2019

# Selecting pairs of non-transitive dice

Baoyue Bi
*Iowa State University*

**Selecting pairs of non-transitive dice**

by

**Baoyue Bi**

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Mathematics

Program of Study Committee:
Steve Butler, Major Professor
James Rossmanith
Michael Young

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

# DEDICATION

I would like to dedicate this thesis to my husband Zehua, without whose support I would not have been able to complete this work so smoothly. I would also like to thank my family and friends for their loving guidance and financial assistance during the writing of this work.

# TABLE OF CONTENTS

**Page**

# LIST OF TABLES

**Page**

# LIST OF FIGURES

**Page**

# ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Steve Butler for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. James Rossmanith and Michael Young. I would additionally like to thank Dr. Sung-Yell Song for his guidance throughout the initial stages of my graduate career and Mrs. Erickson for her detailed assistance and valuable advice.

# ABSTRACT

Non-transitive dice are sets of dice which break transitivity, namely it is possible for $A$ to have an advantage over $B$, for $B$ to have an advantage over $C$, and for $C$ to have an advantage over $A$. These are well-known and studied for the case of selecting a single die (possibly rolling it multiple times).

We introduce the problem of two players selecting pairs of dice out of a common pool, where they alternate taking turns *and* the second person selecting dice has an advantage (note there are multiple ways that the dice can be picked up; we seek dice which give an advantage to the second player in all scenarios). We exhibit sets of dice with these properties, found by computational search, and discuss some theoretical aspects by use of Kneser graphs.

# CHAPTER 1. INTRODUCTION

This is the opening paragraph to my thesis which explains in general terms the concepts and hypothesis which will be used in my thesis.

With more general information given here than really necessary.

Consider the following game for two players using the set of three dice shown in Figure 1.1. Player one picks a die first and then player two picks a die. Both players roll and the one who gets a higher value wins.

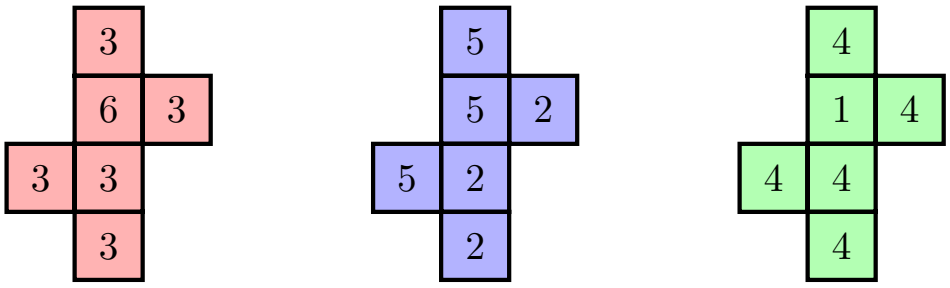


Figure 1.1 A set of three non-transitive dice.

The game is not as fair as it looks. Player two can always select a die that has a *slightly* higher probability of winning. Whichever die player one picks, player two can pick the one to its "left" to get a better result.

This is an example of non-transitive dice, which was popularized by Martin Gardner (1) in 1970, then further explored by Edward J. Barbeau (2) and Richard P. Savage Jr (3). In 2010, James Grime (4) found a set of five non-transitive dice with two non-transitive chains in it termed the Grime dice. The dice are colored by Blue, Magenta, Olive, Red and Yellow, where the first chain is ordered alphabetically, and the second chain is ordered by word-length. see Figure 1.2. In recent years, there is more literature about balanced non-transitive dice (5) and dice tournaments (6; 7).

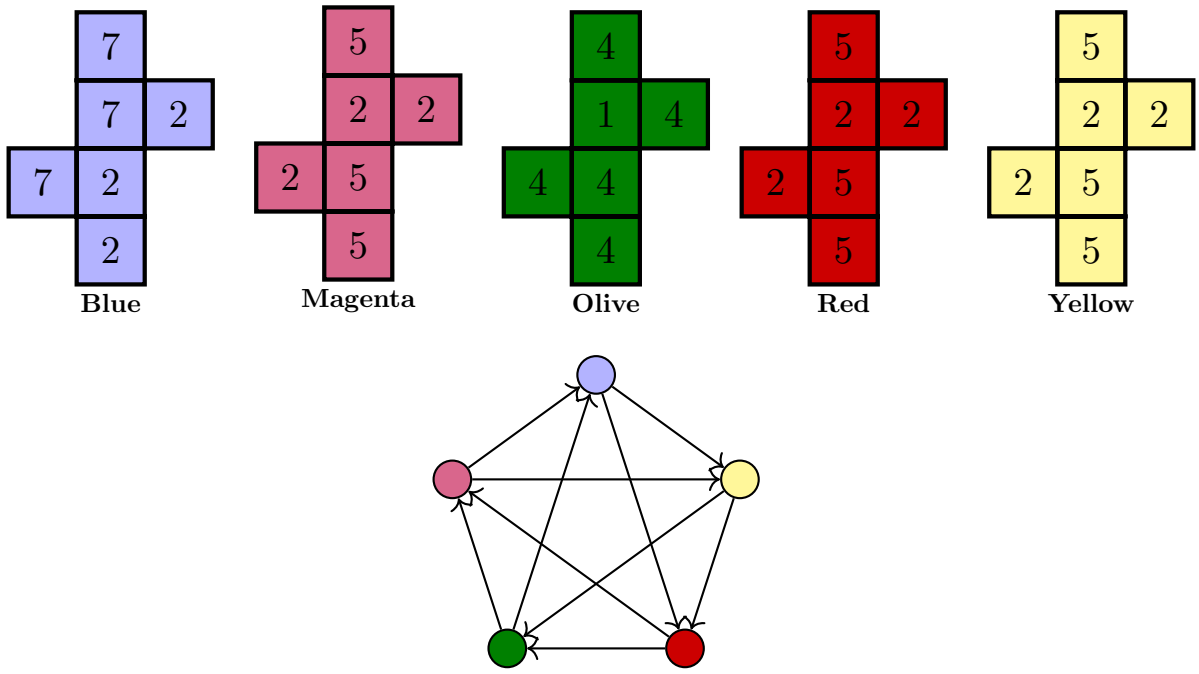


Figure 1.2 Grime dice tournament with arrow points to the better die.

We say that die $A$ beats $B$ if $A$ is more likely to have a better score than $B$ (in the case of ties we reroll), and denote this by $A > B$. In general, if $A > B$ and $B > C$ we would expect for $A > C$, this is a *transitive* relationship. In contrast, for non-transitive dice, one would have $A > B$,$B > C$ and $C > A$. As the number of dice in consideration grows, so to do the number of possible relationships between pairs of dice.

## 1.1 Problem Setup

Here we present a variant problem based on five dice. Consider a game with five dice and two players, each player picks two dice. Both players roll their dice and add up the values to get a result, the one who gets a higher total wins (keep rerolling until there is not a tie). Based on this game, we want to find the sets of dice and corresponding strategies where the second player responds to the first player to gain a slight advantage in winning.

To start, let's clarify two things. First, how do the two players select the dice? Second, how do we calculate the probabilities? To the first question, we actually have three variations.

i) Player one picks a die, player two picks a die and we repeat. Throughout we will use $P1$ to denote player one and $P2$ to denote player two, then the procedure is $P1$-$P2$-$P1$-$P2$.

ii) Player one picks a die, player two then picks two dice and player one picks one die, in short, $P1$-$P2$-$P2$-$P1$.

iii) Player one picks two dice and player two picks two dice, denote $P1$-$P1$-$P2$-$P2$.

Now let's move on to the second question. To see the winning probability for second player, create a table with 37 columns and 37 rows. For a single player, the total number of summed roll values are 36. Leaving the first cell $(1, 1)$ blank, list all possible values for $P1$ along the first row [Cells $(1, 2)$ - $(1, 37)$] and the values for $P2$ on first column [Cells $(2, 1)$ - $(37, 1)$]. We now fill the *inner table* from $(2, 2)$ to $(37, 37)$. If cell $(1, j)$ is less than cell $(i, 1)$, write 1 on cell $(i, j)$, if $(1, j)$ equal to $(i, 1)$, write 0 on $(i, j)$, otherwise leave the cell blank. Tally up the 1's and 0's on the inner table and call the result $w$ and $t$ respectively. Thus the winning rate for player two can be expressed as $\frac{w}{36^2 - t}$, and the winning rate for player one is $\frac{36^2 - w - t}{36^2 - t}$. And we say a player has winning advantage if he/she has a higher winning probability. The division by $36^2 - t$ is done to correct for any ties.

Now we are ready to dig into the problem. The first thing is to find a set of good dice, and for this we develop a computer program to help us search. We then analyze the dice which are produced, which we will discuss the details in the next section.

# CHAPTER 2. EXISTENCE AND COMPUTATIONAL APPROACH

Let's recall the problem: Is it possible for the second player to win a game on all three game variations of selecting dice? To find out the answer (find a set of dice that works), we did the following:

First of all, we constructed a program which randomly generates dice with values ranging from 0 to 9. In order to speed up the process, we add limitations such that each die can have at most two distinct values. On each run of the program we create a potential set of 5 dice. We then take the dice and check to see if they have a non-transitive relationship, i.e. $A > B$, $B > C$, $C > D$, $D > E$, $E > A$; discard any set of dice which are not non-transitive. Finally, for each set of non-transitive dice, calculate the winning probabilities for each variation of the game.

To compute the probabilities in each variation, we used a decision tree algorithm. No matter which order the players select the dice, there are a total of four layers in the tree. It is only the matter of thinking who get to choose in which layer.

To determine who wins in the decision tree, first assign $P2$'s probability as a value to each node in the bottom layer of the tree and work up one layer at a time. On a given layer, if it is $P1$'s turn, pick the lowest winning probability of all the children, and if it is $P2$'s turn pick the highest probability.

A decision tree is an organized way to run through combinations and check possibilities. Additionally, we only need to make a small modification for the code to try all three variations of selecting dice.

We ran the program several million times and below are some examples of the "good" sets of dice we produced.

| | | |
|---|---|---|
| $A*: 3,3,3,3,9,9$ | $A: 4,4,4,4,9,9$ | $A: 2,5,5,5,5,5$ |
| $B*: 1,1,1,8,8,8$ | $B: 3,3,3,8,8,8$ | $B: 3,3,3,3,9,9$ |
| $C*: 1,1,1,7,7,7$ | $C: 2,2,2,8,8,8$ | $C: 1,1,1,9,9,9$ |
| $D*: 1,5,5,5,5,5$ | $D: 1,6,6,6,6,6$ | $D: 1,1,7,7,7,7$ |
| $E*: 1,4,4,4,4,4$ | $E: 5,5,5,5,5,8$ | $E: 3,3,6,6,6,6$ |

We will use the starred set of dice on the left as an example to explore the theoretical facts in the next section. The wining ratio between $P1$ and $P2$ ($P1/P2$), assuming optimal play for the players, is 0.491/0.509 for the variations $P1$-$P1$-$P2$-$P2$, $P1$-$P2$-$P1$-$P2$, and 0.486/0.514 for the variation $P1$-$P2$-$P2$-$P1$.

# CHAPTER 3. KNESER GRAPH (PETERSEN GRAPH)

## 3.1 Introduction

In this section, we are going to relate the sets of dice with the Petersen graph and connect the $P2$ winning strategy with graph orientations.

**Definition 3.1.1.** The Kneser graph $KG_{n,k}$ is the graph whose vertices correspond to the $k$-element subsets of a set of $n$ elements, and where two vertices are adjacent if and only if the two corresponding sets are disjoint.

The most well-known Kneser graph is the Petersen graph, $KG_{5,2}$, an undirected graph with 10 vertices and 15 edges. See Figure 3.1 for an example of an oriented Petersen graph.

For the problem of selecting dice, we want to look at an oriented copy of the Petersen graph. We can treat each of the ten vertices of the Petersen graph as a pair of two dice, connect two vertices if and only if the two sets are disjoint. Moreover, we orient the edge (drawn with an arrow) to point to the one with higher winning probability. Figure 3.1 is an example of an oriented Petersen graph using the good set of dice from the end of Section 2.

## 3.2 The $P1$-$P1$-$P2$-$P2$ Variation

For $P1$-$P1$-$P2$-$P2$ case, $P1$ has the advantage of picking two dice ahead. To win, $P2$ must choose a pair of dice from the leftover three with a higher winning rate. Consider the oriented Petersen graph for our set of dice. The two dice that $P1$ selected correspond to some vertex in the graph; $P2$ needs to pick some other vertex that is available *and* has a better probability.

In our case, we have for every vertex, there is a "better" vertex it is connected with, where "better" means a pair of dice that can beat the dice in the starting vertex. To summarise, we have the following property:

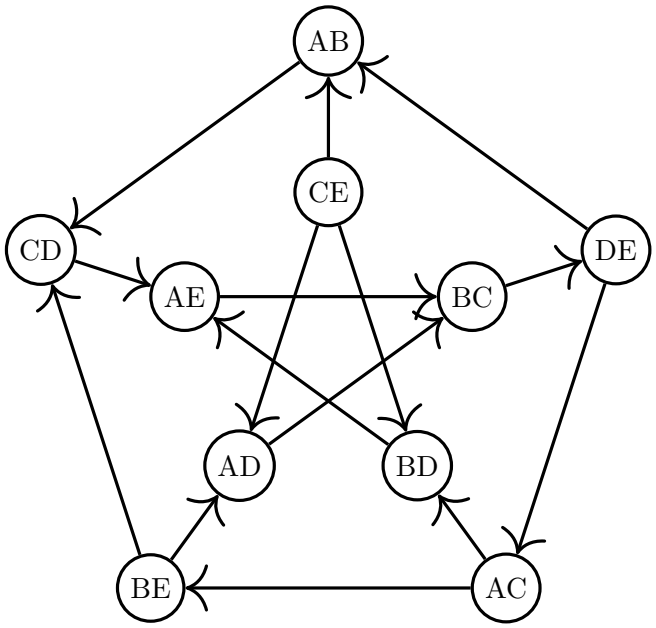


Figure 3.1 Oriented Petersen Graph for the dice from the end of Section 2

**Property 3.2.1.** At every vertex in the Petersen graph, there is always an arc directed out from that vertex. The set of dice in the vertex on the other end of the arc is a "better set".

### 3.3 The $P1$-$P2$-$P2$-$P1$ Variation

We now consider the $P1$-$P2$-$P2$-$P1$ variation, player one picks a die first, then player two picks two dice in a row, player one then responds with his/her last choice.

As an example to help our analysis, let's assume the first player picks die $A$. Then the possible dice combinations for $P1$ at the end are $AB$, $AC$, $AD$, and $AE$. After the second turn, $P1$ still has two choices to play against $P2$. Thus, $P2$ must pick the pair which can beat the two choices left of $P1$ simultaneously. In terms of the graph, go through the neighbors of $AB$, $AC$, $AD$, and $AE$; and find a vertex $v$ so that all edges between $\{AB, AC, AD, AE\}$ and $v$ are oriented towards $v$. As an example in Figure 3.1, $P2$ should pick the pair of dice $BC$ if $P1$ pick $A$ at the first round, since it can beat $AD$ and $AE$ at the same time.

We carry out the full analysis for all choices of $P1$ in Table 3.1. We mark in red any choice for $P2$ which can guarantee a win regardless of $P1$'s last choice.

Table 3.1 Decision table for $P1$-$P2$-$P2$-$P1$ Variation

| First pick | $P1$ possible outcomes | Better dice | $P2$ Winning Choice |
|---|---|---|---|
| A | AB | CD | BC |
| | AC | BD | |
| | AD | BC | |
| | AE | BC | |
| B | BA | CD | CD |
| | BC | DE | |
| | BD | AE | |
| | BE | AD,CD | |
| C | CA | BD | BD |
| | CB | DE | |
| | CD | AE | |
| | CE | AB,AD,BD | |
| D | DA | BC | AE |
| | DB | AE | |
| | DC | AE | |
| | DE | AB,AC | |
| E | EA | BC | AB,AD |
| | EB | AD,CD | |
| | EC | AB,AD,BD | |
| | ED | AB,AC | |

### 3.4 The $P1$-$P2$-$P1$-$P2$ variation

The $P1$-$P2$-$P1$-$P2$ case, does not have a clear connection with the graph. To see the winning strategy, it is easier to look at the decision tree. There are total of four layers in the tree. $P1$ has five choices at the first pick, $P2$ can then choose one from the four left to respond and so on and so forth. Figure 3.2 is a complete decision tree for the set of dice from the end of Section 2. Winning choices for $P2$ have been marked. As an example, suppose $P1$ picks $A$ at the first turn, $P2$ can pick either $B$ or $C$ for a guaranteed win. From the last layer of the tree, if $P2$ picks $B$ or $C$ on the second turn, no matter which die $P1$ choose on the third round, there is another die available for

$P2$ which results in a better set. In general, if $P2$ has a better choice for all five dice, $P2$ will end up winning.

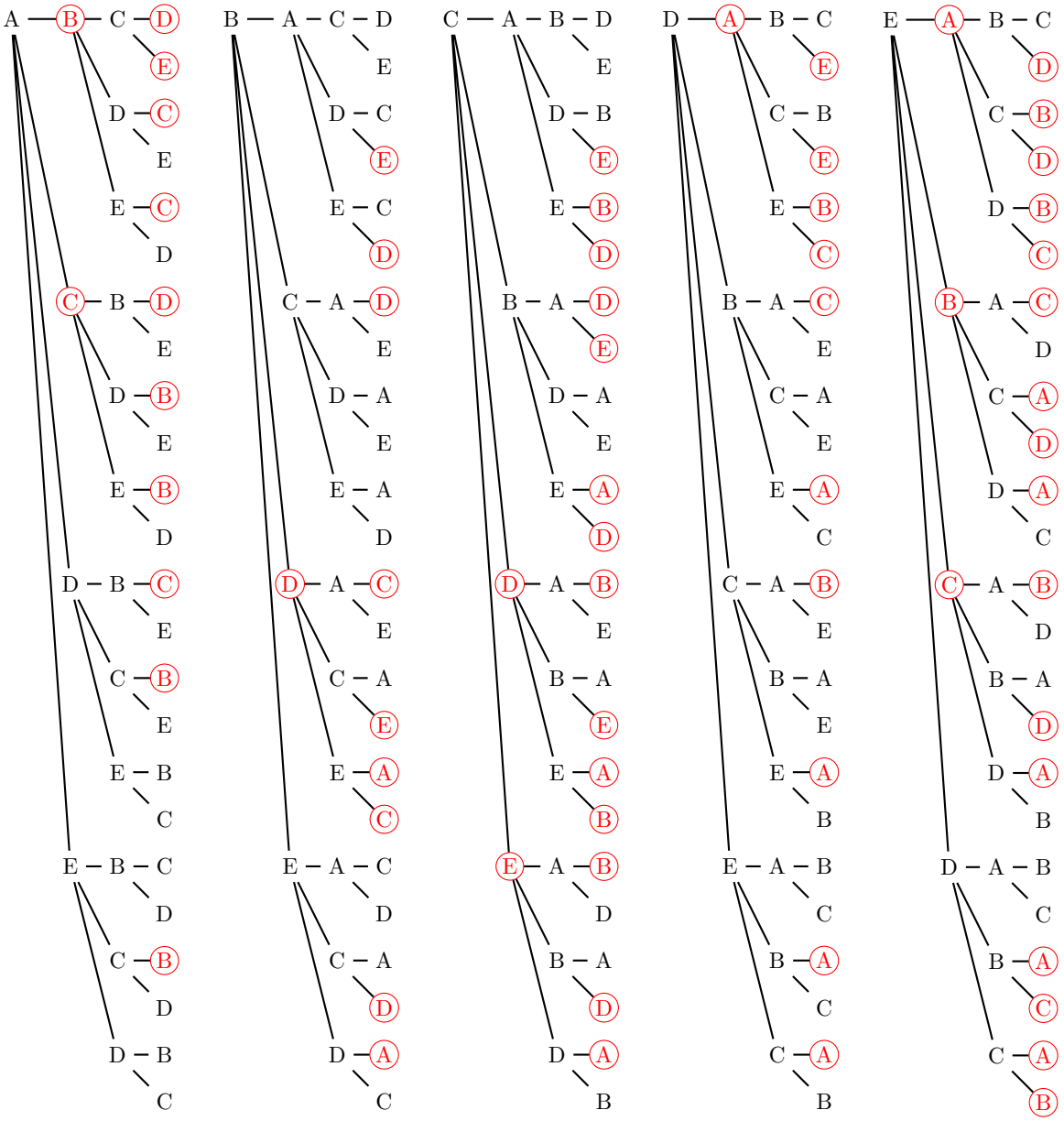


Figure 3.2 Decision tree for the $P1$-$P2$-$P1$-$P2$ case. Red circles indicate a winning choice for player two.

# CHAPTER 4. CONCLUSIONS

There are still several ways you can dig into this type of game. One question to ask is: For a given orientation of the Petersen graph (more generally a Kneser graph), can we find a set of die which will produce that orientation. It seems possible that for any orientation we can find a set of dice. (On a side note orientations for tournaments arising from dice have been studied; these are Kneser graphs with vertices being subsets of size 1.)

The winning strategy for $P2$ seems to exist when there is an odd number of dice. This always allows at least one die of freedom to help $P2$ in selecting better die. What would happen if there is an even number of dice and all dice had to be selected? In particular, at the end of the game, each player has the same number of dice and no die left. Is there any case where player two can still win? This situation seems really unfavourable for $P2$. It would be surprising if one can find a set of dice that works for the even number case. (Of course non-transitive dice are already surprising to begin with; so it might be possible!)

Another interesting variant to this problem can be attained by adding more players to the game. According to James Grime, it is possible to have a three player game with seven dice and each player selecting one die each, where the third player to pick can beat both player one and player two simultaneously (4). Ideally, we want the last player to obtain a winning probability that exceeds 50% (even if just barely) for the game with any number of players if we can find the "right type" of dice.

The results state in this paper reveals a great potential of non-transitive dice problem, and there are many possible directions one can think of. The difficulty and variations of the problem can be controlled by twisting any parameters in the game. A further approach to the five dice game would be to find the "best" set of dice that works for all three variations. If we can find the right

orientation of the "best" set of dice, there will be less computational search contribute by a good space cut down.

## REFERENCES


[1] M. Gardner. The paradox of the nontransitive dice and the elusive principle of indifference. *Scientific American* **223** (1970), 110–114.

[2] Edward J. Barbeau. *Mathematical Fallacies, Flaws, and Flimflam*. The Mathematical Association of America, 2000.

[3] Richard P. Savage Jr. The paradox of nontransitive dice. *The American Mathematical Monthly* **101** (1994), 429–436.

[4] J. Grime, Non-transitive Dice, 2010, https://singingbanana.com/dice/article.htm.

[5] A. Schaefer and J. Schweig. Balanced Non-Transitive Dice. *College Math. J* **48** (2017), no.1, 10-16.

[6] A. Schaefer, Balanced Non-Transitive Dice II: Tournaments, 2016. https://arxiv.org/pdf/1610.08595.pdf.

[7] N. Alon, G. Brightwell, H. A. Kierstead, A. V. Kostochka, and P. Winkler. Balanced Non-Transitive Dice. Dominating Sets in $k$-Majority Tournaments. *Journal of Combinatorial Theory, Series B*, **96** (2006), 374-387.

## APPENDIX. PROGRAM FOR SELECTING DICE

```
from random import sample, randint
class Node:
    def __init__(self, player):
        self.win = 0
        self.player = player
        self.nexts = []
        self.dicesA = []
        self.dicesB = []


def generateDice():
    '''
    Generate dices
    Current rule:
        numbers in [1, 9]
        Only 2 number can be on one dice
    '''
    dices = []
    for i in range(9):
        for j in range(i+1,9):
            for x in range(7):
                dice = [i+1]*x
                dice.extend([j+1]*(6-x))
                if dice not in dices:
```

```
                    dices.append(dice)
    return dices


def pick(dices, n):
    '''
    Randomly pick n dice from all dices
    '''
    return sample(dices, n)


def recCheck(dices, stack):
    '''
    Recursively found nontransitive relationship
    '''
    if len(dices) <= 0:
        return winrate(stack[-1], stack[0]) > winrate(stack[0], stack[-1])
    for diceB in dices:
        if winrate(stack[-1], diceB) > winrate(diceB, stack[-1]):
            dd = dices.copy()
            dd.remove(diceB)
            stack.append(diceB)
            if recCheck(dd,stack):
                return True
            stack.pop()
    return False


def isNonTransitive(dices):
    '''
```

```
    Given a group of n dice, check the transitive relationship
    '''
    stack = []
    for dice in dices:
        dd = dices.copy()
        dd.remove(dice)
        stack.append(dice)
        if recCheck(dd,stack):
            return stack
        stack.pop()
    return None


def generateNodes(dices, node, mode, ind):
    '''
    Recursively generate the tree stucture to simulate each draw action.
    Rule:
    player 1 draw first,
    each player draws 1 die each turn,
    '''
    # print(len(node.dicesA) + len(node.dicesB));
    if len(node.dicesA) + len(node.dicesB) == 4:
        return
    player = mode[ind]
    for dice in dices:
        if dice not in node.dicesA and dice not in node.dicesB:
            nNode = Node(player)
            nNode.dicesA = node.dicesA[:]
```

```
            nNode.dicesB = node.dicesB[:]
            if node.player == 1:
                nNode.dicesA.append(dice)
            else:
                nNode.dicesB.append(dice)
            node.nexts.append(nNode)
            generateNodes(dices, nNode, mode, ind+1)

def checkWin(node):
    '''
    Given the root node of the tree structure,
    check who is going to win by follwing decision:
    player2 choose the greatest player2 win rate of all its choices.
    player1 choose the lowest player2 win rate of all its choices.
    '''
    if len(node.dicesA) + len(node.dicesB) == 4:
        winA = winrate2(node.dicesA, node.dicesB)
        winB = winrate2(node.dicesB, node.dicesA)
        if winA < winB:
            node.win = 2
            return (winA, winB, 2)
        else:
            node.win = 1
            return (winA, winB,1)
    minrate = (0.000001, 1000000.0, 0)
    maxrate = (10000000.0, 0.000001, 0)
    for child in node.nexts:
```

```
        res = checkWin(child)
        if calrate(res) > calrate(maxrate):
            maxrate = res
        if calrate(res) < calrate(minrate):
            minrate = res
    return maxrate if node.player == 2 else minrate


def calrate(rate):
    if rate[0] == 0:
        return float('inf')
    return rate[1] / rate[0]


def goodSet(dices, mode):
    '''
    Check in the given n dices, if player has higher chance to win.
    '''
    root = Node(mode[0])
    generateNodes(dices, root, mode, 1)
    return checkWin(root)


def winrate(diceA, diceB):
    '''
    Calculate win rate between sets of 2 dice
    '''
    count = 0
    for i in range(6):
        for j in range(6):
```

```
            if diceA[i] > diceB[j]:
                count += 1
    return count/36.0


def winrate2(dicesA, dicesB):
    '''
    Calculate win rate between 4 dices
    '''
    count = 0
    for i in range(36):
        for j in range(36):
            if dicesA[0][i//6]+dicesA[1][i%6]>dicesB[0][j//6]+dicesB[1][j % 6]:
                count += 1
    return count/(36.0**2)


def winrate3(dicesA, dicesB):
    '''
    Calculate win rate between 6 dices
    '''
    count = 0
    for i in range(216):
        for j in range(216):
            dA = dicesA[0][i%6]+dicesA[1][i//6%6]+dicesA[2][i//36]
            dB = dicesB[0][j%6]+dicesB[1][j//6%6]+dicesB[2][j//36]
            if dA > dB:
                count += 1
    return count/(216.0**2)
```

```
def main():
    dices = generateDice()
    # print(dices)
    # print(len(dices))
    picked = pick(dices, 5)
    #print(picked)
    ordered = isNonTransitive(picked)
    j = 0
    modes = [[1,2,1,2,3], [1,2,2,1,3], [1,1,2,2,3]]
    with open("result.txt", "w") as f:
        while j < 50000:
            picked = pick(dices, 5)
            ordered = isNonTransitive(picked)
            if ordered:
                ress = []
                con = True
                for i in range(len(modes)):
                    res = goodSet(ordered, modes[i])
                    if res[2] == 2:
                        ress.append(res)
                    else:
                        con = False
                        break
                if con:
                    f.write('p2:'+ str(ordered)+ str(ress)+"\n")
                    print('p2:', ordered, ress)
```

```
                    # break
        print("================")
        print(ordered)

def main2():
    dices=[[5,5,5,5,5,9],[3,3,3,9,9,9],[2,8,8,8,8,8],[5,6,6,6,6,6],[4,6,6,6,6,6]]
    print(goodSet(dices, [1,1,2,2,3]))

if __name__ == '__main__':
    main()
```