Formula 1 Race Strategy Optimization via Weighted Directed Graph Modeling and Dijkstra's Algorithm

David Christian
Program Studi Teknik Informatika
Sekola Teknik Elektrik dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: christiandcdavid@gmail.com , 13525025@std.stei.itb.ac.id

***Abstract*— Race strategy in Formula 1 is a sequential decision-making problem in which teams must determine optimal pit stop timing and tire compound selection to minimize total race time. This paper addresses a graph theoretic formulation of Formula 1 race strategy optimization, modeling the race as a weighted directed graph in which each node represents a discrete race state defined by the current lap, active tire compound, and tire age. Directed edges encode two types of transitions, staying out on current tires or pitting to a fresh tire compound, with edge weights derived from a quadratic tire degradation model fitted to real race data. Dijkstra's shortest path algorithm is then applied to find the globally optimal strategy to get the path that minimizes total race time.**

Keywords*— Graph Theory, Dijkstra's Algorithm, Formula 1, Race Strategy Optimization, Discrete Mathematics.

I. Introduction

Formula 1 (F1) is one of the most technically demanding motorsport competitions in the world, where race outcomes are determined not only by raw vehicle performance and driver capability but also by the strategic decisions made by engineering and strategy teams during a race. Among these decisions, pit stop strategy, specifically the timing of tire changes and the selection of tire compounds, has become an increasingly decisive factor in determining finishing positions. A single mistimed pit stop can cost a driver several positions, while an optimal strategy can transform a midfield starting position into a race victory.

The complexity of pit stop strategy arises from its combinatorial nature. A race spanning 50 to 80 laps requires teams to decide, at each lap, whether to stay on current tires or pit for a fresh set, and if so, which compound to select from up to three available options. Tire performance degrades nonlinearly over time, pit stops incur a fixed time penalty from pit lane transit, and the *Fédération Internationale de l'Automobile* (FIA) mandates that each driver uses at least two distinct tire compounds during a dry race. The resulting search space is large, and teams currently rely on proprietary simulation tools and experienced race engineers to navigate it, often without a formally guaranteed optimal solution.

Discrete mathematics and graph theory in particular, provides a natural and strong framework for modelling this problem. A race can be represented as a weighted directed graph where nodes represent discrete race states and edges represent strategic transitions between those states. Finding the optimal strategy then reduces to finding the shortest path through this graph, a well-studied problem with efficient, provably optimal algorithms.

This paper makes the following contributions. First, we introduce a formal graph-theoretic model of Formula 1 race strategy, defining the state space, edge structure, and weight function in precise mathematical terms. Second, we derive a quadratic tire degradation model from real race data and incorporate it into the edge weight function. Third, we apply Dijkstra's algorithm to solve the shortest path problem across this race graph, providing an analysis of its computational complexity and structural efficiency. Fourth, we implement the proposed framework in Python and validate it against the 2025 Japanese Grand Prix, comparing the algorithmically derived optimal strategy against real data of the race.

II. THEORETICAL BASIS

A. Graph Theory

Graph theory is a branch of discrete mathematics concerned with the study of graphs. Mathematical structures used to model pairwise relations between objects.

1) Definition

A graph G is formally defined as an ordered pair G = (V, E), where V is a finite nonempty set of vertices (also called nodes) and E is a set of edges representing connections between pairs of vertices [1]. Meaning a valid graph must contain at least one node. However, the edge set E is permitted to be empty, which results in a graph consisting purely of isolated vertices without any connecting relationships.

Graphs are classified according to the properties of their edges. In an undirected graph, each edge (u, v) is unordered, meaning (u, v) and (v, u) represent the same connection. In a directed graph, each edge (u, v) is an ordered pair with a defined direction from u to v, and (u, v) ≠ (v, u) in general.

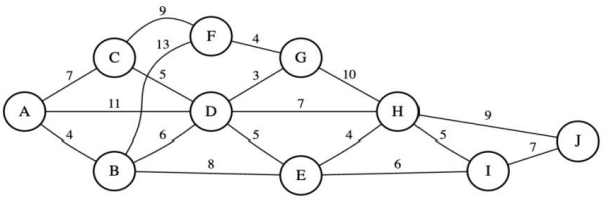

Fig. 1. Example of a Graph
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/matdis25-26.htm

2) Directed Graph (Digraph)

A graph where the edges have a specific direction. In a digraph, the edge set E consists of ordered pairs of vertices. An edge E = (u, v) indicates a one-way connection from the source vertex u to the destination vertex v.

A directed graph G = (V, E) is a graph in which every edge (u, v) ∈ E is an ordered pair, representing a directed connection from vertex u to vertex v.

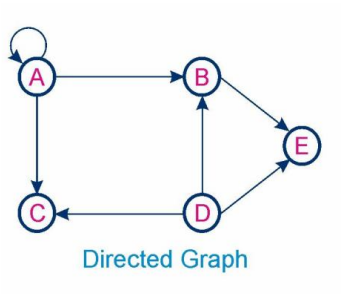

Fig. 2. Example of a Directed Graph
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/matdis25-26.htm

3) Weighted Directed Graph

A weighted graph extends the basic graph model by associating a numerical value, called a weight, with each edge. Weights typically represent costs, distances, capacities, or time, but in this work, each edge weight represents the lap time cost of a strategic transition.

A weighted directed graph G = (V, E, w) is a directed graph augmented with a weight function w: E → ℝ that assigns a real-valued weight w(u, v) to each edge (u, v) ∈ E.

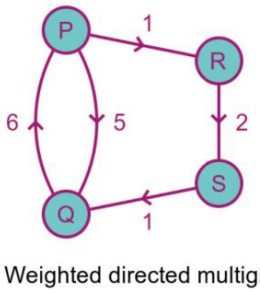

Fig. 3. Example of a Weighted Directed Graph
Source: https://graphicmaths.com/computer-science/graph-theory/graphs/

4) Path

A path in a directed graph is a sequence of vertices $P = \langle v_0, v_1, \ldots, v_k \rangle$ such that $(v_i, v_{i+1}) \in E$ for all $0 \leq i < k$. The length of a path in a weighted graph is the sum of the weights of its edges. The length of a path is the sum of the weights of its edges:

$$w(P) = \sum_{i=0}^{k-1} w(v_i, v_{i+1}) \quad (1)$$

For Fig 4, if v_0 is R and v_k is Q, then total path is 3.

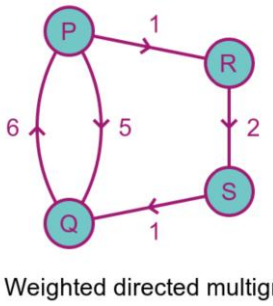

Fig. 4. Example of a Path in Graph
Source: https://graphicmaths.com/computer-science/graph-theory/graphs/

5) Directed Acyclic Graph (DAG)

A directed acyclic graph (DAG) is a directed graph G = (V, E) in which there exists no directed cycle, no path from any vertex v back to itself [2]. Formally, a directed graph G is acyclic if and only if a topological ordering of its vertices exists: a linear ordering $v_1, v_2, \ldots, v_n$ such that for every edge $(v_i, v_j) \in E$, $i < j$.

DAG is particularly useful for modeling processes that progress strictly in one direction, such as scheduling, dependency resolution, and sequential decision-making, where revisiting a previous state is structurally impossible.

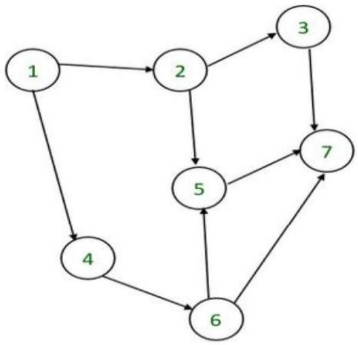

Fig. 5. Example of a DAG

B. Shortest Path Problem

The shortest path problem is one of the most fundamental problems in graph theory and combinatorial optimization. Given a weighted directed graph G = (V, E, w) and a designated source

vertex s ∈ V, the single-source shortest path problem asks for the minimum-weight path from s to every other reachable vertex in the graph [2].

The shortest path from s to a vertex t ∈ V is a path $P^* = \langle s = v_0, v_1, \ldots, v_k = t \rangle$ such that $w(P^*) \leq w(P)$ for all paths P from s to t. The distance function δ(s, v) denotes the weight of the shortest path from source s to vertex v.

$$\delta(s, v) = min\{ w(P) \}, \text{ if such a path exists; else } \infty. \quad (2)$$

Multiple algorithms solve the shortest path problem under different graph conditions. Table I summarize their time and space complexities and applicability to this work.

TABLE I. SHORTEST PATH ALGORITHM COMPARISON

Algorithm	Time Complexity	Space Complexity	Specification
BFS	O(V+E)	O(V)	Unweighted only
Dijkstra's	O((V + E) log V)	O(V)	Non negative weight
Bellman-Ford	O(V*E)	O(V)	None
Floyd-Warshall	$O(V^3)$	$O(V^2)$	None

Source: https://www.geeksforgeeks.org/dsa/comparison-between-shortest-path-algorithms/ , https://www.geeksforgeeks.org/dsa/time-and-space-complexity-of-dijkstras-algorithm/

Among these, Dijkstra's algorithm is the most appropriate for this problem because, all edge weights are strictly positive real values representing lap times, satisfying the non-negative weight requirement, the algorithm finds the globally optimal path, not just a heuristic approximation, and its O((V + E) log V) complexity is efficient for the graph sizes in race modeling.

C. Dijkstra's Algorithm

Dijkstra's algorithm, introduced by Edsger W. Dijkstra in 1959, solves the single-source shortest path problem on graphs with non-negative edge weights [3]. The algorithm operates by maintaining a priority queue of vertices ordered by their current best-known distance from the source, and iteratively finalizing the shortest distance to each vertex through a process called edge relaxation.

The correctness of Dijkstra's algorithm relies on the greedy choice property: once a vertex u is extracted from Q with minimum distance, dist[u] = δ(s, u) is final and will never be improved. This holds because all edge weights are non-negative.

The pseudocode for Dijkstra's Algorithm:

```
PROCEDURE Dijkstra(input G: Graph, input s: Vertex, output dist: Array of Integer, output: Array of Vertex)
{ I.S. A directed graph G = (V,E) with w(u, v) ≥ 0, s as starting vertex and s ∈ V(G) }
{F.S Array dist contains the shortest path distances from vertex s to all vertices v ∈ V(G). Array prev contains the predecessor vertex of each node for path reconstruction }

DECLARATION
u, v : Vertex
temp : Integer
Q : PriorityQueue

ALGORITHM
for each vertex v ∈ V(G) then
        dist[v] ← ∞
        prev[v] ← NULL
dist[s] ← 0
Q ← CreatePriorityQueue()
for each vertex v ∈ V(G) then
        Enqueue(Q, v, dist[v])
while not(isEmpty(Q)) then
        u ← GetMinimum(Q)
        for each vertex v adjacent to u then
                temp ← dist[u] + w(u,v)
                if (temp < dist[v]) then
                        dist[v] ← temp
                        prev[v] ← u
                        DecreaseKey ←(Q, v, temp)
```

D. Functions and Bijections

A function f: A → B is a mapping from a set A (domain) to a set B (codomain) such that every element of A maps to exactly one element of B.

Injective Function: A function f: A → B is injective (one-to-one) if distinct elements of A map to distinct elements of B:

$$\forall a_1, a_2 \in A: f(a_1) = f(a_2) \implies a_1 = a_2 \quad (3)$$

Surjective Function: A function f: A → B is surjective (onto) if every element of B has at least one preimage in A:

$$\forall b \in B: \exists a \in A \text{ such that } f(a) = b \quad (4)$$

Bijection: A function f: A → B is bijective if it is both injective and surjective and every element of B corresponds to exactly one element of A. A bijection establishes a one-to-one correspondence between A and B [4].

Bijections are particularly important in combinatorics and discrete mathematics because they allow two sets to be counted or reasoned about interchangeably. If a bijection exists between set A and set B, then |A| = |B| and any property provable for elements of A has a direct counterpart in B.

III. METHODOLOGY

A. Problem Definition

Formula 1 race strategy optimization is defined as the problem of determining, prior to or during a race, the sequence of pit stops decisions, including the lap at which each pit stop occurs and the tire compound selected at each stop, such that the total race time is minimized, subject to the regulatory and physical constraints of the sport.

Formally, let a race consist of L laps. At each lap $l \in \{1, 2, \ldots, L\}$, the team must make one of the following decisions:

- Stay out: Continue racing on the current tire compound without pitting.
- Pit: Enter the pit lane, incur a time penalty, and resume racing on a newly selected tire compound $c' \in C$, where $C = \{\text{Soft, Medium, Hard}\}$ is the set of available compounds.

The objective is to find the sequence of decisions $d = \langle d_1, d_2, \ldots, d_L \rangle$ that minimizes total race time T:

$$T^* = min(\sum_{l=1}^{L} \tau\,(l_l, c_l, a_l) + \sum_{k=1}^{K} p_k) \quad (5)$$

- $\tau(l, cl, al)$ = lap time at lap l on compound cl with tire age al
- p_k = pit lane time loss incurred at the k-th pit stop

This objective function is subject to a set of strict regulatory and physical constraints of F1.

- C1: At least 2 distinct tire compounds must be used [11]
- C2: Tire age $a_l \geq 0$ for all laps
- C3: $\tau(l, c, a) > 0$ for all valid states
- C4: Lap number strictly increases: $l \in \{1, \ldots, L\}$, no backtracking
- C5: At most one strategic action may be executed at each lap transition.

Constraint C4 is the key structural property that makes this problem suitable for graph modeling because the decision space is acyclic, enabling the application of shortest path algorithms with guaranteed optimality.

B. Model Assumption and Limitations

To produce a tractable and formally analyzable model, the following assumptions are adopted. Each assumption introduces a corresponding limitation on the scope of the model's conclusions.

- Single Vehicle Model

The optimizer finds the optimal strategy for one vehicle racing in free air, without modeling inter-vehicle traffic dynamics. Interactions such as the undercut, overcut, and blocking are excluded. This is consistent with standard practice in the academic race strategy [4], where single-vehicle models serve as a formal foundation prior to extension into multi-agent settings.

Limitation: Strategies that are optimal in isolation may be suboptimal when competitor behavior is accounted for. The model cannot capture reactive decisions such as pitting in direct response to a competitor's stop.

- Deterministic race conditions.

The model assumes green flag racing throughout the full race distance. Stochastic events like safety cars, virtual safety cars, red flags, and rainfall are excluded. This assumption is satisfied by the drivers selected for comparison in Section V, all of whom completed the 2025 Japanese Grand Prix under representative green flag strategic conditions without race-altering interventions.

Limitation: The model cannot evaluate or respond to safety car windows, which in practice represents the most significant source of unplanned strategic variation in modern Formula 1.

- Fixed Pit Lane Loss

The pit stop time penalty Δpit is treated as a fixed constant representing the expected green flag pit lane delta for the circuit. Dynamic variation due to pit lane congestion or unsafe release is not modeled.

Limitation: In races with heavy pit lane traffic, actual pit loss may deviate from the fixed value, affecting the accuracy of optimal pit window predictions.

- Tire Degradation

Tire performance degradation is modeled as a smooth quadratic function of tire age, fitted from the median lap times of drivers running each compound under clean green flag conditions in the 2025 Japanese Grand Prix. Out-laps, in-laps, and laps completed under safety car conditions are excluded from the fitting data. The model does not capture sudden tire cliff behavior, blistering, graining, or driver-specific tire management.

Limitation: In races where tires cliff sharply at a specific age threshold, the quadratic model may underestimate the true cost of running old tires, potentially recommending later pit stops than are physically possible.

C. Graph Representation

To model the race as a graph problem, a discrete state space is defined that fully captures all information required to determine the cost and feasibility of every future decision from any point in the race.

A race state is a 4-tuple:

$$v = (l, c, a, U) \quad (6)$$

- $l \in \{1, \ldots, L\}$, current lap number
- $c \in \{0, 1, 2\}$, active tire compound index (0 = Soft, 1 = Medium, 2 = Hard)

- a ∈ {0, …, l}, current tire age in laps since the last pit stop or race start
- U ∈ {1, …, 7}, compound usage bitmask encoding the set of compounds used so far

A 3-bit integer bitmask(U) is used for representing the tire compound,

$$U = \Sigma_{c' \in used} 2^{c'} \quad (7)$$

Where the summation runs over all compound indices c' that have been used at any point in the race up to and including the current state.

TABLE II. TIRE COMPOUND REPRESENTATION WITH BITMASK

Decimal	Binary	Tire	Valid (C1)
1	001	Soft	No
2	010	Medium	No
3	011	Soft + Medium	Yes
4	100	Hard	No
5	101	Soft + Hard	Yes
6	110	Medium + Hard	Yes
7	111	Soft + Medium + Hard	Yes

```
def count_used_compounds(u_bitmask):
  # calculate popcount(U) to fullfill Constraint C1
    return bin(u_bitmask).count('1')
```

Fig. 6. Function to check if U valid for C1 in Python

D. Graph Model

The race graph G is a directed acyclic graph. Every edge in E increments the lap counter l by exactly one. Therefore, no directed cycle can exist, since returning to any previously visited state would require l to decrease, which no edge in E permits. Consequently, G admits a topological ordering by lap index l, meaning that G is a DAG. The graph is a modeled as:

$$G = (V, E, w) \quad (8)$$

- V : set of all valid race states
- E: set of directed strategic transitions between states
- w: E → $\mathbb{R}^+$ is the weight function assigning a positive real-valued cost to each edge.

Source node: Three source vertices are defined, corresponding to starting the race on Soft, Medium, and Hard compounds respectively, with tire age set to 0, and bitmask initialized to record c_0 to used.

Sink node: For every state v = (L, c, a, U) with count(U) ≥ 2, a directed edge (v → FINISH, 0) is added to E. States at lap L with popcount(U) < 2 have no edge to finish and are dead ends naturally excluded by the algorithm.

From each non-terminal state v = (l, c, a, U) where l < L, two types of directed edges are defined:

- Stay on the race:

The driver completes lap l on the current compound without pitting. Tire age increments by one. Bitmask U is unchanged since no new compound is introduced.

$$(l, c, a, U) \rightarrow (l + 1, c, a + 1, U) \quad (9)$$

$$w = \tau(c, a) \quad (10)$$

- Pit stop to compound c`:

The driver pits at the end of lap l, paying pit lane time loss Δpit on top of the current lap time, and resumes on fresh compound c' with tire age reset to zero. The bitmask is updated to record c'. This edge exists for all c' ∈ {0, 1, 2} including c' = c.

$$(l, c, a, U) \rightarrow \left(l + 1, c', 0, U \cup 2^{c`}\right) \quad (11)$$

$$w = \tau(c, a) + \Delta pit \quad (12)$$

The weight w of any edge represents the time cost incurred during lap l. For a stay on the race edge this is simply the lap time τ(c, a). For a pit edge the driver completes lap l at time τ(c, a) and then loses Δpit in the pit lane and both costs are attributed to the transition at lap l, giving w = τ(c, a) + Δpit.

Using the definitions of injection and surjection from theoretical basis, we now establish a formal correspondence between legal race strategies and directed paths in G. Every legal race strategy σ = ⟨d_1, d_2, …, d_L⟩ satisfying constraints C1–C5 corresponds to exactly one directed path from source s to finish in G, and every directed path from s to finish in G corresponds to exactly one legal race strategy.

Let σ and σ' be two distinct legal strategies. Then there exists a smallest lap l* at which they differ, either one stays out while the other pits, or both pit but to different compounds. At lap l* both paths are in the same state, but they follow different outgoing edges. Since the outgoing edges from any state are distinct, the two paths diverge at l* and are distinct. Therefore, the mapping σ → P is injective.

Let P be any directed path from s to finish in G. At each lap l, read the edge type traversed: a stay-out edge yields d_l = stay on the race, a pit edge to c' yields d_l = pit(c'). This produces a well-defined sequence σ = ⟨d_1, …, d_L⟩. Therefore, σ is a legal strategy and the mapping P → σ is surjective.

This bijection proves that minimizing w(P) over all directed paths from s to finish is the same to minimizing T over all legal race strategies. Therefore, solving the shortest-path problem on G is equivalent to solving the original race-strategy optimization problem.

E. Data Preparation

Race telemetry data were obtained from the 2025 Formula 1 Japanese Grand Prix using the FastF1 Python library. Only dry-weather slick compounds were considered, namely Soft, Medium, and Hard tires [6]. Intermediate and Wet compounds

were excluded because the proposed model assumes a fully dry race.

To reduce noise unrelated to tire wear, only clean green-flag laps were retained. Pit entry laps, pit exit laps, safety car laps, virtual safety car laps, and observations with missing timing information were removed from the dataset. The remaining observations were used to estimate representative degradation parameters for each compound.

F. The Tire Degradation

The edge weight function w depends on a deterministic tire degradation model mapping compound c and tire age a to a predicted lap time. Degradation is modelled as a quadratic polynomial in tire age:

$$\tau(l, c, a) = \beta_0 + \beta_1 a + \beta_2 a^2 \quad (13)$$

- β_0 = baseline lap time on compound c with fresh tires, excluding fuel effects
- β_1 = linear degradation coefficient capturing steady early lap time loss
- β_2 = quadratic degradation coefficient capturing the acceleration of degradation at higher tire ages
- a = tire age

```
def get_lap_time(c, a, tire_params):
  # calculate equation 13
    b0, b1, b2 = tire_params[c]
    return b0 + (b1 * a) + (b2 * (a ** 2))
```

Fig. 7. Function for equation 13 in Python

G. Pit Stop Time Loss Model

When a driver pits, they get a pit lane time loss Δpit. It is the time deficit relative to completing the same lap at race pace without stopping. This accounts for deceleration into the pit entry, pit time in the pit box, and acceleration back through the pit exit:

$$\Delta pit = t_{entry} + t_{box} + t_{exit} - t_{lap} \quad (14)$$

- t_{entry} = time spent decelerating and traveling the pit entry lane under the speed limit
- t_{box} = time required by the mechanics to change the tire compound
- t_{exit} = time spent accelerating out of the pit exit lane
- t_{lap} = the theoretical time it would take the vehicle to traverse the equivalent main straight section of the track at full race pace

However, to preserve the deterministic nature of the directed acyclic graph and ensure the computational viability of the shortest-path algorithm, the pit loss defined to static constant constraint. By holding this penalty fixed with average of Δpit = 23.75 seconds for Suzuka [7] under green-flag conditions, the graph's edge weights (w) for pit transitions remain strictly positive and predictable.

H. Shortest Path Optimization Using Dijkstra's Algorithm

With the graph parameterized, finding the optimal strategy becomes a shortest-path problem. To guarantee an absolute global minimum, Dijkstra's algorithm is executed independently across three source vertices representing each starting compound: Soft, Medium, and Hard.

To simplify termination, a universal sink node t_{finish} is added. Zero-weight edges connect this sink to any valid state, naturally pruning illegal strategies. Because the degradation model ensures all edge weights are strictly positive time costs, Dijkstra's min-priority queue guarantees convergence to the optimal solution.

Once t_{finish} is reached, the minimum-cost path is recovered via predecessor pointers. This mathematical path deterministically translates into the physical race sequence $\sigma^* = \langle d_1, d_2, \ldots, d_L \rangle$, outputting the exact lap intervals and compound selections for the optimal strategy.

$$|V| <= L \times |C| \times L \times 2\text{^}|C| \quad (15)$$

Because each state generates at most |C| outgoing edges, the graph is exceedingly sparse, such that $|E| \approx |C||V|$. Implemented with a standard binary min-heap, the time complexity of Dijkstra's algorithm is bounded by $O((|V| + |E|) \log |V|)$. For a standard Formula 1 race length (L is 50 to 80 laps) and |C| = 3, the state space evaluates to fewer than 100,000 vertices. This low complexity allows the deterministic global optimum to be resolved in milliseconds, rendering the model computationally viable for dynamic, real-time recalculations on the pit wall.

To operationalize this theoretical framework, the shortest path optimization is implemented in Python. The standard library's heapq module is utilized to maintain the min-priority queue, strictly enforcing the logarithmic time complexity derived above. The following implementation demonstrates the core optimization engine, translating the 4-tuple state space, polynomial edge weight calculations, and regulatory constraint verification directly from the formal graph definition into a deterministic computational model.

```python
# DIJKSTRA'S ALGORITHM
def optimize_race_strategy(L, pit_loss, start_compound, tire_params):

    start_u = 1 << start_compound
    start_state = (1, start_compound, 0, start_u)

    pq = [(0.0, start_state)]
    dist = {start_state: 0.0}
    prev = {start_state: None}

    optimal_finish_state = None

    while pq:
        current_time, u = heapq.heappop(pq)
        l, c, a, U = u

        if current_time > dist.get(u, float('inf')):
            continue

        # finish the race
        if l == L + 1:
            if count_used_compounds(U) >= 2:
                optimal_finish_state = u
                break
            else:
                continue

        lap_cost = get_lap_time(c, a, tire_params)

        # Edge 1: stay on the race
        stay_state = (l + 1, c, a + 1, U)
        stay_cost = current_time + lap_cost

        if stay_cost < dist.get(stay_state, float('inf')):
            dist[stay_state] = stay_cost
            prev[stay_state] = (u, "Stay on race")
            heapq.heappush(pq, (stay_cost, stay_state))

        # Edge 2: pit stop
        for c_new in [0, 1, 2]:
            new_u = U | (1 << c_new)
            pit_state = (l + 1, c_new, 0, new_u)
            pit_cost = current_time + lap_cost + pit_loss

            if pit_cost < dist.get(pit_state, float('inf')):
                dist[pit_state] = pit_cost
                prev[pit_state] = (u, f"Pit (Compound {c_new})")
                heapq.heappush(pq, (pit_cost, pit_state))

    # recovery path
    if not optimal_finish_state:
        return None, None

    path = []
    curr = optimal_finish_state
    total_time = dist[curr]

    while curr in prev and prev[curr] is not None:
        parent_state, action = prev[curr]
        path.append((parent_state[0], action, curr[1]))
        curr = parent_state

    path.reverse()
    return total_time, path
```

Fig. 8. Dijkstra Implementation in Python

IV. Result and Discussion

A. Experimental Setup

The proposed optimization framework was evaluated using the 2025 Suzuka Circuit race configuration. The race distance was set to 53 laps, and the shortest-path optimization was executed independently from three possible starting compounds: Soft, Medium, and Hard. The edge weights were computed using the tire degradation model described in Section III-F combined with the pit stop penalty model of Section III-G. Dijkstra's algorithm was then applied to identify the minimum-time path from each source state to the finish node.

B. Tire Degradation Result

The tire degradation model was calibrated using clean green-flag race data from the 2025 Japanese Grand Prix obtained through the FastF1 telemetry database. Relative lap-time normalization was applied within each stint to reduce the influence of driver-specific pace differences, fuel load variation, and track evolution. Separate regressions were performed for the Soft, Medium, and Hard compounds.

TABLE III. Tire Degradation Parameters

Tire compound	β_0	β_1	β_2
Soft	92,222	0,0811	0,000
Medium	92,884	0,0342	0,000
Hard	93,103	0,0184	0,000

The fitted parameters preserve the expected performance hierarchy of Formula 1 tire compounds. The Soft compound exhibits the lowest baseline lap time but the highest degradation rate, while the Hard compound exhibits the slowest baseline pace and the lowest degradation rate. This behaviour is consistent with the intended design characteristics of the Pirelli tire compounds [6]. The primary purpose of the degradation model in this work is to generate realistic edge weights for the graph optimization framework.

C. Optimal Strategy Comparison

To determine the globally optimal race strategy, Dijkstra's algorithm was executed independently using Soft, Medium, and Hard starting compounds.

```python
def evaluate_all_starting_compounds(L, pit_loss, tire_params):
    comp_names = {
        0: "Soft",
        1: "Medium",
        2: "Hard"
    }

    results = {}
    summary_rows = []
    print("\nSTRATEGY FOR EACH COMPOUND\n")

    for start_c in [0, 1, 2]:
        total_time, path = optimize_race_strategy(L, pit_loss, start_c, tire_params)
        if total_time is None:
            print(f"Starting on {comp_names[start_c]:6s}: NO VALID PATH")
            continue
        results[start_c] = {
            "time": total_time,
            "path": path
        }
        minutes = int(total_time // 60)
        seconds = total_time % 60
        strategy = [comp_names[start_c]]
        pit_laps = []

        for lap, action, comp in path:
            if "Pit" in action:
                strategy.append(comp_names[comp])
                pit_laps.append(lap)

        strategy_str = " → ".join(strategy)
        summary_rows.append({
            "Start Compound": comp_names[start_c],
            "Race Time (s)": round(total_time, 3),
            "Strategy": strategy_str,
            "Pit Laps": ", ".join(map(str, pit_laps))
            if pit_laps else "None",
            "Pit Stops": len(pit_laps)
        })
        print(f"{comp_names[start_c]:6s} | "f"{minutes:02d}:{seconds:06.3f} | "f"{strategy_str}")

    if not results:
        print("No strategy found.")
        return

    # Find global optimum
    best_start = min(results, key=lambda k: results[k]["time"])
    best_time = results[best_start]["time"]
    best_path = results[best_start]["path"]
    minutes = int(best_time // 60)
    seconds = best_time % 60

    print("\nGLOBAL OPTIMUM\n")
    print(f"Starting Compound : {comp_names[best_start]}")
    print(f"Race Time         : {minutes:02d}:{seconds:06.3f}")
    print(f"Total Seconds     : {best_time:.3f}")

    strategy = [comp_names[best_start]]
    pit_laps = []
    for lap, action, comp in best_path:
        if "Pit" in action:
            strategy.append(comp_names[comp])
            pit_laps.append(lap)

    print(f"Strategy          : {' → '.join(strategy)}")
    print(f"Pit Stops         : {len(pit_laps)}")

    for i, lap in enumerate(pit_laps):
        print(f"Pit Stop #{i+1}      : Lap {lap}")

    print(f"Finish            : Lap {L}")
    print("SUMMARY TABLE")
    df_summary = pd.DataFrame(summary_rows)
    print(df_summary.to_string(index=False))

    return df_summary
```

Fig. 9. Dijkstra Implementation on all compounds in Python

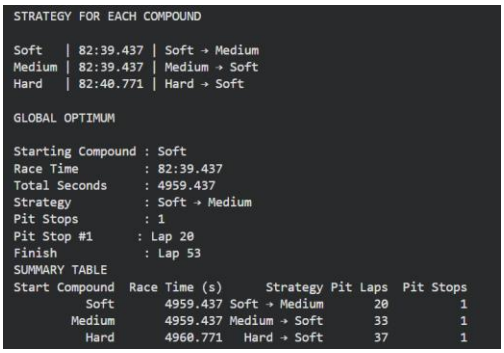

Fig. 10. The result of the code of Fig. 6

The results indicate that the optimizer consistently favors one-stop strategies regardless of starting compound. The Soft-start and Medium-start strategies produce identical predicted race times of 4959.437 seconds, while the Hard-start strategy is approximately 1.33 seconds slower.

Among all evaluated starting conditions, the Soft-start strategy was selected as the global optimum. The optimal path recommends remaining on the soft compound until Lap 20 before performing a single pit stop to the medium compound and completing the remainder of the race on that tire.

D. Discussion of the Optimal Solution

The optimal strategy emerges from the trade-off between tire degradation and pit-lane time loss. Remaining on a tire compound for a longer period avoids the fixed cost associated with entering the pit lane, but simultaneously increases lap times due to tire wear. Conversely, pitting too early reduces degradation but incurs additional pit-lane losses that outweigh the performance gain from fresh tires.

For the Soft-start strategy, the optimizer determines that Lap 20 represents the best compromise between these competing effects. Earlier pit stops increase the number of laps completed on the slower second stint, whereas later pit stops accumulate excessive degradation on the opening soft stint. Consequently, the path containing a pit stop at Lap 20 achieves the minimum total path weight and is therefore selected as the optimal strategy.

E. Comparison with Actual Race Result

The actual race winner (1st), Max Verstappen, completed the 2025 Japanese Grand Prix in 4926.983 seconds using a Medium-to-Hard one-stop strategy with a pit stop occurring around Lap 22 [10].

TABLE IV. COMPARISON WITH ACTUAL RACE

Variable	Max Verstappen	Model
Compound Strategy	Medium -> Hard	Medium -> Soft
Stops	1	1
Pit	Lap 22	Lap 33
Total Race Time	4926.983 s	4959.437 s

The absolute prediction error is:

$$| 4959.437 - 4926.983 | = 32.454 \text{ s} \quad (16)$$

Percentage error of:

$$0.659\% \quad (17)$$

Although the proposed model does not exactly reproduce the observed strategy. The differences in pit timing and tire selection can be attributed to several factors not represented in the model, including fuel load effects, traffic conditions, undercut and overcut opportunities, driver-specific performance, track evolution, and team-dependent vehicle characteristics. If this strategy was used in the race, it's still finished 7th overall.

F. Limitations

First, the model assumes deterministic race conditions and does not account for safety cars, virtual safety cars, weather changes, or mechanical failures. Second, interactions among competing vehicles are excluded. Third, tire degradation is represented by a simplified polynomial model and therefore may not fully capture complex phenomena such as tire cliff behaviour, graining, or blistering.

Nevertheless, the results demonstrate that race strategy optimization can be formulated successfully as a shortest-path problem on a directed acyclic graph. The proposed framework provides a mathematically rigorous foundation that can be extended in future work to incorporate stochastic events, fuel effects, and multi-vehicle interactions.

V. CONCLUSION

This paper presented a Formula 1 race strategy optimization framework based on weighted directed graph modelling and Dijkstra's shortest path algorithm. The race was modelled as a directed acyclic graph in which each vertex represents a race state and each edge represents a strategic decision with an associated time cost derived from a tire degradation model.

Using race data from the 2025 Japanese Grand Prix, the proposed method successfully identified feasible and globally optimal pit-stop strategies while satisfying FIA tire regulations. The results demonstrated that the optimizer consistently favored one-stop strategies, with the Soft-to-Medium strategy producing the lowest predicted race time of 4959.437 seconds.

Although the model does not account for factors such as traffic, fuel effects, weather, or competitor interactions, the results show that shortest path optimization provides an effective and mathematically rigorous approach for Formula 1 race strategy analysis. Future work may incorporate these additional factors to improve realism and predictive accuracy.

VIDEO LINK AT YOUTUBE

Link Video: https://youtu.be/6LoKVGrPChw

ACKNOWLEDGMENT

The author sincerely would like to thank The Almighty God for providing the strength and grace to be able to finish this paper. Deep appreciation to the lecturer of the IF1220 Discrete Mathematics course, Dr. Ir. Rinaldi Munir, M.T. for his teaching, guidance, support this semester. Furthermore, the author also wishes to acknowledge Max Verstappen as one of the best driver in the Formula 1 and as the core inspirations for this study. Finally, the author is deeply thankful towards his family and their support throughout his life.

REFERENCES

[1] R. Munir, "Graf (Bagian 1) ," Institut Teknologi Bandung, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf [Accessed: Jun. 9, 2026].

[2] K. H. Rosen, *Discrete Mathematics and Its Applications*, 8th ed. New York, NY, USA: McGraw-Hill, 2019. Available: https://cis.temple.edu/~latecki/Courses/CIS2166-Fall25/RosenDiscreteMath8Ed.pdf [Accessed: Jun. 9, 2026].

[3] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. Available: https://ir.cwi.nl/pub/9256/9256D.pdf [Accessed: Jun. 9, 2026].

[4] R. Munir, "Relasi dan Fungsi (Bagian 3: fungsi khusus, relasi kesetaraan, relasi pengurutan parsial, dan klosur relasi) ," Institut Teknologi Bandung, 2026. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/07-Relasi-dan-Fungsi-Bagian3-(2026).pdf [Accessed: Jun. 9, 2026].

[5] A. Heilmeier, A. Thomaser, M. Graf, and J. Betz, "Virtual Strategy Engineer: Using Artificial Neural Networks for Making Race Strategy Decisions in Circuit Motorsport," *Applied Sciences*, 2020. https://www.mdpi.com/2076-3417/10/21/7805 [Accessed: Jun. 13 2026]

[6] Pirelli Motorsport, "Formula 1 and Pirelli," Pirelli Motorsport. Available: https://www.pirelli.com/tires/en-us/motorsport/car/formula-1. [Accessed: Jun. 13, 2026].

[7] Formula 1, "NEED TO KNOW: The most important facts, stats and trivia ahead of the 2026 Japanese Grand Prix," *Formula 1*, 2026. Available: https://www.formula1.com/en/latest/article/need-to-know-the-most-important-facts-stats-and-trivia-ahead-of-the-2026-japanese-grand-prix.59xoCL07qjNCWPVzBJOmK2. [Accessed: Jun. 13, 2026].

[8] FastF1, "FastF1 Documentation," Available: https://docs.fastf1.dev/. [Accessed: Jun. 16, 2026].

[9] T. Becker, "FastF1: Formula 1 data analysis package," GitHub, 2025. Available: https://github.com/theOehrly/Fast-F1. [Accessed: Jun 16 2026].

[10] Formula One World Championship Limited, "FORMULA 1 LENOVO JAPANESE GRAND PRIX 2025," Formula1.com. Available: https://www.formula1.com/en/racing/2025/japan. [Accessed: Jun. 17, 2026].

[11] Fédération Internationale de l'Automobile (FIA), "2025 FIA Formula One Sporting Regulations," FIA, Paris, France, 2025. Available: https://www.fia.com/regulation/category/110. [Accessed: Jun 13, 2026].

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, 18 Juni 2026

David Christian - 13525025