $X_i + X_j$ must also be invertible.

If we devine our X_i as described above, then we know that each X_i is invertible, so we never need to test that property. Next, suppose that we have a minimal MDS code for $k = j < w$ composed of j matrices X_0, \dots, X_{j-1} . Then if we define a matrix X_j such that if $X_j + X_i$ is invertible for $0 \leq i < j$, then X_0, \dots, X_j compose a minimal MDS code for $k = j + 1$.

This gives us a simple technique for enumeration — start with $j = 1$ and $X_0 = I$, and repeat the following steps:

- Enumerate all possible X_j . If X_j is such that if $X_j + X_i$ is invertible for $0 \leq i < j$, then increment j and repeat these steps.
- Record each code for which $j = k$.
- When the enumeration for this step is finished, decrement j and continue enumeration from the previous step.

This enumeration does perform some pruning of the original 5.94×10^{40} matrices. However, there are three techniques that we employed to prune the enumeration further. They are described below.

5.2 Permutations

First, it should be clear that if a code defined by matrices X_0, \dots, X_{k-1} is MDS, then so is a code defined by any permutation of these matrices. Since we require that X_0 to be the identity matrix, and it is clear that the X_i must be distinct (or their sums will not be invertible), then whenever we have a code defined by X_0, \dots, X_{k-1} , then that code represents a set of $(k - 1)!$ equivalent codes, where each code is a permutation of X_1, \dots, X_{k-1} .

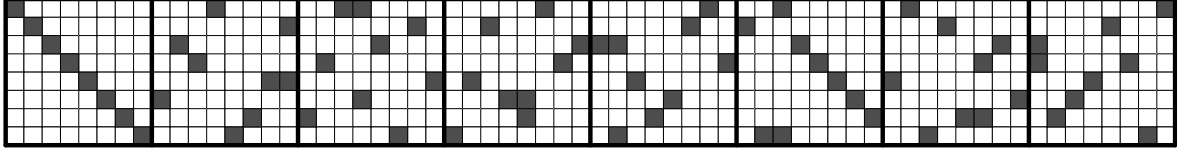


Figure 8: The matrix of Figure 3 where the X_i are replaced with $X_i^{0,1}$.

Second, let M be a square matrix. If M is invertible, then so is any matrix constructed by swapping rows or columns of M . Now, suppose we have a code defined by X_0, \dots, X_{k-1} . Let us construct a new code defined by the matrices $X_0^{x,y}, \dots, X_{k-1}^{x,y}$ where $X_i^{x,y}$ is equal to X_i with rows x and y swapped, and columns x and y swapped. For example, in Figure 8 we replace the X_i of Figure 3 with $X_i^{0,1}$.

Note that if the original code is MDS, then so is the new code. Moreover $X_0^{x,y}$ will equal the identity matrix. Since there are $(w!)$ ways to permute the rows and columns of a $w \times w$ matrix, this means that a code defined by X_0, \dots, X_{k-1} represents a set of $(w!)$ equivalent codes where each code is a permutation of swapping rows and columns of X_0, \dots, X_{k-1} .

Combining the two types of permutations, we see that a single code represents $(k-1)!(w!)$ equivalent codes. This is why the 48 sets of codes discovered in our enumeration represent 9.754×10^9 matrices.

We can use the permutations to limit our enumeration. Specifically, let $X_i = \{\Pi_i, r_i, c_i\}$ and $X_j = \{\Pi_j, r_j, c_j\}$. It should be clear that $r_i \neq r_j$, for if they were equal, $X_i + X_j$ would be a matrix with an even number of ones in each row, and would therefore not be invertible [3]. Similarly $c_i \neq c_j$. This means that each r_i must be distinct, as must each c_j . Since we can permute rows and columns to generate equivalent codes, any MDS code must permute to one where $r_i = w - i$ for $i > 0$. Thus, we can limit our enumeration of X_i to only those where $r_i = w - i$, which is an improvement by a factor of w .

In the discussion that follows, we only consider X_i such that $r_i = w - i$.

5.3 The $I1$ Function

Let M_1 and M_2 be two $r \times w$ matrices where $r \leq w$ and there exist two permutation matrices Π_1 and Π_2 such that M_1 composes the first r rows of Π_1 and that M_2 composes the

first r rows of Π_2 . We call Π_1 and Π_2 *superset* matrices of M_1 and M_2 .

We define the boolean function $I1(M_1, M_2)$ to be true *iff* M_1 and M_2 have two superset permutation matrices Π_1 and Π_2 such that $\Pi_1 + \Pi_2$ plus *any* one bit is an invertible matrix. Note, that means that for every one of the w^2 possible bits, $\Pi_1 + \Pi_2$ plus that bit must be invertible.

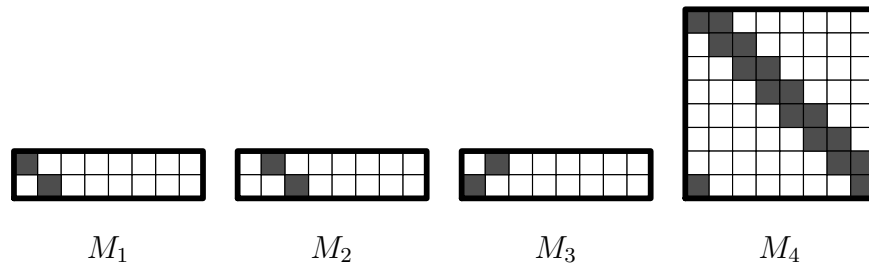


Figure 9: Examples to illustrate the $I1$ function.

We give a few examples in Figure 9. $I1(M_1, M_2)$ is true, because we can find two superset matrices (I and I rotated right one column) such that their sum equals M_4 , and it is easy to verify that M_4 plus any one bit is invertible. $I1(M_1, M_3)$ is false, because the sum of any two superset matrices plus a one bit in any column but the first two will yield a matrix whose leftmost two columns have ones in the first two rows and zeroes in the rest. Such a matrix is clearly not invertible. Similarly, $I1(M_2, M_3)$ is false because the sum of any two superset matrices plus a one bit in any row but the first will have an empty first row. Again, such a matrix is clearly not invertible.

Now, the $I1$ function gives us a way to restrict the X_j matrices when we enumerate them. Specifically, let $X_i = \{\Pi_i, r_i, c_i\}$ and $X_j = \{\Pi_j, r_j, c_j\}$. Then let M_i be X_i with rows r_i and r_j removed, and M_j be X_j with rows r_i and r_j removed. If $I1(M_i, M_j)$ is false, then $X_i + X_j$ cannot be invertible.

When we enumerate X_j as above, we may do so incrementally by constructing rows starting with row 0. After the construction of each row, we check to make sure that the $I1$ function holds with X_j and each other X_i minus rows r_j and r_i . If it does, then we move onto the next row. If it does not, then we continue enumerating the next constructions of the current row. This technique allows for significant pruning during the enumeration of matrices and makes the enumeration proceed much faster.

5.4 When Π_i and Π_j Share Values

The last enumeration pruning technique arises when $X_i = \{\Pi_i, r_i, c_i\}$ and $X_j = \{\Pi_j, r_j, c_j\}$ have the property that for some value of a , $\pi_{i,a} = \pi_{j,a}$. An example of this is in Figure 8, where $X_2 = \{(2, 6, 4, 1, 7, 3, 0, 5), 0, 3\}$ and $X_3 = \{(5, 2, 7, 6, 1, 3, 4, 0), 5, 4\}$. In this example, $a = 5$ and $\pi_{2,5} = \pi_{3,5} = 3$. When this occurs, the bit in row a , column $\pi_{i,a}$ of $X_i + X_j$ will equal zero, and unless r_i or r_j equals a , row a of $X_i + X_j$ will contain all zeros and will therefore not be invertible. Similarly, unless c_i or c_j equals $\pi_{i,a}$, column $\pi_{i,a}$ will contain all zeros.

We can use this observation again to limit our enumeration. Recall from Section 5.2 that we constrain r_i for all X_i in our enumeration. When we incrementally generate row a of X_j , we check to see if $\pi_{j,a}$ is equal to $\pi_{i,a}$ for $0 < i < j$. If it is, then either r_i or r_j must be equal to a , and either c_i or c_j must be equal to $\pi_{i,a}$. In our above example from Figure 8, $c_2 = 3$ and $r_3 = 5$. If all of these conditions do not hold, then we may reject row a of X_j without further testing, again helping us prune the enumeration.

5.5 Pruned Enumeration Results

As mentioned in Section 5, the pruned enumeration completed in approximately four days on one consumer quality laptop computer. In the enumeration, 5064 MDS matrices were discovered, and by generating all permutations of these matrices, we identified the 48 distinct sets of matrices listed in [14].

6 Implementation and Availability

The best code that we discovered, which is depicted in Figure 3r, has been implemented as part of version 2.0 of the the **Jerasure** erasure coding library [13]. This library facilitates all aspects of matrix and bit matrix erasure coding discussed in Section 2.

7 Conclusion

This paper has two contributions to the field of RAID-6 erasure codes. First is the discovery of minimal density MDS codes for $w = 8$. This is an important value of w since it allows coding strips to fit evenly within file system blocks. The performance characteristics of these codes are on par with other minimal density MDS codes [3, 15] and compare favorably with other RAID-6 codes as well [2, 5].

The second contribution details the techniques that we used to turn a largely intractible enumeration to one that completed in four days on a single machine. These techniques may prove useful for other MDS code explorations.

Our future work in this arena is to use our knowledge of RAID-6 codes to start exploring MDS codes for larger numbers of failures, and to explore better algorithms than previous ones [8, 10, 15] for improving the performance of decoding.

8 Acknowledgements

This material is based upon work supported by the National Science Foundation under grant CNS-0615221. The author would like to thank Lihao Xu, Jim Hafner and Jay Wylie for valuable discussions that helped the author to hone the focus of this work.

References

- [1] BECK *et al*, M. Logistical computing and internetworking: Middleware for the use of storage in communication. In *Third Annual International Workshop on Active Middleware Services (AMS)* (San Francisco, August 2001).
- [2] BLAUM, M., BRADY, J., BRUCK, J., AND MENON, J. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computing* 44, 2 (February 1995), 192–202.
- [3] BLAUM, M., AND ROTH, R. M. On lowest density MDS codes. *IEEE Transactions on Information Theory* 45, 1 (January 1999), 46–59.

- [4] BLOMER, J., KALFANE, M., KARPINSKI, M., KARP, R., LUBY, M., AND ZUCKERMAN, D. An XOR-based erasure-resilient coding scheme. Tech. Rep. TR-95-048, International Computer Science Institute, August 1995.
- [5] CORBETT, P., ENGLISH, B., GOEL, A., GRACANAC, T., KLEIMAN, S., LEONG, J., AND SANKAR, S. Row diagonal parity for double disk failure correction. In *4th Usenix Conference on File and Storage Technologies* (San Francisco, CA, March 2004).
- [6] HAFNER, J. L. WEAVER Codes: Highly fault tolerant erasure codes for storage systems. In *FAST-2005: 4th Usenix Conference on File and Storage Technologies* (San Francisco, December 2005), pp. 211–224.
- [7] HAFNER, J. L. HoVer erasure codes for disk arrays. In *DSN-2006: The International Conference on Dependable Systems and Networks* (Philadelphia, June 2006), IEEE.
- [8] HAFNER, J. L., DEENADHAYALAN, V., RAO, K. K., AND TOMLIN, A. Matrix methods for lost data reconstruction in erasure codes. In *FAST-2005: 4th Usenix Conference on File and Storage Technologies* (San Francisco, December 2005), pp. 183–196.
- [9] HUANG, C., CHEN, M., AND LI, J. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *NCA-07: 6th IEEE International Symposium on Network Computing Applications* (Cambridge, MA, July 2007).
- [10] HUANG, C., LI, J., AND CHEN, M. On optimizing XOR-based codes for fault-tolerant storage applications. In *ITW'07, Information Theory Workshop* (Tahoe City, CA, September 2007), IEEE, pp. 218–223.
- [11] MACWILLIAMS, F. J., AND SLOANE, N. J. A. *The Theory of Error-Correcting Codes, Part I*. North-Holland Publishing Company, Amsterdam, New York, Oxford, 1977.
- [12] PLANK, J. S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience* 27, 9 (September 1997), 995–1012.
- [13] PLANK, J. S. Jerasure: A library in C/C++ facilitating erasure coding for storage applications. Tech. Rep. CS-07-603, University of Tennessee, September 2007.
- [14] PLANK, J. S. The 48 sets of minimal density MDS RAID-6 matrices for a word size of eight. Tech. Rep. UT-CS-08-611, University of Tennessee, March 2008.
- [15] PLANK, J. S. The RAID-6 Liberation codes. In *FAST-2008: 6th Usenix Conference on File and Storage Technologies* (San Jose, February 2008), pp. 97–110.
- [16] PLANK, J. S., BUCHSBAUM, A. L., COLLINS, R. L., AND THOMASON, M. G. Small parity-check erasure codes - exploration and observations. In *DSN-05: International Conference on Dependable Systems and Networks* (Yokohama, Japan, 2005), IEEE.

- [17] PLANK, J. S., AND DING, Y. Note: Correction to the 1997 tutorial on Reed-Solomon coding. *Software – Practice & Experience* 35, 2 (February 2005), 189–194.
- [18] REED, I. S., AND SOLOMON, G. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics* 8 (1960), 300–304.
- [19] RHEA, S., WELLS, C., EATON, P., GEELS, D., ZHAO, B., WEATHERSPOON, H., AND KUBIATOWICZ, J. Maintenance-free global data storage. *IEEE Internet Computing* 5, 5 (2001), 40–49.
- [20] STORER, M. W., GREENAN, K. M., MILLER, E. L., AND VORUGANTI, K. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In *FAST-2008: 6th Usenix Conference on File and Storage Technologies* (San Jose, February 2008), pp. 1–16.
- [21] WELCH, B., UNANGST, M., ABBASI, Z., GIBSON, G., MUELLER, B., SMALL, J., ZELENKA, J., AND ZHOU, B. Scalable performance of the panasas parallel file system. In *FAST-2008: 6th Usenix Conference on File and Storage Technologies* (San Jose, February 2008), pp. 17–33.
- [22] WYLIE, J. J., AND SWAMINATHAN, R. Determining fault tolerance of XOR-based erasure codes efficiently. In *DSN-2007: The International Conference on Dependable Systems and Networks* (Edinburgh, Scotland, June 2007), IEEE.
- [23] XU, L., BOHOSSIAN, V., BRUCK, J., AND WAGNER, D. Low density MDS codes and factors of complete graphs. *IEEE Transactions on Information Theory* 45, 6 (September 1999), 1817–1826.
- [24] XU, L., AND BRUCK, J. X-Code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory* 45, 1 (January 1999), 272–276.
- [25] ZHU, B., LI, K., AND PATTERSON, H. Avoiding the disk bottleneck in the Data Domain deduplication file system. In *FAST-2008: 6th Usenix Conference on File and Storage Technologies* (San Jose, February 2008), pp. 269–282.