F1 STRATEGIST

Dynamic Programming

Figure 1: Toto Wolff, Team Principal and CEO of Mercedes

Diego Toledo Luque
Analysis and Design of Algorithms
Double Degree in Mathematics and Computer Science
Universidad de Málaga

12/11/2023

Contents

1 The Problem

You are the strategist of a Formula 1 racing team. You need to decide the strategy for an upcoming race. After talking to the data analysts and track engineers, you gather some important information about the behaviour of your team cars in the next circuit:

- The time to complete a lap is $\frac{L}{v}$, where L is the lap length and v is the average speed during that lap. The maximum speed attainable in each moment will depend on two factors: the weight of the car (which will decrease during the race due to fuel consumption) and the degradation of the tyres (which will increase on each lap). Using the model built by the engineers, let $s(f, d)$ be the maximum speed at which a lap can be taken with f kg of fuel in the tank and the tyres having a degradation d (in %) at the beginning of said lap.

- The speed and weight of the car will determine fuel consumption. Let $g(v, f) \geq 0$ be the fuel consumption in a lap at speed v with f kg of fuel at the beginning of said lap.

- The tyre degradation depends on speed, weight, and current state of the tyres. Let $h(v, f, d)$ be the degradation state of the tyres after a lap at speed v with f kg of fuel and tyres with degradation d at the beginning of said lap.

The scenario in which you are working is the following: the car has just completed the last pit stop and N laps are remaining. Therefore, you have to determine at which speed the car should be driven on each lap, so as to minimize the total time needed to complete the race. Note that at the beginning of this scenario the car has F kg of fuel left in the tanks, and the tyres are brand new ($d = 0$). Of course, if the car runs out of fuel ($f = 0$) or the tyres get completely degraded ($d = 100$) before the N laps are completed, the car will have to stop (i.e., you can assume $s(0, d) = s(f, 100) = 0$, for all f and d) and it will be a failure.

Figure 2: Fernando Alonso 2023

2 Dynamic Programming

2.1 What is the solution?

We want to look for the speed the car should have in each lap in order to minimize the total time needed to complete the race. So the solution will be a list $S = < s_1, s_2, ..., s_i, ..., s_n >$ where s_i represents the speed in the lap i-th since last pit stop.

The solution will have some constrains such us that:
- $\sum_{i=1}^{n} g(s_i, f_i) \leq F$ where F is the total amount of fuel available at the beginning.
- $s_i \leq s(f_i, d_i) \quad \forall\ i \in \{1, ..., n\}$
- $h(v, f, d) > 0$ for all v, f and d.

For future use, let's denote as s_m the maximum speed of the car.

2.2 Objective Functon

We want to define a function $f : S \rightarrow \mathbb{R}$ being S the set of all possibles solutions. As we want to end the race in the lowest of time ammount possible, we can define f as:

$$f(S) = f(< s_1, ..., s_n >) = L \sum_{i=1}^{n} \frac{1}{s_i}$$

Therefore, f give us the time to finish a race given the speeds for each laps. That means we want to minimize the function.

2.3 Decisions

In order to decide what speed should the car have in each lap, we have to take decisions. The decisions can be seen as the answer to the following question:

- What speed should the car have in the next lap?

That means, we will have to choose between multiple decision (multiway decisions).

2.4 Optimal Substructure

In order to use Dynamic Programming, the problem must have an optimal substructure. So we have to proof that.

Proposition 2.1. The problem has an optimal substructure.

Proof.
Let be $S^* =< s_1^*, s_2^*, ..., s_n^* >$ an optimal solution to the problem. As S^* is an optimal solution, $f(S^*)$ is minimal.

Let's proof it by contradiction. Let's take $S_{2\rightarrow n}^* =< s_2^*, ..., s_n^* >$ the solution to the problem of having $n-1$ laps left, with $F - g(s_1^*, F) = f_1$ fuel and a degradation of $h(s_1^*, F, d) = d_1$. Let $S'_{2\rightarrow n} =< s'_2, ..., s'_n >$ be the optimal solution to the subproblem.

That implies, $f(S'_{2\to n}) < f(S^*_{2\to n})$, as $S'_{2\to n}$ is the optimal solution to the subproblem.

It is easy to see that $S'' =< s^*_1, s'_2, ..., s'_n >$ is a solution to the original problem. Therefore, we have that:

$$f(S'') = f(S'_{2\to n}) + \tfrac{L}{s^*_1} < f(S^*_{2\to n}) + \tfrac{L}{s^*_1} = f(S^*)$$

Which is a contradiction because S^* is the optimal solution. Moreover, we have prooved that $S^*_{2\to n}$ is optimal to the subproblem, so following the Bellmam's principle of optimality the problem has optimal substructure.

$\square$

2.5 Subproblems

We can define the subproblems as:

$T_{n,f,d}$ = Minimum time having n laps left, f kg of fuel and a tyre degradation of d %.

2.6 Bellman's Equation

$$T_{n,f,d} = \begin{cases} \infty & \text{if } f = 0 \text{ or } d = 100 \\ \frac{L}{s(f,d)} & \text{if } n = 1 \\ \min\limits_{0\le s\le s(f,d)} (T_{n-1,f-g(s,f),h(s,f,d)} + \frac{L}{s}) & \text{if } n > 1 \end{cases}$$

We take the minimum because we want to finish the race in the least time possible.

2.7 Data Structures

For storing the data of the Bellman's Equation we will need to use auxiliary data strcutures for storing the results. We will use a tridimensional table of order $N \times F \times d$.

As we cannot draw it in 2 dimensions, we are going to represent it using one bidimensional tables for showing the dependencies of $T_{n,f,d}$, which looking to the formula, is easy to see that it depends on what will be the time on the following lap:

$$\min_{0\le s\le s(f,d)} (T_{n-1,f-g(s,f),h(s,f,d)} + \tfrac{L}{s})$$

Moreover we will need to use another tridimensional table to store the optimal speed in each lap, which is really what we want to find (not the time needed to finish the race).

We can not represent the tridimensional table here, but we can represent the dependencies of an arbitrary $T_{n,f,d}$ (See Figure 3):

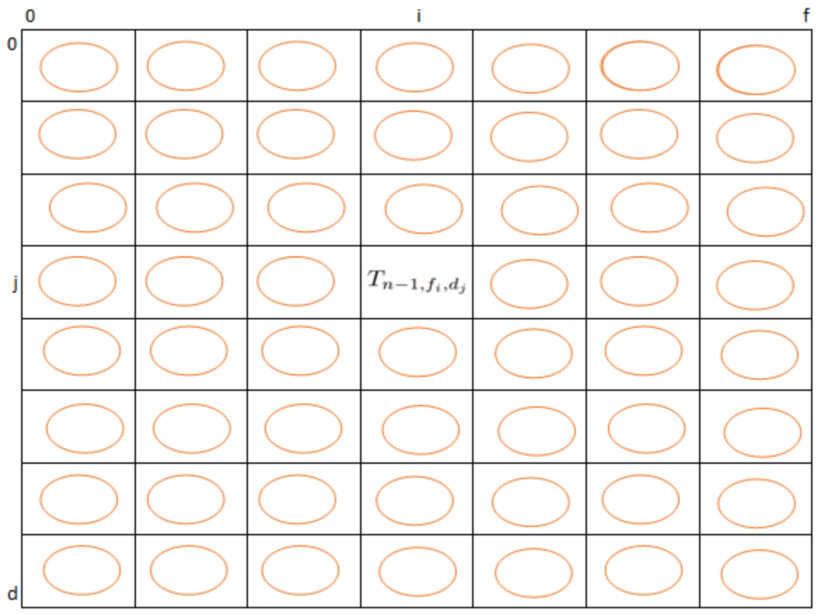

Figure 3: Dependencies of an arbitrary $T_{n,f,d}$

We can conclude that space complexity is $\Theta(NFd)$ being N the number of laps, F the number of fuel available at the beginning and d the degradation of the wheels in %.
Also, we can conclude that time complexity will be $\Theta(NFds_m)$ because of the Bellman's Equation.

Another important aspect is how the table will be filled. It is easy to notice, seeing dependencies, that the table must be filled from less laps remaining to more laps remaining.

2.8 Algorithm

We will use Java as programming language, if you want to take a look into the source code you can check it here `https://github.com/DitoluT/F1_Strategist`.

First of all, we have to compute the 3 base cases of Bellman's Equation, when we ran out of fuel, when the tyres have a degradation of 100% and when there is just one lap left.

Once the base cases are done, we have to fill the general cases. For doing that we implement the Bellman's Equation.

To sum up, we reconstruct the solution into a linear array which contains optimal speed for each lap.

Algorithm 1 Optimal Speed Calculation

```
function OptimalSpeed(laps, initial_fuel, initial_degradation, lap_length)
    Time ← initialize 3D array of doubles
    Speed ← initialize 3D array of integers
                                                ▷ First Step: Filling base cases
    for $j = 1$ to initial_fuel do
        for $k =$ initial_degradation to 99 do
            max_speed ← maxSpeed($j, k$)
            Time[0][$j$][$k$] ← $\frac{\text{lap\_length}}{\text{max\_speed}}$
            Speed[0][$j$][$k$] ← max_speed
        end for
    end for
    for $i = 0$ to laps do
        for $k =$ initial_degradation to 99 do
            Time[$i$][0][$k$] ← $\infty$
            Speed[$i$][0][$k$] ← 0
        end for
    end for
    for $i = 0$ to laps do
        for $j = 0$ to initial_fuel do
            Time[$i$][$j$][100] ← $\infty$
            Speed[$i$][$j$][100] ← 0
        end for
    end for
                                                ▷ Second Step: Bellman's Equation
    for $i = 1$ to laps do
        for $j = 1$ to initial_fuel do
            for $k =$ initial_degradation to 99 do
                min_time ← $\infty$
                opt_speed ← 0
                for $s = 1$ to maxSpeed($j, k$) do
                    time ← Time[$i-1$][$j-$fuelConsumption($s, j$)][tyreDegradation($s, j, k$)]+ $\frac{\text{lap\_length}}{s}$
                    if time < min_time then
                        min_time ← time
                        opt_speed ← $s$
                    end if
                end for
                Time[$i$][$j$][$k$] ← min_time
                Speed[$i$][$j$][$k$] ← opt_speed
            end for
        end for
    end for
                                                ▷ Third Step: Reconstruct Solution
    optimalSpeed ← array of integers with length laps
    optSpeed ← Speed[laps − 1][initial_fuel][initial_degradation]
    fuel ← initial_fuel
    degradation ← initial_degradation
    optimalSpeed[0] ← optSpeed
    for $i = 1$ to laps − 1 do
        degradation ← tyreDegradation(optSpeed, fuel, degradation)
        fuel ← fuel − fuelConsumption(optSpeed, fuel)
        optSpeed ← Speed[laps − $i$ − 1][fuel][degradation]
        optimalSpeed[laps − $i$ − 1] ← optSpeed
    end for
    return optimalSpeed
end function
```

3 Conclusion

To sum up, we can take a look into the complexity $\Theta(NFds_m)$. It may seem to be inefficient, but notice that normally those parameters have an upper bound. By taking integer values, normally a f1 car do not surpase 400 km/h, the tyre degradation can not be larger than 100%, following FIA rules a f1 car cannot have more than 110kg of fuel and the race with the most laps is Monaco GP with 78 laps. That means that even though the complexity may seem of order $\approx \Theta(n^4)$ it will never grow to infinity.

Note that in this document we have supose that the statiscal model to calculate degradation and fuel consumption have constant complexity $\Theta(1)$.

Note that this algorithm only work with discrete values, it is not capable of working with continuous values. However you can add decimals to the values by increasing the tridimensional table size, this will give a more accurate solution, but will have an impact on the algorithm making it slower.

Note that the algorithm gives the average speed in a lap, it will not tell you the exact speed for each curve or straight sector. Furthermore you can divide each lap in sections (or sectors), with their respective fuel consumpution and degradation statistical models, such that the algorithm can give you the optimal speed for each sector, as well as previously it will give more accurate values, with a cost in algorithm speed.