Just a Game? Mathematics and Puzzle-Solving

A thesis submitted to
Loyola Marymount University
The Mathematics Department
and The University Honors Program
in partial fulfillment of the requirements
for Graduation with the Bachelor of Science
and Bachelor of Arts Degrees

by

Jessica Vega

May 2010

Just a Game? Mathematics and Puzzle-Solving

Honors Senior Thesis by

Jessica Vega

Alissa S. Crans, Thesis Director

Abstract

How many times do we find ourselves stumped by a common game, puzzle, or maze and unable to solve it? We all like to play games, but how often do we think of using mathematics to help us strategize our playing? What kind of advanced mathematics can we use on puzzles? Games and puzzles have been the subject of much mathematical inquiry over the years. Using the tools of graph theory, combinatorics, and probability and statistics, we will investigate one popular puzzle, the *KO Labyrinth*. This puzzle, which mechanically moves like a Rubix cube, involves moving a ball through a maze consisting of various colored chambers that can be rotated and aligned in different ways. Some questions we will explore include: What is the fastest way (shortest path) out of the labyrinth? How long will it take to get out of the maze if we randomly move from one chamber to another? Can we visit every chamber only once? What can we say about the colors of the chambers? We will quickly see that mathematics is a powerful tool for answering such questions.

Thesis written by

Jessica Vega

Approved by

Alissa S. Crans, Thesis Director Date

Curtis D. Bennett, Mathematics Department Chair Date

Brad Elliot Stone, Honors Program Director Date

Acknowledgments

First, I must take a moment to thank Dr. Alissa Crans, my thesis advisor, for all her guidance, patience, and enthusiasm throughout the process of completing this honors thesis, as well as through my four years at Loyola Marymount University. I have learned so much from her in mathematics, and in life.

Second, I extend a special thanks to Dr. Robert Rovetti for his help towards the completion of this thesis, particularly with the computer programs. I appreciate the extra time he took out of his schedule to teach me and work through programs.

Last, I thank my mother, who taught me the value of education and without whom I do not know where I would be.

Contents

CHAPTER 1

Introduction

1.1. Just a game?

Games, puzzles, and mazes all have a natural appeal to us. Kicking in our competitive and survival instincts, each of us is moved to use whatever tools at our disposal, be it strength, speed, or wits, to work our way to a solution. We want to see what is possible, to figure out how the pieces fit together, how one action might affect another, or perhaps what is the fastest way to the finish line. We sometimes find ourselves stumped, stuck, and losing, but even in the face of defeat we often still love playing the game. The question, though, that we must first ask ourselves is this: What does this all have to do with mathematics?

Perhaps many people do not initially think of mathematics when solving puzzles, but in reality games and puzzles have been the subject of much mathematical inquiry over the years. For those of us who become excited rather than anxious when we hear the word "math," games and puzzles are friendly when they might otherwise seem difficult and confusing. One puzzle in which this is certainly the case is the *KO Labyrinth*, a spherical maze that is the subject of this thesis project. Using the tools of graph theory, combinatorics, programming, and probability and statistics, we will explore the *KO Labyrinth* in detail to see that mathematics is a powerful tool in puzzle-solving.

Let's play.

1.2. The KO Labyrinth

1.2.1. Description: We start with a description of the basics of the puzzle we will explore, the *KO Labyrinth*, as shown in Figure 1 on the next page.

The *KO Labyrinth* is a spherical puzzle with 26 chambers that can be twisted and rotated around like one can manipulate a Rubix cube. However, the goal of the *KO Labyrinth* is very different than the goal of the Rubix cube. Rather than matching colored sides, solving the Labyrinth involves successfully maneuvering through a maze. On the top of the *KO Labyrinth*, we find an entrance hole; on the bottom, there is an exit hole. On the walls of the chambers there are many other holes that might align based on the current configuration of the puzzle. The goal of this puzzle is to pass two small balls from the first chamber with the entrance hole, through the maze, and to the last chamber with an exit hole.

1.2.2. The chambers: There are three different types of chambers in our puzzle, each labeled accordingly.

FIGURE 1. The KO Labyrinth

- **A chambers.** These are triangular rooms that are the "corners" of our sphere, if we were to think of the sphere like the Rubix cube. There are 8 such chambers, labeled $A1 - A8$. Observing how the walls of these chambers align with other chambers around it, we notice that A chambers connect only to B chambers.[1]
- **B chambers.** These are rectangular rooms that are middle pieces. There are 12 such chambers, labeled $B1 - B12$. We notice that B chambers can connect to both A and C chambers.
- **C chambers.** These are square rooms in the center of each "face" of our sphere. There are 6 such chambers, labeled $C1 - C6$. We notice that C chambers connect only to B chambers.

Within the chambers, each wall that has a hole is also colored. The colors are important–when we rotate the puzzle and try to match up the walls of different chambers so that the holes align, we notice that all matching walls share the same color. We note, however, that not all walls of the same color match. The colors used are orange, blue, red, purple, pink, yellow, gray, and green. The specific color of a wall only matters when matching with another wall of the same color. See Appendix 7.1 for full list of the specific colors in each chamber.

Chambers $B9$ and $C4$ both have knobs around their respective colored hole that do not allow the ball the roll freely in the chamber. While it may be creatively confusing for the player, for our purposes we will say that if the ball is inside the knob, it is effectively inside the chamber.

Now that we know a little about our puzzle, we are ready to introduce some of the basic mathematics we will use to explore the *KO Labyrinth*.

[1]There are a few unintentional exceptions, but we will disregard these.

1.3. Preliminary Information

1.3.1. Graph Theory: The field of graph theory first began with a paper in 1736 by Leonhard Euler on the *Seven Bridges of Königsberg*. The city of Königsberg was set on both sides of the Pregel River. It included two large islands connected to each other and to the mainland by seven bridges. Euler asked if it was possible to walk through the city and cross each bridge once and only once. His proof that the problem has no solution laid the groundwork for graph theory. More than a century later, Cayley studied a particular class of graphs, called *trees*. Many problems of practical interest can be represented by graphs, and so the uses and applications of graph theory are wide. For example, the development of algorithms on graphs is of particular interest in computer science, and graphs can be used to study molecules in chemistry and physics.

Since we will soon consider the *KO Labyrinth* mathematically using graph theory, we begin by reminding the reader of some necessary definitions.

Definition 1.1. A **graph** $G = (V, E)$ consists of a finite set V of points, or **vertices** together with a set E of **edges** joining different pairs of distinct vertices. We will use the notation (a, b) to denote that there is an edge between vertex a and vertex b. These two vertices are said to be **incident** to that edge, or, equivalently, that edge incident to those two vertices.

Figure 2 illustrates examples of graphs:

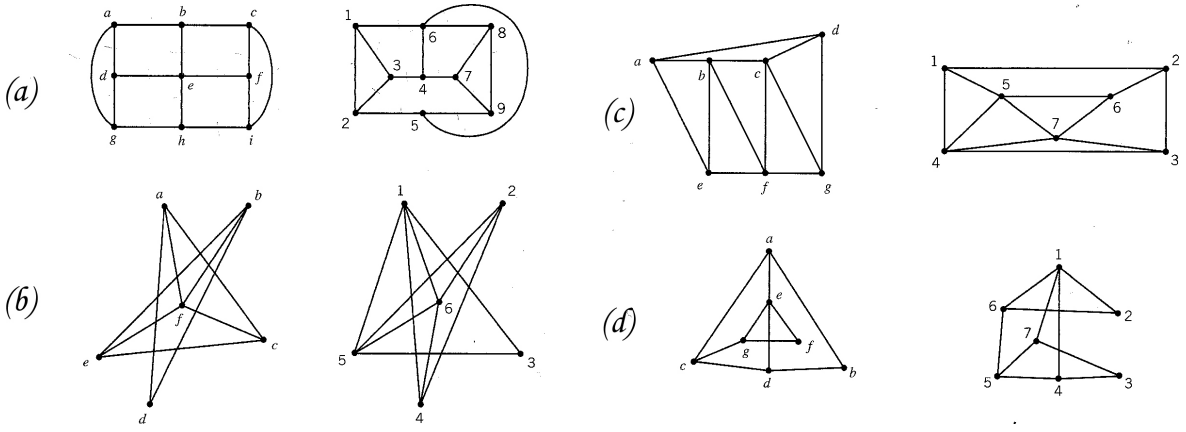


Figure 2. Examples of Graphs

In our graphs, we say that vertices a and b are **adjacent** when there is an edge (a, b). For example, in the first graph of (a) in Figure 2 above, vertices a and b are adjacent, but in the first graph of (b), they are not.

The number of edges incident to a vertex is called the **degree** of a vertex. For example, vertex f of example (d) has degree 2.

DEFINITION 1.2. A **path** P is a sequence of distinct vertices, written $P = x_1 - x_2 - ... - x_n$, where each pair of consecutive vertices in P is joined by an edge. If in addition, there is an edge (x_n, x_1), the sequence is called a **circuit**.

For example, in the second graph of example (c) in Figure 2, we have a circuit $1 - 2 - 3 - 4 - 7 - 6 - 5 - 1$.

We now introduce some special kinds of graphs.

DEFINITION 1.3. A graph with n vertices in which each vertex is adjacent to all the other vertices is called a **complete graph on n vertices**, denoted K_n.

DEFINITION 1.4. A **bipartite** graph is a graph whose vertices can be partitioned into two sets V_1 and V_2 and every edge joins a vertex in V_1 with a vertex in V_2.

DEFINITION 1.5. A **complete bipartite graph**, denoted $K_{n,m}$, is a graph that contains a set of n vertices and a set of m vertices where each vertex in one set is adjacent to all vertices in the other set.

Figure 3 illustrates an example of a complete graph and a complete bipartite graph.

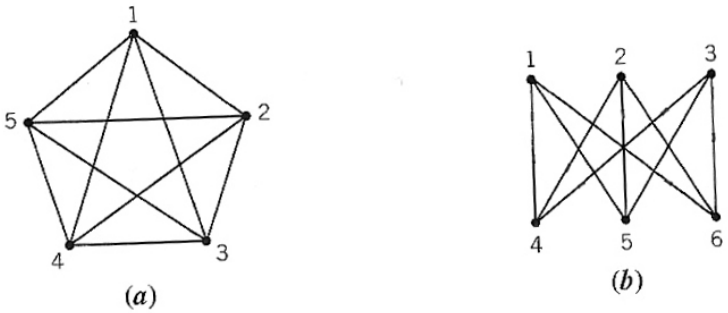


FIGURE 3. Complete Graphs

Graph (a) is a K_5 configuration and graph (b) is a $K_{3,3}$ configuration.

DEFINITION 1.6. A **tree** is a connected graph with no circuits. The **root** of a tree is a designated vertex such that there is a unique path from the root to any other vertex in the tree.

Figure 4 is an example of a tree, with the top vertex as its root. Graphs that represent single-elimination tournament brackets are also examples of trees.

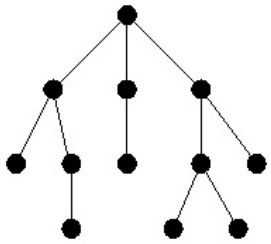


FIGURE 4. A tree

DEFINITION 1.7. Two graphs G and G' are called **isomorphic** if there exists a one-to-one correspondence between the vertices in G and the vertices in G' such that a pair of vertices are adjacent in G if and only if the corresponding pair of vertices are adjacent in G'.

DEFINITION 1.8. A **subgraph** H of a graph G is a graph formed by a subset of vertices and edges of G.

If two graphs are isomorphic, then subgraphs formed by corresponding vertices and edges must be isomorphic.

As an example, we will consider the graphs in Figure 2 on page 3 and determine which are isomorphic. We see that the two graphs of (a) are isomorphic to each other and the two graphs of (d) are isomorphic also. The two graphs of (b) and (c) are not isomorphic to each other. The isomorphism in (a) is given by $a-3, b-4, c-7, d-1, e-6, f-8, g-2, h-5, i-9$. The isomorphism in (d) is given by $a-7, b-3, c-5, d-4, e-1, f-2, g-6$. The graphs of (b) cannot be isomorphic because the right graph has one more edge. The graphs of (c) cannot be isomorphic either because the subgraphs of vertices of degree 3 do not match.

In the next chapter, we proceed by creating a graph of the *KO Labyrinth*.

CHAPTER 2

Graph Theory and the KO Labyrinth

We are now ready to use some of the graph theory we have learned to explore our puzzle, the *KO Labyrinth*. We begin by creating a graph associated to the puzzle. Each of the 26 chambers of the maze represents one vertex in our set V of vertices. There is an edge between two vertices if there exists an alignment of their colored holes so that the ball can pass between the chambers. The graph appears below in Figure 1.

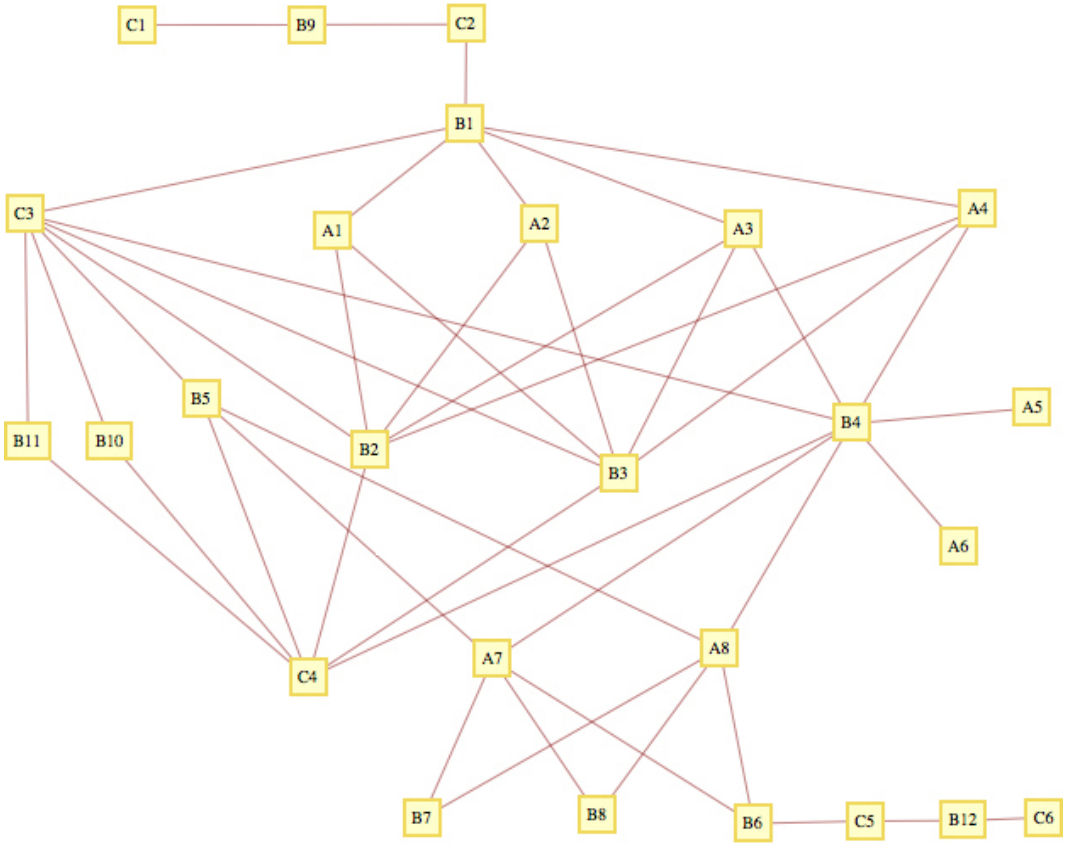


FIGURE 1. KO Graph

It might seem that we have not done much math yet, but already the puzzle seems much less complicated and confusing. Certainly, a beginner can use the information in the graph to find their way through the labyrinth. For example, we could use the

path $C1 - B9 - C2 - B1 - A2 - B3 - A4 - B4 - A7 - B6 - C5 - B12 - C6$ to go through the maze. Or, perhaps if we found ourselves stuck in chamber $C4$ as we are working through the maze, we could look at the above graph and see that our next step could be to go to $B11$, $B10$, $B5$, $B2$, $B3$, or $B4$. Each of these chambers will align correctly with chamber $C4$ so that the ball can pass to that chamber, but, we could not go to chamber $A7$, for example. Then, we can find a path out of the maze, for example, using $C4 - B5 - A8 - B6 - C5 - B12 - C6$.

2.1. Properties of the Graph

Here we consider various properties of the graph.

2.1.1. Connectivity: It is easy to see that the graph is **connected**, that is, that there is a path between every pair of vertices.[1] One way to test for connectivity is by using an algorithm to create a spanning tree of our graph.

DEFINITION 2.1. A **spanning tree** of a graph G is a subgraph of G that is a tree containing all vertices of G. [2]

In order to create a spanning tree, we use either a *depth-first* or *breadth-first* search. We will use a breadth-first search. [3] To build a breadth-first spanning tree, we pick some vertex as the root and put all edges leaving that vertex, along with the vertices at the end of those edges, in the tree. In our case, we choose the starting vertex, $C1$, as our root. Then, we successively add to the tree those edges incident to the vertices just added from the root vertex. We continue in this fashion level by level. If all vertices are reached in the search and a spanning tree is created, then the original graph is connected.

Figure 2 shows the spanning tree for the graph of the *KO Labyrinth*.

[1] see Definition 1.2 of a path

[2] See Definition 1.6 of a tree and 1.8 of a subgraph

[3] See Appendix 7.2 for explanation of a depth-first search

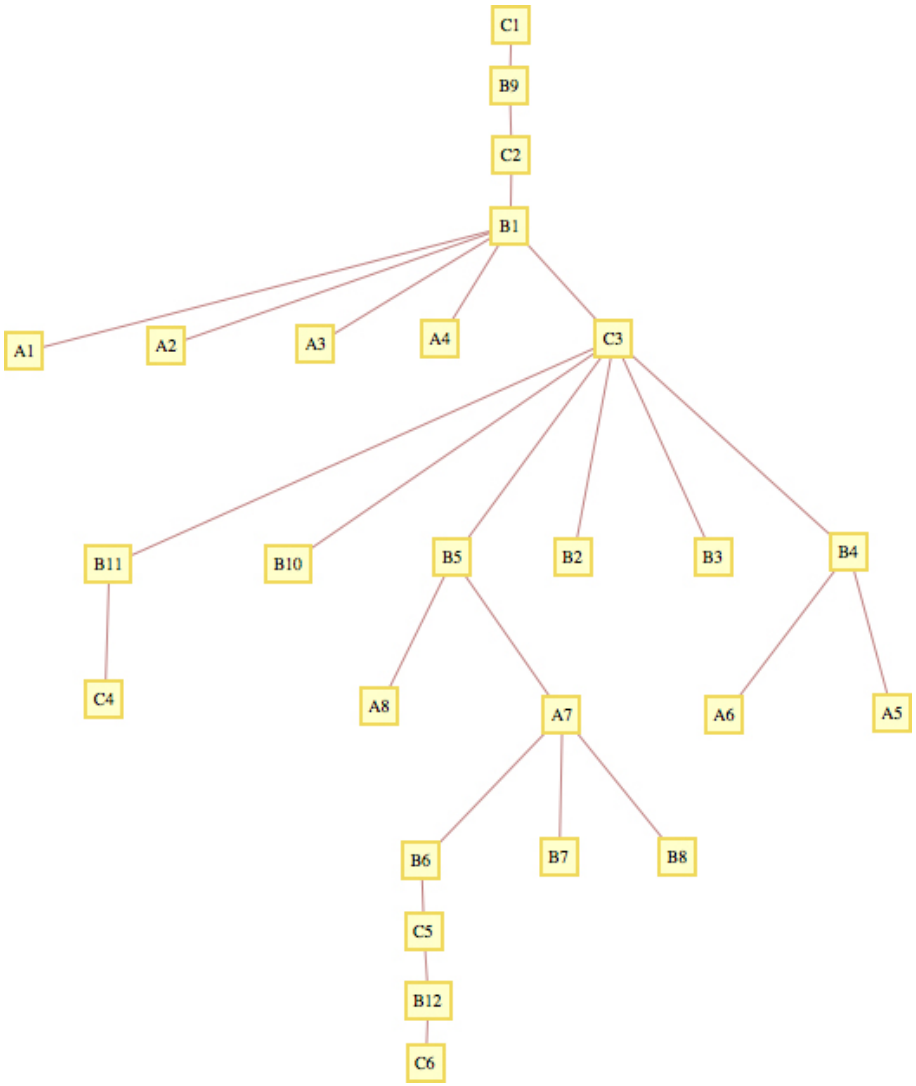


FIGURE 2. Spanning tree using a breadth-first search

We can clearly see that the definition of connected is satisfied with this subgraph. Surely we can see paths between any two vertices. For example, to get from $A7$ to $A5$, we use the path $A7 - B5 - C3 - B4 - A5$. Certainly adding the missing edges to the graph does not break the paths we found in the spanning tree, and therefore does not disconnect the graph.

Recall also that connectivity of a graph is preserved by isomorphisms.[4] Certainly, if there is a path between each pair of vertices in a graph G, and G is isomorphic to G', then our one-to-one correspondence between vertices in G and G' preserves the paths between pairs of vertices in G'. This is important for our purposes because we

[4]See Definition 1.7

will consider other versions of the graph of the *KO Labyrinth* that are isomorphic to the graph we have already seen in Figure 1.

We will return to the subject of connectivity in Section 3.5 after we have explored and understood the graph more deeply.

2.1.2. Graph v. Multigraph: It turns out that our graph is actually a **multigraph**, that is, a graph that allows (1) two or more edges to join the same two vertices and (2) loops, or edges of the form (a, a). We have a multigraph for the *KO Labyrinth* because there are two options for how to move from chamber $C5$ to chamber $B12$. As mentioned in Section 1.2.2,[5] each chamber has one or two colored walls. In order to move the balls from one chamber to another, we must match up and align walls of the same color. In the case of $C5$ and $B12$, the blue walls match *and* the purple walls match; therefore, there are two ways to move between those chambers and therefore two edges between the corresponding vertices. These options are identical in terms of solving the puzzle– it does not matter whether we choose the blue or purple colored walls to move between the chambers. Thus, we choose to simply use one of these options, thereby eliminating the multiple edge. We do this because many of the algorithms and theorems we will use in this paper only hold for graphs, *not* multigraphs. Since it takes away nothing from the graph to eliminate the multiple edge, we do it in order to use these theorems.

2.1.3. Planarity: Our graph is **nonplanar**, that is, it cannot be drawn on a plane without edges crossing. We will use Kuratowski's theorem to prove this.

THEOREM 2.1. *(Kuratowski, 1930) A graph is planar if and only if it does not contain a subgraph that is a K_5 or $K_{3,3}$ configuration.*

To get a sketch of the proof of Kuratowski's theorem, we would have to prove that the $K_{3,3}$ and K_5 graphs are nonplanar, since the theorem depends on that fact.[6] While we will not prove the K_5 case, we will prove that the $K_{3,3}$ graph is nonplanar since our graph contains a $K_{3,3}$ subgraph, seen in Figure 2 below.

[5]Also, see Appendix 7.1 for a full list of each chamber, the colors of each wall of that chamber, and the degree of its corresponding vertex

[6]See Figure 3 in Chapter 1 for a picture of the $K_{3,3}$ and K_5 graphs

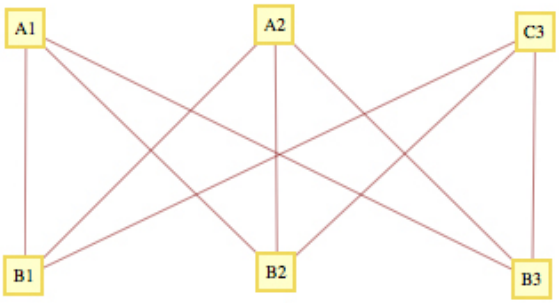


FIGURE 3. $K_{3,3}$ configuration

To prove the nonplanarity of the $K_{3,3}$ graph, we use the **circle-chord method**. We find a circuit that contains all the vertices of our graph and draw this circuit as a large circle. The remaining noncircuit edges, which we call *chords*, must be drawn either inside the circle or outside the circle in a planar drawing. As we draw the chords, they will force other chords to be drawn either inside or outside the circle. If we come to a point where a chord cannot be drawn inside or outside the circle, we know the graph is nonplanar.

THEOREM 2.2. *The graph $K_{3,3}$ is nonplanar.*

PROOF. Recall that the $K_{3,3}$ graph is a complete bipartite graph consisting of two sets of 3 vertices where each vertex in one set is adjacent to all vertices in the other set:

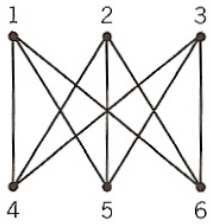
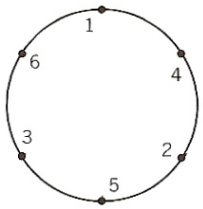
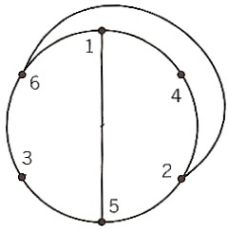


FIGURE 4. $K_{3,3}$ is nonplanar

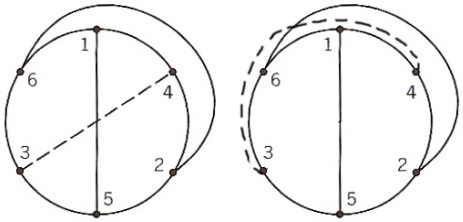
We apply the circle-chord method to form a circuit containing all six vertices. We choose the circuit $1-4-2-5-3-6-1$. Three edges must be added, $(1,5), (2,6)$, and $(3,4)$, since these are the only remaining non-circuit edges of the graph.



Since it does not matter whether we draw the first chord inside or outside the circle, we draw $(1, 5)$ on the inside. This forces the chord $(2, 6)$ to be drawn outside the circuit.



Last, we must draw $(3, 4)$, but if it is drawn outside the circuit, it will cross chord $(2, 6)$. If we drawn it inside, it will cross chord $(1, 5)$.



Thus $K_{3,3}$ cannot be drawn with a planar depiction and hence is nonplanar. □

Now that we have proven that $K_{3,3}$ is nonplanar, we breifly sketch the rest of the proof of Kuratowski's theorem. The proof would show that given a $K_{3,3}$ or K_5 subgraph in a graph G, adding back any other missing edges or vertices of G to the subgraph would continue to be problematic. Simply adding edges or vertices to an already nonplanar graph will result in another nonplanar graph.

Thus, because we have a $K_{3,3}$ subgraph in the graph of the *KO Labyrinth*, our graph is nonplanar.

2.1.4. Isomorphisms. Now, we consider isomorphisms to the graph of the *KO Labyrinth* in Figure 1.[7] Redrawing the graph with another depiction helps reveal some other properties that might not have been apparent before. A natural way to redraw this graph is based on the labeling of the chambers mentioned in Section 1.2.2. As we saw previously, there are A, B, and C chambers, and each type of chamber only connects to certain other types. So, we notice that we can redraw the graph as a **tripartite** graph. We have already discussed bipartite graphs, so analogously, a tripartite graph is partitioned into three subsets of vertices. The tripartite graph of the *KO Labyrinth* appears below:

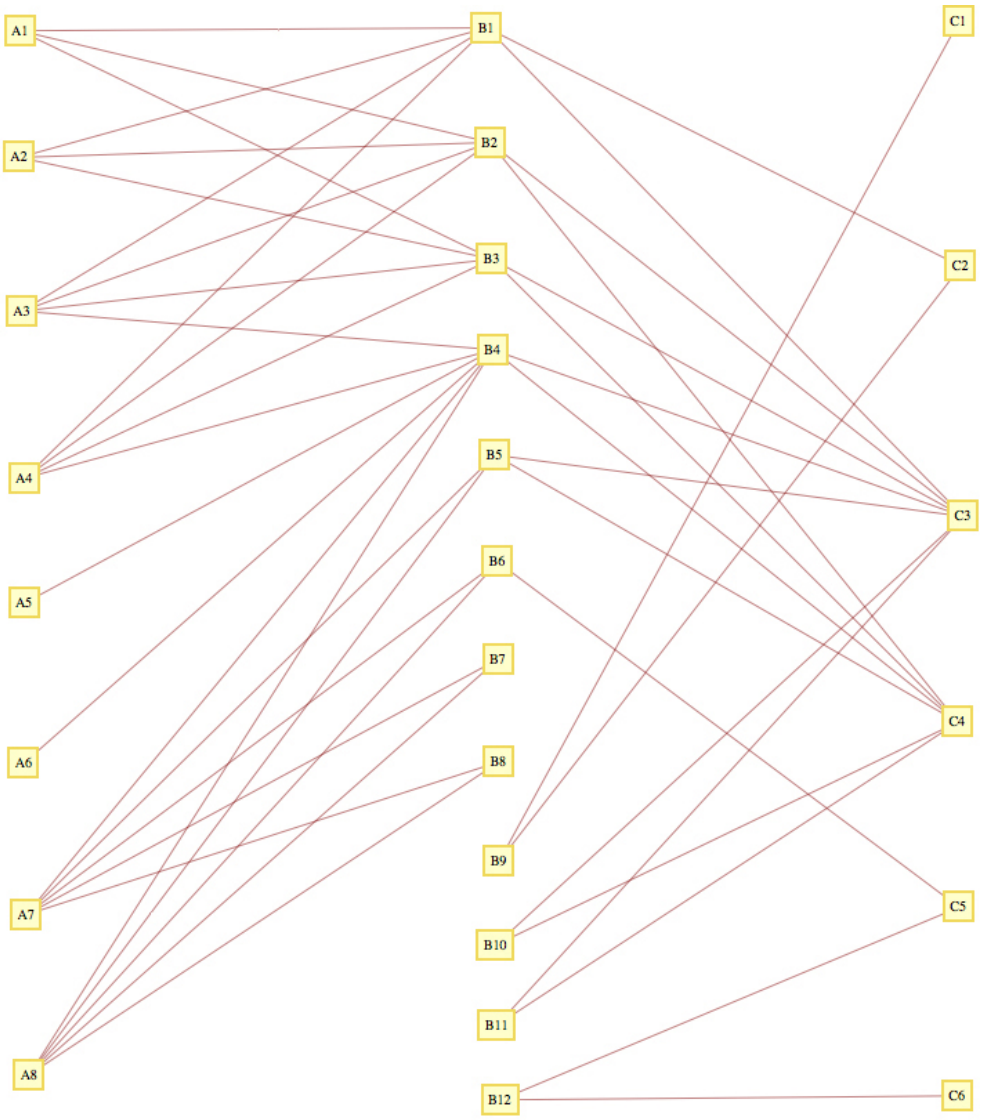


FIGURE 5. Tripartite Graph

[7]See Definition 1.7 of an isomorphism

While this representation of the graph of the puzzle will prove helpful for mathematical analysis, we note that it might not serve as a helpful means to navigate through the puzzle. However, the tripartite graph of the *KO Labyrinth* reveals repetition in the puzzle that we might find interesting. So, we offer a new definition.

DEFINITION 2.2. A chamber a is a **twin chamber** of another chamber b if:

(1) both a and b are adjacent to the same chambers, and
(2) A chamber a is adjacent to another chamber c using a color m if and only if chamber b is also adjacent to chamber c using color m.

In essence, twin chambers have access to the same other chambers by the same colored routes through the maze.

Note that there are several sets of twin chambers: {A1, A2}, {A3, A4}, {A5, A6}, {A7, A8}, {B2, B3}, {B7, B8}, and {B10, B11}. See Appendix 7.1 for details on each chamber.

We will see later that recognizing the twin chambers will be important when answering a traveling-salesman type problem in Chapter 5. For now, we continue with the graph theoretic material in the next chapter by searching for a shortest path through the maze.

CHAPTER 3

Shortest Path: Dijkstra's Algorithm

3.1. The Question

Now, we will ask a question that would interest the most competitive game-players out there: *What is the fastest (shortest) path through the labyrinth?* In order to answer this question, we will first introduce a new definition:

DEFINITION 3.1. A **weighted graph or network** is a graph with a positive integer $k(e)$ assigned to each edge e. This integer represents the "weight" or "cost" or "length" of that edge.

Weighted graphs are useful in representing many realistic situations. For example, consider a graph of cities as vertices and routes between cities as edges. Each edge is assigned a number representing the time it takes to travel between the two cities. Or, we could consider another example, shown below.

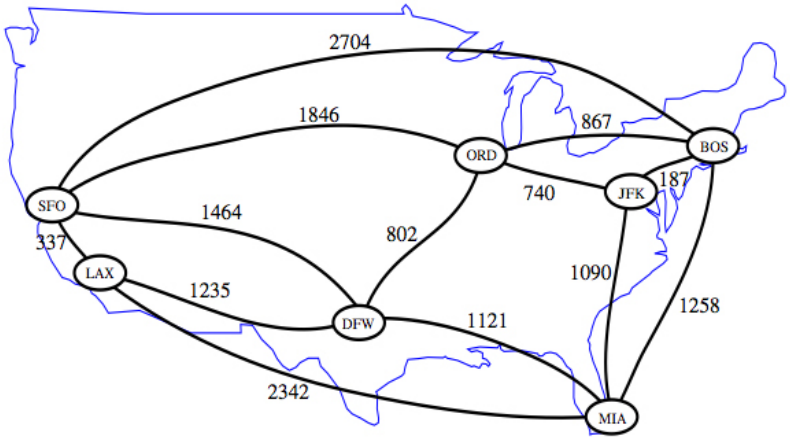


FIGURE 1. Example of a weighted graph

Vertices in our graph represent major U.S. airports and the weights of the edges represent distance in miles. We see, for example, that there is 1258 miles between BOS and MIA. We see that there are many possible paths between JFK and LAX, which leads us again to our beginning question. How do we find the shortest path?

We use Dijkstra's Algorithm to answer this question. Edsger W. Dijkstra (May 11, 1930 - August 6, 2002) was a Dutch computer scientist whose contributions in

the early years of programming helped develop structured control constructs, such as the *while* loop. While he was mostly interested in computer science, his work in developing algorithms and proofs of programs contributed much to mathematics as well. Dijkstra worked against the prevailing opinion of his time that one should first write a correct program and then mathematically prove its correctness. Rather, he believed in working for mathematically correct programs by construction, creating the program and proof at the same time.

Dijkstra developed the algorithm that bears his name in 1959 as a graph search algorithm that finds the shortest path for a weighted graph by producing a shortest path tree. We are now ready to explore the algorithm.

3.2. The Algorithm

We are now ready to explain Dijkstra's Algorithm. Let $k(e)$ denote the length of edge e. Let the variable m be a "distance counter." This variable will help us keep track of the minimum distance between vertices in each step of the algorithm. We choose a starting vertex a. As we increment m by one, our algorithm will produce labels for each vertex of the graph. The label for a vertex y will be an ordered pair $[r, d(y)]$ where r is a vertex adjacent to y and $d(y)$ is the minimum length from starting vertex a to y.

Dijkstra's Algorithm:

1) Choose a starting vertex a. Set $m = 0$
2) Label vertex a with $[-, 0]$ (the "$-$ represents a blank) and set $m = 1$.
3) Let p be a labeled vertex and q be an unlabeled vertex. Suppose $p's$ labels are $[r, d(p)]$. Consider the weight $k(e)$ from each edge $e = (p, q)$. If $d(p) + k(e) = m$, then $d(q) = m$ and label q with $[p, m]$.
4) If all vertices are not yet labeled, increment m by 1 and go to Step 3. Otherwise, go to Step 5.
5) For any vertex y, a shortest path from a to y has length $d(y)$, the second label of y. Such a path may be found by backtracking from y using the first labels.

PROOF. We use an inductive argument to show that Dijkstra's algorithm produces the length of the shortest path between two vertices a and x. The inductive hypothesis will be the following assertion: When $m = c$, a vertex p that is labeled is in fact labeled with the length of the shortest path from the starting vertex a to that vertex p.

When $m = 0$ and the only vertex labeled is a, certainly the length of the shortest path from a to itself is 0. Therefore, the base case is true.

Assume that the inductive hypothesis holds when $m = c$, some positive integer. This means that all labeled vertices are labeled with the shortest paths from a. We increment m by one, so $m = c + 1$. Let q be a vertex labeled when $m = c + 1$ with label $[r, c + 1]$. This means that $m = c + 1 = k(e) + d(p)$, where p is a labeled vertex and $k(e)$ is the weight of the edge $e = (p, q)$. If q could be labeled with a value less than $c + 1$, then we contradict our inductive hypothesis since certainly q would have

been labeled already when $m = c$. If q can be labeled with a value larger than $c + 1$, then either (1) $c + 1 \neq k(e) + d(p)$, a contradiction, or (2) q is already labeled at the step when $m = c + 1$ and we would not replace this smaller label for the larger one. Therefore, q must be labeled with the shortest path from it to a.

Thus, once all vertices are labeled, we have found the shortest paths from all vertices to a □

EXAMPLE 3.1. We consider a simple example on which to demonstrate Dijkstra's Algorithm. Consider the graph below. We choose our starting vertex a, and we wish to label all other vertices.

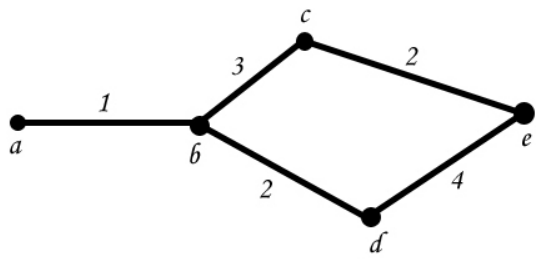


FIGURE 2. Weighted Graph

We start with $m = 1$ and label a with $[-, 0]$. Now, we consider edges incident to a to unlabeled vertices. For this example, we only have one edge, $e = (a, b)$ with $k(e) = 1$. Since $0 + 1 = m = 1$, we label b with $[a, 1]$. Since all vertices are not yet labeled, we return to step 3. We consider edges $e_c = (b, c)$ and $e_d = (b, d)$ with $k(e_c) = 3$ and $k(e_d) = 2$. However, we notice that we are unable to label either vertex without first incrementing m. This is because with $m = 1$, we have $1 + 3 = 4 \neq 1$ and $1 + 2 = 3 \neq 1$, recalling that the one is the minimum distance to b and the three and two are the weight, respectively, of the edges $e_c = (b, c)$ and $e_d = (b, d)$. So, we increment m by one and proceed, returning again to step 3. Now $m = 2$, but we still cannot label a vertex. Similar to what we had above, now we have $1 + 3 = 4 \neq 2$ and $1 + 2 = 3 \neq 2$. So we increment m by one again and now $m = 3$. Since $d(b) + k(e_d) = 1 + 2 = 3 = m$, we label vertex d with $[b, 3]$.

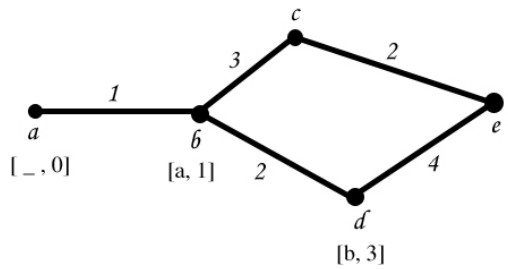


FIGURE 3. Graph with some labels

We still cannot label vertex c, so we increment m again so that $m = 4$. Since $d(b) + k(e_c) = 1 + 3 = 4 = m$, we label vertex c with $[b, 4]$.

Now we consider edges (c, e) and (d, e) of weights two and four, respectively. Since $4 + 2 = 6 \neq 4$ and $3 + 4 = 7 \neq 4$, we cannot label the vertex e. So, we increment so that $m = 5$. Again, we cannot label the last vertex, so we increment again so that $m = 6$. We label vertex e with $[c, 6]$ since $4 + 2 = 6 = m$. Now that all vertices are labeled, we have completed the algorithm and we have obtained the following labeled graph:

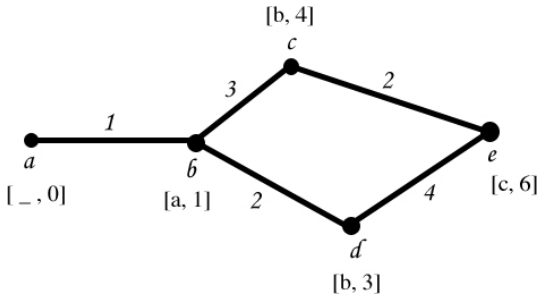


FIGURE 4. Graph with all labels

Recall the example on page 14 of the network representing distances between major airports in the U.S. We saw that there are many paths between airports JFK and LAX. Now that we understand Dijkstra's Algorithm, we can use this algorithm on that graph. We leave it up to the reader to verify that the shortest path between JFK and LAX is 2,777 (miles) using the path JFK - ORD - DFW - LAX.

3.3. Applying Dijkstra's Algortithm to the KO Graph

We now apply Dijkstra's Algorithm to the graph of the *KO Labyrinth*. We set each edge weight $k(e) = 1$ because, for our purposes, the cost of moving from one chamber to any of its adjacent chambers is equal. For example, chamber $A1$ is adjacent to chambers $B1$, $B2$, and $B3$. The "length" or "distance" or "cost" in moving from $A1$ to any of these three options is the same for our puzzle, so the weight of those edges are equal. We choose chamber $C6$ as our starting vertex, which is the end of the puzzle. This is because we want to know how far we are from the end of the maze at any other chamber in the maze. After going through the algorithm, we obtain the following labels for our graph, shown below:

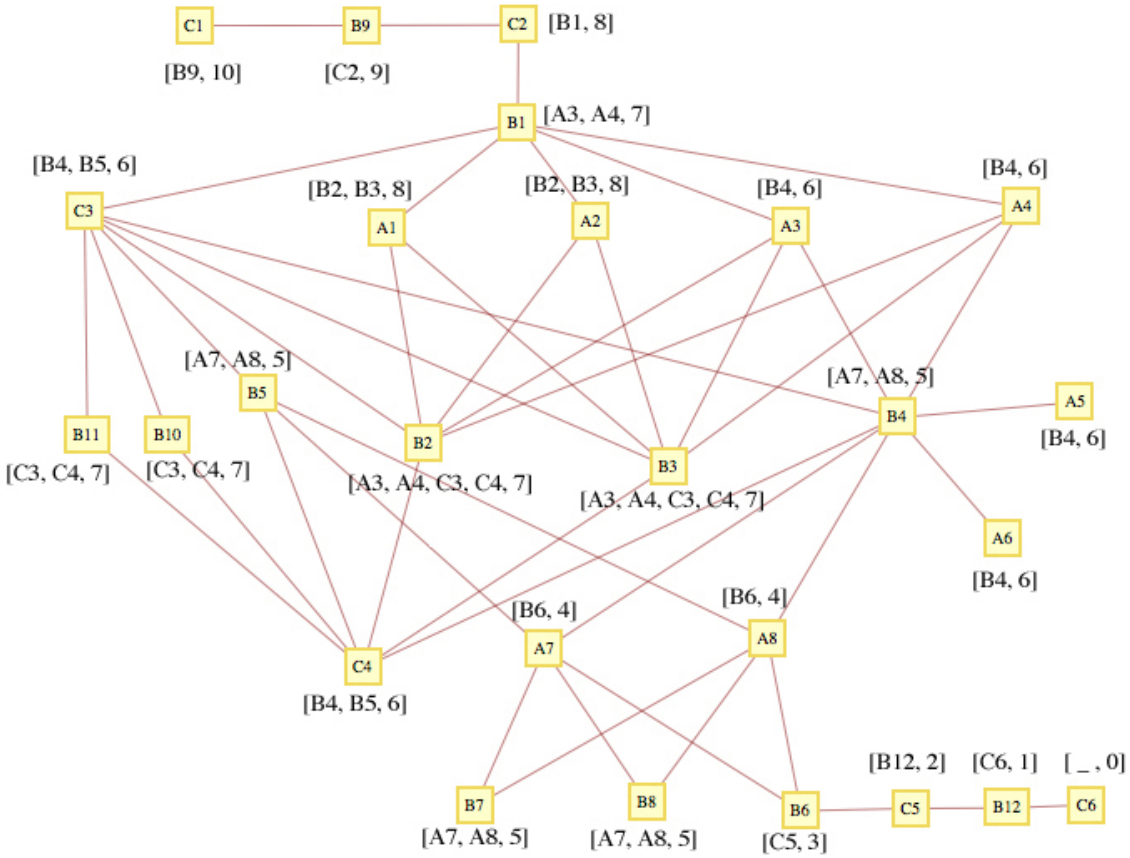


FIGURE 5. Graph of the KO Labyrinth with labels

Since the second entry in the label on vertex $C1$ is 10, this means the shortest path from $C1$ to $C6$ takes 10 moves. Recall that by a "move" we mean to travel from one chamber to the next, not the many rotations and twists the puzzle requires to align the chambers.

Then, we immediately notice that because in our puzzle every edge has the same weight, there are actually several 10-move paths through the maze. Because, for example, we see that there are three different choices in the label on chamber $B1$ for our next move, we know there is not just one shortest path through the maze. Our next question, then, is to count the number of 10-move paths.

3.4. Counting the 10-move paths

To count the number of shortest 10-move paths, we will look at a subgraph of the original *KO Labyrinth* graph.[1] We look at only the vertices and edges relevant to the shortest paths to make it easier to count how many paths there are. The vertices and

[1]See Definition 1.8 of a subgraph

edges we include are based on the labels given by Dijkstra's Algorithm. Here is the subgraph we will consider:

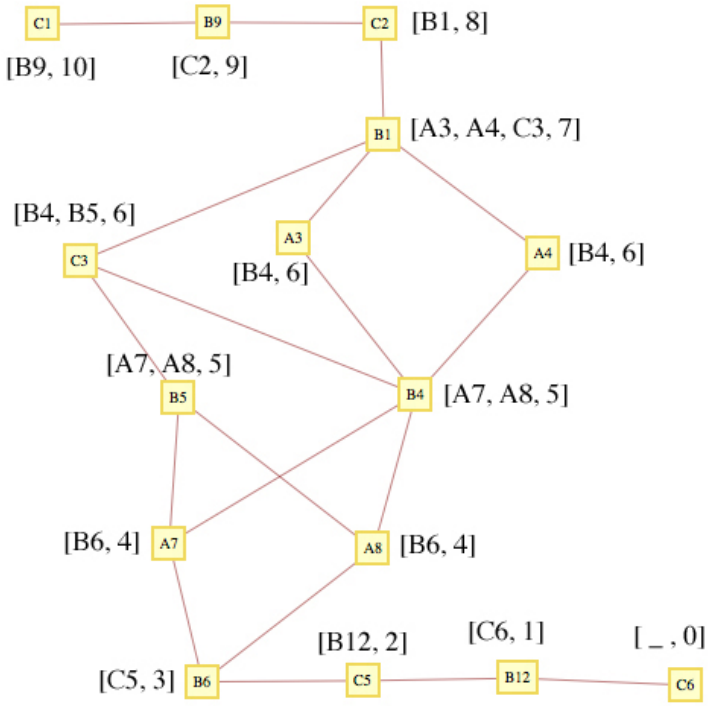


FIGURE 6. Subgraph of the 10-move paths

Certainly, we see that there is only one path from $C1$ to $B1$ and from $B6$ to $C6$. We therefore can start by looking at $B1$ and end at $B6$.

Consider first the paths going from $B1$ to $C3$. From $C3$, we can choose to move to either $B4$ or $B5$. From $B4$, we can choose either $A7$ or $A8$ to get to $B6$; similarly, from $B5$, we can choose either $A7$ or $A8$ to get to $B6$. Therefore, the two choices of either $B4$ or $B5$ times the two choices of either $A7$ or $A8$ gives us $2 \times 2 = 4$ paths starting from $C3$.

Consider next the paths going from $B1$ to $A3$. We must move from Chamber $A3$ to chamber $B4$, and then we have a choice between $A7$ or $A8$ to get to $B6$. That means there are 2 more paths via $A3$.

Consider last the paths going from $B1$ to $A4$. Similar to chamber $A3$, we must next move to chamber $B4$ and then we have a choice between $A7$ and $A8$. That gives us 2 more paths via $A4$.

Thus, we have $4+2+2 = 8$ total shortest 10-move paths. We list the paths below:

$$\{C1 - B9 - C2 - B1 - C3 - B5 - A7 - B6 - C5 - B12 - C6\}$$

$$\{C1-B9-C2-B1-C3-B5-A8-B6-C5-B12-C6\}$$
$$\{C1-B9-C2-B1-C3-B4-A8-B6-C5-B12-C6\}$$
$$\{C1-B9-C2-B1-C3-B4-A7-B6-C5-B12-C6\}$$
$$\{C1-B9-C2-B1-A3-B4-A8-B6-C5-B12-C6\}$$
$$\{C1-B9-C2-B1-A3-B4-A7-B6-C5-B12-C6\}$$
$$\{C1-B9-C2-B1-A4-B4-A8-B6-C5-B12-C6\}$$
$$\{C1-B9-C2-B1-A4-B4-A7-B6-C5-B12-C6\}$$

If we look carefully, we notice that a couple of our twin chambers are involved in our shortest 10-move paths. Chambers $\{A3, A4\}$ and $\{A7, A8\}$ are pairs of twins. Recall that twin chambers are adjacent to the same set of chambers.[2] In fact, this creates repetition in our puzzle, and we see that here as we counted the 10-move paths. Certainly, removing one twin of the $\{A7, A8\}$ pair removes half of our 10-move paths. But, if we were to eliminate one twin of each pairs, we still are not reduced to one 10-move path. Choose one twin of each pair to eliminate, say $A4$ and $A8$. Looking at the list above, we see that paths $\{C1-B9-C2-B1-C3-B5-A7-B6-C5-B12-C6\}$, $\{C1-B9-C2-B1-C3-B4-A7-B6-C5-B12-C6\}$, and $\{C1-B9-C2-B1-A3-B4-A7-B6-C5-B12-C6\}$ do not require chambers $A4$ or $A8$. That still leaves us three 10-move paths. We can even eliminate an entire set of twins, $\{A3, A4\}$, and have four 10-move paths left. Certainly, though, if we eliminate the pair $\{A7, A8\}$, we would no longer have any 10-move paths. Actually, eliminating both $A7$ and $A8$ disconnects the graph, so the puzzle would have no solution at all.

We have successfully answered the question of this chapter, but our last analysis leads us back to a question about connectivity, which we explored in Section 2.1.1. We will now address a question about the connectivity of our graph based on information we have discovered in this chapter.

3.5. Connectivity

We now ask the following question: *Which key chambers or groups of chambers can we eliminate from our puzzle and disconnect the graph?*

We start off by pointing out obvious single-elimination examples. Review the graph of the KO Labyrinth:

[2] See Definition 2.2 of a twin chamber

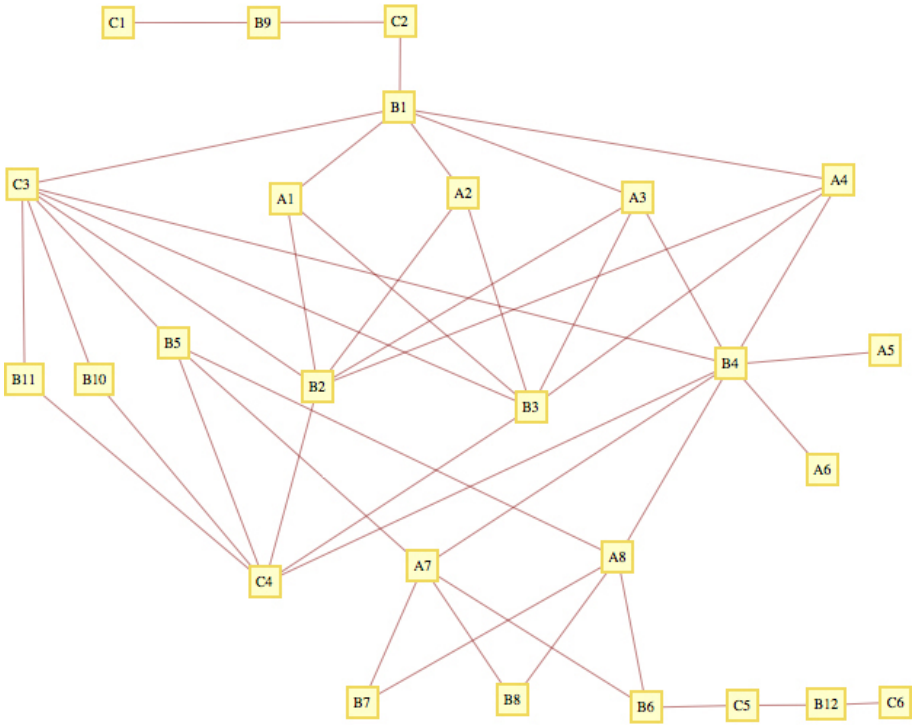


FIGURE 7. KO Graph

We see, again, that there is a single path from the beginning of our puzzle at chamber $C1$ to chamber $B1$. Also, there is a single path from chamber $B6$ to the end, chamber $C6$. Therefore, eliminating any one of these chambers will disconnect the graph, with the exception of the starting chamber $C1$ and ending chamber $C6$. That is, eliminating one single chamber of chambers $B9$, $C2$, $B1$, $B6$, $C5$, and $B12$ will disconnect the graph.

The "middle" section of our graph proves more complicated. Therefore, we look at another version of the graph isomorphic to Figure 7. This graph is organized as follows: There are eleven "levels" of the graph, labeled $0-10$. The levels are based on our results from Dijkstra's algorithm. This means that if the ball is in a chamber in level x, then there are x more moves to make in order to exit the maze. For example, chamber $B4$ in level 5 has a shortest path of 5 more moves to the end of the maze.

The graph is given below:

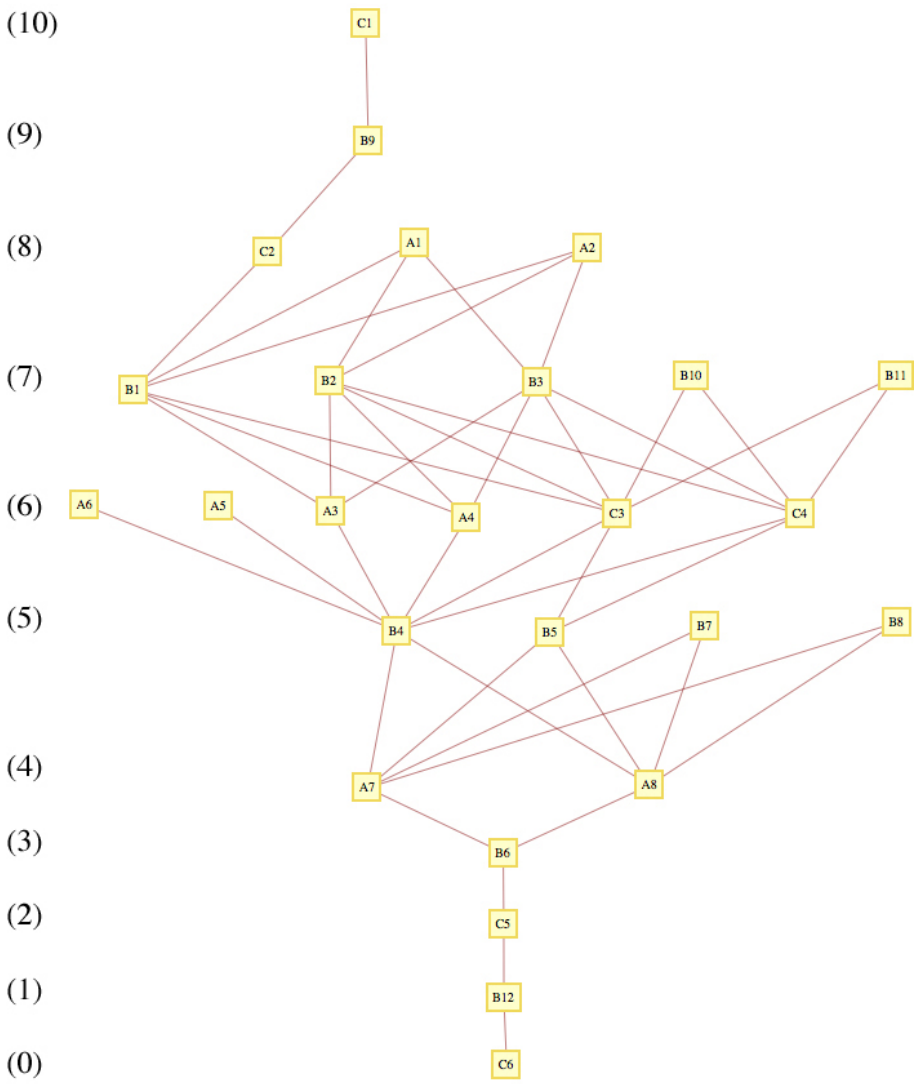


FIGURE 8. The Graph based on Dijkstra's Algorithm

This graph allows us to recognize key groups of chambers whose elimination would disconnect the graph. We follow a simple rule to determine these groups.

Rule: For each level i, we eliminate all chambers in level i that are also adjacent to any chamber in level $i+1$ in order to disconnect the graph.

This rule depends on the fact that whatever way we choose to move through the puzzle, we must visit each level at least once. If we eliminate all means by which to move from a specific level to the next, we disconnect the graph. For example, eliminating chambers $B4$ and $B5$ disconnects the graph (note that this is not easy to see in Figure 7). Level 7 and level 0 are the only exceptions to the rule. Eliminating

chamber $C6$, our end chamber, does not disconnect the graph, as already mentioned. On level 7, only chamber $B1$ needs to be eliminated. This is because chambers $B2$ and $B3$ are only adjacent to chambers $A1$ and $A2$ in the level 8. Chambers $A1$ and $A2$ are only adjacent to chamber $B1$ in addition to $B2$ and $B3$. Thus, we only need to eliminate chamber $B1$.

The following list indicates groups of chambers that will disconnect the graph when eliminated together:

$\{B9\}$, $\{C2\}$, $\{B1\}$, $\{A3, A4, C3, C4\}$, $\{B4, B5\}$, $\{A7, A8\}$, $\{B6\}$, $\{C5\}$, $\{B12\}$

This list is not meant to be exhaustive by any means. Rather, it is to point out key chambers whose elimination would disconnect the graph and leave our puzzle without a solution. Certainly, it means that at least one chamber of each set must be visited to successfully maneuver through the maze. Thus, these sets not only point out key chambers that would disconnect the graph, but also key chambers to be used in any solution of the maze.

In the next chapter, we shift away from the graph-theoretic material and move towards an interesting question answered through computer programming and probability and statistics.

CHAPTER 4

Child's Problem

Before we can consider the question of this chapter, we remind the reader of some useful definitions from probability and statistics.

4.1. Review of Probability and Statistics

We remind the reader of three basic definitions.

DEFINITION 4.1. The **expected value**, $E(X)$ or **mean** μ of a discrete random variable X with sample space S and distribution function $p(x)$ is defined by

$$E(X) = \sum_{x \in S} xp(x)$$

EXAMPLE 4.1. Let an experiment consist of tossing a fair coin three times. Let X denote the number of heads which appear. Then the possible values of X are 0, 1, 2, and 3. The probabilities are 1/8, 3/8, 3/8, and 1/8 respectively. Thus, the expected value of X is

$$0 * (1/8) + 1 * (3/8) + 2 * (3/8) + 3 * (1/8) = 3/2$$

DEFINITION 4.2. The **variance** $Var(X)$, or σ^2 of a random variable X with expected value $\mu = E(X)$ is given by

$$Var(X) = E((X - \mu)^2)$$

The **standard deviation** σ of a random variable X with variance $Var(X)$ is simply $\sigma = \sqrt{Var(X)}$.

We will now prove a theorem that gives us a useful alternative form for computing the variance.

THEOREM 4.1. *If X is any random variable with $E(X) = \mu$, then $Var(X) = E(X^2) - \mu^2$.*

PROOF. We have,

$$\begin{aligned} Var(X) &= E((X - \mu)^2) = E(X^2 - 2\mu X + \mu^2) \\ &= E(X^2) - 2\mu E(X) + \mu^2 = E(X^2) - \mu^2 \end{aligned}$$

□

This theorem will be useful when we compute the variance in the program at the end of this chapter.

4.2. The Question

We are now ready to ask a natural question for the less competitive puzzle-players who just pick up and try out the puzzle: *If we were to randomly go through the maze, allowing backtracking to chambers already visited, on average how long (i.e. how many moves) would it take to reach the end?*

In our question, we understand a *move* to consist of actually passing the KO ball from one chamber to the next, which may require any number of rotations, twists, and turns to successfully execute. We call this the **Child's Problem**, because this is how a child, or perhaps most people, would start exploring the puzzle. Most of us just begin playing without any strategic plan, moving to whatever chambers we can somewhat randomly. We still try to reach the end, but sometimes perhaps we need to backtrack or we get lost. To solve the Child's Problem, we will use computer programming to simulate movement through the maze, and take an average of many trial runs.

4.3. Limitations

Before we explore the algorithm to solve the Child's Problem, we name the limitations of our approach. Our biggest limitation is that our algorithm will not simulate any human intelligence. For example, our algorithm will allow a player to get stuck going back and forth between the same two chambers, or perhaps around the same three chambers. This would add many moves to our count and take much longer to finish the puzzle. Ideally, though, a human being would not choose such moves, unless they wanted to backtrack and try another path through the maze.

Furthermore, as our graph of the *KO Labyrinth* in Figure 1 of Chapter 2 shows, there is only one path from the starting chamber $C1$ to chamber $B1$, and there is only one path at the end of the maze from chamber $B6$ to the exit chamber $C6$. Therefore, one way to produce more realistic results is to run our algorithm only on the "middle" of the maze, counting the number of moves starting at chamber $B1$ and ending at $B6$. We then assume that the smart player will find her way correctly through the single path at the beginning and end of the maze.

We will now explore the algorithm to solve the Child's Problem. For the javascript code, see Appendix 7.4.

4.4. The Child's Algorithm

To understand the Child's algorithm, we begin with a definition:

Definition 4.3. An **adjacency matrix** of a graph is a $(0, 1)$-matrix with a 1 in entry (i, j) if vertex x_i and vertex x_j are adjacent and a 0 otherwise.

The adjacency matrix for the graph of the *KO Labyrinth* is a 26×26 matrix and appears in Appendix 7.3. Again, if there is a 1 in entry (i, j) this means that we can pass the KO balls through chamber i to chamber j. We use this information in our algorithm, which we are now ready to explain.

Child's Algorithm:

1) Begin in chamber $C1$, which corresponds to row $i = 20$ in our adjacency matrix. Set a counter variable $c = 0$.
2) Randomly generate an integer from 0 to 25 to choose another chamber for your column j of the adjacency matrix. This is the chamber to which we will try to move.
3) If the entry of (i, j) is zero, then this means that chamber i and chamber j are not adjacent, so we cannot move from chambers i to j. So, we repeat step 2) and randomly generate another integer for j from 0 to 25. If the entry of (i, j) is one, then this means we can move from chamber i to chamber j, so we do so. Then we let j be the new i and we increment the counter c by one. In less formal language, if the holes in the respective chambers line up properly so that the ball may pass through to the next chamber, make that move.
4) Repeat step 2) and 3) until you reach chamber $C6$, which is $i = 25$, the end of the maze.
5) Repeat the entire simulation at least 10,000 times and take the average.

4.5. Outcome

Counting the number of moves in each trial run and taking the average of $10,000$ trials, we get a result of about **340 moves**. This means that a player moving randomly through the maze will move the ball between different chambers 340 times before finding the end of the maze. Even with $10,000$ trials, we note that this number is just an estimate. Our simulation will give an average of anywhere from 330-350 moves. There is huge variance in each trial as well, where the lowest of just one run through the maze takes only 16 moves, and the highest takes more than $2,000$ moves. The calculated variance σ^2 is around $100,000$ ($\pm 1,000$, for this number is not exact either) and the standard deviation σ is about 317.

These results are significantly large and show, as expected, moving completely randomly through our maze is not the best way to reach the end. Recall that in Chapter 3, we found that the shortest path through the maze is 10 moves, which is significantly smaller than the estimated average of 340 moves.

Recall again the improvement mentioned in Section 4.3. Instead of beginning at chamber $C1$ and ending at chamber $C6$, we begin at chamber $B1$ and end at chamber $B6$ for a more realistic result. When altering the program to take this improvement into account, we find that it takes on average **64 or 65 moves** to go from $B1$ to $B6$ with variance σ^2 at about $3,500$ and standard deviation σ at about 58. We then add the 6 moves $C1$ to $B1$ and $B6$ to $C6$ at the beginning and end of the maze that we assumed the smart player could navigate through without a problem. Adding these 6 extra moves, we have an average of **70 or 71 moves**. This is obviously significantly lower than the average of 340 we obtained above. Still, this is significantly larger than the shortest path of 10 moves.

Now that we have analyzed the puzzle through programming and statistics, we move back to some graph theoretic material in the next chapter. We consider a question related to a famous optimization problem: the Traveling Salesman Problem.

CHAPTER 5

Traveling Salesman Problem

We now ask a version of the traveling salesman question for our graph, but first, a definition.

Definition 5.1. A **Hamilton circuit** is a circuit that visits each vertex in a graph exactly once.

The traveling salesperson problem is a classic question that was first formulated in combinatorial optimization in 1930. While first treated mathematically, the problem has been of great interest to computer scientists as well. It has many practical applications in planning and logistics and can be adapted for other uses, such as in DNA sequencing.

The problem is the following: Given a list of cities to visit and their pairwise distances, a salesman wishes to find the shortest possible route to visit each city exactly once and return home. To do so, he seeks a minimal-cost Hamilton circuit in a complete graph with weighted edges. Vertices in the graph represent cities and weighted edges represent the distance or cost of traveling between the cities. The graph is complete because we are able to travel from one city to any other city.

It turns out that we cannot actually pose the traveling salesman question for the graph of our puzzle for the following reasons:

1) The KO graph is not a complete graph.
2) We are not looking for a circuit (we do not return home).
3) We can see very easily when examining chambers $A5$ and $A6$ that the problem fails.

We show our graph again in Figure 1 on the next page to illustrate reason 3). Since chambers $A5$ and $A6$ both have degree 1 and are incident to chamber $B4$, certainly, we can see that visiting both chambers $A5$ and $A6$ would require us to visit chamber $B4$ more than once.

5.1. A Related Question

Since we cannot ask exactly the traveling salesman question, we instead ask a similar question.

Instead of seeking a minimal-cost Hamilton circuit through a complete graph, we want to see if we can find a **Hamilton path**, that is, a path that visits each vertex exactly once. We know, though, because of chambers $A5$ and $A6$ that we cannot visit

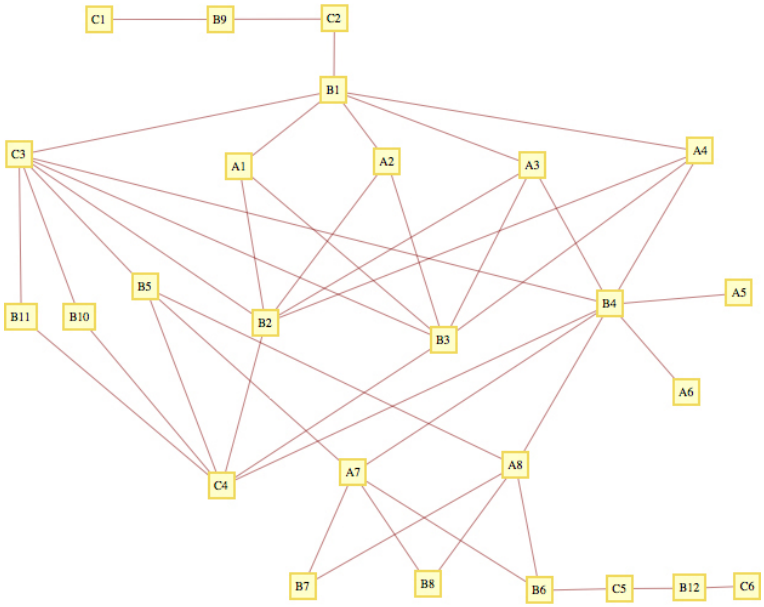


FIGURE 1. Graph of the KO Labyrinth

every chamber exactly once. Therefore we modify the question: *What is the least number of vertices we must remove so that we can have a Hamilton path through our graph?* As we will now see, the answer is **5**.

5.2. Hamilton Paths

First of all, finding a Hamilton path is an **NP-complete** problem. This means it can model the satisfiability problem and can be modeled by it. The satisfiability problem asks whether a logical proposition in n variables is valid. There is no fast algorithm known for evaluating a logical expression for all possible 2^n values of true and false for each variable. Most people believe that a faster, polynomial algorithm will never be found for an NP-complete problem. Other NP-complete problems include graph colorability and finding the size of the largest complete subgraph. It is unknown if determining whether two graphs are isomorphic is an NP-complete problem or not.

While finding a Hamilton path is a difficult problem to consider, we do have some basic rules that we know for finding such a path (or showing that one does not exist). The following three rules will be useful in our search. The underlying idea off which these rules are based is that any Hamilton circuit must contain exactly two edges incident to each vertex.

1. If a vertex x has degree 2, both of the edges incident to x must be part of any Hamilton circuit.
2. No proper subcircuit, that is, a circuit not containing all vertices, can be formed when building a Hamilton circuit.

3. Once the Hamilton circuit is required to use two edges at a vertex x, other (unused) edges incident at x must be removed from consideration.

We will see that rule 1 is particularly useful to us as we search for a Hamilton path through the graph of our maze.

5.3. Using Mathematica

We will use Mathematica to help us determine which vertices we must eliminate in order to have a Hamilton path. In particular, we will use the *FindShortestTour* command. This command in Mathematica actually solves the traveling salesman problem on complete weighted graphs. So, we will adjust the code to "trick" Mathematica into answering our question.

We will make the following adjustments:

1) We first need a weighted graph, so we must set weights for our edges. Since the traveling salesman problem considers a complete graph, then we must assign weights to our graph as if it were complete. We cannot force the *FindShortestTour* command to use only the edges on our graph. Instead, we make it significantly costly to travel between vertices that are not adjacent so that Mathematica will not want to do so. So, if there is an edge between two vertices, the weight is equal to 1. If there is not an edge, set the weight to a high number, say 1,000.

   The way we input this information into Mathematica relates to an adjacency matrix for the graph.[1] Recall that there is a 1 in the (i, j) entry of the matrix if vertices i and j are adjacent and a 0 otherwise. In essence, we take our adjacency matrix filled with 1's and 0's and change the zeros to 1,000. We input each (i, j) entry into an array, which becomes our matrix of "distances."
2) We know that the salesman at the end of the problem returns home. That is, the *FindShortestTour* command will create a circuit rather than a path through our vertices. Therefore, we want to force $C1$ and $C6$ to connect so that the program will create a circuit that we can recognize as a path starting at chamber $C1$ and ending at $C6$. Therefore, we set the weight between those two vertices to 0 so that the program will always choose that move.

Now, running the program several times reveals four problematic sections of our graph.

[1] See Definition 4.3 and the adjacency matrix in Appendix 7.3

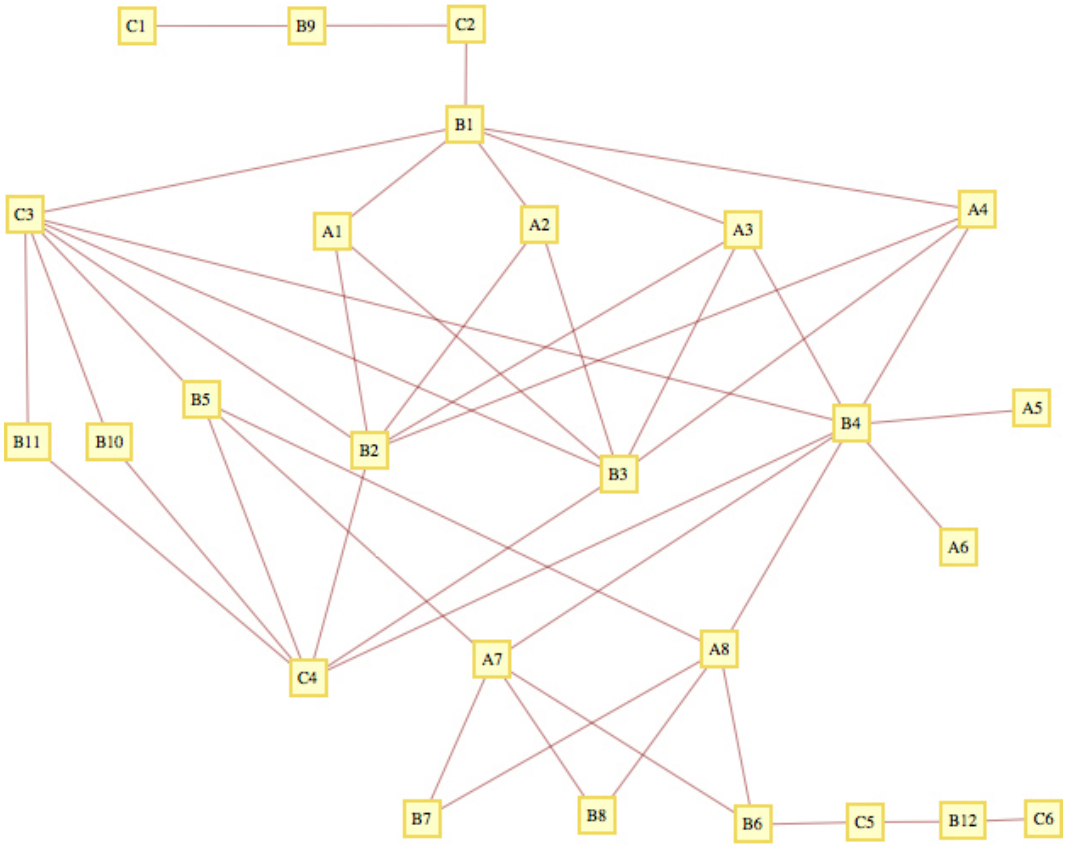


FIGURE 2. Graph of the KO Labyrinth

The first problematic portion of our graph is, as we have already said, chambers $A5$ and $A6$. We cannot have these two vertices of degree 1 and visit chamber $B4$ only once. Therefore, we are forced to get rid of these two vertices. This means we already must throw out two vertices to possibly have a path that visits every vertex exactly once.

The second and third problematic sections of our graph are around chambers $B10$ and $B11$ as well as $B7$ and $B8$. We remind the reader that these chambers are pairs of twins, so they are adjacent to the same vertices as their twin. Each of these vertices have degree 2. However, recall that rule 1 of Hamilton circuits requires that both edges of a vertex with degree 2 must be used in the Hamilton circuit. This would imply, then, that both edges incident to $B7$, edges $(A7, B7)$ and $(A8, B7)$ must be part of the circuit. Also, both edges incident to $B8$, edges, $(A7, B8)$ and $(A8, B7)$, must be part of the circuit. This is not possible, because it would require us to visit chambers $A7$ and $A8$ more than once. Thus, we must toss out one of these chambers, say $B8$. The same scenario occurs with $B10$ and $B11$. We cannot visit both chambers without visiting chambers $C3$ and $C4$ more than once. So, we must toss out another chamber, say $B11$. This now leaves us with a total of four chamber that we must

remove in order to find a path through the KO Graph that visits each vertex exactly once.

The fourth and last problematic section of our graph involves chambers $A1$ and $A2$. This issue is difficult to see when considering the full graph. Therefore, we will focus our attention on a subgraph containing these two vertices to illustrate the difficulty.

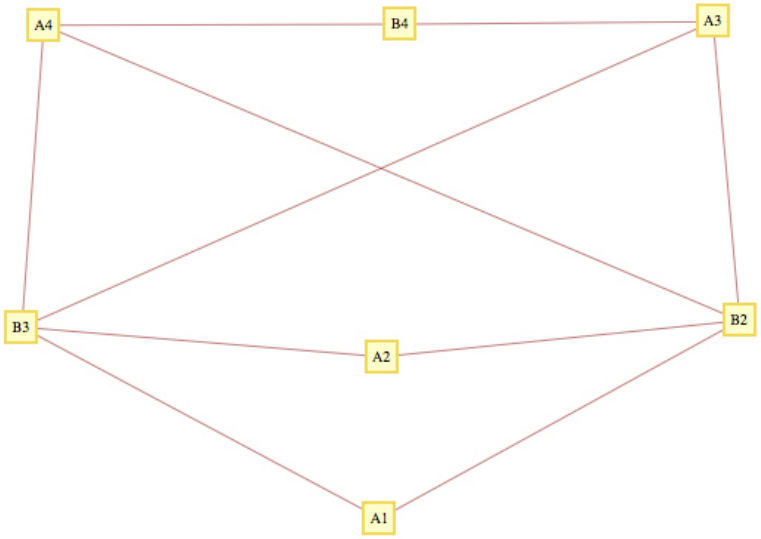


FIGURE 3. Subgraph of the KO Labyrinth

We run our manipulated Mathematica code without the four problematic chambers we have already eliminated. Although chambers $A1$ and $A2$ have degree 3 in the original graph, in our subgraph we notice that those two chambers only have degree 2. This is because the path suggested by Mathematica with our *FindShortestTour* command has us move from chamber $B1$ to chamber $A4$ to create the least costly path through the maze. By rule 3, we see that all other edges incident to $B1$ must be removed from consideration. That means the edges $(B1, A1)$ and $(B1, A2)$ are disregarded. This leaves chambers $A1$ and $A2$ with degree two. Then, as we have seen, by rule 1, both the edges of those chambers must be a part of our path. This would require us, though, to visit chambers $B2$ and $B3$ more than once. Therefore, one chamber, say $A2$, must be tossed out.

Now that we have been forced to toss out 5 chambers, we can create a Hamilton path.

Notice first, however, that all of the chambers that caused problematic sections of our graph were actually twins of each other. With the exception of chambers $A5$ and $A6$ where we were forced to remove both chambers, removing one twin from the pair allowed us to solve the problem. It is interesting to notice that the redundancy

inherent in the graph is what caused the problems and removing some of that redundancy was enough to create a desirable scenario. So, our Hamilton path, which does not visit the five chambers mentioned is given below:

$C1$-$B9$-$C2$-$B1$-$A4$-$B4$-$A3$-$B2$-$A1$-$B3$-$C4$-$B10$-$C3$-$B5$-$A8$-$B7$-$A7$-$B6$-$C5$-$B12$-$C6$.

We can, of course, switch between the different twins to create a different path. For example, we could use chamber $A2$ instead of $A1$. Since there are three sets of twins of which we only use one twin in our path, swapping twins in different ways results in $2^3 = 8$ different modified Hamilton paths that pass through all but five chambers. Thus we have found not one, but eight related versions of a Hamilton path that visits 21 of our 26 chambers and we have justified the elimination of the five chambers based on our rules for Hamilton circuits.

CHAPTER 6

Conclusions

We have seen mathematics be an incredible tool for understanding, solving, and exploring the puzzle the *KO Labyrinth*. We saw how simply creating a graph from the maze helped us navigate through the maze almost immediately. We saw that the shortest path through the maze requires 10-moves using Dijkstra's Algorithm. Furthermore, there are many such 10-move paths, counting 8 in total. We asked the Child's Problem, considering a simulation of randomly moving between the chambers. We concluded using programming that it takes an average of 340 moves to finish the maze if we randomly move through it. We also created an improved version of the code that gave us an estimate of 74 moves through the maze. Still, this is significantly higher than what we know is the shortest path: 10 moves. We then saw that it is impossible to visit all the chambers exactly once, but with the help of Mathematica, we determined that we can create a Hamilton path through the maze that visits 21 of the 26 chambers.

Further research on this puzzle might consider different kinds of questions. For example, what can be said about the colors in each chamber? We know the colors help us match holes through which we pass the ball from one chamber to the next, but is there a reason for specific colors? Can we eliminate using certain colors all together and still maneuver through the maze? Another question one might ask involved the mechanics of the puzzle. While in this paper we have defined a "move" as actually passing the KO ball from one chamber to the next, another question might consider the rotations, twists, and turns, a number of which are required to pass the ball between chambers, as moves. What can be said about how many rotations, twists, and turns are needed to solve the puzzle, given a certain configuration?

While there is still much room for exploration, it is clear that using mathematics to solve and understand puzzles is not just of great use, but great fun as well. Whether we play competitively, casually, or even aimlessly, we are not just playing games. We are exploring possibility in the language of mathematics.

CHAPTER 7

Appendix

7.1. Information on the Chambers

The following is a list of each chamber, its colors, and the degree of the vertex in the graph:

- $A1$, pink, pink, 3.
- $A2$, pink, pink, 3.
- $A3$, pink, yellow, 4.
- $A4$, pink, yellow, 4.
- $A5$, yellow, yellow, 1.
- $A6$, yellow, yellow. 1.
- $A7$, yellow, green, 5.
- $A8$, yellow, green, 5.
- $B1$, blue, pink, 6.
- $B2$, red, pink, 6.
- $B3$, red, pink, 6.
- $B4$, yellow, red, 8.
- $B5$, green, red, 4.
- $B6$, green, purple, 3.
- $B7$, green, gray, 2.
- $B8$, green, gray, 2.
- $B9$, orange, 2.
- $B10$, red, gray, 2.
- $B11$, red, 2.
- $B12$, blue, purple, 2.
- $C1$, orange (entrance), 1.
- $C2$, blue, orange, 2.
- $C3$, red, blue, 7.
- $C4$, red, 6.
- $C5$, blue, purple, 2.
- $C6$, blue (exit), 1.

7.2. Depth-first search

To use a *depth-first search* to find a spanning tree, rather than using the breadth-first search we used for the *KO Labyrinth*, first pick some vertex x as the root and begin to build a path from x made of edges connecting vertices of the graph. Continue

the path until it cannot go any further without repeating a vertex already in the tree. The vertex where the graph must stop is called a **leaf**. Now, backtrack to what is called the **parent** of the leaf, one vertex up the tree. Try to build a path from that vertex y in another direction. Continue to create paths, adding edges and vertices not already in the tree and backtracking to a parent vertex when needed, until we come back to the root and have checked all possible paths from the root.

7.3. The Adjacency Matrix

The following is the adjacency matrix of the graph of the *KO Labyrinth*. Recall that a 1 in the (i, j) entry means that the ball can pass from chamber i to chamber j, and a 0 in that entry means the ball cannot pass between the chambers. Notice that the matrix is symmetric.

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

7.4. The Child's Problem Program in Javascript

Here, we include the program we used to answer the Child's Problem. This program was written in javascript and executed using a browser. Before the script given

below, the information of the adjacency matrix is programmed in as well, using the following bit of code to create arrays in arrays.

```
// Create array to hold adjacency matrix D.
var D = new Array(26);
for (i = 0; i < 26; i++){
     D[i]= new Array(26);
}
```

Then, the programmer can assign values of 0 and 1 to the matrix by brute force. The rest of the code is shown below. Basic html script tags must be added as well for execution in the browser.

```
//Our starting chamber is C1, which is the 21st chamber (minus one since we
// start at zero).
//Our ending chamber is C6, which is the 26th chamber (minus one).

var total = 0*1;
var start;
var counter;
var random;
var vtotal = 0*1;
var squared;
var trials = 10000;

//We want to do this some 10000 times and average it.
document.write("<p>" + "Some trials:" + "</p>")
for(var j = 0; j ¡ trials; j++){
     start = 20*1;
     counter = 0*1;
     while(start != 25){
          //Choose a column at random.
          random = Math.floor(Math.random()*(26));
          // If the start chamber and the random chamber do not
          // connect, choose another.
          while (D[start][random] == 0){
          random = Math.floor(Math.random()*(26));
          }
          //Once you get a 1, continue.
          start = random;
          counter = counter + 1;
     }
squared = counter*counter/trials;

// Print some of the trials.
```

```
if(j % 200 == 0){
document.write("<li>" + counter + " , " + squared + "</li>");
document.write(" ");
}

vtotal = vtotal + squared;
total = total + counter;
}
var average = total/trials;
var variance = vtotal−(average*average);
var standard = Math.sqrt(variance);

document.write("<p>" + "The average is " + average + "</p>");
document.write("<p>" + "The variance is " + variance + "</p>");
document.write("<p>" + "The standard deviation is " + standard + "</p>");
```

To program the alternative version of the program where we only go from chamber $B1$ to chamber $B6$, simply change the starting value to start = 8*1 and the end value in the while loop to start != 13. Additionally, change the adjacency matrix so that chamber $B1$ is no longer connected to chamber $C2$ and chamber $B6$ is no longer adjacent to chamber $C5$.

Bibliography

A. Tucker, *Applied Combinatorics*, 5th Ed., John Wiley & Sons, Inc., New Jersey, 2007.

R. Toal & J.D.N. Dionisio, *Javascript Programming Language*, Jones and Bartlett Publishers, Sudbury, MA, 2010.

S. Ghahramani, *Fundamentals of Probability: With Stochastic Processes*, 3rd Ed., Pearson Education, Inc., Upper Saddle River, NJ, 2005.