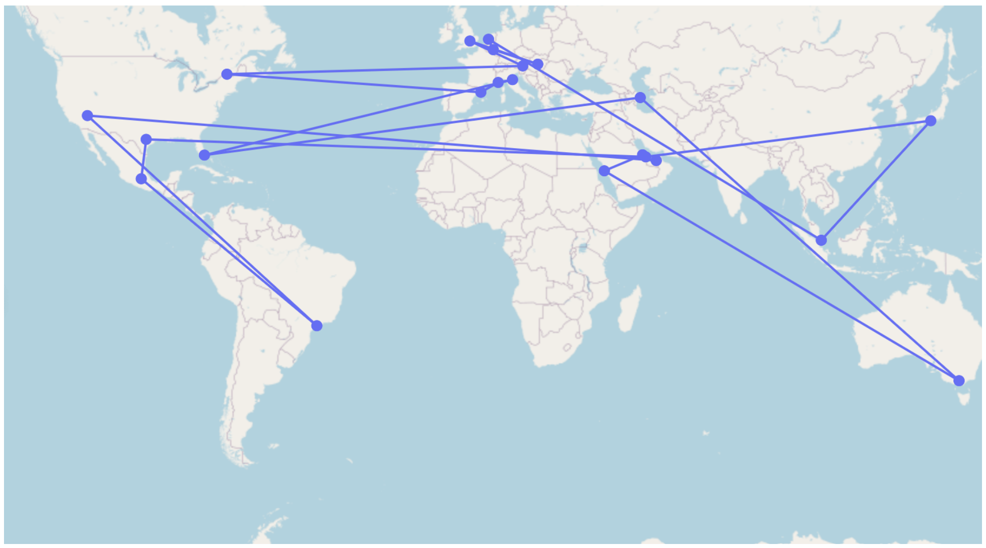
Kian Silva

Ross Parker

Calculus II

Formula One Calendar Optimized

Formula One Racing, a racing sport like no other, combines engineering feats and incredible driving ability. Each year a new car must be developed from scratch including everything from the engine to the brakes to the general shape and aerodynamics of the exterior. Every year further innovations and improvements to these beasts of a vehicle make the racing miles more entertaining. As the innovations come, so does the vehicle's capacity to be more eco-friendly transitioning from gas fuel to a battery-powered hybrid engine. Despite these advancements, Formula 1 still has a major flaw that creates more carbon emissions and costs from travel than there should be. This is the order in which they travel from city to city. The current Formula 1 calendar consists of traveling across the world starting in Bahrain, Saudi Arabia, Australia, Azerbaijan, Miami, Italy, Monaco, Spain, Canada, Austria, England, Hungary, Belgium, Netherlands, Italy, Singapore, Japan, Qatar, Texas, Mexico, Brazil, Las Vegas, Abu Dhabi. On a map, this looks like ridiculous back-and-forth travel and covers over one hundred sixty thousand kilometers of travel.



The travel is not only costly to the wallets of these multi-million dollar teams, but it makes the sport significantly more labor intensive. I looked at several optimization methods and decided to implement a Monte Carlo method known as simulated annealing to create the most efficient calendar to cut travel expenses. This method is utilizing the same methodology that is used in the classic Travel Salesman problem that looks to optimize the route a salesman should travel from city to city to sell the most with the least amount of travel. One of the more recent rules of Formula One is that every team has a budget cap to spend on whatever they may need throughout a calendar season. These expenditures spread from catering for the pit crews and admin to the engine repair costs and shipping. With cut travel expenses teams can allocate some of the funds previously designated for travel into the development of their vehicles making the competition closer and more exciting.

Simulated Annealing is based on the metallurgical practice of welding where a material is heated to a high temperature and then cooled. In this algorithm, a temperature is set to a number that is finely tuned to be cooled to a certain minimum temperature. This starting temperature is cooled to by a fraction, alpha, which cools the starting temperature to the minimum. With each temperature, the main optimization routine is utilized with the list. This optimization picks a random element (city) in the list and swaps its position in the list with another city in the list. Once swapped the list will calculate the total distance traveled between all the cities in the list and will create a random probability of $e^{(f(c)-f(n))}$ which is the distance of the current list minus the distance of the adjacent or swapped list. This optimization technique is put in place to prevent the list from accepting the local minima instead of the global minima. Though this technique runs the risk of potentially swapping the global minima with a worse solution to hope for a better solution in the end.

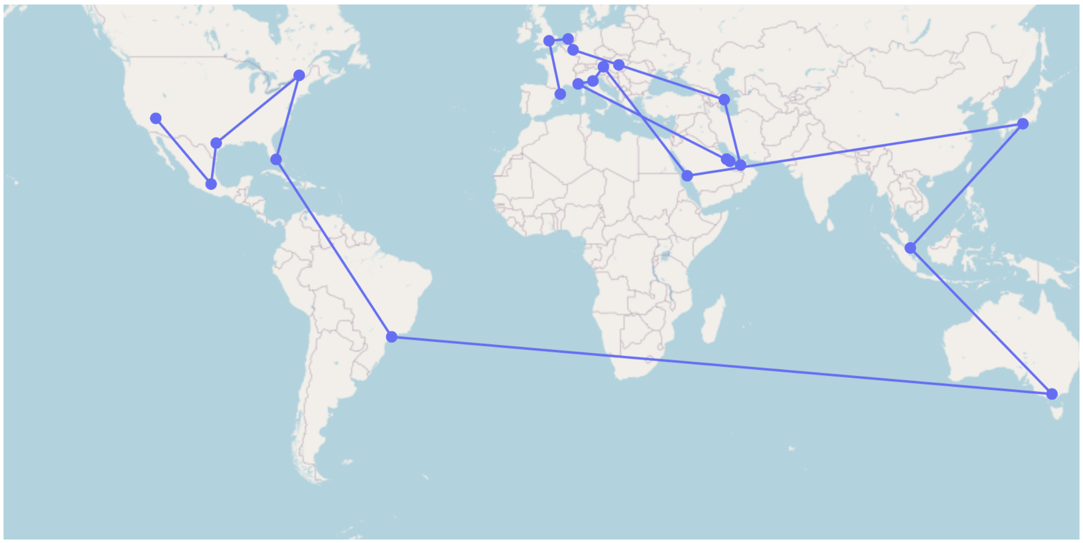
In the algorithm I have written for this problem, after every 250 iterations, the program displays the current error value which is the random probability that the current iteration is accepted or denied. This is displayed alongside the temperature cooling value. As simulated annealing goes, the temperature will start at a manually tuned value and cool by a factor of alpha each time the algorithm accepts the new list up until a certain point; for this algorithm, I chose 10^{-4}. To best calculate the distances between each city I set the distance values to be in Kilometers. Below are the displayed values when running the algorithm I developed.

```
Current F1 calendar map
Current total distance of F1 Calendar Map = 166062.8
max_iter = 2500
start_temperature = 10000.0
alpha = 0.99
iter =      0 | curr error = 166041.78 | temperature = 10000.0000
iter =    250 | curr error = 105778.14 | temperature =   810.5852
iter =    500 | curr error = 98486.10 | temperature =    65.7048
iter =    750 | curr error = 96865.95 | temperature =     5.3259
iter =   1000 | curr error = 87683.62 | temperature =     0.4317
iter =   1250 | curr error = 86416.90 | temperature =     0.0350
iter =   1500 | curr error = 86350.54 | temperature =     0.0028
iter =   1750 | curr error = 86350.54 | temperature =     0.0002
iter =   2000 | curr error = 86350.54 | temperature =     0.0001
iter =   2250 | curr error = 86350.54 | temperature =     0.0001

Best solution found:
Barcelona, Silverstone, Zandvoort, Spa Francorchamps, Budapest, Baku, Abu Dhabi, Lusail, Sakhir, Monte Carlo, Imola,
Spielberg, Jeddah, Suzuka, Singapore, Melbourne, Sao Paulo, Miami, Montreal, Austin, Mexico City, Las Vegas

Total distance = 86371.5
```

This example happens to be one of the better instances and options it returned. However, since simulated annealing commits to the trade-off of worse solutions at times, the algorithm can produce only one of the potential local minimum values. This means that at times the total distance of the new solution found can be upwards of a hundred thousand kilometers. Even though this is still shorter than the current route, it still is not the best solution. To compare on a visual scale how much better this proposed solution is proposed here is the best solution mapped.



This route cuts travel down by 79,691.3 kilometers which can make the formula one season significantly more profitable and easier to manage for the hundreds of workers for each of the 10 teams to set up and break down each pit. With most of the optimized routes, there becomes the option to transport this cargo via boat or truck which would also minimize the cost and emissions to transport the equipment. It is also important to note that the map connecting each city is not a representation of the travel between each city. In some representations, the algorithm will display a connection from Las Vegas to Tokyo by zooming across the Atlantic Ocean, which is the least likely route a plane might take to get between those two cities.

Below is the code used for this algorithm, the starting order of the list is the current order of the Formula One calendar.

```python
import numpy as np
from numpy import random
from geopy import distance
import plotly.graph_objects as go
```

```
circuit_list = [{"lon": 50.512, "lat": 26.031, "location": "Sakhir",
"name": "Bahrain International Circuit"},
                {"lon": 39.104, "lat": 21.632, "location": "Jeddah",
"name": "Jeddah Corniche Circuit"},
                {"lon": 144.970, "lat": -37.846, "location": "Melbourne",
"name": "Albert Park Circuit"},
                {"lon": 49.842, "lat": 40.369, "location": "Baku", "name":
"Baku City Circuit"},
                {"lon": -80.239, "lat": 25.958, "location": "Miami",
"name": "Miami International Autodrome"},
                {"lon": 11.713, "lat": 44.341, "location": "Imola", "name":
"Autodromo Enzo e Dino Ferrari"},
                {"lon": 7.429, "lat": 43.737, "location": "Monte Carlo",
"name": "Circuit de Monaco"},
                {"lon": 2.259, "lat": 41.569, "location": "Barcelona",
"name": "Circuit de Barcelona-Catalunya"},
                {"lon": -73.525, "lat": 45.506, "location": "Montreal",
"name": "Circuit Gilles-Villeneuve"},
                {"lon": 14.761, "lat": 47.223, "location": "Spielberg",
"name": "Red Bull Ring"},
                {"lon": -1.017, "lat": 52.072, "location": "Silverstone",
"name": "Silverstone Circuit"},
                {"lon": 19.250, "lat": 47.583, "location": "Budapest",
"name": "Hungaroring"},
                {"lon": 5.971, "lat": 50.436, "location": "Spa
Francorchamps", "name": "Circuit de Spa-Francorchamps"},
                {"lon": 4.541, "lat": 52.389, "location": "Zandvoort",
"name": "Circuit Zandvoort"},
                {"lon": 103.859, "lat": 1.291, "location": "Singapore",
"name": "Marina Bay Street Circuit"},
                {"lon": 136.534, "lat": 34.844, "location": "Suzuka",
"name": "Suzuka International Racing Course"},
                {"lon": 51.454, "lat": 25.49, "location": "Lusail", "name":
"Losail International Circuit"},
                {"lon": -97.633, "lat": 30.135, "location": "Austin",
"name": "Circuit of the Americas"},
                {"lon": -99.091, "lat": 19.402, "location": "Mexico City",
"name": "Autódromo Hermanos Rodríguez"},
                {"lon": -46.698, "lat": -23.702, "location": "Sao Paulo",
"name": "Autódromo José Carlos Pace - Interlagos"},
                {"lon": -115.168, "lat": 36.116, "location": "Las Vegas",
"name": "Las Vegas Street Circuit"},
                {"lon": 54.601, "lat": 24.471, "location": "Abu Dhabi",
"name": "Yas Marina Circuit"}
               ]




# compute total distance of travelled between circuits using the current
calendar route
def total_distance(circuit_list):
    d = 0.0 # in Km
    n = len(circuit_list)
    nxt = 0
    for i in range(n):
        if i == n -1:
            b = 0
        else:
            b = i + 1
        cordsA = (circuit_list[i]["lat"], circuit_list[i]["lon"])
        cordsB = (circuit_list[b]["lat"], circuit_list[b]["lon"])
        if cordsA < cordsB:
            diff =  distance.distance(cordsA, cordsB).km *1.0
            d += diff
            circuit_list[i]["distance"] = round(diff,3)
        else:
            diff = distance.distance(cordsB, cordsA).km *1.5
            d += diff
            circuit_list[i]["distance"] = round(diff,3)
    return round(d,3)

#calculate an error to find the most optimal distance
def error(circuit_list):
    n = len(circuit_list)
    d = total_distance(circuit_list)
    min_dist = n - 1
    return d - min_dist

#swaps two random indices from the first route making a new route
def swap(circuit_list):
    n = len(circuit_list)
    shuffle = np.copy(circuit_list)
    i, j = np.random.randint(n), np.random.randint(n)
    hold = shuffle[i]
    shuffle[i],shuffle[j] = shuffle[j], hold
    return shuffle
```

```
def printVal(value, circ):
    val = []
    for i in range(len(circ)):
        val.append(circ[i][value])
    return ", ".join(val)

def solve(circuit_list, max_iter, start_temp, alpha, rnd):
    curr_temp = start_temp
    circ = circuit_list
    err = error(circ)
    iteration = 0
    interval = (int)(max_iter / 10)
    p = rnd.random()

    while iteration < max_iter and err > 0.0:
        adj_circ = swap(circ)
        adj_err = error(adj_circ)

        if adj_err < err:
            circ, err = adj_circ, adj_err
        else:
            accept_p = np.exp((err-adj_err) / curr_temp)
            if p < accept_p:
                circ, err = adj_circ, adj_err

        if iteration % interval == 0:
            print("iter = %6d | curr error = %7.2f | temperature = %10.4f "
% (iteration, err, curr_temp))

        if curr_temp < 0.0001:
            curr_temp = 0.0001
        else:
            curr_temp = curr_temp * alpha
        iteration += 1
    return circ

def mapCirc(circ):
    lonC = []
    latC = []
    locC = []

    for i in range(len(circ)):
        lonC.append(circ[i]["lon"])
        latC.append(circ[i]["lat"])
        locC.append(circ[i]["location"])

    fig = go.Figure(go.Scattermapbox(
        mode = "markers+lines",
        lon = lonC,
        lat = latC,
        marker = {'size': 10}))

    fig.update_layout(
        margin ={'l':0,'t':0,'b':0,'r':0},
        mapbox = {
            'center': {'lon': 10, 'lat': 10},
            'style': "open-street-map",
            'center': {'lon': -20, 'lat': -20},
            'zoom': 1 })
    fig.update_traces(
        text = locC)
    fig.show()

def main():
    mapCirc(circuit_list)
    print("Current F1 calendar map")
    circuit_dist = total_distance(circuit_list)
    print("Current total distance of F1 Calendar Map = %0.1f" %
circuit_dist)
    rnd = np.random.RandomState(4)
    max_iter = 2500
    start_temperature = 10000.0
    alpha = 0.99
    print("max_iter = %d " % max_iter)
    print("start_temperature = %0.1f " % start_temperature)
    print("alpha = %0.2f " % alpha)
    best = solve(circuit_list, max_iter, start_temperature, alpha,rnd)
    print("\nBest solution found: ")
    bestLoc = printVal('location', best)
    print(bestLoc)
    dist = total_distance(best)
    print("\nTotal distance = %0.1f " % dist)
    mapCirc(best)
```

```python
if __name__ == "__main__":
    main()
```