5-2010

An Exploration of Optimization Algorithms and Heuristics for the Creation of Encoding and Decoding Schedules in Erasure Coding

Catherine D. Schuman
University of Tennessee - Knoxville, cschuman@utk.edu

An Exploration of Optimization Algorithms and Heuristics for the Creation of Encoding and Decoding Schedules in Erasure Coding

Catherine D. Schuman
University of Tennessee
Department of Electrical Engineering and Computer Science
Department of Mathematics
cschuman@utk.edu
Advisor: Dr. James S. Plank

Abstract

Erasure codes are employed by disk systems to tolerate failures. They are typically characterized by bit-matrices that are used for encoding and decoding. The efficiency of an erasure code using a bit-matrix is directly related to the number of exclusive-or (XOR) operations required during the encoding process. Thus, a problem within the field of erasure coding is how to schedule the XOR operations for any given bit-matrix so that the fewest number of XOR operations are required. This paper develops an algorithm for finding the optimum solution and analyzes the performance of two known heuristics on a set of encoding matrices.

1. Introduction

Erasure coding is a widely used technique to achieve failure resistance in storage devices. Erasure coding provides an alternative to simple data replication in that fewer disks are generally required in order to be resistant to the same number of failed disks. In erasure coding, data disks are encoded onto coding disks. Then, the entire system of data and coding disks is resilient to a certain number of failures, depending on what type of erasure coding is used.

The encoding process in erasure coding is done via matrix-vector multiplication in Galois-Field Arithmetic, where the data disks are considered the vector and the matrix differs for each erasure code. That is, a coding matrix characterizes an erasure code. In a subset of erasure codes, a bit-matrix (i.e., a matrix made up of ones and zeros) is used as the coding matrix. Because of the nature of bit-matrices, matrix-vector multiplication is simply a matter of exclusive-or (XOR) operations. The performance of an erasure code using bit-matrices can then be evaluated by the number of XOR operations required during the encoding process.

It has been observed that one can take advantage of the patterns in erasure coding matrices by scheduling the XOR operations in order to reduce the number of XOR operations required, thus improving the encoding performance of the matrices. There are two known heuristics for this problem: Code Specific Hybrid Reconstruction [5] and a technique by Huang, Li, and Chen using matching [8].

In general, neither of these heuristics provides the optimal schedule for a given matrix. The first goal of this thesis is to motivate the need for a heuristic by examining optimal schedules in general bit-matrices and to develop an algorithm for finding an optimal schedule for a given matrix (i.e., a schedule with the fewest number of operations possible). The second goal of this thesis is to evaluate the performance of two known heuristics by comparing the schedules each finds for a given set of matrices with the optimal schedules for these matrices.

2. Erasure Codes

Erasure codes are used in storage systems in which failures need to be tolerated. For example, disk array systems, data grids, collaborative/distributed storage applications, peer to peer networking, and archival storage are all systems in where failures can occur and where data loss can be catastrophic [9].

Storage companies such as Cleversafe [3] and Data Domain [17]); academic projects such as Oceanstore [14] and Pergamum [15]; and major technology corporations such as Hewlett Packard [16], IBM [4, 6], and Microsoft [7, 8], are all entities that regularly use erasure coding.

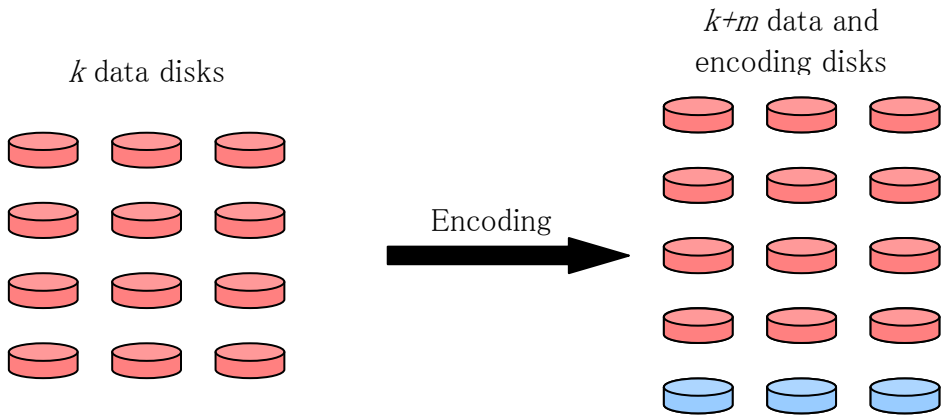


Figure 1: Encoding using an erasure code.

Erasure codes take data on k data disks and encode the data onto m coding disks (Figure 1). In Maximum Density Separable (MDS) erasure codes, the system of $k+m$ disks is then resistant to up to m failures [9]. In other words, up to m of the total $k+m$ disks in the system of disks can be erased while still being able to decode the data from the original k data disks (in Figure 2). Erasure codes also use a third parameter, w, called the word size. The disks are segmented into words and then the entire word is operated on rather than individual bits.

The encoding and decoding process is done using matrix-vector product in Galois Field Arithmetic, in which addition is performed via XOR and multiplication is performed in several different ways [13]. During the process of encoding, the data disks are considered as a vector of size k, while the

matrix, called a generator matrix, is of size (k+m) * k. All elements of the equation are w-bit words, where w is a parameter of the erasure code. The first k rows of the generator matrix form a sub-matrix that is the identity matrix. This results in the first k rows of the product vector of size k+m being the data disks. The remaining m rows of the generator matrix encode the data disks into the last m rows of the product vector. Erasure codes can be characterized by the matrix that is used during encoding in that different codes each have a different matrix.

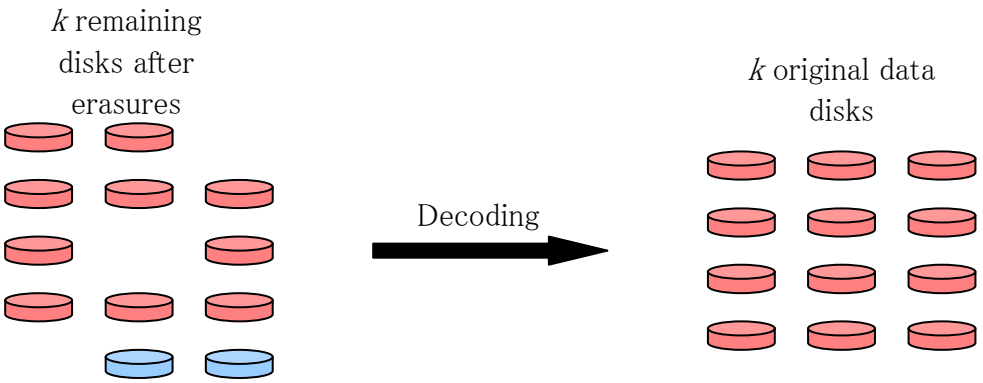


Figure 2: Decoding using an erasure code.

When up to m erasures occur, the system of surviving disks is then considered as a vector of size k. During decoding, the rows of the matrix that correspond to the erased disks are removed and the matrix is inverted. The inverted matrix is then multiplied by the vector formed by the remaining disks in order to obtain the original k data disks. Note that because the matrix is required to be inverted, every k*k matrix that is made up of any k rows of the generator matrix must still be invertible. This limits the number of ways a generator matrix can be constructed.

3. Bit-Matrix Encoding

Bit-matrix encoding was initially described in the original Cauchy Reed-Solomon erasure coding paper by Blomer, et. al. [2]. In bit-matrix encoding, a generator matrix called the *Binary Distribution Matrix* (BDM) is used. The BDM is composed of k+m rows and k columns of w x w matrices of zeros and ones. Because the data disks are held intact, the first k*w rows of the BDM are the identity matrix. The remaining m rows of w x w matrices are the portion of the matrix that differs from one erasure code to the next. Figure 3 shows the matrix-vector multiplication used in bit-matrix encoding.

Each row of the BDM corresponds to a specific coding disk. Because the rows of the matrix are bit strings, the dot-product of a row of the BDM with the vector of data disks is calculated simply using addition (i.e., XOR). Since all elements are bits, multiplication is unnecessary.

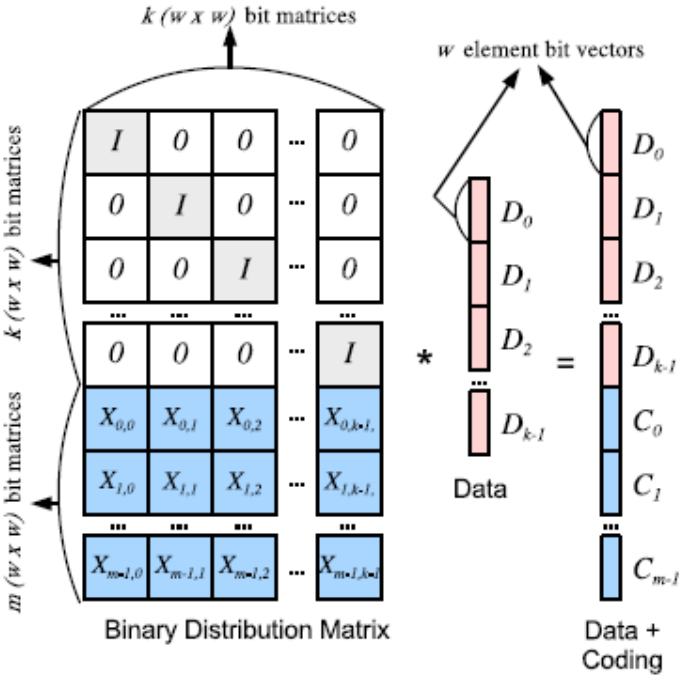


Figure 3: Bit-Matrix Encoding

Figure 4 gives a very simple example of the calculation of a coding disk, given a row of the BDM. In the figure, the shaded portions of the row of the BDM correspond to ones in the matrix. In the figure, k=2, w=3, $d_{i,j}$ denotes the j^{th} word of the i^{th} data disk, and $c_{i,j}$ is the word on the coding disk corresponding to the given row of the BDM.

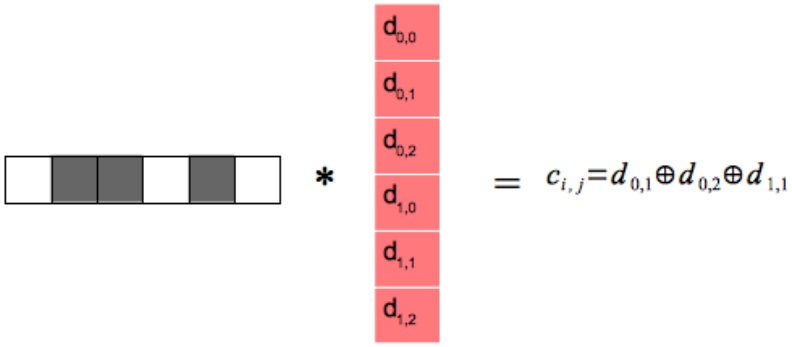


Figure 4: An example of row-vector multiplication to obtain a coding disk

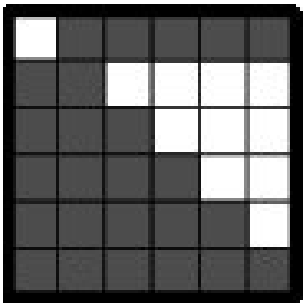
When performing multiplication in the straightforward way, the performance of encoding is directly related to the number of ones within the last mw rows of the BDM, called the *Coding Distribution Matrix* (CDM) [11]. As is demonstrated from the above multiplication, performing the multiplication in this way yields exactly o_j-1 operations for the j^{th} row, where o_j is the number of ones in the j^{th} row. Thus, the maximum number of XOR-operations for a BDM matrix is:

$$\left(\sum_{j=1}^{m*w} o_j\right) - m*w$$

Because of the relationship of the performance of encoding to the number of ones in a bit-matrix, several researchers have attempted to construct sparse coding matrices in order to achieve erasure codes with better performances [1, 10, 11, 12].

4. Scheduling Bit-Matrix Encoding

The performance of an erasure code using bit-matrices is directly related to the number of XOR operations required to encode the matrix. Therefore, it would be beneficial to take advantage of the structure of the CDM to minimize the number of XOR operations in order to improve the performance of the erasure code. Consider the coding bit-matrix and schedule in Figure 5. This bit-matrix is an example of a CDM for k=1, m=1, and w=6. While such a matrix would never be used in practice, since one would simply use replication for these parameters, we employ it simply for illustration.

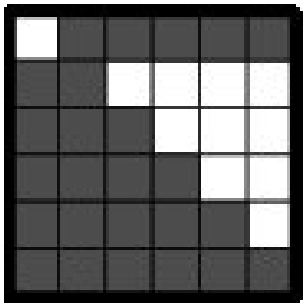


$$c_{0,0} = d_{0,1} \oplus d_{0,2} \oplus d_{0,3} \oplus d_{0,4} \oplus d_{0,5}$$
$$c_{0,1} = d_{0,0} \oplus d_{0,1}$$
$$c_{0,2} = d_{0,0} \oplus d_{0,1} \oplus d_{0,2}$$
$$c_{0,3} = d_{0,0} \oplus d_{0,1} \oplus d_{0,2} \oplus d_{0,3}$$
$$c_{0,4} = d_{0,0} \oplus d_{0,1} \oplus d_{0,2} \oplus d_{0,3} \oplus d_{0,4}$$
$$c_{0,5} = d_{0,0} \oplus d_{0,1} \oplus d_{0,2} \oplus d_{0,3} \oplus d_{0,4} \oplus d_{0,5}$$

Figure 5: Example bit-matrix and associated "naïve" schedule

Proceeding with encoding in the usual way, this particular bit-matrix requires 19 XOR-operations. However, due to the structure of the matrix, there are many commonalities in the rows. For example, rows one and two have ones in the zeroth and first columns. If we perform this operation first and use this common operation as a starting point to calculate remaining rows, we may reduce the total number of XOR operations.

In Figure 6, we note that the same bit-matrix from Figure 5 has an optimum schedule with only six XOR-operations. By first calculating row one of the coding matrix and building all other rows using that row, we can drastically reduce the number of XOR-operations. Many dense bit-matrices have similar schedules, so the performance of the erasure codes of those matrices no longer relies on the number of ones in a given matrix. However, the problem of finding an optimal schedule is a difficult one, as discussed in section 6.



$$c_{0,1} = d_{0,0} \oplus d_{0,1}$$
$$c_{0,2} = c_{0,1} \oplus d_{0,2}$$
$$c_{0,3} = c_{0,2} \oplus d_{0,3}$$
$$c_{0,4} = c_{0,3} \oplus d_{0,4}$$
$$c_{0,5} = c_{0,4} \oplus d_{0,5}$$
$$c_{0,0} = c_{0,5} \oplus d_{0,0}$$

Figure 6: Example bit-matrix and optimum schedule

Two known heuristics for this problem have been developed. Code Specific Hybrid Reconstruction is an algorithm developed by Hafner, Deenadhayalan, Rao, and Tomlin [5] and implemented in the **Jerasure** library by Plank [13]. In this algorithm, the operations for the row with the minimum number of ones are first scheduled. Then, all other rows are checked to see if their calculations may be performed with fewer XORs using this newly calculated rows as a starting point. If so, this is recorded, and the algorithm continues by scheduling the next rows with the minimum number of XOR operations. The algorithm repeats itself until all rows are scheduled.

A second method by Huang, Li, and Chen makes use of common subexpressions [8]. In this method, a weighted graph is first constructed in order to find the most common subexpressions. There are kw nodes of the graph, where each node represents a column in which a one could appear in the BDM. Each node is connected by a single edge to all other nodes in the graph. The nodes are given initial weights of zero. The rows of the matrix are then traversed. If a one appears in both the i^{th} and j^{th} column of a row, then the weight of the edge between the i^{th} and j^{th} nodes is incremented. When this process is complete, the edge with the largest weight in the graph is found, and all edges with weights less than this maximum weight are removed.

A matching algorithm is used to find the maximum matching of the remaining graph. The maximum matching, a set of edges, represents the common subexpressions to remove from the graph and thus the XOR-operations to be added into the schedule. These common subexpressions are removed from the rows of the original bit-matrix but are then added as rows to a new bit-matrix. The algorithm is then repeated on the new bit-matrix until there are no common subexpressions (i.e., all edges in the graph have weights equal to one).

5. Testing Space

For the purposes of this paper, only very small cases of this problem have been considered because finding the optimal schedule for these matrices requires considering an exponential number of schedules, as discussed in section 6. The matrices that we study are the 63 Cauchy matrices for w=6. The matrices are important because a standard construction of bit-matrices in Cauchy Reed Solomon coding for any $(k+m) \bullet 64$ employees these matrices [10]. Specifically, in the Galois-Field

GF(2^6), there are 63 non-zero numbers, and each of these has a representation using a 6x6 bit-matrix. This construction is defined in [2], and the 63 matrices are depicted in Figure 7.

As one can see, there are many rich patterns that can be exploited by scheduling algorithms within the Cauchy matrices. That is, there are many cases in which multiple rows have similar structures, so that the structures can be leveraged as described in section 4. Because of these rich patterns and because of the fact that they are employed in practice, these matrices are a good testing space for this problem.

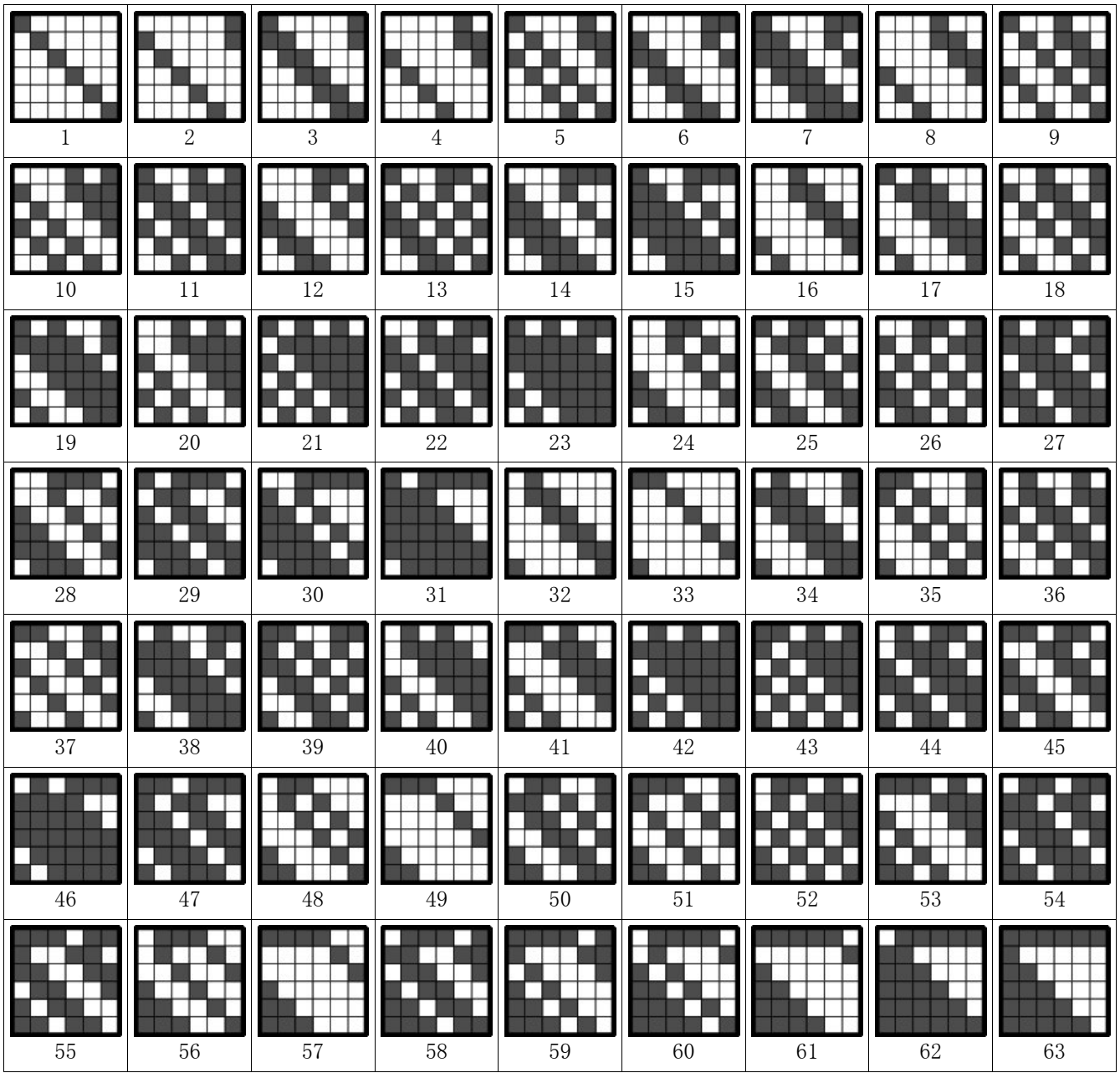


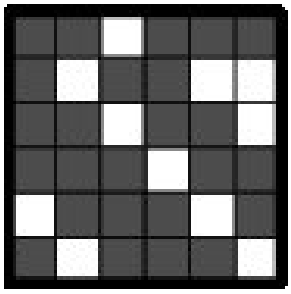
Figure 7: The 63 6x6 Cauchy matrices for $w = 6$

6. Optimal Schedule for Any Matrix

The first step in the analysis and comparison of the current known heuristics was to develop software to find the (not necessarily unique) optimal schedule for any given matrix, where an optimal schedule is defined to be one that obtains the proper encodings with the least possible number of XOR-operations. In this paper, we approach this problem in a set-based manner. Every schedule of XOR-operations for a bit-matrix with $m*w$ (heretofore r) rows and $k*w$ (c) columns can be represented as a set S of integers x where x is between 1 and 2^c-1. We consider the set

$$M = \{ y_1, y_2, \ldots, y_r \}$$

where y_j is the integer that is obtained when the j^{th} row of the CDM is considered as a binary number. Figure 8 shows a bit-matrix and its corresponding set M.



$$M = \{55, 44, 54, 59, 29, 46\}$$

Figure 8: Example of a bit-matrix and its corresponding M set

We then define the relationship *X(S, S')*:

$$X(S,S') \Leftrightarrow S' = S \cup \{x\} \text{ where } x = x_1 \oplus x_2, \quad x_1, x_2 \in S \wedge x_1 \neq x_2 \wedge x \notin S.$$

In other words, a set S' can be obtained from any set S by XORing two distinct items in S together and obtaining a third distinct item that is not a member of S. For example, $X(A,A')$ where $A = \{1, 2, 4, 8\}$ and $A = \{1, 2, 3, 4, 8\}$, because $1 \oplus 2 = 3$ and $3 \notin A$.

Describing schedules in this way, we define $S \rightarrow S'$ if there exist S_0, S_1, $\cdots$, S_n such that $S_0 = S$ and $S_n = S''$, such that

$$X(S,S') \quad \forall\, 0 \leq i < n$$

In determining the optimal schedule for a problem, one begins with the set S_I where

$$S_I = \{ 2^j : 0 \leq j < c \},$$

and the set M as defined above. Note that S_I represents the set of binary numbers less than 2^c that

contain a single one. The goal is to find the smallest set S' such that

$$S_I \rightarrow S' \wedge M \subset S'.$$

Then, the minimum number of XOR-operations required to encode data disks using the bit-matrix defined by M is $|S'| - |S_I|$. The optimal schedule can then be interpreted using the members of S'. Figure 9 shows an example of a bit-matrix, the optimal set S', and the schedule that corresponds to the optimal set S', as well as how the set S' is translated into the schedule.

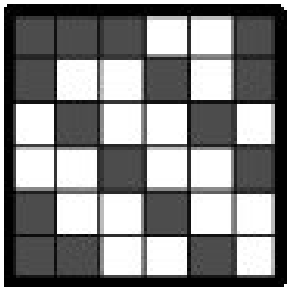
<table>
<tr><td colspan="2">$S' = \{1, 2, 4, 8, 9, 11, 16, 18, 32, 36, 37, 50, 57\}$</td></tr>
<tr><td>  </td><td>$1 \oplus 8 = 9 \rightarrow c_{0,3} = d_{0,2} \oplus d_{0,5}$
$2 \oplus 16 = 18 \rightarrow c_{0,2} = d_{0,1} \oplus d_{0,4}$
$32 \oplus 4 = 36 \rightarrow c_{0,4} = d_{0,0} \oplus d_{0,3}$
$36 \oplus 1 = 37 \rightarrow c_{0,1} = c_{0,4} \oplus d_{0,5}$
$18 \oplus 32 = 50 \rightarrow c_{0,5} = c_{0,2} \oplus d_{0,0}$
$9 \oplus 2 = 11 \wedge 11 \oplus 50 = 57 \rightarrow c_{0,0} = c_{0,3} \oplus d_{0,4} \oplus c_{0,}$</td></tr>
</table>

Figure 9: Example of set S' to schedule for a given bit-matrix

Note that in Figure 9, the number of XOR-operations in the schedule (seven) is exactly equal to $|S'| - |S_I|$ (because $|S'| = 13$, while $|S_I| = 6$).

Because every schedule can be represented using a set of integers between 1 and 2^c-1 as described above, an upper bound on the total number of schedules for a bit-matrix with r rows and c columns is the number of subsets of a set with 2^c items. That is, the total number of schedules to possibly consider is:

$$S(c) = 2^{2^c}$$

As represented in Figure 10, $S(c)$ grows exponentially. In erasure coding, w is typically less than or equal to 64, while k can be arbitrarily large, so this number of schedules ($S(c)$ where $c = kw$) is obviously not feasible to consider for even small values of k and w.

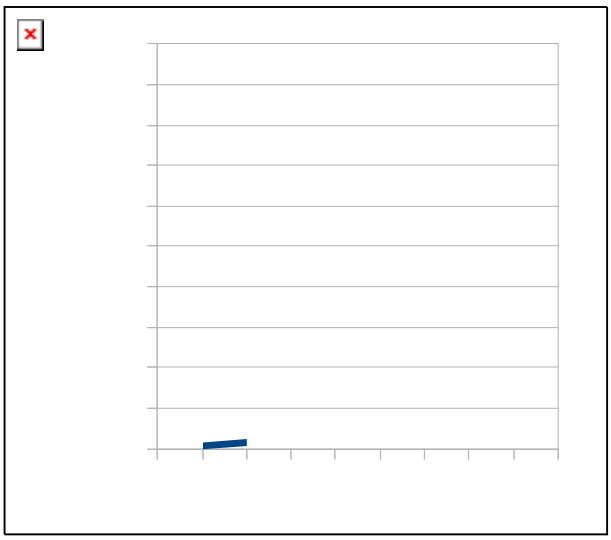


Figure 10: S(c)

However, this number of schedules should never be considered. In general, only a relatively small subset of the total number of schedules should be considered. For any given bit-matrix, the maximum size of sets to be considered can be determined by considering the number of operations in the worst possible schedule, the "naïve" schedule as described in Section 4.

The number of valid schedules (schedules that obey the $S_I \rightarrow S'$ relationship) with j XOR-operations (the number of sets that contain $r+j$ items) is relatively large. The number of unique schedules with exactly one operation is

$$\binom{c}{2} = \frac{1}{2}\left(c^2 - c\right)$$

because an operation can be represented as the combination of any two items in the set. Though this computation is relatively straightforward, the number of unique schedules with exactly two operations has a much more complicated computation. The number of unique schedules with exactly two operations is

$$\binom{c}{2}\left(c + \binom{c}{2} - 3\right) - \frac{1}{2}\binom{c}{2}\left(\binom{c}{2} - 1\right) = \frac{1}{8}\left(c^4 + 2c^3 - 13c^2 + 10c\right).$$

Already, the number of unique schedules with exactly two operations is growing with complexity $O(c^4)$. Because the minimum number of operations required in a schedule is r (in the case in which each row of the CDM has exactly two ones), the number of unique schedules to consider is still quite large.

In developing the code to find the optimal schedule for any matrix, we initially approached the problem using breadth-first search. That is, we first checked every schedule with one operation, then every schedule with two operations, until a schedule that achieved the matrix was found. In this algorithm, in order to determine each schedule with c operations, all schedules with $c-1$

operations are needed.

Because the number of potential schedules with *c−1* operations grows exponentially quickly, this method requires a great deal of memory. In fact, the machines we used ran out of memory before a schedule was found.

The next step in the development of algorithm required a sacrifice of time rather than space. The algorithm employed takes longer to find the optimal schedule, but it uses less memory. Instead of using a breadth-first search approach, we used a depth-first search approach. In this methodology, each schedule is encoded into a string and cached in a red-black tree. Operations are then added to a given schedule, expanding that schedule until one of the following conditions was met:

- The number of operations in the given schedule is greater than the maximum number of operations possible for the given matrix.
- The sum of the number of operations required to obtain the final BDM and the total number of operations in the given schedule is greater than the maximum number of operations.
- The number of operations in the given schedule is greater than the number of operations in the best schedule found thus far.
- The sum of the number of operations required to obtain the final BDM and the total number of operations in the given schedule is greater than the number of operations in the best schedule found thus far.
- The schedule has already been checked (that is, the schedule is cached in the red-black tree).
- The schedule achieves the matrix.

In this way, a smaller number of schedules is checked at a time, and memory is conserved. However, because of the caching of the schedules and the sheer number of schedules with a certain number of operations, the machine still ran out of memory before schedules were found for certain matrices.

The final step in finding the optimal schedule for any matrix was to remove the caching of the schedules from the algorithm. Again, this was a sacrifice of time for space. The same schedule may be checked many times over but virtually no memory is used. For instance, consider the number of schedules with two operations. The total number of schedules with two operations that are checked in this algorithm is

$$\binom{c}{2}\left(c+\binom{c}{2}-3\right)=\frac{1}{4}\left(c^4-7c^2+6c\right)$$

which, as shown in Figure 11, grows more quickly with c than the number of unique schedules with two operations. Not only are the schedules themselves checked multiple times over, but all schedules they are related to are checked multiple times over. As such, this final step was used as a

last resort to find the optimal schedule for some of the Cauchy matrices in which all other algorithms failed because of lack of memory. This algorithm often took over twenty-four hours to find a solution for even these small cases.

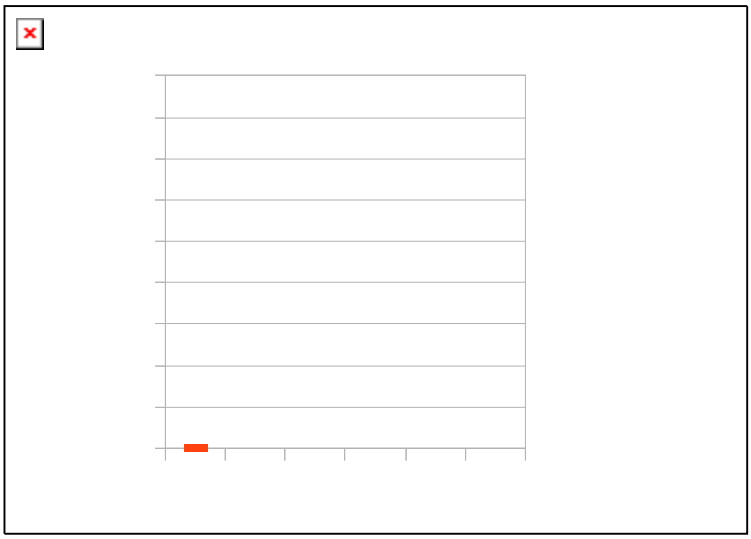


Figure 11: Difference between all schedules checked and unique schedules with two operations

7. Analysis of Two Known Heuristics

Having found the optimal schedule for each of the 63 Cauchy matrices in our test space, we set about comparing these schedules to the two known heuristics. For the Code Specific Hybrid Reconstruction (CSHR) algorithm, the implementation in the **Jerasure** library was modified so that the output would be in the same set notation as described in section 6. Figure 12 shows the comparison between the optimal schedule and the schedule produced by the CSHR algorithm.

As expected, CSHR performs optimally for extremely sparse matrices. However, for codes where entire sub-diagonals or partial sub-diagonals of the bit-matrix are zeros, CSHR performs especially badly. In these cases, the CSHR schedule includes more than three unnecessary operations. Figure 13 shows examples of such matrices. CSHR performs sub-optimally on 33 of the 63 test matrices.

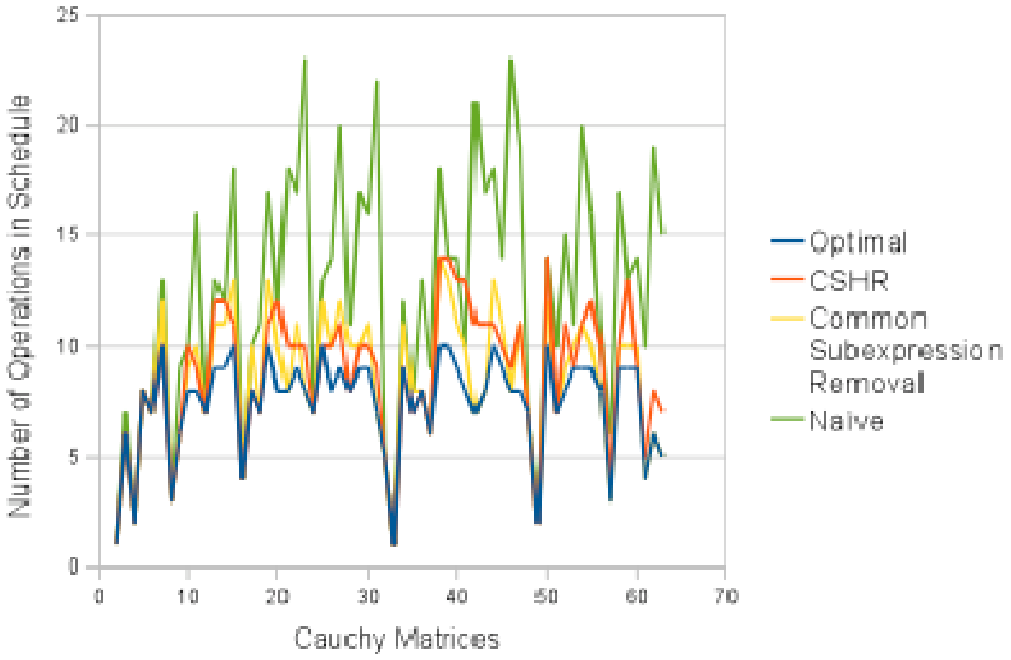


Figure 12: Comparison between schedules

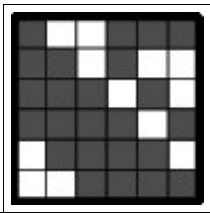
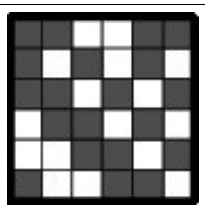
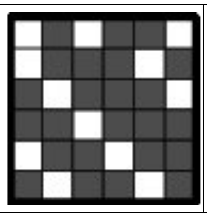
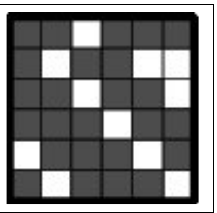
				
Optimal	10	10	10	8
CSHR	13	13	13	11
Common Subexpression Removal	11	14	11	11
Naïve	18	14	18	19

Figure 13: Examples of bit-matrices in which CSHR performs badly and number of operations

For the Huang, et. al., implementation, the algorithm was built around existing code of Edmond's matching algorithm for non-bipartite graphs. Again, this algorithm was tweaked so that the output matches the set notation output of the optimum schedule algorithm. Figure 12 shows the comparison between the schedule produced by this algorithm and the optimal schedule.

As in CSHR, this algorithm performs optimally for extremely sparse matrices. Like CSHR, this algorithm struggles with schedules for codes where entire sub-diagonals or partial sub-diagonals of the bit-matrix are zeros. It also struggles in particular with codes where subsequent sub-diagonals

contain few ones or none at all, such as those shown in Figure 14. This algorithm performs sub-optimally on 41 out of 63 of the test matrices.

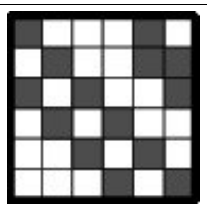
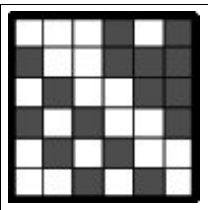
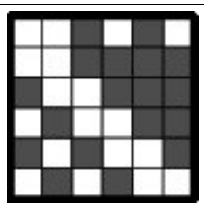
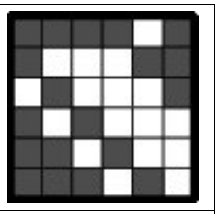
				
Optimal	8	8	8	9
CSHR	10	9	10	10
Common Subexpression Removal	12	10	12	13
Naïve	12	10	12	13

Figure 14: Examples of bit-matrices in which the common subexpression removal algorithm performs badly and number of operations

8. The Development of a New Heuristic

As part of the exploration of current heuristics, we attempted to the development of a new heuristic that could potentially outperform the known heuristics. We began with the optimization code described above. The first step in developing a heuristic was to modify the ordering of the depth first search to find a better schedule more quickly. This is initially determined by considering all schedules with exactly one operation. The schedules are then ranked based on whether or not the result of the operation was represented in the actual bit-matrix. For example, if a bit-matrix is represented by the set $M = \{3, 34, 49, 24, 12, 6\}$, the schedule that included the operation $1 \oplus 2 = 3$ is ranked above all other schedules. This is the path that is pursued initially in the depth-first search.

The schedule this algorithm returns is the first schedule that is found, regardless of length. This heuristic is not competitive at all. In fact, it is often unable to find any schedule other than the worst possible schedule within any reasonable amount of time. However, upon implementing this "best-path" ranking within the optimization code, we found that it improves the performance slightly.

Because the performance is improved in the optimization code, the next step in the development of the new heuristic was to look ahead to all schedules with two operations. Again, the ranking of the paths is based on which schedules contained the most XOR-operations that resulted in values represented in the bit-matrix. This heuristic is more successful than the first. In fact, for many of the test matrices it produces either the optimal matrix, or a schedule with operations equal to the

number of operations provided in the two known heuristics. However, for certain test matrices, the algorithm is not able to produce a schedule in any reasonable amount of time. Thus, there is still room for improvement in developing a new heuristic.

9. Conclusion and Future Work

Constructing the optimal schedule for an arbitrary bit-matrix proved to be extremely expensive, both with respect to computation times and with respect to the amount of memory required to minimize computation time. The setbacks in putting together an algorithm that yields a solution in a reasonable amount of time for any given matrix did not allow for as much of an in-depth analysis as planned. However, we learned much in the construction of the algorithm that can potentially be useful in developing a new heuristic.

Using what we learned in the development of the optimal schedule algorithm and the initial heuristic work, we plan to continue developing new heuristics to solve this problem with the goal of finding a heuristic that not only competes with the two known heuristics, but also outperforms them. Because of our analysis of the two known heuristics, we were able to identify the types of matrices in which these heuristics provided weak solutions. We plan to continue expand the analysis of these algorithms to a larger set of test matrices in order to identify other types of matrices in which they also provide weak solutions in order to avoid some of these weaknesses in the continued development of the heuristic.

References:

[1] Blaum M., and R. M. Roth. "On lowest density MDS codes." IEEE Transactions on Information Theory, 45(1):46-59, January 1999.

[2] Blomer J., M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman. "An XOR-based erasure-resilient coding scheme." Technical Report TR-95-048, International Computer Science Institute, August 1995.

[3] Cleversafe, Inc. Cleversafe Dispersed Storage. Open source code distribution: http://www.cleversafe.org/downloads, 2008.

[4] 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.

[5] Hafner, J.L, V. Deenadhayalan, K. K. Rao, and A. Tomlin. "Matrix methods for lost data reconstruction in erasure codes." In *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, pages 183-196, San Francisco, December 2005.

[6] Hafner, J. L. “HoVer erasure codes for disk arrays.” In DSN-2006: The International Conference on Dependable Systems and Networks (Philadelphia, June 2006), IEEE.

[7] Huang, C., M. Chen, and J. Li. “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).

[8] Huang, C., J. Li, and M. Chen. “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.

[9] Plank, J. S. ``Erasure Codes for Storage Applications,'' Tutorial, *FAST-2005: 4th Usenix Conference on File and Storage Technologies* San Francisco, CA, December 2005.

[10] Plank, J.S., and L. Xu. “Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications.” In NCA-06: 5th IEEE International Symposium on Network Computing Applications, Cambridge, MA, July 2006.

[11] Plank, J.S. “The RAID-6 Liberation codes.” In FAST-2008: 6th Usenix Conference on File and Storage Technologies, pages 97-110, San Jose, February 2008.

[12] Plank, J. S. “Raid-6 Liber8Tion Code.” In *The International Journal of High Performance Computing Applications,* Volume 23, Number 3, pages 242-251, 2009.

[13] Plank, J.S., S. Simmerman and C. D. Schuman. ``**Jerasure**: A Library in C/C++ Facilitating Erasure Coding for Storage Applications,'' Technical Report CS-08-627, University of Tennessee Department of Electrical Engineering and Computer Science, August, 2008.

[14] Rhea, S., C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. “Maintenance-free global data storage.” IEEE Internet Computing 5, 5 (2001), 40-49.

[15] Storer, M. W., K. M. Greenan, E. L. Miller, and K. Voruganti. “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.

[16] Wylie, J. J., and R. Swaminathan. “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.

[17] Zhu, B., K. Li, and H. Patterson. “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.