Linear Programming Model for NHL Roster

Ethan Chilton, Naveen Elliot, Burke Mayling, Max Margolis

ISE 3230

November 2023

Table of Contents

Background

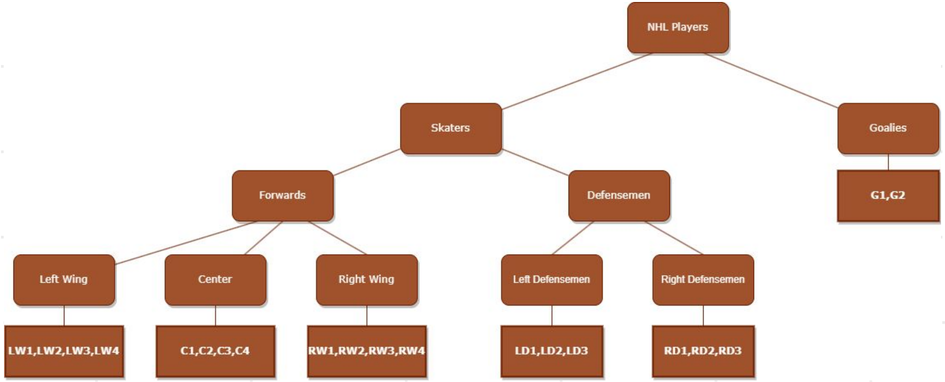
- We are constructing the best possible NHL roster of 20 players (18 skaters and 2 goalies) in terms of player production and cost-effectiveness.
- Optimization revolves around how much money each player gets paid to maximize GSVA (Game Score Value Added) value.
  - GSVA is a catch-all stat that incorporates a player's production and play-driving while accounting for usage.
- For goalies, we are maximizing the SAE (Saves Above Expected) value
  - The job of a goalies is to prevent the puck from going into the net, so we thought it would be appropriate to look at this statistic, because there is no GSVA for goalies

Introduction

- Objective:
  - To maximize the GSVA and SAE value for an NHL roster.
- Process Overview:
  - Scrape data for 2022-23 season from NHL statistics websites to collect GSVA and SAE value, position, and cap hit for each player.
  - Formulate MILP
    - Maximize GSVA and SAE
    - Remain below salary cap
    - Have one player for each position
  - Find optimal roster
  - Use results for post-optimality analysis

Program

- <u>Decision Variables</u>:
  - $skater_i$ – Binary variable (equals 1 if skater is selected).
  - $goalie_i$ – Binary variable (equals 1 if goalie is selected).
- <u>Objective</u>:
  - Maximize $Z = \Sigma\ GSVA_i\ skater_i$ - Total GSVA value of selected skaters.
  - Maximize $Z = \Sigma\ SAE_i\ goalie_i$ - Total SAE value of selected goalies.
- <u>Constraints:</u>
  - Team Cap Hit <= 82.5 million
  - Positions:
    - Right Defender (RD) - 3
    - Left Defender (LD) - 3
    - Center (C) - 3
    - Left Wing (LW) - 4
    - Right Wing (RW) - 4
    - Goalie (G) - 2



Problem Formulation

$$\max Z = \sum_{i=1}^{n(\text{Skaters})} x_i \cdot GSVA_i + \sum_{j=1}^{n(\text{Goalies})} y_j \cdot SAE_j$$

St. ·· Cap Hit ··

$$\sum_{i=1}^{n(\text{Skaters})} (\text{Cap\_Hit}_i \cdot x_i) + \sum_{j=1}^{n(\text{Goalies})} (\text{Cap\_Hit}_j \cdot y_j) \leq 82{,}500{,}000$$

·· Position Constraint For Skaters ··

$$\sum_{i \in P_k} x_i = 1, \quad \forall k \in P$$

where P_k is the subset of skaters eligible for position k

·· Position Constraint For Goalie ··

$$\sum_{j \in G} y_j = 1$$

assuming one goalie position

$$x_i, y_j \in \{0, 1\}$$

$$x_i, y_j \geq 0$$

x_i is binary for skaters (1 is player selected, 0 otherwise)
y_j is binary for goalies (1 is player selected, 0 otherwise)
P is set of all positions for Sktrs
G is set of all positions for Glies

Data Pre-Processing

```
import gurobipy as gp
from gurobipy import GRB
import pandas as pd

# Load the CSV files
file_path = 'skaters_refined.csv'
skaters = pd.read_csv(file_path)
goalies = pd.read_csv('goalies_refined.csv')
goalies

# Ensure that new_position is of type string for the equality comparisons to work correctly
skaters['new_position'] = skaters['new_position'].astype(str)

# Ensure that CAP HIT is of type int for the equality comparisons to work correctly
goalies['CAP HIT'] = goalies['CAP HIT'].astype(int)
skaters['CAP HIT'] = skaters['CAP HIT'].astype(int)

# Using this dataframe with tiering for post optimality
skaters_tiered = skaters.copy()
goalies_tiered = goalies.copy()

# Removing the tiering for optimality problem
def remove_numbers(s):
    return ''.join([char for char in s if not char.isdigit()])
skaters['new_position'] = skaters['new_position'].apply(remove_numbers)
goalies['new_position'] = goalies['new_position'].apply(remove_numbers)
```

Optimality

```python
# Initialize the model
model = gp.Model("NHL Team Optimization")
model.setParam('OutputFlag', 0)

# Add a binary variable for each player
skater_vars = model.addVars(skaters.shape[0], vtype=GRB.BINARY, name="Skaters")
goalie_vars = model.addVars(goalies.shape[0], vtype=GRB.BINARY, name="Goalies")

# Set the objective to maximize the sum of GSVA
gsva = skaters['GSVA'].tolist()
sae = goalies['saves_above_expected'].tolist()
model.setObjective(
    gp.quicksum(gsva[i] * skater_vars[i] for i in range(skaters.shape[0])) +
    gp.quicksum(sae[j] * goalie_vars[j] for j in range(goalies.shape[0])),
    GRB.MAXIMIZE
)

# Add the salary cap constraints
cap_hit_skaters = skaters['CAP HIT'].tolist()
cap_hit_goalies = goalies['CAP HIT'].tolist()

model.addConstr(
    gp.quicksum(cap_hit_skaters[i] * skater_vars[i] for i in range(skaters.shape[0])) +
    gp.quicksum(cap_hit_goalies[j] * goalie_vars[j] for j in range(goalies.shape[0])) <= 82.5e6,
    "MaxCap"
)

# Add constraints to ensure exactly one player is selected for each position
model.addConstr(gp.quicksum(skater_vars[i] for i in skaters.index if skaters.loc[i, 'new_position'] == 'LW') == 4, "Four_LW")
model.addConstr(gp.quicksum(skater_vars[i] for i in skaters.index if skaters.loc[i, 'new_position'] == 'C') == 4, "Four_C")
model.addConstr(gp.quicksum(skater_vars[i] for i in skaters.index if skaters.loc[i, 'new_position'] == 'RW') == 4, "Four_RW")
model.addConstr(gp.quicksum(skater_vars[i] for i in skaters.index if skaters.loc[i, 'new_position'] == 'LD') == 3, "Three_LD"
model.addConstr(gp.quicksum(skater_vars[i] for i in skaters.index if skaters.loc[i, 'new_position'] == 'RD') == 3, "Three_RD"
model.addConstr(gp.quicksum(goalie_vars[j] for j in goalies.index) == 2, "Two_G")

# Optimize the model
model.optimize()

# Check if the model found an optimal solution
if model.status == GRB.OPTIMAL:
    print("The optimal value is:", model.objVal)
    selected_player_indices = [i for i in range(len(skater_vars)) if skater_vars[i].X > 0.5]
    selected_players = skaters.iloc[selected_player_indices]
    selected_goalie_indices = [i for i in range(len(goalie_vars)) if goalie_vars[i].X > 0.5]
    selected_goalies = goalies.iloc[selected_goalie_indices]
    print("Selected players are:")
    print(selected_players[['PLAYER', 'new_position', 'CAP HIT', 'GSVA']])
    print(selected_goalies[['PLAYER', 'new_position', 'CAP HIT', 'saves_above_expected']])
    print("Total Cap Hit:")
    print(sum(selected_players['CAP HIT'])+sum(selected_goalies['CAP HIT']))
    print("Total GSVA:")
    print(sum(selected_players['GSVA']))
```

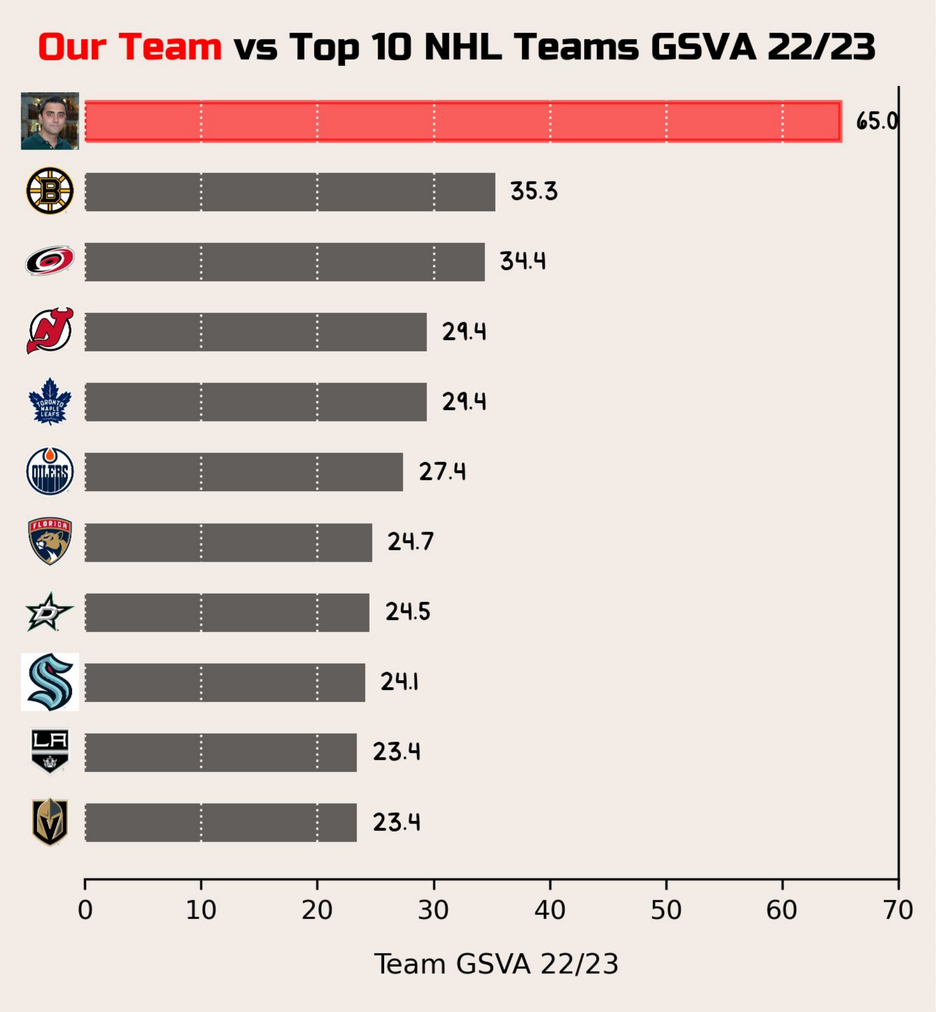
Optimality Results

- Output corresponds to the roster giving the highest GSVA value for each player.
- GSVA and SAE - 154.12
- Total cap hit- 82,448,333

```
The optimal value is: 154.11999999999998
Selected players are:
                  PLAYER new_position   CAP HIT  GSVA
             Brent Burns           RD   8000000   4.1
         Brandon Montour           RD   3500000   3.3
           Evan Bouchard           RD    863333   2.0
              Vince Dunn           LD   4000000   4.0
           Sebastian Aho           LD    825000   3.0
         Erik Gustafsson           LD    800000   2.4
          Connor McDavid            C  12500000   6.7
        Patrice Bergeron            C   2500000   3.4
           Tage Thompson            C   1400000   2.9
             Tim Stützle            C    925000   3.1
         Jason Robertson           LW   7750000   5.0
      Ryan Nugent-Hopkins           LW   5125000   3.7
         Andrei Kuzmenko           LW    950000   3.1
           Stefan Noesen           LW    762500   1.7
         Matthew Tkachuk           RW   9500000   5.4
          David Pastrnak           RW   6666667   4.6
              Zach Hyman           RW   5500000   4.1
           Matthew Boldy           RW    880833   2.5
           PLAYER new_position  CAP HIT  saves_above_expected
       Juuse Saros            G  5000000                 46.67
     Linus Ullmark            G  5000000                 42.45
Total Cap Hit:
82448333
Total GSVA:
65.0
```

Optimality Results

- Our project vs 10 best teams in NHL measured by their GSVA
- Our project is way ahead in first
  - This is because we are allowed to select the top players in each position to create our team



Post-Optimality - Adding Tiers

- Right Winger: RW1, RW2, RW3, RW4
- Center: C1, C2, C3, C4
- Left Winger: LW1, LW2, LW3, LW4
  - Divided into quartiles (groups of 4) based on GSVA
- Left Defender: LD1, LD2, LD3
- Right Defender: RD1, RD2, RD3
  - Divided into tripartites (groups of 3) based on GSVA
- Goalie: G1, G2
  - Divided into halves (groups of 2) based on SAE

```
# Initialize the model
model = gp.Model("NHL Team Optimization")
model.setParam('OutputFlag', 0)

# Add a binary variable for each player
skater_vars = model.addVars(skaters_tiered.shape[0], vtype=GRB.BINARY, name="Skaters")
goalie_vars = model.addVars(goalies_tiered.shape[0], vtype=GRB.BINARY, name="Goalies")

# Set the objective to maximize the sum of GSVA
gsva = skaters_tiered['GSVA'].tolist()
sae = goalies_tiered['saves_above_expected'].tolist()
model.setObjective(
    gp.quicksum(gsva[i] * skater_vars[i] for i in range(skaters_tiered.shape[0])) +
    gp.quicksum(sae[j] * goalie_vars[j] for j in range(goalies_tiered.shape[0])),
    GRB.MAXIMIZE
)

# Add the salary cap constraints
cap_hit_skaters = skaters_tiered['CAP HIT'].tolist()
cap_hit_goalies = goalies['CAP HIT'].tolist()

model.addConstr(
    gp.quicksum(cap_hit_skaters[i] * skater_vars[i] for i in range(skaters_tiered.shape[0])) +
    gp.quicksum(cap_hit_goalies[j] * goalie_vars[j] for j in range(goalies_tiered.shape[0])) <= 82.5e6,
    "MaxCap"
)

# Creating a salary cap constraint that teams need to spend close to their cap
model.addConstr(gp.quicksum(cap_hit_skaters[i] * skater_vars[i] for i in range(skaters_tiered.shape[0])) +
                gp.quicksum(cap_hit_goalies[j] * goalie_vars[j] for j in range(goalies_tiered.shape[0])) >= 80.5e6, "MinC

# Add constraints to ensure exactly one player is selected for each position
# We are using the tiering dataframe for this
for position in skaters_tiered['new_position'].unique():
    position_players = skaters_tiered[skaters_tiered['new_position'] == position].index.tolist()
    model.addConstr(gp.quicksum(skater_vars[i] for i in position_players) == 1, f"One_{position}")
for position in goalies_tiered['new_position'].unique():
    position_players = goalies_tiered[goalies_tiered['new_position'] == position].index.tolist()
    model.addConstr(gp.quicksum(goalie_vars[j] for j in position_players) == 1, f"One_{position}")

# Optimize the model
model.optimize()

# Check if the model found an optimal solution
if model.status == GRB.OPTIMAL:
    print("The optimal value is:", model.objVal)
    selected_player_indices = [i for i in range(len(skater_vars)) if skater_vars[i].X > 0.5]
    selected_players = skaters_tiered.iloc[selected_player_indices]
    selected_goalie_indices = [i for i in range(len(goalie_vars)) if goalie_vars[i].X > 0.5]
    selected_goalies = goalies_tiered.iloc[selected_goalie_indices]
    print("Selected players are:")
    print(selected_players[['PLAYER', 'new_position', 'CAP HIT', 'GSVA']])
    print(selected_goalies[['PLAYER', 'new_position', 'CAP HIT', 'saves_above_expected']])
    print("Total Cap Hit:")
    print(sum(selected_players['CAP HIT'])+sum(selected_goalies['CAP HIT']))
    print("Total GSVA:")
    print(sum(selected_players['GSVA']))
```

Post Optimality Adding Tiers - Results

- Output is based on tiers of quality in each position and a minimum bound of 80,500,000
- GSVA and SAE - 80
- Total cap hit- 81,492,500

The optimal value is: 80.0
Selected players are:

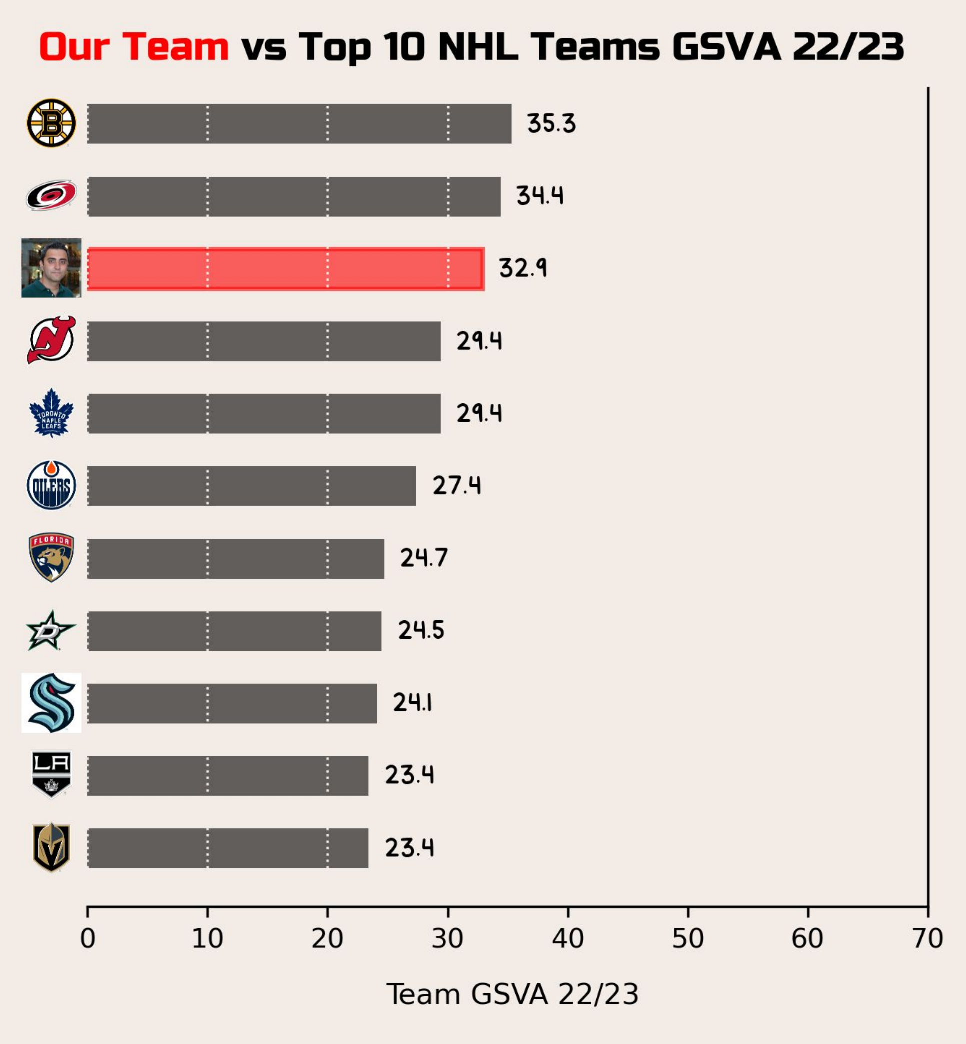
PLAYER	new_position	CAP HIT	GSVA
Adam Fox	RD1	9500000	4.7
Justin Faulk	RD3	6500000	-0.2
John Marino	RD2	4400000	0.9
Hampus Lindholm	LD1	6500000	4.4
Sean Durzi	LD2	1700000	1.0
Martin Fehervary	LD3	791667	-0.3
Connor McDavid	C1	12500000	6.7
Nazem Kadri	C2	7000000	1.6
Juuso Parssinen	C3	850833	0.6
Eric Staal	C4	750000	0.0
Jason Robertson	LW1	7750000	5.0
Marcus Johansson	LW2	1100000	1.3
Lukas Reichel	LW3	925000	0.3
Vitali Kravtsov	LW4	875000	-0.2
Matthew Tkachuk	RW1	9500000	5.4
Jordan Kyrou	RW2	2800000	1.5
Tanner Jeannot	RW4	800000	-0.2
Hudson Fasching	RW3	750000	0.4

PLAYER	new_position	CAP HIT	saves_above_expected
Juuse Saros	G1	5000000	46.67
Craig Anderson	G2	1500000	0.43

Total Cap Hit:
81492500
Total GSVA:
32.9

Post Optimality Adding Tiers - Results

- The 10 best teams compared to us
- Our project comes in third
  - Due to the limitations caused from dividing each position into quartiles/thirds/twos depending on the number of players in that position



Post-Optimality Analysis Code- Adding Variation in Performance

- Calculated the positional standard deviation for each player

- Used the player’s current GSVA (skaters) or SAE (goalkeepers) as the mean

- Randomly selected a point on a normal probability curve based on these two traits to represent the player’s adjusted GSVA or SAE

- Ran 1000 simulations to factor in seasonal variance

```python
# Getting the standard deviations of GSVA for each position based on the tiering (18 skaters and 2 goalies)
std_dev_skaters = skaters_tiered.groupby('new_position')['GSVA'].std().reset_index(name='std_dev')
std_dev_goalies = goalies_tiered.groupby('new_position')['saves_above_expected'].std().reset_index(name='std_dev')
```

```python
# setting a function to find the optimal goalies and skaters based on variance within each position
def refindOptimalPlayers(seed, skaters, goalies):
    np.random.seed(seed)
    skaters2 = skaters.copy()
    goalies2 = goalies.copy()
    # Randomly selecting the adjusted GSVA and SAE for each skater and goalie based on the variance in their position
    perturbed_gsva = np.zeros(len(skaters_tiered))
    for index, row in skaters_tiered.iterrows():
        for index2, row2 in std_dev_skaters.iterrows():
            if row['new_position'] == row2['new_position']:
                perturbed_gsva[index] = np.random.normal(loc=row['GSVA'], scale=std_dev_skaters.iloc[index2,1])
    skaters2['new_GSVA'] = perturbed_gsva
    skaters2['seed'] = seed
    perturbed_sae = np.zeros(len(goalies))
    for index, row in goalies_tiered.iterrows():
        for index2, row2 in std_dev_goalies.iterrows():
            if row['new_position'] == row2['new_position']:
                perturbed_sae[index] = np.random.normal(loc=row['saves_above_expected'], scale=std_dev_goalies.iloc[index2, 1
    goalies2['new_SAE'] = perturbed_sae
    goalies2['seed'] = seed

    # Initialize the model
    model2 = gp.Model("NHL Team Optimization")
    model2.setParam('OutputFlag', 0)

    # Add a binary variable for each player
    skater_vars2 = model2.addVars(skaters2.shape[0], vtype=GRB.BINARY, name="Skaters")
    goalie_vars2 = model2.addVars(goalies2.shape[0], vtype=GRB.BINARY, name="Goalies")

    # Set the objective to maximize the sum of GSVA
    gsva2 = skaters2['new_GSVA'].tolist()
    sae2 = goalies2['new_SAE'].tolist()
    model2.setObjective(
        gp.quicksum(gsva2[i] * skater_vars2[i] for i in range(skaters2.shape[0])) +
        gp.quicksum(sae2[j] * goalie_vars2[j] for j in range(goalies2.shape[0])),
        GRB.MAXIMIZE
    )

    # Add the salary cap constraints
    cap_hit_skaters = skaters2['CAP HIT'].tolist()
    cap_hit_goalies = goalies2['CAP HIT'].tolist()

    model2.addConstr(
        gp.quicksum(cap_hit_skaters[i] * skater_vars2[i] for i in range(skaters2.shape[0])) +
        gp.quicksum(cap_hit_goalies[j] * goalie_vars2[j] for j in range(goalies2.shape[0])) <= 82.5e6,
        "MaxCap"
    )

    model2.addConstr(gp.quicksum(cap_hit_skaters[i] * skater_vars2[i] for i in range(skaters2.shape[0])) +
                     gp.quicksum(cap_hit_goalies[j] * goalie_vars2[j] for j in range(goalies2.shape[0])) >= 80.5e6, "MinCap")

    # Add constraints to ensure exactly one player is selected for each position
    for position in skaters2['new_position'].unique():
        position_players = skaters2[skaters['new_position'] == position].index.tolist()
        model2.addConstr(gp.quicksum(skater_vars2[i] for i in position_players) == 1, f"One_{position}")
    for position in goalies2['new_position'].unique():
        position_players = goalies[goalies2['new_position'] == position].index.tolist()
        model2.addConstr(gp.quicksum(goalie_vars2[j] for j in position_players) == 1, f"One_{position}")

    # Optimize the model
    model2.optimize()

    # Check if the model found an optimal solution
    if model2.status == GRB.OPTIMAL:
        objVal = model2.objVal
        selected_player_indices2 = [i for i in range(len(skater_vars2)) if skater_vars2[i].X > 0.5]
        selected_players2 = skaters2.iloc[selected_player_indices2]
        selected_goalie_indices2= [i for i in range(len(goalie_vars2)) if goalie_vars2[i].X > 0.5]
        selected_goalies2 = goalies2.iloc[selected_goalie_indices2]
        capHit = sum(selected_players2['CAP HIT']) + sum(selected_goalies2['CAP HIT'])
        return capHit, objVal, selected_players2, selected_goalies2
```

```python
import numpy as np

# Running 1000 iterations to limit the amount of randomness
capHitList = []
optimalValueList = []
selectedSkatersFrame = pd.DataFrame()
selectedGoaliesFrame = pd.DataFrame()
for seed in range(1000):
    skaters = skaters.copy()
    goalies = goalies.copy()
    # Getting cap hit, objective function, selected skaters, and selected goalies
    capHit, objVal, selectedSkaters, selectedGoalies = refindOptimalPlayers(seed, skaters_tiered, goalies_tiered)
    # Appending everything on top of each other
    capHitList.append(capHit)
    optimalValueList.append(objVal)
    selectedSkatersFrame = pd.concat([selectedSkatersFrame, selectedSkaters])
    selectedGoaliesFrame = pd.concat([selectedGoaliesFrame, selectedGoalies])
```

```python
# Look at most common players, average cap hit, average optimal value, and average GVSA
print("Average Cap Hit: ", np.mean(capHitList))
print("Average Optimal Value: ", np.mean(optimalValueList))
print(selectedSkatersFrame['PLAYER'].value_counts().head(18))
selectedGoaliesFrame['PLAYER'].value_counts()
print("Average GSVA: ", selectedSkatersFrame.groupby(['seed'])['GSVA'].sum().mean())
```

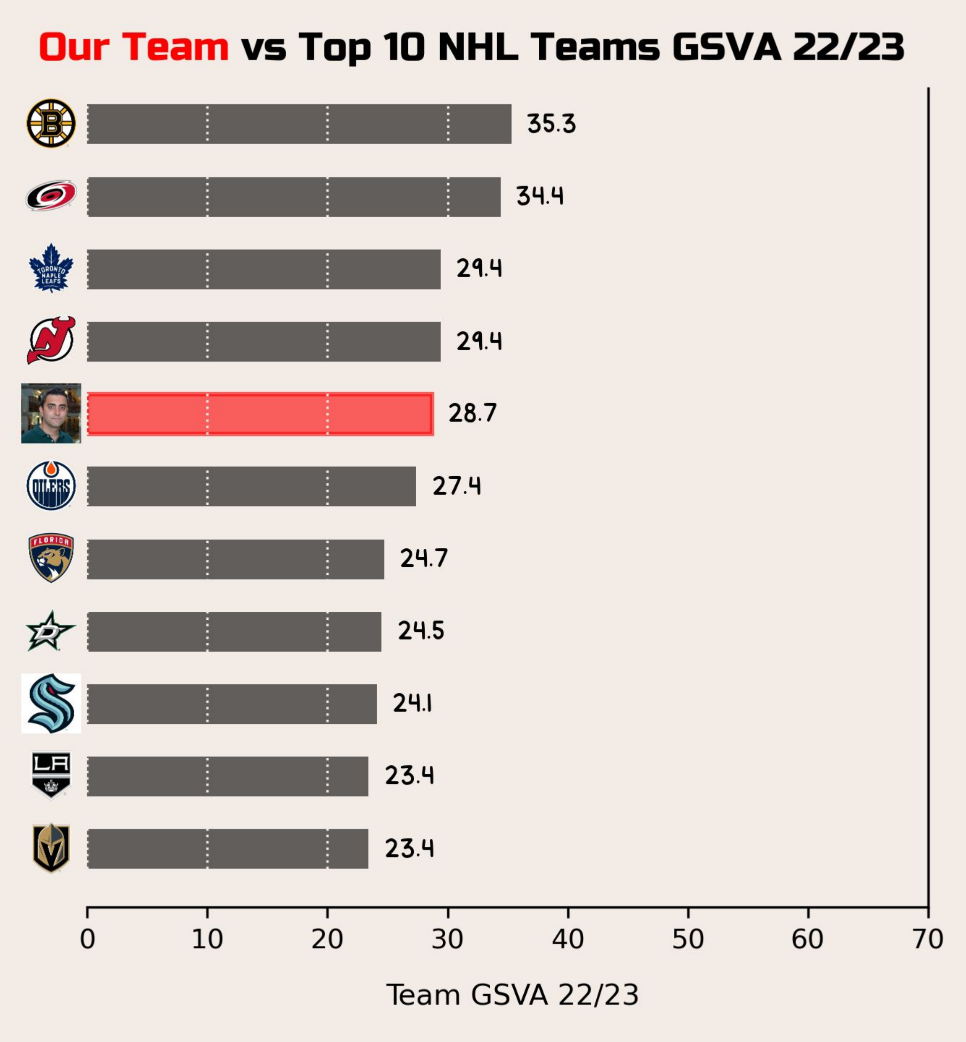
Results- Adding variation in performance

- Output corresponds to the most frequently chosen players out of 1000 iterations.
- Ex: Connor McDavid was chosen on 764 out of 1000 iterations.
- Average cap hit- 81,616,738
- Average maximum GSVA and SAE value- 108.91

```
Average Cap Hit:  81616738.815
Average Optimal Value:  108.90842114965115
PLAYER
Connor McDavid          764
Jason Robertson         709
Matthew Tkachuk         506
Hampus Lindholm         350
Adam Fox                306
Juuso Parssinen         230
Jordan Kyrou            207
Vince Dunn              204
Ryan McLeod             200
Scott Mayfield          175
David Pastrnak          173
Dougie Hamilton         153
John Marino             150
Sean Durzi              147
Matias Maccelli         146
Marcus Johansson        146
Alexander Wennberg      145
Brent Burns             145
Name: count, dtype: int64
Average GSVA:  28.7117
```

Results- Adding variation in performance

- The 10 best teams compared to us
- Our project comes in fifth
  - Due to the limitations caused from dividing each position into quartiles/thirds/twos depending on the demand
  - Also, our positional variance can be quite harsh in certain positions, which could limit the effectiveness of our model



Appendix: Link to Video

https://clipchamp.com/watch/IGwO1bMgdQu

Appendix: Link to Github Repository

https://github.com/burke-m/ISE3230-Project

Appendix: Team Member Tasks

- Burke- Attended group meetings, worked on data cleaning and spreadsheet optimization, worked through drafts of code and wrote code for post optimality analysis.
- Ethan- Attended group meetings, worked through constraints and optimization problem, helped create report.
- Max- Attended group meetings, worked on data cleaning and spreadsheet optimization, revised initial draft of code to work properly, fixed up mathematical problem formulation
- Naveen- Attended group meetings, came up with first draft of the code, helped create report, and adjusted code to make final optimality problem.