Journal of Information Processing Vol.21 No.3 378–392 (July 2013)

[DOI: 10.2197/ipsjjip.21.378]

Regular Paper

A Tale of Two Puzzles: Towers of Hanoi and Spin-Out

Paul Cull[1,a)] Leanne Merrill[2,b)] Tony Van[1]

Received: July 26, 2012, Accepted: November 2, 2012

Abstract: Once upon a time, there were two puzzles. One was the Towers of Hanoi invented or introduced by Eduardo Lucas in 1883. The other was Spin-Out patented by William Keister in 1972. There are many stories about these puzzles. Some of these stories hint or claim that these puzzles have an intimate relationship with the Gray codes invented by Frank Gray in 1947. Here, we wish to show how these puzzles can be generalized and crossed to give puzzles for every base and for every number of pieces. The Gray relationship will become clearer when we describe the graphs associated with the puzzles and the graph labelings induced by the puzzles. These labelings will have the Gray property in the appropriate base. Counter to claims that Gray counting is needed to solve these puzzles, we describe counting algorithms which solve these puzzles using a standard binary counter. We also give recursive and iterative algorithms for these puzzles.

Keywords: puzzles, graphs, algorithms, Towers of Hanoi, Spin-Out, iterated complete graphs

1. Introduction

People enjoy playing with puzzles. The manipulation and concomitant tactile stimulation is satisfying enough, but the mental stimulation in understanding the solution of the puzzle is, for some, even more satisfying. In this paper, we want to relate our tale of investigating two well-known puzzles, Towers of Hanoi and Spin-Out, and how our attempts at understanding these puzzles led us to the creation of a whole family of puzzles, one for each positive integer.

Puzzles are intimately connected with graphs. The vertices of a graph are the configurations of the puzzle, and the graph's edges specify the allowed moves of the puzzle. A solution to the puzzle is a path from a starting vertex corresponding to the initial configuration of the puzzle to a target vertex corresponding to the final configuration of the puzzle. So, in our construction of puzzles, we also create a bi-infinite family of graphs, which we call the *iterated complete graphs*, K_d^n. Here, d which we call the dimension specifies the type of puzzle, and n which we call the iteration indicates the number of pieces in the puzzle.

Some years ago, we investigated the Towers of Hanoi and algorithms to solve this puzzle [4]. We found, in particular, that this puzzle could be solved using only a binary counter. Somewhat to our dismay, whenever we mentioned this result, we were met with the response "Oh yes, you can solve Towers of Hanoi using the Gray code." This response is presumably a reference to an article by Martin Gardner [9] in which he shows that the disk to be moved on the k^{th} move is given by which digit is changed in in going from k to $k+1$ in the Gray sequence. The Gray code referred to here is the binary reflected Gray code patented by Frank Gray in 1947 [8]. In more generality, a Gray code is a sequence of strings over some alphabet, so that the k^{th} and $(k+1)^{\text{st}}$ strings differ in exactly one position [18]. For puzzles, the Gray property is almost automatic, in that it corresponds to the rule that only one piece may be moved at a time. Hence, the sequence of configurations for the solution of a puzzle should have the Gray property. For Towers of Hanoi, the alphabet could be $\{A, B, C\}$ or $\{0, 1, 2\}$ and the solution sequence would have the Gray property over a 3 character alphabet. Where does the binary Gray code fit in? As we will see, Spin-Out corresponds exactly to the binary Gray code. Further, as we will see, the changing digit in the binary Gray code corresponds to the rightmost 0 in a binary counter and this rightmost 0 indicates which piece to move not only in Spin-Out and Towers of Hanoi, but also in the whole family of puzzles we will construct. This *rightmost rule* is the basis for our *counting* algorithms to solve these puzzles.

2. Towers of Hanoi

Here, we will recall some well-know facts about the familar Towers of Hanoi puzzle. More detailed discussion of Towers of Hanoi is in Cull and Ecklund [4]. In the Towers of Hanoi problem, one is given three towers, usually called A, B, and C, and n disks of different sizes. Initially the disks are stacked on tower A in order of size with $disk_n$, the largest, on the bottom, and $disk_1$, the smallest, on the top. The problem is to move the stack of disks to tower C, moving the disks one at a time in such a way that a larger disk is never stacked on top of a smaller disk. An extra constraint is that the sequence of moves should be as short as possible. An algorithm solves the Towers of Hanoi problem

[1] Computer Science, Kelley Engineering Center, Oregon State University, Corvallis, OR 97331, USA
[2] Department of Mathematics, University of Oregon, Eugene, OR 97403, USA
[a)] pc@eecs.oregonstate.edu
[b)] leannem@uoregon.edu

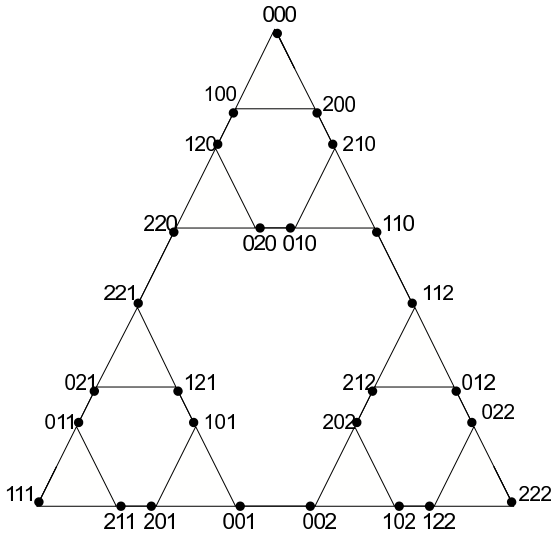


Fig. 1 The labeled graph K_3^3 corresponding to the Towers of Hanoi with 3 disks. For example, 112 means that disks one and two are on tower 1, and disk three is on tower 2.

if, when the algorithm is given as input n the number of disks, and the names of the towers, then the algorithm produces the shortest sequence of moves which conforms to the above rules. To set things up for development of other puzzles, we will indicate the towers by the numbers 0, 1, and 2 because later we will use MOD 3 arithmetic. For the 3 tower Towers of Hanoi we use k = neither i nor j to mean the number that is different from i and j when i and j are distinct.

The following is the well-known recursive algorithm for Towers of Hanoi. It produces the unique minimal move solution to the problem of moving n disks from tower i to tower j.

RECURSIVE ALGORITHM

```
PROCEDURE HANOI(i, j, n)
IF n > 0
   THEN
      k = neither i nor j
      HANOI (i, k, n − 1)
      move the top disk from tower i to tower j
      HANOI(k, j, n − 1)
```

It is easy to give an inductive proof of this fact, and it is also easy to calculate the number of moves in this solution. The number of moves is $2^n - 1$, which follows readily from the difference equation

$$M(n) = 2\,M(n-1) + 1 \quad \text{with} \quad M(1) = 1.$$

Here, of course, $M(n)$ is the number of moves, $M(1) = 1$ because it takes one move to solve the problem for a single disk, and the algorithm calls itself twice using one less disk.

An essential idea is that the states of this puzzle can be represented by the graph in **Fig. 1**. Each node (vertex) represents a state of the puzzle. The label of a vertex indicates the tower on which a disk resides in the corresponding state. The towers are assigned the "names" 0, 1, and 2. For example, 112 means that disk 1, the smallest disk, is on tower 1, disk 2 is on tower 1, and disk 3 is on tower 2. At some point we may want to reverse the labels, so that the tower of the smallest disk is the rightmost character of the label.

The edges of the graph indicate *legal* moves in the puzzle. For example, the edge between 110 and 112 represents the move of the largest disk, disk 3, from tower 0 to tower 2. This is a legal move because the smaller disks are all on tower 1.

These Towers of Hanoi graphs have been widely studied. For example, by Refs. [6], [7], and [14] in the context of error-correcting code, and by Refs. [1] and [15] in the context of coloring and symmetries.

We draw the Towers of Hanoi graphs in levels. The top vertex $00\ldots0$ will be at level 0, and the two vertices adjacent to it will be at level 1. We can then recursively construct H_n, the Towers of Hanoi graph for n disks, by the following diagram:

$$H_n = \begin{array}{c} H_{n-1} \\ / \quad \backslash \\ H_{n-1} — H_{n-1} \end{array}$$

That is, we take three copies of the drawing of H_{n-1}, and connect them as follows. The top H_{n-1} has two corner vertices in its bottom row. We add an edge connecting one of these to the top vertex of the lower left H_{n-1}, and we add an edge connecting the other of these corner vertices to the top vertex of the bottom right H_{n-1}. We also add an edge between the bottom right corner vertex of the lower left H_{n-1} to the bottom left corner vertex of the lower right H_{n-1}. So if H_{n-1} has levels 0 through l, then H_n will have levels 0 through $2l + 1$.

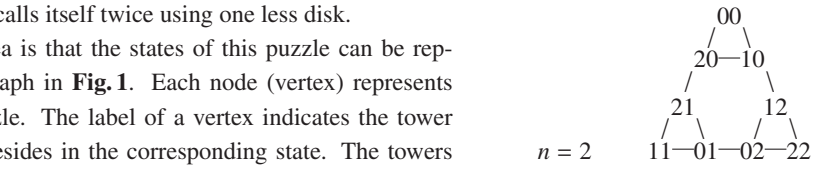
Although the above diagram captures the topology of the graphs, it does not display the labeling which we will need for our puzzles. Let L_n be the labeled graph, then:

$$L_{n+1} = \begin{array}{c} RL_n0 \\ / \quad \backslash \\ \uparrow RL_n1 — \downarrow RL_n2 \end{array}$$

By this we mean that the labeled graph for $n + 1$ disks can be constructed from 3 copies of the labeled graph for n disks. By RL_n we mean the labeled graph which is the mirror image of L_n. In L_n, the lower right vertex is labeled $22\ldots2$, and the lower left vertex is labeled $11\ldots1$. In RL_n, the lower right vertex is labeled $11\ldots1$, and the lower left vertex is labeled $22\ldots2$. The top copy, RL_n0, looks like RL_n, but each vertex has a 0 appended to its label. Similarly, $\uparrow RL_n1$ is a copy of RL_n which has been rotated 120 degrees clockwise and has a 1 appended to each label, and $\downarrow RL_n2$ is a copy of RL_n which has been rotated 120 degrees counterclockwise and has a 2 appended to each label. For example,

$n = 1$

$n = 2$



2.1 Iterative Algorithms

There are a variety of algorithms to solve the Towers of Hanoi

puzzle. Here, we will discuss two of them.

The following iterative algorithm is due to Buneman and Levy [2].

ITERATIVE ALGORITHM

```
move the smallest disk one tower clockwise
WHILE   a disk (other than the smallest)
        can be moved DO
     move that disk
     move the smallest disk one tower clockwise
ENDWHILE
```

The Buneman and Levy algorithm assumes that the towers are arranged in a circle or assigned the numbers 0, 1, 2 mod 3. The minor difficulty is to decide which way to move the smallest disk. If n is ODD, disk 1 (the smallest) should be moved to its target tower, but when n is EVEN, disk 1 should be moved to the non-target tower. The simplest way to handle this is to arrange the towers in circular order 0, 2, 1 when n is ODD and in circular order 0, 1, 2 when n is EVEN. Then moving disk 1 clockwise will always move it in the right direction.

Alternatively, one could always use the circular order 0, 1, 2 but then the n disks would be moved from tower 0 to tower 1 when n is ODD and the n disks would be moved from tower 0 to tower 2 when n is EVEN.

2.2 Counting Algorithm

The iterative algorithm assumes that one can look at the puzzle and see which disk to move. Somewhat surprisingly, the necessary information can be in a simple binary counter as used in the following algorithm.

COUNTING ALGORITHM

```
T:= 0 (*TOWER NUMBER COMPUTED MODULO 3*)
COUNT:= 0 (*COUNT HAS n BITS*)
P:= {1 if n is even } {-1 if n is odd }
     move disk 1 from T to T+P
     T:= T+P
     COUNT:= COUNT + 1
WHILE COUNT ≠ ALL 1's DO
     IF rightmost 0 in COUNT is in even position
               THEN move disk from T-P to T+P
               ELSE move disk from T+P to T-P
     COUNT:= COUNT + 1
     move disk 1 from T to T+P
     T:= T+P
     COUNT:= COUNT + 1
ENDWHILE
```

The above algorithm is due to Cull and Ecklund [4] and a similar counting algorithm was presented by Walsh [20]. We defer the correctness argument until after we have generalized this puzzle.

3. Spin-Out

Spin-Out is another popular, but somewhat lesser-known puzzle. The physically embodied puzzle **Spin-Out** was invented by William Keister in 1970. Copies of this embodied puzzle may be purchased for around $12 [19]. Another embodiment **The Brain** was produced by Mag-Nif but no longer seems to be available [12].

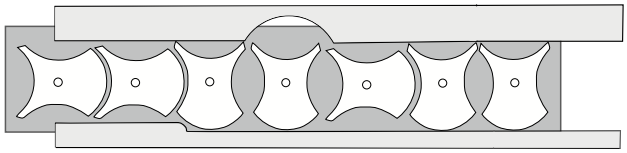


Fig. 2 A configuration of the Spin-Out puzzle, which corresponds to the labeling 0011011. The spinner under the arc may move, and we may also slide the inner rectangle to the right and move the leftmost spinner.

The abstract definition of these puzzles is given in the following box [5].

Spin-Out

The following locking system describes the Spin-Out puzzle.

The locking system has n interconnected locks so that:

(1) Lock 1 may be changed from locked to unlocked or from unlocked to locked, at any time.

(2) For $j > 1$, lock j may be changed from locked to unlocked (or vice versa), only if locks 1 through $j - 2$, are unlocked and lock $j - 1$ is locked.

The physically embodied puzzle is pictured in **Fig. 2**. Here lock 1 is on the left. By design, each spinner can have one of two spins; the rounded part may be pointed down, or the rounded part may be pointed to the right. We correspond the rounded part pointed down to "LOCK," and the rounded part pointed to the right to "UNLOCK." To make this numeric, we correspond "LOCK" to 1 and "UNLOCK" to 0. As one can see a lock (or spinner) may be rotated only when it is under the curved arc in the puzzle. For spinner 1, this is the only condition and so spinner 1 can be changed at any time. If spinners 1 through $j - 2$ are not ALL unlocked then spinner j can not be moved to under the curved arc. Finally, for spinner j to change spin, spinner $j - 1$ must have spin that allows spinner j to move. So this physical puzzle is described by the definition in the above box.

Proposition 3.1. *The state space of the Spin-Out puzzle is the reflected binary Gray code.*

This code may be described in a symbolic fashion by:

$$G_n \;=\; 0\,G_{n-1} \parallel 1\,G^R_{n-1}$$

Here, G_n means the sequence of n bit Gray numbers, i.e., the Gray bit strings corresponding to the integers in their natural order. For example, 000 corresponds to the number 0, while 111 corresponds to the number 5. $\parallel$ means followed by (or concatenation). G^R_{n-1} means the reversed sequence of $n-1$ bit Gray numbers, i.e., the bit strings corresponding to the integers in the reverse of their natural order. See **Fig. 3**.

Clearly, for Spin-Out with $n = 2$ the state space is

$$00 \longleftrightarrow 01 \longleftrightarrow 11 \longleftrightarrow 10$$

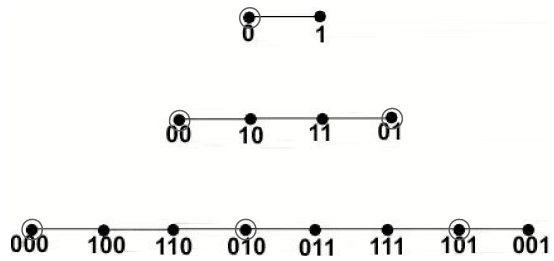


Fig. 3 The reflected binary Gray code for K_2^1, K_2^2, and K_2^3 with every third vertex circled.

which is the same as the binary reflected Gray code on 2 bits. (Often the Gray code is made *cyclic* by allowing 10 to change to 00, but this is not an allowed move in Spin-Out.)

Since we can maneuver spinners 1 through $n-1$ without changing spinner n, we may assume the following. If spinner n in Spin-Out has spin 0, then each state has a leading 0 and all of the other spinners have configurations which correspond to the Gray code on $n-1$ bits. If spinner n in Spin-Out has spin 1, then each state has a leading 1 and all of the other spinners have configurations which correspond to the Gray code on $n-1$ bits.

Changing the n^{th} spinner from 0 to 1 joins the the two state spaces for $n-1$ bits. This involves the two states $010\ldots0$ and $110\ldots0$. Using the orientation on $n-1$ bits which starts with $00\ldots0$ and ends with $10\ldots0$, means that when these two $n-1$ spinner state spaces are joined, the second half (with n spinners) runs from $110\ldots0$ to $10\ldots0$ and when limited to the last $n-1$ bits this sequence is identical to the sequence in the first half taken in reverse order. Using $00\ldots0$ as the first element of this state space forces an orientation on the states which corresponds to the orientation of the binary reflected Gray code.

(Notice that this is NOT how the puzzle is usually described. The puzzle usually starts with all the spinners locked, that is, state $11\ldots1$ and the problem is to reach the state $00\ldots0$, that is all unlocked. Conversely, the binary reflected Gray code is usually given as a sequence binary n-tuples which starts with the tuple $00\ldots0$ and ends with the tuple $10\ldots0$. In this listing of the Gray code, $11\ldots1$ appears about 2/3's of the way along the list.)

This state correspondence gives an easy way to calculate the number of moves needed to solve Spin-Out. Specifically, the (binary) number corresponding to the Gray code string $11\ldots1$ is the minimum number of moves required. As is well known [8], the binary number $b_n\ b_{n-1}\ \ldots\ b_1$ corresponding to the Gray string $g_n\ g_{n-1}\ \ldots\ g_1$ is given by $b_n = g_n,\quad b_{n-i} = g_{n-i} + b_{n-i+1}$. So, Gray $11\ldots1$ becomes either binary $1010\ldots01$ or binary $1010\ldots10$, and it's easy to show by summing in base 4, that this number is $\lceil 2/3\ (2^n - 1)\rceil$. This proves the folowing theorem.

Theorem 1. *The minimum number of moves needed to solve Spin-Out is* $\lceil 2/3\ (2^n - 1)\rceil$.

3.1 Algorithms for Spin-Out

Spin-Out can be solved using a pair of nested recursive procedures. `SOLVE` will take the puzzle from all locked, $11\ldots1$, to all unlocked, $00\ldots0$. `YSOLVE` takes the puzzle from $10\ldots0$ to $00\ldots0$ and also takes the puzzle from $00\ldots0$ to $10\ldots0$.

NESTED RECURSION

```
PROCEDURE SOLVE(n)
        IF n > 0 THEN
                SOLVE(n − 2)
                turn the n^th spinner
                YSOLVE(n − 1)

PROCEDURE YSOLVE(n)
        IF n > 0 THEN
                YSOLVE(n − 1)
                turn the n^th spinner
                YSOLVE(n − 1)
```

Notice that the calls to SOLVE and YSOLVE do NOTHING when the input parameter n is less than or equal to 0.

Let Y_n be the number of moves made by YSOLVE with input n, and S_n be the number of moves made by SOLVE with input n. Then

$$Y_n = 2\,Y_{n-1} + 1$$

and since $Y_1 = 1$ because with input $n = 1$ YSOLVE makes a single move,

$$Y_n = 2^n - 1.$$

From the algorithm

$$S_n = S_{n-2} + 1 + Y_{n-1} = S_{n-2} + 2^n.$$

Which implies

$$S_n = \begin{cases} 2/3\ (2^n - 1) & n \text{ EVEN} \\ 2/3\ (2^n - 1) + 1/3 & n \text{ ODD} \end{cases}$$
$$= \lceil 2/3\ (2^n - 1)\rceil.$$

Similar results were found by Pruhs [16] but there is a typo in his calculation of the number of moves.

MUTUAL RECURSION

```
PROCEDURE UNLOCK(n)
        IF n > 0 THEN
                UNLOCK(n − 2)
                turn the n^th spinner
                LOCK(n − 2)
                UNLOCK(n − 1)

PROCEDURE LOCK(n)
        IF n > 0 THEN
                LOCK(n − 1)
                UNLOCK(n − 2)
                turn the n^th spinner
                LOCK(n − 2)
```

Since LOCK and UNLOCK are inverse procedures, they each make the same number of moves. Letting U_n be the number of moves used by a call to UNLOCK(n), and L_n be the number of

moves used by a call to LOCK(n), we have from the algorithm

$$U_n = U_{n-2} + 1 + L_{n-2} + U_{n-1}$$

and using $U_n = L_n$ we have

$$U_n = U_{n-1} + 2\,U_{n-2} + 1.$$

And, using the initial conditions we get

$$U_n = \lceil 2/3 \ (2^n - 1) \rceil.$$

Like Towers of Hanoi, Spin-Out can also be solved by ITERATIVE and COUNTING algorithms, but we will defer giving these until we have generalized Spin-Out.

4. Iterated Complete Graphs

Our two example puzzles, Towers of Hanoi and Spin-Out, have highly structured graphs for their state spaces. In fact, the graphs for puzzles with more pieces are constructed recursively from the graphs with only a single piece. To constuct more general puzzles, we will first constuct the graphs for these new puzzles by generalizing this recursive graph construction.

We will use the usual definitions for a graph, $G = (V, E)$, and its usual parameters [10]. Also, as usual, K_d indicates the **complete graph** on d vertices. **Figure 4** shows three complete graphs.

Definition 4.1. *An **iterated complete graph** on d vertices with n iterations, denoted K_d^n, is defined recursively. K_d^1 is the complete graph on d vertices. K_d^n is composed of d copies of K_d^{n-1} and edges such that exactly one edge connects each K_d^{n-1} subgraph to every other K_d^{n-1} subgraph and exactly one vertex in each of the K_d^{n-1} subgraphs has degree $d-1$. We say that a graph K_d^n has **dimension** d.*

K_d^n can be thought of as K_d with each vertex replaced by a copy of K_d^{n-1}, or alternatively as the graph K_d^{n-1} with each vertex replaced by a copy of K_d. **Figure 5** shows the graphs K_5^1, K_5^2 and K_5^3, illustrating how each graph is constructed from the graph of the previous dimension.

The edges connecting the copies of K_d^{n-1} can be explained using the idea of **corner vertices.**

Definition 4.2. *A **corner vertex**, or simply **corner**, of the graph K_d^n is a vertex with degree $d-1$. A **non-corner vertex** is simply a vertex that is not a corner. All non-corner vertices of iterated complete graphs have degree d.*

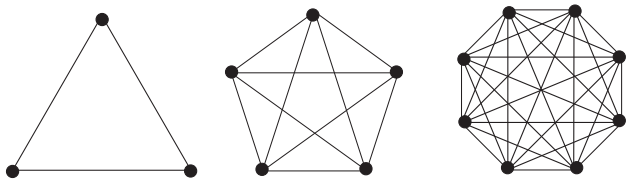


Fig. 4 The complete graphs K_3, K_5, and K_8.

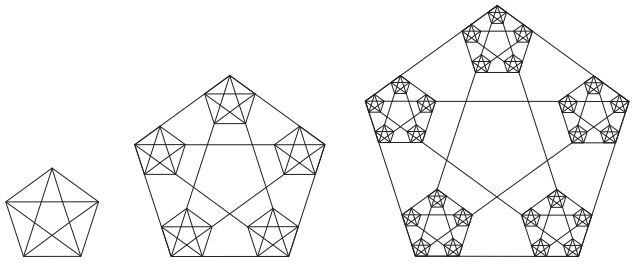


Fig. 5 The iterated complete graphs K_5^1, K_5^2 and K_5^3.

The d copies of K_d^{n-1} are connected by adding edges between corner vertices. $d-1$ corner vertices of a copy of K_d^{n-1} are connected to corner vertices of the other copies of K_d^{n-1}. Each of these $d-1$ corner vertices is attached to a distinct copy of K_d^{n-1}. Notice that this leaves one unattached corner vertex in each K_d^{n-1}. These unattached corners become the corner vertices of K_d^n.

Theorem 2. *The maximum distance between any two vertices in K_d^n is $2^n - 1$, and this maximum is attained when the two vertices are corner vertices.*

Proof. Clearly, $2^n - 1 = 1$ for $n = 1$ and each vertex is at distance 1 from every other vertex. Consider any two vertices in K_d^n, if they are in the same copy of K_d^{n-1} then by hypothesis they are at distance $\leq 2^{n-1} - 1$. If the two vertices, say x and y, are in different copies of K_d^{n-1}, then these two copies are attached by an edge from a corner vertex of x's copy to y's copy. By hypothesis, the distance from x to a corner vertex of its K_d^{n-1} is at most $2^{n-1} - 1$. Similarly, the distance from y to a corner vertex of its K_d^{n-1} is at most $2^{n-1} - 1$. Hence the distance between x and y is at most $2^{n-1} - 1 + 2^{n-1} - 1 + 1 = 2^n - 1$. The $+1$ in this equation corresponds to the edge between the corner vertices of the two copies of K_d^n.

For two corner vertices of K_d^n, the distance is $2^n - 1$ because a path from a corner to a corner must leave a copy of K_d^{n-1}, which can only occur at a corner of K_d^{n-1}, enter another copy of K_d^{n-1} and go from from a corner of K_d^{n-1} to the corner of this copy of K_d^{n-1} which is a corner of K_d^n. □

Similar graphs have been studied as Sierpinski graphs [17] using a labeling that does not have the Gray property and therfore is inconsistent with our idea of puzzles.

5. Generalized Tower Puzzles

The Generalized Towers of Hanoi has d towers where d is an ODD number and $d \geq 3$. ($d = 1$ corresponds to a trivial puzzle with no moves.)

The Geralized Towers of Hanoi puzzle has the same rules as Towers of Hanoi:

(1) Only one disk is moved at a time.
(2) A larger disk is never placed on top of a smaller disk.

In addition, this puzzle has the following restrictions to guarantee that the puzzle's graph is K_d^n:

(1) No disk may be moved unless all of the disks smaller than it are stacked together on the same tower.
(2) When a disk is able to move, if the stack of smaller disks is on tower a and the disk to be moved is on tower b, then the disk may only move to tower $(2a - b) \bmod d$.

Figure 6 shows the labeled graph for 5 towers and 3 disks. This should be compared with Fig. 1 for the traditional puzzle with 3 towers and 3 disks. Observe the smallest disk is able to move to any tower because it is unaffected by these rules. **Figure 7** shows configurations corresponding to labels 220 and 224 on K_5^3. Here the largest disk can only move between towers 0 and 4, and there is an edge between these two vertices in K_5^3, as shown in Fig. 6.

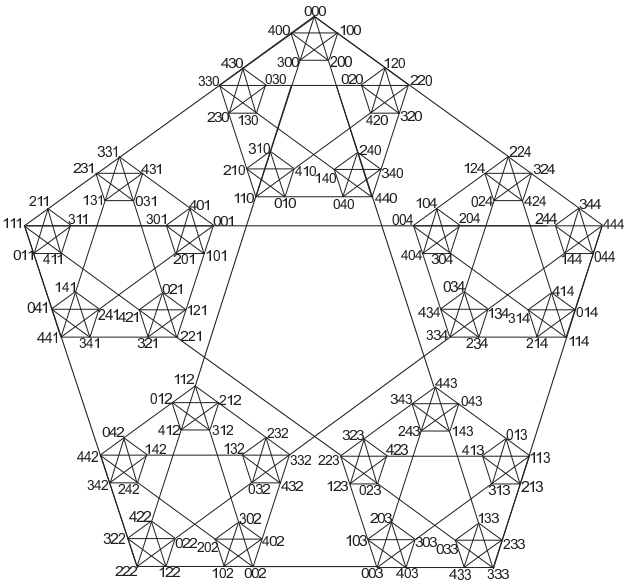


Fig. 6 The labeling of K_5^3 for Generalized Towers of Hanoi with 3 disks and 5 towers.

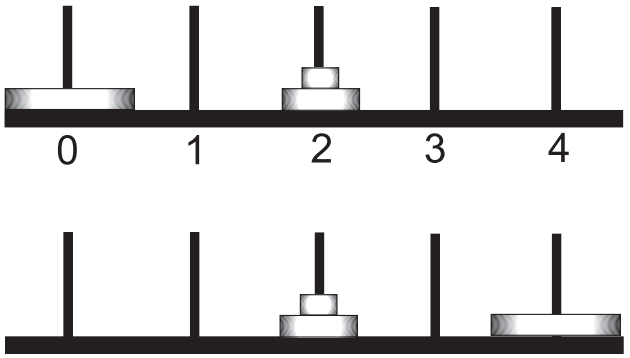


Fig. 7 Configurations corresponding to labels 220 and 224 on K_5^3. The largest disk may move between towers 0 and 4.

5.1 Algorithms to Solve The Generalized Tower Puzzles

Here we will give several algorithms for the Generalized Towers of Hanoi puzzle and see that this puzzle is basically as simple as the traditional Towers of Hanoi puzzle.

5.1.1 Recursive Algorithm

The goal in this puzzle is to move all n disks from tower 0 to tower $d-1$. Define the first tower as 0 and the second tower as 1 and so on until the last tower, $d-1$. The largest disk can only move to tower $d-1$ if all the smaller disks are stacked together on tower a where $(2a-0) \bmod d = d-1$. In the algorithm, rather than starting at tower 0 and going to tower $d-1$, the algorithm will start with tower i and end at tower j. So, the $n-1$ disks must move to tower a where $(2a-i) \bmod d = j$. From these observations, the recursive algorithm is:

RECURSIVE ALGORITHM

```
PROCEDURE HANOI(i, j, n)
IF n > 0 THEN k = [(i + j)/2] mod d
        HANOI (i, k, n − 1)
        move the top disk from tower i to tower j
        HANOI(k, j, n − 1)
```

Where the inputs i and j represent the source and the destination, respectively, and n is the number of disks that will move from the source to the destination.

5.2 Iterative Algorithm

In the traditional puzzle, the smallest disk was always moved one tower, but that was forced since there were only three towers arranged in a circle. The following Theorem tells us how to move the smallest disk in the generalized puzzle and hence gives us the piece of information we need to construct an iterative algorithm for this puzzle.

Theorem 3. (Equal Increments) *When disk 1 is moved in the solution of moving n disks from tower A to tower B, disk 1 is always moved by the same increment* (*in say, the "clockwise" direction*) *and this increment is* $(B-A)/2^{n-1}$.

Proof. Clearly, for $n = 1$, the smallest disk is moved from A to B which is an increment of $(B-A)/2^{1-1}$. For $n > 1$, the disks are moved in the pattern

$$AA\ldots A \implies At\ldots t \to Bt\ldots t \implies BB\ldots B$$
$$\text{where} \quad t = (A+B)/2,$$

here, $\implies$ indicates a sequence of moves and $\to$ indicates a single move. In the first sequence $n-1$ disks are being moved from from A to $(A+B)/2$ and by assumption these moves use an increment of

$$\left(\frac{(A+B)}{2} - A\right)/2^{n-2} = (B-A)/2^{n-1}.$$

In the second sequence $n-1$ disks are being moved from from $(A+B)/2$ to B and by assumption these moves use an increment of

$$\left(B - \frac{(A+B)}{2}\right)/2^{n-2} = (B-A)/2^{n-1}.$$

Since the increment for each half is the same, disk 1 is always moved by the same increment which is $(B-A)/2^{n-1}$. □

In the special case when the starting tower is tower 0, the increment is $(B)/2^{n-1}$. When the target tower is $d-1$, this simplifies to $(-1)/2^{n-1}$. For the traditional Towers of Hanoi, $d = 3$ and $\frac{1}{2} \equiv 2 \equiv -1$, and the Buneman-Levy algorithm will correctly move disk 1 when n is even, but when n is odd, disk 1 should be moved in the opposite direction.

ITERATIVE ALGORITHM

```
move smallest disk clockwise
     [B/2^(n−1)] mod d towers
WHILE a disk, other than the smallest,
      is able to move DO
         move that disk
         move smallest disk clockwise
              [B/2^(n−1)] mod d towers
ENDWHILE
```

Proposition 5.1. *The iterative algorithm HANOI ITERATIVE correctly moves n disks from Tower* 0 *to Tower B. More generally, this algorithm moves n disks from Tower A to Tower A + B.*

Proof. After $2^{n-1} - 1$ moves the algorithm has moved $n-1$

disks from tower 0 to tower $B/2$ because it is using the increment $\frac{B}{2}\frac{1}{2^{n-2}}$. At this point the n^{th} disk can be and is moved from tower 0 to tower B. Then the algorithm follows the same pattern as in the first $2^{n-1} - 1$ moves with increment $\frac{B}{2}\frac{1}{2^{n-2}}$ and this moves $n-1$ disks $\frac{B}{2}$ towers from their previous tower, i.e., to tower B. □

5.2.1 Counting Algorithm

From the iterative algorithm it is clear that every other move involves moving disk 1. This will help define the counting algorithm which involves using the counter to determine which disk should be moved. There are a few extra facts that we need:

(a) the rightmost 0 in BCOUNT tells us which disk to move,
(b) disk j is always moved by an increment which depends on j,
(c) the position of the disk to be moved can be determined from j and the position of disk 1.

We will defer proofs of these facts until we have defined the combination puzzles.

Counting Algorithm for Generalized Towers of Hanoi

```
PROCEDURE    TOWERS ( n )
     T : = 0 (Tower number computed modulo d )
     BCount : = 0 (BCount has n bits)
     P : = (−1)(1/2)^(n−1)  mod d
     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−n−1))  mod d
                       to T - (2^(b−n−1))  mod d
                       BCount : = BCount + 1
     ENDWHILE
```

6. Generalized Spin-Out Puzzles

There is an easy extension of Spin-Out to all dimensions which are powers of 2. The extended puzzles will retain the sliding aspect of Spin-Out, but the spinners will be replaced by pieces which consist of a stack of spinners. When a piece is composed of m spinners, it will have 2^m possible orientations, since each spinner can be in one of two orientations. For n pieces, there will be $(2^m)^n = d^n$ configurations. The sliding rules will determine which pieces can change, and new spinning rules will determine how the pieces can change. Together these rules define which configurations can change to which configurations.

- For dimension $d = 2^m$, each puzzle piece will consist of m spinners stacked one on top of the other.
- To find the orientation of piece j, write j as a binary number. To set a piece in this orientation, let the 1's (rightmost) bit represent the top spinner; a 0 bit means that it is horizontal, while a 1 bit means that it is vertical. Similarly, let the 2's bit represent the spinner just below the top spinner, the 4's bit the next spinner, etc. Continue in this manner; the 2^{m-1}'s bit will represent the bottom spinner.
- Thus for each $s \in \{0, \ldots, d-1\}$ there is a distinct orientation and corresponding binary number.

Example 6.1. *Suppose* $d = 8 = 2^3$. *That is,* $m = 3$, *so the pieces are composed of* 3 *spinners. Then, for example, the* $0 = 000_2$ *orientation consists of 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 with two vertical spinners above it.*

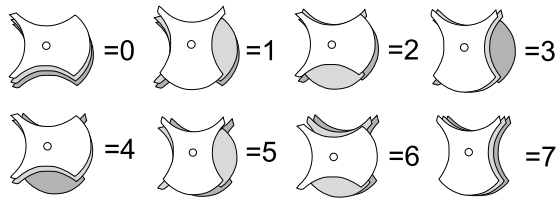


Fig. 8 Piece orientations for the Dimension 8 Puzzle.

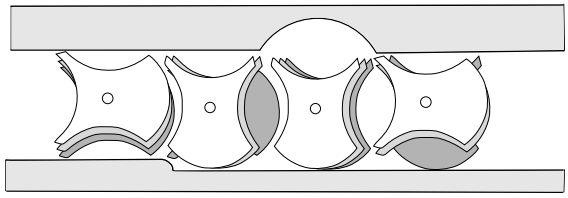


Fig. 9 An example configuration for the Dimension 8 Puzzle. The pieces are numbered left to right 1, 2, 3, and 4 and the configuration has label 0374.

Note that the 0 orientation will always consist of all horizontal spinners.

For an iteration n for $n \geq 1$, there will be n puzzle pieces. Call the leftmost piece the first piece and continue numbering the pieces from left to right. Thus the rightmost piece is the n^{th} piece. Given a configuration of the puzzle, there is a labeling with a string of characters from $\{0, \ldots, d-1\}$, where each piece 1 through n is represented by the number of the orientation it is in.

The rules of this puzzle are an extension of the rules of the Spin-Out Puzzle.

(1) The first piece may always change orientation, and may change to any other orientation.
(2) To spin at least one spinner of the j^{th} piece, s_1 through s_{j-2} must be 0 and $s_{j-1} \neq 0$; that is, pieces 1 through $j-2$ have 0 spin and piece $j-1$ has non-zero spin. If these conditions are satisfied, then move as many spinners of the j^{th} piece as possible; that is, any spinner that can switch between its horizontal and vertical positions must do so.

Example 6.2. *In* ***Fig. 9****, piece* 3 *is able to change orientation. Since the bottom spinner of piece* 2 *is horizontal, the bottom spinner of piece* 3 *cannot move. However, the other two spinners can move, and so they must become horizontal. Thus the orientation of piece* 3 *must change from* 7 *to* 4.

Figure 10 shows the labeled graph corresponding to the Generalized Spin-Out puzzle with 3 spinners in each stack. The first graph is for a puzzle with only one piece, so all moves are possible. The second graph is for a two piece puzzle, and not every move is possible.

6.1 Algorithms for Generalized Spin-Out

In the Spin-Out puzzle there are n spinners and 2 possible orientations for each spinner. In the Dimension 2^m puzzle there are n stacks of spinners and 2^m possible orientations for each stack. Thus the Dimension 2^m puzzle is simply an extension of the Spin-Out puzzle. An algorithm to solve the Dimension 2^m puzzle will

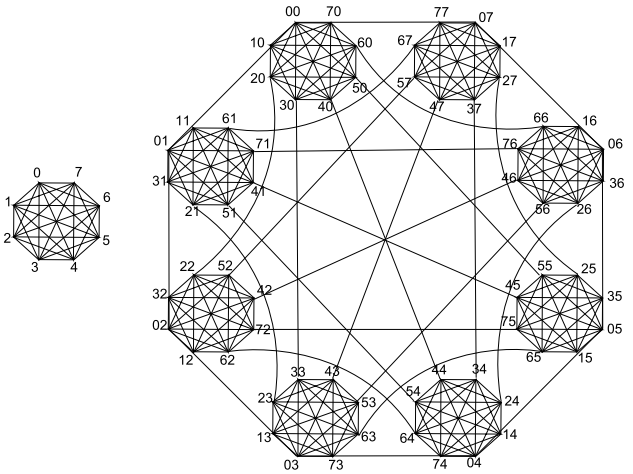


Fig. 10 The labeling for the first and second iterations for the dimension 8 graph, corresponding to the extended Spin-Out puzzles with 1 and 2 pieces respectively.

therefore be similar to an algorithm to solve the Spin-Out puzzle.

6.1.1 Recursive Algorithms

Pruhs [16] presents a two part recursive algorithm to solve the Spin-Out puzzle, which consists of moving the spinners from $11\ldots1$ to $00\ldots0$, with n spinners. A similar recursive algorithm can be written to solve the Dimension 2^m puzzle with n stacks of spinners and $d = 2^m$ possible orientations for each stack. To solve the puzzle move the stacks of spinners from $(d-1)(d-1)\ldots(d-1)$ to $00\ldots0$. Let FLIP i mean to rotate spinner number i from $d-1$ to 0 or from 0 to $d-1$, where $i \in \{1,\ldots,n\}$, and the stacks of spinners are indexed from 1 to n from right to left. In the following, we will use the spins in the order $s_n, s_{n-1}, \cdots, s_1$, and to make the states more pictorial, we use $\uparrow$ for spin 1 and $\leftarrow$ for spin 0.

Mutual Recursion Algorithm for Generalized Spin-Out

```
PROCEDURE ToZ(n) { Takes ↑...↑ to ←···← }
     IF n > 0 THEN   ToZ(n-2)
                     FLIP n
                     To1(n-2)
                     ToZ(n-1)
PROCEDURE To1(n) { Takes ←···← to ↑...↑ }
     IF n > 0 THEN   To1(n-2)
                     ToZ(n-2)
                     FLIP n
                     To1(n-2)
```

Notice that both these procedures are designed to do nothing when $n \leq 0$. Specifically, ToZ(2) calls ToZ(0) which does nothing, FLIPs spinner 2, calls To1(0) which does nothing, and finally calls ToZ(1) which FLIPs spinner 1. A simple induction suffices to show correctness of this pair of procedures.

Instead of the these two procedures which call one another, we can design an algorithm with nested recursion in which one procedure calls another procedure and the second procedure can call itself.

Nested Recursion Algorithm for Generalized Spin-Out

```
PROCEDURE SOLVE(n)
     { Takes ↑...↑ to ←···← }
     IF n > 0 THEN   SOLVE(n-2)
                     FLIP n
                     SUB(n-1)

PROCEDURE SUB(k)
     { Takes ↑←···← to ←←···←}
     { Takes ←←···← to ↑←···← }
     IF k > 0 THEN   SUB(k-1)
                     FLIP k
                     SUB(k-1)
```

As in the Mutual Recursion, these procedures do nothing when called with input less than 1. It is interesting to note that SUB is its own inverse. Again, the correctness can be proved by induction, that is showing that SUB is correct and then showing that SOLVE is correct.

6.1.2 Iterative Algorithms

The iterative algorithms are very simple. Notice that the algorithms really treat a stack of spinners as a single spinner because FLIP changes the spin of all the spinners in the stack. (Actually, FLIP could change between spin 0 and any other fixed spin, but it would still be treating a stack of spinners as a single spinner.) This is the reason that we did not give iterative algorithms for Spin-Out, i.e., the algorithms for Spin-Out would be identical to the algorithms for Generalized Spin-Out.

ITERATIVE Algorithm for Generalized Spin-Out

```
     { Takes ↑...↑ to ←···← }

IF n is ODD THEN FLIP 1
WHILE a spinner other than 1 can be moved DO
               FLIP that spinner
               FLIP 1
ENDWHILE
```

This first algorithm solves Generalized Spin-Out by taking the puzzle from LOCKED to the UNLOCKED state. When $\leftarrow\cdots\leftarrow$ is reached, no spinner other than spinner 1 can be changed and so this algorithm terminates. It's easy to use induction to show that this algorithm is correct and uses $\lceil \frac{2}{3}(2^n - 1) \rceil$ moves.

ITERATIVE Algorithm for Generalized Spin-Out

```
     { Takes ←←···← to ↑←···← }
     { Takes ↑←···← to ←←···← }

FLIP 1
WHILE a spinner other than 1 can be moved DO
               FLIP that spinner
               FLIP 1
ENDWHILE
```

This second algorithm takes the puzzle from the UNLOCKED state to the state in which only the n^{th} is locked (or vice-versa). The loop terminates at $\uparrow\leftarrow\cdots\leftarrow$ (or at $\leftarrow\leftarrow\cdots\leftarrow$) because only piece 1 can be changed in these configurations. Induction

can be used to show correctness and also that this algorithm uses $(2^n - 1)$ moves.

6.1.3 Counting Algorithms

There are also simple algorithms for Generalized Spin-Out which give the correct move by simply keeping tack of the number of moves made. We will give both a count up and a count down algorithm

COUNT-UP Algorithm for Generalized Spin-Out

```
COUNT = 0 { n bit counter }
FLIP 1
COUNT = COUNT + 1
WHILE COUNT ≠ 1010... { i.e. ⌈2/3(2^n − 1)⌉ } DO
    j is the position of the rightmost 0 in COUNT
    FLIP j
    COUNT = COUNT + 1
ENDWHILE
```

This counting up algorithm takes $\leftarrow\leftarrow \cdots \leftarrow$ to $\uparrow\uparrow \ldots \uparrow$.
To go in the other direction, from $\uparrow\uparrow \ldots \uparrow$ to $\leftarrow\leftarrow \cdots \leftarrow$, the following counting down algorithm suffices.

COUNT-DOWN Algorithm for Generalized Spin-Out

```
COUNT = 1010... { i.e ⌈2/3 (2^n − 1)⌉ }
              { n bit counter }
WHILE COUNT ≠ 0 DO
    COUNT = COUNT − 1
    j is the position of the rightmost 0 in COUNT
    FLIP j
ENDWHILE
```

7. Combination Puzzle

In generalizing the Towers of Hanoi and Spin-Out puzzles, we found the iterative complete graphs which generalize the configuration spaces of these puzzles. But, when we constructed generalized puzzles, we only found puzzles which correspond to K_d^n for either d odd or d a power of 2. Our goal here is to form puzzles which correspond to $d = q \cdot 2^m$ where q is odd and m is a positive integer. The obvious idea is to make the new puzzles a "product" of a generalized Towers of Hanoi and a generalized Spin-Out puzzle. Taking a direct product makes the size of the configuration space come out right, but we have to decide how to "combine" moves from the two component puzzles.

In the product graph we want an edge from (v, z) to (u, w) exactly when there is an edge from v to u in the first graph and an edge from z to w in the second graph. This might seem to cause some difficulties because while most vertices in a Towers of Hanoi graph have degree q and most vertices in a Spin-Out graph have degree 2^m, there are some vertices with degree $q - 1$ or $2^m - 1$ in these respective graphs. We take care of this difficulty by declaring that each *corner* vertex has a self-edge which corresponds to a *null* move. Then each vertex in the product graph will have degree $d = q \cdot 2^m$. Notice that the corner vertices of the product graph have a self-edge corresponding to the self-edges on the corner vertices in each of the factor graphs. This self-edge corresponds to a null move in the combination puzzle. There are also edges which correspond to a self-edge in one factor and not in the other factor. These edges correspond to a move in the combination puzzle which allows a move in one of the component puzzles and not in the other component puzzle.

REMARK: We should remark that this product is NOT the Cartesian product of graphs defined in Ref. [11]. In that "Cartesian" product the number of edges is $n_1E_2 + n_2E_1$ where the n's are the number of vertices and the E's are the number of edges in the factor graphs. So, for example if we took the "Cartesian" product of K_3 and K_2 we would get

$$3 \cdot 1 + 2 \cdot 3 = 9 \quad \text{edges.}$$

This is less than the 15 edges, not counting self-edges, that appear in K_6 which is our product of K_3 and K_2.

Proposition 7.1. *The "product" of K_q^n and K_p^n is K_{qp}^n, and in particular, the "product" of K_q^n and $K_{2^m}^n$ is $K_{q2^m}^n$.*

Proof. It is easy to check that the product of K_q and K_p is K_{qp}. Since K_q^n and K_p^n are iterated complete graphs they decompose into q copies of K_q^{n-1} and p copies of K_p^{n-1}. Our "product" takes the direct products of both the vertex sets and the edge sets. So our product certainly contains products of each pair of these subgraphs, i.e., each K_q^{n-1} with each K_p^{n-1}. Each of these subproducts is a copy of K_{qp}^{n-1} by inductive hypothesis. There are also extra edges joining the copies of K_q^{n-1} and joining the copies of K_p^{n-1}. When these edges are crossed there are $q\,p$ edges which join the copies of K_{qp}^{n-1} which were previously constructed. Since each K_q^{n-1} has exactly one edge to every other copy of K_q^{n-1}, and similarly for K_p^{n-1}, a product of a pair of these edges becomes a single edge from a copy of K_{qp}^{n-1} to a copy of K_{qp}^{n-1}. This gives a unique edge between each of the copies of K_{qp}^{n-1} because there was only a single edge between each pair of copies of K_q^{n-1} in K_q^n and a single edge between each pair of copies of K_p^{n-1} in K_p^n. So the product graph satisfies the recursive definition of K_{qp}^n. □

7.1 The Combination Puzzle

Our combination puzzle is a direct product of a generalized Towers of Hanoi puzzle and a generalized Spin-Out puzzle. Let a configuration be

$$\begin{pmatrix} t_n & t_{n-1} & \cdots & t_2 & t_1 \\ s_n & s_{n-1} & \cdots & s_2 & s_1 \end{pmatrix},$$

then the allowed moves are

(a) piece 1 can change at any time in any way
so the result of a piece 1 move is

$$\begin{pmatrix} t_n & t_{n-1} & \cdots & t_2 & x \\ s_n & s_{n-1} & \cdots & s_2 & y \end{pmatrix},$$

where $x \in \{0, \cdots, q-1\}$ and $y \in \{0, \cdots, 2^m - 1\}$ but otherwise x and y are arbitrary,

(b) piece k can move if
(1) all of the "smaller" pieces are on a single tower,
i.e., $t_{k-1} = t_{k-2} = \cdots = t_1$
(2) the first $k-2$ stacks of spinners all have spin 0,
i.e., $s_{k-2} = s_{k-3} = \cdots = s_1 = 0$
the new configuration will be

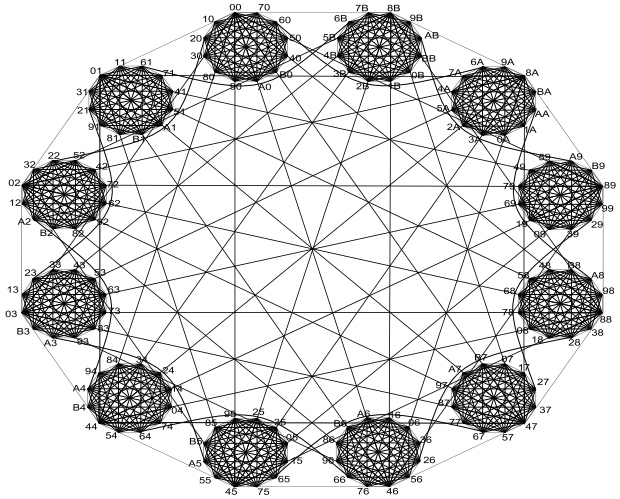


Fig. 11 The Puzzle Labeling for K_{12}^2, notice in the labeling that ten is represented by an A and eleven is represented by a B. The labels are written in the order $x_1\, x_2$.

$$\begin{pmatrix} t_n & \cdots & t_{k+1} & 2t_1 - t_k & t_1 & t_1 & \cdots & t_1 \\ s_n & \cdots & s_{k+1} & s_k \oplus s_{k-1} & s_{k-1} & 0 & \cdots & 0 \end{pmatrix}.$$

In moving piece k the conditions for a change of tower and a change of spin are **both** satisfied. Notice that the spin of piece k does **not** change if $s_{k-1} = 0$. The allowed moves for piece 2 may be a little confusing. If the configuration of the first two pieces is $\left(\begin{smallmatrix} t_2 & t_1 \\ s_2 & s_1 \end{smallmatrix}\right)$ then an allowed move changes this to $\left(\begin{smallmatrix} 2t_1 - t_2 & t_1 \\ s_2 \oplus s_1 & s_1 \end{smallmatrix}\right)$. If $s_1 = 0$, piece 2 can change tower without changing spin. If $t_2 = t_1$ and $s_1 \neq 0$, piece 2 can change spin and not change tower. Of course, if $t_2 = t_1$ and $s_1 = 0$, no change of piece 2 will occur.

7.2 State Space

Proposition 7.2. *The state space of our Combination puzzles is K_d^n, where n is the number of pieces.*

Proof. Consider a one piece puzzle. All moves are possible, so the state space is K_d^1 for some d. Assume that the state space is K_d^{n-1} for an $n-1$ piece puzzle. Consider the n^{th} piece. Since the n^{th} piece has d possible settings, the state space contains d copies of K_d^{n-1}.

In each copy, there are $d-1$ configurations in which piece n can change state and each of these results in a distinct state for piece n. Let $d = q\,2^m$ for some odd q. The changes for piece n allowed in our puzzle are

$$\begin{pmatrix} t_n & t & t & \cdots & t \\ s_n & s_{n-1} & 0 & \cdots & 0 \end{pmatrix} \longrightarrow \begin{pmatrix} 2t - t_n & t & t & \cdots & t \\ s_n \oplus s_{n-1} & s_{n-1} & 0 & \cdots & 0 \end{pmatrix}.$$

Two of these changes are the same only if

$$2\,t - t_n = 2\,t' - t_n$$
$$s_n \oplus s_{n-1} = s_n \oplus s'_{n-1}$$

but these equations imply that $t = t'$ and $s_{n-1} = s'_{n-1}$. So, all these changes are distinct. The number of changes is $q \cdot 2^m$, i.e., the number of distinct t's times the number of distinct s_{n-1}'s. This is d rather than $d-1$, but if $t = t_n$ and $s_{n-1} = 0$ the n^{th} piece stays in the same state, and so, there are really only $d-1$ moves.

So, the subgraphs are tied together in exactly the manner required to construct K_d^n from d copies of K_d^{n-1}. □

Figure 11 displays K_{12}^2 labeled with the configurations of a combination puzzle with 2 pieces. In this figure, $d = 12 = 3 \cdot 2^2$, so there are 3 towers and each piece consists of a stack of 2 spinners.

8. Solving the Combination Puzzle

In the Combination puzzle, a generalized Towers of Hanoi and a generalized Spin-Out puzzle are tied together so that the two component puzzles work in parallel. But, there are some special situations in which one of the component puzzles can change without affecting the other component. Specifically, if all of the pieces are on a single tower, then the spins, the Spin-Out part, can be changed without changing the tower positions, the Towers of Hanoi part. In symbols, if the puzzle configuration is $\begin{smallmatrix} t & t & \cdots & t \\ s_n & s_{n-1} & \cdots & s_1 \end{smallmatrix}$, then the s's can change according to the rules of generalized Spin-Out, and for any change in the j^{th} piece, $2\,t_{j-1} - t_j = 2t - t = t$ and the j^{th} piece does not change tower. (In the special case when $j = 1$, s_1 can be changed to an arbitrary value and the piece can still stay on tower t.)

On the other hand, if the spin of every piece is 0 then the pieces can be moved between towers without changing the spins. In symbols, if the puzzle configuration is $\begin{smallmatrix} t_n & t_{n-1} & \cdots & t_1 \\ 0 & 0 & \cdots & 0 \end{smallmatrix}$, then if the j^{th} piece is moved between towers, $s_j \oplus s_{j-1} = 0 \oplus 0 = 0$ and the spin of the piece does not change. (Again, in the special case, the first piece can always be moved without changing its spin.)

The above suggests that solving the Combination puzzle is "serializable," that is, the n pieces can be moved from the start tower to the target tower with all spins remaining at 0, and then the spins can be changed without moving the pieces. This serial solution has the unfortunate property that it uses $2^n - 1 + \lceil \frac{2}{3}(2^n - 1) \rceil$ moves, and this is more moves than necessary. (In Theorem 2 we showed that vertices, and therefore configurations, are at a distance of at most $2^n - 1$ moves.)

So, we want to find "parallel" solution methods. To do so, we first have to ask what it means to solve a Combination puzzle. Obviously, we want to move n disks from a start tower to a target tower, but for the spins, we seem to have several choices. We could start with all spins 0 and change all of the spins to a specific spin value, say spin 1. In the following, we will use spin 1 with the understanding that 1 could be replaced by any non-zero value. In particular, spin 1 could stand for spin $2^m - 1$. Conversely, we could start with all spins 1 and then change to all spins 0. These two choices have the unfortunate property that they take only $\lceil \frac{2}{3}(2^n - 1) \rceil$ moves which will conflict with the $2^n - 1$ needed to move the pieces between towers. One way out is to start with spins $0\,0 \cdots 0$ and go to spins $1\,0 \cdots 0$ (or vice-versa) because these spin configurations are $2^n - 1$ moves apart. Another way is to sometimes change a piece's tower without changing the piece's spin. This will require maneuvering the puzzle into a configuration in which the spins do not change.

As with the other puzzle, we expect recursive, iterative, and counting algorithms for the combination puzzle. But first, we have to decide what should be the starting configuration and what should be the target configuration. Three special types of configurations suggest themselves:

$$\begin{matrix} A & A & \cdots & A \\ 0 & 0 & \cdots & 0 \end{matrix} \quad \text{and} \quad \begin{matrix} B & B & \cdots & B \\ 1 & 1 & \cdots & 1 \end{matrix}$$

$$\text{and} \quad \begin{matrix} C & C & \cdots & C \\ 1 & 0 & \cdots & 0 \end{matrix}.$$

Here A, B, and C are distinct names for towers, 0 means all of the spinners are in one orientation, and 1 means that the spinners are in an orientation different from from the 0 orientation. Moving from the first to the second of these configuration (or vice-versa) would mean solving both a Towers of Hanoi puzzle and a Spin-Out puzzle by moving all of the pieces from one tower to another and changing all of the spins from one orientation to another orientation. Moving from the first to the third configuration is easier to deal with because these configurations are *corners* of an iterated complete graph. Also, moving from the first to the third configuration will be useful as a subproblem in moving between the first and second configurations.

Eventually, we want to say that the solution to the combination puzzle is changing the initial configuration $(0\ 0\ \cdots\ 0)$ to the target configuration $(d-1\ d-1\ \cdots\ d-1)$ where we map the two component pairs $\binom{t}{s}$ to $2^m t + s$. So our initial configuration will be $\begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{pmatrix}$ and the target configuration will be $\begin{pmatrix} q-1 & q-1 & \cdots & q-1 \\ 2^m-1 & 2^m-1 & \cdots & 2^m-1 \end{pmatrix}$. This will be a problem of changing a type 1 configuration to a type 2 configuration where $B = q-1$ and 1 stands for $2^m - 1$.

8.1 Algorithms for Configurations One and Three

Here, we will give recursive, iterative, and counting algorithms to take the combination puzzle between configurations of types One and Three.

8.1.1 Recursive Algorithm

The following recursive algorithm solves the problem of moving from the first configuration to the third configuration. Nicely enough it also solves the problem of moving from the third configuration to the first configuration. To make the procedure succinct, we will only pass the names of the towers and the number of pieces as the procedure's parameters. We will not pass the spins of the pieces because pieces 1 through $n-1$ are assumed to have spin 0 and piece n will have spin 1 or 0 depending on which direction we are going between configuration three and configuration one. We will use the instruction MOVE(A, C) to mean: move the indicated piece from tower A to tower C. We will use the instruction FLIP to mean: change the spin of the indicated piece from 0 to 1 or from 1 to 0.

RECURSIVE Algorithm for Combination Puzzle: Configurations One and Three

$\begin{smallmatrix} A & A & A & \cdots & A \\ 0 & 0 & 0 & \cdots & 0 \end{smallmatrix}$ to $\begin{smallmatrix} C & C & C & \cdots & C \\ 1 & 0 & 0 & \cdots & 0 \end{smallmatrix}$
or $\begin{smallmatrix} A & A & A & \cdots & A \\ 1 & 0 & 0 & \cdots & 0 \end{smallmatrix}$ to $\begin{smallmatrix} C & C & C & \cdots & C \\ 0 & 0 & 0 & \cdots & 0 \end{smallmatrix}$

```
PROCEDURE SOL( A, C, n)
    IF n > 0 THEN
            t = (A + C)/2 mod q
            SOL( A, t, n − 1 )
            MOVE( A, C ) piece n
            FLIP piece n
            SOL( t, C, n − 1 )
```

Proposition 8.1. *This algorithm takes* $\begin{smallmatrix} A & A & A & \cdots & A \\ 0 & 0 & 0 & \cdots & 0 \end{smallmatrix}$ *to* $\begin{smallmatrix} C & C & C & \cdots & C \\ 1 & 0 & 0 & \cdots & 0 \end{smallmatrix}$ *and takes* $\begin{smallmatrix} A & A & A & \cdots & A \\ 1 & 0 & 0 & \cdots & 0 \end{smallmatrix}$ *to* $\begin{smallmatrix} C & C & C & \cdots & C \\ 0 & 0 & 0 & \cdots & 0 \end{smallmatrix}$.

The proof of correctness is a straight-forward induction starting from the observation that for $n = 1$, SOL takes $\begin{smallmatrix} A \\ 0 \end{smallmatrix}$ to $\begin{smallmatrix} C \\ 1 \end{smallmatrix}$ and takes $\begin{smallmatrix} A \\ 1 \end{smallmatrix}$ to $\begin{smallmatrix} C \\ 0 \end{smallmatrix}$. Notice that this is essentially the recursive algorithm for Towers of Hanoi.

8.1.2 Iterative Algorithm

An Iterative algorithm for this problem is also easy.

ITERATIVE Algorithm for Combination Puzzle: Configurations One and Three

$\begin{smallmatrix} A & A & A & \cdots & A \\ 0 & 0 & 0 & \cdots & 0 \end{smallmatrix}$ **to** $\begin{smallmatrix} C & C & C & \cdots & C \\ 1 & 0 & 0 & \cdots & 0 \end{smallmatrix}$ **or** $\begin{smallmatrix} A & A & A & \cdots & A \\ 1 & 0 & 0 & \cdots & 0 \end{smallmatrix}$ **to** $\begin{smallmatrix} C & C & C & \cdots & C \\ 0 & 0 & 0 & \cdots & 0 \end{smallmatrix}$

```
INCREMENT = [(C − A)/2^(n−1)] mod d
move piece 1 INCREMENT towers clockwise
FLIP 1
WHILE a piece, other than piece 1,
      is able to move DO
      change that piece
      { move it and flip it }
      move piece 1 INCREMENT towers clockwise
      FLIP 1
ENDWHILE
```

This is really the Towers of Hanoi algorithm working in parallel with a Spin-Out algorithm which changes $\leftarrow\leftarrow\ \cdots\ \leftarrow$ to $\uparrow\leftarrow\ \cdots\ \leftarrow$. Of course, this Spin-Out algorithm also changes $\uparrow\leftarrow\cdots\leftarrow$ to $\leftarrow\leftarrow\cdots\leftarrow$.

Note that each piece is BOTH moved and flipped. This is required as can be shown by following the recursive algorithm. Correctness of this ITERATIVE algorithm follows from the recursive algorithm and the fact that at most one piece other than piece 1 can be moved in any configuration. Termination occurs when all pieces are on a single tower and hence no piece other than piece 1 can be moved.

8.1.3 Counting Algorithm

A counting algorithm for this problem is also easy.

COUNTING Algorithm for Combination Puzzle: Configurations One and Three

$\begin{smallmatrix} A & A & \cdots & A \\ 0 & 0 & \cdots & 0 \end{smallmatrix}$ to $\begin{smallmatrix} C & C & \cdots & C \\ 1 & 0 & \cdots & 0 \end{smallmatrix}$ or $\begin{smallmatrix} A & A & \cdots & A \\ 1 & 0 & \cdots & 0 \end{smallmatrix}$ to $\begin{smallmatrix} C & C & \cdots & C \\ 0 & 0 & \cdots & 0 \end{smallmatrix}$

```
COUNT = 0 { n bit counter }
T = A
P = (C − A)/2^(n−1) mod q
MOVE (T, T + P) piece 1
FLIP 1
T = T + P
COUNT = COUNT + 1
WHILE COUNT ≠ 11···1 DO
     j is the position of the
              rightmost 0 in COUNT
     MOVE piece j
     FLIP j
```

```
        COUNT = COUNT + 1
        MOVE (T, T + P) piece 1
        T = T + P
        FLIP 1
        COUNT = COUNT + 1
    ENDWHILE
```

Notice that if we restrict this algorithm to the special case where A = 0 and C = $q-1$ and the spins are all initially 0, then the start configuration is $0^{(n)}$, i.e., all zeros, and if we use FLIP to mean change ALL of the spinners in a stack, then the target configuration will be $(d-1)[2^m(q-1)]^{(n-1)}$, all pieces on tower $q-1$, piece n having spin 2^m-1 and all of the other pieces having spin 0.

8.2 Algorithms for Configurations Two and One

The above algorithms are essentially a Towers of Hanoi algorithm and a Spin-Out algorithm being run in parallel. This parallel execution works because moving n disks from one tower to another in Towers of Hanoi takes 2^n-1 moves, and changing $\leftarrow\leftarrow\cdots\leftarrow$ to $\uparrow\leftarrow\cdots\leftarrow$ in Spin-Out also takes 2^n-1 moves. But when changing configurations One to Two or vice-versa, only $\lceil 2/3\,(2^n-1)\rceil$ moves are needed in the Spin-Out component while 2^n-1 moves are needed in the Towers of Hanoi component. Thus, to run these two algorithms in parallel, we will need a *switch* which can turn off the Spin-Out component while the algorithm is still making progress in the Towers of Hanoi component.

8.2.1 Recursive Algorithms

For recursive algorithms the switch is not explicit. Instead we use previously constructed algorithms as subroutines. In SOLVE, we use both SOL and HANOI. SOL flips and moves pieces, while HANOI only moves pieces without flipping them.

RECURSIVE Algorithm for Combination Puzzle: Configurations One and Two

$\begin{smallmatrix}A&A&\cdots&A\\0&0&\cdots&0\end{smallmatrix}$ to $\begin{smallmatrix}C&C&\cdots&C\\1&1&\cdots&1\end{smallmatrix}$

```
PROCEDURE SOLVE( A, C, n )
    IF n > 0 THEN
            t = (A + C)/2 mod q
            SOL( A, t, n − 1 )
            MOVE( A, C ) piece n
            FLIP piece n
            t1 = (t + C)/2 mod q
            HANOI( t, t1, n − 2 )
            MOVE( t, C ) piece n − 1 {If n > 1}
            SOLVE( t1, C, n − 2 )
```

To go the other way between these configurations we can use algorithmm SOLVE2. In SOLVE2, we again use both SOL and HANOI as subroutines.

RECURSIVE Algorithm for Combination Puzzle: Configurations Two and One

$\begin{smallmatrix}C&C&\cdots&C\\1&1&\cdots&1\end{smallmatrix}$ to $\begin{smallmatrix}A&A&\cdots&A\\0&0&\cdots&0\end{smallmatrix}$

```
PROCEDURE SOLVE2( C, A, n )
    IF n > 0 THEN
            t = (A + C)/2 mod q
            t1 = (t + C)/2 mod q
            SOLVE2( C, t1, n − 2 )
            MOVE( C, t ) piece n − 1
            HANOI( t1, t, n − 1 )
            MOVE( C, A ) piece n
            FLIP n
            SOL( t1, C, n − 2 )
```

8.2.2 Iterative Algorithm

ITERATIVE Algorithm for Combination Puzzle: Configurations One and Two

$\begin{smallmatrix}A&A&\cdots&A\\0&0&\cdots&0\end{smallmatrix}$ to $\begin{smallmatrix}C&C&\cdots&C\\1&1&\cdots&1\end{smallmatrix}$

```
SWITCH = ON
TEST = n
INC = (C − A)/2^(n−1) mod q
move piece 1 INC towers (clockwise)
IF SWITCH = ON THEN FLIP 1
WHILE piece j  (j ≠ 1) can be moved DO
      CHANGE piece j (move and flip)
      { if s_(j−1) = 0 this flip does not change s_j }
      IF j = TEST THEN Complement SWITCH
      TEST = TEST - 1
      move piece 1 INC towers (clockwise)
      IF SWITCH = ON THEN FLIP 1
ENDWHILE
```

In this Iterative algorithm, the switch is explicit.

This algorithm changes

$$\begin{matrix}A&A&\cdots&A\\0&0&\cdots&0\end{matrix} \text{ to } \begin{matrix}C&C&\cdots&C\\1&1&\cdots&1\end{matrix}.$$

This algorithm can be modified to change

$$\begin{matrix}C&C&\cdots&C\\1&1&\cdots&1\end{matrix} \text{ to } \begin{matrix}A&A&\cdots&A\\0&0&\cdots&0\end{matrix}.$$

The needed modifications are:

(a) change the inialization of TEST to: TEST = 2
(b) TEST = TEST + 1 (rather than TEST = TEST - 1)
(c) after SWITCH = ON insert IF n is even THEN SWITCH = OFF.

8.2.3 Counting Algorithm

The Iterative algorithm can be converted into a Counting algorithm by specifing the moves that need to be made.

COUNTING Algorithm for Combination Puzzle: Configurations One and Two

$\begin{smallmatrix}A&A&\cdots&A\\0&0&\cdots&0\end{smallmatrix}$ to $\begin{smallmatrix}C&C&\cdots&C\\1&1&\cdots&1\end{smallmatrix}$

```
T = A
COUNT = 0 { n bits }
SWITCH = ON
TEST = n
INC = ( C - A )/2^(n−1) mod q
MOVE( T, T + INC ) piece 1
IF SWITCH = ON THEN FLIP 1
```

```
COUNT = COUNT + 1
T = T + INC
WHILE COUNT ≠ 1 ··· 1 { i.e.  2^n − 1 } DO
     j is the position of the rightmost 0 in COUNT
     STEP = INC · 2^(j−2) mod q
     MOVE( T - STEP, T + STEP ) piece j
     IF SWITCH = ON THEN FLIP j
     IF j = TEST THEN Complement SWITCH;
                      TEST = TEST - 1
     MOVE( T, T + INC ) piece 1
     IF SWITCH = ON THEN FLIP 1
     T = T + INC
     COUNT = COUNT + 2
ENDWHILE
```

As for the Iterative algorithm, this Counting algorithm can be modified to change

$$\begin{array}{cccc} C & C & \cdots & C \\ 1 & 1 & \cdots & 1 \end{array} \quad \text{to} \quad \begin{array}{cccc} A & A & \cdots & A \\ 0 & 0 & \cdots & 0 \end{array}.$$

The needed modifications are:

(a) change the inialization of TEST to: TEST = 2
(b) TEST = TEST + 1 (rather than TEST = TEST − 1)
(c) after SWITCH = ON insert
IF n is even THEN SWITCH = OFF.

9. Justification for the Counting Algorithms

For the counting algorithms, we make use of the RULE:

Move disk j where j is the position of the rightmost 0 in COUNT.

The following is the justification for this rule.

Theorem 4. (*Rightmost Zero Rule for* **Towers**.) *In* (*generalized*) *Towers of Hanoi the rightmost zero in the counter specifies which piece to move.*

Proof. From the recursive algorithm, after $2^{n-1}-1$ moves $n-1$ pieces have been moved from the starting tower to an intermediate tower, and then the n^{th} piece is moved. Of course $2^{n-1}-1$ is $011\cdots1$ in binary and the rightmost zero is in position n specifying correctly that the n^{th} is to be moved, and then the counter is incremented to $100\cdots0$. The first and last $2^{n-1}-1$ moves involve only the smaller $n-1$ disks and by hypothesis these moves obey the rule. As a base, for $n = 1$, the counter initially contains 0 and so the rightmost zero is in position 1 and the first piece is moved. (For $n = 0$, the claim is vacuously true.) □

The rightmost zero rule also holds for generalized Spin-Out. In the following to "flip" a piece in spin state s we mean that the piece's new state is $s \oplus s'$ where s' is assigned a fixed non-zero value.

Theorem 5. (*Rightmost Zero Rule for* **Spinners.**) *In* (*generalized*) *Spin-Out with staring configuration* $s00\cdots0$ *the rightmost zero in the counter specifies which piece to flip.*

Proof. Clearly, the first move flips piece 1 and the counter which starts at all zeros has its rightmost 0 in the first position. After $2^{n-1}-1$ moves, the configuration will be $s\,s'\,0\cdots0$ and the rightmost zero in the counter will be in the n^{th} position and the n^{th} piece is flipped. After the counter is incremented, it contains $100\cdots0$ and the configuration is $s \oplus s'\ s'\ 0\ \ldots\ 0$. So, the procedure continues as if it had only $n-1$ pieces with starting configuration $s'\ 0\ \ldots\ 0$, and by assumption this reaches $s' \oplus s'\ 0\ \ldots\ 0$ (which is $0\ 0\ \ldots\ 0$) when the $n-1$ bit counter has only 1's. But, for the n piece puzzle, this is configuration $s \oplus s'\ 0\ 0\ \ldots\ 0$ when the n bit counter contains only 1's. □

This rightmost rule does **not** hold for **all** initial configurations of Spin-Out. In particular, if the puzzle starts in the configuration $11\cdots1$ and the counter starts with all zeros, the rule does not hold. In our counting algorithms we solve this problem by starting the counter at the value it would have had if the starting configuration would have been one of the configurations which obeys the rule.

We also need a justification for the moves our counting algorithms make. First, we show how to calculate how many towers a piece should be moved.

Theorem 6. *In the minimal move solution to moving n disks from tower A to tower C, disk j always moves* $(C-A)/2^{n-j} \bmod q$ *towers clockwise.* (*This is* $2^{j-1}\cdot INC$, *where* $INC = (C-A)/2^{n-1} \bmod q$ *is the increment always moved by disk 1.*)

Proof. From the recursive algorithm for generalized Towers of Hanoi, if $j = n$ then the only move of disk n is from A to C which uses an increment of $(C-A)/2^{n-n}$. Again from the algorithm, the first half is a Towers of Hanoi with $n-1$ disks in which each disk is moved from A to $(A+C)/2$, and by inductive assumption, the increment for disk j is $[(A+C)/2 - A]/2^{n-1-j}$ which equals $(C-A)/2^{n-j}$. For the second half, disk j is moved from $(A+C)/2$ to C, and by inductive assumption, the increment for disk j is $[C - (A+C)/2]/2^{n-1-j}$ which again equals $(C-A)/2^{n-j}$. □

Now we need to show where disk j is and where it has to be moved to. When piece j is moved the Towers of Hanoi component is $t_n, \cdots, t_{j+1}, t_j, t, \cdots, t$, i.e., all smaller pieces on tower t which is the tower that holds piece 1. The new tower for t_j is $2t - t_j$ and since t_j is moved by INC_j (see the above theorem), we have

$$\begin{aligned} &2t - t_j = t_j + INC_j \\ &\text{giving} \quad t_j = t - \frac{1}{2}INC_j \\ &\text{and} \quad 2t - t_j = t - \frac{1}{2}INC_j + INC_j = t + \frac{1}{2}INC_j. \end{aligned}$$

So our counting algorithms move the j^{th} piece correctly.

We should also comment on our use of $\frac{1}{2}$. Since we assume that q is an odd number, $\frac{1}{2} \equiv \frac{q+1}{2} \pmod q$, and $\frac{1}{2}$ can be computed with a simple integer division. (In binary, this would be a one bit shift.)

In several of the algorithms, we use $(C-A)/2^{n-1}$. This can be computed using 2^{n-1} divisions or from above 2^{n-1} multiplications. By using repeated squaring, we can cut this down to $O(n-1)$ multiplications. Notice that since these operations are mod q, we can assume that they take constant time independent of n. The other increments are all multiples of this basic increment, and since the number of multiplies times their frequency is

$j/2^j$, these operations take constant time in an amortized sense.

10. Time and Space Usage

Here, we will show that all of our puzzles have essentially the same complexity. Further, we show that the counting algorithms use both the minimum amount of time and the minimum amount of space. In the following, the subscripted c's are constants which are independent of n.

Theorem 7. *Each puzzle with n pieces has time complexity* $\Theta(2^n)$*, i.e., any algorithm for the puzzle executes at least* $c_1 2^n$ *operations, and at least one algorithm uses at most* $c_2 2^n$ *operations. In fact, all of our algorithms use* $O(2^n)$ *operations.*

Proof. A solution to an n piece puzzle will consist of making the moves or at least printing the moves. In either case, each move will require at least a constant amount of time or number of instructions. Since Theorem 2 and Theorem 1 show that $c_1 2^n$ moves are required for all our puzzles, the lower bound is established. For many of our algorithms, each move is calculated using a constant number of operations. The exception are our counting algorithms which can take up to n operations to increment the counter. But, this incrementing (and in some algorithms multiplication by powers of 2) will only take j steps in about $1/2^j$ of the moves, and since $\sum_{j=1}^{\infty} j/2^j$ converges, the total number of operations is bounded by $c_2 2^n$. □

Theorem 8. *Every algorithm for our puzzles requires* $n + c_3$ *bits of storage, and our counting algorithms use this minimal amount of storage.*

Proof. Since each puzzle requires $c_1 2^n$ moves, any algorithm which solves the puzzle must have at least $c_1 2^n$ internal states. Otherwise, an internal state would be repeated before the puzzle is solved, and the algorithm would be in a dead loop endlessly repeating the same instructions. The number of internal states is $c_4 2^B$ where B is the number of memory bits. To avoid a dead loop, $c_4 2^B \geq c_1 2^n$ is required, and so $B \geq n + c_3$ is necessary.

Each counting algorithms uses an n bit counter and a few other variables which only use a constant number of bits. □

For space usage, we should mention what our other algorithms use. While the recursive algorithms do not use any visible memory, the recursion stack uses $\Theta(n \log n)$ bits to store n stack frames each of which uses $\log n$ bits to represent the number of pieces being dealt with.

The iterative algorithms do not use explicit storage, but they need a representation of the puzzle to see which piece to move. Since a puzzle has d^n configurations, a representation of these configurations requires at least $n \log d$ bits of storage. We will not discuss various tricks which can be used to determine which piece to move without a full representation of the puzzle.

The short conclusion, here, is that the counting algorithms are best possible. They use the minimal amount of storage up to an additive constant, and the minimal number of operations up to a multiplicative constant.

11. Conclusion

We have shown that two familiar puzzles, Towers of Hanoi and Spin-Out, can be generalized and "crossed" to give an infinite family of puzzles, one for each $d \geq 2$. Each of these puzzles is really an infinite sequence of puzzles, one for each number of pieces $n \geq 1$. These puzzles correspond to the bi-infinite family of iterated complete graphs, K_d^n. Of course, every graph corresponds to a puzzle in which the vertices are the puzzle's configurations and the edges indicate allowed moves.

There are some things we did not do. Specifically, we did not discuss the "generalized" Towers of Hanoi in which piece j can be moved to any tower if pieces 1 through $j-1$ are on a single tower. This is a notorious problem, whose algorithm is presumably known, but whose proof has been lacking for 70 years [13]. There are many other variants of Towers of Hanoi, e.g., Ref. [3] which we also did not discuss.

For all of our puzzles, we presented a variety of recursive, iterative, and counting algorithms, and showed that while all of these algorithms were time optimal, only the counting algorithms were both time and space optimal.

Finally, we clarified the significance of the Gray code for these puzzles by showing that the Gray property corresponds to the stipulation that only one piece may be changed at a time, and showing that the Rightmost Rule applies to both Towers of Hanoi and Spin-Out and so which piece to move is given by the rightmost 0 in a binary counter which is also the position of the bit that is flipped while counting in the Gray code.

Acknowledgments Much of the work here was carried out by students in the REU Summer Program at Oregon State University in previous years. We would like to thank the following people for their contributions: Lindsay Baun, Sonia Chauhan, Elizabeth Skubak, Nicholas Stevenson, Ingrid Nelson, Jessica Cavanaugh, Kevin Stoller, David Bode, Be Birchall, Jason Tedor, Shaun Alspaugh, Nathan Knight, Kathleen Meloney, Christopher Frayer, Shalini Reddy, Stephanie Kleven, Kathleen King, Pamela Russell, and Elizabeth Weaver. Many of their papers appear on the website: http://math.oregonstate.edu/~math_reu/REU_Proceedings/.

References

[1] Arett, D. and Doree, S.: Coloring and counting on the Tower of Hanoi graphs, *Mathematics Magazine*, Vol.83, No.3, pp.200–209 (June 2010).
[2] Buneman, P. and Levy, L.S.: The Towers of Hanoi Problem, *Inf. Process. Lett.*, Vol.10, No.4/5, pp.243–244 (1980).
[3] Chen, X., Tian, B. and Wang, L.: Santa Claus' Towers of Hanoi, *Graphs and Combinatorics*, Vol.23, No.suppl. 1, pp.153–167 (2007).
[4] Cull, P. and Ecklund Jr., E.F.: Towers of Hanoi and Analysis of Algorithms, *American Mathematical Monthly*, Vol.92, No.6, pp.407–420 (June-July 1985).
[5] Cull, P., Flahive, M. and Robson, R.: *Difference Equations*, Springer, New York (2005).
[6] Cull, P. and Nelson, I.: Error-correcting codes on the Towers of Hanoi graphs, *Discrete Math.*, 208/209, pp.157–175 (1999).
[7] Cull, P. and Nelson, I.: Perfect Codes, NP-Completeness, and Towers of Hanoi Graphs, *Bull. Inst. Combin. Appl.*, Vol.26, pp.13–38 (1999).
[8] Doran, R.W.: The Gray code, *Journal of Universal Computer Science*, Vol.13, No.11, pp.1573–1597 (2007).
[9] Gardner, M.: Curious properties of the Gray code and how it can be used to solve puzzles, *Scientific American*, Vol.227, No.2, pp.106–109 (1972).
[10] Gross, J. and Yellin, J.: *Graph Theory and its Applications* (*2nd Edition*), Chapman & Hall, Boca Raton, FL (2006).
[11] Imrich, W., Klavzar, S. and Rall, D.F.: *Topics in Graph Theory: Graphs and Their Cartesian Product*, A.K. Peters, Ltd., Wellesley,

Massachusetts (2008).
[12] Jaap: Jaap's puzzle page, available from ⟨http://www.jaapsch.net/puzzles/spinout.htm⟩.
[13] Klavžar, S., Milutinović, U. and Petr, C.: On the Frame-Stewart algorithm for the multi-peg Tower of Hanoi problem, *Discrete Applied Mathematics.*, Vol.120, No.1-3, pp.141–157 (2002).
[14] Li, C.-K. and Nelson, I.: Perfect codes on the Towers of Hanoi graph, *Bull. Austral. Math. Soc.*, Vol.57, pp.367–376 (1998).
[15] Park, S.E.: The group of symmetries of the Towers of Hanoi graph, *American Mathematical Monthly*, Vol.117, No.4, pp.353–360 (April 2010).
[16] Pruhs, K.: The SPIN-OUT puzzle, *ACM SIGCSE Bulletin*, Vol.25, pp.36–38 (1993).
[17] Petr, C., Klažar, S. and Milutinović, U.: 1-perfect codes in Sierpinski graphs, *Bull. Austral. Math. Soc.*, Vol.66, pp.369–384 (2002).
[18] Savage, C.: A survey of combinatorial Gray codes, *SIAM Review*, Vol.39, pp.605–629 (1996).
[19] Spin-Out, Amazon, available from ⟨http://www.amazon.com/Think-Fun-5401-Thinkfun-Spinout/dp/B000EGI4IA⟩.
[20] Walsh, T.R.: The Towers of Hanoi revisited: Moving the rings by counting the moves, *Inf. Process. Lett.*, Vol.15, pp.64–67 (1982).

Paul Cull is Professor (emeritus) of computer science at Oregon State University where he has worked for over 40 years. He has taught a variety of undergraduate courses and graduate courses in theory of computation, analysis of algorithms, and cybernetics. During the summers he works with students in a research experience for undergraduates program which focuses on problems at the interface between computer science and mathematics. In 2001, Cull was given the Alumni Professor Award. His research interests are mathematical biology, theory of computation, and analysis of algorithms. His early work was on inferring gene linkage from pedigree data. He earned his Ph.D. at the University of Chicago where he studied with the committees on mathematical biology and information science. He wrote his thesis on the analysis of neural networks. Over the years, he has worked on a large variety of topics and published papers in many venues. His most recent papers are in machine learning, neural nets, interconnection networks, biological string alignment, discrete iterations, and error-correcting codes. He is the co-author of the book "Difference Equations: From Rabbits to Chaos" published by Springer in 2005.

Leanne Merrill is currently a Ph.D. student at the University of Oregon. Her interests include topological graph theory, probability theory, axiomatic set theory, algebraic topology, and mathematics education. She holds a Master's degrees in Mathematics and Bachelor's degree in Mathematics and Music, all from the State University of New York at Potsdam. In addition to participation in the Oregon State University Research Experience for Undergraduates in 2010, she participated in the SUNY Potsdam/Clarkson REU in 2009.

Tony Van is a graduate from the University of Pennsylvania. He was awarded the Bachelor of Arts with distinction in mathematics and Latin honors in May of 2012. He plans on pursuing doctoral studies in statistics and applied mathematics. Afterwards, he hopes to do research in the private sector and to contribute to academia in the long run. His academic research interests include wavelet analysis and probability theory.