CARD GAME STRATEGY ANALYSIS
tarneeb

Yuanyuan Jin
Bassem Kaddour
Matthias Lee

THE UNIVERSITY OF BRITISH COLUMBIA
The Department of Mathematics

December 2016

ABSTRACT

The purpose of this paper is to provide a preliminary report of our understanding and analysis of the strategies for playing a Middle Eastern card game called Tarneeb. The objective was to find a strategy for playing the cards that maximized the number of hands won each game. The stochastic nature of this card game required a simplified version of the game to be formulated and analyzed in order to examine the optimal playing strategies. The main results indicated that the game is not predetermined, and so it was appropriate to attempt to develop a strategy. The strategy found showed that a player should be 'greedy', and win as many hands as possible early in the game. This can allow the player to influence how the game unfolds, maximizing the number of hands won in total. The data suggests that further investigation of optimal strategies for this game is merited.

This project and report is produced with the objective to apply mathematical knowledge from the course taught by Dr. Richard Anstee: -Mathematical Modeling of Discrete Optimization Problems-, and from other undergraduate courses, on a problem that is dynamic, interesting, and challenging to us. We thank Dr. Richard Anstee for all of his help, guidance, and emotional support while embarking on this project.

TABLE OF CONTENTS

1: Introduction 1

2: Tarneeb

2.1 Players 1

2.2 Cards & Card Dealing 1

2.3 Bidding Stage 1

2.4 Bidding Strategies 2

2.5 Rules of Play 3

2.6 Playing Strategies 3

2.7 Payoff & Game Objective 3

3: Simplified Tarneeb

3.1 Reasons for Simplification 4

3.2 Game Formulation 4

3.3 Playing Strategies 4

3.4 Payoff & Game Objective 5

4: Computational Logic

4.1 Generating the Payoff Matrix 6

4.2 Strategy 7

5: Results

5.1 Non-Determined Game 8-9

5.2 Analysis of Optimal Strategies 10

5.3 Limitations 11

5.4 Extrapolations to the Full Game 11

6: Conclusion 12

Appendix A: Code 13-18

Appendix B: References 19

1: Introduction

Tarneeb, which literally refers to “trump” in Arabic, is a card game that is widespread throughout various countries in the Middle East. Its origins trace back to Bilad al-Sham in the 7th century, which is the region that encompasses modern day Jordan, Lebanon, Palestine, and most of Syria. The game, however, was not popularized until the early 18th century.

2: Tarneeb

2.1 Players

The game is played among four players divided into two teams. Each player has an opponent beside them, and an opponent across from them. The teams are assigned randomly or chosen by the players themselves.

2.2 Cards & Card Dealing

The game is played with a standard deck of 52 cards, with 13 cards in each of the following four suits: Spades, Hearts, Clubs, and Diamonds. The 13 cards in each of the four suits rank in the following order from least to greatest: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A.

Several games of Tarneeb are generally played in succession, with 13 rounds for each game. Cards are shuffled before the first game and after every game. The first dealer can be randomly chosen, and the next dealer is the person on the right of the dealer of the previous game. The cards are dealt clockwise after shuffling, one card at a time, starting with the player to the left of the dealer. These cards are dealt face down, such that no player has any information on the cards dealt to the other three players. The players are dealt 13 cards each for every game.

2.3 Bidding Stage

The bidding stage proceeds when the deck is completely distributed. The bid starts with the player to the right of the dealer. The minimum bid is 7 and the maximum bid is 13. The first bidder can bid any number in this range, while subsequent bids must always be higher than the previous bid. Each player will bid the number of rounds they anticipate their respective team can win. It is possible for any player to pass their turn of bidding, however, once a player decides to renounce their right to bid, they cannot bid again for this game. If all four players pass on their first turn to bid, then the cards will be collected, shuffled, and dealt again. Otherwise, bidding continues counter-clockwise until all but one player has passed or until one player bids 13. The player that wins the bidding (by having the highest bid) chooses the Tarneeb (trump suit of the game) and leads the first round of the game.

2.4 Bidding Strategies

The bidding strategies available to the first bidder are: {7, 8, 9, 10, 11, 12, 13, Pass}. Thus, the number of possible bidding strategies available to the first bidder at the beginning of each game is: 8C1 = 8.

Let the bid chosen by the first bidder be i ϵ {7, 8, 9, 10, 11, 12, 13, Pass}, then the bidding strategies available to the second bidder are: j ϵ {i+1, i+2, …, 13, Pass}. The number of possible bidding strategies for the second bidder is {i, i-2, i-4, i-6, i-8, i-10,1,8}.

Let the bid chosen by the first bidder be i ϵ {7, 8, 9, 10, 11, 12, 13, Pass} and the bid by second bidder be j ϵ {i+1, i+2, …, 13, Pass}, then the bidding strategies available to the third bidder are: k ϵ {j+1, j+2, …, 13, Pass}.

If i+1=j, then the number of possible bidding strategies for player three is:
{j-2, j-4, j-6, j-8, j-10,1}
If i+2=j, then the number of possible bidding strategies for player three is:
{j-4, j-6, j-8, j-10,1}
If i+3=j, then the number of possible bidding strategies for player three is:
{j-6, j-8, j-10,1}
If i+4=j, then the number of possible bidding strategies for player three is:
{j-8, j-10,1}
If i+5=j, then the number of possible bidding strategies for player three is:
{j-10,1}
If i+6=j, then the number of possible bidding strategies for player three is:
{1}
If i=j={Pass}, then the number of possible bidding strategies for player three is:
{8}
If (i ϵ {7, 8, 9, 10, 11, 12, 13}, j = {Pass}),
Then the number of possible bidding strategies for player three is:
{i, i-2, i-4, i-6, i-8, i-10,1}
If (i = {Pass}, j ϵ {7, 8, 9, 10, 11, 12, 13}),
Then the number of possible bidding strategies for player three is:
{j, j-2, j-4, j-6, j-8, j-10,1}

The above bidding strategies are listed for only one bidding round, which could branch into various bidding outcomes. As there may possibly be more branches in the next bidding rounds, the bidding stage would convolute the game and playing strategies. Thus, the next rounds of bidding strategies will not be presented in this analysis, and will be eliminated in the simplified version of the game.

2.5 Rules of Play

The player that wins the bid starts the game and subsequently decides the initial round suit (the first card played each round determines the round suit). Whoever wins the round plays the first card of the next round. Each player plays only one card per round, moving clockwise. Every player has to follow the round suit if possible, but if a player has no cards of matching suit to the round suit, they may play a Tarneeb or discard a card of any other suit. If no Tarneeb has been played, the winner of the round is the team that played the highest card of the round suit. If Tarneeb has been played, the winner of the round is the team that played the highest Tarneeb. Each hand won is referred to as a trick, and counts for one point in the game. The team that won the bidding round wins the game when they have collected a number of tricks equal to their bid - if they are unable to do so then the other team wins the game.

2.6 Playing Strategies

The number of possible card-playing strategies available to each player in each game is: 13!, an equivalent of 6,227,020,800 ways for each player to play out their cards. The total number of possible outcomes of the card-playing stage = $(13!)^4$ ways that cards can be played in 13 rounds, although given the game's rules not all of these outcomes will be valid/possible games.

2.7 Payoff & Game Objective

There are variations on scoring when multiple games are played in succession, and games may continue for an arbitrary amount of time or until a total score threshold is broken by one team. In this analysis, higher level scoring of multiple games will not be considered. The goal is to develop a strategy that maximizes the number of tricks won each round, which will ultimately lead to a strategy that allows a team to maximize the number of games won (contingent on sensible bidding strategies).

3: Simplified Tarneeb

3.1 Reasons for Simplification

Tarneeb is a game with many complexities that must be considered before a strategy can be developed. As indicated above, there is a vast number of possibilities of hand arrangements and card playing strategies for each player, contingent upon the way the cards are dealt. Each player has no information about the distribution of cards other than their own, and so there is also a prominent stochastic element to how any strategy will play out. The vast pool of game states and stochastic elements involved amount to a quantity of information that could not be processed given the current constraints on time, game theory knowledge, computational power, and computational (programming) skill level. Due to these factors, a simplified version of the game was formulated with the intention of extrapolating patterns found in strategies for the simplified game to the full game.

3.2 Game Formulation

In formulating a simplified version of the game, several decisions were made in order to remove some of the game's stochastic elements. The total size of the game was also reduced to help meet the computational restrictions.

This simplified version of the game is played with only two players on opposing teams, instead of four players playing in teams of two (i.e. there are no partners).

Similarly, the number of suits was reduced to two. The bidding stage was eliminated, and one of the two suits was chosen to be the pre-determined trump suit. The number of cards chosen from each of the two suits was also reduced.

Besides these changes, the simplified game will play in the same manner as the full game. The solution is formulated to find the optimal strategy for the player that goes first (which player goes first can be determined randomly).

3.3 Playing Strategies

The number of possible card-playing strategies available to each player for a game with 2n total cards is n!. For example, in a five card game this is equivalent to 5! = 120 ways for the player to play out their cards. Therefore, the number of possible outcomes of this simplified game $\leq (n!)^2$. The number of possible games will only be $(n!)^2$ if every combination of the row player's (first player) cards is valid against every combination of the column player's (second player) cards. This is unlikely due to the game rules rendering many of these combinations impossible.

3.4 Payoff & Game Objective

The objective of the simplified game is to maximize the number of tricks a player wins in one game with a hand of n randomly dealt cards from the 2n-card deck. The total possible number of tricks is n.

The payoff matrix was created by comparing two games, and adding one point for each trick won. Illegal games were screened for and excluded from the payoff matrix.

The game was formulated against both an opponent that makes random (but legal) card plays, and against a competent opponent that will always make the best move.

4: Computational Logic

The computations for this solution were performed using the Python programming language, version 2.7. The code can be found in Appendix A. The following steps outline how the code was formulated to produce an optimal strategy.

4.1 Generating the Payoff Matrix

The solution to the game was determined by formulating a payoff matrix comparing each possible way for the row player to play out their cards with each possible way for the column player can play out their cards. Each row of the payoff matrix corresponds to one particular permutation of the row player's cards (for every possible permutation), and each column of the payoff matrix corresponds to one particular permutation of the column player's cards (for every possible permutation). Each cell of the payoff matrix then gives the value of the game where each player plays their cards in the order specified by the given row and column. Here, the value of the game is the number of tricks that the row player will win.

The value of each cell in the payoff matrix was obtained by comparing each card in the row player's and column player's hands, in sequence. If the row player's card beat the column player's card, a value of 1 was added to the payoff. If the row player's card lost to the column player's card, the payoff value was unchanged.

Many of the games represented by cells in the payoff matrix are impossible outcomes, because of illegal plays that defy the rules. An example of an illegal play would be for a player to put down a card that did not have the same suit as the round suit, even though a card belonging to the round suit could still be found in that player's hand. For this reason, these games were assigned a large negative value, ensuring that they would not be considered when looking for the optimal strategy.

4.2 Strategy

Once the payoff matrix was generated, two methods were used to determine an effective strategy: one for an opponent playing randomly (but with legal plays), and one for an opponent that plays competently, making the best move possible.

A strategy against a randomly playing opponent was determined by examining the expected payoff of each row. The expected payoff was determined by summing the positive payoff values in each row of the payoff matrix, and dividing that number by the number of valid games in that row. Then, each row that gave the maximum payoff was considered to be a valid strategy for an optimal game.

A strategy against a competent opponent was determined by examining the minimum positive payoff value for each row. This is because a competent opponent was assumed to always make the best move, and so for any given row chosen by the row player the competent opponent will minimize how much the row player is able to win.

The discussed strategies for each opponent were used to create a strategy vector that gave each optimal strategy an equal weighting of how often it should be played.

5: Results and Discussion

Note: For the remainder of this report, card values will be reported as vectors [x,y]: x represents the card value with 1 as the lowest card, y=0 indicates the card belongs to the non-trump suit, and y=1 indicates that the card belongs to the trump suit. Cards were represented in this manner for computational purposes.

5.1 Non-Determined Game

If all plays made in a game lead to the same outcome, it can be said that the game is determined. A game that is completely determined has no need of strategy, so the game was first examined for this behavior.

It was found that while the game is likely to be determined at low n (when there are only a few cards left to play in the game), it is not determined at higher values of n.

The following is an example of a determined game with three cards left to play:

Player's hand: {[1,0], [2,1], [3,1]}
Opponent's hand: {[1,1], [2,0], [3,0]}

This game yields the payoff matrix:

	[1 1] [2 0] [3 0]	[1 1] [3 0] [2 0]	[2 0] [1 1] [3 0]	[2 0] [3 0] [1 1]	[3 0] [1 1] [2 0]	[3 0] [2 0] [1 1]
[1, 0] [2, 1] [3, 1]	-1	-1	2	-1	2	-1
[1 0] [3 1] [2 1]	-1	-1	2	-1	2	-1
[2 1] [1 0] [3 1]	2	2	-1	-1	-1	-1
[2 1] [3 1] [1 0]	2	2	-1	-1	-1	-1
[3 1] [1 0] [2 1]	2	2	-1	-1	-1	-1
[3 1] [2 1] [1 0]	2	2	-1	-1	-1	-1

The payoff matrix has the first row and first column representing the cards of each player in the order that they are played. The row player corresponds to the cards in "player's hand", and the column player to the cards in "opponent's hand". The negative payoff values represent impossible games, while the non-negatives represent the number of tricks won by the row player in each scenario.

The strategy vector for this game is [1/6, 1/6, 1/6, 1/6, 1/6, 1/6], for both the random and competent opponents, since each row chosen yields the same result (four invalid games, and two games with a payoff of 2). Here, playing the cards in any particular order will make no difference to the game's outcome, so a strategy is unnecessary.

The following is an example of a non-determined game with four cards left to play:

Player's hand: {[2,0], [5,1], [5,0], [4,0]}
Opponent's hand: {[1,0], [3,0], [4,1], [2,1]}

Two rows selected from this game's payoff matrix:

-1	-1	3	-1	3	-1	-1	-1	2	-1	2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	3	-1	-1	-1	-1	3	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	3

The first row corresponds to playing: [2,0], [5,1], [5,0], [4,0].
The second row corresponds to playing: [4 0], [5 0], [5 1], [2 0].

There are four valid games for each row. The first row has two games that will result in a payoff of 2, and two games that result in a payoff of 3. The second row has four games that all result in a payoff of 3. Therefore, starting with the card [4,0] results in a superior strategy than starting with the card [2,0]. The strategy vector for this game is:

[0, 0, 0, 0, 0, 0, 0, 0, 0, 1/8, 0, 1/8, 0, 0, 0, 1/8, 1/8, 1/8, 0, 0, 0, 1/8, 1/8, 1/8].

Here, there are 8 games out of 24 that are optimal, so following one of these game paths is desired.

Since the player's choice matters in the second scenario, the simplified version of Tarneeb is shown to be a non-determined game. This indicates that it may be possible to develop a strategy that is superior to playing the cards in a random order.

5.2 Analysis of Optimal Strategies

Various combinations of game states were analyzed with each player receiving between two and seven cards (the computation time became too large beyond seven cards per player). The data from each individual analysis is not included in this report due to the sheer volume of data. A quantitative analysis of the data was performed, and the trends in the optimal strategies found will be discussed.

It was found that the random and competent opponents should be played with the same strategy, although a random opponent still has a higher expected payoff in general. The higher expected payoff of playing a random opponent is due to the possibility of the random opponent making a 'bad' (non-optimal) move. The strategies end up being the same for each type of opponent due to the nature of the strategy calculation – the rows that maximize the minimum value (coherent opponent) also give the highest expected value since the payoffs in these rows are larger.

Analysis of the strategy vectors of non-determined game states indicated that, for any arrangement of cards, there are multiple optimal game paths. Hence, no game arrangement results in a pure strategy where only one game path should be followed. This is likely due to the simple nature of the game formulation, where choosing from two different strategies can still lead to the same cards being pitted against each other.

Since the game tends to become determined as n decreases, an analysis was carried out on which cards should be played first in non-determined games. The pattern that emerged was that the first card to be played should be a card that will win definitely (those cards that the player knows are of the highest value in a suit that an opponent still has). This can be thought of as a player keeping control of the game, since winning a round lets a player begin the next round. The opponent also can't take advantage of a bad move and win an extra trick if the card is the highest. This 'greedy' approach is sensible for the competent opponent since playing winning hands first allows the row player to restrict the flow of the game towards an optimal outcome. This approach is even better against a random opponent, since the random opponent may waste a high value card that would have won them a trick later in the game.

5.3 Limitations

While providing a sensible approach to strategy for this game, the method outlined does have its limitations. The primary limitation with this method is that the optimal strategy is determined by comparing each row of the payoff matrix independently. In practice, each card played restricts the game tree to a set of rows in the payoff matrix. This means that the column player then can select the best move from among any of the valid rows. In light of this, the solutions given are still good strategies because the column player will then chose a card that restricts the game to a set of columns, and the row player will have the same advantage of selecting the best move from any of the valid columns.

Using this observation, the solution could be improved by looking at which cards lead to the largest proportion of optimal games. For example, two different choices of which first card to play may both lead to a possible payoff of 4 at the end of a particular game, but one of the two card choices may have a larger proportion of possible sub-games that have payoff 4 (and therefore would be a better strategical decision). This more in-depth strategy was not included in this formulation due to time and computational constraints.

5.4 Extrapolation to the Full Game

While the full game has more depth and is much less predictable than the simplified game, some of the strategical aspects outlined can be applied to the full game. Namely, it is best to take a 'greedy' approach while playing, to try and maximize the total number of tricks obtained. By opening with cards that are sure to win, a player can restrict the game to a more narrow state before deciding how to play cards that may or may not win the round, depending on how the opponents chose to play their losing hands.

This strategy accounts for the increased likelihood of a player making a bad move in the full game. Although the opponents would not be playing randomly, the stochastic elements will require them to take some chances with which cards they play. By being greedy initially, opponents are given more time to make bad plays while you stick with safe hands. It is also possible to take note of which cards have been played by the opposing team, and then take advantage of any bad decisions they make since this approach allows a player to keep control at the beginning of the game by winning hands early on.

6: Conclusion

The simplified version of Tarneeb was found to be a non-determined game, and so it was possible to develop a strategy that directed the game towards maximum payoff. While the patterns found in the strategies of the simplified game are sensible and can be applied to the full game, certain limitations indicated that further analysis would allow even better solutions to the simplified game to be revealed. As more elegant solutions to the simplified game are uncovered, additional strategies to the full game would likely also emerge. Ultimately, the results give reason to believe that this is a game that does have optimal strategies, and that further investigation of this problem is warranted.

Appendix A: Code

```
import numpy as np
import itertools
import csv
np.set_printoptions(threshold=np.inf)

# Let a card be represented as [x,y], with x as the card value and y as the suit (0 or 1, with 1 as trump)

suit =  [[1,0], [2,0], [3,0], [4,0], [5,0]] #the regular suit
trump = [[1,1], [2,1], [3,1], [4,1], [5,1]] #the trump suit
hand = [[1,0], [2,0], [3,0], [4,1], [5,0]] # players cards
ohand = [[1,1], [2,1], [3,1], [4,0], [5,1]] #opponents cards

#Sort players hands by suits
def sort(cards, trump, suit):
  for x in cards:
    if x[1] == 1:
      trump.append(x)
    else:
      suit.append(x)

#Generate the rows and columns of the payoff matrix
payoff_row = list(itertools.permutations(hand, len(hand)))
payoff_col = list(itertools.permutations(ohand, len(ohand)))
payoff_row_array = np.asarray(payoff_row)
payoff_col_array = np.asarray(payoff_col)

#Now, we want to fill in a payoff matrix that would have the row x columns represented as
# payoff_row x  payoff_col
#We take the first entry in the array, and set it against each other possible hand to fill in the first row
#for the payoff matrix, then repeat for each other value


# Evaluate two cards to see how they match up

def evaluate(x, y):
  if (x[0] >= y[0] and x[1] >= y[1]) or (x[1] > y[1]):
    return 1
  else:
    return 0
```

#Compare two games, assuming that there are no identical cards in each hand

```
def compare(player_game, opponent_game):
    score = 0
    count = 0
    while count <= (len(hand) - 1):
        if count == 0:
            in_hand_trump = []
            in_hand_suit = []
            o_hand_trump = []
            o_hand_suit = []
            sort(player_game.tolist(), in_hand_trump, in_hand_suit)
            sort(opponent_game.tolist(), o_hand_trump, o_hand_suit)
        if player_game[count, 1] == 1 and opponent_game[count, 1] == 0 and len(o_hand_trump) != 0 :  # x
trump, y suit, opponent has trump (illegal)
            score += -10
        if player_game[count, 1] == 0 and opponent_game[count, 1] == 1 and len(o_hand_suit) != 0 :  # x suit,
y trump, opponent has suit (illegal)
            score += -10
        else:
            if player_game[count, 1] == 1:
                in_hand_trump.remove(player_game[count].tolist())
            else:
               in_hand_suit.remove(player_game[count].tolist())
            if opponent_game[count, 1] == 1:
                o_hand_trump.remove(opponent_game[count].tolist())
            else:
                o_hand_suit.remove(opponent_game[count].tolist())
            score += evaluate(player_game[count], opponent_game[count])
        count += 1
    return score
```

```
#Generate the payoff matrix
payoff = []

for x in range(len(payoff_row)):
    count = 0
    while count <= (len(payoff_col) - 1):
        payoff.append(compare(payoff_row_array[x], payoff_col_array[count]))
        count +=1

payoff_size = len(payoff_row)
payoff_array = np.array(payoff)
payoff_array.shape = (payoff_size, payoff_size)

# This can export the payoff matrix as an excel file
payname = str(hand) + "payoff.csv"
np.savetxt(payname, payoff_array, fmt='%1.0f', delimiter=',')


# Now we want to be able to access the payoff matrix and extract the optimal strategy
# for the randomly playing opponent and the competent opponent
# The code blocks for the random and competent opponent must be run separately.

solution = []


#RANDOM OPPONENT:
# For our random opponent, we maximize the amount of total payoff over all possible games
# that an opponent will randomly choose

for x in range(payoff_size):
    rowsum = 0
    count = 0
    for y in np.nditer(payoff_array[x]):
        if y >= 0:
            rowsum += y
            count += 1
    if count == 0 :
         solution.append(np.asarray([0]))
    else:
        solution.append([rowsum/float(count)])
```

```
solution_array = np.asarray(solution)

# Now we take every strategy that gives the maximum expected value,
#and give that index in the solution array a value of 1 while giving all other strategies the value of 0
#we can then 'multiply' this and our payoff row matrix to show each strategy that is optimal


for x in np.nditer(solution_array, op_flags=['readwrite']):
   if x == max(solution):
      x[...] = 1
   else:
      x[...] = 0

strategy = solution_array / np.sum(solution_array)

solution_readout = []

#List all the optimal games
for x in range(payoff_size):
   if solution_array[x] == 1:
      solution_readout.append(payoff_row_array[x])
solution_readout_array = np.asarray(solution_readout)

#this lets us output the optimal games in an excel file
name1 = str(hand)
name2 = str(ohand)
title = "rand" + name1 + name2 + ".csv"
output_file = open(title, 'w')
a = csv.writer(output_file)
a.writerows(solution_readout_array)
output_file.close()

#save strategy array as csv
np.savetxt("randstrat" + title, strategy,  delimiter=',')

#Random opponent code block ends here
```

```
#COMPETENT OPPONENT:

# For our competent opponent, we know that for whatever row we choose as our strategy,
# our opponent will choose the minimum value in that row
# so we want to find the rows with the maximum minimum value

for x in range(payoff_size):
    rowmin = 0
    count = 0
    for y in np.nditer(payoff_array[x]):
        if y >= 0:
            if count == 0:
                rowmin = y
            if y < rowmin:
                rowmin = y
            count += 1
    solution.append(rowmin)

solution_array = np.asarray(solution)


# Now we take every strategy that has the maximum value in our solution array
#and give that index in the solution array a value of 1 while giving all other strategies the value of 0
#we can then 'multiply' this and our payoff row matrix to show each strategy that is optimal

for x in np.nditer(solution_array, op_flags=['readwrite']):
    if x == max(solution_array):
        x[...] = 1
    else:
        x[...] = 0

solution_array.shape = (payoff_size, 1)

strategy = solution_array / float(np.sum(solution_array))

solution_readout = []


for x in range(payoff_size):
    if solution_array[x] == 1:
        solution_readout.append(payoff_row_array[x])
solution_readout_array = np.asarray(solution_readout)
```

```
#this lets us output the optimal games in an excel file
name1 = str(hand)
name2 = str(ohand)
title = name1 + name2 + ".csv"
output_file = open(title, 'w')
a = csv.writer(output_file)
a.writerows(solution_readout_array)
output_file.close()

#save strategy array as csv
np.savetxt("strat" + title, strategy,  delimiter=',')

#code end
```

Appendix B: References

Drake, Fred L., et al. “Python 2.7.12 Documentation.” *python.org.* Python Software Foundation, 26 Nov. 2016. 29 Nov. 2016. <https://docs.python.org/2/>

Wassaf, Salah, et al. “Tarneeb.” *pagat.com.* Pagat: Card Game Rules, 10 May. 2012. 29 Nov. 2016. <https://www.pagat.com/boston/tarneeb.html>