Chapter 5
Mathematical Games

5.1 Introduction

We now change our focus from classical game theory to the theory of mathematical games. Mathematical game theory covers a broad spectrum of games with relatively simple rules and constraints, but usually reveals deep mathematical structure. There are two books, *Winning Ways for your mathematical plays* by Berlekamp, Conway and Guy written in 1982, and *Fair Game*, by Guy in 1989, that offer a wide variety of interesting mathematical games and analyses. This chapter will focus on mathematical games with perfect information. There are many other mathematical games with imperfect information that are just as interesting, but far more complex to analyze.

5.1.1 The Subtraction Game

The subtraction game, also known as the takeaway game, invented by Thomas S. Ferguson, is a game with two players.

**The Subtraction Game**

- There are two players. Player 1 makes the first move, and Player 2 makes the second move.
- There is a pile of 10 tokens between the players.
- A move consists of taking 1, 2, or 3 tokens from the pile.
- The player that takes the last token wins.
Let's define the **game state** of the subtraction game as how many tokens are still left in the pile, and which Player is going to make a move. We use an ordered pair (Game state, Player) to tell us about the game state. With this definition we can use a technique similar to backward induction to show us the solution to this game.

Assume the game state has changed from (10,1) to (4,2). This means that there are 4 tokens left in the pile, and it is Player 2's turn. If Player 2 takes 1, 2, or 3 tokens (all of her available moves), then there remains 3, 2, or 1 tokens, respectively, in the pile. All of the game states (3,1), (2,1), and (1,1) are what are called **winning states** for Player 1. This implies that (4,2) was a **losing state** for Player 2. If the game state was (4,1), that is a losing state for Player 1 and a winning state for Player 2. We can continue with this style of analysis all the way up to 10 tokens, and we can see that the game states (4,\(i\)), (8,\(i\)) are losing states for Player \(i\), \(i = 1, 2\). Let's focus on the winning states for Player 1 in the 10 token subtraction game. For simplicity, the notation \(X\) will be used to denote the game state \((X,1)\). Consider Figure 5.1. The winning and losing states for Player 1 are given.

Keep in mind that Figure 5.1 is in reference to Player 1. If there are 0 tokens, Player 1 loses. If Player 1 has their turn when the game state has 4 or 8 tokens, Player 1 will lose if Player 2 uses the correct strategy. In all other states, Player 1 can maneuver the game into a losing state for Player 2.

The generalized subtraction game is defined in the following manner. Notice that a repeating pattern forms of length 4, LWWW. If we extended our result to more than ten tokens, it should not be too difficult to see why this pattern extends for any number of tokens. The size of the repeating pattern becomes relevant in the general game.

- Let \(\mathcal{S}\) be a set of positive integers, called the **subtraction set**.

- Let there be \(n\) tokens, and two players, with the convention that Player 1 moves first.
5.2. NIM

- A move consists of removing \( s \in S \) tokens from the pile.

- The last person to move wins.

There is a known result that shows that subtraction games with finite subtraction sets have eventually periodic win-loss patterns [?]. This repeating pattern has a special name, called the **Nim sequence**. There are backward induction style algorithms that will find the Nim sequence under certain conditions, but we don’t need them for our purposes. For example, with the subtraction set \( S = \{1, 3, 4\} \), the Nim sequence is of length seven and is LWLWWW. The Nim sequence and its length are of importance in the following conjecture.

**Conjecture**

Consider a set \( S \) in the subtraction game with finite tokens, \( n \). Use \( S \) to construct the Nim sequence of W’s and L’s of length \( l \). Then the number of tokens \( n \mod l \) determines which player will win with optimal play. If \( n \mod l = k \), and \( k \) is a winning position, then Player 1 will win the game with \( n \) tokens with optimal play. If \( k \) is a losing position, then Player 1 will lose if Player 2 plays optimally.

If we consider the game with \( S = \{1, 2, 3\} \), that has a Nim sequence of length 4. If the game originally starts with 2001 tokens, we do not need to build the pattern out to that point. Instead, if we find \( 2001 \% 4 = 1 \), and 1 is a W position for Player 1, then we know that Player 1 can win the game when it starts with 2001 tokens.

While there is an algorithm for determining the Nim sequence, there is still the following open problem: Is there a way to determine the Nim sequence from a subtraction set directly?

# 5.2 Nim

Nim, invented by Charles L. Bouton in 1901, is an extension of the subtraction game. Rules of Nim:

- There are 2 players, Player 1 moves first.
CHAPTER 5. MATHEMATICAL GAMES

- There are now \( n \) piles of tokens, rather than one. The piles do not necessarily have the same number of tokens, but they all have at least one.

- A move consists of taking a positive number of tokens from a single pile.

- The winner is the player who takes the last token.

We can do some similar analysis to what we did in the subtraction game. We can represent the game state of Nim using an ordered \((n+1)-\)tuplet. The first \( n \) terms in the game state represent the amount of tokens in each pile, and the last term indicates whose turn it is. If we consider the 3 pile game with \( n_1, n_2, n_3 \) tokens and it is Player 2’s move, then the game state would be written as \((n_1, n_2, n_3, 2)\). Similar to what was done in the subtraction game, we will evaluate all positions relative to Player 1 rather than Player 2.

We’ll restrict our analysis to the 3 pile game, but these results can easily be extended to the \( n \)-pile game. First, consider the game state where only one pile has any remaining tokens. This is clearly a winning state, since Player 1 can simply remove the whole pile.

Two non-empty piles have two possibilities. Either the piles have an equal number of tokens, or not. If the piles are equal, that is a losing state. Can you think of why that should be so? If the piles are not equal, that is a winning state, since the player making the move can now make the piles equal.

Three non-empty piles are a good deal more interesting. Consider \((1,1,2)\), \((1,1,3)\), and \((1,2,2)\). Are these winning or losing states for Player 1? It shouldn’t take too long to figure out that \((1,1,2)\) is a winning state, \((1,1,3)\) is also a winning position, and \((1,2,2)\) is also a winning state. Rather than working it out for every single case, there is an easier way to determine the winning and losing states in Nim. We need something called the Nim-sum, which is reminiscent of the Nim sequence, but unrelated.

**Definition 1** The Nim-sum of two non-negative integers is their addition without carry in base 2.

If you need a refresher on how to turn base 10 numbers into their base 2 representations, http://www.wikihow.com/Convert-from-Decimal-to-Binary
offering an excellent tutorial.

**Calculating the Nim-sum**

The following example is meant to clarify how find the Nim-sum of a set of positive integers. Consider 14, 22, and 31 as the numbers whose Nim-sum we would like to find. First, convert all of the numbers into base 2. 

\[ 14 = (01110)_2, \quad 22 = (10110)_2, \quad \text{and} \quad 31 = (11111)_2 \]

Stack the base 2 expansions (order is irrelevant here, since addition is commutative) and add them columnwise, without carry. So,

\[
\begin{array}{cccc}
0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 \\
+ & 1 & 1 & 1 \\
\hline
0 & 0 & 1 & 1 \\
\end{array}
\]

Notice that in the middle column three ones are added, but because we are working in base 2 we need to use modular arithmetic, \((1+1+1) \mod 2 = 1\). The Nim-sum of 14, 22, and 31 yields \((00111)_2\), or 7 in base 10.

**Theorem 1** A game state \((x_1, x_2, \ldots, x_n)\), where \(x_i \geq 0, i = 1, 2, \ldots, n\), is a losing state in Nim if and only if the Nim-sum of its piles is 0.

The nice thing about this theorem is that is works for an arbitrary finite number of piles. If the Nim-sum does not produce 0, then those piles represent a winning position for the player that makes an optimal move. The proof of this theorem, given by C. L. Bouton in 1902, contains the algorithm that shows you what an optimal move will be. Without going into details about the proof, we offer the algorithm below.

**Algorithm for Finding a Winning Move from a Winning Position**

Form the Nim-sum column addition
- Look at the left-most column with an odd number of 1’s
- Change a 1 in that column to a 0
- Change the numbers in the row of the changed 1, such that there are an even number of 1’s in each column

**Note:** There may be more than one winning move based on this algorithm.
5.2.1 Moore’s Nim

There are several variations on the game of Nim, but we will take a look at this one. Moore’s Nim, developed by E.H. Moore in 1910, also has the name $Nim_k$. There are $n$ piles of tokens and the rules are the same as Nim, except that in each move a player may remove as many chips as desired from any $k$ piles, where $k$ is fixed and $k < n$. At least one chip must be removed from at least one pile. He also came up with the following theorem.

**Theorem 2** $(x_1, x_2, ..., x_n)$ is a losing position in $Nim_k$ if and only if when the $x_i$’s are expanded into base 2 and added in base $k + 1$ without carry, the sum is zero.

There is an algorithm similar to Buton’s that allows a player in a winning position to find the optimal move, the only difference being the number produced by the Nim-sum needs to be evaluated in base $k + 1$.

5.3 Sprouts

Sprouts is a game invented by Conway and Paterson in the 1960’s, featured in *Winning Ways for your mathematical plays* and despite its simplicity it caught the mathematical community by storm for a while. Sprouts is another excellent example of how a mathematical game with simple rules can have deep and interesting properties. Sprouts is played in the following manner:

- Sprouts is a two player game, with the convention that Player 1 moves first.
- On a piece of paper, or a computer screen in this age, a finite number of dots are placed in distinct locations on the paper. **Note:** The location of the dots is irrelevant to actual game play, but our brains trick us into thinking that it matters.
- A move consists of drawing an arc that connects one dot to another, possibly itself. This arc may not pass through another dot on the way, and it may not cross itself. Once the arc is completed, a new dot is placed at some point on the arc connecting the dots, that is not either endpoint.
5.3. SPROUTS

- You cannot draw an arc to or from a dot that has degree 3 (meaning there are three arcs already ending at the dot).

- An arc cannot cross another arc.

- The game ends when no more arcs can be drawn.

- The player who draws the last arc wins.

Consider the following example game with three dots, shown in Figure 5.2.

Figure 5.2: The starting configuration for a 3 dot game of Sprouts.

The first player draws an arc from the top dot to the right dot, shown in Figure 5.3. In the middle of this arc, the first player places a dot. This dot counts as already having two arcs sprouting from it.
The second player then draws an arc from the top dot to the left dot and places a dot in the middle of that arc, shown in Figure 5.4.

The first player then draws an arc from the left dot to the right dot and places a dot in the middle of that arc, shown in Figure 5.5.
5.3. **SPROUTS**

The second player then draws an arc from the right dot to the dot in between the top and right dot, and places a dot in the middle of that arc, shown in Figure 5.6. Notice that the right dot and the the dot in between the top and right dot now have circles drawn around them, to indicate they are have three arcs connected to them and can no longer be used in the game.

The first player then draws an arc from the top dot to the left dot, and places a dot in the middle of that arc, shown in Figure 5.7. Similar to the last picture, the top dot and the left dot now have circles drawn around them to indicate they can no longer be used in the game.
CHAPTER 5. MATHEMATICAL GAMES

Figure 5.7: The fifth move for a 3 dot game of Sprouts.

The second player then draws an arc from the middle dot of the first arc connecting the top and left dot, and the middle dot of the second arc connecting the top and left dot, while placing a dot in the middle of that new arc. The two dots used to start the arc have circles drawn around them.

Figure 5.8: The six move for a 3 dot game of Sprouts.

The first player then draws an arc from the right most dot to the bottom middle dot, and places a dot in the middle of that arc, shown in Figure 5.9. The game ends, as the second player cannot make a move, and player 1 wins.
If a game of Sprouts starts with $d$ dots, then the maximum number of moves that can occur is $3d - 1$. Each dot starts with 3 lives (how many curves can be connected to a vertex), and each move reduces 2 lives from the game and adds one new dot with one life. Thus, each move globally decreases the number of lives by 1, and at most the number of moves is $3d - 1$. The minimum number of moves is $2d$, and it depends on creating enclosed regions around dots with remaining lives. Despite the fact that a finite number of dots means there are a finite number of strategies, analysis of Sprouts has required the invention of new representations and some interesting mathematics involving Nimbers, also known as Sprague-Grundy numbers. Games that begin with 2 to 44 dots have been completely analyzed, it is known whether the first or second player will win for those games. The game that starts with 45, 48, 49, 50, 51, and 52 dots are currently unknown. The game with 53 dots is the highest known starting configuration for Sprouts. A list of who has done the analyzing and what game configurations are known can be found at http://sprouts.tuxfamily.org/wiki/doku.php?id=records

The paper written by Julien Lemoine and Simon Viennot, “Computer analysis of sprouts with nimbers”, is an excellent read on this topic, but it involves mathematics beyond the scope of this course. However, one interesting conjecture given by [?], is at least supported by their work:

**The Sprouts Conjecture**
The first player has a winning strategy in the $d$-dot game if and only if $d$
modulo 6 is 3, 4, or 5.

This conjecture has been supported up to 44 dots, but as of this time there is no formal proof, and so it remains an open problem.

5.4 The Graph Domination Game

The game of graph domination, invented by Daniel Ashlock in 2009, is played on a combinatorial graph. There is a similar game of graph domination involving the colouring of vertices and finding minimal spanning trees, but this game does not use those mechanics. The graph domination game is a game played between two players. Each player has a pile of tokens and by convention, one player is labelled Red, the other Blue, and Red moves first. Consider the graph on the next page, we call it a game board.
Each circle on the board, including the entry points, is called a *vertex*. The lines joining the vertices are called *edges*. Two vertices with an edge between them are *adjacent*. The rules of the Graph Domination Game are as follows:

1. Each player has an entry point. The red player’s entry point is red, the blue player’s entry point is blue.

2. On their turn a player may either place one of their tokens on their entry point, if it is currently unoccupied, or they may move one of their tokens from a vertex to an adjacent vertex, except as specified in rule 3.

3. No player may place or move a token into a vertex that is adjacent to a vertex occupied by one of their opponent’s tokens.

4. A player must place or move a token on each turn unless this is impossible because of rule 3. In this case the player does nothing during that turn.

5. The game continues until neither player can move.

6. At the end of the game a player’s score is three point for each vertex occupied by one of their tokens. Each unoccupied vertex is worth one point to a player for each token he has adjacent to that vertex.

7. The player with the highest score is the winner, if the scores are equal the game is a tie.

8. Players may wish to play two games with each player going first in one of the games and add the scores of the two games.

On the next few pages are other examples of game boards.
5.4. THE GRAPH DOMINATION GAME
CHAPTER 5. MATHEMATICAL GAMES
5.5. **EXERCISES**

We need to introduce some notation before we begin our analysis. Although a combinatorial graph can be drawn in many ways, let us assume that the circular structure displayed in the example game boards will be the standard one. Let us define a ring as a group of vertices that rest on a given circle created by edge arcs when the game board is drawn. The **Red Ring** is the ring on which the red entry point is located, and the **Blue Ring** is the ring on which the blue entry point is located.

This game has yet to be analyzed in any real depth. There are still many open questions:

- Does the first mover have the advantage?
- All of the game boards are symmetric, and more importantly, vertex transitive. Is there an optimal number of vertices per ring that promotes game complexity while keeping it interesting?
- How much does edge structure affect the game? The current game boards are all 3-regular (meaning three edges are connected to every vertex). What do 4-regular game boards look like? What strategy should be employed there? Does that strategy deviate significantly from the 3-regular board strategy?
- What affect does the number of rings between the Blue and Red ring have on the strategy employed?

### 5.5 Exercises

1. Find the Nim sequence of the subtraction game that has the subtraction set \( S = \{1, 4, 5, 8\} \). If the game starts with 1234 tokens, which player wins?

2. Find the Nim sequences of the subtraction games with sets \( S_4 = \{1, 2, 4\}, S_5 = \{1, 2, 5\}, S_6 = \{1, 2, 6\} \). Is there a pattern that develops? Does it work for \( S_3 \)?

3. Consider the Nim position \( (14, 11, 22, 21, 17) \). Is this a winning or losing position in Nim? Find all moves that could be made to move the game into a losing state for your opponent if it is your move. Is this a winning or losing position in \( Nim_4 \)?
4. Play a game of Sprouts with 5 dots. Find the winning strategy for Player 1. How many moves does it take to finish the game?

5. Play a game of Sprouts with 6 dots. Find the winning strategy for Player 2. How many moves does it take to finish the game?

6. Consider the Graph Domination Game. Play against a friend (or enemy), and describe what kind of strategy you used. Did you win? Analyze your strategy against your opponent’s, and determine why the winning strategy was superior. Pick one of the open questions and attempt to give an answer.

7. Squares: Two players are facing a square piece of paper that measures 1 metre x 1 metre. The first player draws a square whose center is in the middle of the larger square, such that the sides of the big and small square are parallel. The second player then draws another, larger, square such that the corners of the first square just touch the midpoints of the edges of the second square. The first player then draws a third square larger than the second, such that the corners of the second square touch the midpoints of the edges of the third square, and so on. See the picture below for a visual representation.

Here is the game: The person who draws the square whose edges or corners go beyond the 1 metre x 1 metre edge loses. If the first player is restricted by the rule that they can draw the first square with a non-zero area no larger than 16 cm², can the first player pick a square size that guarantees a win? What size of the initial square guarantees the second player will win?

8. Pack: Two players are facing another square piece of paper that measures 10 centimeters by 10 centimeters. The first player draws a 1
centimeter line segment on the piece of paper such that the every point on the line segment is at least 1 centimeter away from the edges of the square. The second player then draws a new line segment such that every point on the line segment is at least 1 centimeter away from the edges of the square, and every point of the line segment that the first player put down. And so on. Each line placed on the paper must have every point on the line segment at least 1 centimeter away from the edges, and every point of every other line segment already drawn. The last player who can put a line down wins. Is there a strategy that guarantees Player 1 will win? What about Player 2?
Chapter 6

Deck-Based Games

Deck-based games were invented by asking the question, “what if, instead of being able to move any way we wanted, we played prisoner’s dilemma with a deck of cards printed with cooperate and defect?” The answer to this question turned out to be much deeper and more startling than expected. Formally, a deck-based game is created when a restriction is placed on the available moves in a typical mathematical game, such as rock-paper-scissors, or the Prisoner’s Dilemma, 

6.0.1 General Mechanics of Deck-Based Games

A deck-based game is called strict if a player must use all of the cards in his hand. The original version of a mathematical game, such as the Subtraction Game, is called the open version of the game. There exists a transition from deck-based to open games by allowing the total number each move available to a player to grow larger than the number of rounds played. A deck-based game can be derived from games that have sequential or simultaneous play, but we’ll focus on games with simultaneous play in this book. From now on we’ll refer to moves as cards, encapsulating the fact that there are restrictions on both how many times a move may be used, and how many moves are available. A few more definitions:

- A hand is the set of cards currently available to a player.
The deck is the total set of cards available in the game.

A deck can be shared, in games such as Poker or Blackjack.

A deck can be individual, which is a set of cards that one player has access to, such as in games like Magic: The Gathering™.

If the players are choosing cards from a limited subset of the deck, this is called card drafting.

If the players are constructing their individual decks as play proceeds, usually though card drafting, this is called deck building.

### 6.1 Deck-based Prisoner’s Dilemma

As an example of how imposing restrictions on a classic mathematical game fundamentally changes its nature, we now consider the deck-based Prisoner’s Dilemma (DBPD). Let the cards be labelled with C or D, the moves cooperate and defect, respectively. If we limit the game to one round and one of each type of card, we’ve effectively recreated the original Prisoner’s Dilemma game.

If instead we increased the number of rounds and the amount of each type of card available to a player, a novel development occurs. Adding restrictions to the Prisoner’s Dilemma actually creates three new games, dependent on the payoff matrix. Recall the general payoff matrix for the Prisoner’s Dilemma.

\[
\begin{array}{c|cc}
\text{C} & \text{D} \\
\hline
\text{C} & (C,C) & (S,T) \\
\text{D} & (T,S) & (D,D)
\end{array}
\]

\[S \leq D \leq C \leq T, 2C \geq S + T\]

Figure 6.1: The generalized payoff matrix for the Prisoner’s Dilemma.

Let’s begin our analysis with the following assumptions. Assume the game lasts for \( N \) rounds, finite, and that each player has sufficient cards in their hand to play out all \( N \) rounds, satisfying the strict condition. If we assume that both players have the exact same distribution of cards in their hand at the start of the game, an unexpected result happens.
Theorem 3 In the strict DBPD, if both players have the exact same distribution of cards in their hand at the start of the game and do not add new cards to their hand, then their score will be equal regardless of what strategy they employ.

Proof 1 It is not too hard to see the case with one card is true. If we extend to the case where \( k \) cards are in play, that means that each player has some number \( d \) of defect cards and some number \( c \) of cooperate cards, such that \( c + d = k \). Every time a player discards a card, either the card is matched, meaning \( D \) with \( D \) or \( C \) with \( C \), or not matched, \( C \) with \( D \) or \( D \) with \( C \). If there is a match, both players have \( c - 1 \) or \( d - 1 \) of cooperates or defects left, and they both have the same amount of each type of card. If there is a match, both players receive the same payoff of \( C \) or \( D \). If there is no match, then we get the following. Without loss of generality, if Player 1 defects and Player 2 cooperates, Player 1 receives a payoff of \( T \) and Player 2 receives a payoff of \( S \). At some round later on, Player 2 will have one more defect card than Player 1, and Player 1 will have one more cooperate card than Player 2. When those cards are subsequently played, perhaps during the last round of the game, the payoff to Player 1 will be \( S \) and the payoff to Player 2 will be \( T \), thus balancing out the difference in scores. By the end of the game, both players will have the same score. \( \square \)

A natural question to ask based on this is, “If both players receive the same score, what is the point in playing the game?” If there are just two players playing the game, then the answer is “There is no point, we should not play this game.” However, consider the case where there are \( P \) players playing in a round robin tournament. Let your score during the tournament be the sum of the scores from every game you play. Theorem 3 indicates that you will achieve the same score as your partner during a single game with \( N \) rounds, but it says nothing about what the maximum or minimum score of that game will be.

What your maximum or minimum score per game will be depends on the payoff matrix. If \( T + S > C + D \), a player attains the maximum score if they discoordinate with their opponent, playing \( D \)'s against \( C \)'s, and \( C \)'s against \( D \)'s. Coordination with your opponent will produce the minimum score. If \( T + S < C + D \), then coordinating will produce the maximum score, and discoordination will produce the minimum score. If \( T + S = C + D \), then the game is trivial and the maximum and minimum score will be equal, and
all players will receive the same score each game.

An open question that has been partially answered by [?] is, “What is the optimal strategy to employ when the game is no longer strict?” meaning that there are more cards in a hand than number of rounds.

6.2 Deck-based Rock-Paper-Scissors(-Lizard-Spock)

In the classic game Rock-Paper-Scissors (RPS), two players simultaneously choose a move from the set \{Rock, Paper, Scissors\}. We restrict our attention to the zero sum version of RPS. The payoff matrix, with respect to Player 1, for RPS is given below:

$$\begin{array}{c|ccc}
\text{Player 2} & \text{Rock} & \text{Paper} & \text{Scissors} \\
\hline
\text{Player 1} & & & \\
\text{Rock} & 0 & -1 & 1 \\
\text{Paper} & 1 & 0 & -1 \\
\text{Scissors} & -1 & 1 & 0 \\
\end{array}$$

For the purpose of analyzing the available moves in a deck-based game, it is useful to employ a directed graph representation, as in Figure 6.2. We refer to directed graphs as digraphs.

![Figure 6.2: The directed graph representation of Rock-Paper-Scissors. R for Rock, P for Paper, S for scissors.](image)

The digraph almost instantly conveys exactly how RPS works without the need for a payoff matrix. When using a digraph representation, specifying
the payoff to the winner and loser is necessary and can be done using a typical payoff matrix, or assigning a pair of weights to each edge in the digraph. Notice that there are no arrows drawn from a move to itself; this is because a move cannot win or lose against itself, in this particular game. Since each edge is worth the same amount we leave the weighting out of the diagram. The convention for most games is that the payoffs for Losing, Tieing, and Winning are \( L \leq T \leq W \), and at least one of those inequalities should be strict for a game to be at least mildly interesting. We can use a digraph for more complex games, like Rock-Paper-Scissors-Lizard-Spock, invented by Sam Kass and Karen Bryla, http://www.samkass.com/theories/RPSSL.html. Consider the digraph in Figure 6.3.

![Figure 6.3: The directed graph representation of Rock-Paper-Scissors-Lizard-Spock. L is for Lizard, and K is for Spock.](image)

The digraph representation also allows us to analyze a game rapidly. In both the RPS and RPSLK digraphs, it quickly becomes clear that no move is dominant, and we can formalize this with the concept of a local balance.
The *outdegree* of a vertex is the number of arrows leaving that vertex, which is the number of moves that move defeats. The *indegree* of a vertex is the number of arrows incoming to that vertex, which is the number of moves that defeat that move. The local balance is the ratio of a vertex’s outdegree over its indegree. Notice that in both RPS and RPSLK, the local balance of each vertex is 1. We call a game *balanced* if this is the case. Not every game can, or should, be balanced. We consider Rock-Paper-Scissors-Dynamite (RPSD), where Dynamite can beat all other moves and is not defeated by any move. RPSD is pictured in Figure 6.4.

![Figure 6.4: The directed graph representation of Rock-Paper-Scissors-Dynamite. D is for Dynamite.](image)

The local balance of the Dynamite move is $\infty$, and the local balance of R, P, and S is $\frac{1}{2} = 0.5$. Some things we can conclude from using the digraph’s local balance to analyze a game.

- **Moves with a local balance of $\infty$ are unbeatable.** If you are designing a deck-based game, include moves like this sparingly, or not at all.
6.3. DECK-BASED DIVIDE THE DOLLAR

- Moves with a balance of 0 (where they have some indegree > 0 and an outdegree of 0) are worthless moves and should never be included in a game, unless there is some extra point to consider.

- Moves with a balance of one, or close to it, should make up the bulk of the deck in a game.

- It is important to consider the cycles created by a deck-based game. In RPS, there is only one Hamilton cycle of edge length 3 in one direction, but there are two Hamilton cycles of edge length 5 in RPSLK. The relationship of cycle length to number of players in a game should be considered.

6.3 Deck-based Divide the dollar

Deck-based Divide the Dollar (DBDD) is a card game based off of Divide the dollar and is another example of taking a classic mathematical game and applying restrictions to it.

6.3.1 Divide the dollar

Divide the dollar is a simplified version of Nash’s Bargaining Problem [?]. One dollar sits on the table between two players, and the players simultaneously make demands on how much of the dollar they want. If the sum of their demands are equal to or less than a dollar, they both receive their demand. If the sum of their demands exceeds a dollar, they both receive nothing.

Divide the dollar can be generalized in a number of ways. Modifying the goal amount is one, and an infinite family of generalizations of divide the dollar to any number of players appears in [?].

6.3.2 How to play DBDD

Deck-based Divide the Dollar, or DBDD, has the following rules:

- The game is played by $N$ players.
There is a goal deck that contains a distribution of goal amounts placed on cards. This distribution is dependent on how many players are involved in the game, and on the possible distribution of the players’ hands.

Each player has a deck to draw from, and draws some number of cards from their deck at the beginning of the game. When a player discards, they draw a new one from their deck.

During a given round, each player discards a card. If the sum of the played cards is less than or equal to the goal card, each player receives a payoff equal to the value of the card they played. If the sum exceeds the goal card, each player receives a payoff of zero.

The player with the highest score at the end of the game is the winner.

Game play can be simultaneous, or sequential, with the player who goes first rotating around the player group.

DBDD has novel dynamics to consider. In the open game of Divide the dollar, where a player may bid any value, the Nash equilibrium is that all players should bid the goal divided by the number of players, rounding down if the amounts are discretized. That is no longer the case, since the moves are restricted by the cards available in a player’s hand. If a player has an idea of the distribution of goal cards and the distribution of cards in the other players’ hands, they may be able to make an informed decision about optimal strategy. Sequential versus simultaneous play also drastically changes the game and how strategies are implemented. It is clear that the player who plays the last card has tremendous advantage over the player who plays the first card. Effectively, they often will control whether or not the goal card will be exceeded. However, that analysis still needs to be completed at this point.

6.4 FLUXX-Like Game Mechanics

Fluxx, created by Looney Labs, is a very entertaining card game. The cards that are played modify the conditions under which the game can be won, as the game is played. So the payoff for any given card is determined by in-game conditions that change as the game is played. Deck-based games are
interesting in and of themselves, but adding a Fluxx inspired game mechanic can move the deck-based game in a very different direction than the original open version. If we allow certain cards to change the payoff of moves, then we are broadening the possibilities of events that can occur in the game.

6.4.1 Fluxx-Like Deck-Based Prisoner’s Dilemma

At the risk of our acronyms getting out of hand, FDBPD has Fluxx-like cards as part of the deck from which players are allowed to draw. There are many possible choices to make about what sort of cards can change the nature of the game, but we can keep it simple and show the complexity that can fall out of an adjustment like this one. Assume there is a new kind of card available to each player, called a force card. This card is played simultaneously with another card, so a player using a force card discards two cards in the same round. A force card allows a player to change their opponent’s move. So if Player 1 throws a defect and a force card, and Player 2 also throws a defect, then Player 1 can use the force card to change the value of Player 2’s card to cooperate. The force card does not physically change the cards, rather it changes the payoffs to each player. So instead of the payoff being (D,D), instead it becomes (T,S). If both players use a force card in the same round, then the payoff is (0,0). This addition of this fairly simple mechanic changes the game dramatically.

When we considered the case that $T + S = C + D, T > S, C > D$, in the DBPD, each player achieved the same score and that score did not depend on the strategy either player used. If we introduce the force card, then we will find that the strategy used by the players can matter in the final score. We can show this with an example: each player has 4 cards, 2 C’s and 2 D’s, with 1 force card, and the game is strict. Figure 6.5 shows an example of when the force card changes payoffs to the players.

Notice that Player 1’s total is $2C + S + D$, whereas Player 2’s total is $2C + T + D$, and since $T > S$, Player 2 is the winner of that game. Clearly, the addition of even one new card makes this a completely different game. Adding more Fluxx-like cards can change a game even further.

6.4.2 Fluxx-Like Rock-Paper-Scissors

Fluxx-like Rock-Paper-Scissors can be created using any variety of cards, but for now we consider Rock-Paper-Scissors-Lava-Water-Rust (RPSLWT).
Rock, paper, and scissors have all the usual interactions from the original RPS game, but the other three card types have the following unusual properties.

- Playing Lava defeats all Rock cards, by “melting” them. In the N-person version of RPSLWT, the player who lays down a lava card gets one point for every rock card played by the other players, and more importantly, removes those rock cards from the game so if a player were to gain points by playing a rock card, it would now be the case they receive no points. Any player that played Paper in that round would no longer gain points for beating Rock, since they have been removed.

- The Water card removes all Paper cards, with similar effects Lava has on Rock.

- Rust removes Scissors.

We call these types of cards *negation* cards, for their obvious effects. When analyzing a game with negation cards, we use another kind of arrow to denote the negating move. Figure 6.6 demonstrates the digraph.

The addition of negation moves inherently changes the game, and creates a far richer strategy space to explore. In Figure 6.4.2, a few hands are shown for the 5 player game.

Notice that in the first hand Player 3, who threw Paper as their move, should have had 2 points for both Rocks thrown, but since Player 5 threw Lava, it removed those Rocks from the payoff calculations. In the same way,
6.5. A NOTE ON ADDING NEW MECHANICS TO MATHEMATICAL GAMES

Figure 6.6: The directed graph representation of Rock-Paper-Scissors-Lava-Water-Rust. L is for Lava, W for Water, and T is for Rust. The dotted lines represent one move negating another move.

Hand 3 was affected by the Rust card. In Hand 2, Water was thrown, but no Paper moves were available for it to affect, so nothing was gained by Player 5 in that instance.

6.5 A Note on Adding New Mechanics to Mathematical Games

Given here are just a few examples of how adding an extra mechanic to an otherwise simple mathematical game can drastically change the nature of that game, and increase the size of the game space, often by several orders of magnitude. Extra game mechanics need to be added sparingly and with a great deal of forethought about what they will do to the game, especially Fluxx-like mechanics. This is an area of exploration that is currently wide open, and probably needs new techniques to fully analyze what an added mechanic will do to a game. Repurposing existing techniques, such as the digraph, for analysis is a start, but this area will likely need discrete analytic and computational approaches before any real headway can be made.
CHAPTER 6. DECK-BASED GAMES

Figure 6.7: Three examples of play with 5 players. The payoffs to each player, 1 through 5 left to right, are given underneath the cards.

6.6 Exercises

1. Find the payoffs to the 5 players in FDBRPS in each of the following hands

   (a) RRPSW
   (b) RPSRS
   (c) WPRLT
   (d) TRPS

2. Create a digraph for the following game (bonus points if you can draw it with the minimal number of crossing edges):

   - Move A beats Move C, Move E and Move F, with payoffs of 3, 4, and 5, respectively.
   - Move B beats A and D, payoffs 2 and 7
   - C beats B, payoff of 3
   - D beats A and C, for 3 and 4
   - E beats D and F, for 1 and 2
   - F beats B and C, for 4 and 4

   (a) Find the local balance of each move.
(b) In our work on local balance, the payoffs are not mentioned. Come up with a way to incorporate the payoffs into the local balance of a move. Does this way change the ordering of the local balances of the moves? Does this make sense?

3. Imagine you are playing simultaneous deck based divide the dollar. You and your opponent have finite decks with identical distributions of cards: $\frac{1}{4}$ of the deck is made up of 1’s, $\frac{1}{2}$ of the deck is made up of 2’s, and $\frac{1}{4}$ of the deck is made up of 3’s. You pull your hand of 5 cards randomly from the deck, your opponent does the same. Each deck has 60 cards total. The goal deck also has 60 cards, and has the following distribution: $\frac{1}{4}$ of the cards are 3’s, $\frac{1}{2}$ of the cards are 4’s, and $\frac{1}{4}$ of the cards are 5’s. Come up with a winning strategy for this version of the DBDTD. Write this strategy as a finite state automata.

4. Consider an open mathematical game, or create one on your own, and turn it into a deck-based game. Define the moves and payoffs, draw a digraph, and create a card count for that game, and then write a page that justifies it as being a fun game.
Chapter 7

Tournaments and Their Design

7.1 Introduction

The earliest known use of the word ‘tournament’ comes from the peace legislation by Count Baldwin III of Hainaut for the town of Valenciennes, dated to 1114. It refers to the keepers of the peace in the town leaving it ‘for the purpose of frequenting javelin sports, tournaments and such like.’ There is evidence across all cultures with a written history that people have been competing in organized competitions for at least as long as humanity has been around to write about it. This chapter offers a treatment on tournaments, a useful representation, and some new mathematics involving unique tournaments.

Note: The word tournament has been used in the mathematics of directed graph theory, but those series of works are not related to this topic. A tournament is a name for a complete directed graph, and while that is tangentially related, the research done in this area is not applicable to this treatment.

7.1.1 Some Types of Tournaments

There are a large variety of tournament types, listed here are some commonly known ones.

Elimination Tournaments

Perhaps the most common style of tournament, this style is used in many
professional sports leagues to determine playoff schedules and the yearly winner.

- Most often single or double elimination tournaments are held.
- Teams will play against one another, and are not allowed to play in the tournament if they lose once, or twice, depending on the style of tournament.
- The tournament continues until only one team remains, called the winner.

**Challenge Tournaments**

Also known as pyramid or ladder tournaments.

- Teams begin by being ranked by some other criteria.
- A team may challenge any team above them in the ranking. If the challenger wins, they move into that team’s ranking. Every team in between the winning and losing team goes down the ladder. If the challenging team loses, the ranking does not change, or the challenging team can drop in rank.
- There are variations where teams may only challenge teams that are a specific number of ranks above them.
- The winner is the team with the top rank, after some specified number of games or time.

**Multilevel Tournaments**

Also known as consolation tournaments.

- Similar to single and double elimination tournaments, teams that lose one or two games are shifted down to another tier, where they play other teams that have been eliminated from the original tier.
- Many variations and tiers can accompany this style of tournament.
Round Robin Tournaments

A round robin tournament is the style that takes the longest to complete, and it, or some variation, is most often used in league play during a sports season.

- Teams play every other team in the tournament at least once.
- Rankings are determined by some cumulative score, or total number of wins with tiebreakers predetermined.
- There are many variations to the round robin style tournament, such as the double and triple split.

There are advantages and disadvantages to each style of tournament. Elimination tournaments are best employed when there is a limited number of games that can be played, due to constraints such as time. However, a bad seeding (initial arrangement of teams and games) combined with a poor showing can knock out high quality teams early. The most commonly used heuristic to avoid poor seedings is during the first round have the highest quality team face the lowest quality team, the second highest quality team face the second lowest quality team, and so on. Challenge tournaments provide an interesting method of ranking a team, but can take a long time and because challenges have to be issued, and there is no guarantee that the final rankings will be reflective of the quality of teams. Multilevel tournaments have the advantage of allowing eliminated teams to continue playing, but ultimately share the same drawbacks as elimination tournaments. Round robin tournaments take the most time to complete, but offer the best way to truly evaluate the relative quality of teams. However, a round robin tournament of 16 teams, for example, will take 120 games to complete. With 32 teams, it would take 496 games. In games that involve time and physical constraints, round robin tournaments are not practical for large numbers of teams. However, by bringing automation into tournaments, the round-robin becomes practical again, and is the best way to evaluate a team’s relative quality. The rest of this chapter will focus on tournaments using the round robin style.


7.2 Round Robin Scheduling

The study of round robin scheduling began with the work done by Eric Gelling in his Master’s thesis [?]. Due to the graph theoretic approach of Gelling, we now include some terms that will be useful in dealing with the results of his investigation.

- A factor $G_i$ of a graph $G$ is a spanning subgraph of $G$ which is not totally disconnected.

- The set of graphs $G_1, G_2, ..., G_n$ is called a decomposition of $G$ if and only if:

  \[ \bigcup_{i=1}^{n} V(G_i) = V(G) \]

  \[ E(G_i) \cap E(G_j) = \emptyset, i \neq j \]

  \[ \bigcup_{i=1}^{n} E(G_i) = E(G) \]

- An $n$-factor is a regular factor of degree $n$. If every factor of $G$ is an $n$-factor, then the decomposition is called an $n$-factorization, and $G$ is $n$-factorable.

While there are many interesting results from that work, there are few we will focus on due to their application to round robin tournaments. The following theorem is offered without proof.

**Theorem 4** The complete graph $K_{2n}$ is 1-factorizable, and there are $t = (2n - 1)(2n - 3)...(3)(1)$ different 1-factorizations of the complete graph.

For the graph $K_4$, for example, there are $t = (4 - 1)(4 - 3)(1) = 3$ 1-factorizations, and those are pictured in Figure 7.1.

![Figure 7.1: The three 1-factorizations of $K_4$.](image)
7.3. ROUND ROBIN SCHEDULING WITH COURTS

Adding labels to the vertices, such as the numbers \{1, 2, ..., 2n\} for \(K_{2n}\), complicates matters, because the ordering of the vertices creates equivalence classes of 1-factorizations. Gelling does an exhaustive search for the equivalence classes of 1-factorizations of \(K_6\), \(K_8\), and \(K_{10}\). From a combinatorics and graph theory perspective, this represents an interesting challenge to categorize them all, but from a tournament design angle it is less relevant to our discussions.

**Note:** It is important to be specific in our discussions henceforth. A **game** will defined as one team playing against another team, defined by the 1-factorizations of the complete graph. A **round** will be defined as the set of games defined by the edge set of a single 1-factorization. Without any physical or temporal restrictions, a round robin tournament with 2n teams in which teams play each other once requires \(n(2n - 1)\) games to complete, and 2n – 1 rounds.

**Theorem 5** *In a standard round robin tournament, when \(2n \leq 4\), the number of 1-factorizations is equal to the number of rounds required to complete the tournament. As \(2n \rightarrow \infty\), the ratio of 1-factorizations to rounds approaches infinity.*

What we can take from this theorem is that we will be spoiled for choice in the number of possible 1-factorizations we could use to complete the 2n-1 rounds of a tournament with 2n teams. When there are no other restrictions placed on the tournament, then this is not much of a concern. When we have restrictions, like the number of available courts or fields upon which matches are to be played, then the combinatorially increasing number of 1-factorizations becomes an interesting search problem.

### 7.3 Round Robin Scheduling With Courts

Designing round robin tournaments while taking courts into consideration is an interesting combinatorics and design problem all on its own. Let us use the definitions of rounds and matches defined in the previous section, and add the following:

A **court** is the physical location where a match between two teams occur. If the tournament has 1 court, then this discussion is rather short. Every team plays on that court until the tournament is over. However, if you have more than one court, perhaps of unequal attractiveness, then the problem of
choosing a set of 1-factorizations that guarantees each team will have equal playing opportunities on all of the courts becomes a difficult search problem. In fact, the graph theoretic approach, while defining the 1-factorizations as matches in a round, is no longer sufficient for this purpose. In [?], the general methods of creating balanced tournament designs were established.

7.3.1 Balanced Tournament Designs

Let us consider a round robin tournament with $2n$ players, with $n$ courts of unequal attractiveness. This could be due to quality of the court, location, weather, and a variety of other reasons. The tournament is to take place over $2n - 1$ rounds, and we add the condition that no team plays on any court more than twice.

- A tournament design, $TD(n)$, defined on a $2n$ set $V$, is an arrangement of the $2n\choose 2$ distinct unordered pairs of the elements of $V$ into an $n \times 2n - 1$ array such that:
  - every element of $V$ is contained in precisely one cell of each column
  - no element of $V$ is contained in more than two cells of any row.

A tournament design with these properties is called a balanced tournament design for $2n$ teams, denoted as $BTD(n)$. By letting the columns correspond to player matches and the rows to court assignments, this is a representation of a round robin tournament. Optimally, each team will appear twice in each row, except for the row in which it will appear once. An example of the $BTD(3)$ is given in Figure 7.2.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>4</th>
<th>6</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 7.2: An example of a $BTD(3)$. 
7.3. ROUND ROBIN SCHEDULING WITH COURTS

A way to construct BTD(3) is given in the following algorithm.

Create an initial round of play between the 6 teams, randomly or otherwise. Place the pairs in the first column of the lattice shown in Figure 7.2. Call the first pairing Block 1 (B1), the second pairing Block 2 (B2), and the third block Block 3 (B3). The first team of B1 will be denoted B11, the second team of B1 denoted B12, and so on.

In round 2 of the tournament, B11 plays B21, and is placed in row 3, B22 plays B32 and is placed in row 1, and B12 plays B31 and is placed in row 2.

Round 3: B11 plays B22 row 3, B21 plays B31 row 1, and B12 plays B32 row 2.


Round 5: B11 plays B31 row 2, B21 plays B32 row 1, and B12 plays B22 row 3.

Without loss of generality, the steps in the previous algorithm can be rearranged in any order. There are some notes about generalization that can be mentioned here: notice that after the initial pairing, teams from Bn never appear in row n again. If the number of teams is evenly divisible by three, then we can generalize the algorithm to include a choosing method to create sub-blocks that will fill the lattice. If the number of teams does not have 3 as a divisor, then care will have to be taken in generalizing the algorithm. There are no balanced tournament designs with n being even, and this can be seen using the pigeon-hole principle.

**Theorem 6** There exists a BTD(n) for every odd positive integer n.

In [?] a proof is given for this theorem and a way to construct certain cases of n, but it is heavily accented with group theory beyond the scope of this course.

### 7.3.2 Court Balanced Tournament Designs

We now move on from balanced tournament designs. Notice that the BTD(n) does not guarantee a team will be able to play on their most desirable court as many times as other teams will. To counter this, we move our focus to balancing the number of times a team will play on a given court in a round robin tournament [?]. A court balanced tournament design of n teams with
c courts, denoted $CTBD(n, c)$, is a round robin tournament design such that every element of $V$ (the set of $n$ even teams) appears the same number of times in each row.

- Consider some positive integer $t$, and let $t$ be the number of columns in the new tournament construction.
- Let $\alpha$ be the number of times each team will appear in a given row.

The necessary conditions for the existence of a CBTD$(n, c)$ are:

1. $ct = \binom{n}{2}$
2. $1 \leq c \leq \left\lfloor \frac{n}{2} \right\rfloor$
3. $c\alpha = n - 1$

The third condition forces the court balancing property, and we’ll focus on cases where the number of courts is at least two. The bounds on $t$ and $\alpha$ are given as

$$n - 1 + (n \mod 2) \leq t \leq \binom{n}{2}, \quad 1 \leq \alpha \leq n - 1$$

When $n = 2c$, we are dealing with a $BTD(c)$ and it is impossible to satisfy condition 3. However, assuming $n \neq 2c$, and the 3 conditions are satisfied, we can get the following results.

The first result deals with $CBTD(2c + 1, c)$ for all positive $c$. *Odd balanced tournament design of side $c$*, denoted as OBTD$(c)$, occur when the number of teams satisfies this condition. It is fairly easy to construct the tournament schedule for $c$ courts, and no team will appear more than twice in any row. Consider Figure 7.3.2. In case the pattern is not clear on how to construct the $OBTD(c)$, consider the first row in Figure 7.3.2. The top of the first row has the 7 teams listed in order, and the bottom of the first row has the seven teams listed in order, but shifted so that the bottom starts with 7. These are the pairings for that court. The second row is the result of shifting the teams on the top to the left by one spot, called a permutation, and shifting the bottom row to the right by one position. The third row is created by shifting 2 positions to the left and right, respectively. This technique can be generalized to $2c + 1$
teams. List the teams along the top of the first row from 1 to $2c + 1$, and the bottom of the first row as $2c + 1, 1, ..., 2c$. Then shift the top of the first row to the left 1 position, and the bottom of the first row to the right 1 position. This generates the second row. If you shift 2 positions, you produce the third row, and so on. It only requires $c - 1$ total shifts to produce the $c$ rows necessary to complete the tournament. The following two theorems are offered without proof, but serve as a useful guide.

**Theorem 7** Let $n$ and $c$ be positive integers satisfying conditions 1-3 and $n$ be odd. Then there exists a $CBTD(n,c)$, and $\alpha$ is even.

In fact, this theorem can be extended even further.

**Theorem 8** Let $n$ and $c$ be positive integers satisfying conditions 1-3. Then there exists a $CBTD(n,c)$.

Despite the power offered by these two theorems, the number of pairs $(n, c)$ that actually satisfy these conditions are fairly minimal. For example, with 100 teams only 3, 9, 11, and 33 courts satisfy conditions 1-3.

### 7.4 Cyclic Permutation Fractal Tournament Design

This section is the result of trying to plan a special kind of tournament. A group of high school students were put into teams, and those teams were to compete in an Enhanced Ultimatum Game tournament. There were no courts to play on, as it was all done electronically, but organization of the
tournament was complex. Each team had to play another team as both the Proposer and the Acceptor, so there were to be two meetings between each pair of teams. Also, I wanted every team to be playing during each round (this was only possible because the teams were even), and I didn’t want the teams playing their matches with changed roles back to back. I wanted to minimize the number of rounds as the last objective, since I could only count on so much attention span and time. This led to the generalized question:

**Question 1** Suppose we have \( n \) teams that are to participate in a tournament. A match is played between 2 distinct teams. During a match, each team takes on a single role, offence or defence, and cannot change that role during the match. A round consists of the matches played between designated pairs of teams. Is there a way to organize the tournament such that each team faces one another at least once, with other matches in between, and plays as offence or defence an equal number of times, while minimizing the number of rounds?

To answer this question, we begin by recalling the definition of a permutation:

**Definition 2** A **permutation** \( \sigma \), of a set of \( n \) elements \( X \), \( \sigma : X \to X \), such that \( \sigma \) is a bijection.

For example, given the set \( X = \{1, 2, 3, 4, 5\} \), a permutation could be given by:

\[
\sigma(1) = 5, \quad \sigma(2) = 3, \quad \sigma(3) = 2, \quad \sigma(4) = 4, \quad \sigma(5) = 1
\]

\( \sigma(X) \) would be given as \{5, 3, 2, 4, 1\}.

**Definition 3** Let \( X = \{1, 2, ..., n\} \) and \( \sigma : X \to X \). Let \( x_1, x_2, ..., x_k \) be distinct numbers from the set \( \{1, 2, ..., n\} \). If

\[
\sigma(x_1) = x_2, \quad \sigma(x_2) = x_3, \ldots, \sigma(x_{k-1}) = x_k, \quad \sigma(x_k) = x_1
\]

Then \( \sigma \) is called a **k-cycle**

For a set \( X = \{1, 2, ..., n\} \) a more compact representation of a cyclic permutation is \( \sigma = (1 \ 2 \ 3 \ ... \ n-1 \ n) \), indicating that the element in position 1 will now occupy position 2, the element in position 2 will occupy position 3, and so on, and the element in position \( n \) will occupy position 1. Thus after the permutation is applied, the set will now be \( \sigma(X) = \{n, 1, 2, ..., n-1\} \).
7.4. CYCLIC PERMUTATION FRACTAL TOURNAMENT DESIGN

Figure 7.4: The odd numbers are placed around the inside of the edge of the circle, the even numbers are placed around the outside of the edge of the circle such that all the numbers line up.

7.4.1 Case 1: \( n = 2^k, \ k \in \mathbb{Z}^+ \)

It is easiest to construct a round robin style tournament that satisfies the conditions of Question 1 when the number of teams is a power of two.

Theorem 9 When the number of teams is a power of two, the minimum number of rounds to fulfill the requirement that each team play every other team at least once and take on the role of offence or defence an equal number of times is \( 2^{k+1} - 2 \).

Proof:

The proof is contained within the tournament construction. First, if the teams have not been ordered, uniquely order them using positive integers, such that they form the set \( T = \{1, 2, 3, ..., n\} \). Then, create two new sets, \( T^{O1} = \{1, 3, ..., n - 1\} \) and \( T^{E1} = \{2, 4, 6, ..., n\} \), standing for the first odd team set and the first even team set, respectively. Once the first odd team set and the first even team set are determined, place the teams in the manner drawn in Figure 7.4.

The way the circle has been constructed designates the first round of matches. Without loss of generality, let the odd numbered teams play offence in the first round. Once the first round is complete, we ‘turn’ the circle to the right to create the match ups of the next round, as in Figure 7.5.
In the second round, the even teams take on the role of offence. Each turn of the circle creates a new matching of teams from $T^{O_1}$ against teams from $T^{E_1}$. The number of rounds this construction will produce is $2^{k-1}$, since half of the total number of teams will play against one another, and a team cannot play against itself. Each team will have also taken on the role of offense and defense the same number of times. The circle representation is illustrative, but cumbersome, so I now switch to a representation using cyclic permutations. Let $\sigma_{O_1}$ be the permutation such that $\sigma_{O_1} : T^{O_1} \rightarrow T^{O_1}$ and $\sigma_{E_1}$ be the permutation such that $\sigma_{E_1} : T^{E_1} \rightarrow T^{E_1}$. Fixing $T^{E_1}$ on the line beneath $\sigma_{O_1}(T^{O_1})$, this gives the matches of the first round, and directions on how to construct the matches in the second round, shown in Figure 7.6. Permuting $T^{O_1}$ will create the match ups indicated in Figure 7.7. While keeping $T^{E_1}$ fixed, continually permuting $T^{O_1}$ will generate the same set of matches as the circle construction.

So far, this has generated $2^{k-1}$ rounds in which each team has played offence and defence an equal number of times. The second part of the tour-
7.4. CYCLIC PERMUTATION FRACTAL TOURNAMENT DESIGN

Figure 7.7: A cyclic permutation representation of the circle technique, displaying the second round of matches.

Tournament construction requires splitting the team sets $T^{O1}$ and $T^{E1}$ in half, forming new team sets $T^{O2}$, $T^{O3}$, $T^{E2}$, and $T^{E3}$. All of these new team sets will be of size $2^{k-2}$. Without loss of generality, let

$$T^{O2} = \left\{ 1, 3, \ldots, \frac{n}{2} - 1 \right\}$$

$$T^{O3} = \left\{ \frac{n}{2} + 1, \frac{n}{2} + 3, \ldots, n - 1 \right\}$$

$$T^{E2} = \left\{ 2, 4, \ldots, \frac{n}{2} \right\}$$

$$T^{E3} = \left\{ \frac{n}{2} + 2, \frac{n}{2} + 4, \ldots, n \right\}$$

and fix $T^{O2}$ atop $T^{O3}$, and fix $T^{E2}$ atop $T^{E3}$. This will be the first round of matches. Then cyclically permute $T^{O2}$ and $T^{E2}$ in the manner described in Figure 7.6, while keeping $T^{O3}$ and $T^{E3}$ fixed. Let $T^{O2}$ and $T^{E3}$ take on the role of offence in this next set of rounds, which will total $2^{k-2}$.

Assuming there are still enough teams in the team sets to evenly split, continue the splitting and cyclic permutation (or the circle, if you prefer) construction until there are only team sets of size 1 left. At this point, it should not be difficult to see that $2^k - 2$ rounds have passed. Since there is only one more round of matches left, it becomes clear that half of the teams will have taken the role offence one more time than defence after the final match has been played, giving a total of $2^k - 1$ rounds. This is the minimum number of rounds required to guarantee a team is matched against every other team if the number of teams is $2^k$, since a team cannot play itself. To correct this imbalance, retain all of the orderings of rounds and play all of the matches again, but this time reverse the offence and defence designations of all of the matches. This guarantees that each team takes on the role of
offence and defence the same number of times. Since the number of rounds is doubled, \(2(2^k - 1) = 2^{k+1} - 2\) is the of rounds necessary and sufficient to fulfill the condition that each team faces one another at least once, and plays as offence or defence an equal number of times, while minimizing the number of rounds. \[\square\]

**Case 2: \(n \in \mathbb{Z}^+\)**

The case where the number of teams is not a power of 2 requires the following lemmas:

**Lemma 1** If a team set has two elements, it requires two rounds for each team to have played one another and have taken the role of offence and defence an equal number of times. Each team will have played 2 matches. If a team set has three elements, it requires six rounds for each team in that team set to have played one another and taken on the role of offence and defence an equal number of times. Each team will have played four matches.

**Proof:**
Consider a team set with teams labelled 1 and 2. Without loss of generality, Team 1 takes the role of offence in the first round, Team 2 takes the role of offence in the second round.

Now consider a team set with teams labelled 1, 2, and 3. Without loss of generality, Team 1 is matched with Team 2, then matched with Team 3, and finally Team 2 is matched with Team 3. Due to the definition of round, each match is also a round in this case. Team 1 will take on the role of offence twice, Teams 2 and 3 will take on the role of defence twice. Now play the same matches again, but switch the roles during the matches. Three games from the first set of matches plus three games from the second set of matches is six rounds. Each team plays the other team twice, totalling four matches per team. \[\square\]

**Lemma 2** If \(|T^A| - |T^B| = \pm 1\), where \(T^A\) and \(T^B\) are the two team sets created by splitting an existing team set, then the number of rounds required to fulfill the condition that that each team faces one another at least once, and plays as offence or defence an equal number of times, while minimizing the number of rounds, is

\[
2 \cdot \text{argmax}(|T^A|, |T^B|)
\]
7.4. CYCLIC PERMUTATION Fractal TOURNAMENT DESIGN

Proof:
Without loss of generality, let $|T^A| > |T^B|$. Then we can use the following construction in Figure 7.8. The team from $T^A$ on the end of the cyclic permutation schedule will sit out each round while the teams from $T^A$ and $T^B$ play their matches. If $|T^B| = m$, then it will take $m + 1$ rounds to ensure that all teams from each team set are matched against one another. One round for the initial match ups, and then $m$ iterations of the cyclic permutation on $T^A$. From our initial assumption, $|T^A| = m + 1$. Similar to the idea from the previous case, if we double the rounds played while switching the roles of offence and defence, then each team plays offence and defence an equal number of times. During these rounds, the teams from $T^B$ will play two extra matches than the teams from $T^A$. However, those matches are made up in the next set of rounds, as shown in Lemma 3. □.

\[
\begin{array}{cccccc}
A1 & A2 & A3 & \ldots & A_{n-1} & A_n \\
B1 & B2 & B3 & \ldots & B_n \\
A_n & A1 & A2 & \ldots & A_{n-2} & A_{n-1} \\
B1 & B2 & B3 & \ldots & B_n
\end{array}
\]

Figure 7.8: A cyclic permutation representation where one team set has one more element than the other. Displayed are two rounds.

Lemma 3 Let $|T^A| - |T^B| = \pm 1$. From Lemma 2, the teams from $T^B$ have played two more matches than the teams from $T^A$, or vice versa. After the next round is completed, the number of matches played by all of the teams in those team sets is equal.

Proof:
Begin by splitting $|T^A|$ and $|T^B|$ into $|T^{A1}|$, $|T^{A2}|$, $|T^{B1}|$, and $|T^{B2}|$. Without loss of generality let $|T^A| > |T^B|$ and let $|T^{A1}| \geq |T^{A2}|$ and $|T^{B1}| \geq |T^{B2}|$. Since $|T^A| - |T^B| = 1$, $T^{A1}$ has one more element than $T^{B1}$, then there is one more match that needs to be played amongst the A teams than the B teams. We double the number of matches to equalize the roles of offence and
defence, thus creating two extra matches amongst the A teams than the B teams. Thus, the number of matches played over the course of two sets of rounds is equal. □

**Theorem 10** When the number of teams, \( n \), is not a power of two, the minimum number of rounds to fulfill the requirement that each team play every other team at least once and take on the role of offence or defence an equal number of times is \( 2(n + z) \), where \( z \) is the number of rounds a team split creates teams of uneven size. The exception is when \( n = 7 \) or \( 7 \times (2k), k \in \mathbb{Z}^+ \), or when a team split causes a team size to be seven or an even multiple of seven. If this occurs, this split does not add to the \( z \) count.

**Proof:**
Begin by assuming the number of teams in the tournament is even but not a power of 2. Based on the construction given in the proof of Theorem 9, the number of rounds to play all of the necessary matches will be \( \frac{n}{2} \). After the first set of rounds is complete, then a team split occurs. If the team split results in all teams being of size \( \frac{n}{4} \), then the number of rounds to play all of the matches will be \( \frac{n}{4} \), and so on. However, since \( n \) is not a power of 2, at some point a team split will result in the sizes of the new team sets being odd. Once that occurs, every split afterwards will result in team sets of unequal sizes. Once there are team sets of unequal sizes, we know from Lemma ?? that the number of rounds necessary to play all of the matches in this case is \( m + 1 \), where \( m \) is the size of the smaller team set. Thus, every time a team split creates team sets of unequal size, one more round is necessary to ensure every team plays against one another. Let \( z \) be the number of times a team split creates team sets of unequal size, and let \( n \) be the number of teams in the tournament. Then it requires at least \( n - 1 \) rounds to ensure every team plays against one another. \( n \) is not a power of 2, so during the last round of play there will be at least one team set of size three. From Lemma 1, we know that it requires an extra round for team set of three to play against one another. This extra round means that it will require \( n \) rounds for \( n \) teams to play one another. Adding the necessary rounds created from unequal splits, and the sum is \( n + z \). As it was necessary in the earlier construction, doubling the number of rounds while reversing the roles guarantees satisfaction of the conditions set out in Question 1, thus the number of rounds necessary is \( 2(n + z) \).

If we assume \( n \) is odd, then every round will result in an uneven team split,
and following reasoning from the case where \( n \) is even, every time there is a team split one more round must be added to accommodate the uneven number of teams.

The exception is when a team size is seven. If we split seven, we get team sets of four and three, requiring four rounds to play all of the matches. Once we split again for the next sets of rounds, we now have team sets of size three, two, and two. The teams from the size three split require three rounds to play one another. The two team sets of size two created from the size four split require two rounds to play one another. Team sets of size three do not need to be split, and when the size two teams are split into team sets of size one, they require one round to play one another, still totalling three rounds. Thus, four rounds from the first split plus three rounds from the second and third split total seven rounds. No extra rounds are necessary for a team set of size seven. Any even multiple of seven will have this property as well, since all splittings will eventually create team sets of size seven. □

### 7.4.2 A recursive generation of the minimum number of rounds

There is a recursive method to generate the minimum number of rounds required to play a tournament that satisfies the conditions given in Question 1.

**Theorem 11** Let \( n \) be the number of teams in the tournament and let \( r(n) \) be the minimum number of rounds to complete the tournament. Then

\[
    r(n) = \begin{cases} 
        n + r\left(\frac{n}{2}\right) & \text{if } n \text{ is even} \\
        r(n + 1) & \text{if } n \text{ is odd} 
    \end{cases}
\]

**Proof:**

Consider when \( n \) is even. The first split creates 2 team sets of size \( \frac{n}{2} \). It will take \( 2 \cdot \frac{n}{2} = n \) rounds to play all of the matches required to satisfy the conditions laid out in Question 1. Once that is completed, assuming there are enough teams in the team sets to split again, the required number of rounds will be equivalent to a tournament of half of the original size.

Consider when \( n \) is odd. By Lemma 2, the number of rounds required to have the two team sets play each other is the same as if we had started with \( n + 1 \) teams. □
7.5 Example Tournaments

Three tournaments offered in a game theory class are outlined here.

7.5.1 The Enhanced Ultimatum Game Tournament

The Enhanced Ultimatum game represents an interesting way to both explore mathematical thinking and picking a successful strategy when you have an idea of how other people are going to play. This tournament has gone through two iterations so far. In the initial tournament play, teams would have a face to face interaction, but this was both time consuming and led to bargaining, which changed the nature of the game entirely. In the interest of removing as many confounding factors as possible, we have sinned move the EUG tournament to a strictly computational format. Two versions of this tournament now exist.

Version 1

In this version determination of rounds and matchups is necessary, in the fashion given by cyclic permutation fractal design. Each team faces some other team, taking on the role of Proposer or Acceptor. The Acceptor team delivers a demand electronically to the Proposer, the Proposer responds with a proposal. The minimum accept is predetermined by the Acceptor team, and if the proposal exceeds or equals the minimum accept the deal goes through. Each team’s total score is displayed so every team knows the score of every other team, but not their individual match results. The number of rounds that has passed is also displayed.

Pros:

- Allows teams to develop a sense of strategy over the course of the tournament.
- Generates a sense of excitement amongst the teams doing well, often there is fairly close competition.
- Fosters a team’s need to communicate with one another, and decide on a course of action based on their current score, and the score of their opponent.
- Strategies can be changed during the tournament, so a sense of the evolution of strategies can be determined.
7.5. EXAMPLE TOURNAMENTS

Cons:

- Teams that are losing often lose interest, can sabotage the rest of the tournament with unreasonable demands.

- Even with computational aid, running a tournament with many teams can still take a long time. With 32 teams, for example, 62 rounds have to pass for a complete tournament to take place. If each round allows 5 minutes total for deciding demands and proposals, we are still looking at 310 minutes, or just over 5 hours.

- Requiring repeated user input is rarely a smooth process. This can be alleviated by using techniques such as default values, but that takes away from the whole point of the tournament.

For the most part, version 1 only works for a smaller number of teams, and even then it can be problematic.

**Version 2**

Version 2 of this tournament requires that teams secretly submit an array of values, or a function, \( p = f(d) \) (proposal as a function of the demand), along with a minimum accept and a demand to the tournament. These values are static, and do not change. Assuming that there are twenty dollars on the table to split, teams submit an array of 21 values. The first 19 values are what they would propose if they received a demand of that value’s position. So for example, if the value in position 1 was 1, that would mean the team would offer 1 if the other team demanded 1. If the value of position 10 was 8, that would mean the team would offer 8 if the other team demanded 10, and so on. The last two values are the minimum accept and the demand. If teams submit a function, they must specify the function and their minimum accept and demand. Once every team has submitted their functions or their arrays, the tournament can be run using a computer. Roles, demands, and proposals are all predetermined, so the actual running of the tournament tends to go very quickly, even in Excel, and the results of the tournament are often known nearly instantly for even a large number of teams.

Pros:

- Tournaments can be run immediately. This allows for teams to discuss a strategy before submitting it, and there are no problems of loss of interest in the results.
• Instead of having one tournament, this allows for the running of several subtournaments in a very short time span. Allowing teams to see the results of each subtournament along with allowing them to change submissions is a nice way to see evolution taking place, and with the computational aspect there is no need to wait a long time to see results.

• Encourages strong mathematical thinking about average payoffs and likelihoods of demands.

Cons:

• Once a decision is made about an array or a function, it is final. There is nothing a team can do about their strategy once it is entered into the tournament.

• No chance of evolution of strategy over the course of a single tournament.

• Teams may have different strategies they’d like to employ when facing one team or another, this version does not allow for that.

For the most part, I find version 2 of the tournament superior to version 1. If you treat the subtournaments as rounds of a larger tournament and allow teams to resubmit strategies from round to round, this is a good approximation to evolution of strategy that can take place during a tournament similar to version 1.

7.5.2 The Vaccination Game Tournament

The Vaccination Game Tournament (VGT) does not involve teams playing against specific teams, rather all teams are competing against one another every round. The VGT was created by Scott Greenhalgh and myself, and is based on the following, admittedly ludicrous, scenario:

• You are an immortal and you will have $X$ children every generation. There is a terrible mutating disease that scours the land before your children are born.

• You get to make a choice between two options, vaccinate or not, before the epidemic hits. You know it shows up without fail.
7.5. EXAMPLE TOURNAMENTS

- If you choose to vaccinate, it is 100% effective, but you lose \( pX \) children, where \( 0 < p < 1 \). The rest are guaranteed to live through that generation. Vaccinations are only good through one generation. After your children survive that first generation, they are safe from the disease to live their lives.

- If you do not vaccinate and do not get the disease, all \( X \) of your children live through that generation. If you get the disease, all of your children die that generation. The disease goes away before the next time the epidemic returns.

- The immortal with the most children after some number of generations is the winner of the tournament.

- Everyone makes their decision about whether or not to vaccinate before the disease runs rampant.

- If you vaccinate, you cannot be infected with the disease, and cannot infect someone else.

- You can be infected by the background initially at some rate \( b \), if you don’t vaccinate.

- Then everyone interacts with everyone, and has a chance to pass on the disease if they were infected to begin with, with transmission rate \( t \).

- At the end of the interactions, those who are not infected get their children, minus \( pX \) if they vaccinated.

- Once everything is sorted, a report is given on a percentage of the population: how many people vaccinated and how many people got sick of that subpopulation. This is to simulate the fact that imperfect information is usually available.

This tournament is interesting, because while the rounds are independent, meaning being sick last round does not affect this round, the distribution of information can affect the strategy of a given player round to round. For example, imagine a player chooses not to vaccinate, and does not get sick in a given round. The report is issued after the round is complete, and she
notices that only a small percentage of people vaccinated, and many did get sick. She may decide that she got very lucky that round and vaccinate the next one. In the same vein, if she vaccinates, and then notices that everyone is vaccinating, she may choose to ride the herd immunity effect in the next round and not vaccinate. The algorithm of how the VGT runs is given below.

Collect player choices about whether to vaccinate or not
Check to see who is infected by the background infection
Have players interact, determine who becomes infected by contact if they weren't already infected by the background
Determine payoffs based on outlined criteria
Give players a chance to make a new choice
Repeat

The algorithm is simple, along with the game, but it can produce some interesting and complex behaviour from the population that plays it.

This tournament drives theoretical biologists up the wall, because often it does not conform to what the mathematics says should happen: just about everyone who does not vaccinate should get sick. The law of mass action, usually applied to chemistry, has been used in the biological literature for many years modeling the spread of pathogens throughout a population. However, there must be sufficient mass and connectivity for the law to work. In the VGT, if there aren’t enough people to satisfy that requirement, the stochasticity of the game makes it very difficult to predict what will actually happen, even when how many people are playing, infection rates, and transmission rates are known. This is another aspect of a recent phenomenon of evolutionary computation called small population effects \[\ldots\]. It captures the idea that small populations are prone to unusual results when placed in competitions, like tournaments or evolution simulations. An open question is the following:

\[
\text{At what size } N \text{ does a population go from exhibiting small population effects to something more akin to the law of mass action?}
\]

A Different Kind of IPD Tournament

The main differences between this tournament and Axelrod’s Tournament and its variations run by other people is that it includes a spatial structure
and it involves human input. There has been some work done on adding a spatial structure to the study of the Iterated Prisoner’s Dilemma. Those investigations were all done from an evolutionary dynamics perspective, rather a tournament perspective. In Axelrod’s original formulation, every strategy played against every other strategy a specified number of times, for a specified number of rounds. In the second tournament he changed the number of rounds by having a certain probability that a game could end on any given round. However, the tournament still maintained that every strategy would face off against every other strategy. A spatial structure defines the connectivity between agents in a game, and restricts which agents can play against one another. There are several representations possible, but a combinatorial graph is a very useful one. As of the time of this writing, there have been no attempts to incorporate structure into an IPD tournament played by agents, no attempts to have human input along with programmed agents, and certainly no combination of those two things. The tournament is conducted as follows:

1. If there are $N$ teams, each team starts with $N$ agents, all of whom will employ the same strategy. Those agents will be randomly placed on an $N$ by $N$ lattice structure, called The Board, such that the opposite ends are considered connected, forming a torus. If $N$ is small, you may want to consider increasing the number of agents allotted per team. My personal recommendation is a minimum $N$ of 25.

2. Each team will select a single location on the Board in which to play, and will play the IPD against their four neighbours to the north, east, west and south, called the von Neumann neighbourhood. The agent may face opponent agents from other teams, or its opponent may be an agent from its own team. The number of rounds should be at least somewhat randomized. I personally use 150 rounds ± 5. It may also be interesting to use the convention created in Axelrod’s second tournament, where the game has a chance of ending after each round.

3. The scores are tallied for each team when the four games are completed for each agent that was selected to play, and then sorted. Have ties broken at random. The top third of teams gets to replace another team’s agent in the neighbourhood in which they just played with a new agent from their own team. So if there are 30 teams in the tournament,
after one round the top 10 teams now have N+1 agents on the board, and the 20 poorer performing teams now have N-1 teams on the board.

4. Repeat 1-2 as many times as desired.

5. A team’s score in the tournament is how many agents they currently have on the board, not their score from round to round. The team with the most agents on the board after a pre-specified time is the winner.

At certain points during the tournament, such as the end of every class week, allow teams to update their strategy. There is also the question of allowing teams to see one another’s strategy during the week. I allow it. Copying successful behaviour has been done, and is currently being done, in just about every human endeavour. It also creates interesting tournament dynamics. As teams begin to converge on the most successful strategy, it behooves teams to stay innovative in order to maintain their edge. It is this aspect of the tournament, more than anything, that forces the students to apply their knowledge in creative ways. From the reports of my students, it is also makes it more fun.

7.6 Some Things to Consider Before Running a Tournament in a Classroom

While teaching a game theory class I had the idea that a tournament is an excellent way to encourage the learning of class material, and to investigate some new tournament properties and dynamics. Running these tournaments resulted in some interesting lessons learned, both about the tournaments themselves and how to run a tournament. Some things to consider before attempting to run a tournament:

- Keep language simple, direct, and do not assume anyone knows what you are trying to tell them. Ever.

- Automate everything that should be automated. Relying on teams or yourself to repeat tasks that would be better and more quickly done by a computer is a mistake that will cause your tournament to stagnate.
7.7. **EXERCISES**

- Decide what program(s) you are going to use to help you run the tournament. I use EXCEL, as it has the built in connectivity I desired and can handle some fairly complex agents in the form of programs. I also use Oracle, which is a database management system that allows for user access from a variety of platforms, and has a central data storage system. This is key for any kind of tournament that involves submission by teams.

- Have your teams submit strategies as either a program (FSA in some cases) or strategies that can be written as a program.

- Encourage your teams to be larger than one person. Individuals participating is fine, but a lot of excellent learning opportunities occur when trying to make a strategy clear to another teammate, or deciding what your agent’s next move should be.

- Make available as much information of how the tournament is progressing as possible. This includes standings, game state, and possibly strategies. This can be done using a variety of free programs, or through a class website.

- Do not run a tournament over the class time of an entire week. It creates chaos in the course and does not allow the students enough time to truly grapple with the material.

- With computational power at its current level and cloud data storage so cheaply available, let a tournament run over the course of a semester. Dedicate a few minutes of class time every class, or have a designated time a few days a week, to further the tournament’s progress. The students will have time to think about their strategies, and develop better understanding of what is going on.

- Maybe most importantly, have a prize for those who do well in the tournament. Often, competitive drives in teams are crushed by laziness or other factors, so be sure to incentivize them to do their best.

---

**7.7 Exercises**

1. Provide a lattice that satisfies the conditions of a BTD(5). Show that a BTD(4) is impossible, and generalize this result to all BTD(n) with
2. The number of pairs \((n,c)\) that satisfy the conditions satisfying a CBTD\((n,c)\) for \(n\) teams have been explored in sequence A277692, in the Online Encyclopedia of Integer Sequences (https://oeis.org/A277692).

(a) Investigate the distribution of this sequence by finding its mean, mode, and median for the first 1000 terms.
(b) How many values of \(n\) only have 1 court that allows a CBTD? What do those values of \(n\) have in common?
(c) Using software provided or creating your own, plot the number of teams versus the terms of the sequence. What do you notice?
(d) Perform some other analyses you think appropriate. Can you get an idea of why this sequence behaves as it does?

3. Create a tournament schedule for 22 teams using the cyclic permutation fractal tournament design. How many games does it take to complete? How many rounds does it take to complete? Which values of \(c\) satisfy a CBTD\((22,c)\)?

4. Create a payoff function for a participant in the VGT. This function should include:

- the choices: vaccinate or do not vaccinate
- the number of children, \(X\)
- the background infection rate, \(b\)
- the vaccination rate, \(v\)
- the transmission rate, \(t\)
- and the number of people involved in the game, \(N\)

5. Consider yourself as a player in the VGT. Fix the values of \(X\), \(b\), \(v\), \(t\). For what values of \(N\) do you choose to vaccinate? When is the risk acceptable to not vaccinate?

6. The VGT assumes complete connectivity throughout the population. Consider yourself as a player in the tournament. How does your strategy change if you know that there are small world network effects in play?
7. Do some research on: “At what size $N$ does a population go from exhibiting small population effects to something more akin to the law of mass action?” Is there anything in the literature that gives a suggestion?