Executive Summary

To design a race calendar more efficiently for the 2024 Formula 1 season that minimizes the total distance traveled and reduces greenhouse gas emission to reach sustainability goals. Applied integer programming and heuristics to the Traveling Salesman Problem (TSP), representing the race locations as nodes in a graph. Utilized brute-force and nearest neighbor approaches for smaller subsets. The findings show that the original F1 calendar was significantly optimized by employing advanced mathematical models, reducing the total distance traveled by approximately 49.02%. Further, compared to a heuristic solution, namely the nearest neighbor approach, an additional 9.04% reduction was achieved, showcasing the effectiveness of more refined optimization techniques. The result demonstrated that logistical efficiencies and environmental benefits can be realized through mathematical optimization. Consider the real-world scenarios, the new solution compared to the optimal solution increased 3991 miles, which is worst 10.4% than optimal solution.

Background / Introduction

The Formula 1 World Championship is a pinnacle of motor racing, taking place across various countries and continents. The scheduling of races (or 'Grands Prix') is a complex logistical endeavor that involves the movement of teams, equipment, and personnel around the globe. The traditional calendar can result in suboptimal travel routes, leading to increased costs, time inefficiencies, and a larger carbon footprint, which contrasts with the growing emphasis on environmental sustainability in the sport. The objective of this project is to optimize the F1 calendar to minimize the total distance traveled during the season. The project employs operations research and optimization techniques to tackle this problem. Gathering the geographical coordinates for each race track to construct a distance matrix by haversine in Python representing the travel distances between venues. Assume that they start from the beginning and go back to the beginning again, formulating the problem as a variant of the Traveling Salesman Problem (TSP) that seeks the shortest possible route. Applying integer programming techniques and heuristics to find an optimal or near-optimal sequence of races. While exact methods guarantee the shortest route, they are computationally expensive for a large number of races. Hence, heuristics such as the nearest neighbor algorithm are considered for faster approximations.

Modeling

Decision variable:

x_{ij} is 1 when the race track chooses City i to City j; x_{ij} is 0, otherwise.

u_i represents the position of track i in the tour.

Objective function:

Minimize $\sum_{i=1}^{22}\sum_{j=1}^{22} d_{ij}x_{ij}$ $(i \neq j)$, minimize the total distance traveled for the entire season of 2024.

d_{ij} is the distance from the race track i to the race track j, Table 1 shows detailly number.

Subject to:

$\sum_{j=1}^{22} x_{ij} = 1, for\ all\ i\ (i \neq j)$, race track i can go to all the other race track j, but only one race track j can be chosen.

$\sum_{i=1}^{22} x_{ij} = 1, for\ all\ j\ (i \neq j)$, all race track i can go to the race track j, but only one race track i can be chosen to go to j.

$u_i - u_j + nx_{ij} \leq n - 1\ for\ all\ 1 \leq i \neq j \leq n$

$u_1 = 0$

$1 \leq u_i \leq n - 1\ for\ all\ 2 \leq i \leq n$, to prevent the solution from having disconnected "subtours" that don't visit all the cities.

x_{ij} is 0 or 1.

Extensive model for extra task of 7)

Decision variable:

y_{ij} is 1 when the race track chooses City i at round j; y_{ij} is 0, otherwise.

Objective function:

Minimize $\sum_{i=1}^{22} \sum_{j=1}^{22} d_{ij} y_{ij}\ \ (i \neq j)$, minimize the total distance traveled for the entire season of 2024.

d_{ij} is the distance from the race track i to the race track j, Table 1 shows detailly number.

Subject to:

$y_{9j} = 0\ for\ j \leq 8$, the Montreal=9 race must happen after round 8.

$\sum_{j=1}^{7} y_{16j} + \sum_{j=18}^{22} y_{16j} \leq 1$, the Singapore race must happen between R1-R7 or R18-R22.

$y_{i,j-1} + y_{i,j} + y_{i,j+1} = 1\ for\ all\ j, i = 6, 17{,}20$, the races in the same countries (USA) cannot be at adjacent/consecutive rounds.

$y_{11} = 1$, the race in Bahrain must be the first race of the season.

$y_{22,22} = 1$, the race in UAE must be the last race of the season.

$y_{4,j-1} + y_{4,j} + y_{5,j-1} + y_{4,j} = 2$, the races in Suzuka and Shangai must be consecutive

$\sum_{j=1}^{11} y_{8j} \leq \sum_{j=12}^{22} y_{19j}$, if the Monaco race happens in the first half of the season, then the Sao Paulo race must be in the second half of the season.

$\sum_{j=1}^{11} (y_{2j} + y_{21j}) = 2$, either the Jeddah or Lusail race must be in the first half of the season

$\sum_{j=1}^{22} y_{ij} = 1, for\ all\ i\ (i \neq j)$, city i can assign to all the other round j, but only one round j can be chosen.

$\sum_{i=1}^{22} y_{ij} = 1, for\ all\ j\ (i \neq j)$, all city i can assign to the round j, but only one city i can be chosen at round j.

y_{ij} is 0 or 1.

Results and Analysis

1. Haversine's formula is an equation used to estimate the distance between two points on the Earth's surface, and is particularly well suited for calculating the distance between two points on the surface of a sphere with a large circular arc. The distance between the two points along a great circle of the sphere is

$$d = 2r * arcsin\sqrt{\sin^2(\frac{\varphi_2 - \varphi_1}{2}) + \cos\varphi_1 * \cos\varphi_2 * \sin^2(\frac{\lambda_2 - \lambda_1}{2})}$$

r is the radius of the sphere, $r = 6371km$.
φ_1, φ_2 are the latitude of point 1 and latitude of point 2.
λ_1, λ_2 are the longitude of point 1 and latitude of point 2.
d_{ij} is the distance between location of Round i and location of Round j, $i = 1, \ldots, 21; j = 2, \ldots, 22$.
The total miles for the entire duration of the 2024 season is

$$z = \sum_{i,j=1}^{22} d_{ij}, (i \neq j, i < j)$$

By using programming packages (import haversine in Python) to calculate the distance between two coordinates on the globe, the total miles for the entire duration of the 2024 season is 74991 miles.

2.

Figure 1 points out that the route exists many backtracking and long detours, which increases travel distance and time between races to increase carbon emissions. From travel efficiency and sustainable development perspective, the FIA need to minimize the travel distance to reduce carbon emissions by reprogramming race calendar, which is significant impact on reaching its sustainability goals by the greenhouse gas emission.

3.

Assuming that teams and racers only travel between the race track. By using programming packages (import haversine in Python) to calculate the distance.
Table 1 shows the distance the distance between the race track i and the race track j. Be cause the number of distances is enough large and the error is small, thus we preserve ze ro decimal place.

4.

There are 4! = 24 different calendars. Table 2 shows the total distance of 24 different calendars, the minimum distance is 1542 miles. There are 8 best calendars. Assume that they start from the beginning and go back to the beginning again, thereby it's a TSP problem.

For the 2024 season, if we assume there are 22 races, the number of different calendars would be 22 factorials ($22! = 1.24 \times 10^{21}$). This is a significantly larger number.

This is an extraordinarily large number, making it practically impossible to enumerate all calendars due to the computational and time constraints. Thereby, we would not want to take this approach to solve this problem.

5.

The race calendar is Sakhir, Lusail, Yas Marina, Jeddah, Baku, Budapes, Spielberg, Imola, Monza, Monaco, Spa-Francorchamps, Silverstone, Montreal, Miami, Austin, Mexico City, Las Vegas, Suzuka, Shangai, Singapore, Melbourne, Sao Paulo, Sakhir.

Total distance traveled is 42031 miles.

Figure 2 indicates fewer circulate than part 1), which behave shorter distance. The short er distance directly correlates to lower fuel consumption and fewer carbon emissions fro m the transportation of teams and equipment. The FIA has been actively working on sus tainability, aiming to have a net-zero carbon footprint by 2030. An optimized route like t his is a step in the right direction and reflects well on their commitment to this goal. A dditionally, Shorter travel distances can lead to reduced logistical complexity and potentia l cost savings. This could also mean less time spent in transit and more time available f or teams to prepare for races.

6.

The race calendar is Sakhir, Lusail, Yas Marina, Shangai, Suzuka, Singapore, Melbourne, Sao Paulo, Miami, Mexico City, Austin, Las Vegas, Montreal, Silverstone, Spa-Francorchamp s, Monza, Monaco, Imola, Spielberg, Budapest, Baku, Jeddah, Sakhir.

Total distance traveled is 38233 miles.

Figure 3 indicates fewer circulate than part 1), which behave shorter distance. 49.02% i mprovement of total distance traveled compared to the original F1 calendar. 9.04% impr ovement of total distance traveled compared to the nearest neighbor heuristic solution. This is a substantial reduction in travel distance compared with part 1), which directly c orrelates to carbon emissions, contributing to the FIA's sustainability goals.

TSP problem is a NP-hard problem, with the increasing of the location, the time and cos t required to obtain an exact solution is very high. The improvement rate of the solutio n does not grow significantly compared the accuracy solution with the solution of heuris tic algorithm. Therefore, from the cost point of view, heuristic solutions can be considere d when the problem is not very accurate.

7.

The race calendar is Sakhir, Jeddah, Lusail, Singapore, Melbourne, Shangai, Suzuka, Las V egas, Mexico City, Miami, Sao Paulo, Austin, Montreal, Silverstone, Spa-Francorchamps, Mo naco, Monza, Imola, Spielberg, Budapest, Baku, Yas Marina, Sakhir.

Total distance traveled is 42224 miles. The new solution compared to the optimal solutio n increased 3991 miles, which is worst 10.4% than optimal solution.

Figure 7 shows a longer route than optimal solution. In an idealized model, the shortest p ath is the only goal. However, real-world scenarios require complex considerations that ca n result in longer routes. These considerations can include geopolitical concerns, transportat ion infrastructure, or scheduling requirements for events.

8.

Assumption 1: TSP typically assumes the distance from A to B is the same as from B to A. In reality, logistical routes may have different distances and travel times due to one-way roads, traffic patterns, or availability of direct flights. We can use an asymmetric TSP model or include actual travel time data that accounts for directional differences.

Assumption 2: The model assumes that distances and travel times are constant and do not change over time. However, real-life scenarios involve dynamic factors such as weather conditions, traffic, or political events that can affect travel. We can integrate dynamic routing capabilities that can update the route in response to real-time information.

Assumption 3: The TSP aims to minimize total travel distance, but real-life logistics optimization might involve multiple objectives, such as minimizing cost, time, or environmental impact. We can develop a multi-objective optimization model that balances distance with other factors like cost and carbon emissions.

Appendix

Table 1:

Dij	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	0	782	7526	5004	4228	7572	2497	2692	6375	2430	3205	2255	2884	2636	991	3931	8021	8693	7341	8043	70	277
2	782	0	7964	5774	5010	7211	2212	2367	6170	2228	2992	2105	2677	2360	1440	4569	7849	8435	6559	8107	826	1004
3	7526	7964	0	5051	5020	9690	9996	10204	10402	9867	10531	9660	10260	10120	8071	3764	8877	8428	8117	8159	7457	7255
4	5004	5774	5051	0	918	7596	5964	6135	6576	5719	5908	5550	5820	5977	4563	3129	6729	7208	11643	5708	4973	4839
5	4228	5010	5020	918	0	8232	5584	5780	7047	5351	5701	5159	5547	5628	3938	2366	7483	8029	11529	6517	4189	4035
6	7572	7211	9690	7596	8232	0	5078	4891	1401	5144	4377	5327	4695	4937	6845	10534	1098	1283	4099	2170	7640	7829
7	2497	2212	9996	5964	5584	5078	0	217	3959	247	791	425	499	148	1948	6262	5638	6248	5972	5960	2566	2761
8	2692	2367	10204	6135	5780	4891	217	0	3804	429	695	629	468	159	2164	6477	5483	6076	5783	5849	2761	2959
9	6375	6170	10402	6576	7047	1401	3959	3804	0	3970	3192	4129	3514	3812	5549	9199	1679	2319	5071	2244	6439	6608
10	2430	2228	9867	5719	5351	5144	247	429	3970	0	779	211	457	283	1796	6111	5643	6279	6210	5899	2498	2685
11	3205	2992	10531	5908	5701	4377	791	695	3192	779	0	951	322	646	2504	6774	4867	5500	5917	5170	3272	3455
12	2255	2105	9660	5550	5159	5327	425	629	4129	211	951	0	632	491	1589	5901	5795	6443	6397	6005	2322	2504
13	2884	2677	10260	5820	5547	4695	499	468	3514	457	322	632	0	366	2203	6496	5188	5822	6045	5468	2951	3135
14	2636	2360	10120	5977	5628	4937	148	159	3812	283	646	491	366	0	2059	6375	5491	6102	5937	5815	2705	2898
15	991	1440	8071	4563	3938	6845	1948	2164	5549	1796	2504	1589	2203	2059	0	4317	7139	7849	7591	7067	1032	1132
16	3931	4569	3764	3129	2366	10534	6262	6477	9199	6111	6774	5901	6496	6375	4317	0	9845	10323	9931	8836	3864	3655
17	8021	7849	8877	6729	7483	1098	5638	5483	1679	5643	4867	5795	5188	5491	7139	9845	0	747	5024	1093	8083	8239
18	8693	8435	8428	7208	8029	1283	6248	6076	2319	6279	5500	6443	5822	6102	7849	10323	747	0	4617	1512	8758	8927
19	7341	6559	8117	11643	11529	4099	5972	5783	5071	6210	5917	6397	6045	5937	7591	9931	5024	4617	0	6082	7384	7549
20	8043	8107	8159	5708	6517	2170	5960	5849	2244	5899	5170	6005	5468	5815	7067	8836	1093	1512	6082	0	8092	8198
21	70	826	7457	4973	4189	7640	2566	2761	6439	2498	3272	2322	2951	2705	1032	3864	8083	8758	7384	8092	0	209
22	277	1004	7255	4839	4035	7829	2761	2959	6608	2685	3455	2504	3135	2898	1132	3655	8239	8927	7549	8198	209	0

Table 1: Distance from the race track i to the race track j

Table 2:

Routes	Round 1	Round 2	Round 3	Round 4	Total distance
1	Monaco	Silverstone	Spa-Francorchamps	Monza	1542
2	Monaco	Monza	Spa-Francorchamps	Silverstone	1542
3	Silverstone	Monaco	Monza	Spa-Francorchamps	1542
4	Silverstone	Spa-Francorchamps	Monza	Monaco	1542
5	Spa-Francorchamps	Silverstone	Monaco	Monza	1542
6	Spa-Francorchamps	Monza	Monaco	Silverstone	1542
7	Monza	Monaco	Silverstone	Spa-Francorchamps	1542
8	Monza	Spa-Francorchamps	Silverstone	Monaco	1542
9	Monaco	Spa-Francorchamps	Silverstone	Monza	1595
10	Monaco	Monza	Silverstone	Spa-Francorchamps	1595
11	Silverstone	Spa-Francorchamps	Monaco	Monza	1595
12	Silverstone	Monza	Monaco	Spa-Francorchamps	1595
13	Spa-Francorchamps	Monaco	Monza	Silverstone	1595
14	Spa-Francorchamps	Silverstone	Monza	Monaco	1595
15	Monza	Monaco	Spa-Francorchamps	Silverstone	1595

16	Monza	Silverstone	Spa-Francorchamps	Monaco	1595
17	Monaco	Silverstone	Monza	Spa-Francorchamps	2175
18	Monaco	Spa-Francorchamps	Monza	Silverstone	2175
19	Silverstone	Monaco	Spa-Francorchamps	Monza	2175
20	Silverstone	Monza	Spa-Francorchamps	Monaco	2175
21	Spa-Francorchamps	Monaco	Silverstone	Monza	2175
22	Spa-Francorchamps	Monza	Silverstone	Monaco	2175
23	Monza	Silverstone	Monaco	Spa-Francorchamps	2175
24	Monza	Spa-Francorchamps	Monaco	Silverstone	2175

Table 2: The possible calendars for the 1950 F1 season

Figure 1:

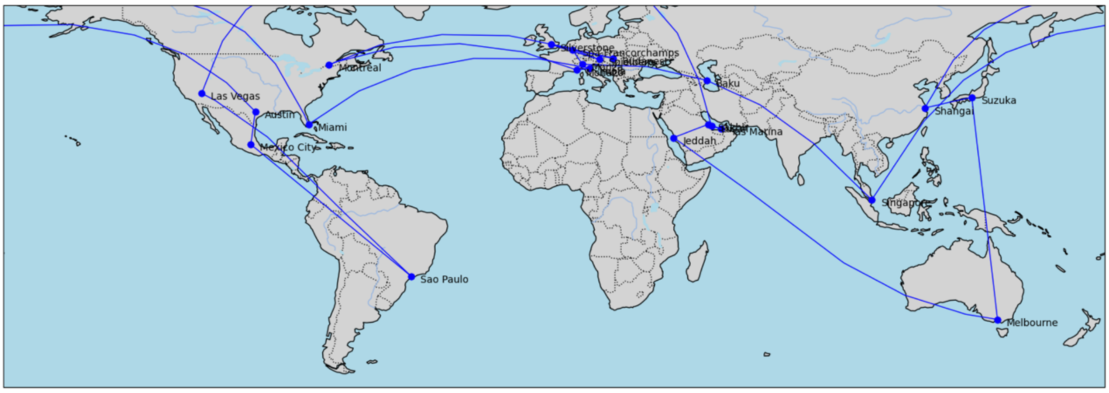

Figure 1: F1 2024 Calendar Route

Figure 2:

Figure 2: F1 2024 Season Race Locations Ordered

Figure 3:

Figure 3: F1 2024 Season Race Locations

Figure 4:

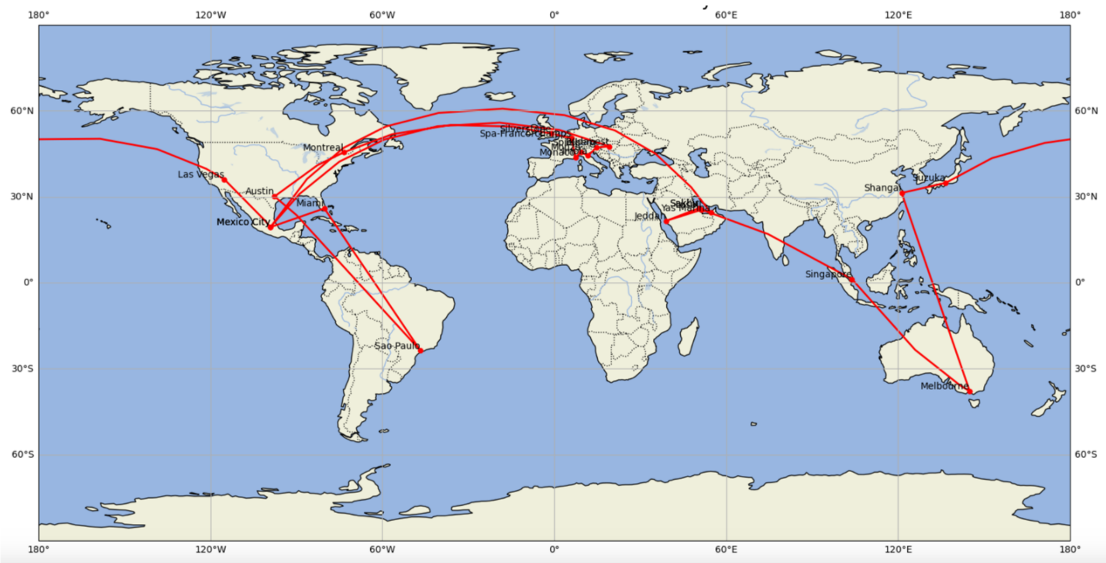

Figure 4: F1 2024 Season Race Location for extensive model

The code of calculating the total distance of original schedule in 1).

```
import numpy as np

def haversine(lat1, lon1, lat2, lon2):

    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    dlat = lat2 - lat1

    dlon = lon2 - lon1

    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2

    c = 2 * np.arcsin(np.sqrt(a))

    r = 6371

    return c * r

def total_travel_distance(data):

    total_distance_km = 0

    for i in range(len(data) - 1):

        total_distance_km += haversine(data.iloc[i]['Latitude'], data.iloc[i]['Longitude'],

                                       data.iloc[i + 1]['Latitude'], data.iloc[i + 1]['Longitude'])



    total_distance_km += haversine(data.iloc[-1]['Latitude'], data.iloc[-1]['Longitude'],

                                   data.iloc[0]['Latitude'], data.iloc[0]['Longitude'])
```

```
    total_distance_miles = total_distance_km * 0.621371
    return total_distance_miles
total_distance = total_travel_distance(f1_data)
total_distance
```

The code of plotting the original schedule figure in 2).

```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(1, 1, 1, projection=ccrs.PlateCarree())
ax.add_feature(cfeature.COASTLINE)
ax.add_feature(cfeature.BORDERS, linestyle=':')
ax.add_feature(cfeature.LAND, color='lightgray')
ax.add_feature(cfeature.OCEAN, color='lightblue')
ax.add_feature(cfeature.LAKES, color='lightblue')
ax.add_feature(cfeature.RIVERS)
ax.set_extent([-180, 180, -60, 65], crs=ccrs.PlateCarree())
for i, row in f1_data.iterrows():
    ax.plot(row['Longitude'], row['Latitude'], marker='o', color='red', markersize=5, transform=ccrs.Geodetic())
    ax.text(row['Longitude'] + 3, row['Latitude'] - 2, row['City'], horizontalalignment='left',
            transform=ccrs.Geodetic())
for i in range(len(f1_data) - 1):
    ax.plot([f1_data.iloc[i]['Longitude'], f1_data.iloc[i + 1]['Longitude']],
            [f1_data.iloc[i]['Latitude'], f1_data.iloc[i + 1]['Latitude']],
            color='blue', linewidth=1, marker='o', transform=ccrs.Geodetic())
ax.plot([f1_data.iloc[-1]['Longitude'], f1_data.iloc[0]['Longitude']],
        [f1_data.iloc[-1]['Latitude'], f1_data.iloc[0]['Latitude']],
        color='blue', linewidth=1, marker='o', transform=ccrs.Geodetic())
plt.title('F1 2024 Season Race Locations', fontsize=20)
plt.show()
```

The code of calculating the distance from the race track i to the race track j in 3).

```
def calculate_distance_matrix(df):
    n = len(df)
    distance_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if i != j:
                distance_matrix[i, j] = haversine(df.iloc[i]['Latitude'], df.iloc[i]['Longitude'],
                                                  df.iloc[j]['Latitude'], df.iloc[j]['Longitude'])
            else:
                distance_matrix[i, j] = 0
    return distance_matrix
distance_matrix_km = calculate_distance_matrix(f1_data)
distance_matrix_miles = distance_matrix_km * 0.621371
distance_matrix_miles
```

The code of calculating the all possible calendars for the 1950 F1 season in 4).

```
import numpy as np
from itertools import permutations
distance_matrix_1950 = np.array([
    [0, 695, 468, 159],    # Monaco
    [695, 0, 322, 646],    # Silverstone
    [468, 322, 0, 366],    # Spa-Francorchamps
    [159, 646, 366, 0]     # Monza
])
track_permutations = list(permutations(range(4)))
permutation_distances = {}
for permutation in track_permutations:
    total_distance = sum(distance_matrix_1950[permutation[i], permutation[(i + 1) % 4]] for i in range(4))
    permutation_distances[permutation] = total_distance
sorted_permutations = sorted(permutation_distances.items(), key=lambda item: item[1])
```

```
shortest_path = sorted_permutations[0]

sorted_permutations, shortest_path
```

The code of calculating F1 2024 Season Race Locations by Heuristic method in 5)

```
import numpy as np

import pandas as pd

file_path = '/Users/caiya/Desktop/Table 2-Distance.xlsx'    # Replace with the path to your Excel file

distance_matrix_df = pd.read_excel(file_path, index_col=0)

distance_matrix = distance_matrix_df.values

def nearest_neighbor_heuristic(matrix, starting_index):

    n = matrix.shape[0]    # Get the size of the distance matrix

    visited = set([starting_index])    # Start with the initial point as visited

    path = [starting_index]    # The path starts with the initial point

    total_distance = 0

    while len(visited) < n:

        last_visited = path[-1]

        nearest = np.argmin([matrix[last_visited][i] if i not in visited else np.inf for i in range(n)])

        path.append(nearest)

        total_distance += matrix[last_visited][nearest]

        visited.add(nearest)

    total_distance += matrix[path[-1]][path[0]]

    path.append(starting_index)

    return path, total_distance

starting_index = distance_matrix_df.index.tolist().index('Sakhir')    # Replace 'Sakhir' with the exact name from your distance matrix index

race_calendar, total_race_distance = nearest_neighbor_heuristic(distance_matrix, starting_index)

print("Race calendar:", race_calendar)

print("Total distance traveled:", total_race_distance)
```

The Ampl code of calculating integer programming in 6)

Tsp-6.mod code

```
param n := 22;
```

```
param locationX{1..n} := Uniform01();

param locationY{1..n} := Uniform01();

var loopVars{1..n};

param distances{i, j};

var travel{1..n, 1..n} binary;

#objective

minimize totalTravelDistance: sum {i in 1..n, j in 1..n: i <> j} distances[i, j] * travel[i, j];

# entering city

constraintEntry{k in 1..n}: sum{i in 1..n: i <> k} travel[i, k] = 1;

# leaving city

constraintExit{k in 1..n}: sum{i in 1..n: i <> k} travel[k, i] = 1;

# circuit

constraintNoSubtours{i in 1..n, j in 1..n: i > 1 and j > 1 and i <> j}: loopVars[i] - loopVars[j] + n * travel[i, j] <= n - 1;

constraintBoundLoopVar{i in 1..n: i > 1}: loopVars[i] <= n - 2;

option solver cplex;

solve;
```

Tsp-6.dat code

```
param n := 22; # Number of locations

param distances:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 :=

1 0 782 7526 5004 4228 7572 2497 2692 6375 2430 3205 2255 2884 2636 991 3931 8021 8693 7341 8043 70 277

2 782 0 7964 5774 5010 7211 2212 2367 6170 2228 2992 2105 2677 2360 1440 4569 7849 8435 6559 8107 826 1004

3 7526 7964 0 5051 5020 9690 9996 10204 10402 9867 10531 9660 10260 10120 8071 3764 8877 8428 8117 8159 7457 7255

4 5004 5774 5051 0 918 7596 5964 6135 6576 5719 5908 5550 5820 5977 4563 3129 6729 7208 11643 5708 4973 4839

5 4228 5010 5020 918 0 8232 5584 5780 7047 5351 5701 5159 5547 5628 3938 2366 7483 8029 11529 6517 4189 4035

6 7572 7211 9690 7596 8232 0 5078 4891 1401 5144 4377 5327 4695 4937 6845 10534 1098 1283 4099 2170 7640 7829

7 2497 2212 9996 5964 5584 5078 0 217 3959 247 791 425 499 148 1948 6262 5638 6248 5972 5960 2566 2761

8 2692 2367 10204 6135 5780 4891 217 0 3804 429 695 629 468 159 2164 6477 5483 6076 5783 5849 2761 2959

9 6375 6170 10402 6576 7047 1401 3959 3804 0 3970 3192 4129 3514 3812 5549 9199 1679 2319 5071 2244 6439 6608

10 2430 2228 9867 5719 5351 5144 247 429 3970 0 779 211 457 283 1796 6111 5643 6279 6210 5899 2498 2685
```

```
11 3205 2992 10531 5908 5701 4377 791 695 3192 779 0 951 322 646 2504 6774 4867 5500 5917 5170 3272 3455

12 2255 2105 9660 5550 5159 5327 425 629 4129 211 951 0 632 491 1589 5901 5795 6443 6397 6005 2322 2504

13 2884 2677 10260 5820 5547 4695 499 468 3514 457 322 632 0 366 2203 6496 5188 5822 6045 5468 2951 3135

14 2636 2360 10120 5977 5628 4937 148 159 3812 283 646 491 366 0 2059 6375 5491 6102 5937 5815 2705 2898

15 991 1440 8071 4563 3938 6845 1948 2164 5549 1796 2504 1589 2203 2059 0 4317 7139 7849 7591 7067 1032 1132

16 3931 4569 3764 3129 2366 10534 6262 6477 9199 6111 6774 5901 6496 6375 4317 0 9845 10323 9931 8836 3864 3655

17 8021 7849 8877 6729 7483 1098 5638 5483 1679 5643 4867 5795 5188 5491 7139 9845 0 747 5024 1093 8083 8239

18 8693 8435 8428 7208 8029 1283 6248 6076 2319 6279 5500 6443 5822 6102 7849 10323 747 0 4617 1512 8758 8927

19 7341 6559 8117 11643 11529 4099 5972 5783 5071 6210 5917 6397 6045 5937 7591 9931 5024 4617 0 6082 7384 7549

20 8043 8107 8159 5708 6517 2170 5960 5849 2244 5899 5170 6005 5468 5815 7067 8836 1093 1512 6082 0 8092 8198

21 70 826 7457 4973 4189 7640 2566 2761 6439 2498 3272 2322 2951 2705 1032 3864 8083 8758 7384 8092 0 209

22 277 1004 7255 4839 4035 7829 2761 2959 6608 2685 3455 2504 3135 2898 1132 3655 8239 8927 7549 8198 209 0;
```

The Ampl code of calculating extensive integer programming in 7)

Tsp-7.mod code

```
param n := 22;

param locationX{1..n} := Uniform01();

param locationY{1..n} := Uniform01();

var loopVars{1..n};

param distances{i, j};

var travel{1..n, 1..n} binary;

minimize totalTravelDistance: sum {i in 1..n, j in 1..n: i <> j} distances[i, j] * travel[i, j];

x_{9, j} = 0 for j in 1...8

sum(x_{16, j} for j in 1...7) + sum(x_{16, j} for j in 18...22) <= 1

x_{4, j} + x_{5, j+1} = 1 or x_{5, j} + x_{4, j+1} = 1 for j in 1...21

If sum(x_{8, j} for j in 1...11) = 1 then sum(x_{19, j} for j in 12...22) = 1

x_{1, 1} = 1

x_{22, 22} = 1

sum(x_{2, j} for j in 1...11) + sum(x_{21, j} for j in 1...11) = 2

constraintEntry{k in 1..n}: sum{i in 1..n: i <> k} travel[i, k] = 1;

constraintExit{k in 1..n}: sum{i in 1..n: i <> k} travel[k, i] = 1;
```

```
option solver cplex;

solve;
```