Extractor Codes with Applications

Rong-Jaye Chen Ming-Yu Liu Chin-Yuan Teng Shi-Chun Tsai*

Abstract

Extractors are functions which can "extract" random bits from certain distributions that contain some randomness. There are many applications of extractors in complexity theory. Extractor codes are codes which use extractors to encode information. These codes have the very good distance property. Therefore they have the soft-decision decoding ability and could be suitable for highly noisy channels.

In this paper, we first explain extractors and some relations with codes. We introduce an explicit extractor code based on Trevisan's extractor. Then we show an application of extractor codes on EC-RAID. We also bring up explicit and efficient calculating and recovering algorithms in our EC-RAID system, so that this new system can offer high reliability and performance.

Keywords: Extractors, list decoding, soft-decision decoding, extractor codes, RAID.

1 Introduction

An extractor is a function to extract truly random bits from weak random sources. Extractors were first defined and constructed by Nisan and Zuckerman [5]. In recent years, an important discovery of the extractor is Trevisan's extractor. Trevisan [10] connected Nisan-Wigderson generator [4] with extractors. This extractor only needed $O(\log n)$ truly random bits , and the idea offered another way to construct extractors. Later there were several improved methods derived from Trevisan' extractor, as in [7, 9]. Ta-Shma and David Zuckerman [9] discovered that extractors have the good distance property and can be used in coding theory.

Our main contribution is to build a RAID-like system using extractor coding. This RAID-like system has high fault-tolerance ability, so it can offer good reliability. We bring up an explicit calculating algorithm to describe how to store data efficiently, and three explicit recovering algorithms in different serious situations to recover original data. We show it is possible to rescue original data even if there are more than half storage devices broken. This RAID-like system can be designed as the multi-level fault-tolerance system, so that the manager can maintain and classify data easily.

The organization of this paper is described as follows. We introduce soft decoding concepts and extractor codes in Section 2. We introduce Trevisan's extractor in Section 3. In section 4, we introduce a RAID systems using Reed-Solomon coding. Section 5 contains all the details of our RAID-like system using extractor coding, and Section 6 shows an example.

2 Preliminaries

2.1 Soft-decision decoding

Here we use the notation $[n, k, d]_q$ to describe a code over F_q. The codeword length is n, the length of original information is k, and the distance of C is d. In coding theory, when receiving a word r, the receiver tries to find the closest codeword. This is the well-known *maximum likelihood decoding (MLD).* If there were less than or equal to $\lfloor (d-1)/2 \rfloor$ errors made, the receiver can correct the received word by MLD. Therefore we try to use another reasonable solution to deal with this bad condition: find all possible codewords within a specified Hamming distance from a received word. This decoding solution is called *list decoding. Reed-Solomon codes* is a well-known code which has the list decoding.

Now we consider more highly noisy channels. Suppose a code C with a large alphabet Σ. If each received symbol may be one of some $\frac{\Sigma}{2}$ symbols with equal probability, it is very hard to find a small set of possible codewords by list decoding. There is a more generalized solution to deal with these problems, called *soft-decision decoding.* The word "soft" means that we take total symbols with probabilities instead of the correct one.

Let C be a code of length T over alphabet $[M] = \{1, 2, ..., M\}$. In soft decoding, we want to describe that the ith received signal induces a probability distribution over all possible symbols in the alphabet $[M]$, for any $i \in [T]$. We model this as a *weight function*: $w : [M] \times [T] \to [0, 1]$. So we can think $w(y, i)$ is the probability that ith received signal becomes $y \in [M]$. Define the *relative weight* of w to be $\frac{\sum_{y\in[M],\ i\in[T]} w(y,i)}{MT}$. It is easy to see that $\rho(w)$ is a real number between 0 and 1.

Definition 2.1. (Agreement) *The agreement $Ag(u, w)$ between a word u and a weight function w is defined as $Ag(u, w) = \sum_{i\in[T]} w(u_i, i)$.*

In maximum likelihood decoding or list decoding, we can simplify a weight function to describe a received word w. Let the weight function have exactly one "1" in each column and the other entries be "0". In this condition, the agreement $Ag(u, w)$ is the number of digits in which u and w are the same. So the decoding process of MLD is to find a unique codeword $u \in C$ that $Ag(u, w) \geq T - \lfloor (d-1)/2 \rfloor$, and list decoding is to find the set of possible codewords $c_1, ..., c_m \in C$ that $Ag(u, w) \geq t \approx d/2$.

Definition 2.2. (Soft decoding) *A code C has*

*Computer Science and Info Engineering Dept, National Chiao-Tung University, Hsinchu 30050, Taiwan, Email: {rjchen, myliu, cydeng, sctsai}@csie.nctu.edu.tw.
The work was supported in part by the National Science Council of Taiwan under contract NSC 91-2213-E-009-102.

(L,δ) soft decoding if for every weight function w the set $\Gamma_\delta(w) = \{u \in C | Ag(u,w) > (\rho(w)+\delta)T\}$ has size at most L.

2.2 Extractors

Definition 2.3. (Statistical difference) *The statistical difference between two random variables X and Y with range $\{0,1\}^n$ is $\|X-Y\| = \max\limits_{T:\{0,1\}^n \to \{0,1\}} |\Pr[T(X)=1] - \Pr[T(Y)=1]|$*

If the distance between two distributions X and Y is less than ε, then we say that X and Y are *ε-close.* A distribution on n bit strings is *ε-uniform* if it is ε-close to U_n.

Definition 2.4. (Extractor) *A function $E : [N] \times [T] \to [M]$ is a (L,δ) extractor if for every subset $X \subseteq [N]$ of cardinality at least L, the distribution $E(X,Y)$ (where Y is uniformly distributed in $[T]$) is δ-uniform.*

Definition 2.5. (Strong Extractor) *A function $E : [N] \times [T] \to [M]$ is a (L,δ) strong extractor if for every subset $X \subseteq [N]$ of cardinality at least L, the distribution $E(X,Y) \circ Y$ (where Y is uniformly distributed in $[T]$) is δ-uniform.*

Definition 2.6. (Extractor codes) *Let $E : [N] \times [T] \to [M]$ be an extractor. For each $x \in [N]$, define a word $u(x) = (u_1, u_2, ..., u_T) \in [M]^T$ where $u_i = E(x,i)$. The extractor code $C_E = \{u(x)| \ x \in [N]\} \subseteq [M]^T$.*

3 Trevisan's extractor

Amnon Ta-Shma and David Zuckerman [8] showed some relations between extractors and codes. They also proved that a good extractor yields a code with good soft decoding and efficient encoding.

Theorem 3.1. ([8]) *If $E : [N] \times [T] \to [M]$ is an (L,δ) strong extractor, then the extractor code C_E has (L,δ) soft decoding. Conversely, if C_E has (L,δ) soft decoding, then E is a $(\frac{L}{\delta}, 2\delta)$ strong extractor.*

They used Trevisan's extractor to construct an explicit extractor code with efficient encoding and decoding. Trevisan's extractor uses a main combinatorial object, called a *design.* It is a collections of sets with small pairwise intersection. Here we introduce an improved version given by Ran Raz, Reinfold, and Vadhan [7]. It is called a *weak design.*

Definition 3.1. (Weak design) *For $\rho \geq 1$, a family of sets $S_1, ..., S_m \subset [t]$ is a (l,ρ) weak design if*

1. *For all i, $|S_i| = l$, and*
2. *For all i, $\sum_{j<i} 2^{|S_i \bigcap S_j|} \leq \rho \cdot (m-1)$*

Besides the weak design, Trevisan's extractor uses a good error correcting code $BC : \{0,1\}^n \to \{0,1\}^{\bar{n}}$, $\bar{n} = \text{poly}(n, \frac{1}{\alpha})$. This code BC need to have the list decoding property: There has at most $\frac{1}{\alpha^2}$ codewords in every ball of radius $(\frac{1}{2}-\alpha)\bar{n}$.

Let $x \in \{0,1\}^n$ be the input choosing from a weak random source, $y \in \{0,1\}^t$ be another input choosing from uniform distribution. Trevisan constructed an extractor $E_{TR}(x,y)$ as follows. First encode the input x using the error correcting code BC, denoted $\hat{x}$. Let $l = \log \bar{n}$. We use a (l,ρ) weak design $S_1, ..., S_m$, for all $S_i \subset [t]$. We define $y|_{S_i} \in \{0,1\}^l$ to denote the string chosen from y corresponding to the set S_i. Then consider $\hat{x}$ as a truth table. It means that $\hat{x}$ can be seen as a function $f_{\hat{x}}$ with the length of input is $\log \bar{n} = l$. For all $i \in [m]$, we use each $y|_{S_i}$ as the input of the function $\hat{x}$. Therefore we can get m output bits finally. Trevisan's extractor is generally defined in the following way:

$$E_{TR}(x,y) = f_{\hat{x}}(y|_{S_1}) \circ f_{\hat{x}}(y|_{S_2}) \circ ... \circ f_{\hat{x}}(y|_{S_m})$$

The extractor codes based on Trevisan's extractor has the following property.

Theorem 3.2. ([8]) *For every $\delta = \delta(n) > 0$ and $M = M(n)$, Let $\{C_{TR,n}\}$ denote a Trevisan's extractor code with $C_{TR} \subseteq [M]^T$. The code has efficient encoding and efficient (L,δ) soft decoding, and $|C_{TR}| = 2^n$, $T = 2^{O((\log n + \log m + \log(\frac{1}{\delta}))^2)}$, $L \leq p(T, M, \frac{1}{\delta})$ for some polynomial $p(\cdot)$.*

4 Reed-Solomon codes in RAID

The concept of *Redundant Array of Inexpensive Disks (RAID)* was first introduced by Paterson, Gibson, and Katz [6]. The RAID technique is using small and inexpensive disk arrays instead of big and expensive simple disk. Using disk arrays, we can simply spread data over multiple disks, called *striping.* It not only increases the available storage but also increases the effective I/O bandwidth by allowing multiple disk accesses to data striped on different disks at same times. However, disk arrays are prone to failure because the failure rate increase linearly with the number of disks in an array compared with single large disk. We can avoid this condition by adding redundancy information in the disk array. In the way, disk array will also have more reliability.

4.1 Reed-Solomon Codes

Reed-Solomon codes, abbreviated *RS codes*, are one of the most practical and well-known codes. These codes can be used as both error-correcting and *erasure* codes.

Definition 4.1. (Reed-Solomon codes) *Let γ be a fixed primitive element of F_q and let k be an integer with $0 \leq k \leq n = q-1$. We let $\mathcal{P}_k$ denote the set of polynomials of degree less than k, including the zero polynomial. Then*

$$C = \{\left(f(1), f(\gamma), f(\gamma^2), ..., f(\gamma^{q-2})\right) | f \in \mathcal{P}_k\}$$

is an $[n, k, n-k+1]$ Reed-Solomon code over F_q.

The Definition is the narrow sense Reed-Solomon codes. It can be generalized that n can achieve q by adding the first digit of the original information to the codeword. The well-known decoding algorithm of Reed-Solomon codes is Berlekamp-Welch decoding algorithm in [11].

4.2 Reed-Solomon coding in RAID-like systems

We now use the Reed-Solomon coding in RAID-like systems. First we define our storage specification.

Let there be k main storage devices, $D_1, D_2, ..., D_k$. There are called the *Data devices*. Let there be m additional storage devices, $C_1, C_2, ..., C_m$. There are called the *Checksum devices*. Each data device stores the original data, and the contents of each checksum device will be calculated from data devices. This RAID-like system, called *RS-RAID*, can reconstruct the contents from the non-failed devices if at most any m of $D_1, D_2, ..., D_k, C_1, C_2, ..., C_m$ fail.

We define our failure model is that of an erasure. When a device fails, it shut down, and the system recognizes this shutting down. It is reasonable because all operating systems today can detect whether the hard drives work properly or not.

The RS-RAID method breaks up each storage device into *words*. Let the size of each word is l bits, l being chosen by the management with some constraints that we will discuss later. So one block of the storage devices contains

$$(512 \text{ bytes})\left(\frac{8 \text{ bits}}{\text{byte}}\right)\left(\frac{1 \text{ word}}{l \text{ bits}}\right) = \frac{4k}{l} \text{ words.}$$

If a storage device contains w blocks, it can store $\frac{4kw}{l}$ words. The calculation of the contents of each checksum device C_i requires a function F_i applied to all the data devices. This function F_i operates on a word-by-word basis, as in Figure 1, where $d_{i,j}$ (or $c_{i,j}$) represents the jth word of device D_i (or C_i).

To make the notation simpler, we now assume that each device holds just one word and drop the extra subscript. Thus there are consisting of k data words $d_k, ..., d_k$ and m checksum words $c_1, ..., c_m$ which are computed from the data words in such a way that the loss of any m words can be tolerated. Now we define the RS-RAID algorithm. There are three main parts of this algorithm.

4.2.1 Calculating checksum words

We consider each word as one digit of a codeword of the Reed-Solomon code. The codeword length is $n = k + m$, and the operations are over $GF(2^w)$. So we need the constraint that $k + m < 2^l$. Let γ be a fixed primitive element of $GF(2^w)$. Suppose the data that we want to store are $w = (w_1, ..., w_k)$, and all $w_i \in GF(2^w)$. We can calculate the contents of each device by using the encoding matrix.

$$\begin{bmatrix} 1 & 1 & .. & 1 \\ 1 & \gamma & .. & \gamma^{k-1} \\ \vdots & \vdots & & \vdots \\ 1 & \gamma^{k+m-1} & .. & \gamma^{(k+m-1)(k-1)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ d_k \\ c_1 \\ \vdots \\ c_m \end{bmatrix}$$

Using the Reed-Solomon codes, all data devices do not store the original data w. Even if reading the data, we need to decode the codeword. We can overcome this drawback by simplifying the encoding matrix. We modify the matrix to let $d_i = w_i$ for $1 \le i \le k$ and every subset of k rows of the matrix are still linearly independent.

$$\begin{bmatrix} 1 & 0 & ... & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & ... & 1 \\ 1 & 1 & ... & 1 \\ \vdots & \vdots & & \vdots \\ 1 & \gamma^{m-1} & ... & \gamma^{(m-1)(k-1)} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_k \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ d_k \\ c_1 \\ \vdots \\ c_m \end{bmatrix}$$

The checksum word c_i is calculated by the function F_i which is a linear combination of the data words.

$$c_i = F_i(d_1, d_2, ..., d_k) = \sum_{j=0}^{k-1} \gamma^{(i-1)j} d_i$$

Define the function F_i as rows of the calculating matrix F. We can see that F is an $m \times k$ Vandermonde matrix. Without loss of generality, we can define $\gamma^i = i + 1$ for $0 \le i \le n - 1$. So we get the calculating matrix F as follows.

$$\begin{bmatrix} 1 & 1 & ... & 1 \\ \vdots & \vdots & & \vdots \\ 1 & m & ... & m^{k-1} \end{bmatrix} \begin{bmatrix} d_1 \\ \vdots \\ d_k \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_m \end{bmatrix}$$

4.2.2 Maintaining checksum words

When one of the data words d_j changes to d'_j, then we need to update each of the checksum words. The general method is using the calculating matrix F to recalculate all checksum words. But this way wastes much time on the calculation of unchanged data words. Let the updating function $U_{i,j}$ for ith checksum word and jth changed data word. We can use the addition property of the matrix to simplify the maintaining process.

$$\begin{aligned} c'_i = U_{i,j}(c_i, d_j, d'_j) &= c_i + F_{i,j}(d'_j - d_j) \\ &= c_i + i^{j-1}(d'_j - d_j) \end{aligned}$$

4.2.3 Recovering from failures

If there are at most m broken disk, the entire RS-RAID system can be reconstructed. We now explain how to recover from errors. Review the matrix A which we use to encode. $A = \begin{bmatrix} I \\ F \end{bmatrix}$. It composes a $k \times k$ identity matrix I and a $m \times k$ calculating matrix F. Define a vector E to represent all devices. $E = \begin{bmatrix} D \\ C \end{bmatrix}$. D is a $k \times 1$ matrix containing the data words $d_1, ..., d_k$, and C is a $m \times 1$ matrix containing the checksum words $c_1, ..., c_m$. We can see the equation $AD = E$ in above works, and each device in the system as having a corresponding row of the matrix A and the vector E.

When a device fails, we reflect the failure by deleting the device's row from A and from E. What results a new matrix A' and a new vector E' that adhere to the equation: $A'D = E'$. Suppose exactly m devices fail. Then A' is a $k \times k$ matrix. Because F is a Vandermonde matrix and I is an identity matrix, every subset of k rows of matrix A is guaranteed to be linearly independent. Thus,

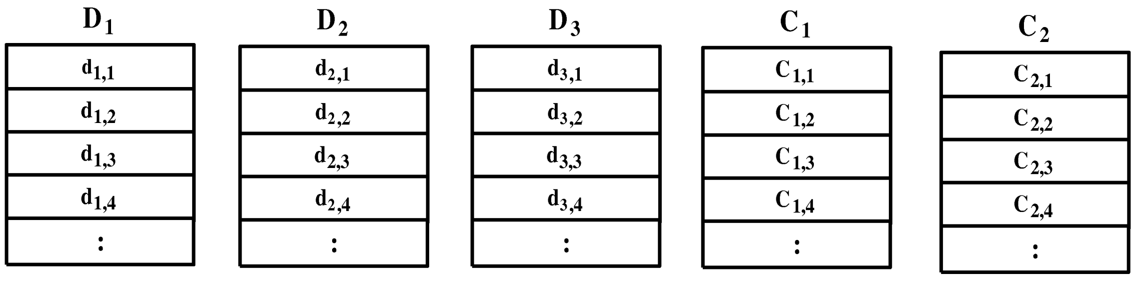
$\mathbf{D_1}$
$\mathbf{D_2}$
$\mathbf{D_3}$
$\mathbf{C_1}$
$\mathbf{C_2}$
$d_{1,1}$ $d_{1,2}$ $d_{1,3}$ $d_{1,4}$:
$d_{2,1}$ $d_{2,2}$ $d_{2,3}$ $d_{2,4}$:
$d_{3,1}$ $d_{3,2}$ $d_{3,3}$ $d_{3,4}$:
$C_{1,1}$ $C_{1,2}$ $C_{1,3}$ $C_{1,4}$:
$C_{2,1}$ $C_{2,2}$ $C_{2,3}$ $C_{2,4}$:

Figure 1: Dividing the storage devices into words. $(k = 3, m = 2)$

A' is non-singular and there exists an inverse matrix A'^{-1} that $A'^{-1}A' = I$. We can get D by calculating from $A'^{-1}A'D = D = A'^{-1}E'$ or using Gaussian elimination. Hence all data devices can be recovered.

If there are less than m devices that fail, we choose k devices of the unbroken devices arbitrarily. The system can also be recovered in the same manner. Thus, the RS-RAID can tolerate any number of device failures up to m.

5 Extractor codes in RAID

The extractor codes based on Trevisan's extractor consequently need an explicit code with list decoding property. We now use a simple idea "concatenation" which puts two codes together and obtains a code. Here we use be an $[n, k, n-k+1]_n$ Reed-Solomon code as the outer code, denoted RS, and an $[n, \log n, n/2]_2$ Hadamard code as the inner code, denoted Had_n. The concatenated code $RS \circ \mathrm{Had}_n$ is an $[n^2, k\log n, \frac{n(n-k+1)}{2}]_2$ code. We denote this concatenated code C_{RS-Had}.

Lemma 5.1. *Let C_{RS-Had} be the concatenated code as above. Then C_{RS-Had} has list decoding property $\sqrt{\frac{k}{2n} - \frac{1}{2n}}$.*

Proof. Let $\epsilon = \frac{k}{2n} - \frac{1}{2n}$. We want to prove the list decoding property $\delta = \sqrt{\epsilon}$. Note that the distance of C_{RS-Had} is $\frac{n(n-k+1)}{2} = (\frac{1}{2} - \frac{k}{2n} + \frac{1}{2n})n^2 = (\frac{1}{2} - \epsilon)n^2$. Since $\delta = \sqrt{\epsilon} > \frac{1}{2}\sqrt{2\epsilon}$, By the Johnson bound from [8], this code has $(\frac{2}{4\delta^2 - 2\epsilon} = \frac{1}{\epsilon}, \delta)$ soft decoding. That means every Hamming ball of relative radius $(\frac{1}{2} - \sqrt{\epsilon})$ has at most $\frac{1}{\epsilon}$ codeword. □

The concatenated code with an outer Reed-Solomon code and an inner Hadamard code also can be efficiently list decoded. So we can take advantage of the property of C_{RS-Had} to use in the construction of Trevisan's extractor. Another important thing of Trevisan's extractor is a (l, ρ) weak design. It was proved in [7] that there exists a (l, ρ) weak design $S_1, ..., S_m \subset [t]$ with $t = \left\lceil \frac{l}{\ln \rho} \right\rceil \cdot l$, for $\rho > 1$. Let $\rho = 2$, we choose t at least $2l^2$.

It is trivial that we can construct a weak design with $t = ml$. The subsets $S_1, ..., S_m$ are disjointed. Here we introduce a simple construction of a $(l, 2)$ weak design with $t = \frac{ml}{4}$, if $m = 4cl$, for an integer $c \geq 2$. The construction uses four different methods to partition t into $\frac{m}{4} = cl$ disjoint subsets of size l. Define $S_i^{(j)}$ be the ith subset constructed by the jth method.

First , we construct the subsets by choosing elements in sequence. It means that $S_1^{(1)} = \{1, 2, ..., l\}$, $S_2^{(1)} = \{l+1, l+2, ..., 2l\}$, ..., $S_{cl}^{(1)} = \{t-l+1, t-l+2, ..., t\}$.

Second, we separate all subsets $S_i^{(1)}$ into c groups, so every group have l subset. For each group, we can construct subsets by choosing elements that have the same value modulo l. So the first group contains l subsets: $S_1^{(2)} = \{1, l+1, ..., l^2-l+1\}$, $S_2^{(2)} = \{2, l+2, ..., l^2-l+2\}$, ..., $S_l^{(2)} = \{l, 2l, ..., l^2\}$, and the second group contains l subsets: $S_{l+1}^{(2)} = \{l^2+1, l^2+l+1, ..., 2l^2-l+1\}$, ..., $S_{2l}^{(2)} = \{l^2+l, l^2+2l, ..., 2l^2\}$. There are c groups, so we have another $cl = \frac{m}{4}$ subsets.

Third, let $S_i^{(3)}$ contain the kth element in $S_{i+k-1}^{(2)}$ that $1 \leq k \leq l$. It means that we pick up elements with an interval $l+1$. There are cl subsets $S_i^{(3)}$. $S_1^{(3)} = \{1, l+2, ..., l^2\}, S_2^{(3)} = \{2, l+3, ..., l^2+1\}$, ..., $S_{cl}^{(3)} = \{t-l+1, l+1, ..., l^2-1\}$. For convenience, if $i > cl$, we define $S_i^{(2)} = S_{i-cl}^{(2)}$ recursively.

Fourth, let $S_i^{(4)}$ contain the kth element in $S_{i-k+l}^{(2)}$ that $1 \leq k \leq l$. It means that we pick up elements with an interval $l-1$. There are cl subsets $S_i^{(4)}$. $S_1^{(4)} = \{l, 2l-1, ..., l^2-l+1\}, S_2^{(4)} = \{l+1, 2l, ..., l^2-l\}$, ..., $S_{cl}^{(4)} = \{l-1, 2l-2, ..., t\}$. We finally collect all subsets $S_i^{(k)}$, for $1 \leq i \leq cl$ and $1 \leq k \leq 4$. There are exactly $4cl = m$ subsets.

Lemma 5.2. *The collection of subsets $S_i^{(k)}$, where $1 \leq i \leq cl$ and $1 \leq k \leq 4$, is a $(l, 2)$ weak design with $t = \frac{ml}{4}$.*

Proof. It is easy to see that for any two different subsets $S_i^{(k)}$ and $S_j^{(k')}$:

- if $k = k'$, then $\left| S_i^{(k)} \bigcap S_j^{(k')} \right| = 0$.
- if $k \neq k'$, then $\left| S_i^{(k)} \bigcap S_j^{(k')} \right| \leq 1$.

Notice that $\sum_{j<i} 2^{|S_i \bigcap S_j|} = 3l \cdot 2^1 < 2(m-1)$, for all i. Therefore, this simple construction is a $(l, 2)$ weak design with $t = \frac{ml}{4}$. □

5.1 Calculating algorithm

The classification of storage devices is the same with the RS-RAID system. Let there be x data devices, $D_1, ..., D_x$ and y checksum devices $C_1, ..., C_y$. The RAID-like system, called *EC-RAID*, also breaks up each storage device into words of size m bits. Let w, k be two positive integers, $n = 2^w$ and $k \leq n$. We divide the original information needed to store into *units* of size $k \log n$ bits. For every unit, we encode it with an extractor code and store each digit of the codeword in a corresponding checksum device. Besides, we separate each unit into the data devices that contain m bits individually.

Suppose there is a unit $u = u_1, u_2, ..., u_{k \log n}$, where every $u_i \in \{0, 1\}$, $1 \leq i \leq k \log n$. First, we encode the unit u with a concatenated code, C_{RS-Had}. We store full original information of the unit to the data words $d_1, ...d_x$, so the number of data devices is $x = \frac{k \log n}{m}$.

Define the codeword $C_{RS-Had}(u) = \hat{u}$. We treat this codeword as the outputs of hard functions, denoted $f_{\hat{u}}$. The length of codeword $\hat{u}$ is n^2, so the size of the truth table is n^2. We need that each subset in a $(l = \log(n^2) = 2w, 2)$ weak design has size Let $z \in \{0, 1\}^t$, run Trevisan's extractor $E_{TR}(u, z) = f_{\hat{u}}(z|_{S_1}) \circ f_{\hat{u}}(z|_{S_2}) \circ ... \circ f_{\hat{u}}(z|_{S_m})$. We consider the output of $E_{TR}(u, z)$ which is just m bits a checksum word, and store it in the checksum device C_{z+1}. So the number of checksum devices is $y = 2^t$. In fact, we store each symbol of the codeword $C_{TR}(u)$ in the checksum devices.

Calculating Algorithm

Parameters : k, w, m are positive integers, with:

- $k \leq 2^w = n$, $m | kw$, and $t \geq \min\{8w^2, 2mw\}$.

Storage devices :

- Data devices $D_1, D_2, ..., D_{\frac{kw}{m}}$.
- Checksum devices $C_1, C_2, ..., C_{2^t}$.

Prepare : A fixed $(l = 2w, 2)$ weak design $S_1, ..., S_m \subseteq [t]$.
Input : A unit $u \in \{0, 1\}^{kw}$.
Procedure :

1. Divide u into $\frac{kw}{m}$ words of size m bits, store ith word in the data devices D_i.
2. Let C_{RS-HAD} be a $[2^{2w}, kw, (\frac{1}{2} - \epsilon)2^{2w}]_2$ code with $\epsilon = \frac{k-1}{2^{w+1}}$, and calculate $\hat{u} = C_{RS-HAD}(u)$.
3. For every $z \in \{0, 1\}^t$, and $i \in [m]$, calculate $f_{\hat{u}}(z|_{S_i})$ and store it to ith bit in the $(z+1)$th checksum devices.

The maintaining algorithm of EC-RAID is the same as the calculating algorithm. We have to encode new information u again with the concatenated code C_{RS-HAD} to get $\hat{u}$.

5.2 Unique recovering algorithm

Here we suppose that all of the data devices and some of the checksum devices are broken. The way to recover the original data is equal to the method to decode extractor codes.

The first step is that finding the contents of codeword $\hat{u}$ encoded by C_{RS-Had}. Consider the checksum devices as an oracle. For $\omega \in \{0, 1\}^l$, if there exists at least one subset S_i and $z \in \{0, 1\}^t$ such that $z|_{S_i} = \omega$ and $(z + 1)$th checksum device does not fail, we can know what the value $f_{\hat{u}}(\omega)$ is. Giving different ω exactly 2^l times, we can recover the fully codeword $\hat{u}$.

It is easy to see that the bit, which equals to $f_{\hat{u}}(\omega)$, is stored 2^{t-l} times in the ith location of each checksum device. Therefore if there exists at least one unbroken checksum device that contains $f_{\hat{u}}(\omega)$ in the ith location for all ω, we can only read the ith location of each checksum word and restore the codeword $\hat{u}$.

We can build an *inverted table* to find S_i with the mentioned property efficiently. Let F_{inv} be the inverted table that

$$F_{inv}(\omega, i) = \{z_1, z_2, ..., z_{2^{t-l}}\}.$$

For every $j \in [2^{t-l}]$, we have $z_j \in \{0, 1\}^t$ and $z_j|_{S_i} = \omega$. From the contents of the weak design, we can constructed this table in time poly(m, T). It needs space of size $t \cdot 2^{t-l} \cdot l \cdot m$ bits. Given the inverted table, for any S_i, we can check whether S_i satisfies the property or not in time $O(t \cdot 2^{t-l}) = O(T \log T)$. After getting the codeword $\hat{u} \in \{0, 1\}^{n^2}$, we divide this string into n partitions.

The second step is to decode the Hadamard code. We can use the brute force to compare all codewords and find the correct one. So there are total n strings of size $\log n$ bits we get.

The third step is to decode the Reed-Solomon code by using the Berlekamp-Welch decoding algorithm or calculating equations with Gaussian elimination. Finally, we can get the original data u.

Unique Recovering Algorithm UR-I

Prepare : An inverted table F_{inv}
Input : A codeword $C_{TR}(u) \in [M]^T$ with some erased symbols.
Procedure :

1. Find a subset S_i with the property: There exists at least one $z = z(\omega)$ that $z(\omega) \in F_{inv}(\omega, i)$ and the $(z(\omega) + 1)$th symbol is not erased, for each $\omega \in \{0, 1\}^l$.
2. Construct the codeword $\hat{u} \in \{0, 1\}^{2l}$ by
$$\hat{u}_{\omega+1} = \left(C_{TR}(u)_{z(\omega)+1}\right)_i$$
3. Decode $\hat{u}$ by first decoding the Hadamard code with brute force and second decoding the Reed-Solomon code with Berlekamp-Welch algorithm.

Output : A corresponding $u \in \{0, 1\}^{kw}$.

The algorithm UR-I works if and only that we can find a subset with the above property. If there is no subset with this property, it is still possible to uniquely recover the original data. We improve this point and describe another algorithm as follows.

```
Unique Recovering Algorithm UR-II
Prepare : An inverted table $F_{inv}$
Input : A codeword $C_{TR}(u) \in [M]^T$ with unknown $e$ locations.
Procedure :
  1. Define $u' \in \{0,1,-1\}^{2^l}$ and initialize $u'_j = -1$ for all $1 \le j \le 2^l$.
  2. For every $\omega \in \{0,1\}^l$, $1 \le i \le m$, $z \in F(\omega, i)$,
     let $u'_{\omega+1} = (C_{TR}(u)_{z+1})_i$ if $C_{TR}(u)_{z+1}$ is not erased.
  3. Construct the codeword $\hat{u} \in \{0,1\}^{2^l}$ by
     $\hat{u}_k = \begin{cases} u'_k & \text{if } u'_k \ne -1 \\ \text{unknown} & \text{if } u'_k = -1 \end{cases}$
  4. Decode $\hat{u}$ by first decoding the Hadamard code and second decoding the Reed-Solomon code.
Output : A corresponding $u \in \{0,1\}^{kw}$.
```

The following theorem shows that the unique recovering algorithm UR-II is correct and efficient.

Theorem 5.3. *Let $\epsilon = \frac{k-1}{2^{w+1}}$ and $\eta = \lfloor \frac{t}{l} \rfloor$. If the number of unknown locations e is less than $(1 - (\frac{1}{2} + \epsilon)^\eta)2^t$, we can find the correct u uniquely in time poly(n, m, T) with the algorithm UR-II.*

Proof. The distance of C_{RS-Had} is $(\frac{1}{2} - \epsilon)2^{2w}$. Suppose that the last $(\frac{1}{2} - \epsilon)2^{2w}$ bits are different. So if there is a subset chooses one bit in this region, the corresponding symbols of $C_{TR}(u)$ and $C_{TR}(v)$ are different. Without loss of generality, assume there are $\eta = \lfloor \frac{t}{l} \rfloor$ disjoint subset $S_1, ..., S_\eta$ in the weak design. Consider S_1 and all locations of $C_{TR}(u)$ and $C_{TR}(v)$, there are $(\frac{1}{2} - \epsilon)$ fraction are different. Consider S_2 and the partial locations which are the same corresponding to S_1, there are still $(\frac{1}{2} - \epsilon)$ fraction are different because S_1 and S_2 are disjoint. In the similar way, we have that the identical fraction are at most $(\frac{1}{2} + \epsilon)^\eta$. So the relative distance of C_{TR} is at least $(1 - (\frac{1}{2} + \epsilon)^\eta)$. We need time $O(2^l)$ for the step 1 and 3 in UR-II, $O(l \cdot m \cdot t \cdot 2^{t-l})$ for step 2, $O(n^2 + n \log n)$ for step 4. Therefore the algorithm runs in time poly(n, m, T). □

This theorem also shows the recovering ability of EC-RAID system. It can tolerate up to arbitrary broken $(1 - (\frac{1}{2} + \epsilon)^\eta)2^t$ checksum devices. If there are more checksum devices failed, it is obvious that we may not get original data uniquely. Next we present the second algorithm to deal with this terrible situation.

5.3 Soft recovering algorithm

We have to run the identical step 1 and step 2 in the algorithm UR-II before starting the second algorithm. Then we decode the Hadamard code and construct a weight function $w : [n] \times [n] \to [0,1]$ to describe the probability of each digit in a Reed-Solomon code. $w(y, i)$ means the probability that ith digit equals to some $y \in [n]$. The way to construct the weight function is the same to decode the Hadamard code with brute force.

Now consider how to soft-decision decode Reed-Solomon codes. A trivial solution is to calculate the agreements of all codewords and pick up those with the highest value. But the time is exponential to the codeword length. Here we use the algorithm presented by in [3].

We describe some essential elements in this algorithm.

Definition 5.1. (weighted degree) *For non-negative weights w_1, w_2, the (w_1, w_2)-weighted degree of the monomial $x^i y^j$ is defined to be $iw_1 + jw_2$. For a bivariate polynomial $\mathcal{P}(x, y)$, the (w_1, w_2)-weighted degree of $\mathcal{P}$, denoted $deg_{w_1,w_2}(\mathcal{P})$, is the maximum over all monomials with non-zero coefficients in $\mathcal{P}$ of the (w_1, w_2)-weighted degree of the monomial.*

Let $N_{w_1,w_2}(\delta)$ be the number of monomials in $\mathcal{P}$ that $deg_{w_1,w_2}(\mathcal{P}) \le \delta$.

Lemma 5.4. ([3]) $N_{1,k}(\delta) = \lceil \frac{\delta+1}{k} \rceil (\delta - \frac{k}{2} \lfloor \frac{\delta}{k} \rfloor + 1) \ge \frac{\delta}{k}(\frac{\delta+2}{2})$.

Given a point (α, β) and a bivariate polynomial $\mathcal{P}(x, y)$, we know that if (α, β) is one root of $\mathcal{P}(x, y)$, then $\mathcal{P}(\alpha, \beta) = 0$. If this point (α, β) is a root with multiplicity m, all monomials in the *shifted polynomial* $\mathcal{P}(x+\alpha, y+\beta)$ have degree at least m. In other words, the coefficients of the monomials in the shifted polynomial of degree less than m are all zero. Let a_{ij} be a coefficient of monomial $x_i y_j$ in $\mathcal{P}(x, y)$, and b_{kl} be a coefficient of monomial $x_k y_l$ in $\mathcal{P}(x + \alpha, y + \beta)$. From the well-known explicit relation of coefficients between the bivariate polynomial and the shifted polynomial, we have

$$b_{kl} = \sum_{i \ge k} \sum_{j \ge l} \binom{i}{k} \binom{j}{l} \alpha^{i-k} \beta^{j-l} a_{ij}$$

So $b_{kl} = 0$ if $k + l < m$.

Thus $\mathcal{P}(x, y)$ passes through a given point with multiplicity at least m if and only if its coefficients a_{ij} satisfy that $\frac{1}{2}m(m+1)$ constraints in the shifted polynomial are zero. The first step of Koetter-Vardy algorithm is to find a bivariate polynomial $\mathcal{Q}(x, y)$ over $GF(n)$ of $(1, k-1)$-weight degree at most $n - 1$. The bivariate polynomial $\mathcal{Q}(x, y)$ must pass through all given pairs with different multiplicity. We can think that the multiplicity indicates the importance. For example, let $y_1, y_2 \in [n]$ and $w(y_1, i) > w(y_2, i) \ne 0$ in some weight function, it means that the probability of ith digit equals to y_1 is more than the ith digit equals to y_2. So it is sensible to let y_1 have more multiplicity than y_2. We describe how to change a weight function w to a $n \times n$ multiplicity matrix $\mathcal{M}$.

```
Constructing Multiplicity Matrix Algorithm
Input : A weight function $w : [n] \times [n] \to [0,1]$ and a positive integer $s$.
Procedure :
  1. Initialize $m_{ij} = 0$ for every $0 \le i \le n-1$, $1 \le j \le n$.
  2. Let $\bar{W}$ be an $n \times n$ matrix and $\bar{w}_{ij} = w(i,j)$.
  3. Find the position $(i,j)$ of the largest entry $\bar{w}_{ij}$ in $\bar{W}$, and set $\bar{w}_{ij} = \frac{w(i,j)}{m_{ij}+2}$, $m_{ij} = m_{ij} + 1$ and $s = s - m_{ij}$.
  4. If $s > 0$, go to the step 3.
  5. If $s < 0$, $m_{ij} = m_{ij} - 1$.
Output : The multiplicity matrix $\mathcal{M}$.
```

For convenience, we consider the $[n, k, n - k + 1]_n$ Reed-Solomon code defined as $C_{RS} =$

$\{(f(1), f(2), f(3), ..., f(0))\,|f \in \mathcal{P}_k\}$. Therefore the input pair is (j, i) with multiplicity m_{ij}, for every i, j such that $m_{ij} \neq 0$.

Definition 5.2. (multiplicity matrix cost) *Given a $n \times n$ multiplicity matrix $\mathcal{M}$, we define the cost of $\mathcal{M}$ by $C(\mathcal{M}) = \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=1}^{n} m_{ij}(m_{ij}+1)$.*

The cost $C(\mathcal{M})$ equals to the number of restricted coefficients in shifted polynomial. So we list $C(\mathcal{M})$ linear equations to solve unknown coefficients of $\mathcal{Q}(x,y)$. We can always find a solution if $N_{1,k-1}(n-1) > C(\mathcal{M})$. Let a positive integer s in the multiplicity algorithm be $N_{1,k-1}(n-1)-1$. So $C(\mathcal{M})$ is certainly less than $N_{1,k-1}(n-1)$.

The second step of Koetter-Vardy algorithm is to factor the bivariate polynomial $\mathcal{Q}(x,y)$ and identify all the factors of $\mathcal{Q}(x,y)$ of type $y-p(x)$, where $p(x)$ has degree at most $k-1$. Then it outputs a list of codewords that correspond to these factors.

```
Soft Recovering Algorithm SR
Input : A word $u' \in \{0,1,-1\}^{n^2}$.
Procedure :
    1. For every $1 \le i \le n$, find the set $h_i$ that contains all symbols $c \in \{0,1\}^{\log n}$ that $Had(c)_j = (u'_i)_j$ for every $(u'_i)_j \neq -1$.
    2. Define a weight function $w : [n] \times [n] \to [0,1]$ by: If $y \in h_i$, $w(y,i) = |h_i|^{-1}$, else $w(y,i) = 0$.
    3. Construct a multiplicity matrix $\mathcal{M}$ with $w$ and $s = N_{1,k-1}(n-1)-1$.
    4. Compute a bivariate polynomial $\mathcal{Q}(x,y)$ over $GF(n)$, $deg_{1,k-1}(\mathcal{Q}) \le n-1$, that has a zero of multiplicity at least $m_{ij}$ in $\mathcal{M}$ at point $(j,i)$ for every $i,j$ that $m_{ij} \neq 0$.
    5. Find all polynomials $p(x)$ over $GF(n)$, $deg(p) \le k-1$, that $y-p(x)$ is a factor of $\mathcal{Q}(x,y)$.
    6. For all $p(x)$, let $u(p)$ be the corresponding codeword, calculate the agreement $\sum_{i=1}^{n} w(u(p)_i, i)$. Add $u(p)$ that has maximum agreement to the output list.
Output : A list of $u = u(p) \in \{0,1\}^{kw}$.
```

It is easy to see that the step $1 \sim 3$ take time $O(n^2)$. Solving the linear equations and factoring a polynomial can be accomplished in time poly(n) by some efficient algorithms, given in [2] and [12]. We now show the correctness of the soft recovering algorithm.

Lemma 5.5. ([1]) *Suppose that a bivariate polynomial $\mathcal{Q}(x,y)$ over $GF(n)$ passes through a point (α,β) with multiplicity at least m, and let $p(x)$ be any polynomial over $GF(n)$ such that $p(\alpha)=\beta$. Then the polynomial $\mathcal{Q}(x,p(x))$ is divisible by $(x-\alpha)^m$.*

For a word v, we define the score of v with a given multiplicity matrix $\mathcal{M}$ as $\mathcal{S}_{\mathcal{M}}(v) = Ag(v,\mathcal{M})$.

Theorem 5.6. *Let $C = C(\mathcal{M})$ be the cost of a given multiplicity matrix $\mathcal{M}$. The polynomial $\mathcal{Q}(x,y)$ has a factor $y-p(x)$, where $p(x)$ evaluates to a codeword $c \in C_{RS}$, if $\mathcal{S}_{\mathcal{M}}(c) > \sqrt{2(k-1)C+1}$.*

Proof. Let $c = (c_1, ..., c_n)$ be a codeword $\in C_{RS}$, and let $p(x)$ be the polynomial that evaluates to c. That means $g(i) = c_i$ for $1 \le i \le n$. Given $\mathcal{Q}(x,y)$, we define the polynomial $g(x) = \mathcal{Q}(x,p(x))$. By Lemma 5.5, we have the fact that the polynomial $g(x)$ is divisible by the product

$$(x-1)^{m_{c_1,1}}(x-2)^{m_{c_2,2}}...(x-n)^{m_{c_n,n}}$$

where a point (i,c_i) with multiplicity at least $m_{c_i,i}$ in the matrix $\mathcal{M}$. It is clear that either $deg(g) \ge m_{c_1,1}+m_{c_2,2}+...+m_{c_n,n} = \mathcal{S}_{\mathcal{M}}(c)$ or $g(x)$ is an all-zero polynomial. We need to show that $deg(g) < \mathcal{S}_{\mathcal{M}}(c)$, so $g(x) = \mathcal{Q}(x,p(x)) \equiv 0$. Therefore $\mathcal{Q}(x,y)$ has a factor $y-p(x)$.

We define the function $\Delta(C)$ to represent the minimum degree such that the number of monomials are exactly more than ν, so

$$\Delta_{w_1,w_2}(\nu) = \min\{\delta \in Z | N_{w_1,w_2}(\delta) > \nu\}$$

Here we simplify the notation, let Δ be $\Delta_{1,k-1}(C)$. Notice that $N_{1,k-1}(\Delta) > C \ge N_{1,k-1}(\Delta-1)$. By Lemma 5.4, we have $N_{1,k-1}(\Delta-1) \ge \frac{\Delta^2-1}{2(k-1)}$. Then $\sqrt{2(k-1)C+1} \ge \Delta$.

Since $deg(p) \le k-1$, it is easy to see that degree of $g(x)$ can not exceed the $(1,k-1)$-weighted degree of $\mathcal{Q}(x,y)$. Thus we have $deg_{1,k-1}\mathcal{Q}(x,y) \le \Delta \le \sqrt{2(k-1)C+1} < \mathcal{S}_{\mathcal{M}}(c)$. □

6 An example

As an example, suppose $k = 2$ and $m = 2$. We choose w to be two, since $n = 2^w > k$. It means that the C_{RS} we use is a $[4,2,3]_4$ Reed-Solomon code and Had$_4$ is a $[4,2,2]_2$ Hadamard code. Consider the trivial case $t = ml = 2mw = 8$, then the $(4,2)$ weak design contains two subsets: $S_1 : \{1,2,3,4\}$ and $S_2 : \{5,6,7,8\}$. Here we need $\frac{kw}{m} = 2$ data devices, denoted D_1, D_2, and $T = 2^t = 256$ checksum devices, denoted $C_1 ..., C_{256}$.[1]

Define the operations $+$ and $\cdot$ over $GF(4)$. We use "0" to denote 00, "1" denote 01, "2" denote 10, and "3" denote 11, so we have

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	0	1

·	0	1	2	3
0	0	0	0	0
1	0	2	3	1
2	0	3	0	1
3	0	3	1	2

Review data stored in each device by the word of size $m = 2$ bits. Now consider we have to store data $u = (u_1, u_2) = (2,3) = 1011$. First, divide u into words and place in data devices: $D_1 = 10, D_2 = 11$. Next, we encode u with the Reed-Solomon code. The polynomial $g(x)$ that u indicate is $2+3x$, so we have

$$C_{RS}(u) = g(1) \circ g(2) \circ g(3) \circ g(0) = 1302$$

Then we encode each digit of $C_{RS}(u)$ with the Hadamard matrix

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

[1]In this example, we want to explain how a design works, so there are 256 checksum devices needed. Actually, we can choose $m = 1$, then there are only $T = 2^{1\cdot 4} = 16$ checksum devices, and $\frac{kw}{m} = \frac{2\cdot 2}{1} = 4$ data devices.

So the codeword $\hat{u} = C_{RS-Had}(u)$ to represent a truth table of size $l = 4$.

0000	0	0100	0	1000	0	1100	0
0001	1	0101	1	1001	0	1101	0
0010	0	0110	1	1010	0	1110	1
0011	1	0111	0	1011	0	1111	1

We store $f_{\hat{u}}(z|_{S_1}) \circ f_{\hat{u}}(z|_{S_2})$ in the checksum devices C_{z+1} for every $z \in \{0,1\}^8$. For example, $C_{47} = f_{\hat{u}}(00101110|_{1234}) \circ f_{\hat{u}}(00101110|_{5678}) = f_{\hat{u}}(0010) \circ f_{\hat{u}}(1110) = 01$.

Now consider data devices D_1 and D_2 are lost. Then we use the unique recovering algorithm to save original data from the checksum devices. In the first time, we have to build an inverted table F_inv for searching needed bits quickly. Recall that $F_{inv}(w,i)$ contains a set of strings which equal to w by choosing bits according to S_i. For example, $F_{inv}(1100,1) = \{11000000, 11000001, ..., 11001110, 11001111\}$. Therefore, if we want to know the 13th bit of codeword $\hat{u}$, we can read the first corresponding bit in any one of the checksum devices $C_{193} \sim C_{208}$.

Suppose $C_1, C_2, ..., C_{16}$ still works, we get the contents 00, 01, 00, 01, 00, 01, 01, 00, 00, 00, 00, 00, 00, 00, 11, 11. Because all of the corresponding second bit in checksum devices $C_1, ..., C_{16}$ exactly go through the entries in the truth table. So we have $\hat{u} = 0101011000000011$.

Decode every four bits in $\hat{u}$ by the Hadamard code, we have $C_{RS}(u) = 1302$. That means

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 0 \\ 2 \end{bmatrix}$$

By applying Gaussian elimination, we can choose any two equations and calculate these. Finally, we can get $u = (2,3)$.

Suppose there are all data devices and checksum devices lost expect C_1, C_3, C_5, C_7, C_9, C_{12}, C_{13}, C_{14} and C_{15}. The contents of these unbroken checksum devices are 00, 00, 00, 00, 01, 00, 00, 00, 01. We fill up the truth table and get the string $u' = 0\Box 0\Box 0\Box 1\Box 0\Box\Box 0001\Box$, where $\Box$ denotes the erased symbol. Because there are total 7 unknown symbols more than $d(C_{RS-Had}) = 3 \cdot 2 = 6$, we need to use the soft recovering algorithm.

Construct the set h_1, h_2, h_3 and h_4 and we have $h_1 = \{0,1\}, h_2 = \{2,3\}, h_3 = \{0,3\}, h_4 = \{2\}$. Therefore, a weight function $w : [4] \times [4] \to [0,1]$ that

$$w = \begin{bmatrix} 0.5 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 1 \\ 0 & 0.5 & 0.5 & 0 \end{bmatrix}$$

Since $N_{1,k-1}(n-1) = N_{1,1}(3) = 10$, let $s = N_{1,k-1}(n-1) - 1 = 9$. The iterative construction of $\mathcal{M}$ is

Iteration	Point (i,j)	Later $\bar{w}_{ij}$	m_{ij}	Total cost
1	$(2,4)$	0.5	1	1
2	$(0,1)$	0.25	1	2
3	$(0,3)$	0.25	1	3
4	$(1,1)$	0.25	1	4
5	$(2,2)$	0.25	1	5
6	$(2,4)$	0.33	2	7
7	$(3,2)$	0.25	1	8
8	$(3,3)$	0.25	1	9

We have the multiplicity matrix $\mathcal{M}$ as follows:

$$\mathcal{M} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

The point set contains pairs: $(1,0)$, $(1,1)$, $(2,2)$, $(2,3)$, $(3,0)$, $(3,3)$ with multiplicity 1 and $(4,2)$ with multiplicity 2. Suppose a bivariate polynomial $\mathcal{Q}(x,y)$ that passes through these points. By solving the linear system, we can get

$$\begin{aligned} \mathcal{Q}(x,y) &= 3x + 3y + 2x^2 + 2xy + x^3 + y^3 \\ &= (y + 3x + 2)(y + x)(y + 2x + 2) \end{aligned}$$

We have three factors $p_1(x) = 2 + 3x$, $p_2(x) = x$, $p_3(x) = 2 + 2x$. So there are candidates $u(p_1) = 1302$, $u(p_2) = 1230$, $u(p_3) = 0132$. Because $Ag(u(p_1), w) = 2.5$, $Ag(u(p_2), w) = 1$, $Ag(u(p_3), w) = 2$. Therefore, the output is $u = (2,3)$.

References

[1] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes," *IEEE Transactions on Information Theory*, 45:1757–1767, September 1999.

[2] E. Kaltofen, "A Polynomial-Time Reduction from Bivariate to Univariate Integral Polynomial Factorization," *In 23rd Annual Symposium on Foundations of Computer Science*, 57–64, 1982.

[3] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed-Solomon codes" *IEEE Transactions on Information Theory*, August 31 2001

[4] N. Nisan and A. Wigderson, "Hardness vs randomness," *Journal of Computer and System Science*, 49:149–167, 1994.

[5] N. Nisan and D. Zuckerman, "More deterministic simulation in logspace," *In Proceedings of the 25th Annual ACM Symposium on the Theory Computing*, 235–244, 1993.

[6] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," *ACM SIGMOD International Conference on Management of Data*, 109–116, June 1988.

[7] R. Raz, O. Reingold, and S. Vadhan, "Extracting all the randomness and reducing the error in Trevisan's extractors," *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, 149–158, 1999.

[8] A. Ta-Shma and D. Zuckerman, "Extractor Codes," *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 193–199, July 2001.

[9] A. Ta-Shma, C. Umans, and D. Zuckerman, "Lossless Condensers, Unbalanced Expanders, and Extractors," *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, 2001.

[10] L. Trevisan, "Extractors and pseudorandom generators," *Journal of the ACM*, 48(4):860–879, July 2001.

[11] L. R. Welch and E. R. Berlekamp, "Error correction for algebraic block codes," *US patent*, Number 4, 633, 470, 1986.

[12] X-W. Wu and P.H. Siegel, "Efficient root-finding algorithm with application to list decoding of algebraic-geometric codes," *IEEE Trans. Inform. Theory*, 47:2579–2587, September 2001.