A Graph Theoretic Algorithm for Placing Data and Parity to Tolerate Two Disk Failures in Disk Array Systems

Narsingh Deo and Sanjeeb Nanda
School of Computer Science, University of Central Florida, Orlando, Florida 32816
deo@cs.ucf.edu, sanjeeb@earthlink.net

Abstract

In recent years commercial Redundant Arrays of Inexpensive Disks (RAID) systems have become increasingly popular because of their enhanced I/O bandwidths, large capacities and low cost. However, the continued demand for larger capacities at low cost, has led to the use of larger arrays with increased probability of random disk failures. Hence the need for RAID systems to tolerate two or more random disk failures without sacrificing performance or storage space. In this paper, we devise a novel graph-theoretic method for placing data and parity in an array of \( N \) disks (\( N \geq 3 \)) to enable its recovery from any two random disk failures. We first provide an algorithm for the case when the number of disks \( N = P - 1 \), where \( P \) is a prime number, and then generalize the solution for any arbitrary \( N \). We also determine the fraction of space used for storing parity in an array of \( N \) disks employing our algorithm, and show that this fraction has the optimal value of \( 2/N \) for all \( N = P - 1 \). For illustration, this fraction and the percentage of its difference from the optimal ratio are graphed for values of \( N \) between 5 and 255. Finally, we describe a method for determining the data-blocks from where the reconstruction of two failed disks can be started in such an array.

1. Introduction

While CPU throughputs have doubled approximately every two years, enhancements to I/O bandwidth, in contrast, have not kept pace. In fact, I/O bottlenecks continue to restrict many applications from fully exploiting their processors. This performance gap led to the introduction of Redundant Arrays of Inexpensive Disks (RAID) [6, 13, 14], which substitutes large expensive disks with arrays of inexpensive disks to provide greater effective I/O bandwidth. (Note that, the term *inexpensive* in RAID is sometimes substituted with *independent* in contemporary literature). However, arrays are probabilistically more susceptible to failure than each individual disk. To address this problem, Patterson, et al., originally defined RAID levels 1 through 5 [13] and the RAID Advisory Board subsequently added level 6.

Since RAID Levels 1 through 5 can recover from only one disk failure, their use is generally confined to small arrays where the probability of two or more disk failures is negligible. Larger arrays need to tolerate two or more arbitrary disk failures. For this purpose, the RAID Advisory Board defined RAID Level 6 as one that can tolerate two arbitrary disk failures. Several encoding schemes for double-disk failure recovery (in RAID6) have been developed over the years, each with their advantages and disadvantages. Blaum’s technique for recovery against double disk failures [2] requires the array size to be a prime number. Similarly, the EVENODD technique [3] restricts the size of the array to \( (P + 2) \) disks, \( P \) a prime. Both use bit-wise exclusive-OR of data blocks to calculate parity, and store it using an optimal fraction of the overall space in the array. However, both techniques have disadvantages. First, they restrict the size of the array to a prime, which is commercially unappealing. Second, they store parities exclusively on two disks. This chokes throughput when writing to the array since the corresponding parity updates are confined to two disks. On the other hand, read throughput from the array does not utilize the cumulative bandwidth of all disks since the two disks that exclusively contain parity do not contain any data to be read. On the other hand, techniques based on Reed-Solomon codes [1, 15] allow arbitrary array sizes. However, their use of Galois Field arithmetic to compute parities makes them computationally more expensive than those using exclusive-OR only. Hence, they are implemented using specialized hardware that can be economically unviable. Park’s algorithm [11] evaluates the number of disks \( N \) and the configuration of data and parity, when given the ratio of disk space
$R$ used for storing parity. However, the solution suffers from two drawbacks. The algorithm does not guarantee a solution for all values of $R$ and $N$ and even when it does so, the fraction of space in the array used for storing parity is generally not optimal.

In this paper, we present a novel graph-theoretic encoding algorithm for an array of $N$ disks, $N \geq 3$, which can recover from the failure of any two arbitrary disks in the array. The algorithm generates a solution with at most $(P - 3)/2$ equal sized blocks of data and one block of parity per disk, where $P$ is the smallest prime greater than $N$. Each parity block’s value is then obtained by computing the bit-wise exclusive-OR of data blocks specified by the algorithm. The solution is then extended for practical use by repeating the data and parity configuration of the data and parity blocks an integral number of times.

The algorithm has three distinct advantages over the competing techniques. First, the algorithm distributes parity uniformly over all the disks, permitting data to be read from, and parity to be written to all disks concurrently. This maximizes effective I/O throughput of the disk array. Second, the algorithm is computationally simple and inexpensive because it calculates each parity block’s value by using exclusive-OR operations. Finally, unlike some other schemes, our algorithm works for arrays with an arbitrary number of disks. Next, we describe our algorithm and then prove its correctness.

2. Algorithm to Create an Array of 
(Prime – 1) Disks to Tolerate Two Disk Failures

Given $P$ a prime, our algorithm generates an array of $(P - 1)$ disks with $(P - 3)/2$ equal sized blocks of data and one block of parity per disk. Each data-block is assigned a tag $\{u, v\}$, $0 \leq u < v \leq P - 1$, $u \neq v$, and its contents used for computing the bit-wise parity stored in the parity-blocks having the tags $\{u, u\}$ and $\{v, v\}$. Thus, the data in any block having a tag containing $v$ are used for computing the parity stored in the block with tag $\{v, v\}$. Also, the parity-block in disk $d$ is assigned the unordered pair $\{d, d\}$ as its tag.

For example, Fig. 1 displays such an array comprised of six disks. In this array, the content of the parity-block on disk $d$ having tag $\{d, d\}$ is calculated by taking the bit-wise exclusive-OR of all data-blocks having tags containing $d$. For instance, the parity stored in the parity-block having tag $\{3, 3\}$ is computed by taking the bit-wise exclusive-OR of the data-blocks having tags $\{3, 4\}, \{1, 3\}, \{3, 5\}$ and $\{0, 3\}$. Also, this array is tolerant to the failure of any two

| $\{3, 4\}$ | $\{0, 2\}$ | $\{1, 3\}$ | $\{2, 4\}$ | $\{0, 1\}$ | $\{0, 3\}$ |
| $\{2, 5\}$ | $\{4, 5\}$ | $\{0, 4\}$ | $\{1, 5\}$ | $\{3, 5\}$ | $\{1, 2\}$ |
| $\{0, 0\}$ | $\{1, 1\}$ | $\{2, 2\}$ | $\{3, 3\}$ | $\{4, 4\}$ | $\{5, 5\}$ |

Fig. 1. A 6-disk array created using our algorithm.

We can represent this fault-tolerant array by a graph where each data-block with tag $\{u, v\}$ corresponds to an edge $(u, v)$ and each parity-block with tag $\{v, v\}$ corresponds to a self-loop $(v, v)$. Furthermore, all edges and self-loops in the graph are labeled so that the blocks corresponding to the edges and self-loops with a common label belong to the same disk. For example, the graph in Fig. 2 represents the array in Fig. 1, and the blocks in the array corresponding to the edges $(3, 4)$, $(2, 5)$ and self-loop $(0, 0)$ that are labeled 6 belong to the same disk.

The aforesaid example illustrates the sufficient condition for a graph to correspond to a 2-disk fault tolerant array, which is as follows. A graph on $(P - 1)$
vertices with self-loops at each vertex corresponds to a 2-disk fault tolerant array of \((P - 1)\) disks if it admits a decomposition of its edges and self-loops into \((P - 1)\) matchings such that the union of any two matchings forms paths, each with a self-loop on at most one end. We show how to construct such graphs and prove the assertion in the following two theorems. First we state the following definition. A near-1-factor of a graph \(G\) is a subgraph of \(G\) that includes all the vertices of \(G\) and in which one vertex is isolated and all others have degree 1. (Note that the order of \(G\) must then be odd.)

**Theorem 1.** Let \(Q_P = (V, E)\) be the complete graph on \(V = \{0, 1, \ldots, P - 1\}\), \(P\) a prime, with a self-loop at each vertex. Then, \(Q_P\) can be factored into \(P\) near-1-factors, each associated with a self-loop, such that the union of any two near-1-factors forms a Hamiltonian path with their associated self-loops at the ends.

**Proof.** Label each edge \((u, v)\) of \(Q_P\) with \(2P - u - v - 1 \pmod{P}\) (including the self-loop when \(u = v\)). Then, we shall show that, (a) the graph on \(V(Q_P)\) formed by the edges with a common label is a near-1-factor of \(Q_P\), and (b) the union of no two near-1-factors contains a cycle, and thus the union of any two near-1-factors forms a Hamiltonian path. Next we shall associate with each near-1-factor the self-loops labeled the same as its edges. We shall then show that, one self-loop is at each end of the Hamiltonian path formed by the union of their associated near-1-factors.

(a) First, we show that there are \((P - 1)/2\) independent edges with a common label by proving that no two edges having a common label can be adjacent. Suppose to the contrary that, there exist a pair of adjacent edges \((u, v)\) and \((v, w)\) both labeled \(e\), using the labeling scheme stated at the beginning of the proof. Then, we must have \(e \equiv 2P - u - v - 1 \pmod{P}\) and \(e \equiv 2P - v - w - 1 \pmod{P}\). That is, \(2P - u - v - 1 \equiv 2P - v - w - 1 \pmod{P}\) and therefore, \(u \equiv w \pmod{P}\). Since \(0 \leq u, w \leq P - 1\), we must have \(u = w\). A contradiction. Now, if there are less than \((P - 1)/2\) edges labeled \(e_1\) say, then there must be more than \((P - 1)/2\) edges label \(e_2\), where \(e_1 \neq e_2\), \(0 \leq e_1, e_2 \leq P - 1\). This is a consequence of the pigeonhole principle, for there are \(P(P - 1)/2\) edges in \(Q_P\) and \(P\) labels. Then, two edges labeled \(e_2\) must be adjacent. A contradiction. Thus, we have shown that there are exactly \((P - 1)/2\) independent edges having a common label, and therefore, the graph on \(V(Q_P)\) formed by the edges with a common label is a near-1-factor of \(Q_P\).

(b) Next, we show that the union of no two near-1-factors contains a cycle. Suppose to the contrary, there exists a cycle \(v_1, v_2, v_3, \ldots, v_t, v_1\) in the union of two near-1-factors with edges alternately labeled \(e_1\) and \(e_2\). Note that each near-1-factor has exactly \((P - 1)/2\) edges, therefore \(t \leq P - 1\). Without loss of generality assume that the edges \((v_1, v_2), (v_3, v_4), \ldots, (v_{t - 1}, v_t)\) are labeled \(e_1\), and \((v_2, v_3), (v_4, v_5), \ldots, (v_t, v_1)\) are labeled \(e_2\). Then,

\[
\begin{align*}
e_1 &\equiv 2P - v_1 - v_2 - 1 \pmod{P}, \\
e_2 &\equiv 2P - v_2 - v_3 - 1 \pmod{P}, \\
e_1 &\equiv 2P - v_3 - v_4 - 1 \pmod{P}, \\
e_2 &\equiv 2P - v_4 - v_5 - 1 \pmod{P}, \\
&\vdots \\
e_1 &\equiv 2P - v_{t - 1} - v_t - 1 \pmod{P}, \\
e_2 &\equiv 2P - v_t - v_1 - 1 \pmod{P}
\end{align*}
\]

Then, it can be proved that \(t(e_1 - e_2)/2 \equiv 0 \pmod{P}\). This is a contradiction since \(P\) is a prime and \(t, e_1\) and \(e_2\) are each less than \(P\).

From (a) and (b) we see that, the union of any two near-1-factors contains \((P - 1)\) edges and does not have a cycle. Furthermore, since no two edges in any near-1-factor can have a common vertex, a vertex in \(Q_P\) can have at most two edges from the union of any two near-1-factors incident to it. Therefore, the union of any two near-1-factors must contain a Hamiltonian path.

Now, we associate with each near-1-factor the self-loops having the same label as its edges. Then, each near-1-factor must be associated with exactly one self-loop, for otherwise by the Pigeonhole principle some near-1-factor would be associated with two self-loops on two distinct vertices \(v\) and \(w\). But since they must have the same common label, we would have \(2P\)

![Fig. 2. A 6-disk array represented as a graph.](image-url)
\[ -2v - 1 \equiv 2P - 2w - 1 \pmod{P}, \] which would yield, \( 2(v - w) \equiv 0 \pmod{P} \), and this would imply that \( v = w \). A contradiction since \( P \) is a prime. Hence, each near-1-factor is associated with a distinct self-loop. Now, suppose a self-loop is not at the end of the Hamiltonian path formed by the union of its associated near-1-factor and another near-1-factor. Suppose that self-loop is on some vertex \( v \). Then, \( v \) must have two vertices \( u \) and \( w \) adjacent to itself on that Hamiltonian path. Since edges in a near-1-factor are independent, \( (u, v) \) and \( (v, w) \) must belong to different near-1-factors, and since the self-loop is associated with one of the near-1-factors in that union, we must have, \( 2P - u - v - 1 \equiv 2P - 2v - 1 \pmod{P} \) or \( 2P - v - w - 1 \equiv 2P - 2v - 1 \pmod{P} \). This implies that, either \( u = v \) or \( v = w \). A contradiction. Thus, a self-loop must be at an end of the Hamiltonian path formed by the union of its associated near-1-factor and another near-1-factor. Thus, we have shown that the self-loops are at each end of the Hamiltonian path formed by the union of their associated near-1-factors.

Let \( \Phi \) be the set of \( P \) near-1-factors and their associated self-loops of \( Q_P \), as described in the proof of Theorem 1. For instance, Fig. 3 displays the labeling of the edges of \( Q \) to form seven near-1-factors and the Hamiltonian path formed by the edges in the union of the near-1-factors containing edges labeled 0 and 1. Note also that, the self-loops labeled 0 and 1 occur at the ends of that Hamiltonian path.

![Fig. 3. The graph \( Q \) with the Hamiltonian path formed by the edges labeled 0 and 1 highlighted by greater thickness.](image)

**Theorem 2.** \( Q_P = (V, E) \) contains a solution for a two-disk fault-tolerant array with \( (P - 1) \) disks, \( P \) a prime.

**Proof.** Let \( W \) be a singleton subset of \( V \). Remove from \( \Phi \), the self-loop at the vertex in \( W \) and all edges in the near-1-factors that have the same label as that self-loop. This will result in one near-1-factor and its associated self-loop being removed. Next, remove each edge that is incident to the vertex in \( W \) from the remaining \( (P - 1) \) near-1-factors to yield \( (P - 1) \) matchings. Let \( \Pi \) be the set containing these \( (P - 1) \) matchings and their associated self-loops. Now consider the edges in the union of any pair of matchings in \( \Pi \). By Theorem 1, they form a Hamiltonian path minus those edges incident to the vertex in \( W \). Therefore, any vertex incident to those removed edges can have at most one edge incident to it in that union, and every other vertex can have at most two edges incident to it.

We use the matchings in \( \Pi \) to represent an array of \( (P - 1) \) disks as follows. Each matching corresponds to a disk, an edge \( (u, v) \) in a matching represents the data block with tag \( \{u, v\} \) in the corresponding disk, and a self-loop \( (v, v) \) that is associated with a matching, represents the parity-block with tag \( \{v, v\} \) in the disk corresponding to the matching. From this representation we observe that, the contents of any parity-block is derived from at most two data-blocks in any pair of disks. Note that, each matching in \( \Pi \) is obtained by removing those edges incident to the vertex in \( W \) from a near-1-factor in \( \Phi \).

Now, suppose two disks have failed in the array. Then, we can reconstruct the data blocks in those disks as follows. Select an edge \( (u, v) \) in a near-1-factor in \( \Phi \) corresponding to a failed disk such that \( u \) is adjacent to the vertex in \( W \) in the near-1-factor corresponding to the other failed disk. Then, no edge other than \( (u, v) \) in the two matchings in \( \Pi \) corresponding to the failed disks can be incident to the vertex \( u \). Hence, there is no data block in the two failed disks except the one having tag \( \{u, v\} \) that contributes to the contents of the parity-block having tag \( \{u, u\} \). Furthermore, the parity-block having tag \( \{u, u\} \) cannot be on either failed disk for the following reason. Parity-block tags correspond to self-loops, and self-loops are on the ends of the Hamiltonian path formed by the edges in the union of any pair of near-1-factors in \( \Phi \). If the parity-block having tag \( \{u, u\} \) were on a failed disk, \( u \) would be an end of the Hamiltonian path formed by the edges in the near-1-factors in \( \Phi \) corresponding to the failed disks. This is a contradiction, since by assumption \( (u, v) \) is an edge.

Thus, we can reconstruct the data block having tag \( \{u, v\} \) by taking the XOR of the data in all blocks.
containing \( u \) in their tags. We can then reconstruct a data block having a tag containing \( v \), \( \{v, w\} \) say, by taking the XOR of the data in all blocks with tags containing \( v \). This is possible because a pair of disks can have at most two data blocks that contribute to the contents of any given parity-block. Next, we can reconstruct a data block having a tag different from \( \{v, w\} \) containing \( w \). Hence, in each iteration, we select a data block for reconstruction that has a tag with an element in common with the tag of the data block reconstructed in the previous iteration. Thus, a data block having tag \( \{u, v\} \) can be reconstructed if \( (u, v) \) is on a path comprised of the edges in the matchings in \( \Pi \) corresponding to the failed disks, and the path has an end-vertex \( x \), with \( x \) adjacent to the vertex in \( W \) in the corresponding near-1-factor in \( \Phi \). Since the edges corresponding to the data-block tags of two failed disks form a Hamiltonian path minus those edges incident to the vertex in \( W \), each edge is therefore on a path having an end-vertex that is adjacent to the vertex in \( W \). Thus, we can reconstruct each and every data block on a failed disk.

Let \( Q_p = (V, E) \) be the complete graph on \( V = \{0, 1, \ldots, P - 1\} \) with a self-loop at each vertex, where \( P \geq 5 \) and \( P \) a prime. Each edge \( (u, v) \) in \( E \) is assigned the label \( l(u, v) = (2P - u - v - 1) \mod P \), and the self-loop \( (v, v) \) at each vertex \( v \) is assigned the label \( l(v, v) = (2P - 2v - 1) \mod P \). Using \( Q_p \), we create an array of \( (P - 1) \) disks as described by the algorithm in Fig. 4.

3. Algorithm for Parity and Data Placement in Arbitrary Sized Arrays

We note that the union of any two matchings in \( \Pi \) is obtained by removing the edges incident to the vertex in \( W \) from the Hamiltonian path formed by the union of the corresponding near-1-factors in \( \Phi \). Now, if \( W \) were to have two or more vertices, then the union of any two matchings in \( \Pi \) would be obtained by removing the edges incident to all the vertices in \( W \) from the Hamiltonian path formed by the union of the corresponding near-1-factors in \( \Phi \). It is then easy to see that, the union of any two matchings in \( \Pi \) would still yield paths, each with a self-loop on at most one end. Then, the disks corresponding to a pair of such matchings can be reconstructed in exactly the same manner as that described in the proof for Theorem 2.

### Fig. 4. Pseudo-code for the algorithm to place data and parity blocks in a \((P - 1)\) disk array to tolerate two random disk failures.

```plaintext
Step 1: for \( u \leftarrow 0 \) to \((P - 1)\) do
  for \( v \leftarrow u \) to \((P - 1)\) do
    Assign \((u, v)\) the label \( l(u, v) \leftarrow (2P - u - v - 1) \mod P \);
Step 2: Let \( W \) be a singleton subset of \( V \);
Step 3: for each disk \( d \) in \( V - W \) do
  \{ b \leftarrow 0;
    for \( u \leftarrow 0 \) to \((P - 2)\) do
      for \( v \leftarrow (u + 1) \) to \((P - 1)\) do
        if\((u \in V - W) \) and \((v \in V - W) \) and \( (l(u, v) = l(d, d)) \) then
          \{ block \( b \) in disk \( d \) is assigned the data-block tag \( \{u, v\} \);
            \( b \leftarrow b + 1; \)
          \}
      \}
  \}
  block \( b \) of disk \( d \) is assigned the parity-block tag \( \{d, d\} \);
```

Note that, Fig. 2 displays the edges of \( Q_p \) remaining upon deleting the self-loop and all the edges that have the same label as the self-loop at the vertex in \( W \), in addition to deleting all the edges incident to the vertex in \( W \), where \( W = \{6\} \). Then, the 6-disk system shown in Fig. 1 is obtained by placing the data-block and parity-block tags corresponding to the edges
We can then extend the algorithm presented in Fig. 4 to create an array of $N$ disks that tolerates two disk failures by choosing $W$ in Step 2 such that $|W| = P - N$, $N \geq 3$. We apply this algorithm to create an array of eight disks and display it in Fig. 5.

<table>
<thead>
<tr>
<th>Disk 0</th>
<th>Disk 1</th>
<th>Disk 2</th>
<th>Disk 3</th>
<th>Disk 4</th>
<th>Disk 5</th>
<th>Disk 6</th>
<th>Disk 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>{4, 7}</td>
<td>{0, 2}</td>
<td>{0, 4}</td>
<td>{0, 6}</td>
<td>{1, 7}</td>
<td>{1, 3}</td>
<td>{1, 5}</td>
<td>{2, 4}</td>
</tr>
<tr>
<td>{5, 6}</td>
<td>{6, 7}</td>
<td>{1, 1}</td>
<td>{2, 2}</td>
<td>{2, 6}</td>
<td>{4, 5}</td>
<td>{3, 5}</td>
<td>{5, 7}</td>
</tr>
<tr>
<td>{0, 0}</td>
<td>{0, 1}</td>
<td>{0, 3}</td>
<td>{3, 3}</td>
<td>{3, 4}</td>
<td>{4, 6}</td>
<td>{5, 5}</td>
<td>{6, 6}</td>
</tr>
</tbody>
</table>

Fig. 5. Algorithm applied to create an eight-disk array with $V - W = \{0, 1, 2, 3, 4, 5, 6, 7\}$.

An observation made from the eight-disk array shown in Fig. 5 is that, the ratio of its space used for storing parity is 8/26. Now, the optimal ratio of space used for storing parity in an $N$ disk array equals $2/N$ for the following reason. Each data bit on a failed disk must be reconstructed using a unique parity bit. Then there must be at least as many parity bits on the surviving disks as there are data bits on the failed disks. Suppose the ratio of space used for storing parity in each disk in an $N$ disk array is $r$ and that two disks have failed. Then to reconstruct the contents of the failed disks, we must have $(N - 2)r \geq 2(1 - r)$, and therefore $r \geq 2/N$. Hence the optimal ratio of space used for storing parity in an eight-disk array is 2/8. In contrast, the ratio of space used by our algorithm for storing parity in an eight-disk array exceeds the optimal ratio. In comparison, the ratio of space used for storing parity in the six-disk array shown in Fig. 1 has the optimal value of 6/18. Clearly, our algorithm does not consume an optimal ratio of space for storing parity for all array sizes. The following section derives an upper bound for the ratio of space used for storing parity for any arbitrary value of $N$, and shows that the value is optimal for all $N = P - 1$, $P$ a prime.

### 4. Fraction of Space Used for Parity

To obtain the ratio of space used for parity in an array with $N$ disks, we first obtain the number of data blocks in the array. In the preceding section we described the manner in which we can delete edges from the graph $Q_P$ to obtain a 2-disk fault-tolerant solution with $N = P - k$ disks and noted that, the number of data blocks in the array equals the number of remaining edges. Let us now determine that number by iteratively removing a vertex $v$, followed by the edges in the matching that are labeled $2P - 2v - 1$ (mod $P$), and finally, the edges in every matching containing $v$.

Suppose that in the first iteration we remove from $Q_P$ the vertex $u$, the $(P - 1)$ edges incident to $u$ and the $\frac{1}{2}(P - 1)$ edges labeled with the value $2P - 2u - 1$ (mod $P$). Then, the remaining vertices are each of degree $(P - 3)$. Suppose that in the second iteration we remove from $Q_P$ the vertex $v$, the $(P - 3)$ edges incident to $v$ and the $\frac{1}{2}(P - 2)$ edges labeled with the value $2P - 2v - 1$ (mod $P$). Since $P$ is prime, and therefore odd, $\left\lfloor \frac{1}{2}(P - 2) \right\rfloor = \frac{1}{2}(P - 3)$. At this point, one vertex is of degree $(P - 4)$ and the remaining ones are each of degree $(P - 5)$ for the reason that, one vertex does not have an edge labeled with the value $2P - 2v - 1$ and must have degree $(P - 4)$. Now, to ensure that we delete the least number of edges in the next iteration, we choose for deletion the vertex $w$ with the least degree i.e., $(P - 5)$ in addition to the $(P - 5)$ edges incident to $w$ and the $\frac{1}{2}(P - 3)$ edges labeled with the value $2P - 2w - 1$ (mod $P$). Proceeding in this manner, we observe that the maximum number of edges removed in each iteration is as follows.

- **Iteration 1**: $(P - 1) + \frac{1}{2}(P - 1)$
- **Iteration 2**: $(P - 3) + \frac{1}{2}(P - 3)$
- **Iteration 3**: $(P - 5) + \frac{1}{2}(P - 3)$
- **Iteration 4**: $(P - 7) + \frac{1}{2}(P - 5)$
- **Iteration 5**: $(P - 9) + \frac{1}{2}(P - 5)$
- : : : 
- **Iteration $i$**: $(P - 2i + 1) + \frac{1}{2}(P - \frac{1}{2}(2i + 1 + (-1)^i))$

Thus the total number of edges removed up to and including iteration $k$ is:

$$
\leq \sum_{i=1}^{k} (P - 2i + 1) + \frac{1}{2}(P - \frac{1}{2}(2i + 1 + (-1)^i))
= \frac{1}{2}[k(6P - 5k - 2) + k \mod 2]
$$
Thus, the number of edges remaining at the end of $k$ iterations is:
$$\geq \frac{1}{2}P(P-1) - \frac{1}{4}[k(6P - 5k - 2) + k \mod 2]$$

Hence, the ratio of space used for parity in the array is:
$$\leq \frac{1}{2}P(P-1) - \frac{1}{4}[k(6P - 5k - 2) + k \mod 2] + (P - k)$$

It is easy to verify that this ratio has the optimal value of $2/(P - 1)$ for $k = 1$ by observing from the preceding section that, the optimal ratio of space used for storing parity in an $N$ disk array equals $2/N$. For example, Fig. 6 displays the percentage difference in the ratio of space used for parity by the algorithm compared to the optimal value for array sizes between 5 and 255.

5. Reconstruction

In the proof of Theorem 2, we started reconstructing the failed disks from a block $\{u, v\}$ where $(u, v)$ was an edge in the near-1-factor in $\Phi$ corresponding to one of the failed disks and $u$ was adjacent to a vertex in $W$ in the near-1-factor corresponding to the other failed disk. We shall now derive the value of $u$ as a function of a vertex in $W$. Suppose that disks $d_1$ and $d_2$ have failed. We determine the block from where we can start reconstructing the failed disks as follows. In our algorithm, each block in a disk indexed $d$ has a label of the form $\{u, v\}$ with $2P - u - v - 1 \equiv 2P - 2d - 1 \pmod{P}$. From the proof of Theorem 2 we observe that, we can reconstruct a block having tag $\{u, v\}$ in disk $d_1$ if, for some vertex $w$ in $W$, the edge $(u, w)$ is in a near-1-factor in $\Phi$ corresponding to disk $d_2$. The edge $(u, w)$ is labeled $2P - u - w - 1$. Thus, we can start reconstruction of a block $(u, v)$ on disk $d_1$ that satisfies $2P - u - w - 1 \equiv 2P - 2d_2 - 1 \pmod{P}$. That is, $u \equiv 2d_2 - w \pmod{P}$.

Conversely, we can start reconstruction of a block $(u, v)$ on disk $d_2$ that satisfies $u \equiv 2d_1 - w \pmod{P}$.

We illustrate the aforementioned idea using the six-disk array given in Fig. 2. As noted earlier following the proof to Theorem 2, this array is created using $W = \{6\}$ and $P = 7$. Now, suppose that the disks indexed 0 and 1 have failed. Then $d_1 = 0$ and $d_2 = 1$. Then, the block on the disk indexed 0 from which we can start reconstruction is the one labeled $\{u, v\}$, where $u \equiv 2 \cdot 0 - 6 \pmod{7} = 1$. This is the parity block having the label $\{1, 1\}$. Similarly the block on the disk indexed 1 from which we can start reconstruction is the one labeled $\{u, v\}$, where $u \equiv 2 \cdot 1 - 6 \pmod{7} = 3$. This is the data block with label $\{3, 4\}$.

6. Conclusion

The algorithm we have presented in this paper is very practical because of it ability to generate solutions for creating arbitrary sized arrays using bitwise XOR operations only. However, the non-optimal ratio of space utilized for storing parity in arrays other than those having prime minus one number of disks, is
a drawback. Overcoming this challenge is an area for future research.

7. References


