A TALE OF TWO PUZZLES

LEANNE MERRILL AND TONY VAN

PAUL CULL
OREGON STATE UNIVERSITY

ABSTRACT. Towers of Hanoi and Spin-Out are two puzzles with different physical manifestations but similar graphical properties. Towers of Hanoi is well known and Spin-Out less so, and in this paper we discuss a puzzle formed by a combination of concepts from both puzzles. Towers of Hanoi may be generalized to a puzzle in any odd dimension and Spin-Out to any 2^m-dimension puzzle, and so their combination may be generalized to any dimension as any natural number may be expressed as the product of an odd number and a power of two. Algorithms exist to solve both puzzles separately; here, we introduce a counting algorithm for the generalized Spin-Out puzzle and recursive, iterative, and counting algorithms for the combination puzzle. Interestingly, these algorithms do not explicitly utilize the Gray code properties of the associated graphs; instead, they use only simple binary counters. The graphs associated with these puzzles are complete iterated graphs with interesting properties. We discuss some properties of Hamiltonian paths on these graphs and a method for ternary to ternary reflected Gray code conversion.

1. INTRODUCTION

Mathematicians have studied the classic puzzle Towers of Hanoi for over a century. The traditional game is played with three pegs and n uniquely sized disks. The disks are initially stacked together on one of the three towers starting with the biggest disk at the bottom of the stack. The goal of the game is to move all the disks off of its starting tower and re-stack them onto some target tower. In doing so, two simple rules must be obeyed. Only one disk may be moved at a time, and a larger disk can never be placed on top of a smaller disk. It is also well known that the minimum number of moves to re-stack the n disks from one tower onto another is is $2^n - 1$ [10]. Furthermore, the solution path that uses this minimum number of moves is unique. The Towers of Hanoi is easily represented by a complete iterated graph whose vertices correspond to legal orientations and whose edges correspond to legal moves between them. We develop a finite state machine to determine whether a state of the puzzle lies on the minimal solution path, and if so, the corresponding index of that state within the sequence of moves defined by the minimal solution path.

Spin-Out is another puzzle that has been well studied. Playing Spin-Out is analogous to unlocking a lock with n dials, where the ability to reconfigure a dial will depend on the configuration of the other $n-1$ dials. The traditional toy has seven dials, and each dial can be in one of two states, that is, it is either locked or unlocked. Like the Towers of Hanoi, standard recursive and iterative algorithms exist to solve Spin-Out. In addition to these, we develop a counting algorithm

Date: August 13, 2010.
This work was done during the Summer 2010 REU program in Mathematics at Oregon State University.

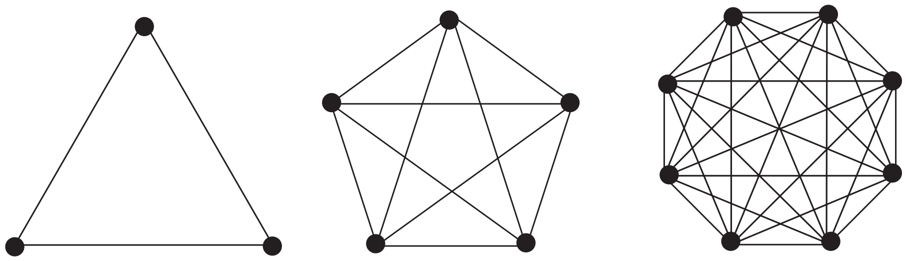


FIGURE 1. The complete graphs on 3, 5, and 8 vertices, respectively.

for Spin-Out and describe the Gray labeling representation of Spin-Out. We also explore Gray code properties of the complete iterated graphs used to represent Spin-out.

The Towers of Hanoi and Spin-Out puzzles can be generalized to higher-dimensional puzzles known as the SF puzzle and the Dimension 2^m puzzle, respectively. In the case of the Towers of Hanoi, this entails adding more towers, and in the case of Spin-Out, the Dimension 2^m puzzle allows each dial to be in more than two states. We describe the iterated complete graphs that are induced by the generalized puzzles. Finally, with the addition of the counting algorithm for solving Spin-Out, there now exists recursive, iterative, and counting algorithms for solving these generalized versions [31] and [4].

Now, a new puzzle can be created by merging together the Towers of Hanoi and the Spin-Out puzzle. This new puzzle is the Combination (or Product) puzzle, developed by Skubak and Stevenson [31]. The Combination is played with q pegs and n pieces. These pieces are no longer simply disks; instead they are stacks of spinners, each of which can spin relative to each other. The major focus of our paper will be describing the recursive, iterative, and counting algorithms for solving the Combination puzzle.

2. GRAPHS AND LABELINGS

2.1. **Complete Iterated Graphs.**

Many of the definitions in this section were modeled on those in previous papers by [4] and [31].

Definition 2.1. *A **graph** $G(V,E)$ consists of a finite set $V(G)$ of* vertices *and unordered pairs (v_i, v_j) of vertices that form the set $E(G)$ of* edges. *We call two vertices v_i, v_j **adjacent** if there is an edge between them, meaning that $(v_i, v_j) \in E$. A graph is called **simple** if no edges are duplicated and no edge is of the form (v_i, v_i); that is, there is no single edge from one vertex to itself.*

Definition 2.2. *The **degree** of a vertex v is the number of vertices that are adjacent to v.*

Definition 2.3. *A **complete graph on** q **vertices**, K_q, is a simple graph with q vertices such that each vertex is adjacent to every other vertex. In a simple complete graph on q vertices, each vertex will have degree $q-1$. See Figure 1 for examples.*

Definition 2.4. *A **complete iterated graph on** q **vertices with** n **iterations**, K_q^n, is defined recursively. K_q^1 is K_q, the complete graph on q vertices. K_q^n is composed of q copies of K_q^{n-1} and edges such that exactly one edge connects each K_q^{n-1} subgraph to every other K_q^{n-1} subgraph and exactly one vertex in each of the K_q^{n-1} subgraphs has degree $q-1$. The rest of the vertices will have degree q. This fact motivates the following definitions.*

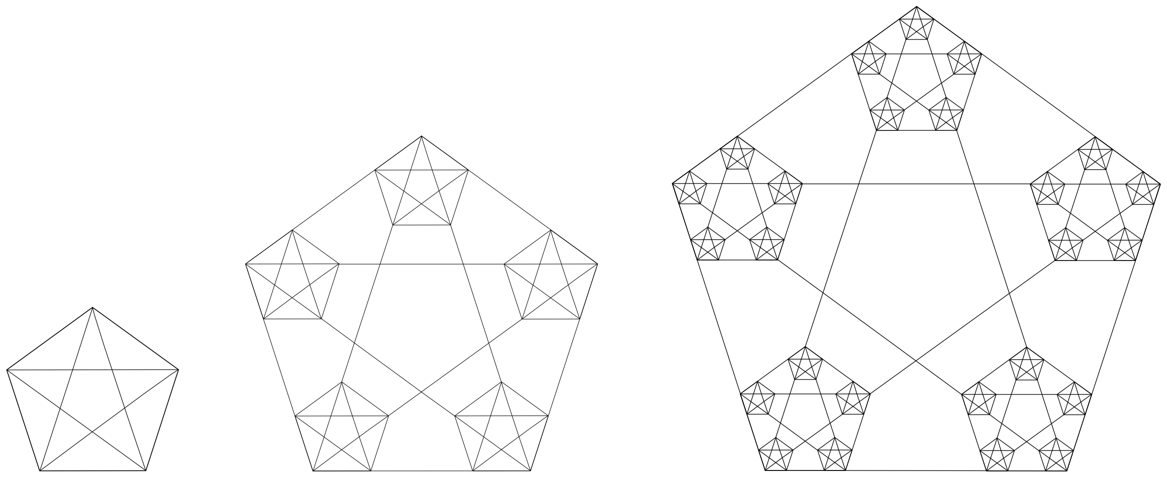


FIGURE 2. The complete iterated graphs K_5^1, K_5^2, K_5^3.

Definition 2.5. *A* ***corner vertex*** *in a complete iterated graph on q vertices with n iterations is a vertex with degree $q-1$. A* ***non-corner vertex*** *is a vertex that is not a corner vertex. A non-corner vertex has degree q.*

Definition 2.6. *A* ***subgraph*** *H of a graph G consists of a subset $V(H) \subseteq V(G)$ along with a subset $E(H) \subseteq E(G)$ of the edges associated with the vertices in $V(H)$.*

Note that the q copies of K_q^{n-1} from which K_q^n is constructed are all subgraphs of K_q^n. It is helpful to think of K_q^n as K_q^{n-1} with each vertex replaced with a copy of K_q^1. Figure 2 shows the complete iterated graphs K_5^1, K_5^2, and K_5^3.

2.2. **Gray Code Labeling.**

Definition 2.7. *A* ***labeling*** *on a graph G is a method of assigning strings of characters to the vertices of G such that there is a bijection between vertices and strings. The string assigned to a vertex will be called the* ***label*** *of that vertex.*

Definition 2.8. *Let G be a graph. A labeling of G has the* ***Gray code property*** *if every pair of adjacent vertices have labels that differ in exactly one position.*

Definition 2.9. *The* ***binary reflected Gray code*** *is constructed iteratively as follows. First, list the binary numbers 0 and 1. Next, reflect them, and prepend a 0 to the first two and a 1 to the second two. Continue reflecting and prepending in this manner. In general, the n^{th} iteration of the binary reflected Gray code can be represented as follows:*

$$0G_{n-1} \,||\, 1G_{n-1}^R$$

where R indicates a reflection.

Note that reflected Gray codes for any base number system exist; however, the binary reflected Gray code has a simple relation to the binary numbers that will become important in our puzzles.

The conversion from binary to binary reflected Gray code is a bijection that depends only on the previous two bits of the binary number.

Definition 2.10. *We will use the following formulae to convert between binary reflected Gray code and binary numbers.*

$$b_{n-1} = g_n \quad ; \quad b_{n-i} = g_{n-i} + b_{n-i+1}$$

and

$$g_{n-1} = b_n \quad ; \quad g_{n-i} = b_{n-i} + b_{n-i+1}$$

where $i \geq 2$.

These are inverses as addition and subtraction are equivalent in binary. Below are the binary numbers 0 through 7 and the corresponding binary reflected Gray code.

Binary	Gray Code
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Theorem 2.11. *The above formulae for conversion from binary to the binary reflected Gray code are correct.*

Proof. Let G be the function that converts a string in binary to a string in binary reflected Gray code. Then $G(b_k,\dots,b_1) = (b_k, b_k+b_{k-1}, b_{k-1}+b_{k-2},\dots,b_2+b_1)$. We will proceed by induction.

Base Cases: If $b_k = 0$, then $G(0,b_{k-1},\dots,b_1) = 0G(b_k,\dots,b_1)$.

If $b_k = 1$, then $G(1,b_{k-1},\dots,b_1) = 1G(2^k-1-(b_{k-1},\dots,b_1))$ as the Gray words are in reverse order when $b_k = 1$; that is, in the second half of the list.

Inductive Step: If $b_{k-1} = 0$, then $G(2^k-1-(b_{k-1},\dots,b_1)) = 1G(2^k-1-(b_{k-2},\dots,b_1))$.
If $b_{k-1} = 1$, then $G(2^k-1-(b_{k-1},\dots,b_1)) = 0G(2^k-1-(b_{k-2},\dots,b_1))$.
In either case,

$$G(2^k-1-(b_{k-1},\dots,b_1)) = (b_k+1)G(2^k-1-(b_{k-2},\dots,b_1))$$

and

$$G(1(b_{k-1},\dots,b_1)) = 1, b_k+1, G(2^k-1-(b_{k-2},\dots,b_1)).$$

Thus, the claim holds.

□

3. Towers of Hanoi and the SF Labeling

3.1. **The Towers of Hanoi.** The Towers of Hanoi is a puzzle dating from 1883. Typically, it is composed of three towers and any number n of disks, all of which are of different sizes. Typically, the disks start off initially stacked on tower 0 (the leftmost tower), in size order with the largest on the bottom and the smallest on the top. The goal is to move all of the disks to tower 2 (the rightmost tower), again stacked from largest to smallest, with the largest on the bottom. There are only two rules for the Towers of Hanoi. First, only one disk may be moved at a time. Second, a larger disk may never be placed on top of a smaller disk.

It is well known that the minimum number of moves necessary to solve the Towers of Hanoi is $2^n - 1$, where again n is the number of disks. Furthermore, it has been proven that the the minimum solution is unique; that is, there is only one sequence of moves that follows the rules and solves the puzzle in the smallest number of moves [10]. We will refer to this sequence of moves as the "minimal" or "optimal" solution, and we will define any method that finds this sequence of moves as "correct." Note that there are many other possible configurations for the puzzle that are legal (can be reached by legal sequence of moves) but that are not part of the minimal solution.

The Towers of Hanoi can be played on any odd number $q \geq 3$ of towers, and the minimal solution will still contain on $2^n - 1$ moves. We will introduce an analogous puzzle for higher dimensions, the SF Puzzle, for any odd number q of towers with rules similar to those of the classic Towers of Hanoi. We will sometimes refer to the SF Puzzle as the Generalized Towers of Hanoi. This language is clarified below.

Definition 3.1. *Let n be the number of disks and $q \geq 3$ with q odd be the number of towers in the Towers of Hanoi puzzle. A **configuration** in the Generalized Towers of Hanoi is an arrangement of specific disks on specific towers. We will call a configuration **legal** if it is possible to be reached by a sequence of moves that does not violate the rules of the puzzle. We will represent configurations by strings of n characters, where each character is an integer in $\mathbb{Z}/q$. The strings will be read from left to right, with the n^{th} character corresponding to the tower number of the n^{th} disk. We will call a solution to the puzzle **correct** if it performs legal moves and completes the puzzle in the minimal number of moves, which is known to be $2^n - 1$ where n is the number of disks.*

Definition 3.2. *The **SF Labeling** of an odd-dimensional iterated graph K_q^n is a recursively-constructed labeling of the vertices of the graph corresponding to all legal configurations in the Generalized Towers of Hanoi with n disks and q towers. Figure 3 corresponds to the puzzle with 5 towers and 3 disks.*

3.2. **Construction of of SF Labeling.** Let $q \geq 3$ be an odd number. The labeling of K_q^n is constructed recursively from the labeling of K_q^{n-1}. Designate a "top" vertex in K_q^1 and label it 0. Continue counterclockwise with $1, 2, \ldots, d-1$.

For each K_q^{n-1}, apply the permutation $\alpha(z) = \frac{q+1}{2} z \mod q$ where z is a digit in the label of a vertex in K_q^{n-1}. Make q copies of the permuted K_q^{n-1}s. Rotate each i^{th} copy $\frac{2\pi i}{q}$ radians clockwise, and append i to each word in this copy. Finally, connect the d copies to form K_q^n with the i^{th} copy centered at $\frac{2\pi i}{q}$ radians clockwise from 0. See Figure 4 for the construction of K_3^1, K_3^2, and K_3^3.

The SF Puzzle is a generalization of the Towers of Hanoi for any number n of disks and any odd number $q \geq 3$ of towers. The initial and final configurations are the same as those in the classic

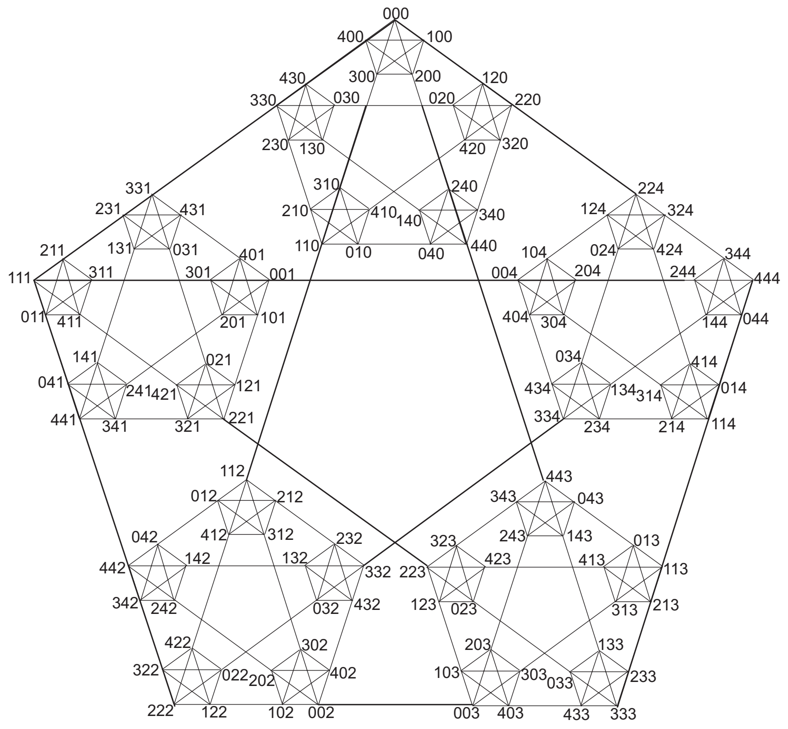


FIGURE 3. The complete iterated graph K_5^3 with the SF Labeling.

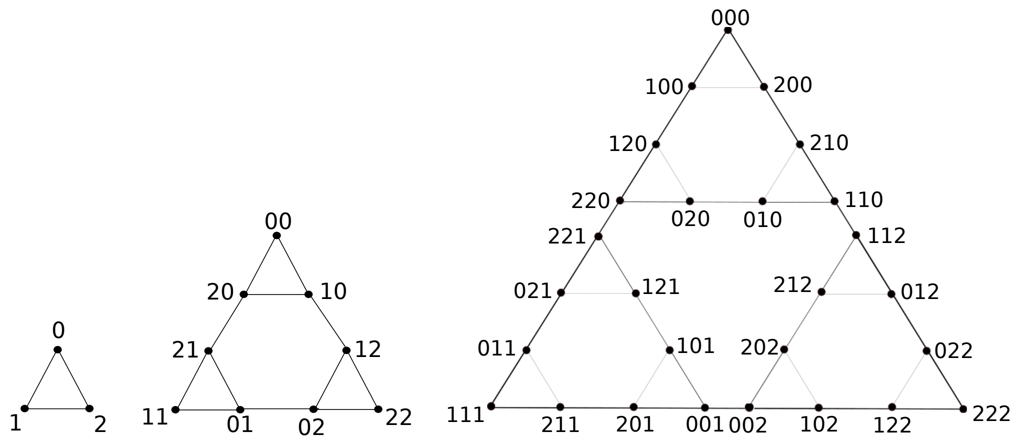


FIGURE 4. K_3^1, K_3^2, and K_3^3 with the SF Labelings.

Towers of Hanoi; i.e., the disks will begin stacked on tower 0 and end stacked on tower $q-1$, ascending from largest to smallest.

The construction of the SF Labeling of the complete graphs gives all legal permutations of these strings. Further, edges connecting adjacent vertices correspond to legal moves in the SF Puzzle.

The SF Puzzle has the following rules, each of which are analogous to the rules for the Towers of Hanoi puzzle and ensure that the puzzle satisfies the SF Labeling:

(1) Only one disk moves at a time;
(2) A larger disk is never placed on top of a smaller disk;
(3) No disk may move unless all of the smaller disks are stacked on the same tower;
(4) If a disk other than the smallest is able to move, and that disk is on tower b and the stack of smaller disks is on tower a, that disk may only move to tower $(2a-b) \mod q$.

3.3. **Algorithms to Solve the SF Puzzle.** Below we present three algorithms that correctly solve the SF Puzzle. The proofs are similar to those in [4] and [31].

3.3.1. *Recursive Algorithm.* The goal of the SF puzzle is to move all n disks from tower q to tower $q-1$. In the following algorithm, we generalize this to move the disks from any tower i to any other tower j. Therefore, the middle step will occur when $n-1$ disks are moved to tower a, where $(2a-0) \mod q = j$. This gives the following recursive algorithm where i and j are the source and destination tower repsectively and n the number of disks to be moved:

Recursive Algorithm for the Generalized Towers of Hanoi

```
PROCEDURE    Hanoi ( $i$, $j$, $n$ )
MID : $= (i+j)/2 \mod q$
        IF $n = 1$
                THEN move the top disk from tower $i$ to tower $j$
                ELSE Hanoi( $i$, $MID$, $n-1$)
                     move the top disk from tower $i$ to tower $j$
                     Hanoi( $MID$, $j$, $n-1$ )
```

Proposition 3.3. *The recursive algorithm HANOI correctly solves the SF Puzzle.*

We will define the following inductive hypothesis that we'll use in the proof of Proposition 3.3. This proof will be very similar to that of the standard Towers of Hanoi recursive algorithm.

Definition 3.4. *Define HYP(n) = HANOI(i, j, n) correctly moved the n disks from* $1,2,\ldots,n$ *from tower i to tower j.*

Proof. Note first that $i, j \in \{0,1,\ldots,q-1\}$.

Base Case: We will show that $HYP(1)$ is **TRUE**. From the definition, $HYP(1)$ says that $HANOI(i, j, 1)$ correctly moves the first (smallest) disk from tower i to tower j. In the algorithm the **IF** condition holds because $n = 1$ and therefore executes the **THEN** condition which moves the top disk from tower i to tower j. Observe that this indeed moves disk 1 from tower i to tower j. This move is *correct* as it satisfies all of the rules. After this move, *HANOI* runs out of moves and terminates. Thus, $HYP(1)$ is **TRUE**.

Inductive Step: We will show that $HYP(n-1)$ implies $HYP(n)$ for all $n > 1$. Assume that $HYP(n-1)$ is **TRUE**. Consider the algorithm *HANOI* with the inputs (i, j, n) where $n > 1$. Because the **IF** condition is **FALSE**, we move to the **ELSE** condition of the algorithm. The **ELSE** condition tells us to call $HANOI(i, [(i+j)2^{-1}] \mod q, n-1)$. Because we assumed that

$HYP(n-1)$ is **TRUE**, we know that $HANOI(i,[(i+j)2^{-1}] \mod q, n-1)$ *correctly* moves the $n-1$ disks from tower i to tower $[(i+j)2^{-1}] \mod q$.

The next instruction is to move the top disk from tower i to tower j. Notice that the top disk of tower i is disk n, the largest disk, which we will show *correctly* moves to tower j. We will do this by showing that the rules of the game are obeyed. Rule (1) is obeyed as only disk n is being moved. Rule (2) is obeyed because the disks smaller than disk n are placed on tower $[(i+j)2^{-1}] \mod q$. Rule (3) is obeyed as disks $1,2,\dots,q-1$ are stacked together on tower $[(i+j)2^{-1}] \mod q$. To show that rule (4) is obeyed, we first observe that all of the smaller disks are on tower $[(i+j)2^{-1}] \mod q$ and disk n is on tower i. Disk n may only *correctly* move to tower $[2(i+j)2^{n-1}-i] \mod q$ in order for Rule (4) to hold. We see that

$$[2[(i+j)2^{-1}]-i] \bmod q = [(i+j)-i] \bmod d = j \bmod q.$$

This proves that Rule (4) is obeyed since disk n can only move to tower j. Therefore, the top disk, disk n, moves from tower i to tower j *correctly*.

The next part of the algorithm calls $HANOI([(i+j)2^{-1}] \mod q, j, n-1)$, and as $HYP(n-1)$ is **TRUE**, $HANOI([(i+j)2^{-1}] \mod q, j, n-1)$ *correctly* moves disks $1,2,\dots,n-1$ from tower $(i+j)2^{-1}] \mod q$ to tower j obeying Rule (1). By $HYP(n-1)$, none of the disks from among $1,2,\dots,n-1$ is ever placed on top of a smaller disk from among $1,2,\dots,n-1$, and as n is larger than all of the disks $1,2,\dots,n-1$, Rule (2) is obeyed. Rules (3) and (4) hold because $HYP(n-1)$ is **TRUE** and the location of disk n does not affect this because it is the largest disk. Therefore, the algorithm $HANOI(i,j,n)$ with $n>1$ *correctly* moves all n disks from tower i to tower j. By induction, $HYP(n)$ is **TRUE** for all $n \geq 1$ and so Proposition 3.3 holds. □

3.3.2. *Iterative Algorithm.* When we call $HANOI(i,j,n)$, we move n disks from tower i to tower j where $i,j \in \{0,1,\dots,q-1\}$. Next, we observe that the iterative algorithm for $q \geq 3$. For this, we will prove that the smallest disk, disk 1, will always move at the same increment of towers.

Lemma 3.5. *The recursive algorithm, HANOI, moves disk 1 at the same increment for all $n \geq 1$, and this increment is solely a function of $(j-i) \mod q$.*

Proof. Base Case: Consider the case where $n=1$. For $HANOI(i,j,1)$ the **IF** condition is **TRUE**, so the algorithm executes the **THEN** condition and moves disk 1 from tower i to j. This is the only move and therefore disk 1 always moves at the same increment.

Inductive Step: Assume that disk 1 moves at the same increment in the algorithm $HANOI(i,j,n-1)$. We will show that this implies that disk 1 moves at the same increment for $HANOI(i,j,n)$. The algorithm $HANOI$ with the input (i,j,n) with $n>1$ first calls $HANOI(i,[(i+j)2^{-1}] \mod q, n-1)$. From out assumption, this call will always move disk 1 at the same increment. Call this increment I. The next step is to move the largest disk, disk n, which will not change I. We then call $HANOI([(i+j)2^{-1}] \mod q, j, n-1)$ which, by our assumption, will always move disk 1 at the same increment. Call this increment K.

Next we must show that $I=K$. When we call $HANOI(i,[(i+j)2^{-1}] \mod q, n-1)$, we are simply moving $n-1$ disks from i to $[(i+j)2^{-1}] \mod q$. In other words, the $n-1$ disks are moved at the increment of

$$[(i+j)2^{-1}] \bmod q - i = [(i+j)2^{-1} - (2)(2^{-1})i] \bmod q$$

$$= [2^{-1}(i+j-2i)] \bmod q = [2^{-1}(j-i)] \bmod q$$

Similarly, the next call $HANOI([(i+j)2^{-1}] \bmod q, j, n-1)$ moves the $n-1$ disks at the increment of

$$[j-(i+j)2^{-1}] \bmod q = [(2)(2^{-1})j-(i+j)2^{-1}] \bmod q$$

$$= [2^{-1}(2j-i+j)] \bmod q = [2^{-1}(j-i)] \bmod q.$$

Therefore, both calls move the $n-1$ disks at the same total increment. But, this is just a relabeling of the move in which the tower names are cyclically shifted. Hence $I = K$.

Since both calls always move disk 1 by the same increment we can say disk 1 moves at the same increment for $HANOI(i, j, n)$ and, by induction, we conclude disk 1 always moves at the same increment for all $n \geq 1$.

□

Lemma 3.5 will be helpful in creating an iterative algorithm for the SF Puzzle. For this, we must know that the increment in which disk 1 is moving. Call this increment I.

Iterative Algorithm for Generalized Towers of Hanoi

```
PROCEDURE
   Move smallest disk $(j)(1/2)^{n-1} \bmod q$ tower(s) clockwise
        WHILE a disk other than smallest is able to move, DO
                 Move that disk
                 Move smallest disk $(j)(1/2)^{n-1} \bmod q$ tower(s) clockwise
        ENDWHILE
```

Proposition 3.6. *The Iterative Algorithm for Generalized Towers of Hanoi correctly moves n disks from tower i to tower j.*

The proof of Proposition 3.6 is similar to the proof of the Counting Algorithm in the following section.

3.3.3. *Count Algorithm.* From the iterative algorithm it is clear that every other move involves moving disk 1. This will help define the counting algorithm that involves using a binary counter to determine which disk should be moved.

Counting Algorithm for Generalized Towers of Hanoi

```
PROCEDURE   TOWERS ( $n$ )
        T : = $0$ (Tower number computed modulo $q$ )
        BCount : = 0 (BCount has $n$ bits)
        P : = $(-1)(1/2)^{n-1} \mod q$
        Move disk 1 from T to T+P
        T : = T+P
        BCount : = BCount + 1
        WHILE BCount is not 11...1 (n 1s) DO
              IF Rightmost 0 in BCount is in position $b$
              THEN move disk $b$ from T + $(2^{b-2})(1/2)^{n-1} \mod q$ to
                   T - $(2^{b-2})(1/2)^{n-1}) \mod q$
              BCount : = BCount + 1
              Move disk 1 from T to T+P
              T : = T+P
              BCount : = BCount + 1
        ENDWHILE
```

In order to prove the upcoming proposition we will show when the **COUNT**= 2^k-1 the count algorithm has completed the same moves as HANOI$(0,[-(2^{k-2})(2^{-1})^{n-1}] \bmod d,k)$.

Lemma 3.7. *When decimal count* $= 2^k-1$*, that is binary* ***COUNT*** $= 00...01...1$ *with* $k1's$*, then the correct moves for HANOI*$(0,[-(2^{k-1})(2^{-1})^{n-1}] \bmod d,k)$ *have been completed by the count algorithm and disk* 1 *is on tower* $[-(2^{k-1})(2^{-1})^{n-1}] \bmod d$.

Proof. BASE CASE: If $k=1$, **COUNT** $= 0...01$, the single move T to $T+P$ has been completed, where $T=0$ and $T+P=[(-1)(2^{-1})^{n-1}] \bmod d$. Because$[-(2^{k-1})(2^{-1})^{n-1}] = [(-1)(2^{-1})^{n-1}]$ for when $k=1$ this completes the moves for HANOI $(0,[(-1)(2^{-1})^{n-1}] \bmod d,1)$. In addition, disk 1 is on tower $[(-1)(2^{-1})^{n-1}] \bmod d$. This agrees with our claim.

INDUCTIVE STEP: Observe that the **COUNT** can only equal the value 2^k-1 immediately before the **IF....RETURN** statement. Assume the moves for HANOI$(0,[-(2^{k-1})(2^{-1})^{n-1}] \bmod d,k)$ have been completed and disk 1 is on tower $[-(2^{k-1})(2^{-1})^{n-1}] \bmod d$. We want to show when **COUNT** $= 2^{k+1}-1$ the moves for HANOI$(0,[-(2^k)(2^{-1})^{n-1}] \bmod d,k+1)$ have been completed and disk 1 is on tower $[-(2^k)(2^{-1})^{n-1}] \bmod d$.

The next move would involve knowing where the rightmost 0 is within the **COUNT**. This is very simple since the rightmost 0 in the **COUNT** would be in position $k+1$. This would move disk $k+1$ from tower 0 to tower

$$-2[(2^{k-1})(2^{-1})^{n-1}] = [-(2^k)(2^{-1})^{n-1}].$$

Next **COUNT** will be incremented to 0...010...0, where there are $k0's$ after the 1. And when the **COUNT** $= (2^{k+1}-1$, the algorithm will have repeated the same sequence of moves as before since it only *sees* the rightmost information in **COUNT**, with the difference that T will have started with a different value. Therefore the next moves up until **COUNT** $= (2^{k+1}-1$ would have moved the

k disks from tower $[-(2^{k-1})(2^{-1})^{n-1}]$ mod d to tower $[-(2^k)(2^{-1})^{n-1}]$. At this point it is clear that we have moved $k+1$ disks from tower 0 to tower $[-(2^k)(2^{-1})^{n-1}]$ and disk 1 is on tower $[-(2^k)(2^{-1})^{n-1}]$. We conclude, by induction, when **COUNT** $= 2^k - 1$ the count algorithm has made the same moves as HANOI$(0, [-(2^{k-1})(2^{-1})^{n-1}] \bmod d, k)$. □

Proposition 3.8. *The count algorithm TOWERS(n) correctly moves n disks from tower* 0 *to tower j.*

Proof. Lemma 3.7 tells us that TOWERS(n) applies the same moves as HANOI when **COUNT** $= 2^k - 2$. Since HANOI has been proven to be correct we know TOWERS is also correct as long as the last **COUNT** is in the form $2^k - 1$, which it is since the total number of moves is $2^n - 1$. □

4. Spin-Out and the Dimension 2^m Puzzle

4.1. **Spin-Out.** Spin-Out is another puzzle that presents opportunities for building and designing solution algorithms. The goal of Spin-Out is to remove a rectangular block from its casing. In order to remove the block, we must first unlock the seven dials that are attached to the block and obstruct its movement within the casing. The initial configuration of the locks are all vertical, and to remove the block we must align them so that they are all horizontal. The traditional puzzle has seven locks or spinners. We will use the terms *locks* and *spinners* interchangeably. A spinner may rotate only if the spinner is positioned over the arc (divot) on the side of the casing.

Like the Towers of Hanoi, a configuration of the puzzle can be described using a string of characters, in this case, of 1s and 0s. When Spin-Out is oriented such that the divot is on the bottom, note that the rightmost spinner can always change its position on any move. Starting from this less restricted spinner on the right, we will label each spinner starting from 1. To describe the configuration, we will use 1 to mean that a spinner is oriented vertically (locked) and a 0 to mean that a spinner is oriented horizontally (unlocked). It turns out that it is never necessary to position a spinner down or to the right [30]. To label the configuration, the rightmost bit of the string will correspond to the rightmost spinner, the next bit will correspond to the spinner immediately to the left of that, and so on.

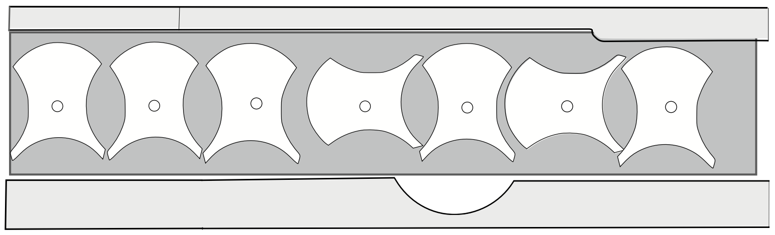


FIGURE 5. An example configuration for the Spin-Out puzzle with the label 1110101.

Remark 4.1. *The term* **orientation** *will be used to describe the state of a spinner. Here the possible states are horizontal and vertical.*

The restrictions and rules for the movement of the spinners are enforced by the physical make-up of the sliding case. Nonetheless, these rules can be stated formally.

(1) Spinner 1 (rightmost spinner) may change its orientation on any move.
(2) Conditions for Movement of Spinner j: Spinner j may change its orientation provided all of the following are true.
  - Spinner 1 through spinner $j-2$ are all horizontal.
  - Spinner $j-1$ is vertical.

Remark 4.2. *We use the superscript notation to denote a string (or part of a string) that is made up of consecutive repeated characters. For example, 1^n is a string of n 1s, that is, $1^n = \overbrace{11\ldots1}^{n\ 1s}$. As another example, 10^{k-1} is a string of length k, where the leading character is a 1, follow by $k-1$ 0s. That is, $10^{k-1} = 1\overbrace{00\ldots0}^{k-1\ 0s}$*

Definition 4.3. *A piece is **free** in Spin-Out if and only if it is the rightmost spinner or it is immediately to the left of the rightmost vertical spinner.*

Definition 4.4. *A move is **legal** in Spin-Out if the piece moving is free.*

Remark 4.5. *Note that as a vertical piece in Spin-Out corresponds to a 1 in the binary reflected Gray code, a bit in Gray code can be analogously called **free** if it is the lowest order bit or it is immediately to the left of the rightmost 1.*

Theorem 4.6. *The binary reflected Gray code represents the state space for the Spin-Out puzzle.*

The key idea in the following proof is that the axis of reflection in the binary reflected Gray code represents the point at which the n^{th} spinner in n-spinner Spin-Out is switched from vertical to horizontal. In order for that to happen, the piece immediately to its right, piece $n-1$, must be vertical, and all other pieces $1\ldots n-2$ must be horizontal. For any n, the first half of the binary reflected Gray code achieves this configuration, and the second half undoes the configuration, as it is simply the first half in reverse and all of the moves are invertible. The only difference is that, in the first half of the code, the highest-order bit is 0, and in the second half it is 1. Note that this is true for each reflection that is a sub-puzzle of the original puzzle, and we will use this recursive construction to generate the next reflection in the Gray code inductively.

Proof. We will prove this by induction on the number of spinners. Let H_i be the state space for a Spin-Out puzzle with i spinners.

Base Case: $n = 2$. We see that the binary reflected Gray code for $n = 2$ is 00, 01, 11, 10, which is indeed the correct sequence of moves to solve Spin-Out with 2 spinners. The rightmost spinner switches on every other move, and on the other moves, the spinner immediately to the left of the rightmost 1 spins.

Induction hypothesis: The state space for Spin-Out with $n-1$ spinners, H_{n-1}, is the binary reflected Gray Code with $n-1$ bits.

We will show: The state space for Spin-Out with n spinners, H_n is the binary reflected Gray code with n bits.

Let $n \geq 3$. The binary reflected Gray code on n bits may be represented as $H_n = 0H_{n-1}||1H^R_{n-1}$, where R indicates a reflection of the code. Observe that $0H_{n-1}$ is a subset of the state space for Spin-Out with n spinners as, by our inductive hypothesis, H_{n-1} is the state space for $n-1$ spinners and, although 0 is prepended throughout as the n^{th} bit, the n^{th} spinner is never spun. The fact that bit n is 0 therefore violates no conditions for movement and does not affect the legality of the moves. We then have the claim for the first half of the state space for n bits.

At the point of reflection, we change bit n from 0 to 1. This is legal because the $n-1$ bits are of the form $10\ldots00$, and therefore the conditions for movement hold and rule (2) is obeyed. Thus this move is in the state space for Spin-Out.

In the second half of the code, we see that $1H_{n-1}^R$ is also a subset of the state space for Spin-Out with n spinners because, by our inductive hypothesis, H_{n-1} is the state space for $n-1$ spinners and, although 1 is prepended through as the n^{th} bit, it is never changed from a 1 to a 0. Note that the reflection of H_{n-1} is unimportant as all moves in Spin-Out are invertible. Thus the claim holds for the second half of the Gray code.

Thus, the claim is true for the entire Gray code, and we conclude that H_n is indeed the binary reflected Gray code with n bits.

□

Theorem 4.6 tells us that we may find the configuration $11\ldots1$ in the binary reflected Gray code and follow each successive entry to the $00\ldots0$ configuration, and each move will be a legal move in Spin-Out.

4.1.1. *Algorithms to Solve Spin-Out.* We present two recursive algorithms for solving Spin-Out, the mutual recursion Unlock, and the nested recursion Solve. Let n be the number of spinners. The input configuration is 11..1 (n 1s), that is, all the spinners are locked. Unlock (n) is the procedure that unlocks all n spinners. To unlock the last spinner, the first $n-2$ spinners must all be unlocked first. Then, to unlock the second to last spinner, those first $n-2$ spinners must be re-lock so that the procedure Unlock can be called on again. To perform the re-locking, we call the procedure Lock (n), where n is the number of spinners with input configuration 00..0 (n 0s).

Mutual Recursion for Spin-Out

```
PROCEDURE    Unlock ( $n$ )
          IF $n > 0$
                  THEN Unlock ($n-2$ )
                       turn the n-th piece
                       Lock ($n-2$)
                       Unlock ($n-1$)


PROCEDURE    Lock ( $k$ )
          IF $k > 0$
                  THEN Lock ($k-1$ )
                       Unlock ($k-2$)
                       turn the k-th piece
                       Lock ($k-2$)
```

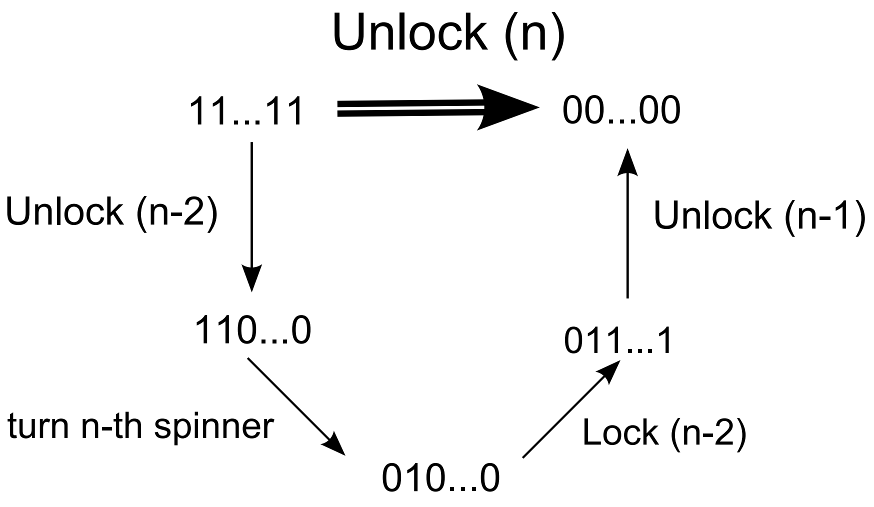


FIGURE 6. An illustration of the procedure Unlock.

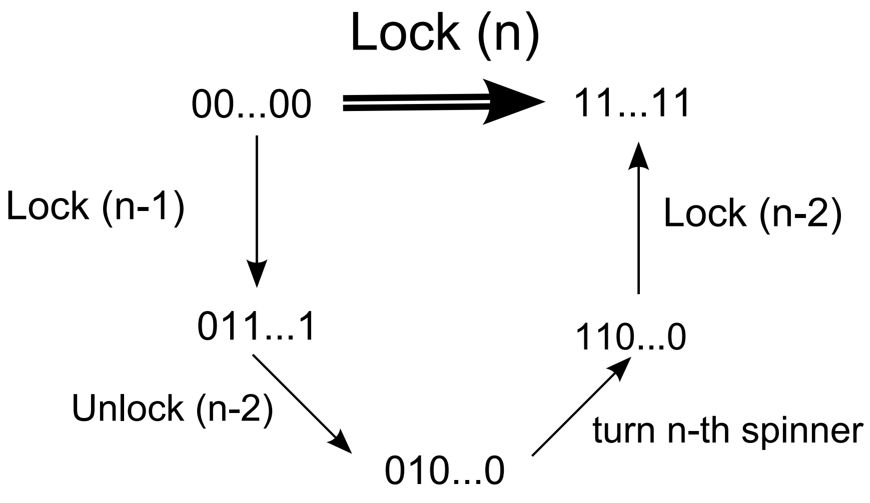


FIGURE 7. An illustration of the sub-procedure Lock, which is called by Unlock.

Theorem 4.7. *The mutual recursion Unlock(n) correctly solves the Spin-Out puzzle with n spinners.*

Proof. Base Case: Suppose $n = 1$. The **IF** condition is satisfied, and the procedure Unlock executes the **THEN** statement and calls Unlock(-1). Unlock(-1) does nothing because the **IF** condition is not fulfilled. Unlock(1) then turns the first spinner (and only spinner). Lock(-1) and Unlock(0) does nothing because the **IF** conditions are not satisfied. Similarly, the reader can verify that Lock(1) correctly locks an unlocked spinner.

Inductive Hypothesis: Assume that Unlock(n) correctly solves the Spin-Out puzzle with n or fewer spinners. Assume also that Lock(n) correctly locks all the spinners in the Spin-Out puzzle with n or fewer spinners. Suppose we call Unlock($n+1$) for the Spin-Out puzzle with $n+1$ spinners. The **IF** condition is satisfied, and the procedure executes the **THEN** statement and calls Unlock($n+1-2$), that is, Unlock($n-1$). By our induction hypothesis, this correctly unlocks the first $n-1$ spinners. Next, the procedure turns (unlocks) the $(n+1)^{st}$ spinner. Observe that this is the spinner to the left of the rightmost vertical spinner. Then the procedure calls Lock($n-1$). By our induction hypothesis, this sub-procedure must correctly lock the first $n-1$ spinners. Finally, the algorithm calls itself in the form Unlock(n). Again, by our induction hypothesis, this procedure correctly unlocks the first n spinners. Observe that the procedure has unlocked all $n+1$ spinners.

Suppose we call Lock($n+1$) for the Spin-Out puzzle with $n+1$ *unlocked* spinners. The **IF** condition is satisfied, and the procedure calls itself in the form Lock($n+1-1$), that is, Lock(n). By our induction hypotheses, this procedure correctly locks the first n spinners. Next, the procedure calls Unlock($n+1-2$), that is, Unlock($n-1$). Again, by our induction hypotheses, this procedure correctly unlocks the first $n-1$ spinners. Observe that at this point the spinner to the left of the rightmost vertical (locked) spinner is spinner $n+1$. The next step is turning spinner $n+1$, thus locking it. Finally, the algorithm calls Lock($n+1-2$), which by our induction hypotheses correctly locks the first $n-1$ spinners. Observe that the procedure has locked all $n+1$ spinners. □

Nested Recursion for Spin-Out

```
PROCEDURE   Solve ( $n$ )
        IF $n > 0$
                THEN Solve ( $n-2$ )
                     turn the $n$-th spinner
                     YSolve ($n-1$)


PROCEDURE   YSolve ( $k$ )
Comment:  takes the puzzle from $10^{k-1}$ to $0^k$ or from $0^k$ to $10^{k-1}$
        IF $k > 0$
                THEN YSolve($k-1$)
                     turn the $k$-th spinner
                     YSolve ($k-1$)
```

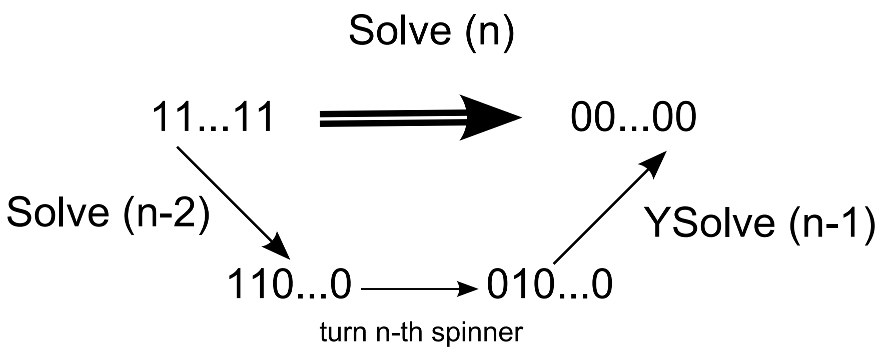


FIGURE 8. The figure above illustrates the main parts of the recursion “Solve” for Spin-Out.

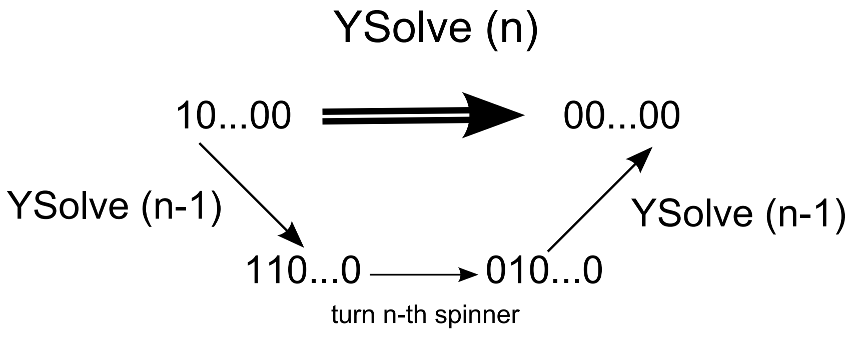


FIGURE 9. The sub-procedure “YSolve” that is called by “Solve.”

Theorem 4.8. *The nested recursion Solve(n) correctly solves the Spin-Out puzzle with n spinners.*

Proof. Proceed by induction. A proof is also provided by [30]. □

Remark 4.9. *There exist iterative and counting algorithms for Spin-Out. We will describe them in 4.2 since the same algorithms may also be used to solve traditional Spin-Out by letting $m = 1$.*

4.2. **The Dimension 2^m puzzle.** The Spin-Out puzzle can be extended to a family of puzzles called the Dimension 2^m puzzles. In the Dimension 2^m puzzle, each spinner is replaced by a piece consisting of a *stack* of m spinners for some $m \in \mathbb{N}$. Each spinner on the piece can take two positions, horizontal (0) or vertical (1). Hence, the total number of possible orientations that a piece can be in is 2^m. Note that we will use the terms *orientation* and *spin* interchangeably for describing the configuration of a single piece. For n pieces, there will be $(2^m)^n$ possible configurations that the whole puzzle can have. If we set $d = 2^m$, then this number is d^n.

We can associate to each orientation a natural number from 0 to $d-1$. Next we present the method for defining the orientations of the pieces as developed by Skubak and Stevenson [31].

(1) For a dimension $d = 2^m$ generalized Spin-Out puzzle, each puzzle piece will consist of m spinners stacked one on top of the other.
(2) To find the orientation t for $t \in \{0, \ldots, d-1\}$, first write t as a binary number. To set a piece to this orientation, let the rightmost bit (1's bit) represent the top spinner, and let the next bit from the right (2's bit) represent the second spinner from the top of the stack. Continuing in this manner, the leftmost bit will represent the spinner at the bottom of the stack. A 0 bit means that the spinner that the bit represents in the stack is horizontal, and a 1 bit means that the spinner is vertical.
(3) Now, for each $t \in \{0, \ldots, d-1\}$, there is associated to it a distinct orientation and a corresponding binary number.

Example 4.10. *Suppose $d = 8 = 2^3$. Then the number of spinners on each puzzle piece is $m = 3$. A piece in the $0 = 000_2$ orientation has all three spinners in the horizontal position (in fact, a piece in the 0 orientation will have all horizontal spinners). The $7 = 111_2$ orientation has all vertical spinners, and the $3 = 011_2$ orientation has a horizontal spinner on the bottom, and two vertical spinners above it.*

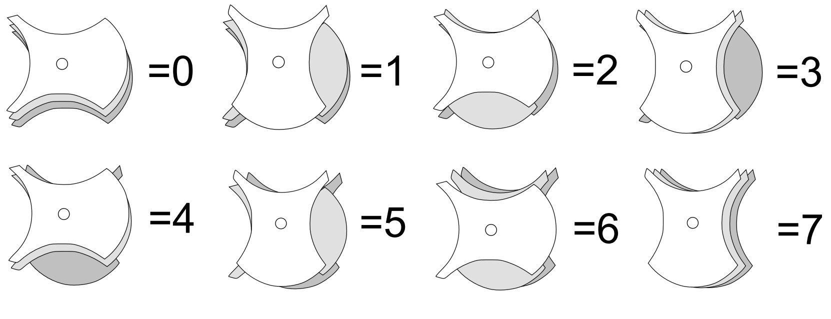


FIGURE 10. The orientation numbers for the Dimension 8 puzzle.

For a Dimension 2^m puzzle with n puzzle pieces, we number the pieces from right to left beginning at 1. Given a configuration of the puzzle, we can label it with a string of characters from

$\{0,\dots,d-1\}$, where each piece is represented by the number of the orientation it is in. The *convention* for the label will be that the rightmost character represents piece 1. For a piece j, we will use $f(j)$ to denote the orientation number of piece j.

Example 4.11. *A labeling of a configuration in the Dimension 8 puzzle with 4 pieces.*

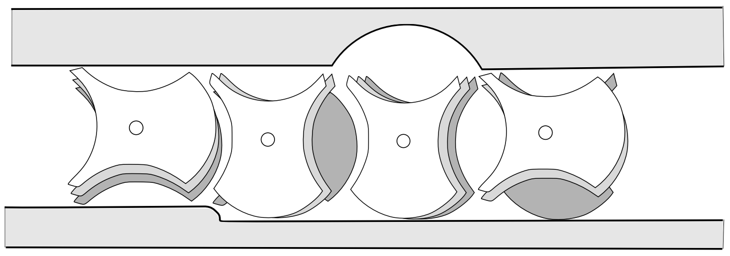


FIGURE 11. An example configuration for the Dimension 8 puzzle, labeled 0374.

Next we outline the rules for the Dimension 2^m puzzle.

(1) The first piece, piece 0, may always change its orientation to any other orientation on any given move.
(2) To spin at least one spinner of piece j, the orientations of piece 0 through piece $j-2$ must all be 0, and the orientation of piece $(j-1)$ must not be 0. In other words, $f(0)$ through $f(j-2)$ are all 0s, and $f(j-1) \neq 0$. If these conditions are met, then we must rotate as many spinners of piece j as possible; that is, any spinner on the stack that can switch between its horizontal and vertical position must do so.
(3) The goal of the puzzle is, given the initial configuration of all locked spinners $((2^m-1)(2^m-1)\dots(2^m-1))$, to move all the pieces to orientation 0.

4.3. **Algorithms to Solve the Dimension 2^m Puzzle.**

Remark 4.12. *While Spin-Out generalizes in an interesting way to any $2^m = d$ dimension, we note that the optimal solution (that which is both legal and uses the minimal number of moves) involves only the orientations $(d-1)$ and 0 for each spinner. That is, given a puzzle of the form $(d-1)^n$, we need only switch between $(d-1)$ and 0 for each piece. The puzzle has many other possible configurations, as seen in the previous examples, and it would be interesting to know how to get from any configuration to any other. Previous papers, most notably* [31] *and* [4], *have accomplished this for small m by describing the complete iterated graphs associated with these puzzles. Nevertheless, we are concerned with writing algorithms that correctly solve these puzzles, and we thus do not take into account these configurations. Throughout, we use the fact that there is an obvious bijection between puzzles of the form $(d-1)^n = 1^n$, and we will sometimes use 1 to represent $d-1$ for the analogous generalized puzzle. In the algorithms we will use $d-1$ and 1 interchangeably, and will often use 1 in examples for notational ease.*

4.3.1. *Recursive Algorithm.* For the Dimension 2^m puzzle with n pieces, the initial configuration is $(d-1)(d-1)\dots(d-1)$, that is, each piece is in the orientation $d-1$ (each piece has all vertical spinners). To solve the puzzle, we must change the configuration to 00..0, so that each piece will have all horizontal spinners. In the following recursive algorithm developed by Baun and Chauhan [4], rotate(i) means to rotate piece i from $d-1$ to 0 or from 0 to $d-1$, where $i \in \{1,\dots,n\}$. Remember, each piece is indexed from 1 to n from *right* to *left*.

Nested Recursion for Dimension 2^m

```
PROCEDURE   A ( $n$ )
Comment:  takes the puzzle from $(d-1)^n$ to $0^n$
          IF $n > 0$
                  THEN A( $n-2$ )
                       rotate piece n
                       C($n-1$)
END

PROCEDURE   C ( $k$ )
Comment:  takes puzzle from $(d-1)0^{k-1}$ to $0^k$ or from $0^k$ to $(d-1)0^{k-1}$
          IF $n > 0$
                  THEN C($k-1$ )
                       rotate piece k
                       C($k-1$)
END
```

We can also construct a mutual recursion for the Dimension 2^m puzzle that is analogous to the procedure “Unlock” by substituting $(d-1)$ for 1 into the mutual recursion for Spin-Out as discussed in Remark 4.12.

4.3.2. *Iterative Algorithm.* The following iterative algorithm is a modification of a similar one developed by Baun and Chauhan [4].

Iterative Algorithm for Dimension 2^m Puzzle

```
PROCEDURE
   IF n is odd
        THEN rotate piece 1 from $d-1$ to 0
        ELSE rotate piece 2 from $d-1$ to 0
             rotate piece 1 from $d-1$ to 0
   WHILE a piece other than piece 1 can rotate DO
             Rotate that piece
             Rotate piece 1
   ENDWHILE
```

Proposition 4.13. *The iterative algorithm correctly solves the Dimension 2^m puzzle.*

Proof. An analogous proof is provided by Pruhs [30]. □

4.3.3. *Counting Algorithm.* Next we present the counting algorithm for the Dimension 2^m puzzle.

Count Algorithm for Generalized Spin-Out (Dimension 2^m)

```
PROCEDURE
GrayLabel := All 1s (n bits)
BCount := $\lceil \frac{2}{3}(2^n - 1) \rceil$ (n bits)
IF $n = 1$
     THEN switch position 1 in GrayLabel from $d-1$ to 0
IF $n = 2$
     THEN switch position 2 in GrayLabel from $d-1$ to 0
          switch position 1 from $d-1$ to 0
ELSE
     WHILE rightmost 0 is in position b in BCount DO
               BCount:= BCount - 1
               Switch position b in GrayLabel from 0 to $d-1$ or $d-1$ to 0
               IF GrayLabel = all 0s THEN return
     ENDWHILE
```

We begin with a table for $n = 5$ to illustrate the connection between Spin-Out and the binary reflected Gray code. This table will take Spin-Out with 5 spinners from all locked to all unlocked by following the Gray code column from the Gray code 11111 (binary number 10101) and moving upwards.

Binary	Gray	Binary	Gray
00000	00000	10000	11000
00001	00001	10001	11001
00010	00011	10010	11011
00011	00010	10011	11010
00100	00110	10100	11110
00101	00111	10101	11111
00110	00101	10110	11101
00111	00100	10111	11100
01000	01100	11000	10100
01001	01101	11001	10101
01010	01111	11010	10111
01011	01110	11011	10110
01100	01101	11100	10010
01101	01011	11101	10011
01110	01001	11110	10001
01111	01000	11111	10000

To prove that the counting algorithm is correct, we first make observations about the relationship between binary numbers and the binary reflected Gray code.

Lemma 4.14. *For any n, $\lceil \frac{2}{3}(2^n-1)\rceil$ has a n-bit binary representation with alternating zeroes and ones; that is, it is of the form $\ldots 010101\ldots$ where $n-1$ is the position of the leftmost 1.*

Proof. Consider the number $\ldots 010101\ldots$, and let $n-1$ be the position of the leftmost 1.

Case 1: The number is even. ($\ldots 101010$)

Then the final bit will be a 0. We may represent this number in base 4 as $\ldots 222$, with k twos. Note that $n = 2k$. We may rewrite this further as

$$\sum_{i=0}^{k-1} 2\centerdot 4^i = 2\sum_{i=0}^{k-1} 4^i = 2\frac{4^k-1}{4-1} = \frac{2}{3}(4^k-1) = \frac{2}{3}(2^{2k}-1) = \frac{2}{3}(2^n-1)$$

as $n = 2k$ in this case. Thus $\ldots 101010 = \frac{2}{3}(2^n-1)$ where the leftmost 1 is in the $(n-1)^{st}$ bit.

Case 2: The number is odd. ($\ldots 010101$)

Then the final bit will be a 1. We may represent this number in base 4 as $\ldots 111$, with k ones. Note that $n = 2k-1$, and so $n+1 = 2k$. We may rewrite this further as

$$\sum_{i=0}^{k-1} 4^i = \frac{4^k-1}{4-1} = \frac{2^{2k}-1}{3} = \frac{2^{n+1}-1}{3} = \frac{2\centerdot 2^n-1}{3} =$$

$$\frac{2}{3}(2^n-\frac{1}{2}) = \frac{2}{3}(2^n-1+\frac{1}{2}) = \frac{2}{3}(2^n-1)+\frac{1}{3} = \lceil\frac{2}{3}(2^n-1)\rceil$$

as $n+1=2k$ in this case and $\lceil \frac{2}{3}(2^n-1) \rceil$ must be an integer.

Thus, in either case, $\lceil \frac{2}{3}(2^n-1) \rceil$ is equivalent to $\ldots 01010 \ldots$ where $n-1$ is the position of the leftmost 1.

□

Lemma 4.15. *The algorithm above gives a sequence of legal moves in the Spin-Out Puzzle.*

Proof. Let B^R be a binary number. We will use R as the (presumably decimal) index of the binary numbers and their corresponding Gray code values, G^R. We will use j as the index of bits, and all numbers will have n bits with the rightmost (lowest-order) bit labeled as the first bit. The indices will run from lower to higher order, so index $j+1$ is to the left of index j. The highest-order index will be bit n.
Case 1: B^R ends in a 0. Then piece 1 will change in Spin-Out because the first binary bit, b_1, is 0. This is a legal move.
Case 2: B^R ends in a 1. Let j be the position of the rightmost 1 in G^R. Then positions 1 through $j-1$ in G^R are 0. By the binary-Gray conversion formulae, we see that positions 1 through $j-1$ in B^R are the same as the complement of b_{j+1}; we denote this $\overline{b_{j+1}}$. As we assumed that B^R ends in a 1, $1=b_1$ through $b_{j-1}=\overline{b_{j+1}}$. Therefore, $b_{j+1}=0$. As bits b_1 through b_{j-1} are 1, and $b_{j+1}+g_j=0+1=1=b_j$, bits b_1 through b_j are in fact 0, making b_{j+1} the rightmost 0 in the binary number. The algorithm tells us to switch the corresponding position $j+1$ in Gray count, which is by assumption the position immediately to the left of the rightmost 1. This is indeed a legal move.
Thus, in either case, the algorithm produces only legal moves in Spin-Out.

□

Theorem 4.16. *The Count algorithm correctly solves the Spin-Out puzzle.*

Proof. We first observe that, by the conversion formulae given above, a GrayLabel with the form $0\ldots01\ldots1$ occurs if and only if the BCount is in the form $\ldots010101\ldots$ with a potentially trivial number of zeroes at the high-order end and terminating at the low-order end. Call the position of the leftmost 1 in the GrayLabel position n. This represents a n-spinner puzzle in its starting position. By Lemma 4.14 above, note that this binary number corresponds to $\lceil \frac{2}{3}(2^n-1) \rceil$ in decimal representation. This is known to be the number of moves in the minimal solution to Spin-Out with n spinners. By Lemma 4.15, the Gray code gives a legal sequence of moves. By the conversion formula for the binary reflected Gray Code, we see that counting in binary numbers (which is precisely what the count algorithm does) generates exactly this Gray code by Theorem 2.11. Further, the algorithm terminates after $\lceil \frac{2}{3}(2^n-1) \rceil$ steps. On the $(\lceil \frac{2}{3}(2^n-1) \rceil - 1)^{th}$ step, the BCount will read $\ldots 0001$, and therefore all of the spinners will be in position 0, with the exception of spinner 1. The final move will therefore move the BCount to 0 and the last spinner from vertical to horizontal, thus solving the Spin-Out puzzle. As it has solved the puzzle in the minimum number of moves, and such a solution is known to be unique, we may conclude that the Count algorithm correctly solves the Spin-Out puzzle.

□

5. The Combination Puzzle

5.1. **Introduction and Rules.** The Combination puzzle (or Product puzzle) is created by stitching together the rules for the SF puzzle and the rules for the Dimension 2^m. We will first describe its set-up and the goal of the game. Like the Towers of Hanoi, the Combination puzzle will be played with n pieces and q towers where q is odd. Now, we can imagine that each piece of this puzzle is made up of m spinners. Each spinner on the piece can take one of two orientations, either horizontal (unlock) or vertical (lock). Hence, each piece has a total of 2^m possible orientations or spins. Note that we will use the terms *orientation* and *spin* interchangeably. Also note the change in terminology from *disks* in Towers of Hanoi (which have no orientation or spin) to *pieces* in the Combination puzzle. The notion of spin is inherited from the Spin-Out puzzle. In addition to its spin, each piece can reside on 1 of q towers. These two attributes define the *total orientation* of the piece.

Definition 5.1. *The* ***total orientation*** *of a piece is an ordered pair of integers (t_j, s_j) where the first integer designates the tower that the piece is on and the second integer represents the spin of the piece. We can represent the total orientation of piece j by the integer given by $f(j) = t_j \cdot 2^m + s_j$.*

This definition gives $q \cdot 2^m$ possible total orientations in all because $t_j \in \{0,\dots,q-1\}$ and $s_j \in \{0,\dots,2^m-1\}$ where the numbering for the spin is defined in the same manner as in the generalized Spin-Out puzzle. Finally, we also inherit from the Towers of Hanoi the notion of *size ordering*. The *convention* will be that the first piece (piece 1) is the smallest, the second piece (piece 2) is the next smallest, and so on up to the n^{th} piece, which is the largest.

The goal of the puzzle is simply a combination of the goals for the two individual puzzles. Suppose the pieces are stacked by size on some initial tower i with piece $n-1$ at the bottom. Furthermore, suppose the spin of each piece is 1. We are allowed to move (change the total orientation of) one piece at a time. The objective will be to move all the pieces and stack them by size onto some target tower l and in the process change all their spins to 2^m-1.

Definition 5.2. *A* ***movement*** *is defined to be a change in total orientation of a piece. This entails either a change in both tower and spin, or a change in one and not the other.*

Remark 5.3. *On any given move, only the first piece (piece 1) may change its total orientation to any other total orientation. All other pieces must change both their tower and spin according to the* total orientation change function. *It is possible that these change functions (one for the tower and one for the spin) may not have any effect.*

We emphasize that this new puzzle may not have a physical embodiment and is at best an abstraction played with pen and paper (and some imagination). The goal of the game is to use a newly defined set of rules to deduce the sequence of allowable configurations needed to go from the initial configuration to the final configuration. The important question is how do we decide what should constitute the rules and legality for movement in this combination puzzle? In what sense should the rules for Towers and Spin-Out be combined? It seems reasonable to stipulate that if a movement in the Combination puzzle is illegal from a purely Spin-Out (or Towers of Hanoi) standpoint, it should be restricted in the combined setting. This suggests that a movement in the Combination puzzle can be viewed from two different perspectives simultaneously. However, the suggestion above admits much room for ambiguity because a movement may be *illegal* from a

Spin-Out perspective but may be *legal* from a Towers of Hanoi perspective or vice versa. Instead, the right idea to keep in mind is that a piece j in the Combination puzzle can change its total orientation if the *total orientation change function* that governs the move agrees with the allowable configurations for moving from *both* the Towers configuration and the Spin-Out configuration. Clearly then, if the movement is illegal from *both* the Towers and Spin-Out configurations, it is illegal in the Combination puzzle.

Definition 5.4. *If a piece j in the Combination puzzle can move, then its movement is controlled by the* **total orientation change function** *which says that the tower of j must change to* $(2 \cdot t_{j-1} - t_j \mod q)$ *and its spin must change to* $s_{j-1} \oplus s_j$.

We are now in position to outline in detail the rules for the Combination puzzle.

(1) The **First Piece Rule**: Piece 1 can always change its total orientation into any other.
(2) The **Second Piece Rule** : If $j = 2$, then the tower of piece 2 may change to $2 \cdot t_1 - t_2 \mod q$ and its spin changes to $s_1 \oplus s_2$. If $t_1 = t_2$, the second piece may only change its spin, and the spin does not change if $s_1 = 0$.
(3) **Conditions for Movement of Piece** j: Piece j, for $j \neq 1, 2$, may move if *all* of the following are true
    (a) Piece 1 through piece $j-2$ are all on the same tower and their spins are all 0. If we denote their tower number by T, then this condition says $T = t_1 = t_2 = \cdots = t_{j-2}$, and $0 = s_1 = s_2 = \cdots = s_{j-2}$.
    (b) Piece $j-1$ is on tower T; that is, $t_{j-1} = t_1 = t_2 = \cdots = t_{j-2}$.
    (c) If piece j is on tower T, then we can change its spin (s_j) provided $s_{j-1} \neq 0$. If $t_j = T$, condition (1) and (2) implies that all the smaller pieces are actually stacked on top of piece j, hence we would not be able to change towers, and this agrees with the tower change function because $2 \cdot t_{j-1} - t_j \mod q = 2 \cdot T - T = T$.

Spin-Out has a greater degree of influence than does Towers of Hanoi in the combined setting. In some sense this is to be expected, but it is an important subtlety. Condition (a) in the conditions for movement of piece j is derived from Spin-Out while condition (b) is derived from Towers. Condition (a) ensures we have the correct spin configuration and condition (b) ensures all the smaller pieces are stacked elsewhere away from the piece we wish to move. Condition (c) is where Spin-Out overtakes Towers. Condition (c) says that if all the smaller pieces are actually stacked on top of the piece we wish to move, then the spin of the previous piece must not be 0. In the towers setting, this piece can't change towers, but that is allowable because the change function will always return the same tower in that case. So the only possibility is that the spin changes, and for this to happen we have the condition made in (c) combining with condition (a) to change the spin. So in order to move a piece, it is necessary that we are able to reach it in the Spin-Out configuration, which is essentially what condition (a) accounts for. But the surprising subtlety is that, in order to change the spin of a piece, we do not have to remove all the pieces smaller than it off to a different tower. The same is not true for the reverse. That is, we cannot change the tower of a piece if the puzzle does not have the correct spin configuration, even if the change function leaves the spin unaffected.

5.2. **Algorithms to Solve the Combination Puzzle.**

5.2.1. *Recursive Algorithm.* Before presenting the recursive algorithm for the Combination puzzle, we will begin by analyzing the skeleton of this master procedure and the major intermediate tasks that will be needed. For the sake of clarity, we will use as a guide the dimension 6 puzzle played with three towers and $n = 5$ pieces. Each piece has only two possible spins, either vertical (lock) or horizontal (unlock). The initial set-up has all the pieces stacked on tower 0 with the spin configuration at 11111; that is, they are all "locked." The objective will be to move all these pieces onto tower 2, and in the process change each of their spins to 0.

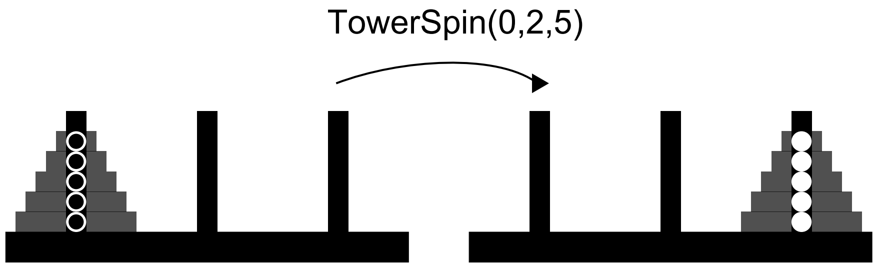


FIGURE 12. The left configuration is the input configuration, and the right configuration is the desired end configuration. Here, the procedure TowerSpin takes the 5 locked pieces from tower 0 to tower 2 and in the process unlocks all the pieces. The *convention* is that a centered white circle represents unlocked (0), and one with a centered black circle represents locked (1).

Let TowerSpin(i, j, n) be that master algorithm that takes us from the initial configuration to the desired final configuration. More importantly, TowerSpin accomplishes its job all within the legal restrictions of the game. The inputs are the initial tower i, the target tower j, and the number of pieces n. It will be implicit that this algorithm only deals with pieces in the initial spin configuration $11\dots1$ and will transform it into $00\dots0$. Now consider the configuration just before piece n (largest piece) is taken from $(i,1)$ to $(j,0)$. The configuration represents the bottleneck stage. If instead we choose to move piece n to the target tower and then change its spin later, we will be adding an extra move to the mimimum solution sequence, which is not allowed. The first $n-1$ pieces must be moved to some intermediate tower different from j, and the spin configuration on that tower must be 10^{n-2}.

The first major task will be to arrive at the stage above where we can move piece n in the desired fashion. Denote that intermediate stage by C. To obtain configuration C, we first need to move piece $n-1$ to a middle tower without changing its spin. Before that can happen, we need piece 1 through piece $n-2$ to be stacked elsewhere on some other tower, and the spin configuration of the first $n-2$ pieces on that other tower must be 0^{n-2}. Once we reach that configuration, say D, we will be in position to take piece $n-1$ from $(i,1)$ to $(k,1)$.

Observe that to reach configuration D, we call TowerSpin$(i, j, n-2)$. Finally, to arrive at C, we must move the first $n-2$ pieces back on top of piece $n-1$ such that the spins of those $n-2$ pieces

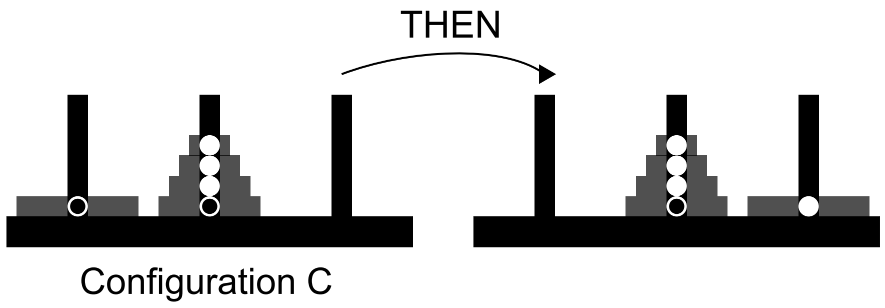


FIGURE 13. The left configuration represents the configuration C immediately before the last piece can be moved from tower 0 to tower 2 and have its spin changed from 1 to 0.

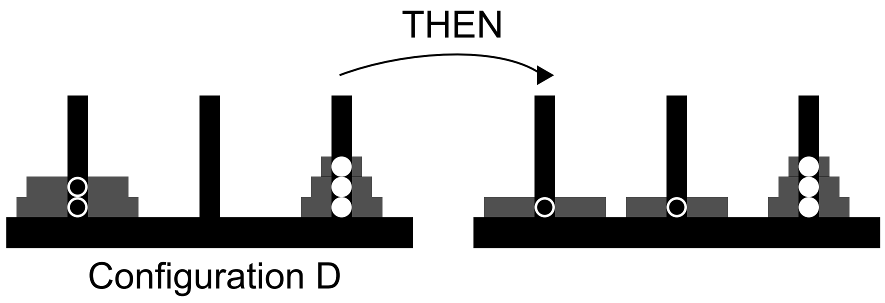


FIGURE 14. The left configuration represents the configuration D, and the auxiliary tower k is tower 1. The right right configuration represents the next move after D, where piece 4 (piece $n-1$) has been taken from tower 0 to tower 1 without any change in spin. Observe that the configuration D is obtained by calling TowerSpin$(0,2,3)$.

remain unchanged (stays all 0s). To accomplish this, all that is required is changing towers. The sub-procedure that will be called on will be similar to a (modified) Hanoi procedure.

Now we can consider the second half of the algorithm TowerSpin. The first $n-1$ pieces are stacked on some middle tower k, and the spin configuration on that tower is 10^{n-2}. Furthermore, piece n is now on tower j and with spin 0. Let F be the sub-procedure that completes the second half of the algorithm TowerSpin.

"Flip"(i,j,m), or $F(i,j,m)$ for short, moves m pieces from tower i to tower j and changes the spin of the largest piece (piece m). Let t represent the spin of piece m, where $t \in \{0,1\}$. Define $\bar{t} = t+1 \mod 2$. We have as the initial set up m pieces stacked on tower i with spin configuration $t\ 0^{m-1}$. The procedure F will move them onto tower j and take the spin into the configuration $\bar{t}\ 0^{m-1}$; that is, on the final tower, only the spin of the largest piece will change while the spins of all the smaller pieces on top of it will remain 0s. Consider the configuration before piece m is

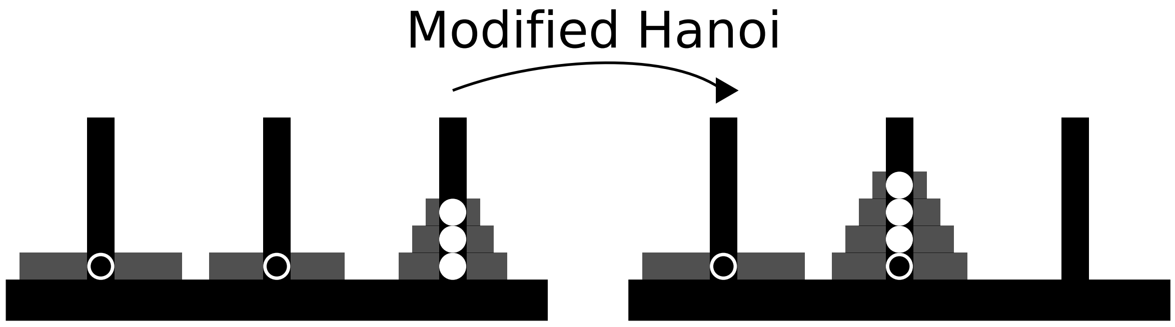


FIGURE 15. The left configuration is the configuration immediately after D, and the right configuration is the configuration after calling Modified Hanoi. Here Modified Hanoi acts on the 3 unlocked pieces on tower 2 and transfers them onto the middle tower 1 without locking any of the pieces.

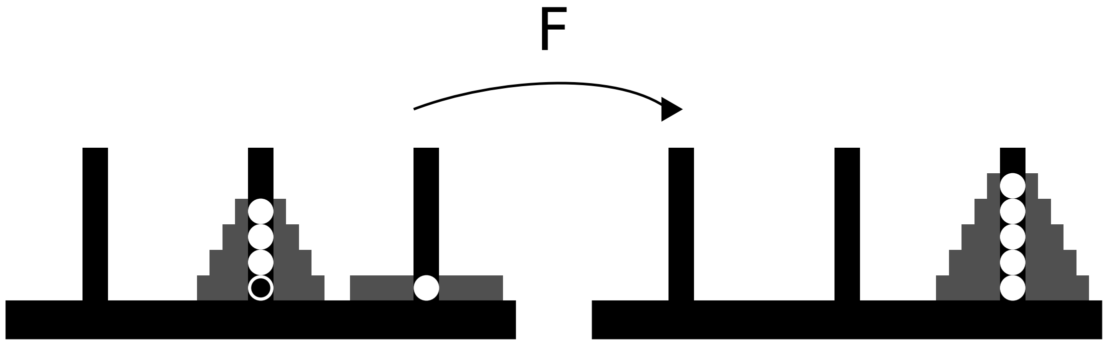


FIGURE 16. The left configuration results from the move made after Modified Hanoi has been completed. It is the next configuration after configuration C. The sub-procedure F completes the rest of the algorithm TowerSpin.

taken from (i,t) to $(j,\bar{t})$. The first $m-1$ pieces must be stacked elsewhere on some middle tower k. Furthermore, the spin configuration of those $m-1$ pieces on tower k must be 10^{m-2}.

Thus, the intermediate step in F is to call itself for one piece fewer. Once we move piece m, the last thing to do is to take the $m-1$ pieces from tower k and move them on tower j and change the spin of piece $m-1$. Observe that the procedure F does exactly this.

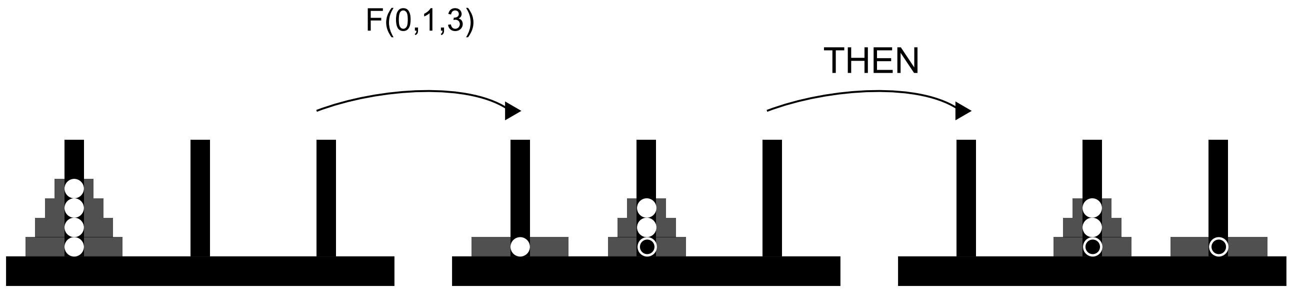


FIGURE 17. The leftmost configuration is the input configuration for the procedure F$(0,2,4)$. To correctly move piece 4, F first calls itself in the form F$(0,1,3)$.

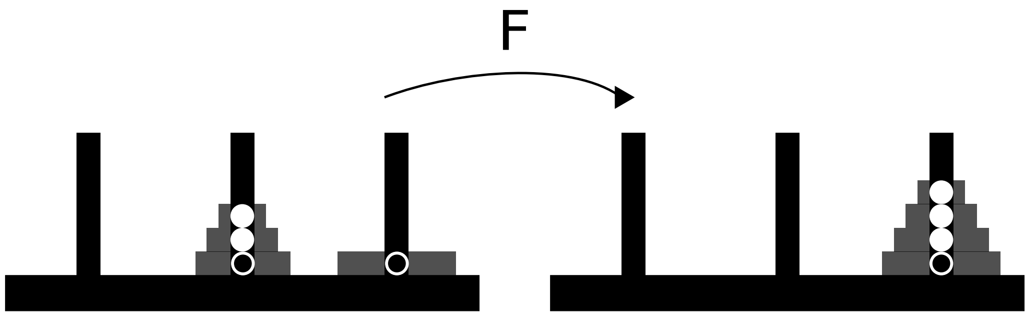


FIGURE 18. To complete the algorithm F, the procedure calls itself in the form F$(1,2,3)$.

Recursive Algorithm for Combination Puzzle

```
PROCEDURE   TowerSpin( $i$, $j$, $n$ )
        IF $n = 1$
              THEN Move the top piece from tower $i$ to tower $j$
                   Change its spin (rotate)
        IF $n = 2$
              THEN Move piece 1
                         from tower $i$ to tower $(i+j)/2 \mod q$
                   Move piece 2 from tower $i$ to tower $j$
                   Change its spin to $s_1 \oplus s_2$
                   Move piece 1
                         from tower $(i+j)/2 \mod q$ to tower $j$
                   Change its spin (rotate)
              ELSE TowerSpin( $i$, $j$, $n-2$ )
                   Move the top piece $k$
                         from tower $i$ to tower $(i+j)/2 \mod q$
                   Change its spin to $s_{k-1} \oplus s_k$
                   Modified Hanoi( $j$, $(i+j)/2 \mod q$, $n-2$ )
                   Move the top piece $t$ from tower $i$ to tower $j$
                   Change its spin to $s_{t-1} \oplus s_t$
                   F( $(i+j)/2 \mod q$, $j$, $n-1$ )


PROCEDURE   Modified Hanoi( $i$, $j$, $n$ )
        IF $n = 1$
              THEN Move the top piece from tower $i$ to tower $j$
              ELSE Modified Hanoi($i$, $(i+j)/2 \mod q$, $n-1$ )
                   Move the top piece $k$ from tower $i$ to tower $j$
                   Change its spin to $s_{k-1} \oplus s_k$
                   Modified Hanoi( $(i+j)/2 \mod q$, $j$, $n-1$ )


PROCEDURE   F( $i$, $j$, $n$ )
        IF $n = 1$
              THEN Move the top piece from tower $i$ to tower $j$
                   Change its spin (rotate)
              ELSE F( $i$, $(i+j)/2 \mod q$, $n-1$ )
                   Move the top piece $k$ from tower $i$ to tower $j$
                   Change its spin to $s_{k-1} \oplus s_k$
                   F( $(i+j)/2 \mod q$, $j$, $n-1$ )
```

Lemma 5.5. *The recursive procedure Modified Hanoi(i, j, n) correctly moves n pieces with initial spin configuration* 0^n *from tower i to tower j without changing the spin configuration, i.e, the spin of each piece remains 0 on tower j.*

Proof. Base Case: If $n = 1$, then the procedure moves the top piece from tower i to tower j. Because there is no instruction to rotate, the spin does not change and remains 0.
Induction Hypothesis: Assume that Modified Hanoi(i, j, $n-1$) correctly moves $n-1$ pieces from tower i to tower j without changing the initial spin configuration.

Suppose that $n > 1$. The procedure then executes the ELSE part of the "IF" condition and calls Modified Hanoi(i, $(i+j)/2 \mod q$, $n-1$). By our induction hypothesis, this procedure correctly moves the first $n-1$ pieces from tower i to some auxiliary tower $k = (i+j)/2 \mod q$ without changing the spin. Since the input spin configuration is all 0s, they end in all 0s. Next we move the top piece (piece n) from tower i to tower j. Since all the smaller pieces are stacked on some middle tower k, we know that we are not placing piece n on top of a smaller piece. Furthermore, the spin of piece n remains 0 because the spins of piece 1 through piece $n-1$ are all 0s on tower k. Finally, the procedure calls Modified Hanoi($(i+j)/2 \mod q$, j, $n-1$). By our induction hypothesis this procedure correctly stacks the $n-1$ pieces from the middle tower k onto the target tower j without changing the spin configuration. Since piece n was the largest piece, we know that we are never placing a larger piece on top of a smaller piece. Observe that the algorithm has achieved the desired final configuration. □

Lemma 5.6. *Let t represent the spin of the largest piece, where* $t \in \{0, d-1\}$*. If* $t = 0$*, then define* $\bar{t} = d-1$*. If* $t = d-1$*, then define* $\bar{t} = 0$*. The procedure F(i, j, n) correctly moves n pieces with spin configuration t* 0^{n-1} *from tower i onto tower j and into* $\bar{t}$ 0^{n-1}.

Remark 5.7. *For* $n = 1$*, we interpret* 0^0 *to mean that no pieces are horizontal (unlocked).*

Proof. Base Case 1: If $n = 1$, then F moves the top piece from tower i to tower j. Since we only have one piece, the spin of this piece is initially t. Hence, after rotation the spin changes to $\bar{t}$.
Induction Hypothesis: Assume that F(i, j, $n-1$) correctly moves $n-1$ pieces with spin configuration t 0^{n-2} from tower i to tower j and to the spin configuration $\bar{t}$ 0^{n-2}.

Let $n > 1$. The procedure F executes the ELSE part of the statement and calls itself in the form F(i, $(i+j)/2 \mod q$, $n-1$). By our induction hypothesis, F correctly moves $n-1$ pieces from tower i to the auxiliary tower $k = (i+j)/2 \mod q$ and changes the spin of piece $n-1$. Since the spin of piece $n-1$ was 0, it changes to $d-1$. Next, F moves the top piece (piece n) from tower i to tower j. Since all the smaller pieces are on tower k, we know that this piece will not be placed on top of a smaller piece. Furthermore, since the spin of piece $n-1$ is $d-1$, the spin of piece n is changed from t to $\bar{t}$. Finally, F calls F($(i+j)/2 \mod q$, j, $n-1$), which by our induction hypothesis correctly stacks the $n-1$ pieces from tower k onto tower j and changes the spin of piece $n-1$ from $d-1$ to 0. Observe that F has achieved the desired end configuration.

□

Lemma 5.8. *For n pieces, the minimum number of moves needed for F (Flip) to solve its initial input problem is* $2^n - 1$.

Proof. The algorithm F yields the following recurrence relation which can be solved:

$$\tilde{F}(n) = 2\tilde{F}(n-1) + 1 \text{ with } \tilde{F}(1) = 0$$

to give $\tilde{F}(n) = 2^n - 1$. □

Theorem 5.9. *The recursive procedure TowerSpin(i, j, n) correctly solves the combination puzzle.*

Proof. Base Case: If $n = 1$ or $n = 2$, then observe that the procedure correctly takes the puzzle from the initial configuration to the final configuration.
Induction Hypothesis: Assume that the procedure works correctly for $n-2$ pieces. That is, TowerSpin(i, j, $n-2$) correctly moves $n-2$ pieces with spin $(d-1)^{n-2}$ from tower i to the target tower j and changes the spin of each piece to 0.

Suppose $n > 2$. Then the procedure executes the ELSE statement and calls TowerSpin($i, j, n-2$). By our induction hypothesis, this procedure correctly moves the first $n-2$ pieces from tower i to the target tower j, and takes the spin from $(d-1)^{n-2}$ to 0^{n-2}. Next the procedure moves the top piece (piece n) from tower i to the middle tower $k = (i+j)/2 \mod q$. Now, since the spin of piece 1 through piece $n-2$ are all 0s, the spin of piece $n-1$ does not change, that is, it remains $d-1$. The algorithm then calls Modified Hanoi(j, $(i+j)/2 \mod q$, $n-2$), which correctly moves the $n-2$ pieces from tower j onto tower k. Since piece $n-1$ is bigger than piece 1 through piece $(n-2)$, we know that we are not stacking those $n-2$ pieces from tower j on top of a smaller piece. Furthermore, once those $n-2$ pieces are stacked on tower k, their spins will be unchanged (all 0s). Next, the procedure moves the top piece (piece n) from tower i to tower j. Since the spin of piece $(n-1)$ is $(d-1)$, the spin of piece n is changed from $(d-1)$ to 0. Finally, TowerSpin calls F($(i+j)/2 \mod q$, j, $n-1$). The sub-procedure F correctly moves the $n-1$ pieces from tower k onto the target tower j, and changes the spin configuration from $(d-1)0^{n-2}$ to 0^{n-1}. Observe that we have achieved the desired final configuration. □

Lemma 5.10. *For n pieces, the minimum number of moves needed to solve the Combination puzzle is* $2^n - 1$.

The proof of Lemma 5.10 is similar to the proof of the optimal solution of the generalized Towers of Hanoi. It makes use of the fact that the starting and ending configurations are corner vertices in the complete iterated graph K_q^n and the minimum distance between two corner vertices in such a graph is $2^n - 1$.

5.2.2. *Iterative Algorithm.* Next we present the iterative algorithm.

Iterative Algorithm for Combination Puzzle

```
BEGIN
   TestString : = all 0s ( $n$ bits)
   Sum : = $0$
   STEP : = $(j)(1/2)^{n-1} \mod q$
   Move the smallest piece STEP towers clockwise
   TestRotate($1$)
        WHILE a piece $i$ other than the smallest is able to change towers DO
                  change the tower of piece $i$
                  TestRotate($i$)
                  Move the smallest piece STEP towers clockwise
                  TestRotate($1$)
        ENDWHILE
END

PROCEDURE    TestRotate ( $i$ )
          IF TestString[i] = 0
                  THEN TestString[i] : = 1
                       Sum : = Sum + 1 mod 2
          IF Sum = n mod 2
                  THEN rotate piece i
END
```

Conjecture 5.11. *The iterative algorithm correctly solves the Combination puzzle.*

5.2.3. *Count Algorithm.* Finally, we give the count algorithm.

Count Algorithm for Combination Puzzle

```
BCount : = all 0's ($n$ bits)
S : = all 1s ($n$ bits)
T : = 0
Z : = $n \mod 2$ (Z $\in \{0,1\}$ )
P : = $(-1)(1/2)^{n-1} \mod q$
IF Z = 1 THEN Spin-Out : = ON ELSE Spin-Out : = OFF
Move piece 0 from T to T+P
IF Spin-Out = ON
    THEN switch 0-th bit in S from 0 to 1 or 1 to 0
BCount : = BCount + 1
WHILE BCount does not equal 11...1 ( $n$ 1s) DO
    IF BCount = 0...01...1
            THEN switch state of Spin-Out
    IF Rightmost 0 is in position $b$ in BCount
            THEN Move piece $b$
                from T - $(2^{b-2})(P) \mod q$
                to T + $(2^{b-2})(P) \mod q$
    IF Spin-Out = ON THEN switch b-th bit in S
    BCount : = BCount + 1
    Move piece 0 from T to T+P
    T : = T + P
    IF Spin-Out = ON THEN switch 0-th bit in S
    BCount : = BCount + 1
ENDWHILE
```

The Count algorithm for the Combination puzzle is an amalgam of the Counting Algorithms for the SF Puzzle and the 2^m dimension puzzle. We first observe that when any information about Spin-Out is deleted, we are simply left with the Counting Algorithm for the Puzzle. Lemma 5.12 is therefore not proven as it is analogous to the theorems in Section 3.3.3. We also note that, as a consequence of 5.12, the algorithm terminates after completing $2^n - 1$ moves.

Lemma 5.12. *The tower configuration given in the Counting Algorithm for Combination Puzzle is exactly that given in the Recursive Algorithm for the Combination Puzzle. The Counting Algorithm for the Combination Puzzle thus gives correct tower configurations.*

We are left, then, to show that the spins of pieces are manipulated correctly. The following table illustrates the relationship between the spins of pieces and the binary counter for $n = 5$ with one spinner per piece ($m = 1$). As discussed in Section 4, this may be easily generalized to any number m of spinners per piece.

Binary Count	Spin Configuration		Binary Count	Spin Configuration	
00000	11111	OFF	10000	01000	ON
00001	11110	ON	10001	01001	
00010	11110	OFF	10010	01011	
00011	11110		10011	01010	
00100	11010	ON	10100	01110	
00101	11011		10101	01111	
00110	11001		10110	01101	
00111	11000		10111	01100	
01000	11000	OFF	11000	00100	
01001	11000		11001	00101	
01010	11000		11010	00111	
01011	11000		11011	00110	
01100	11000		11100	00010	
01101	11000		11101	00011	
01110	11000		11110	00001	
01111	11000		11111	00000	

The position of the horizontal lines between $2^n - 1$ and 2^n show exactly where the Spin-Out switch is turned **ON** and **OFF**. We observe that when Spin-Out is **OFF**, the spins of the pieces are unaffected. It is during these moves that, in the recursive algorithm, the subprocedure Modified Hanoi is being called. It may seem that Modified Hanoi is changing the Spin, but in fact it isn't; the $(n-i)$ pieces on which it is being called all have spin 0, and therefore adding the spin of any two consecutive pieces to obey the total orientation change function will not change their spin. Notice that these periods of unchanging spin occur precisely for $(2^j - 1) - (2^{j-1} - 1)$ moves, which is precisely the number of moves that Modified Hanoi takes.

In the sections of the Spin Configuration table when Spin-Out is **ON**, we see that it is following parts of the binary reflected Gray code. By Theorem 2.11, these are legal moves in Spin-Out and thus for the Spin Configuration of the Combination puzzle. The reason that it is exactly the binary reflected Gray code is that the same procedure is followed as the Counting Algorithm for Spin-Out, which is to find the rightmost 0 in the binary counter and then switch the corresponding bit in Gray code. The proof of Lemma 5.13 is therefore analogous to the proof of Theorem 4.16.

Lemma 5.13. *The Counting Algorithm for the Combination Puzzle makes correct moves to the spin configuration of the pieces.*

Theorem 5.14. *The Counting Algorithm for the Combination Puzzle correctly solves the Combination Puzzle.*

The proof of Theorem 5.14 is an easy consequence of Lemmas 5.12 and 5.13.

6. Hamiltonian Paths, the Ternary Reflected Gray Code, and a Finite State Machine

6.1. **Hamiltonian Paths.**

Definition 6.1. *A* Hamiltonian path *is a path in a graph that visits each vertex in the graph exactly once. A* Hamiltonian circuit *is a cycle in a graph that visits each vertex exactly once and has the same end vertex as its beginning vertex.*

We will begin with proofs that Hamiltonian paths and circuits exist.

Proposition 6.2. *For any pair of corner vertices in K_d^n there is a Hamiltonian path between them. If $d > 2, K_d^n$ has a Hamiltonian circuit.*

Proof. For $n = 1$, the claim is obvious because K_d^1 is a complete graph and thus has a Hamiltonian path between any pair of vertices, and except when $d = 2$ (the straight line), there is a Hamiltonian circuit. By its construction K_d^n is a complete graph whose vertices are d copies of K_d^{n-1}. Any vertex in K_d^n can be represented as (x,C) where C is one of the copies of K_d^{n-1} and x is a vertex within C. If (x,C) is a corner vertex of K_d^n, then x is also a corner vertex of C. Let $v_1 = (x_1,C_1)$ and $v_2 = (\hat{x}_d,C_d)$. Use the assumed Hamiltonian path of K_d^1 from C_1 to C_d to give an ordering $C_1,C_2,...,C_d$ on the copies of K_d^{n-1}. Complete the Hamiltonian path of C_1 from x_1 to the corner $\hat{x}_1$ which is adjacent to a corner of C_2 and call this corner x_2. Similarly, use the Hamiltonian path of C_2 from x_2 to $\hat{x}_2$ where $\hat{x}_2$ is adjacent to a corner of C_3. Continue this construction and finally use the Hamiltonian path of C_d to end up at $\hat{x}_d$. For the Hamiltonian circuit, assume $d > 2$, so that K_d^1 has a Hamiltonian circuit. Use this Hamiltonian circuit to put an ordering $C_1,C_2,...,C_d$ on the d copies of K_d^{n-1}. The pick pairs of corner vertices v_i and $\hat{v}_i$ for each C_i, so that v_i is adjacent to a corner vertex $\hat{v}_{i-1}$ of C_{i-1} and $\hat{v}_i$ is adjacent to a corner vertex of C_{i+1}. (Of course, C_{d+1} is C_1 and C_d is C_{i-1}.) Then use the above Hamiltonian path construction to connect the Hamiltonian paths of the C_i's into a Hamiltonian circuit of K_d^n. □

We will now discuss the creation of a Hamiltonian path for K_3^n and some of its properties. We will focus on paths between corner vertices because of their nice relation to the ternary reflected Gray code. The construction of such a path is simple and requires tracking only the lower-order bit of each label. Beginning at $0\dots0$, follow the pattern 0, 1, 2, 2, 1, 0 in the lower-order bits.

We may also think of such a path as being constructed recursively. Orient the graph such that $0\dots0$ is at the top of the graph. Begin with the Hamiltonian path on the top K_3^1 of the graph. If n is even, construc a path in the same pattern on each K_3^{n-1}, following the subgraphs clockwise around the graph. If n is odd, do the same but in the counterclockwise direction. Figure 20 shows the construction for K_3^1, K_3^2, and K_3^3. This construction is shown in the following theorem to be unique.

Theorem 6.3. *There is a unique Hamiltonian path between vertices any two corner vertices in the complete iterated graph K_3^n.*

Proof. We will proceed by induction.
Base cases: $n = 1$: It is clear that there is a unique Hamiltonian path between any two corner vertices. $n = 2$: We will name the 3 K_3^1 subgraphs G_1, G_2, and G_3. Assume that the endpoints are in G_1 and G_3. Note that in order for the path to be Hamiltonian, it must also pass through G_2. As each subgraph is connected by only one edge to the other two, these edges must be transversed in

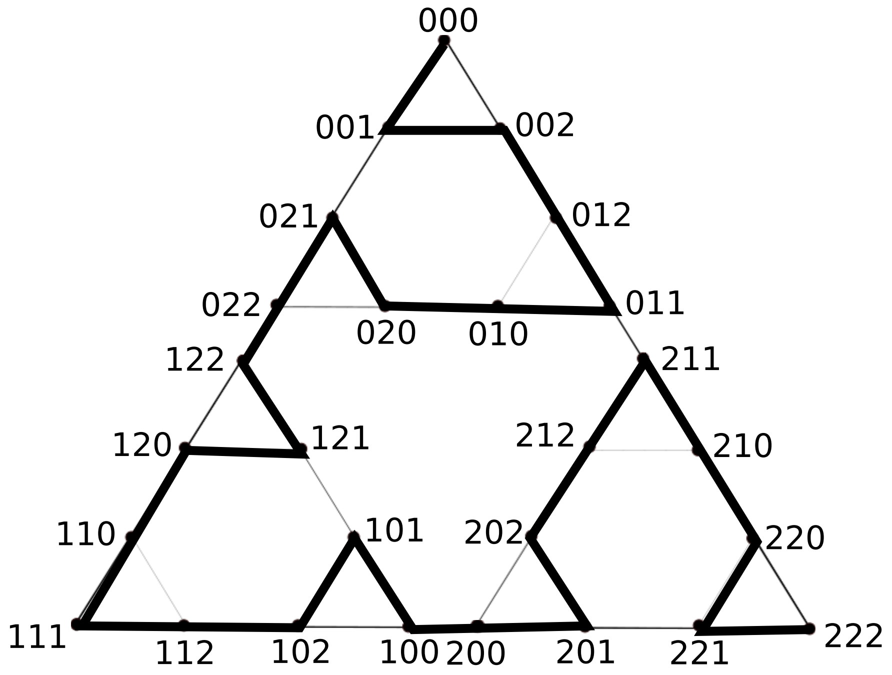


FIGURE 19. The thick line represents the unique Hamiltonian path on K_3^3. Note that the vertex labels are reflected.

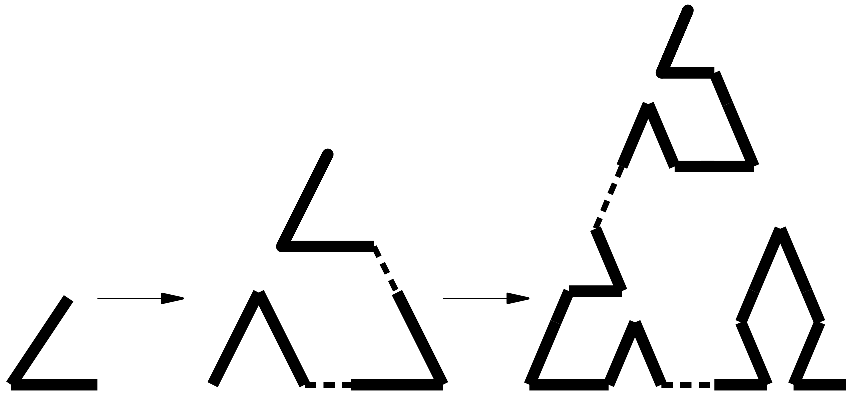


FIGURE 20. The construction of the unique Hamiltonian path for K_3^1, K_3^2, and K_3^3.

the order $G_1 - G_2$ and $G_2 - G_3$. As each G_i contains a unique Hamiltonian path and the connections between each subgraph are unique, the path is unique.
Inductive hypothesis: Assume that there exists a unique Hamiltonian path between any two corner vertices in K_3^{n-1}.

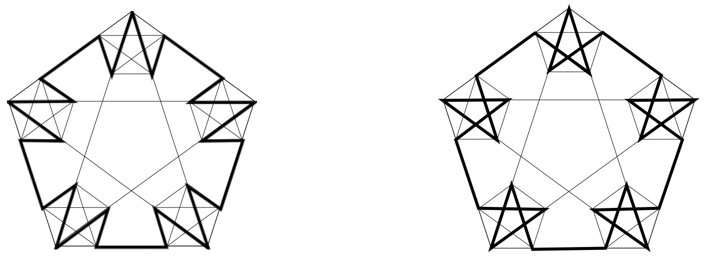


FIGURE 21. Two possibilities for Hamiltonian paths in K_5^2. Without labelings, these paths are the same, but with labelings, they would be distinct.

Induction step: Let $G_1, G_2,$, and G_3 be the 3 K_3^{n-1} subgraphs. Assume without loss of generality that the endpoints of the Hamiltonian path are in G_1 and G_3. The Hamiltonian path must also pass through G_2. The Hamiltonian path must also pass along the edges $G_1 - G_2$ and $G_2 - G_3$. These edges begin and end on corner vertices in the G_2 subgraph, and by the induction hypothesis there is a unique Hamiltonian path within it. The paths from the beginning corner to the edge $G_1 - G_2$ and the edge $G_2 - G_3$ to the ending vertex are also Hamiltonian paths within G_1 and G_3 respectively. As the connections between the subgraphs are unique, and the paths within them are unique, there is a unique Hamiltonian path from one corner vertex to another in K_3^n. □

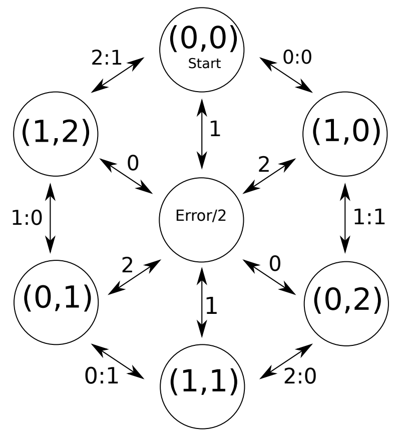
It is clear that the above result does not generalize for K_q^n when $q > 3$. Figure 21 shows two possible Hamiltonian paths on K_5^2. It would be an interesting combinatorial problem to investigate the number of unique Hamiltonian paths for other odd qs and their implications for Gray code conversion.

6.2. **The Ternary Reflected Gray Code.** As previously described, the Hamiltonian path on K_3^n defines a sequence of strings (n bits) that correspond to the ternary reflected Gray code. Each string in the sequence differs from the next in exactly one digit (in base 3). We describe the following conversion formula for converting from ternary numbers to ternary reflected Gray code. Let t be some ternary number. Denote the n^{th} digit in t, which will be the leftmost or highest order digit, by t_n. We will refer to the lower order digits by $n-i$ for some $i \in \{1,\dots,n-1\}$. Given t_{n-i}, to obtain the $(n-i)^{th}$ digit in the corresponding ternary reflected Gray code, use the following conversion formulas.

$$\text{If } \sum_{j=0}^{i} t_{n-j} = 0 \bmod 2 \text{ then } g_{n-i} = t_{n-i}$$

$$\text{If } \sum_{j=0}^{i} t_{n-j} = 1 \bmod 2 \text{ then } g_{n-i} = 2 - t_{n-i}$$

6.3. **Finite State Machine for the Towers of Hanoi.**



Definition 6.4. *Let $A = \{a,b\dots\}$ be a finite alphabet. A* finite state machine *consists of:*

(1) *A finite non-empty set F, the set of* states*;*
(2) *A element s_0 of F called the* start state*;*
(3) *Unary functions $f_x : F \to F$, one for each $x \in A$, called the* transition functions.

The Finite State Machine in Figure 6.3 correctly identifies vertices in the optimal solution of the Towers of Hanoi with three towers and returns in binary a number corresponding to their position in the sequence of moves.

Example 6.5. *Consider the vertex labels 110 and 220 in K_3^3, which corresponds to the Towers of Hanoi with 3 towers and 3 disks. We would like to know whether or not these configurations are a part of the minimal solution, and if they are, the binary number corresponding to their place in the minimal solution's unique sequence of moves. The Finite State Machine reads the strings from lowest to highest order, or right to left. We always begin in the start state and follow the arrows. Notice that each state has three arrows for each of the three ternary digits. The notation $a : b$ may be read, "given a, output b." Thus, when given the string 110, we travel from state (0,0) to state (1,0), then to state (0,2), and then back to state (1,0), while being given the output 011. This means that this configuration is legal and is the third configuration on the minimal path on the complete iterated graph K_3^3 corresponding to the correct solution of Towers of Hanoi with three towers and three disks. When given the string 220, we travel from (0,0) to (1,2), then to (0,1), and then to Error/2. We are given the output 112, which is not a valid binary number and therefore the configuration is not on the minimal solution path.*

The labels of the states in the finite state machine are of the form (y_1, y_2). This labeling was adopted in an attempt to create an equation or system of equations that would remember perhaps the last ternary or binary digits and the immediately previous state information and output binary numbers or an error. In other words, we would like to have equations that serve the purpose of the finite state machine without the machine itself. The transition function in terms of the ternary digit input x and state label (y_1, y_2) is given by $f(x, y_1, y_2) = ((-1)^{y_1+1}x + y_2) \bmod 3$. Attempts were made to express y_1 and y_2 in terms of a small number of the previous ternary or binary numbers, but none were successful. This is a possibility for future research.

7. Conclusions and Future Research

We successfully developed recursive, iterative, and counting algorithms for the Combination puzzle as well as a counting algorithm for the Dimension 2^m puzzle. We made slight modifications to the older nested recursive algorithm for the Dimension 2^m puzzle and introduced a mutual recursive algorithm for the same puzzle. We illustrated more deeply the connection between the binary reflected Gray code and the Dimension 2^m and Combination puzzles. We explored properties of Hamiltonian paths on the complete iterated graphs associated with all of the puzzles we studied. We presented a ternary to ternary reflected Gray code conversion formula that has been verified computationally with Maple, but still requires a proof. We suspect that there may be another formula that takes into account only a few of the previous ternary digits and maybe the previous Gray digit. Lastly, we hope that the finite state machine may be converted into formulae that would indicate the presence of a vertex along a given path in a graph. Finally, we present various Maple programs to make generating examples easier. Future research on these complete iterated graphs would include:

(1) Exploring the connection between Hamiltonian paths, Gray codes, and perfect one-error correcting codes in iterated complete graphs;
(2) Developing a way of generating Hamiltonian paths and circuits in complete iterated graphs K_q^n with $q > 3$ and finding which paths have simple relations to a base q counter;
(3) Proving the correctness of the iterative algorithm for the Combination puzzle;
(4) Verifying the correctness of the ternary to ternary reflected Gray code conversion and perhaps finding a formula that takes fewer inputs and is a bijection;

Appendix A. Maple Programs

A.1. **Maple Programs.** The following program gives a seqence of arrays corresponding to configurations in the minimal solution path for Towers of Hanoi where n is the number of disks and q is the number of towers.

```
Towers:= proc(n,q)
> # n is the number of disks/pieces, q is the number of towers
> # output is seq of arrays corresponding to configurations
>  # in min solution path
> local T, P, counter, tower, moves, b, binarynum ;
>
> moves:= [Config(n,0)];
> tower:= Config(n,0); # all the pieces are on tower 0
> counter:=0;
> T:= 0 mod q ;
> P:= [(-1)*(1/2)^(n-1)] mod q;
```

```
>
> while (counter <> 2^n - 2 )      do
>
> tower[-1]:= op(T+P) mod q ;
>          moves:= [op(moves),tower];
>    T:= (T+P mod q) ;
>               counter:= counter + 1;
>
>    binarynum:=convert(counter,binary);
>    b:=RightZero(convert(binarynum,string));
>
> tower[-b]:= (op(T)-(2^(b-2)*(1/2)^(n-1)) mod q);
>          moves:=[op(moves),tower];
>               counter:=counter+1;
>                                 od;
>
> return concat([op(moves),Config(n,q-1)]);
> end:
```

The following program gives a sequence of arrays corresponding the the spin configuration in the minimal solution path for the Combination Puzzle where $n = 4$ is the number of pieces.

```
Spin:= proc(n)
> # n is number of pieces
> # output is seq of arrays corresponding to SPIN config. in
> # min sol path for COMBINATION puzzle
> local moves, GrayLabel, counter, binarynum, b, k, z, spinout    ;
>
> GrayLabel:= Config(n,1);
> counter:=0;
>
> moves:=[Config(n,1)];
> z:= n mod 2;
> if z = 1 then spinout:= true; else spinout:= false; fi;
> # true = ON , false = OFF
> if spinout = true then GrayLabel[-1] := GrayLabel[-1] + 1 mod 2;
>                       moves:= [op(moves),GrayLabel];
>                  else moves:= [op(moves),GrayLabel];    fi;
> counter:= counter + 1 ;
>
>
> while   (counter <> 2^n - 1)                                       do
>
> binarynum:= convert(counter, binary);
> k:= BinaryTest(binarynum) ;
> b:= RightZero(convert(binarynum,string));
>
> if k > 0 then spinout:= not spinout; fi;
>
> if spinout = true then GrayLabel[-b]:= GrayLabel[-b] + 1 mod 2;
>                        moves:= [op(moves),GrayLabel];
>                   else moves:= [op(moves),GrayLabel];     fi;
>
> counter:= counter + 1;
>
> if spinout = true then GrayLabel[-1] := GrayLabel[-1] + 1 mod 2;
>                        moves:= [op(moves),GrayLabel];
>                   else moves:= [op(moves),GrayLabel];     fi;
>
> counter:= counter + 1;
>                                                                    od;
> return concat(moves);
> end:
```

The following program gives a sequence of arrays corresponding to configurations in the minmum solution path of Spin-Out; that is, it gives the binary reflected Gray code beginning with all 1s and ending with all 0s where *n* is the number of pieces. It calls the subprocedures RightZero and concat which look for the rightmost zero in the binary string and concatenated arrays, respectively.

```
Spinout:= proc(n)
> # n is the number of spinners
> # output is seq of arrays corresponding to configurations in
> #min sol path to SPINOUT
> local GrayLabel,counter,binarynum,b,moves;
> GrayLabel:= Config(n,1);
> moves:=[Config(n,1)];
> counter:= ceil( (2/3)*(2^n - 1) );
> while (counter <> 0)                     do
> counter:= counter - 1;
> binarynum:= convert(counter,binary);
> b:= RightZero(convert(binarynum,string));
> GrayLabel[-b]:= GrayLabel[-b]+1 mod 2;
> moves:= [op(moves),GrayLabel];
>                                          od;
> return op(moves);
> end:

RightZero:=proc(b)
> # input is a binary number in string, output is the
> # position (counting from right to left) of the rightmost 0
> local  x,y;
> with(StringTools):
> y:=Reverse(b);
> x:=FirstFromLeft("0",y);
> if x=0 then x:= length(b)+1; fi;
> return x;
> end:

concat:= proc(x)
> # input is array of arrays
> # output is array of concatenation of arrays
> local z,i    ;
> z:= [];
> for i from 1 to nops(x)     do
> z:= [op(z),cat(op(x[i]))];
>                             od;
> return z;
> end:
```

The following program removes the last bit from a given binary string.

```
Chopbinary:=proc(b)
> # b is a binary number, output will chop off the last bit and
> #return the remaining binary string
> local x;
> x:= convert(floor(convert(b,decimal,binary)/2),binary);
> return x;
> end:
```

The following program creates an array with *n* entries and assigns a value *m* to each of them.

```
Config:=proc(n,m)
> # Creates an array of size n, where m is the value
> #assigned to each entry of the array
> local x ,i  ;
> x:=[];
> for i from 1 to n do
> x:= [op(x),m];
>                    od;
> end:
```

The following program tests whether a binary number is of the form $0\dots01\dots1$; that is, if it is equal to $2^i - 1$ for some i in decimal notation.

```
BinaryTest:=proc(b)
> # input is binary number
> # the procedure will test if b is of the form 00..0011..11
> # if b is of the form, return the number of 1s else return 0
> local counter,temp   ;
> counter:= 0;
> temp:= b;
> while (temp <> 0)                do
> if temp mod 2 = 1 then counter:= counter + 1;
>        else return (0);          fi;
> temp:= Chopbinary(temp);
>                                  od;
> return counter;
> end:
```

The following program generates the ternary reflected Gray code.

```
RTGray := proc(n)
> # n is number of bits, output is reflected Ternary gray code
> local  i,z,up,mid,down,l,matr,zeros,ones,twos ;
> with(ArrayTools);
>
>
> if n = 1 then
>               z:= Array([]);
>               for i from 1 to 3 do
>                   z(i,1):=i-1;
>               od; return z;
> fi;
> if n > 1 then
>                  up:= RTGray(n-1);
>                  mid:= FlipDimension(RTGray(n-1),1);
>                  down:= RTGray(n-1);
>                  z:= Concatenate(1,up,mid,down);
>                  zeros:= Vector(1..3^(n-1));
>                  ones:= Vector(1..3^(n-1),fill=1);
>                  twos:= Vector(1..3^(n-1),fill=2);
>                  l:= Concatenate(1,zeros,ones,twos);
>                  matr:= Concatenate(2,l,z) ;
>                  return matr;fi;
> end:
```

The following program uses the ternary to ternary reflected Gray code conversion formula to take an integer n with m bits to the first $n+1$ terms of the ternary reflected Gray code.

```
TG:=proc(n,m)
> # input integer n
> # input integer m for number of bits
> # output a list of the first n+1 reflected ternary gray code (starting from 0)
> # This algorithm uses the TERNARY TO TERNARY CONVERSION FORMULA
>
> local g,t,i,j,k,l,w,z;
> with(ListTools);
> with(ArrayTools);
>
> for i from 0 to n  do
>
> g[i]:= Array(1..m);   #print( g[i]);
> t[i]:= Array(Reverse(convert(i,base,3))); #print(t[i]);
>
> if  Size(t[i])[2] < m then
>                             w:= Array(1..(m-Size(t[i])[2]),fill=0);
>                             t[i]:= Concatenate(2,w,t[i]);
> fi; #print(t[i]);
>
>   for j from 1 to Size(t[i])[2]   do
>       if add((t[i])[k],k=1..j) mod 2 = 0 then (g[i])[j] := (t[i])[j];
>          else  (g[i])[j]:= 2-(t[i])[j] mod 3;   fi;      od;
>                              od;
>
> z:= Concatenate(1,seq(g[l],l=0..n));
> return z;
> end:
```

The following program outputs the binary reflected Gray code.

```
RBGray:=proc(n)
> # input n, number of bits
> # Output reflected binary gray code
> local  i,z,up,down,zeros,ones,l,matr  ;
> with(ArrayTools);
> if n = 1 then
>                 z:= Array([]);
>                 for i from 1 to 2 do
>                 z(i,1):= i - 1;
>                 od; return z;
>
> fi;
>
> if n > 1 then
>               up:= RBGray(n-1);
>               down:= FlipDimension(RBGray(n-1),1);
>               z:= Concatenate(1, up, down);
>               zeros:= Vector(1..2^(n-1));
>               ones:= Vector(1..2^(n-1),fill=1);
>               l:= Concatenate(1,zeros,ones);
>               matr:= Concatenate(2,l,z) ;
>                 return matr;
> fi;
> end:
```

REFERENCES

[1] Danielle Arett and Suzanne Doree. Coloring and counting on the tower of hanoi graphs. *Mathematics Magazine*, 2010.

[2] A. Barg. Some new NP-complete coding problems. *Problems of Information Transmission*, 30(3):44–49, 1994.

[3] A. Barg. Complexity issues in coding theory. *Handbook of Coding Theory*, 1998.

[4] Lindsay Baun and Sonia Chauhan. Puzzles on graphs: The towers of hanoi, the spin-out puzzle, and the combination puzzle. 2009.

[5] E.R. Berlekamp, R.J. McEliece, and C.A van Tilborg. On the inherent intractability of certain coding problems. *IEEE Trans. Inf. Theory*, IT-24(3):384–386, May 1978.

[6] N. Biggs. Perfect codes in graphs. *J. Combinatorial Theory(B)*, 15:289–296, 1973.

[7] J. Bruck and M. Noar. The hardness of decoding linear codes with preprocessing. *IEEE Trans. Inf. Theory*, IT-36:331–335, 1990.

[8] Peter Buneman and Leon S. Levy. The Towers of Hanoi Problem. *Information Processing Letters*, 10(4/5):243–244, 1980.

[9] Xiaomin Chen, Bin Tian, and Lei Wang. Santa Claus' towers of Hanoi. *Graphs and Combinatorics*, 23(suppl. 1):153–167, 2007.

[10] P. Cull and E.F. Ecklund Jr. Towers of Hanoi and Analysis of Algorithms. *American Mathematical Monthly*, 92(6):407–420, June-July 1985.

[11] Paul Cull. *A Brief Introduction to Algorithms*. CS 325 Class Notes http://classes.engr.oregonstate.edu/eecs/fall2009/cs325/ .
, 2005.

[12] Paul Cull, Mary Flahive, and Robby Robson. *Difference Equations*. Springer, New York, 2005.

[13] Paul Cull and Ingrid Nelson. Error-correcting codes on the towers of Hanoi graphs. *Discrete Math.*, 208/209:157–175, 1999.

[14] Paul Cull and Ingrid Nelson. Perfect Codes, NP-Completeness, and Towers of Hanoi Graphs. *Bull. Inst. Combin. Appl.*, 26:13–38, 1999.

[15] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Comple teness*. W. H. Freeman, San Francisco, 1979.

[16] Jonathan Gross and Jay Yellin. *Graph Theory and its Applications (2nd Edition)*. Chapman & Hall, Boca Raton, FL, 2006.

[17] R. Hamming. Error detecting and error correcting codes. *Bell Syst. Tech. J.*, 29:147–160, 1950.

[18] R. Hill. *A First Course in Coding Theory*. Oxford University Press, Oxford, 1986.

[19] Wifried Imrich, Sandi Klavzar, and Douglas F. Rall. *Topics in Graph Theory: Graphs and Their Cartesian Product*. A.K. Peters, Ltd., Wellesley, Massachusetts, 2008.

[20] Kathleen King. A new puzzle based on the SF labelling of iterated complete graphs. 2004.

[21] Sandi Klavžar, Uroš Milutinović, and Ciril Petr. On the Frame-Stewart algorithm for the multi-peg Tower of Hanoi problem. *Discrete Applied Mathematics.*, 120(1-3):141–157, 2002.

[22] Stephanie Kleven. Perfect Codes on Odd Dimension Serpinski Graphs. 2003.

[23] J. Kratochvii. *Perfect Codes in General Graphs*. Academia, Prague, 1991.

[24] J. Kratochvii. Regular Codes in Regular Graphs are Difficult. *Discrete Mathematics*, 133:191–205, 1994.

[25] J. Kratochvii and M. Krivanek. On the computational complexity of codes in graphs. *Lecture Notes in Computer Science*, 324:396–404, 1988.

[26] Chi-Kwong Li and Ingrid Nelson. Perfect codes on the Towers of Hanoi graph. *Bull. Austral. Math. Soc.*, 57:367–376, 1998.

[27] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam, 1977.

[28] Ingrid Nelson. Coding Theory on the Towers of Hanoi. 1995.

[29] S.C. Ntafos and S.L. Hakimi. On the complexity of some coding problems. *IEEE Trans. Inf. Theory*, IT-27(6):794–796, Nov 1981.

[30] Kirk Pruhs. The SPIN-OUT puzzle. *ACM SIGCSE Bulletin*, 25:36–38, 1993.

[31] Nick Stevenson and Beth Skubak. A new puzzle for complete iterated graphs of any dimension. 2008.

[32] A. Tietavainen and A. Perko. There are no unknown perfect binary codes. *Ann. Univ. Turku*, 148:3–10, 1971.

[33] T. R. Walsh. The Towers of Hanoi revisited: moving the rings by countion the moves. *Information Processing Letters*, 15:64–67, 1982.

[34] Elizabeth Weaver. Gray codes and puzzles on iterated complete graphs. 2005.

[35] Dejan Živković. Hamiltonianicity of the Towers of Hanoi problem. *Univ. Beograd. Publ. Elektrotehn. Fak. Ser. Mat.*, 17:31–37, 2006.

SUNY POTSDAM
E-mail address: `merrille190@potsdam.edu`

UNIVERSITY OF PENNSYLVANIA
E-mail address: `vant@sas.upenn.edu`