Representing X-Code Using Latin Squares*

Gang Wang, Sheng Lin, Xiaoguang Liu and Jing Liu
Nankai-Baidu Joint Lab, College of Information Technology Science, Nankai University
Weijin Road 94, Tianjin, China
wgzwp@163.com, shshsh.0510@gmail.com, liuxg74@yahoo.com.cn, jingliu@nankai.edu.cn

Abstract

X-Code is an important 2-erasure correcting vertical array code. In this paper, we present a combinatorial representation for X-Code. It is based on graph representation of binary linear/array codes. We represent X-Code by Cayley tables of the cyclic groups of prime order - a special column-Hamiltonian Latin squares (CHLS). This representation is helpful to comprehend X-Code's 2-erasure correcting ability. We also show the inner relationship between X-Code and the Liberation codes though the former is a vertical code and the latter is a horizontal code. A possible way to construct the 2-erasure vertical codes like X-Code using common CHLS is also proposed.

1. Introduction

In recent years, as hard disks have grown greatly in size and storage systems have grown in size and complexity, it is more frequent that a failure of one disk occurs in tandem with unrecovered failures of other disks or latent failures of blocks on other disks. On a system using single-erasure correcting code such as RAID-5, this combination of failures leads to a permanent data loss [1]. Thus, applications of multi-erasure correcting codes have become more pervasive. X-Code is an important 2-erasure correcting array code [2]. It requires that the number of disks must be a prime, which is the major disadvantage of it. This paper presents a combinatorial representation for X-Code, shows the correspondence between X-Code and the Liberation codes [3], and gives a possible way to construct X-like codes with non-prime size using common column-Hamiltonian Latin squares.

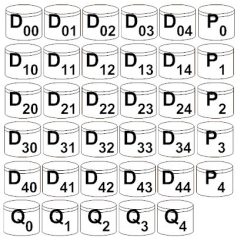
*This paper is supported partly by the National High Technology Research and Development Program of China (2008AA01Z401), RFDP of China (20070055054), and Science and Technology Development Plan of Tianjin (08JCYBJC13000)

2. Related Work

An erasure code for storage systems is a scheme that encodes the content on n data disks into m check disks so that the system is resilient to any m device failures [4]. Unfortunately, there is no consensus on the best coding technique for $n, m > 1$. All the known coding schemes have trade-offs.

The best known multi-erasure codes are *Reed-Solomon codes* [5]. High computational complexity is a serious problem because of Galois Field based operations.

Binary linear codes [6] are inherently XOR-based, hence have perfect computational complexity. Fig 1.a shows a *2d-parity code* [6], where D_{ij} denotes a data disk that participates in the i^{th} horizontal parity group (P_i is the parity disk) and the j^{th} vertical parity group (Q_j is the parity disk). We can see that binary linear codes divide data disks into overlapping parity groups to tolerate multi-erasures. Bad storage efficiency is their major drawback.



a. A 35-disks 2d-parity code

disk0	disk1	disk2	disk3	disk4
D_{00}	D_{41}	D_{32}	D_{23}	D_{14}
D_{11}	D_{02}	D_{43}	D_{34}	D_{20}
D_{22}	D_{13}	D_{04}	D_{40}	D_{31}
P_3	P_2	P_1	P_0	P_4
Q_3	Q_4	Q_0	Q_1	Q_2

b. The 5-disk X-Code.

Figure 1. 2d-parity code and X-Code.

Another category is *parity array codes*. This kind of codes arrange symbols into an array and divide them into overlapping parity groups, hence the name. Array codes try to combine the advantages of RS codes and binary linear codes - they inherit linear codes' pure parity architecture and use less disks. EVENODD [7], RDP [1] and Liberation [3] are typical *horizontal codes* in which data and parity are stored separately. X-Code [2] and B-Code [8] are

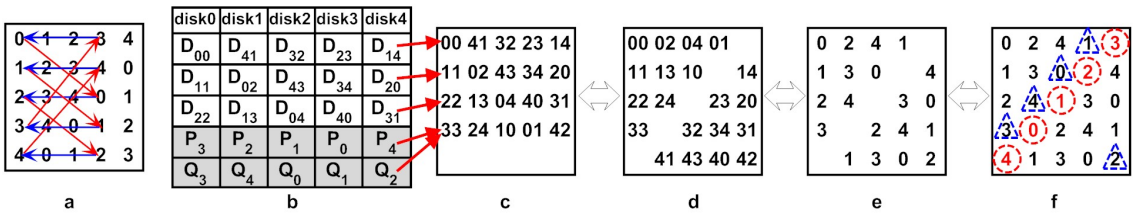


Figure 2. X-Code and LS.

typical vertical codes in which data and parity are stored together. Vertical codes have some advantages over horizontal codes: fit distributed systems well; optimal encoding performance.

In this paper, we focus mainly on X-Code, an important 2-erasure vertical code. As shown in Fig 1.b, a p-disk X-Code system is described by a $p \times p$ array (p must be a prime). Each column denotes a disk and each symbol denotes a disk block. X-Code organizes the parity groups along diagonals and skew diagonals - this is the reason why it was named 'X'-Code. Because all parities are independent and every data symbol belongs to exactly two parity groups, X-Code achieves optimal encoding performance.

Array codes can be regarded as data layouts of binary linear codes. For example, the 5-disk X-Code can be constructed by deleting D_{01}, D_{03}, D_{10}, D_{12}, D_{21}, D_{24}, D_{30}, D_{33}, D_{42}, and D_{44} from the 2d-parity code shown in Fig 1.a and then packing the remaining symbols into 5 disks.

3. Related Combinatorics Knowledge

Some literature refers to simple graph representation of 2-erasure binary linear codes [6, 8, 9]: each vertex denotes a parity symbol (group) and each edge denotes a data symbol - the two endpoints of an edge is just the two parities of the data. So a 2d-parity code is described by a bipartite graph. An array code can be described by a graph partition in which each sub-graph denotes a disk. We have proven the following theorem [9]:

Theorem 1. *If an array code can be described by a partition of a simple graph, it is a 2-erasure code iff the union of any pair of sub-graphs of the partition doesn't contain the following two types of structures:*

1) *A path and its two endpoints. We call this kind of unrecoverable erasure* ***C****losed* ***P****arity* ***S****ymbols* ***S****ubset, CPSS for short.*
2) *A cycle. We call it CDSS -* ***C****losed* ***D****ata* ***S****ymbols* ***S****ubset.*

Theorem 1 is intuitive. We know that a parity group tolerates only single-erasures. For two adjacent edges X and Y, because they belong to the same parity group, one of them, say X must be recovered by another group (in which X is the only failed symbol) and then Y is recovered by their common group. In other words, X and Y are *decoding dependent*. Therefore a closed path mentioned above means a unresolvable dependency chain, namely the erasure can't be recovered. On the contrary, for an open path, the end edge (the only failed symbol in its group) is recovered first, and other edges and vertices can be recovered one by one.

Theorem 1 doesn't suggest how to construct array codes. *Perfect one-factorization of graph* (P1F) [10] is a useful tool for 2-erasure array codes constructing. A *one-factor* of a graph G is a set of edges in which every vertex appears exactly once. A *one-factorization* (1F) of G is a partition of the edge-set of G into one-factors. A perfect one-factorization is a one-factorization in which every pair of distinct one-factors forms a Hamiltonian cycle. There is a widely believed conjecture in graph theory: every complete graph with an even number of vertices has a P1F [10].

Latin square is another useful tool for array code constructing. For $k \leq n$, a $k \times n$ *Latin rectangle* is a $k \times n$ matrix of entries chosen from some set of symbols of cardinality n, so that no symbol is duplicated within any row or any column. In this paper, we use $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ as the symbol set. When $k = n$, the Latin rectangles are called Latin squares of order n. The symbol in row r, column c of a Latin rectangle R is denoted by R_{rc}. A Latin square of order n also can be described by a set of n^2 triples of the form $(row, column, symbol)$.

Each row r of a Latin rectangle R is the image of some permutation σ_r of the symbol set, namely $R_{ri} = \sigma_r(i)$. Each pair of rows (r, s) defines a permutation by $\sigma_{r,s} = \sigma_r \sigma_s^{-1}$. If $\sigma_{r,s}$ consists of a single cycle for each pair of rows (r, s) in a Latin square L, we say L is *row-hamiltonian*. Similar concepts can be defined in terms of the column and symbol. In this paper, we are concerned with *column-hamiltonian Latin squares* (CHLS). Fig 2.a shows a CHLS of order 5. It belongs to a special family of LS - Cayley tables of the cyclic groups. The symbol in row r, column c of a Cayley table C_n of the cyclic group of order n is $(r + c)$ mod n. Apparently, C_n is a CHLS iff n is a prime. $\sigma_{0,3}$ of C_5 is also shown in Fig 2.a.

There is a CHLS L of order n iff the complete bipartite graph $K_{n,n} = (V, W, E)$ has a P1F $F = \{F_0, , F_{n-1}\}$ [10].

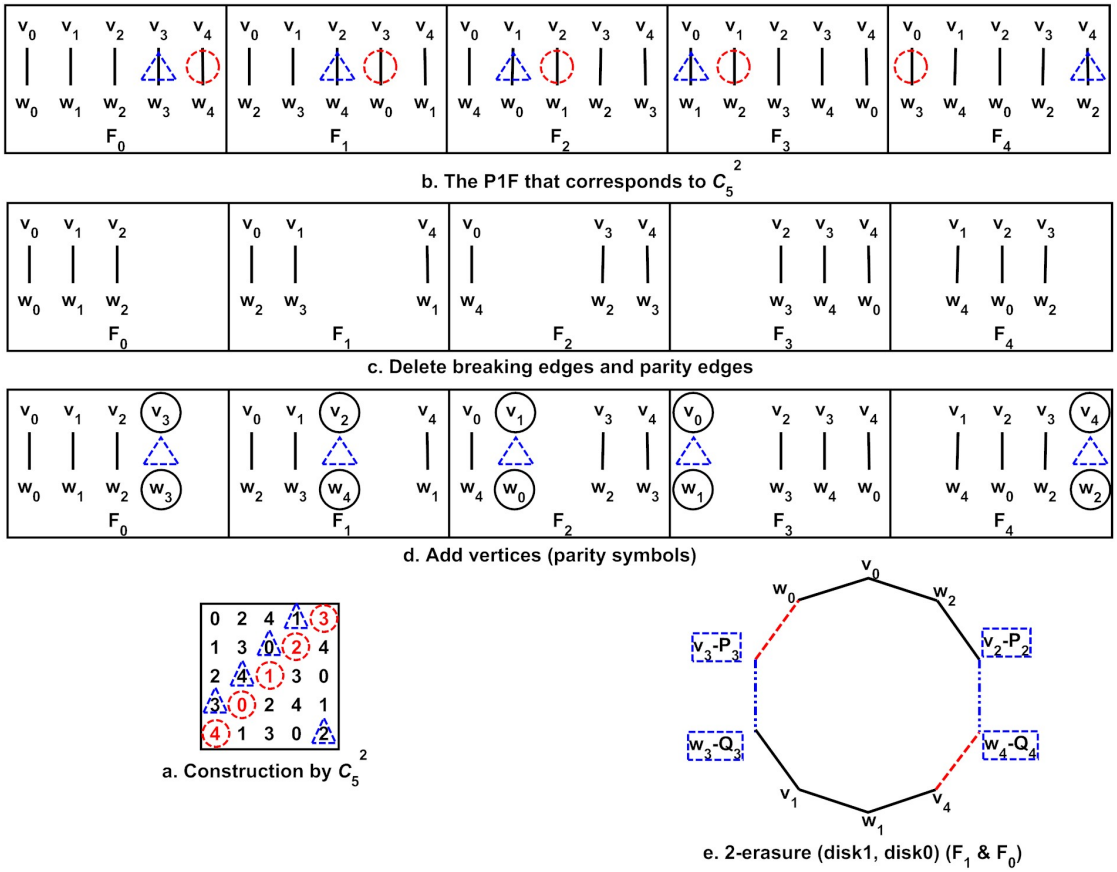


Figure 3. X-Code and P1F.

To show this, we convert $(i, j, k)(\in\ L)$ into the edge (v_i, w_k) in F_j. Thus the cycle pattern in $\sigma_{r,s}$ in L corresponds to that in $F_r \cup F_s$. There is another conclusion [10]: if the complete graph K_{n+1} has a P1F, then so does $K_{n,n}$. Thus we have a conjecture: $K_{n,n}$ has a P1F (CHLS of order n exists) for $n = 2$ and all odd positive integers n. Graph theorists have found some families of "P1F number" and many other individual P1F numbers. Thus array codes based on P1F(CHLS) are superior to other array codes (with "prime size" restriction) in parameter flexibility.

4. Combinatorial Representation For X-Code

A p-disk X-Code can be described by a $p \times p$ Latin square. Fig 2.b-f shows how to translate the 5-disk X-Code into a LS of order 5. We denote each data symbol by a pair (P_index, Q_index), each parity symbol by its index. So the 5-disk X-Code (Fig 2.b) is translated into the array in Fig 2.c. Note that two parity rows are combined and placed in row 4 and row 5 is blank.

Next, we shift all pairs in column i to the up by i steps (with wraparound). Fig 2.d shows the result. Now, all pairs in row i belong to the i^{th} diagonal parity group. So we delete the first index (diagonal parity index) from each pair. The array becomes the form shown in Fig 2.e. Finally, we construct an LS L by filling the missing number into the blank position in each row (column). Fig 2.f shows L. We can see that $L_{ij} = ((i + 2j) \bmod n)$ and L is isomorphic to C_n. We denote it by C_n^2. We can similarly denote LS composed of symbols $(i, j, (i + mj) \bmod n)$ by C_n^m.

This example shows the injection from X-Code to C_n^2 of prime order. In fact, the inverse method converts a C_n^2 of prime order into a X-Code. Thus there is a bijection between X-Code and C_n^2 of prime order. As Fig 2 shows, each symbol in the LS produces a data/parity symbol in the X-Code. Its column index designates the disk, its row index designates the diagonal parity group, and the symbol itself designates the skew diagonal parity group. The symbols in the secondary diagonal of LS are unused, we call them *the U_symbols*. Each symbol in the skew diagonal above the secondary diagonal constructs a diagonal parity symbol and a skew diagonal parity symbol, we call it *the P_symbol*. Other symbols are used to construct data symbols, they are called *the D_symbols*. In Fig 2.f, U_symbols

are surrounded by circles, and P_symbols are surrounded by triangles. Combinatorics theorists call them *transversals*. A transversal in a LS of order n is a set of n symbols, one from each row and column, containing each of the n symbols exactly once. We can see that the diagonal and skew diagonal in a C_n^m are transversals.

Can the codes produced by the method described above tolerate any 2-erasure? Certainly we can verify easily that the codes conform to the definition of X-Code, therefore are 2-erasure code. Now we prove this using a combinatorial method. The proof is based on theorem 1, so we give graph representation for X-Code first.

Algorithm 1 Constructing X Code by P1F

Require: A P1F $F = \{F_0, \ldots, F_{p-1}\}$ of $K_{p,p}$. $F_j = \{(v_i, w_{(i+2j)\%p}) | 0 \le i \le p-1\}, 0 \le j \le p-1$.

Ensure: The p-disk X-Code.

1: Delete edges $(v_{p-1-j}, w_{(j-1)\%p})$ and $(v_{(p-2-j)\%p}, w_{j-2\%p})$ from each F_j $(0 \le j \le p-1)$. We call the former *the breaking edge* and the latter *the parity edge*.

2: Add vertices $v_{(p-2-j)\%p}$ and $w_{(j-2)\%p}$ into each F_j $(0 \le j \le p-1)$.

3: For each $(v_a, w_b) \in F_j$ $(0 \le j \le p-1)$, place the data symbol that participates in P_a and Q_b in the j^{th} disk. For each $v_a \in F_j$, place P_a in the j^{th} disk. For each $w_b \in F_j$, place Q_b in the j^{th} disk.

We can see that F corresponds to C_n^2, and edges $(v_{p-1-j}, w_{(j-1)\%p})$ are just U_symbols, and edges $(v_{(p-2-j)\%p}, w_{(j-2\%p})$ are just P_symbols. Fig 3 shows the procedure of performing algorithm 1 on the P1F corresponding to C_5^2. Fig 3.c corresponds to step 1, and Fig 3.d corresponds to step 2. Because each pair of factors F_i and F_j forms a Hamiltonian cycle, each pair are broken into four paths by step 1. Therefore there are no CDSS in any pair of disks. We add two vertices into each disk in step 2. They are just the two vertices of the parity edge deleted from the same factor in step 1. Therefore, in order to prove the code 2-erasure, the only thing we need to do is to prove that there are no CPSS in any pair of disk. Fig 3.e shows a 2-erasure (disk0, disk1) (F_0 and F_1) in a 5-disk X-Code system. The Hamiltonian cycle is broken into four paths ($w_0 - v_0 - w_2 - v_2$, empty, $v_4 - w_1 - v_1 - w_3$ and empty, in fact two paths) by deleting two breaking edges ((v_4, w_4) and (v_3, w_0)) and two parity edges ((v_3, w_3) and (v_2, w_4)) (denoted by dashed lines). It is easy to see that there are no CPSS iff the breaking edges and the parity edges appear interleaved in the cycle (therefore four parity vertices just fall into different paths). This 2-erasure indeed satisfies this condition. But how about other 2-erasures?

Lemma 2. *For any pair of disks in a p-disk code constructed by algorithm 1, the two parity edges and the two breaking edges appear in the cycle interleaved.*

Proof. Suppose the i^{th} disk and the j^{th} disk fail, $0 \le i < j \le p-1$. All of the operations are with mod p.

According to algorithm 1, the breaking edges are (v_{p-1-i}, w_{i-1}) and (v_{p-1-j}, w_{j-1}), and the parity edges are (v_{p-2-i}, w_{i-2}) and (v_{p-2-j}, w_{j-2}). We rewrite them as (v_{-i-1}, w_{i-1}), (v_{-j-1}, w_{j-1}), (v_{-i-2}, w_{i-2}) and (v_{-j-2}, w_{j-2}).

We travel the Hamiltonian cycle induced by F_i and F_j along the same direction as $\sigma_{i,j}$ in LS (anticlockwise in Fig 3) from (v_{-i-2}, w_{i-2}). The visit path is $v_{-i-2} - w_{i-2} - v_{-i-2+2(i-j)} - w_{i-2+2(i-j)} - \cdots - v_{-i-2+2k(i-j)} - w_{i-2+2k(i-j)} - \cdots - w_{i-2+2(p-1)(i-j)} - v_{-i-2+2p(i-j)} (= v_{-i-2})$. k is the sequence number of the vertices in the cycle. We represent the relative order of (v_{-i-1}, w_{i-1}), (v_{-j-1}, w_{j-1}), (v_{-i-2}, w_{i-2}) and (v_{-j-2}, w_{j-2}) by the sequence numbers of v_{-i-2}, v_{-i-1}, w_{j-2} and w_{j-1}. The sequence number of v_{-i-2} is obviously 0, suppose the sequence numbers of other three vertices are x, y and z respectively, so

$$\begin{aligned} -i-1 &\equiv -i-2+2x(i-j) \bmod p \\ &\Rightarrow 2x(i-j) \equiv 1 \bmod p \quad (1) \\ j-2 &\equiv i-2+2y(i-j) \bmod p \\ &\Rightarrow (2y+1)(i-j) \equiv 0 \bmod p \quad (2) \\ j-1 &\equiv i-2+2z(i-j) \bmod p \\ &\Rightarrow (2z+1)(i-j) \equiv 1 \bmod p \quad (3) \end{aligned}$$

Because i, j are different integers in $[0, p-1]$, x, y and z are integers in $[1, p-1]$ and p is a prime, $(i-j)$ and p have no common divisors except 1. According to equation 2, $(2y+1) \equiv \bmod p$. Moreover $3 \le 2y+1 \le 2p-1$, thus y must be $(p+1)/2$.

Equation 1 plus equation 2 minus equation 3 derives $2(x+y-z)(i-j) \equiv 0 \bmod p$, thus $2(x+y-z) \equiv 0 \bmod p$. So, $x+y = z$ or $x+y = z+p$. The former derives $0 < x \le y < z$. $x = y$ means that the breaking edge (v_{-i-1}, w_{i-1}) and the parity edge (v_{-j-2}, w_{j-2}) have a common vertex $w_{i-1} (= w_{j-2})$. $x+y = z+p$ derives $0 < z < y < x$. Anyway, breaking edges and parity edges appear in the cycle interleaved. □

So the following theorem is obvious.

Theorem 3. *The codes constructed by algorithm 1 are all 2-erasure codes.*

5. Relationship Between X-Code and Liberation Code

The Liberation code is a new kind of 2-erasure horizontal code presented recently [3]. Unlike most other array codes, its construction is based on "addition" instead of "subtraction" - it is constructed by arranging the third parity group

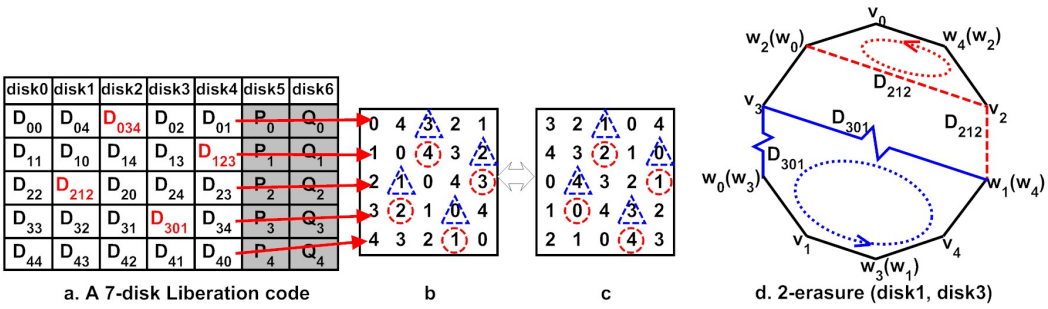


Figure 4. Liberation code.

for some data symbols instead of deleting some symbols. Fig 4.a shows a 7-disk Liberation code. D_{ij} denotes the data symbol that participates in parity groups P_i and Q_j. D_{ijk} denotes the data symbol that participates in P_i, Q_j and Q_k. This example belongs to the family presented in [3] that can be described as:

1) The number of data disks is a prime p. Each disk contains p symbols. The first parity disk stores horizontal parity symbols, and the second parity disk stores diagonal parity symbols.
2) The i^{th} data symbol in the j^{th} disk participates in the i^{th} horizontal parity symbol P_i and the $(i-j)^{th}$ diagonal parity symbol $Q_{(i-j)}$.
3) When j is odd, the $(\frac{p+j}{2}-1)_{th}$ data symbol in the j^{th} disk is the "3-group" data symbol and its third parity symbol is $Q_{\frac{p-j}{2}}$. When j is even, the 3-group symbol is in row $\frac{j}{2}-1$ and its third parity symbol is $Q_{p-\frac{j}{2}}$.

We call this family of Liberation codes *the Liberation-1 codes*. They can be described by CHLS. Let's take the 7-disk Liberation code in Fig 4.a for instance. If we represent each data symbol by its second parity index, the code is translated into an LS of order 5 as depicted in Fig 4.b. The first parity index of each data symbol is just the row index of the LS symbol, so we don't write it explicitly. We can see this LS is just C_5^4. In fact, the $(p+2)$-disk Liberation-1 code corresponds to C_p^{p-1}. Next, we deal with the third parity groups of 3-group data symbols. For D_{ijk} (in disk $(i-j)$), we mark the symbol 'k' in column $(i-j)$ by a circle in the LS, and mark the symbol 'j' by a triangle as Fig 4.b shows. It's very close to the LS in Fig 2.f. In fact, we can transform it into the latter. We perform a permutation $\{0\to 3, 1\to 4, 2\to 0, 3\to 1, 4\to 2\}$ on the symbol set. So the LS is transformed into the LS shown in Fig 4.c. It is almost the LS in Fig 2.f after proper column rearrangement. The only difference is that its first column is not marked because the first data disk contains no 3-group data symbol.

We have proven the Liberation-1 codes 2-erasure in [11]. The key idea appears as depicted in Fig 4.d. The circle corresponds to the 2-erasure (disk1, disk3). The two dashed lines both denote the 3-group data symbol D_{212} in disk 1, and the two wavy lines both denote D_{301} in disk 3. So we call w_0 and w_1 *the real 3-vertices*, because they touch three edges (data symbols). We call v_2 and v_3 *the fake 3-vertices* because they in fact touch only two edges. If a 2-erasure composed of disk 0 and another data disk, it can be recovered definitely. If a 2-erasure composed of two data disks excluding disk 0, it can be recovered iff the real and fake 3-vertices appear on the cycle interleaved. We have shown that the Liberation-1 codes satisfy this condition [11]. The method used is similar to that used in the proof of lemma 2.

The real 3-vertices correspond to the symbols surrounded by circles in Fig 4.d. Namely, they corresponds to U_symbols. The fake 3-vertices actually correspond to P_symbols. If we perform the permutation used in the last paragraph but one on Fig 4.d (the w vertices are replaced by those in the parentheses), it is converted into Fig 3.e. The edges corresponding to 3-group data symbols are just the parity edges in Fig 3.e, and the edges incident to real 3-vertices are just the breaking edges. That is to say, there is an interesting internal relationship between X-Code and Liberation-1 code although the former is a vertical code and the latter is a horizontal code. We can convert a p-disk X-Code into a $(p+2)$-disk Liberation-1 code using LS as the intermedium . Namely, there is an injection from X-Code to Liberation-1 code. But given a Liberation-1 code, the corresponding X-Code may not exist.

6. Constructing X-Code Using CHLS Other Than C_p^2

A big limitation of X-Code is that the number of disks must be a prime. Examining algorithm 1 and lemma 2, we have a possible way to construct X-like codes. We replace the input of algorithm 1 - C_p^2 by a common CHLS of order n and two transversals of it. If the two transversals satisfy lemma 2, we will obtain an n-disk X-like code. Because CHLS of non-prime order exist, we may produce X-like code with non-prime size. We have written a search pro-

gram according to this idea. The algorithm takes a CHLS as input and searches its two transversals conforming to lemma 2. Compared with previous algorithm based on matrix operations (such as Plank's algorithm for Liberation code searching [11]), our algorithm prunes effectively.

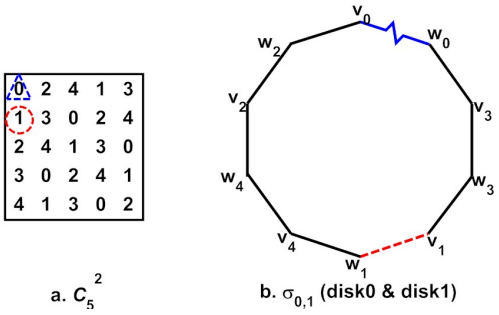


Figure 5. Effective code searching.

Suppose the input is C_n^2 (n doesn't have to be a prime). Without lemma 2, we have to enumerate all possible combinations of U_symbol and P_symbol for each column. Our algorithm does only about half of work for a column. Fig 5 shows the idea. Fig 5.a shows C_5^2 and the starting point of search - symbol '0' and symbol '1'. Fig 5.b shows the Hamiltonian cycle induced by column 0 and column 1. The breaking edge (v_1, w_1) and the parity edge (v_0, w_0) are denoted by a dashed line and a wavy line respectively. They divide the circle into two semicircles. A plain algorithm may check all possible positions of the U_symbol and the P_symbol in column 1, while our algorithm only searches in each semicircle. The reason is that if the breaking edge and the parity edge from column 1 fall into different semicircles, the two breaking edges and the two parity edges from column 0 and column 1 are not interleaved. The plain algorithm enumerates about $\binom{n}{2}$ possibilities for a column, while our algorithm enumerates only about $\binom{\frac{n}{2}}{2}$ possibilities. The total workload our algorithm is about $\frac{1}{2^n}$ of the plain algorithm. Moreover, after a tentative code is produced, our algorithm doesn't really check whether it is 2-erasure like the plain algorithm, because lemma 2 has been satisfied.

Certainly, this algorithm is still an exponential algorithm. We have tested some small CHLS. But until now, we haven't found any code that is not isomorphic to standard X-Code.

7. Conclusion and Future Work

In this paper, we presented a combinatorial representation for X-Code. This representation is based on a graph representation of linear/array codes. We showed that there is a bijection between X-Code and a special family of CHLS. We also gave a combinatorial proof for the MDS property of X-Code. This method is helpful to comprehend the 2-erasure correcting ability of X-Code. We showed an interesting inner relationship between X-Code and the Liberation codes though the former is a kind of vertical code and the latter is a kind of horizontal code. Finally, we presented an effective algorithm to search X-like code.

Constructing X-like code with non-prime size by both theoretical method and algorithmic method is the most important future work. We think the algorithmic method is more promising. Certainly, we should optimize the search algorithm further. We must find more strong necessary conditions to prune more effectively.

References

[1] P. F. Corbett, R. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-Diagonal Parity for Double Disk Failure Correction. In *Proceedings of the FAST '04 Conference on File and Storage Technologies*, pages 1–14, San Francisco, California, USA, December 2004.

[2] L. Xu and J. Bruck. X-Code: MDS Array Codes with Optimal Encoding. *IEEE Transactions on Information Theory*, 45(1):272–276, 1999.

[3] J. S. Plank. The RAID-6 Liberation Codes. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies, FAST 2008*, pages 97–110, San Jose, CA, USA, February 2008.

[4] J. S. Plank. Erasure Codes for Storage Applications. Tutorial Slides, presented at *the 4th Usenix Conference on File and Storage Technologies, FAST 2005*, December 2005.

[5] J. S. Plank. A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-Like Systems. *Softw., Pract. Exper.*, 27(9):995–1012, 1997.

[6] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson. Coding Techniques for Handling Failures in Large Disk Arrays. *Algorithmica*, 12(2/3):182–208, 1994.

[7] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures. *IEEE Trans. Computers*, 44(2):192–202, 1995.

[8] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner. Low-density MDS Codes and Factors of Complete Graphs. *IEEE Transactions on Information Theory*, 45(6):1817–1836, 1999.

[9] J. Zhou, G. Wang, X. Liu, and J. Liu. The Study of Graph Decompositions and Placement of Parity and Data to Tolerate Two Failures in Disk Arrays: Conditions and Existance. *Chinese Journal of Computer*, 26(10):1379–1386, 2003.

[10] I. M. Wanless. Perfect Factorisations of Complete Bipartite Graphs and Latin Squares without Proper Subrectangles. *Electron. J. Combin*, 6(R9), 1999.

[11] G. Wang, X. Liu, S. Lin, G. Xiu, and J. Liu. Constructing Liberation Codes Using Latin Squares. In *14th IEEE Pacific Rim International Symposium on Dependable Computing, PRDC 2008*, pages 73–80, Taipei, Taiwan, December 2008.