Operation Research: Assignment Problem

MONFERO, JOHN BENEDICT

Top 5 Swimmers on a 200-yard Junior Swimming Olympics

The coach of an age group swim team needs to assign swimmers to a 200 - yard medley relay team to send to the Junior Olympics. Since most of his best swimmers are very fast in more than one stroke, it is not clear which swimmer should be assigned to each of the four strokes. The five fastest swimmers and the best times (in seconds) they have achieved in each of the strokes (for 50 yards) are as follows:

Goal: Minimize	Carl	Chris	David	Tony	Ken
Backstroke	37.70	32.90	33.80	37.00	35.40
Breaststroke	43.40	33.10	42.20	34.70	41.80
Butterfly	33.30	28.50	38.90	30.40	33.60
Freestyle	29.20	26.40	29.60	28.50	31.10
Unassigned	0	0	0	0	0

The coach wishes to determine how to assign four swimmers to the four different strokes to minimize the sum of the corresponding best times.

In [1]:

```
import numpy as np
from scipy.optimize import linear_sum_assignment

# Define the cost matrix (including the dummy row)
cost_matrix = np.array([
    [37.7, 32.9, 33.8, 37.0, 35.4],  # Backstroke
    [43.4, 33.1, 42.2, 34.7, 41.8],  # Breaststroke
    [33.3, 28.5, 38.9, 30.4, 33.6],  # Butterfly
    [29.2, 26.4, 29.6, 28.5, 31.1],  # Freestyle
    [0.0,  0.0,  0.0,  0.0,  0.0]    # Dummy task
])

# Solve the assignment problem
row_ind, col_ind = linear_sum_assignment(cost_matrix)

# Display the optimal assignment
strokes = ["Backstroke", "Breaststroke", "Butterfly", "Freestyle", "Unassigned"]
swimmers = ["Carl", "Chris", "David", "Tony", "Ken"]

assignment = [(strokes[row], swimmers[col]) for row, col in zip(row_ind, col_ind)]

# Print the results
print("Optimal Assignments:")
for stroke, swimmer in assignment:
    print(f"{stroke} -> {swimmer}")
```

```
Optimal Assignments:
Backstroke -> David
Breaststroke -> Tony
Butterfly -> Chris
Freestyle -> Carl
Unassigned -> Ken
```

Goal: Minimize	Carl	Chris	David	Tony	Ken
Backstroke	37.70	32.90	33.80	37.00	35.40
Breaststroke	43.40	33.10	42.20	34.70	41.80
Butterfly	33.30	28.50	38.90	30.40	33.60
Freestyle	29.20	26.40	29.60	28.50	31.10
Unassigned	0	0	0	0	0

Minimum Lines	Carl	Chris	David	Tony	Ken
Backstroke	2.00	0.00	0.00	2.50	0.00
Breaststroke	7.50	0.00	8.20	0.00	6.20
Butterfly	2.00	0.00	9.50	0.30	2.60
Freestyle	0.00	0.00	2.30	0.50	2.20
Unassigned	0	0	0	0	0

Row Reduction	Carl	Chris	David	Tony	Ken
Backstroke	4.80	0.00	0.90	4.10	2.50
Breaststroke	10.30	0.00	9.10	1.60	8.70
Butterfly	4.80	0.00	10.40	1.90	5.10
Freestyle	2.80	0.00	3.20	2.10	4.70
Unassigned	0	0	0	0	0

Decision	Carl	Chris	David	Tony	Ken
Backstroke	2.00	0.00	0.00	2.50	0.00
Breaststroke	7.50	0.00	8.20	0.00	6.20
Butterfly	2.00	0.00	9.50	0.30	2.60
Freestyle	0.00	0.00	2.30	0.50	2.20
Unassigned	0	0	0	0	0

Column Reduction	Carl	Chris	David	Tony	Ken
Backstroke	2.00	0.00	0.00	2.50	0.00
Breaststroke	7.50	0.00	8.20	0.00	6.20
Butterfly	2.00	0.00	9.50	0.30	2.60
Freestyle	0.00	0.00	2.30	0.50	2.20
Unassigned	0	0	0	0	0

Airline-Crew Management

An airline that operates flights between Delhi and Bombay has the following timetable. Pair the flights, so as to minimize the total layover time for the crew. The plane, whichreaches its destination, cannot leave that place before 4 hours of rest.

Station A:	Delhi	
Flight Number	Departure	Arrival
101	9am	11am
102	10am	12nn
103	4pm	6pm
104	7pm	9pm

Station B:	Bombay	
Flight Number	Departure	Arrival
201	10am	12nn
202	12nn	2pm
203	3pm	5pm
204	8pm	10pm

Consider all pairs shall be able to perform and exhange their flights within the same-day basis

In [4]:
```
# Define the flight schedules (arrival and departure times in hours, 24-hour format)
delhi_to_bombay = {
    "101": 11,  # 11:00 AM
    "102": 12,  # 12:00 NN
    "103": 18,  # 6:00 PM
    "104": 21   # 9:00 PM
}

bombay_to_delhi = {
    "201": 10,  # 10:00 AM
    "202": 12,  # 12:00 NN
    "203": 15,  # 3:00 PM
    "204": 20   # 8:00 PM
}

# Create the cost matrix with large values for invalid assignments
num_outbound = len(delhi_to_bombay)
num_return = len(bombay_to_delhi)
cost_matrix = np.full((num_outbound, num_return), np.inf)
```

```python
outbound_flights = list(delhi_to_bombay.keys())
return_flights = list(bombay_to_delhi.keys())

# Fill the cost matrix with valid layovers (minimum 4-hour layover constraint)
for i, out_flight in enumerate(outbound_flights):
    for j, ret_flight in enumerate(return_flights):
        layover = bombay_to_delhi[ret_flight] - delhi_to_bombay[out_flight]
        if layover >= 4:
            cost_matrix[i, j] = layover  # We want to minimize this

# Preview the cost_matrix values:
cost_matrix
```

Out[4]:

```
array([[inf, inf,  4.,  9.],
       [inf, inf, inf,  8.],
       [inf, inf, inf, inf],
       [inf, inf, inf, inf]])
```

The `cost_matrix` values above shows that only at most three (3) possible scenario could satisfied the given condition as they cannot leave that place before 4 hours of rest

In [3]:

```python
# Apply the Hungarian algorithm
row_ind, col_ind = linear_sum_assignment(cost_matrix)

# Print optimal flight pairings
print("Optimal Flight Pairings (Minimized Layover Time):")
for i in range(len(row_ind)):
    if cost_matrix[row_ind[i], col_ind[i]] < np.inf:  # Ignore invalid assignments
        print(f"Flight {outbound_flights[row_ind[i]]} → Flight {return_flights[col_ind[i]]}
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[3], line 12
      9             cost_matrix[i, j] = layover  # We want to minimize this
     11 # Apply the Hungarian algorithm
---> 12 row_ind, col_ind = linear_sum_assignment(cost_matrix)
     14 # Print optimal flight pairings
     15 print("Optimal Flight Pairings (Minimized Layover Time):")

ValueError: cost matrix is infeasible
```

Hence, it gives us infeasible solution to the given assignment problem

Suppose that in between exchange flights could be at least a 2-day basis: As an example, flight 101 may leave 11am on day 1, but flight 201 shall leave 10am on another day

In [13]:

```python
# Define the flight schedules (arrival and departure times in hours, 24-hour format)

delhi_to_bombay = {
    "101": 11,  # 11:00 AM
    "102": 12,  # 12:00 NN
    "103": 18,  # 6:00 PM
    "104": 21   # 9:00 PM
}

bombay_to_delhi = {
    "201": 10,  # 10:00 AM, initially same day
    "202": 12,  # 12:00 NN, initially same day
    "203": 15,  # 3:00 PM, initially same day
    "204": 20   # 8:00 PM, initially same day
}

# Create the cost_matrix values:
```

```python
cost_matrix = np.array([[25., 27., 4., 9.],
                        [24., 26., 29., 8.],
                        [18., 20., 23., 28.],
                        [15., 17., 20., 25.]])
```

Hence, we have more feasible values to consider which satisfied the condition of the Airline-Crew Management

In [14]:
```python
# Apply the Hungarian algorithm
row_ind, col_ind = linear_sum_assignment(cost_matrix)

# Print optimal flight pairings
print("Optimal Flight Pairings (Minimized Layover Time):")
for i in range(len(row_ind)):
    if cost_matrix[row_ind[i], col_ind[i]] < np.inf:  # Ignore invalid assignments
        print(f"Flight {outbound_flights[row_ind[i]]} → Flight {return_flights[col_ind[i]]}
```

```
Optimal Flight Pairings (Minimized Layover Time):
Flight 101 → Flight 203 (Layover: 4.0 hrs)
Flight 102 → Flight 204 (Layover: 8.0 hrs)
Flight 103 → Flight 201 (Layover: 18.0 hrs)
Flight 104 → Flight 202 (Layover: 17.0 hrs)
```

Goal: Minimize Layover hours between Flight Numbers (2-Day Basis)

Flight	201	202	203	204
101	25	27	4	9
102	24	26	29	8
103	18	20	23	28
104	15	17	20	25

Row Reduction	201	202	203	204
101	21	23	0	5
102	16	18	21	0
103	0	2	5	10
104	0	2	5	10

Column Reduction	201	202	203	204
101	21	21	0	5
102	16	16	21	0
103	0	0	5	10
104	0	0	5	10

Optimal Assignment can obtain

Decision	201	202	203	204
101	21	21	0	5
102	16	16	21	0
103	0	0	5	10
104	0	0	5	10