CS/ECE/ISyE 524 — Introduction to Optimization — Spring 2023

Final Course Project: Due 5/5/23

Dynasty Fantasy Football Trade Analyzer

Luke Neuendorf (lneuendorf@wisc.edu (mailto:lneuendorf@wisc.edu))

Table of Contents

1. Introduction
2. Mathematical Model
3. Solution
4. Results and Discussion
   A. Optional Subsection
5. Conclusion

1. Introduction

In this project, I created an optimization model to calculate the value of the assets in a dynasty fantasy football trade. The goal is to see which side of the trade contains more value, or which side "wins" the trade. I also created a second model to generate a fair trade counteroffer to the user's input offer based on the asset values generated with the first model.

In "regular" fantasy football, a group of 8-12 people draft teams of NFL offensive players. These teams compete in head-to-head matchups for a period of fourteen weeks. After fourteen weeks, the top teams compete in a three-week playoff tournament, where one team is crowned champion. A fantasy football starting lineup varies depending on the format of the league and can consist of 1-2 quarterbacks, 2 running backs, 2-3 wide receivers, 1 tight end, and 1-2 flex spots. A flex spot can hold a running back, wide receiver, or tight end.

Dynasty fantasy football differs from "regular" fantasy football in that dynasty managers keep the same players from year to year. Furthermore, dynasty fantasy football has yearly rookie drafts where each team drafts several of that year's NFL rookie players onto their teams. The order of these drafts is decided based on the previous year's standings. Given that teams are left with the same players year after year, trading is vital. Every league member chooses a different strategy. Some dynasty managers build young rosters looking for longevity, while others look to build older rosters full of household names. The purpose of a trade analyzer is to see which team is "winning" a potential trade or if a trade is even. Analyzing trade value through the unbiased form of a computer-generated model can take away the bias a player may have towards certain types of players, such as flashy or young players. Over time, it can help a manager's team accrue more value by consistently making trades that are even or that they're projected to "win."

The dataset for this project consists of 3,532 trades that happened from April 10 to April 30 in 12-team, two-quarterback dynasty leagues. The dataset was scraped from fantasycalc.com (https://www.fantasycalc.com/database). The trades consist of two sides, with a maximum of four pieces per side. The pieces consist of NFL offensive players or rookie draft picks.

2. Mathematical models

2.A. Model 1 : Quadratic Program (QP)

The first model's objective is to minimize the total squared difference between the value of two sides of a trade. This is done across all 3,532 example trades in the dataset. The two sides of a trade are denoted as side "A" and side "B". The value of one side of a trade is defined as the sum of the values/weights of the assets on that side of the trade. The individual asset values/weights are denoted as w_i. Note that an asset is a player or rookie draft pick. The number of assets on side A of a trade (n_assets_A_i) and the number of assets on side B of a trade (n_assets_B_i) vary depending on the trade. However, trades in the dataset are constrained to a maximum of four assets per side. The assumption is made that no asset has a value of zero, so the weights have a lower-bound of one in order to prevent the model from bringing asset values to zero.

$$
\begin{aligned}
\underset{w \in \mathbb{R}^n}{\text{minimize}} \quad & \sum_{i=1}^{\text{n\_trades}} (\text{diff}_i)^2 & \\
\text{subject to:} \quad & \text{diff}_i = \text{ValueA}_i - \text{ValueB}_i & \forall i \in \{1, 2, \dots, \text{n\_trades}\} \\
& \text{ValueA}_i = \sum_{j=1}^{\text{n\_assets\_A}_i} w_j & \forall i \in \{1, 2, \dots, \text{n\_trades}\} \\
& \text{ValueB}_i = \sum_{j=1}^{\text{n\_assets\_B}_i} w_j & \forall i \in \{1, 2, \dots, \text{n\_trades}\} \\
& w_i \geq 1.0 & \forall i \in \{1, 2, \dots, \text{n\_assets}\} \\
& \text{n\_assets\_A}_i,\ \text{n\_assets\_B}_i \in \{1, 2, 3, 4\} & \forall i \in \{1, 2, \dots, \text{n\_trades}\} \\
& \text{n\_trades} = 3,532\ , \quad \text{n\_assets} = 209 &
\end{aligned}
$$

2.B. Model 2 : Mixed-Integer Linear Program (MILP)

The second model is using the asset values, or weights, generated from model 1 to output a counter offer closest in total value to a given an input offer. It seeks to minimize the magnitude of the difference in value between the input offer and generated counter offer. The asset weights/values are denoted as w_i. A binary value y_i is used to denote if an asset was included in the input offer, and is a 209 element vector for each possible player and rookie draft pick. The variable y_i is one if that corresponding player/pick is in the input offer, and zero otherwise. The variable x_i is simular to y_i, except it is selected in order to minimize the objective. This 209 element vector, x_i, acts as an indicator variable to denote if an asset should be included in the counter offer. The model is solving for the optimal combination of players not in the input offer closest to the input offer value. The model is constrained to returning a combination of players/picks equal to the number of players/picks decided by the user with the "n_assets_in_counter" variable.

$$
\begin{aligned}
\underset{x}{\text{minimize}} \quad & |\,\text{InputValue} - \text{CounterValue}\,| \\
\text{subject to:} \quad & \text{InputValue} = \sum_{i=1}^{n=209} w_i y_i \\
& \text{CounterValue} = \sum_{i=1}^{n=209} w_i x_i \\
& \sum_{i=1}^{n=209} x_i = \text{n\_assets\_in\_counter} \\
& x, y \in \{0, 1\} \\
& \text{n\_assets\_in\_counter} \in \{1, 2, \dots, 209\}
\end{aligned}
$$

3. Solution

Load the necessary libraries:

In [10]:
```
using JuMP, HiGHS, Ipopt  # optimization packages
using CSV, DataFrames     # data reading/handling packages
using Printf              # printing package
```

There are two datasets, we will load them into this notebook below:

In [11]:
```
players_and_picks = CSV.read("players_and_picks.csv", DataFrame)
trade_data = CSV.read("trade_data.csv", DataFrame);
```

The "players_and_picks" dataset lists all 209 players/picks and their corresponding index. We will use this index as an id for each player/pick.

In [12]:
```
first(players_and_picks,5)
```

Out[12]: 5×1 DataFrame

Row	Assets
	String31
1	Patrick Mahomes
2	Josh Allen
3	Justin Jefferson
4	Jalen Hurts
5	Ja'Marr Chase

The "trade_data" dataset consists of the 3,532 trades, storing the two sides of the trade: side A and side B. The "Side A Assets" and "Side B Assets" columns list the player id's corresponding with their index in the "players_and_picks" dataset.

In [13]:
```
first(trade_data,5)
```

Out[13]: 5×5 DataFrame

Row	Date	Side A List	Side B List	Side A Assets	Side B Assets
	String7	String	String	String31	String31
1	27-Apr	['Kyler Murray']	['DK Metcalf']	[35]	[33]
2	25-Apr	['Travis Etienne', '2024 Round 1', 'Gabe Davis']	['Stefon Diggs', "D'Andre Swift", 'Aaron Rodgers']	[23, 107, 204]	[27, 66, 81]
3	24-Apr	['2023 Round 1', 'Davante Adams', 'Jeff Wilson']	['2023 Round 2', 'Christian Watson', 'Brian Robinson']	[37, 179, 200]	[50, 113, 201]
4	16-Apr	['Justin Jefferson']	['2023 Round 1', '2023 Round 2', '2024 Round 1', 'Pat Freiermuth']	[3]	[75, 200, 201, 204]
5	24-Apr	['2023 Round 3', 'Garrett Wilson', 'DJ Moore', '2024 Round 3']	['Justin Jefferson']	[12, 46, 202, 206]	[3]

3.A. Model 1 Implementation

We will use the trades from the "players_and_picks" dataset to generate weight values associated with each player. The code for the model is shown below:

In [14]:
```
n_assets = size(players_and_picks,1)  # number of players/picks (209)
n_trades = size(trade_data,1)         # nubmer of trades (3,532)

m = Model(Ipopt.Optimizer)
@variable(m, asset_weights[1:n_assets]) # weight value of each player/pick (what we are solving for)
@variable(m, difs[1:n_trades]) # variable storing abs value of difference between side_A & side_B of ith trade
@variable(m, side_A[1:n_trades])
@variable(m, side_B[1:n_trades])

# Assign assets in Side A
for i in 1:n_trades
    assets = parse.(Int, split(trade_data[i,4][2:end-1], ",")) # parse the string containing the asset ids/indices
    n_assets_A = length(assets)
    @constraint(m, side_A[i] == sum(asset_weights[assets[j]] for j=1:n_assets_A)) # side A value in ith trade
end

# Assign assets in Side B
for i in 1:n_trades
    assets = parse.(Int, split(trade_data[i,5][2:end-1], ",")) # parse the string containing the asset ids/indices
    n_assets_B = length(assets)
    @constraint(m, side_B[i] == sum(asset_weights[assets[j]] for j=1:n_assets_B)) # side B value in ith trade
end

@constraint(m, [i=1:n_assets], asset_weights[i] >= 1.0) # set lower bound on asset weights

# Take the difference in value between the two sides of the trade
@constraint(m, [i=1:n_trades], difs[i] == side_A[i] - side_B[i])

# Minimize the sum of the differences squared
@objective(m, Min, sum(difs.^2))
set_silent(m)
optimize!(m)

# Store the generated asset (player/pick) weight values
asset_weights = value.(asset_weights);
```

A preview of the assigned weights generated from this model can be seen with the code snippet below, although I will show all of the assigned values in the results section.

In [15]:
```
for i in 1:10
    @printf("%.0f %s: %.5f\n",i,players_and_picks[i,1],asset_weights[i])
end
```

```
1 Patrick Mahomes: 3.40373
2 Josh Allen: 3.31371
3 Justin Jefferson: 2.94678
4 Jalen Hurts: 3.06686
5 Ja'Marr Chase: 2.63691
6 Joe Burrow: 2.98827
7 Justin Herbert: 2.42373
8 Lamar Jackson: 2.14667
9 CeeDee Lamb: 2.12642
10 Trevor Lawrence: 2.50231
```

3.B. Model 2 Implementation

The below funciton uses an implementation of model 2 to generate the closest counter trade offer in terms of value to the users input offer by solving a mixed-integer linear program. The input variable "asset_list" is a list of players/picks in the input offer, and the input variable "n_assets_in_counter" is the desired number of assets the counter offer should consist of.

In [16]:
```
function generate_counter_offer(asset_list, n_assets_in_counter)
    # Convert the players_and_picks DataFrame object to a List
    all_asset_list = collect(players_and_picks.Assets)

    # Convert names in input asset list to their corresponding indice ids in the "players_and_picks" dataset
    input_indices = Int[]
    n_assets = size(players_and_picks,1)
    for asset in asset_list
        if asset in all_asset_list
            push!(input_indices, findfirst(x -> x == asset, all_asset_list))
        else
            @printf("Error: \"%s\" is not in the \"players_and_picks\" dataset.",asset)
            return
        end
    end

    ###################### Model 2: Solving for optimal counter-offer ###################################
    n_input = size(input_indices,1) # store the number of assets in input offer

    m = Model(HiGHS.Optimizer)
    @variable(m, x[1:n_assets], Bin) # indicator variable: 1 if asset is in counter-offer, 0 otherwise
    @variable(m, difference)

    @constraint(m, [i=1:n_input], x[input_indices[i]] == 0) # don't allow model to return input assests
    @constraint(m, sum(x) == n_assets_in_counter) # force model to return counter with n_assets_in_counter assets

    # Calculate the total value of the input and counter-offer
    @expression(m, input_value, sum((asset_weights[input_indices[i]]) for i=1:n_input))
    @expression(m, output_value, sum(asset_weights[i]*x[i] for i=1:n_assets))

    # Take the absolute vlaue of input value and counter-offer value
    @constraint(m, difference >= input_value -  output_value)
    @constraint(m, difference >=  output_value - input_value)

    # Minimize the difference between the input value assets and the counter-offer assets value
    @objective(m, Min, difference)
    set_silent(m)
    optimize!(m)
    indicator_vars = value.(x)
    ###################################################################################################

    # Print Input Offer
    input_sum = 0
    for (i,val) in enumerate(asset_list)
        input_sum += asset_weights[input_indices[i]]
    end
    @printf("\033[1mYour Offer Total Value: %.5f\033[0m\n",input_sum)
    for (i,val) in enumerate(asset_list)
        @printf("\t%s - (%.5f)\n",all_asset_list[input_indices[i]],asset_weights[input_indices[i]])
    end

    # Print Generated Counter Offer
    counter_sum = 0
    for (i,val) in enumerate(indicator_vars)
        if (0.99 < val) & (val < 1.01)
            counter_sum += asset_weights[i]
        end
    end
    @printf("\033[1m\nCounter Offer Total Value:  %.5f\033[0m\n",counter_sum)
    for (i,val) in enumerate(indicator_vars)
        if (0.99 < val) & (val < 1.01)
            @printf("\t%s - (%.5f)\n",all_asset_list[i],asset_weights[i])
        end
    end
end;
```

4. Results and discussion

4.A. Model 2 Results

A counter offer can be generated by updating the inputs below.

In [17]:
```
########## Input Variables: Feel Free to Update These ###############
asset_list = ["Patrick Mahomes","Garrett Wilson","Stefon Diggs"]
n_assets_in_counter = 4
#####################################################################

# Run Model 2:
generate_counter_offer(asset_list, n_assets_in_counter)
```

```
Your Offer Total Value: 6.75547
        Patrick Mahomes - (3.40373)
        Garrett Wilson - (1.86111)
        Stefon Diggs - (1.49063)

Counter Offer Total Value:  6.75547
        Trevor Lawrence - (2.50231)
        Travis Etienne - (1.39457)
        Dak Prescott - (1.78358)
        DJ Moore - (1.07502)
```

4.B. Model 1 Results

In the tables below are the weight values generated corresponding to each player/pick. One major limitation of this model is that many of the values are equal to 1, the minimum possible value. This is because the model is trying to minimize the difference in the values of each side of a trade, and consequently minimizes the values themselves.

Rank	Player/Pick	Value
1	Patrick Mahomes	3.4037
2	Josh Allen	3.3137
3	Jalen Hurts	3.0669
4	Joe Burrow	2.9883
5	Justin Jefferson	2.9468
6	Ja'Marr Chase	2.6369
7	Trevor Lawrence	2.5023
8	Justin Herbert	2.4237
9	Lamar Jackson	2.1467
10	CeeDee Lamb	2.1264
11	Justin Fields	1.9999
12	A.J. Brown	1.993
13	Deshaun Watson	1.9626
14	Garrett Wilson	1.8611
15	Jonathan Taylor	1.8464
16	Dak Prescott	1.7836
17	Amon-Ra St. Brown	1.7798
18	Christian McCaffrey	1.7497
19	Mark Andrews	1.7255
20	Chris Olave	1.6715
21	Jaylen Waddle	1.6492
22	Breece Hall	1.6311
23	Tyreek Hill	1.6229
24	Travis Kelce	1.6082
25	Kyle Pitts	1.5495
26	Kenneth Walker	1.5431
27	Cooper Kupp	1.5129
28	Stefon Diggs	1.4906
29	Josh Jacobs	1.4836
30	Kyler Murray	1.4543
31	Tee Higgins	1.4497
32	Davante Adams	1.4256
33	DK Metcalf	1.4015
34	Travis Etienne	1.3946
35	T.J. Hockenson	1.375
36	Saquon Barkley	1.3434
37	Tua Tagovailoa	1.3429
38	Drake London	1.3115
39	George Kittle	1.2323
40	DeVonta Smith	1.2019

Rank	Player/Pick	Value
41	Najee Harris	1.1786
42	Deebo Samuel	1.1468
43	J.K. Dobbins	1.1459
44	Austin Ekeler	1.1388
45	Daniel Jones	1.1357
46	Aaron Jones	1.1344
47	Nick Chubb	1.1338
48	Zay Jones	1.1331
49	Zach Ertz	1.1089
50	Rhamondre Stevenson	1.0883
51	DJ Moore	1.075
52	Jordan Love	1.0698
53	Michael Gallup	1.068
54	2023 Round 1	1.0543
55	Geno Smith	1.0465
56	Kenneth Gainwell	1.0382
57	David Njoku	1.0321
58	Derrick Henry	1.0154
59	Christian Watson	1.0134
60	Jerry Jeudy	1.001
61	Trey Lance	1.001
62	Tony Pollard	1
63	Michael Pittman	1
64	Javonte Williams	1
65	Chris Godwin	1
66	Treylon Burks	1
67	Brandon Aiyuk	1
68	Jameson Williams	1
69	Terry McLaurin	1
70	George Pickens	1
71	Kenny Pickett	1
72	Dallas Goedert	1
73	Miles Sanders	1
74	Russell Wilson	1
75	D'Andre Swift	1
76	Marquise Brown	1
77	Derek Carr	1
78	Dameon Pierce	1
79	Jahan Dotson	1
80	Jared Goff	1

Rank	Player/Pick	Value
81	Amari Cooper	1
82	Christian Kirk	1
83	Pat Freiermuth	1
84	DeAndre Hopkins	1
85	Dalvin Cook	1
86	Kirk Cousins	1
87	Calvin Ridley	1
88	Aaron Rodgers	1
89	David Montgomery	1
90	Alvin Kamara	1
91	Brock Purdy	1
92	Cam Akers	1
93	Diontae Johnson	1
94	Rachaad White	1
95	Mike Evans	1
96	Joe Mixon	1
97	Darren Waller	1
98	Mac Jones	1
99	Mike Williams	1
100	Isiah Pacheco	1
101	Matthew Stafford	1
102	Elijah Moore	1
103	AJ Dillon	1
104	Keenan Allen	1
105	Desmond Ridder	1
106	James Cook	1
107	Rashod Bateman	1
108	Kadarius Toney	1
109	James Conner	1
110	Jimmy Garoppolo	1
111	Evan Engram	1
112	Gabe Davis	1
113	JuJu Smith-Schuster	1
114	Alexander Mattison	1
115	Courtland Sutton	1
116	Sam Howell	1
117	Tyler Lockett	1
118	Brian Robinson	1
119	Greg Dulcich	1
120	Dalton Schultz	1

Rank	Player/Pick	Value
121	Antonio Gibson	1
122	Khalil Herbert	1
123	Tyler Allgeier	1
124	Michael Thomas	1
125	Chigoziem Okonkwo	1
126	Allen Lazard	1
127	Wan'Dale Robinson	1
128	Jakobi Meyers	1
129	Cole Kmet	1
130	Trey McBride	1
131	Alec Pierce	1
132	Jamaal Williams	1
133	Rondale Moore	1
134	Darnell Mooney	1
135	Skyy Moore	1
136	Ryan Tannehill	1
137	Devin Singletary	1
138	Rashaad Penny	1
139	Elijah Mitchell	1
140	Baker Mayfield	1
141	Brandin Cooks	1
142	Chase Claypool	1
143	Odell Beckham	1
144	Dawson Knox	1
145	Romeo Doubs	1
146	Isaiah Likely	1
147	John Metchie	1
148	Damien Harris	1
149	D'Onta Foreman	1
150	Kareem Hunt	1
151	K.J. Osborn	1
152	Samaje Perine	1
153	Donovan Peoples-Jones	1
154	Adam Thielen	1
155	DJ Chark	1
156	Ezekiel Elliott	1
157	Mike Gesicki	1
158	Irv Smith	1
159	Joshua Palmer	1
160	Nico Collins	1

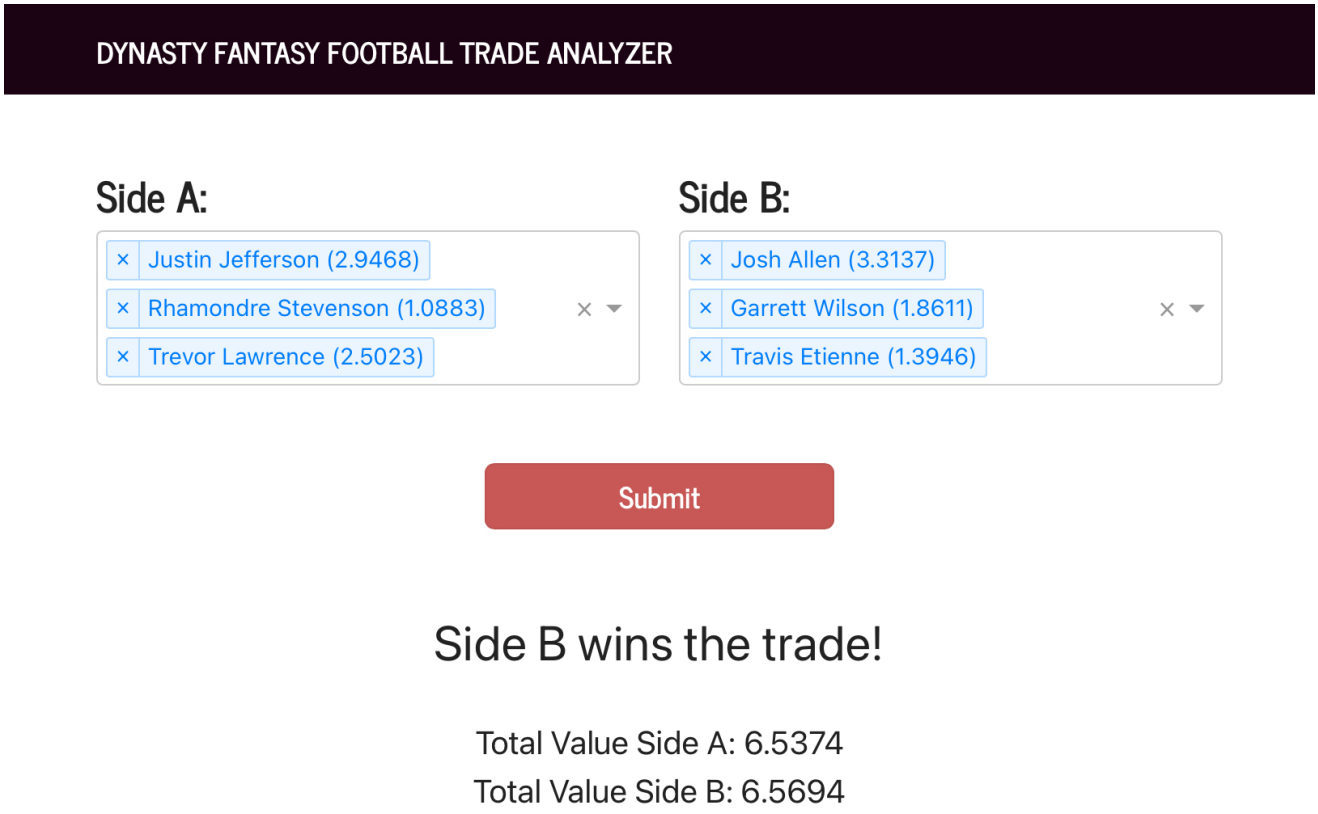
Rank	Player/Pick	Value
161	Michael Carter	1
162	Terrace Marshall	1
163	Zach Wilson	1
164	Tyler Boyd	1
165	Clyde Edwards-Helaire	1
166	Rashid Shaheed	1
167	Leonard Fournette	1
168	Malik Willis	1
169	Juwan Johnson	1
170	Noah Fant	1
171	Hunter Renfrow	1
172	Tyler Higbee	1
173	Cade Otton	1
174	Jelani Woods	1
175	Allen Robinson	1
176	Curtis Samuel	1
177	Gerald Everett	1
178	Jaylen Warren	1
179	Isaiah Hodgins	1
180	Jeff Wilson	1
181	Sam Darnold	1
182	Jake Ferguson	1
183	Hayden Hurst	1
184	Tyquan Thornton	1
185	Raheem Mostert	1
186	Zamir White	1
187	Justyn Ross	1
188	Bailey Zappe	1
189	Khalil Shakir	1
190	Gardner Minshew	1
191	Parris Campbell	1
192	Mike White	1
193	Daniel Bellinger	1
194	Isaiah Spiller	1
195	James Robinson	1
196	Cordarrelle Patterson	1
197	Mecole Hardman	1
198	Tyler Huntley	1
199	Chuba Hubbard	1
200	Jerome Ford	1

Rank	Player/Pick	Value
201	2023 Round 2	1
202	2023 Round 3	1
203	2023 Round 4	1
204	2024 Round 1	1
205	2024 Round 2	1
206	2024 Round 3	1
207	2025 Round 1	1
208	2025 Round 2	1
209	2025 Round 3	1

I also implemented a web app which allows you to select the pieces of a dynasty trade and analyzes the trade.

The app can be found at dynasty-trade-analyzer.com (https://dynasty-trade-analyzer.herokuapp.com/).

Below is a screenshot of the web app.



5. Conclusion

In this project, I generated dynasty fantasy football trade values corresponding to 209 different players and rookie draft picks using a quadratic program. Furthermore, I used a mixed-integer linear program to output potential counter offers to a trade input.

One of the limitations of model 1, is that it minimizes the differences between the total value of two sides of a trade. This results in the model making the asset values as small as possible, which led to over half of the values being one. In the future, one could reformulate the problem to generate a unique value/weight for every player/pick. This could be done by adding a constraints where the weight of the i^{th} element must be greater than or equal to the $(i+1)^{th}$ element plus some small number.

$$w_i \geq w_{i+1} + b \quad \text{where b is some small value}$$

However, weight value corresponding to an asset would need to be dynamic for this to work.