Real Time Tyre Strategy Decision in Formula 1 Using the A* Algorithm

Samuel Gerrard Hamonangan Girsang- 13523064
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: gerrardgrs@gmail.com , 13523064@std.stei.itb.ac.id

***Abstract*—In Formula 1 racing, tyre strategy plays a critical role in determining race outcomes, particularly under dynamic conditions such as changing weather, safety car deployment, and tire degradation. This paper presents an algorithmic approach to real-time pit stop decision-making using the A search algorithm. The race is modeled as a series of state transitions, where each state represents a specific lap, tire compound, wear level, and track condition. The A algorithm is applied to evaluate and select the optimal sequence of pit stops and tire choices to minimize total race time. The cost function considers lap time degradation due to tire wear, while the heuristic estimates future lap times based on compound performance. Simulation experiments are conducted on a simplified 50-lap dry race scenario. Results demonstrate that the algorithm adapts dynamically, selecting pit stops based on performance drop-offs and trade-offs between tire types. This study shows that search-based algorithmic strategies can support intelligent, adaptive decision-making in competitive motorsport environments.**

***Keywords*—tyre degradation; pit strategy; A* Algorithm**

I. Introduction

In Formula 1 (F1), strategy is very crucial for teams to determine the outcome of the race. One of the most important decision to make during a race is when to pit and which tyre compound to use during a certain condition on the race. These choices are not only influenced by initial race plans but also by dynamic, unpredictable factors such as tyre wear, weather changes, safety car deployments, and the strategies of competing teams.

Usually, F1 teams already determine their strategy by data simulation and tests. However, in the uncertainty nature of a race, decision-making must be both fast and adaptive. To solve this problem, we can use algorithmic approaches and use computational power to determine which decision is best. One of the algorithm is A* Search Algorithm, which is widely used in pathfinding and decision-making problems due to its ability to find the shortest or most optimal path in a weighted search space.

This paper explores the use of the A* algorithm to model real-time tyre strategy decisions in Formula 1. The race is stated as decision state, each representing a specific lap, tyre type, tyre wear level, and track condition. Then we can determine the cost by using a heuristic between transitions. The transition is the decision changes made from one decision state to another, which one would minimize the total race time of the team.

By simulating a simplified race scenario, this study can show how an algorithmic approach can help find the best tyre decision strategy during a Formula 1 Grand Prix.

II. Theoretical Basis

A. Heuristics

In the context of search algorithms, a heuristic is a function that estimates the cost value of reaching a goal from a given state. It's a strategy used in algorithms using accessible and applicable information from the problem. The result of a heuristic function from applying those said accessible information provides direction for the algorithm by suggesting which paths appear more promising. Heuristic is not guaranteed to give a definitive answer to a problem, but a good enough heuristic can significantly improve the efficiency of a search algorithm by the number of states explored. A heuristic function represents compromises between two requirements, that is to make the criteria for reaching a goal is simple and, at the same time, the need of them to discriminate correctly between which is a good choice and which is a bad choice.

Heuristic-based search algorithm is categorized as informed algorithm, since it already has accessible information of where the goal node is, in contrast to the uninformed search algorithms like Breadth-First Search (BFS) or Depth-First Search (DFS).

Heuristic function is widely used with the term:

$$h(n) = estimated\ cost\ from\ a\ node\ to\ another\ node$$

There are two characteristics of a heuristic that must be considered, that is admissible and consistent. A heuristic being admissible means that it never overestimates the actual cost to reach a goal. This characteristic ensures that the heuristic function is more optimal to use in algorithms, since it already has a lower cost than the real cost. A heuristic being consistent means that for every node N and their successor as S, the estimated cost of reaching the goal from N is no greater than the cost of getting to S from N plus the estimated cost of reaching the goal from S, which stated in this expression:

$$h(N) \leq c(N, S) + h(S)$$

where:

- h: heuristic function
- N: any node in the graph
- P: descendant of the node N
- c(N,S) is the estimated cost of reaching node S from N

This characteristic ensures that the estimated cost of every node is always less than or equal to the estimated cost of the next node, also in other words, the estimated cost always decreases as the search gets closer to the goal.

In heuristic applications, we must first know what the goals of our function are. In general, there are three types of goals, that is optimizing, satisficing, and semi-optimizing. Determining which one of the goal types we aim will help us in determining how accurate our heuristic needs to be and how much computation we're willing to trade for quality. Optimizing aims to find the best possible solution according to a defined metric, it's used when optimality matters, like path finding. Satisficing goal aims to find a solution that is good enough or meets a minimum requirement, this type of goal is used when speed matters the most than the optimal solution. Semi-Optimizing goal is when the acceptance criteria tolerate a neighboring solution about the optimal solution. It falls to two categories, if the boundaries of acceptance are defined (requires the solution to be within a specified factor of the optimal solution), it's called near-optimization. If we let loose the acceptance criteria and the solution be near optimal with high probability, it's called approximation-optimization.

Heuristic function can be used in a wide range of problems. That's why to help in the estimation calculation, we must represent the problem in a much simpler state, we call this a Problem Representation. Problem representation is the process of defining a problem in a formal, structured way so that a computer (or algorithm) can understand, manipulate, and solve it. For example, in the 8 Queens Problem, we have to put 8 queens in a 8x8 chess board without those queens aligning horizontally, vertically, and diagonally.

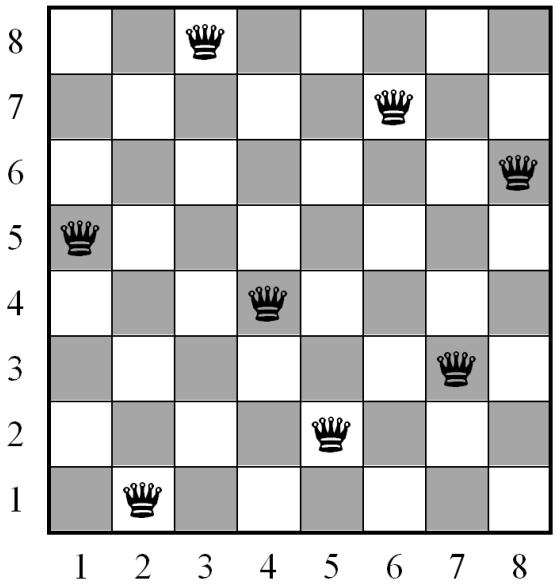


Figure 2.1 The 8 Queens Problem

Source: https://www.researchgate.net/figure/a-A-solution-to-the-non-attacking-8-queens-problem-b-An-optimal-solution-to-the_fig1_278681097

This problem can be represented as a bunch of states, it consists of the current state, the initial state, and the goal state. We can also provide the operators that we can do with the state and also constraints of the state. In this problem, we start with the empty board state, then we can use our heuristic function to determine the operator to do in the board, such as adding pieces on the certain tiles of the board, keeping in mind of the constraint that no two queens can be in the same row, same column, or same diagonal tiles.. The goal state is what is shown in the figure. To solve any problem using heuristic approach, we must represent each problem similarly to this 8 Queens Problerm to simplify the thought process of the heuristic function.

B. A Search Algorithm*

The A* Search Algorithm is a widely used algorithm to solve path finding or decision-making problems, mainly in problems that require finding the optimal path on a weighted graph. The A* Search Algorithm is an informed search algorithm, also known as heuristic search algorithms, that uses additional information which is called as heuristic functions. It tries to find the lowest cost path from a start node to a goal node in a graph, where each edge has associated cost. The idea of A* Search Algorithm is to avoid expanding paths that are already expensive

The A* Search Algorithm has the combination of other two path-finding algorithm, that is Uniform Cost Search (UCS) and Greedy Best First Search (GBFS). Uniform Cost Search uses the idea of always expanding the node with the lowest total path cost ,widely used as the term g(n), from the start node, regardless of the direction of depth which is like Dijkstra's Algorithm since UCS is a variant of it. Greedy Best First Search in the other hand, only uses the value of the heuristic function estimation (h(n)), it does not consider the cost to reach the node. A* Search Algorithm combined both ideas to create a cost estimation of:

$$f(n) = g(n) + h(n)$$

where:

- f(n): the total estimated cost from one node to another
- g(n): the cost to reach a node from one node
- h(n): the heuristic estimation from one node to another

This combination of idea allows A* Search Algorithm to explore the cheapest known path (g(n)) and prioritize promising nodes based on the estimated heuristic cost (h(n)). For A* Search Algorithm able to reach its optimal solution, then the heuristic function used to measure the total estimated cost needs to fulfill the characteristics of heuristic, being admissible and consistent.

In computing A* Search Algorithm, usually the process of searching is represented in a tree, with the start position acts as the root and every possible move will be its children. A* Search Algorithm also uses priority queue to determine which node will be expanded first. The estimated total cost for each node will be

the elements of the queue and for each iteration it selects the node with the lowest total estimated cost, expanding the node by generating life nodes which is the children the process it accordingly. But A* Search Algorithm allows the process to return to the previous depth of the tree if the successor of the expanded node has higher total estimated cost than the node on the previous depth of the tree.

To put it simply, we can see the process of A* Search Algorithm from the simple pseudocode provided:

Algorithm 1: A* Search Algorithm

```
function A_STAR(start, goal):
   open_set ← priority queue ordered by
f(n)
    open_set.insert(start)
    g_score[start] ← 0
    f_score[start] ← g_score[start] +
heuristic(start, goal)
    while open_set is not empty:
        current ← node in open_set with
lowest f_score
        if current is goal:
         return reconstruct_path(current)
        open_set.remove(current)
        for each neighbor of current:
           tentative_g ← g_score[current]
+ cost(current, neighbor)
           if tentative_g <
g_score[neighbor] or neighbor not in
open_set:
               g_score[neighbor] ←
tentative_g
               f_score[neighbor] ←
g_score[neighbor] + heuristic(neighbor,
goal)
               if neighbor not in
open_set:
open_set.insert(neighbor)
    return failure
```

This algorithm has the complexity of $O(b^m)$ where b is the average children of a node and m is the number of nodes on the resulting path.

C. Formula 1

Formula 1 (F1) is a single-seater motorsport, held by Fédération Internationale de l'Automobile (FIA). It's the pinnacle of motosport and automotive engineering, driver performance, and strategic decision-making.

Each Formula 1 season consists of a series of races known as Grands Prix, held on circuits and public roads across various countries. Each race weekend includes multiple sessions: practice, qualifying, and the main race. The starting grid is determined through qualifying, and the final race result contributes to the World Drivers' Championship and Constructors' Championship, which are awarded at the end of the season.

To win those titles, each Formula 1 team must build their own car, choose their driver, and then deliver strategies during a Grand Prix. There are a lot of factor in a car and how well they can perform during a race, that is:

- **Aerodynamics**: Utilizing the wind pressure to create downforce, a force that pushes the car down, making the car sticks to the road and allow them to perform high speed in corners.
- **Power Unit**: The hybrid V6 engine and Energy Recovery System for power.
- **Tyres**: A crucial part of the car that determines how much grip the car has.
- **Chassis and Suspension**: The skeleton of the car, determine the weight and the wind flow of the car.

A car's performance is influenced not only by its mechanical setup but also by external factors like track temperature, fuel load, tyre selection, and weather conditions. There are a lot of strategic decisions that Formula 1 team have to make during a race that is:

- When to perform pit stops
- Which tyre compounds to use
- How to react to conditions (rain, crashes, etc)
- Whether to be aggressive or be conservative

D. Pit Stops

In modern Formula 1 racing, pit stop strategy is a critical component of overall race performance and can often be the determining factor in the outcome of a Grand Prix. A pit stop involves temporarily exiting the racetrack to perform actions such as changing tyres, adjusting front wing angles, or addressing mechanical issues. While a well-executed pit stop typically lasts between 2 to 3 seconds of stationary time, the overall time **loss** (including deceleration, pit lane speed limit, and rejoining the track) usually ranges from 18 to 25 seconds**,** depending on the circuit. Due to the high time cost, the decision of when to perform a pit stop and which tyres to switch to must be made with careful consideration.

There are two common strategies Formula 1 teams do in a Grand Prix:

- **Undercut:** The driver pits earlier than a rival and uses the advantage of fresher tyres to set faster lap times and overtake when the rival later pits. This is effective when fresh tyres provide an immediate and substantial performance gain.
- **Overcut:** A driver stays out longer while a rival has pitted and aims to go faster on older tyres. This may

be effective if the rival is stuck in traffic or if the tyre degradation is low.

E. Tyre Degradation

We talked about tyres and tyre degradation previously, but what exactly are they? Tyres in Formula 1 is vastly different than a normal road car tyre, it's smooth in contrast of grooved road car tyre, which is why it's also called slicks. Grooved tyre provides grip for normal road car on the street conditions. Grip is the ability of a car to stay stick to the road and not going anywhere. Normal tyre is grooved to add more friction to the road so the car can stick on the ground. But Formula 1 car is smooth, so logically it should have less friction and less grip, but Formula 1 tyre works differently.

Formula 1 car must perform at peak 350 km/h and the friction between the tyre and the ground will create a tremendous heat on the tyre. The tyre is made out of rubber, so when the temperature reaches a certain point, the rubber will melt. Melted rubber is sticky and this is where Formula 1 car gains their grip, by maintaining their tyre temperature, making it sticks more to the ground, hence more grip.

But those rubber won't persist on the tyre, the melted rubber will leave the tyre and sticks to the ground, eventually the tyre won't have good enough rubber to melt and create more grip, this is called tyre degradation. Formula 1 Driver must maintain their tyre temperature not only to make a good enough grip but also to maintain the amount of rubber melting (also called as tyre wear) to delay the degradation and allow them to perform with the same tyre much longer, minimizing the need for pit stops. Other than that, car setup can also determines how much their tyre degrades, this is why Formula 1 team must carefully engineer their car, not only for its aerodynamic, but also how well it affects the tyre degradation.

In modern Formula 1, in general there are 5 different tyre compound with each their own pros and cons.

Figure 2.2 F1 Tyre Compounds

Source: https://www.pirelli.com/tires/en-us/motorsport/f1/tires

To list it from the above picture, there are:

- **Hard compound (white):** It's the most durable tyre, it can last the longest because the rubber hardly melts, but it provides less grip.
- **Medium compound (yellow):** The balanced version between performance and durability.
- **Soft compound (red):** This tyre provides the most grip, since it's the softest material and can easily melt, but this tyre degrades quickly.
- **Intermediate (green):** This tyre is the most versatile rain tyre, it can be used on a wet track with no standing water, as well as a drying surface.
- **Full Wet (blue):** This tyre is the most effective on rain conditions, providing most grip on wet track.

III. IMPLEMENTATION

Before we dive into the implementation, first we must present the problem into a state space. In this implementation code we will call it a RaceState. A RaceState contains the current lap the car is in, the current tyre the car is using, the tyre wear up until the current lap, tyres available and compound used to match the race regulations, total time, and their parent state. The code for this RaceState is shown below:

```
class RaceState:
    def __init__(self,
                 current_lap,
                 current_tyre, tyre_wear,
                 tyres_available, compounds_used,
                 total_time,
                 parent=None):
        self.current_lap = current_lap
        self.current_tyre = current_tyre
        self.tyre_wear = tyre_wear
        self.tyres_available = dict(tyres_available)
        self.compounds_used = set(compounds_used)
        self.total_time = total_time
        self.parent = parent
```

Figure 3.1 RaceState Class
Source: Author Documentation

From this implementation, we are looking for the minimum total race time. So, the value of g(n) for this implementation is the total race time from the start node until node n. For the heuristic function, to keep it admissible and consistent, we keep an optimistic approach that is a driver can drive the remaining lap with the fastest lap time, which is using the soft tyre without any degradation, as shown in this code:

```python
def _heuristic_cost(self):
        remaining_laps = TOTAL_LAPS - self.current_lap
        if remaining_laps <= 0:
            return 0
        fastest_base_lap_time = BASE_LAP_TIME['soft'][WEATHER]
        return remaining_laps * fastest_base_lap_time
```

Figure 3.2 Heuristic Function
Source: Author Documentation

To further help this implementation, we must declare some constant to be the data of this implementation, Formula 1 teams usually have their own data about their tyre degradation. Since the car set up also determines the tyre degradation, they need to have every data accessible to improve their race strategy. In this implementation, to keep it simple, we will declare our own data, these datas are tyre degradation for each compound, the tyre allocation according to FIA's rulebook, the weather of the race (for simplicity, we do the race in dry weather), the base lap time of each tyre compound, and the time needed for pit stops since in F1 regulations, driver must maintain their speed to a certain limit, making them lose some time in the pit lane. Detailed data is provided in this code:

```python
TYRES = ['soft', 'medium', 'hard']
DEGRADATION = {
    'soft': 0.06,
    'medium': 0.015,
    'hard': 0.003
}
BASE_LAP_TIME = {
    'soft': {'dry': 76, 'wet': 88},
    'medium': {'dry': 79, 'wet': 91},
    'hard': {'dry': 82, 'wet': 94}
}
PIT_STOP_PENALTY = 30
TOTAL_LAPS = 50
TYRE_ALLOCATION = {
    'soft': 8,
    'medium': 3,
    'hard': 2
}
MANDATORY_COMPOUNDS = 2
WEATHER = 'dry'
WEAR_IMPACT_FACTOR = 0.3
```

Figure 3.3 Declared Race Variables
Source: Author Documentation

The WEAR_IMPACT_FACTOR is one important tuning parameter within our simulation. While DEGRADATION defines how much wear accumulates per lap, the WEAR_IMPACT_FACTOR scales how severely that accumulated wear affects the actual lap time performance. By adjusting this factor, we can model a more nuanced relationship between tire degradation and pace loss, allowing us to find strategies that reflect real-world F1 tendencies of extending tire life.

During a race, when we are in a RaceState there are multiple possibilities for the next Racestate available. In general, the next state is either we continue with the current tyre or we pit for this lap. When we pit, we are also provided with a bunch of tyre compound options. We turn all these possibilities into a race that will be the children of the current RaceState. For getting the next state logic, we implement it into this part of code:

```python
def get_possible_next_states(self):
        next_states = []

        if self.current_lap < TOTAL_LAPS:
            new_tyre_wear = self.tyre_wear + DEGRADATION[self.current_tyre]
            new_tyre_wear = min(new_tyre_wear, 1.0)

            lap_cost = self.calculate_lap_time()

            next_states.append(RaceState(
                current_lap=self.current_lap + 1,
                current_tyre=self.current_tyre,
                tyre_wear=new_tyre_wear,
                tyres_available=self.tyres_available,
                compounds_used=self.compounds_used,
                total_time=self.total_time + lap_cost,
                parent=self
            ))

        if self.current_lap < TOTAL_LAPS:
            for new_tyre_type in TYRES:
                if self.tyres_available[new_tyre_type] > 0:
                    updated_tyres_available = dict(self.tyres_available)
                    updated_tyres_available[new_tyre_type] -= 1

                    updated_compounds_used = set(self.compounds_used)
                    updated_compounds_used.add(new_tyre_type)

                    cost_of_new_lap_on_fresh_tyre = BASE_LAP_TIME[new_tyre_type][WEATHER] * (1 + 0.0)

                    next_states.append(RaceState(
                        current_lap=self.current_lap + 1,
                        current_tyre=new_tyre_type,
                        tyre_wear=DEGRADATION[new_tyre_type],
                        tyres_available=updated_tyres_available,
                        compounds_used=updated_compounds_used,
                        total_time=self.total_time + PIT_STOP_PENALTY + cost_of_new_lap_on_fresh_tyre,
                        parent=self
                    ))
        return next_states
```

Figure 3.4 Get Next State Function
Source: Author Documentation

We then get to the main implementation, that is the A* Search Algorithm. We use the library heapq from python to be our main priority queue. We use two sets, open set and closed set. Open set contains every race state possible currently then we take the race state with the least f score, f score is what we define f(n) here, while g(n) is g score and h(n) is h score. Close set contains race states that is already determined as the member of the optimal path to prevent redundancy and infinite loops. Here is the full implementation of the A* Search Algorithm:

```
def a_star_search():
    open_set = []
    g_score = {}
    f_score = {}

    for start_tyre in TYRES:
        if TYRE_ALLOCATION[start_tyre] > 0:
            initial_tyres_available_for_start = dict(TYRE_ALLOCATION)
            initial_tyres_available_for_start[start_tyre] -= 1

            initial_lap_time = BASE_LAP_TIME[start_tyre][WEATHER] * (1 + 0.0)

            start_state = RaceState(
                current_lap=1,
                current_tyre=start_tyre,
                tyre_wear=DEGRADATION[start_tyre],
                tyres_available=initial_tyres_available_for_start,
                compounds_used={start_tyre},
                total_time=initial_lap_time,
                parent=None
            )

            heapq.heappush(open_set, (start_state.total_time + start_state._heuristic_cost(), start_state))
            g_score[start_state] = start_state.total_time
            f_score[start_state] = start_state.total_time + start_state._heuristic_cost()

    closed_set = set()

    best_final_state = None

    while open_set:
        current_f_score, current_state = heapq.heappop(open_set)

        if current_state in closed_set:
            continue

        closed_set.add(current_state)

        if current_state.current_lap == TOTAL_LAPS:
            if len(current_state.compounds_used) >= MANDATORY_COMPOUNDS:
                if best_final_state is None or current_state.total_time < best_final_state.total_time:
                    best_final_state = current_state
            continue

        for neighbor in current_state.get_possible_next_states():
            if neighbor in closed_set:
                continue

            cost_added_to_reach_neighbor = neighbor.total_time - current_state.total_time
            tentative_g_score = current_state.total_time + cost_added_to_reach_neighbor

            if neighbor in g_score and tentative_g_score >= g_score[neighbor]:
                continue

            neighbor.parent = current_state
            neighbor.total_time = tentative_g_score
            g_score[neighbor] = tentative_g_score
            f_score[neighbor] = tentative_g_score + neighbor._heuristic_cost()
            heapq.heappush(open_set, (f_score[neighbor], neighbor))

    if best_final_state:
        return reconstruct_path(best_final_state)
    else:
        return None
```

Figure 3.5 A* Search Algorithm
Source: Author Documentation

Since a driver can start with different tyres. We also include every possible tyre compound as the starting node, this is implemented in the for start_tyres in TYRES loop, this will help the team to determine which compound to start first. Then we get all possible next states from the current state using the previous function, calculating the f score then put it in our priority queue. When we find the optimal path, we put the state in the close set, then we reconstruct the path when the best possible path is found.

IV. Results And Discussion

After running the implementation, we get one of the possible result with the summary:

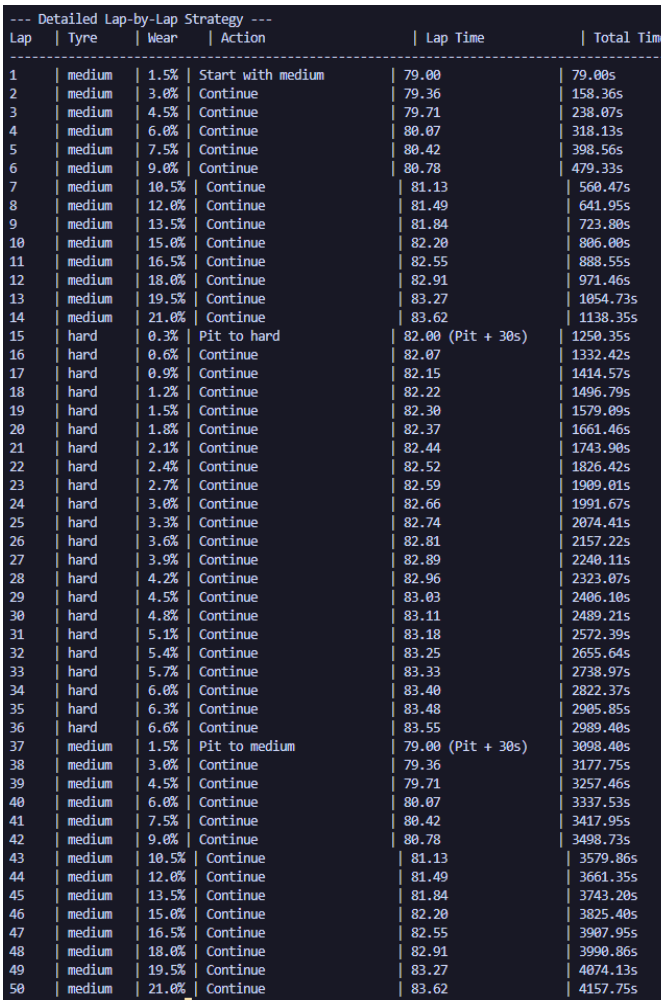
```
--- Optimal Race Strategy Summary ---
Total Race Time: 4157.75 seconds
Mandatory Compounds Used: hard, medium
Total Pit Stops: 2
```

Figure 4.1 Result Summary

Source: Author Documentation

We don't violate the rules that every team must at least two different compound during a race, and we get a total race time of 4149.53 seconds or approximately one hour and fifteen minutes, a reasonable race time without any accident happening on the track or any safety cars.

From one of the experiments we did, it decided to start with medium tyre and do a two pit stop strategy. Here is the detailed data of the results:



```
--- Detailed Lap-by-Lap Strategy ---
Lap  | Tyre     | Wear   | Action                | Lap Time           | Total Tim
-------------------------------------------------------------------------------
1    | medium   | 1.5%   | Start with medium     | 79.00              | 79.00s
2    | medium   | 3.0%   | Continue              | 79.36              | 158.36s
3    | medium   | 4.5%   | Continue              | 79.71              | 238.07s
4    | medium   | 6.0%   | Continue              | 80.07              | 318.13s
5    | medium   | 7.5%   | Continue              | 80.42              | 398.56s
6    | medium   | 9.0%   | Continue              | 80.78              | 479.33s
7    | medium   | 10.5%  | Continue              | 81.13              | 560.47s
8    | medium   | 12.0%  | Continue              | 81.49              | 641.95s
9    | medium   | 13.5%  | Continue              | 81.84              | 723.80s
10   | medium   | 15.0%  | Continue              | 82.20              | 806.00s
11   | medium   | 16.5%  | Continue              | 82.55              | 888.55s
12   | medium   | 18.0%  | Continue              | 82.91              | 971.46s
13   | medium   | 19.5%  | Continue              | 83.27              | 1054.73s
14   | medium   | 21.0%  | Continue              | 83.62              | 1138.35s
15   | hard     | 0.3%   | Pit to hard           | 82.00 (Pit + 30s)  | 1250.35s
16   | hard     | 0.6%   | Continue              | 82.07              | 1332.42s
17   | hard     | 0.9%   | Continue              | 82.15              | 1414.57s
18   | hard     | 1.2%   | Continue              | 82.22              | 1496.79s
19   | hard     | 1.5%   | Continue              | 82.30              | 1579.09s
20   | hard     | 1.8%   | Continue              | 82.37              | 1661.46s
21   | hard     | 2.1%   | Continue              | 82.44              | 1743.90s
22   | hard     | 2.4%   | Continue              | 82.52              | 1826.42s
23   | hard     | 2.7%   | Continue              | 82.59              | 1909.01s
24   | hard     | 3.0%   | Continue              | 82.66              | 1991.67s
25   | hard     | 3.3%   | Continue              | 82.74              | 2074.41s
26   | hard     | 3.6%   | Continue              | 82.81              | 2157.22s
27   | hard     | 3.9%   | Continue              | 82.89              | 2240.11s
28   | hard     | 4.2%   | Continue              | 82.96              | 2323.07s
29   | hard     | 4.5%   | Continue              | 83.03              | 2406.10s
30   | hard     | 4.8%   | Continue              | 83.11              | 2489.21s
31   | hard     | 5.1%   | Continue              | 83.18              | 2572.39s
32   | hard     | 5.4%   | Continue              | 83.25              | 2655.64s
33   | hard     | 5.7%   | Continue              | 83.33              | 2738.97s
34   | hard     | 6.0%   | Continue              | 83.40              | 2822.37s
35   | hard     | 6.3%   | Continue              | 83.48              | 2905.85s
36   | hard     | 6.6%   | Continue              | 83.55              | 2989.40s
37   | medium   | 1.5%   | Pit to medium         | 79.00 (Pit + 30s)  | 3098.40s
38   | medium   | 3.0%   | Continue              | 79.36              | 3177.75s
39   | medium   | 4.5%   | Continue              | 79.71              | 3257.46s
40   | medium   | 6.0%   | Continue              | 80.07              | 3337.53s
41   | medium   | 7.5%   | Continue              | 80.42              | 3417.95s
42   | medium   | 9.0%   | Continue              | 80.78              | 3498.73s
43   | medium   | 10.5%  | Continue              | 81.13              | 3579.86s
44   | medium   | 12.0%  | Continue              | 81.49              | 3661.35s
45   | medium   | 13.5%  | Continue              | 81.84              | 3743.20s
46   | medium   | 15.0%  | Continue              | 82.20              | 3825.40s
47   | medium   | 16.5%  | Continue              | 82.55              | 3907.95s
48   | medium   | 18.0%  | Continue              | 82.91              | 3990.86s
49   | medium   | 19.5%  | Continue              | 83.27              | 4074.13s
50   | medium   | 21.0%  | Continue              | 83.62              | 4157.75s
```

Figure 4.2 Detailed Strategy Result

Source: Author Documentation

From the result, the decision to start on the medium compound (Lap 1) rather than the faster soft or slower hard demonstrates the algorithm's balance of initial pace and durability. The medium offered a reasonable compromise for the first stint, preserving the faster soft compound for later, potentially crucial, phases of the race. The subsequent pit stops to hard and then medium compounds, respectively, illustrate the algorithm's dynamic adaptation to tyre wear characteristics and the mandatory compound rule. The long stints on the harder compounds are a direct result of their low degradation and reduced wear impact, making them efficient for prolonged running despite a slightly slower base pace.

The A* algorithm efficiently navigates this complex state space by maintaining an 'open set' of possible paths to explore, prioritized by their f(n) score. This score is a sum of g(n), the actual accumulated time from the start of the race to the current state, and h(n), an admissible heuristic estimating the minimum time to complete the remaining laps. By always prioritizing states with the lowest f(n), the algorithm explores the most promising strategies, ensuring the identified path is the global optimum.

While this model provides a robust theoretical optimum, real-time Formula 1 strategy is influenced by numerous unpredictable variables not captured here. These include, but are not limited to, track temperature changes, traffic management, tire 'cliffs' (a sudden, sharp drop in performance beyond a certain wear threshold), adverse weather changes mid-race, safety car periods, red flags, and driver errors. Integrating such dynamic elements would require a more complex state representation and potentially a different search paradigm, but this foundational A* model provides a strong baseline for understanding the core trade-offs.

V. Conclusion

With this result, we can conclude that we can find the optimal pit stop strategy for Formula 1 teams using the A* Search Algorithm. We can represent each lap of a race as a state then act accordingly. Determining the cost between state then find the optimal way to reach the final lap without costing too much time.

The A* algorithm proves to be an effective choice for this optimization problem due to its balance of completeness and efficiency. While a brute-force approach would have to evaluate every single possible strategy, leading to a combinatorially explosive search space, A* leverages its heuristic to intelligently prune unpromising paths, making the computation feasible for typical race lengths and tyre allocations.

The heuristic function plays a big role in this algorithm. While we use a simple and optimistic approach, the heuristic can be improvised to further boost the optimality of the result.

This experiment also highlights the varied use of path-finding algorithms such as A* Search Algorithm. If a problem can be represented into a space state with determined cost and has a definitive starting and ending state. We can practically use any path-finding algorithms and declare our way to approach the heuristic function.

Acknowledgment

With this, the author expresses the deepest gratitude to various parties who have assisted the author in the process of creating this paper. First, the author is grateful to God Almighty for His blessings that have guided and strengthened throughout the learning and writing process of this paper. The author also sincerely thanks the lecturer of class K-02 for the Algorithmic Strategy IF2211 course, Dr. Ir. Rinaldi Munir, M.T., for his guidance and teaching throughout the semester. Additionally, the author would like to extend heartfelt thanks to family and friends who have continuously supported and assisted the author in the learning process during the semester.

References

[1] Rafael Mart, Panos M. Pardalos, and Mauricio G. C. Resende. 2018. *Handbook of Heuristics (1st. ed.)*. Springer Publishing Company, Incorporated.

[2] Pearl, Judea. 1985. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley. https://archive.org/details/heuristicsintell00pear.

[3] A. Candra, M. A. Budiman and K. Hartanto, "Dijkstra's and A-Star in Finding the Shortest Path: a Tutorial," 2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA), Medan, Indonesia, 2020, pp. 28-32, doi: 10.1109/DATABIA50434.2020.9190342. keywords: {Computer science;Data science;Business;Shortest path problem;Time complexity;Tutorials;A-Star;Dijkstra's;Big-Theta;running time;SPBU},

[4] Noble, Jonathan. 2021. *Formula One Racing for Dummies*. 2nd ed. Hoboken, NJ: Wiley. https://books.google.co.id/books?id=pAnXEAAAQBAJ&lpg=PA3&ots=6tLMVDmlJJ&dq=formula%201%20race&lr&pg=PP1#v=onepage&q=formula%201%20race&f=false

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Juni 2025

Samuel Gerrard Hamonangan Girsang 13523064