Using Simulated Annealing (SA) and Genetic Algorithm (GA) to minimize travel distance for the F1 season 2023

Atif
Griffith College, Limerick

October 2023

1 Introduction

NOTE: read the whole assignment brief first before implementing it contains very important information

In this assignment you will be tasked with doing four things:

- Create a simulation of the 2023 F1 calendar and determine the amount of travel required for teams to cover getting to and returning from those races. You will also need to simulate average temperatures for each of the locations for every month of the year. The objective of the simulation is to reduce overall travel distance for all teams while keeping the temperature for each race between 20 and 35 degrees Celcius
- A mapping onto the simanneal library that will use Simulated Annealing algorithm to determine the shortest distance for teams near Silverstone and teams near Monza using the current calendar. You will also allow a version where the dates are allowed to move except the starting race Bahrain, ending race Abu Dhabi, and Monaco
- A mapping onto the deap library that will use genetic evolution to determine the best free calendar similar to the task above.
- Determining which of the strategies produced is the best itinerary.

In terms of effort the vast majority of your time will be spent on the first part creating the simulation. For your benefit a number of unit tests have been provided for you for implementing the simulation. Your simulation will be correct when you have produced code that passes all unit tests. It is a way for you to self verify your progress on the assignment. It would be suggested that you start with the first of the unit tests and work your way down as the later tests will depend on correct functionality from code passing previous tests. The mappings to the libraries will take a bit of time as you will start with direct mapping to begin with but the free calendar cases will be more complex to implement. Once you have the free calendar cases implemented then the time to run the cases will not take long.

Read the details of the simulation carefully and if you have any questions do not hesitate to ask. There are very detailed notes here as this is a problem you've unlikely dealt with or implemented before. Don't attempt to do the simulated annealing or genetic algorithms until your simulation passes all the unit tests.

2 Simulation Details

The reason why this is an interesting topic is that F1 (and motorsports as a whole) is trying to become more sustainable overall. One of the ways F1 in particular is looking to reduce CO2 output is to optimise the race calendar such that teams will travel as short a distance as possible while still maintaining good weather conditions for racing to take place. Thus there are two main constraints we will model and optimise in this problem:

1. Reducing the overall distance travelled to and from races. This is to reduce fatigue on team personnel. Also most race freight is done by air rather than sea or land to far away races. We will assume air travel for everything so distances will be as the crow flies.
2. Keeping race temperatures between 20 and 35 degrees Celsius. This seems to be the conditions F1 has favoured over the last 30 years between not being too cold for the tyres to work and being too hot such that cockpit temperatures get too high and lead to driver heat exhaustion.

In terms of teams there are 10 in total with 8 having their nearest track as Silverstone and the other two having Monza as their nearest. We will assume these will be the home locations for these teams. There are a number of assumptions we will make about the simulation that will be factored into the code.

- Teams will travel directly to a track on a given race weekend.
- If the next weekend after a race is not a race weekend the teams will travel home after a race
- if the next weekend after a race is a race weekend the teams will travel directly to the next track from the location of the last race.
- It is possible that there can be two races or three races in a row without a break in between (we refer to these as Double Header and Triple Header events) but never four or more.
- If a race weekend (Friday to Sunday) overlaps two months the Sunday will decide what month it is classed as
- The dates and locations for the opening race (Bahrain), closing race (Abu Dhabi), and Monaco will not change

In terms of distance the unit tests will give you an overall distance for teams from Silverstone of approx 185,000Km and the teams from Monza of 179,000Km. When you apply both Simulated Annealing and Genetic Algorithms to this you should be able to get race calendars between 140,000Km and 162,000 Km

3 f1-calendar-base.py

Discussion of what is contained in the python file you have been provided. There are also three support files given as well which contains all of the data you need for the simulations: race-weekends.csv, sundays.csv, and track-locations.csv.

3.1 UnitTests

A series of unit tests designed to help you get the details of the simulation correct. It is suggested with this that you work your way from the top of the tests downwards. Some of the later tests will rely on previous functionality being correct in order for them to work. testReadCSV is already working for you so start from the second test onwards. Once you have all of the tests passing then your simulation is correct and is replicating the solution I have. You can then proceed onto implementing the Annealer and Genetic Algorithms cases.

You are not permitted to modify the unit tests

3.2 functions

Will discuss the function stubs that you have to write to implement the simulation and pass the tests

3.2.1 calculateSeasonDistance

This function is responsible for calculating the total distance travelled by a team through out the season using the assumptions listed in the Simulation Details. This takes in three parameters.

- tracks: the track information as read in from track-locations.csv
- weekends: a list that defines on which weekends of the year each of the given races will run
- home: determines which track will be the home track that a team will start off at and will travel to and from when going to and from races

This should iterate through every week of the year and determine if any travelling is to be done. If travel is to be done it should add the distance of that travel to an overall running total and return that result at the end. To calculate travel distances this should use the **haversine** function

3.2.2 checkFourRaceInARow

Function that takes in a single parameter weekends representing the weekends that each race runs on. This should return true if anywhere in the calendar there are four races on successive weekends. It should return false otherwise.

3.2.3 checkTemperatureConstraint

Function that takes in three parameters:

- tracks: the track information as read in from track-locations.csv

- weekends: a list that defines on which weekends of the year each of the given races will run
- sundays: a list denoting which month of the year each sunday falls in. 0 is January while 11 is December

This will go through each of the race weekends and check the temperature that the race can expect based on the month the Sunday of the race weekends falls in. If a single race has an expected temperature outside the range of [20, 35] degrees Celsius False should be returned. If no race falls outside the range return True

3.2.4 checkSummerShutdown

Function that takes in a single parameter weekends representing the weekends that each race runs on. This should return true if there are three successive weekends in any period at the start of July until the end of August where no race takes place. The summer shutdown is a requirement in every calendar at this time of the year.

3.2.5 convertColToFloat

Function that takes in two parameters

- rows: A 2D list containing the set of data extracted returned from the **readCSV** function
- column_index: the index of the column whos strings you wish to convert to floating point numbers

A function that will take a set of data read in from CSV and will take the given column and convert and replace its values with floating point representations of themselves. We need this as by default no conversion is done when a CSV file is read. All values will have a type of String by default.

3.2.6 convertColToInt

Does a similar job to **convertColToFloat** but convert to integers instead

3.2.7 haversine

Function that takes in three parameters

- rows: should be the information read in from track-locations.csv
- location1: index into rows representing the starting point of our travel
- location2: index into rows representing the ending point of our travel

The haversine formula is used to calculate the distance needed to travel along the surface of a sphere between two locations both denoted by a latitude and longitude. This will return a distance in kilometers.

The formula for the haversine is the following note that lat1, lat2, long1, and long2 have been converted from degrees to radians for this formula

$haversine = 2 * arcsin(sin(\frac{lat2-lat1}{2})^2 + cos(lat1) * cos(lat2) * sin(\frac{long2-long1}{2})^2) * 6371$

6371 is the radius of the earth in kilometres

3.2.8 printItinerary

prints out the calendar for the year. It should state what happens each weekend based on the assumptions in Simulation Details. on each line it will state one of these four things:

- In the case where the team is currently home and the next weekend has no race: Staying at home thus no travel this weekend
- In the case where the team is currently at home and the next weekend has a race: Travelling from home to ⟨*location*⟩. Race temperature is expected to be ⟨*temp*⟩ degrees
- in the case where the team is currently at a race but the next weekend has a race: Travelling directly from ⟨*currentlocation*⟩ to ⟨*location*⟩. Race temperature is expected to be ⟨*temp*⟩ degrees
- in the case where the team is currently at a race but there is no race the following weekend: Travelling home from ⟨*location*⟩

This function will take in four parameters

- tracks: the track information as read in from track-locations.csv
- weekends: the weekends that the races will fall on
- home: an index into tracks denoting which track is the home track for a team
- sundays: a list denoting which month each sunday falls in

3.2.9 readRaceWeekends

Used to read and parse the 2023 race calender from race-weekends.csv

3.2.10 readSundays

Used to read the map of sundays to months from sundays.csv

3.2.11 readTrackLocations

Used to read track information from track-locations.csv. This should be parsed as well

3.2.12 SACases

This function should be used to setup and trigger your simulated annealing cases. It should report the calendar and total distance calculated from each of the cases

3.2.13 GACases

This function should be used to trigger your genetic algorithms case. It should report the best calendar and total distance calculated for the one case used here

4 Suggestion for how you should proceed and other relevant information

The recommendation here would be to first write the code to pass all of the unit tests. The unit tests represent a working implementation of the simulation. If you have passed the unit tests then you are replicating the simulation I've written and you will produce results consistent with mine.

Once you have this done move onto implementing the first two cases of Simulated Annealing. This involves using the same weekends as the 2023 calendar as read in from race-weekends.csv and swapping the races around to try and minimise the distance that is travelled to and from races. However for this to be a valid solution the temperature constraint must be satisfied at all times. Also Bahrain (week 9), Monaco (week 21) and Abu Dhabi (week 47) cannot be swapped and must remain where they are. The first case will come up with a calendar that is best suited to teams with Silverstone as their home track. The second will be a calendar that will suit teams with Monza as their home track. Both of these cases should use 200,000 steps for Simulated Annealing

The last case for simulated annealing is a relatively free calendar where race weekends can move around along with swapping two races. The following conditions must be observed for this:

- the temperature constraint is satisfied for all races
- there are no more than three races in a row during the season
- all 22 races occur during the season
- there are no repeating races
- the summer shutdown on weekends 31, 32, and 33 is observed
- Bahrain, Abu Dhabi, and Monaco must take place on weekends 9, 21, and 47 respectively
- Bahrain opens the season, Abu Dhabi closes it

Every time you go to make a move you have one of two choices (50/50 chance):

- you swap two of the races around

- you pick a race and move it forward or back by a week (50/50 chance). You may only move forward or back by a week so long as there is not another race on that weekend

For this case you will need to use 400,000 steps for annealing as there will be a period where the default answer of 1,000,000 will be returned while it finds a valid case. it will then start searching the space for better solutions.

The genetic algorithms case should implement the last of the simulated annealing cases but using a population of 300 instances. Crossover should be based on the roulette wheel method while mutation should look at moving individual races back or forward by a weekend so long as there is not another race that weekend. You should use 1,000 generations for this as like simulated annealing there will be an initial period of time before GA finds a first valid solution.

5 Notes

You are required to submit this assignment by 2023-12-17 (Sunday 17th of December) by 23:55. You are required to submit two separate components to the Moodle

- A single python file containing all of your code.
- A PDF containing documentation of your experiments along with discussions of your cases **If you do not provide documentation your code will not be marked.**

There are also a few penalties you should be aware of

- Code that fails to compile will incur a 30% penalty before grading. At this stage you have zero excuse to produce non compiling code. I should be able to open your project and be able to compile and run without having to fix syntax errors.
- The use of libraries outside the python SDK and permitted libraries (simanneal, deap, sklearn, matplotlib) will incur a 20% penalty before grading. You have all you need in the standard SDK. I shouldn't have to figure out how to install and use an external library to get your app to work
- The standard late penalties will also apply

You should be aware that I will remove marks for the presence of bugs anywhere in the code and this will incur a deduction of between 1% and 15% depending on the severity. If you have enough of these bugs it is entirely possible that you may not score very many marks overall. Particularly for the simulation this should be avoidable as you've been given unit tests to verify your code works.

Also note that the percentage listed after each task is the maximum mark you can obtain if you complete that many brackets without error.

6 Tasks

1. Write the simulation using the detail above and test it works by by getting it to pass the unit tests. NB. You cannot modify the code of the unit tests (50%)
2. Map the simulation so it can run the simulated annealing cases (70%)
3. Map the simulation so it can run using genetic algorithms to run the last case (80%)
4. Write documentation on each of the four cases you have run and the best results you have achieved in each. You should discuss any interesting effects they produce (e.g. did they use lots of double and triple headers, did it avoid them, are there any patterns for how it groups races etc), This should be no more than 1,000 words total (100%)