Universiteit Leiden
The Netherlands

BSc Computer Science

Optimizing Formula 1 Pit Stop Strategies using QUBO and Annealing-Based Methods

Sophie Kolstee

Supervisors:
Jan Krzywda & Evert van Nieuwenburg

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

April 13, 2026

Abstract

This thesis studies the Formula 1 pit stop strategy problem as a combinatorial optimization problem and formulates it as a Quadratic Unconstrained Binary Optimization (QUBO) model. The objective is to minimize total race time by choosing tire compounds and pit stop moments over a fixed number of laps, while satisfying regulatory constraints. The model is built incrementally, starting from a base formulation and extending it with additional factors such as weather, traffic, fuel load, safety car periods, and tire degradation. The QUBO formulations are evaluated using simulated annealing (SA) and quantum-inspired annealing (QIA), and compared in terms of feasibility, solution quality, runtime, and scalability. Small instances are validated using brute-force and a greedy baseline. The results show that the problem can be modeled effectively as a QUBO and that the incremental approach allows realistic extensions without changing the structure. For simpler models, both solvers find feasible solutions, but SA generally achieves better results as problem size increases. For more complex models with tire-age tracking, SA remains applicable while QIA fails to find feasible solutions. Overall, QUBO provides a flexible modeling framework for this problem, but simulated annealing performs more reliably than quantum-inspired annealing on the tested instances.

Contents

1 Introduction

Optimization plays a central role in decision-making across domains such as logistics, transportation, and engineering. Many real-world problems can be formulated as combinatorial optimization problems, where an optimal solution must be selected from a discrete and often exponentially large set of possibilities [KHG+14, Pap98]. A substantial subset of these problems is NP-hard, making exact solution methods computationally infeasible for large instances [Pap98].
Classical approaches include exact algorithms as well as heuristic and metaheuristic methods. Among these, simulated annealing (SA) is widely used due to its general applicability and simplicity [KGV83]. However, its performance depends on parameter tuning and may degrade as problem size and complexity increase.
These limitations motivate alternative modeling and solution approaches. Quadratic Unconstrained Binary Optimization (QUBO) provides a unified framework for representing a wide range of combinatorial problems [KHG+14, GKD22]. Its solver-independence allows problems to be addressed using both classical and emerging optimization techniques.
Quantum annealing (QA) has been proposed as a method for solving QUBO and Ising formulations by exploiting quantum effects such as tunneling [JAG+11]. However, access to large-scale quantum hardware remains limited in practice. As a result, recent research has focused on quantum-inspired annealing (QIA) methods, which emulate key principles of QA on classical hardware [Iov25]. In Formula 1 racing, pit stop strategy plays a critical role in race performance. During a race, a driver must decide when to perform pit stops and which tire compounds to use. Softer compounds offer higher initial performance but degrade more quickly, whereas harder compounds are more durable but slower. Regulations require the use of at least two different compounds under dry conditions, enforcing at least one pit stop [Fé25]. The complexity of the problem arises from the interdependence of decisions over time: pit stop timing affects both immediate cost and future lap times.

1.1 Research Objective and Research Question

The primary goal of this thesis is to formulate the Formula 1 pit stop strategy problem as a Quadratic Unconstrained Binary Optimization (QUBO) model. The focus lies on constructing a representation that captures key aspects of the problem, including tire degradation, pit stop timing, and regulatory constraints, within a unified quadratic framework.
In addition to model construction, the resulting QUBO formulation is evaluated using different optimization approaches. In particular, classical simulated annealing and quantum-inspired annealing methods are applied to assess the model.
This leads to the following research question:

> *How can the Formula 1 pit stop strategy problem be effectively formulated as a QUBO model, and how do simulated annealing and quantum-inspired annealing methods perform when applied to this formulation?*

To support this question, the evaluation focuses on four criteria:

- feasibility of the solution

- solution quality, measured by total race time;
- runtime, reflecting computational efficiency;
- scalability, describing performance as problem size increases.

1.2 Thesis overview

This thesis was conducted as part of the Computer Science program at LIACS, Leiden University, under the supervision of Jan Krzywda and Evert van Nieuwenburg.
The remainder of this thesis is organized as follows. Section 2 presents the formal problem formulation and introduces the QUBO framework. Section 3 discusses relevant literature on classical and quantum-inspired optimization methods, as well as existing approaches to modeling Formula 1 pit stop strategies. In Section 4, the proposed QUBO model for Formula 1 pit stop strategies is developed. Section 5 outlines the experimental setup, while Section 6 presents and evaluates the results. Finally, Section 7 concludes the thesis and suggests directions for future work.
The full implementation and experimental pipeline are available at: https://github.com/skolstee/f1-pitsop-qubo

2 Definitions and Problem Formulation

This chapter introduces the formal problem setting and the notation used throughout this thesis. We describe the pit stop strategy problem in a simplified setting and define the key components of the decision problem. In addition, we provide a brief introduction to QUBO as a general modeling framework.

2.1 Problem Setting

The objective is to minimize total race time, subject to regulatory constraints and additional factors such as fuel load, safety car conditions, and traffic effects, which are formalized in later sections.
We consider a simplified model of a Formula 1 race consisting of $N = 50$ laps. The problem is studied for a single car under deterministic conditions, without interactions with other drivers.
Let $W = \{\text{dry}, \text{wet}\}$ denote the set of weather conditions. The set of available tire compounds is given by $T = \{\text{soft}, \text{medium}, \text{hard}\}$ under dry conditions, and is extended in wet conditions with $\{\text{intermediate}, \text{wet}\}$, where dry compounds remain available but are less effective.
A race strategy specifies, for each lap, whether a pit stop is performed and which tire compound is used. Each pit stop incurs a fixed time loss of 20 seconds, while lap times depend on the selected compound, its degradation, and the prevailing weather conditions.

2.2 Decision Variables and Objective Function

To represent the race strategy, we introduce binary decision variables.
For each lap $l \in \{1, \ldots, N\}$ and each compound $c \in T$, we define

$$x_{l,c} \in \{0, 1\},$$

where $x_{l,c} = 1$ if compound c is used on lap l, and 0 otherwise.
In addition, for each compound $c \in T$, we define

$$u_c \in \{0, 1\},$$

where $u_c = 1$ if compound c is used at least once during the race.

2.3 QUBO Formulation

A Quadratic Unconstrained Binary Optimization (QUBO) problem has the general form

$$\min_{z \in \{0,1\}^m} z^\top Q z,$$

where z is a vector of binary variables and $Q \in \mathbb{R}^{m \times m}$ is a symmetric matrix [KHG+14].
In this formulation, linear terms represent individual contributions to the objective, while quadratic terms capture interactions between decision variables. Constraints are incorporated into the objective function through penalty terms, allowing constrained problems to be expressed in an unconstrained quadratic form [GKD22, Luc14].
The formulation of the pit stop strategy problem within this framework is presented in Chapter 4.

3 Related Work

This chapter reviews related work relevant to the modeling of optimization problems and, in particular, the application of such methods to Formula 1 pit stop strategies. First, existing approaches to modeling and optimizing pit stop strategies are discussed. Next, classical approaches to combinatorial optimization are reviewed. This is followed by an overview of QUBO as a modeling framework and its applications in similar domains. The chapter concludes with a discussion of the research gap addressed in this thesis.

3.1 Optimization of Pit Stop Strategies in Formula 1

Existing research on Formula 1 pit stop strategies primarily focuses on modeling the problem as a sequential decision process. A well-established approach is dynamic programming, where optimal pit stop timing is determined based on tire degradation and race length. Carrasco Heine and Thraves [CHT23]show that, under deterministic assumptions, such models can be solved efficiently for simplified race scenarios.
To capture more realistic race conditions, later work incorporates interactions between drivers. Aguad and Thraves [AT24] extend the problem with game-theoretic elements, demonstrating that optimal strategies depend on the anticipated actions of competitors. This significantly increases the complexity of the decision problem.
Other approaches move away from exact optimization. Simulation-based methods are widely used to evaluate strategies under uncertainty, for example by modeling stochastic race events or varying track conditions [VVWV10]. In addition, optimal control formulations describe strategy as a continuous-time decision problem, allowing for more detailed modeling of performance trade-offs.

More recently, data-driven techniques have been explored to support strategic decisions using historical race data.
While these approaches provide valuable insights, they are typically tailored to a specific modeling framework. As a result, they offer limited flexibility when comparing different optimization methods within a unified representation.

3.2 Combinatorial Optimization and Modeling Approaches

Different modeling approaches for combinatorial optimization problems mainly vary in how they structure the decision space and handle constraints.
Dynamic programming is effective when the problem has a clear sequential structure, as is the case for pit stop strategies. However, the size of the state space grows rapidly when additional factors such as weather or traffic are included, making the approach difficult to scale.
Integer linear programming (ILP) offers a more general and structured way to model decision problems. It allows constraints and objectives to be expressed explicitly, but solving large instances can become computationally expensive, especially when the model includes many interacting components [Pap98].
Heuristic methods take a different perspective by focusing on solution quality rather than optimality. These methods are often better suited for large or complex instances, but their performance depends on the chosen representation of the problem and the design of the search process.[KGV83]

3.3 QUBO and Quadratic Optimization Models

Quadratic Unconstrained Binary Optimization (QUBO) has emerged as a general modeling approach for a wide range of combinatorial optimization problems. Instead of developing problem-specific formulations, QUBO focuses on encoding decision variables and constraints into a quadratic objective function [KHG+14, GKD22].
This modeling perspective has been applied to problems such as scheduling, routing, and resource allocation, where complex constraints can be incorporated through penalty terms. The resulting formulation provides a compact representation that captures both objective contributions and interactions between decisions.
In contrast to traditional approaches such as dynamic programming or ILP, QUBO separates the modeling of the problem from the choice of solver . This makes it possible to evaluate different optimization techniques on the same underlying formulation, without modifying the model itself[KHG+14, GKD22].
In this thesis a QUBO model for the Formula 1 pit stop strategy problem will be formulated and its performance will be evaluated using classical and quantum-inspired annealing methods.

4 QUBO Model and Objective Function

This chapter presents the Quadratic Unconstrained Binary Optimization (QUBO) formulation used in this thesis. The model is built in a step-by-step way. It starts with a basic formulation of the pit stop problem and is then extended with additional components that capture important aspects of

a Formula 1 race. This structure keeps the model transparent and makes it possible to evaluate the effect of each extension separately.

4.1 Overview of the Modeling Approach

The goal is to find the sequence of compounds that minimizes total race time while satisfying the relevant sporting rules. On each lap, exactly one tire compound must be selected. Since the decision variables are binary and the problem is discrete, it can be expressed naturally as a QUBO model. The model uses binary variables for lap-by-lap compound assignment and a small number of auxiliary variables for global or temporal structure. In the later models, extra variables are introduced only when they are needed to represent effects that cannot be captured directly in the base formulation. At a high level, the objective has two main parts. The first reflects lap time. The second reflects the cost of pitting. In the base formulation, pit stops are not counted explicitly. Instead, the model assigns a cost to changes in compound usage between consecutive laps. Later extensions modify these costs to reflect race context more accurately. Throughout this chapter, $H(\mathbf{z})$ denotes the scalar energy function that assigns a total cost to each candidate solution $\mathbf{z}$, and whose minimum corresponds to the optimal race strategy.
The model assumes that all relevant inputs are known in advance. This includes lap-time parameters, weather conditions, traffic effects, and safety car periods. The result is a deterministic formulation that is suitable for controlled comparison between solvers.

4.2 Base Model (Model 0)

Model 0 provides the basic QUBO formulation of the pit stop strategy problem. It includes the core assignment decision and the minimum dry-compound rule, but no race-context effects. This makes it the simplest valid version of the model and a natural starting point for the later extensions.
The main decision variables are $x_{l,c} \in \{0,1\}$, where $x_{l,c} = 1$ if compound c is used on lap l. In addition, usage indicators $u_c \in \{0,1\}$ are introduced to represent whether a compound is used at least once during the race. A slack variable $s \in \{0,1\}$ is used to express the minimum-compound rule in quadratic form. The total number of variables is therefore $N|C| + |C| + 1$.
The first constraint ensures that exactly one compound is selected on each lap:

$$H_A = P_A \sum_l \left(1 - \sum_c x_{l,c}\right)^2, \tag{1}$$

where $P_A > 0$ is a penalty weight that enforces the constraint by making violations costly in the objective.
The second constraint links the lap variables to the usage indicators:

$$H_B = P_B \sum_c \sum_l x_{l,c}(1 - u_c) + 0.2\, P_B \sum_c u_c. \tag{2}$$

The additional linear term on u_c discourages the solver from setting usage indicators to one when a compound is not actually used.

The third constraint enforces the rule that at least two dry compounds must be used:

$$H_C = P_C \left(\sum_c u_c - s - 2 \right)^2 . \tag{3}$$

The racing objective consists of a lap-time term and a pit-stop term. The lap-time term is

$$H_{\text{lap}} = \sum_l \sum_c T_c x_{l,c}, \tag{4}$$

where T_c is the base lap time associated with compound c.
The pit-stop term is

$$H_{\text{pit}} = \Delta_{\text{pit}} \left[(N-1) - \sum_c \sum_{l=0}^{N-2} x_{l,c} x_{l+1,c} \right]. \tag{5}$$

This term rewards continuity between consecutive laps and therefore penalizes compound changes, which correspond to pit stops in the model.

4.3 Model Extension 1: Weather Effects (Model 1)

Model 1 adds weather as the first race-context extension. This is necessary because tire legality and tire performance both depend on whether the race is dry or wet. In wet conditions, the rule that at least two compounds must be used no longer applies, so the base model is not sufficient.
A race-level parameter weather $\in \{\text{dry}, \text{wet}\}$ is introduced. It is fixed before the QUBO is constructed and applies to the whole race. Lap-by-lap weather changes are not considered in this model.
When the race is wet, the set of available compounds is expanded from $\{S, M, H\}$ to $\{I, W, S, M, H\}$, where I denotes Intermediate and W denotes Full Wet. This introduces additional assignment variables and usage indicators for the wet compounds. In the dry case, the structure remains unchanged.
The lap-time objective is modified accordingly, where T_I and T_W denote the base lap times for the intermediate and full wet compounds, and T_c^{wet} denotes the penalised lap-time coefficient for dry compound c used under wet conditions:

$$\sum_l \left(T_I x_{l,I} + T_W x_{l,W} + T_S^{\text{wet}} x_{l,S} + T_M^{\text{wet}} x_{l,M} + T_H^{\text{wet}} x_{l,H} \right). \tag{6}$$

In wet conditions, the dry compounds receive substantially worse lap-time coefficients than the wet compounds, which makes inappropriate tire choices unattractive.
The compound-usage constraint is handled differently depending on the weather. In dry races, H_C remains active. In wet races, it is omitted completely, since the regulation no longer applies. As a consequence, the slack variable is not needed in wet instances.

4.4 Model Extension 2: Traffic Effects (Model 2)

Model 2 adds traffic effects to the pit-stop cost. In a real race, the time lost by pitting depends not only on the time spent in the pit lane, but also on the circumstances of the rejoin. Rejoining into heavy traffic can make a pit stop much less attractive.

Traffic is represented by a deterministic lap-dependent parameter $\tau_{l+1} \geq 0$, which denotes the additional loss associated with rejoining the track on lap $l+1$. No new variables are introduced. The effective pit-stop cost becomes

$$\Delta_{\text{eff}}(l) = \Delta_{\text{pit}} + \tau_{l+1}. \tag{7}$$

Using this coefficient, the pit-stop term is written as

$$H_{\text{pit}} = \sum_{l=0}^{N-2} \Delta_{\text{eff}}(l) \sum_{\substack{c_1,c_2 \in C \\ c_1 \neq c_2}} x_{l,c_1} x_{l+1,c_2}. \tag{8}$$

This form penalizes compound changes directly, with a penalty that depends on the lap on which the stop occurs. If traffic is high on a certain lap, pitting there becomes less attractive.
This extension does not change the size of the QUBO, but it does make the energy landscape lap-dependent. As a result, the model can distinguish between more and less favorable pit windows.

4.5 Model Extension 3: Fuel Load Effect (Model 3)

Model 3 includes the effect of fuel burn on lap times. At the start of the race, the car is heavier and therefore slower. As the race progresses and fuel is consumed, lap times improve. Without this effect, all laps are treated too uniformly.
The fuel effect is modeled as a linear improvement over the race distance. For each lap l, a gain proportional to l is applied using a parameter $\gamma > 0$, which represents the time gained per lap due to fuel consumption. Since fuel load does not depend on the tire compound, the same correction applies to every $x_{l,c}$ on lap l.
The additional term is

$$H_{\text{fuel}} = -\gamma \sum_{l=0}^{N-1} l \sum_{c \in C} x_{l,c}. \tag{9}$$

This contributes a negative diagonal value to the QUBO, with larger reductions on later laps. The effect is modest compared to the cost of a pit stop, but it improves the realism of the total race-time estimate and can influence the balance between early and late stops.

4.6 Model Extension 4: Safety Car Conditions (Model 4)

Model 4 adds Safety Car (SC) and Virtual Safety Car (VSC) conditions. These periods reduce the time loss of a pit stop, which often makes them strategically important.
Two lap-indexed binary input arrays are used: sc_l and vsc_l. Based on these inputs, a benefit β_l is assigned to each lap:

$$\beta_l = \begin{cases} \beta_{\text{SC}} & \text{if } \text{sc}_l = 1, \\ \beta_{\text{VSC}} & \text{if } \text{vsc}_l = 1 \text{ and } \text{sc}_l = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

This benefit reduces the effective pit-stop cost:

$$\Delta_{\text{eff}}(l) = \Delta_{\text{pit}} - \beta_l + \tau_{l+1}. \tag{11}$$

The pit-stop term keeps the same structure as in Model 2:

$$H_{\text{pit}} = \sum_{l=0}^{N-2} \Delta_{\text{eff}}(l) \sum_{\substack{c_1,c_2 \in C \\ c_1 \neq c_2}} x_{l,c_1} x_{l+1,c_2}. \tag{12}$$

The effect of this extension is straightforward. On laps with SC or VSC conditions, compound changes become cheaper, so pit stops on those laps are favored by the objective. The model size does not change, but the race context becomes richer.

4.7 Model Extension 5: Tire Degradation — Uniform (Model 5)

Model 5 introduces tire degradation. This is the first extension that requires a larger structural change to the QUBO, since tire age must be represented explicitly. Degradation is important because it creates the central trade-off in pit strategy: longer stints avoid pit-stop losses but become slower over time.

A direct one-hot encoding of tire age would lead to many variables and constraints. Instead, a chain-based encoding is used. The first new variables are continuation variables $w_{l,c} \in \{0, 1\}$, defined by

$$w_{l,c} = x_{l,c} x_{l+1,c}. \tag{13}$$

These variables indicate whether the same compound is used on two consecutive laps.

On top of this, chain variables $R_{l,c,q} \in \{0, 1\}$ are introduced for $q \geq 2$. A variable $R_{l,c,q}$ is one if compound c has been used for at least q consecutive laps ending on lap l. Their recursive structure is given by

$$R_{l,c,q} = w_{l-1,c} R_{l-1,c,q-1}. \tag{14}$$

This construction encodes tire age implicitly. If a stint continues, the chain extends. If a pit stop occurs, the continuation variable becomes zero and the chain resets automatically.

The continuation variables are enforced by the quadratic penalty

$$H_{D3a} = \lambda_D \sum_{l,c} \left(3w_{l,c} + x_{l,c} x_{l+1,c} - 2x_{l,c} w_{l,c} - 2x_{l+1,c} w_{l,c}\right). \tag{15}$$

The recursive chain structure is enforced by

$$H_{D3b} = \lambda_D \sum_{c} \sum_{q=2}^{K_c} \sum_{l=q}^{N-1} \left(3R_{l,c,q} + R_{l,c,1} R_{l-1,c,q-1} - 2R_{l,c,1} R_{l,c,q} - 2R_{l-1,c,q-1} R_{l,c,q}\right). \tag{16}$$

The degradation cost is then added as a linear term:

$$H_{\text{deg}} = \sum_{c \in C} \sum_{l=0}^{N-1} d \sum_{q=1}^{K_c} R_{l,c,q}, \tag{17}$$

where d is a uniform degradation rate shared by all compounds.

This extension increases the number of variables and couplings substantially. In return, it allows the model to represent the timing trade-off that is central to pit-stop strategy.

4.8 Model Extension 6: Compound-Specific Tire Degradation (Model 6)

Model 6 replaces the uniform degradation rate by compound-specific degradation. This is necessary because Soft, Medium, and Hard tires do not wear at the same rate and are not intended for the same stint lengths.
The structure introduced in Model 5 is kept unchanged. The difference is that each compound now has its own degradation rate d_c and its own maximum chain depth K_c. This makes the degradation behavior compound-dependent without introducing new variable types or new constraint families. The degradation term becomes

$$H_{\text{deg}} = \sum_{c \in C} d_c \sum_{l=0}^{N-1} \sum_{q=1}^{K_c} R_{l,c,q}. \tag{18}$$

Long stints on soft compounds therefore receive a larger penalty than equally long stints on hard compounds. This gives the optimizer a real distinction between compounds: softer tires become attractive for shorter stints, while harder tires become more attractive for longer ones.
The chain constraints remain the same as in Model 5, except that the index range now depends on the compound. The price of this extra realism is a further increase in QUBO size, especially for compounds with larger values of K_c.

4.9 Final QUBO Formulation

The final model combines all objective components and all active penalty terms into a single quadratic function over binary variables:

$$\min_{\mathbf{z} \in \{0,1\}^n} \; H(\mathbf{z}) = H_{\text{lap}} + H_{\text{pit}} + H_{\text{fuel}} + H_{\text{deg}} + H_A + H_B + H_C + H_D. \tag{19}$$

The lap-time term is

$$H_{\text{lap}} = \sum_{l=0}^{N-1} \sum_{c \in \mathcal{C}} t_c \, x_{l,c}. \tag{20}$$

The pit-stop term is

$$H_{\text{pit}} = \sum_{l=0}^{N-2} P_{\text{pit}}(l) \left(1 - \sum_{c \in \mathcal{C}} x_{l,c} x_{l+1,c} \right), \tag{21}$$

where

$$P_{\text{pit}}(l) = \Delta_{\text{pit}} + \tau_{l+1} - \beta_l. \tag{22}$$

The fuel term is

$$H_{\text{fuel}} = -\sum_{l=0}^{N-1} \sum_{c \in \mathcal{C}} l \gamma x_{l,c}. \tag{23}$$

The compound-specific degradation term is

$$H_{\text{deg}} = \sum_{c \in \mathcal{C}} d_c \sum_{l=0}^{N-1} \sum_{q=1}^{K_c} R_{l,c,q}. \tag{24}$$

The constraint terms are

$$H_A = P_A \sum_{l=0}^{N-1} \left(1 - \sum_{c\in\mathcal{C}} x_{l,c}\right)^2, \tag{25}$$

$$H_B = P_B \sum_{l=0}^{N-1} \sum_{c\in\mathcal{C}} x_{l,c}(1-u_c) + 0.2\, P_B \sum_{c\in\mathcal{C}} u_c, \tag{26}$$

$$H_C = P_C \left(\sum_{c\in\mathcal{C}} u_c - s - 2\right)^2, \tag{27}$$

where H_C is omitted in wet race conditions, and

$$H_D = H_{D3a} + H_{D3b}. \tag{28}$$

The continuation constraints are

$$H_{D3a} = P_D \sum_{l=0}^{N-2} \sum_{c\in\mathcal{C}} \left(3w_{l,c} + x_{l,c}x_{l+1,c} - 2x_{l,c}w_{l,c} - 2x_{l+1,c}w_{l,c}\right), \tag{29}$$

and the recursive chain constraints are

$$H_{D3b} = P_D \sum_{c\in\mathcal{C}} \sum_{q=2}^{K_c} \sum_{l=q}^{N-1} \left(3R_{l,c,q} + w_{l-1,c}R_{l-1,c,q-1} - 2w_{l-1,c}R_{l,c,q} - 2R_{l-1,c,q-1}R_{l,c,q}\right). \tag{30}$$

Together, these terms define the final QUBO:

$$\begin{aligned}
\min_{\mathbf{z}\in\{0,1\}^n} H(\mathbf{z}) = {} & \sum_{l=0}^{N-1} \sum_{c\in\mathcal{C}} t_c\, x_{l,c} + \sum_{l=0}^{N-2} P_{\text{pit}}(l) \left(1 - \sum_{c\in\mathcal{C}} x_{l,c}x_{l+1,c}\right) \\
& - \sum_{l=0}^{N-1} \sum_{c\in\mathcal{C}} l\,\gamma\, x_{l,c} + \sum_{c\in\mathcal{C}} d_c \sum_{l=0}^{N-1} \sum_{q=1}^{K_c} R_{l,c,q} \\
& + P_A \sum_{l=0}^{N-1} \left(1 - \sum_{c\in\mathcal{C}} x_{l,c}\right)^2 \\
& + P_B \sum_{l=0}^{N-1} \sum_{c\in\mathcal{C}} x_{l,c}(1-u_c) + 0.2\, P_B \sum_{c\in\mathcal{C}} u_c \\
& + P_C \left(\sum_{c\in\mathcal{C}} u_c - s - 2\right)^2 \\
& + P_D \sum_{l=0}^{N-2} \sum_{c\in\mathcal{C}} \left(3w_{l,c} + x_{l,c}x_{l+1,c} - 2x_{l,c}w_{l,c} - 2x_{l+1,c}w_{l,c}\right) \\
& + P_D \sum_{c\in\mathcal{C}} \sum_{q=2}^{K_c} \sum_{l=q}^{N-1} \left(3R_{l,c,q} + w_{l-1,c}R_{l-1,c,q-1} - 2w_{l-1,c}R_{l,c,q} - 2R_{l-1,c,q-1}R_{l,c,q}\right).
\end{aligned} \tag{31}$$

4.10 Penalty Weights and Scaling

Because the constraints are enforced through penalties, the choice of penalty weights is important. Each penalty must be large enough to make constraint violations unattractive, but not so large that the whole objective becomes dominated by penalty structure alone.

For the assignment constraint, the penalty must exceed any possible gain from choosing an invalid lap configuration. The value $P_A = 1000$ was determined empirically and verified to exceed the largest relevant objective difference in the tested instances, so this constraint is strongly enforced in practice.

For the usage-indicator constraint, the penalty must ensure that the variables u_c remain consistent with the actual tire assignments. This is why $P_B = 5000$ is chosen as the largest penalty. The additional linear term on u_c supports this by discouraging artificial activation of unused compounds.

For the minimum-compound constraint, the penalty must be larger than the benefit of avoiding a pit stop. Since the pit-stop cost is much smaller than the chosen value $P_C = 800$, violating this rule is not attractive.

For the degradation-chain constraints, the penalty must dominate the degradation reward associated with satisfying or violating a product relation. The chosen value $P_D = 2000$ is much larger than any single degradation contribution and therefore preserves the intended chain structure.

The resulting hierarchy is

$$P_B > P_D > P_A > P_C.$$

This ordering reflects the role of the constraints in the model. Logical consistency must be enforced first, then chain integrity, then the per-lap assignment, and finally the minimum-compound rule.

All four penalty values were determined empirically on the base model and kept fixed throughout all experiments. A partial sensitivity analysis on P_C is reported in Section 6, confirming that feasibility outcomes for Models 0–4 are stable across a wide range of that parameter. The remaining penalties were not subjected to systematic tuning, which is a limitation discussed further in Section 7.

Using the same penalty weights for both solvers ensures that the problem definition is identical across solvers, but it does not eliminate penalty configuration as a potential source of performance differences, since the two solvers may respond differently to the same landscape.

5 Experiments

This chapter evaluates both the QUBO formulation of the Formula 1 pit stop strategy problem and the performance of the optimization methods applied to it. The experiments address the research question by analysing how effectively the problem can be solved using simulated annealing (SA) and quantum-inspired annealing (QIA). Also the implementation of the solvers is explained.

5.1 Overview

The experimental setup consists of a structured grid over models, solvers, problem sizes, and random seeds. Seven models (M0–M6) are evaluated using SA and QIA across multiple race lengths, with five independent seeds per configuration, resulting in 295 runs.

For Models 0–4, SA is evaluated at $N \in \{5, 10, 20, 30, 50\}$, while QIA is limited to $N \in \{5, 10, 20\}$ due to its higher computational cost. For Models 5 and 6, both solvers are evaluated up to $N = 20$, as the number of variables increases substantially.

Baseline methods are included for reference. Brute-force enumeration is applied to small instances ($N \leq 10$), and a greedy heuristic is evaluated at all sizes.
All experiments use identical model parameters and penalty weights. No solver-specific tuning is applied. Differences in performance reflect the behavior of the optimization methods on this specific formulation, though implementation differences mean that observed gaps cannot be attributed to algorithmic differences alone. Due to differences in algorithm design, runtime comparisons between SA and QIA must be interpreted carefully, as QIA consistently requires more computation per run.

5.2 Problem Instances

Each problem instance is defined by (m, N, C, s), where m is the model version, N the number of laps, $C = \{S, M, H\}$ the compound set, and s the random seed. Base lap times are fixed at $t_S = 90.0$, $t_M = 91.0$, and $t_H = 92.5$ seconds, with a pit stop time of $\Delta_{\text{pit}} = 20.0$ seconds.
Problem sizes are chosen to cover different regimes. Small instances ($N = 5, 10$) allow for exact solutions and are used for validation. Medium instances ($N = 20, 30$) form the main comparison range. Larger instances ($N = 50$) are included for scalability analysis but only evaluated with SA. Models 5 and 6 introduce chain-based encodings, which significantly increase the number of variables. At $N = 20$, these models already contain around 630 variables, making larger instances computationally impractical.
All experiments are repeated using five fixed seeds $\{42, 123, 456, 789, 1011\}$. Baseline methods are deterministic and executed once per instance. Direct comparisons between SA and QIA are restricted to the shared range $N \in \{5, 10, 20\}$.

5.3 Solvers

Two stochastic solvers are used: simulated annealing (SA) and quantum-inspired annealing (QIA).

Simulated Annealing SA is implemented using the `neal` library. Each solver call performs 200 independent runs, returning the best solution. The number of sweeps scales with problem size as

$$n_{\text{sweeps}} = \max(2000,\ 10 \cdot n_{\text{vars}}).$$

A default temperature schedule is used.

Quantum-Inspired Annealing QIA is implemented as a simulated quantum annealing method using 8 coupled replicas. The number of sweeps is given by

$$n_{\text{sweeps}} = \max(1500,\ 6 \cdot n_{\text{vars}}).$$

The annealing schedule combines a decreasing transverse field and an adaptive temperature scaling. A single solution is returned per run.

Baseline Methods Brute-force enumeration is used for $N \leq 10$ to obtain the exact optimum. A greedy heuristic is applied at all sizes as a simple reference. Both operate directly on the combinatorial problem.

5.4 Evaluation Metrics

Solver performance is evaluated using feasibility, solution quality, optimality gap, and runtime.

Feasibility A solution is feasible if all hard constraints are satisfied. The feasibility rate is defined as

$$\mathrm{FR} = \frac{n_{\text{feasible}}}{n_{\text{runs}}}.$$

Infeasible solutions are not considered in further analysis.

Solution Quality The total race time is computed as

$$T_{\text{race}} = \sum_{l=0}^{N-1} t_{c_l} + n_{\text{stops}} \cdot \Delta_{\text{pit}}.$$

Mean and best values are reported.

Optimality Gap For $N \leq 10$, the optimality gap is defined as

$$\text{gap} = \frac{T_{\text{solver}} - T_{\text{optimal}}}{T_{\text{optimal}}}.$$

Runtime Runtime is measured as wall-clock time per solver call. It is used to assess scalability rather than efficiency.

5.5 Model and Solver Configuration

Penalty weights are defined as

$$P_A = 1000, \quad P_B = 5000, \quad P_C = 800, \quad P_D = 2000.$$

Solver configurations scale with problem size. No parameter tuning, warm-starting, or post-processing is applied.
For Models 5 and 6, the maximum tire age is capped at $\min(K_c, N-1)$ to reduce unnecessary variables in small instances.

5.6 Experiment Design

The experiments consist of three components.
The primary campaign evaluates all models and solvers across the defined problem sizes and seeds, forming the main dataset used in Chapter 6.
Baseline experiments provide exact solutions for small instances and reference performance across all sizes.
Additional sensitivity experiments verify that the chosen parameter settings are stable and do not bias the results.

5.7 Implementation and Reproducibility

All experiments are implemented in Python using `dwave-neal`, `dimod`, and standard scientific libraries. The QIA solver is implemented within the codebase.
Each run is stored in a SQLite database, including all parameters and results. This ensures that every experiment can be reproduced exactly.
Randomness is fully controlled through fixed seeds. Re-running a configuration with the same parameters produces identical results.
The experiment pipeline is fully automated and resumable. Completed runs are not repeated, and all results are generated from a single consistent dataset.
The codebase is organized into modules for models, solvers, experiments, and analysis. A test suite verifies the correctness of key components.

6 Results and Analysis

This chapter presents the results of the experiments and analyses them with respect to the research question. The evaluation focuses on three aspects: feasibility, solution quality, and runtime scalability.

6.1 Feasibility Analysis

Feasibility is the primary evaluation criterion, since only constraint-satisfying solutions represent valid race strategies.
For Models 0–4, the two solvers show clearly different behavior at small instance sizes. At $N = 5$, SA produces no feasible solutions across all runs. This behavior is consistent across seeds, indicating convergence to a lower-energy infeasible configuration rather than a stochastic failure. The dominant infeasible solution is a single-compound strategy without a pit stop, which minimizes lap time but violates the compound-use constraint.
As the number of laps increases, the feasibility of SA improves. At $N = 10$, feasibility ranges between 20% and 60%, depending on the model. At $N \geq 20$, SA reaches 100% feasibility and maintains this level for larger instances. The increase in horizon length raises the penalty cost of constraint violations, shifting the energy landscape toward feasible solutions.
QIA shows a different pattern. For Models 0–4, it achieves high feasibility across all tested sizes, including $N = 5$, where it reaches 86% to 92%. At larger values of N, feasibility approaches 100%. This indicates that QIA is able to escape the single-compound local minimum that traps SA at small instance sizes.

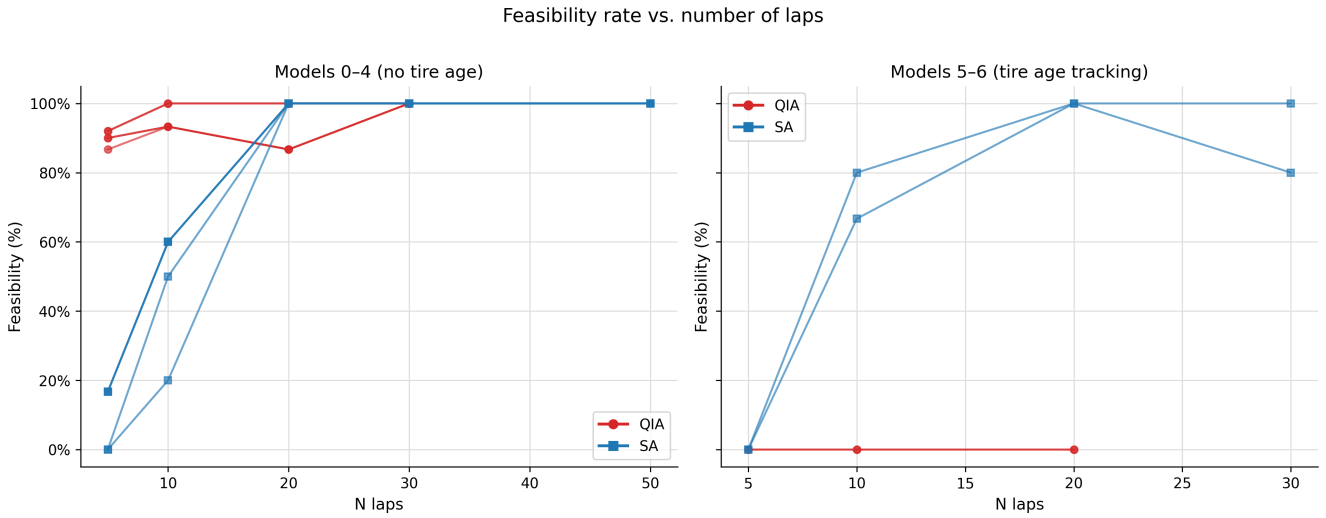

Figure 1: Feasibility rate as a function of the number of laps for Models 0–4.

The behavior changes significantly when tire-age modeling is introduced in Models 5 and 6. At $N = 5$, both solvers fail to produce feasible solutions. The additional chain variables required to track tire age introduce many dependencies that must be satisfied simultaneously, making the constraint structure harder to satisfy.
SA partially recovers as the problem size increases. At $N = 10$, feasibility reaches 67% to 80%, and at $N = 20$ it reaches 100%. QIA does not recover and produces no feasible solutions at any of the tested sizes ($N \in \{5, 10, 20\}$) for Models 5 and 6.

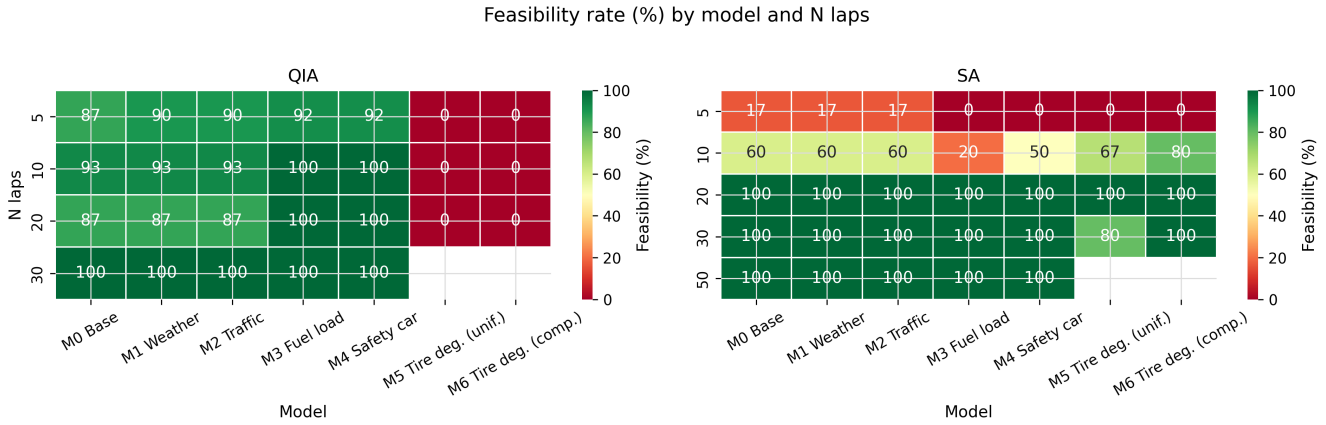

Figure 2: Feasibility rates across all models and instance sizes for both solvers.

Overall, QIA is more effective at achieving feasibility in small instances of simpler models, while SA becomes reliable as the problem size increases. For the tire-age models, only SA remains effective.

6.2 Solution Quality

Solution quality is evaluated only for feasible solutions, ensuring that all reported race times correspond to valid strategies.

For the smallest instances, an exact comparison with deterministic baselines is possible. In Model 0, the brute-force optimum equals the greedy solution at both $N = 5$ and $N = 10$, confirming that the greedy heuristic is optimal in these cases.

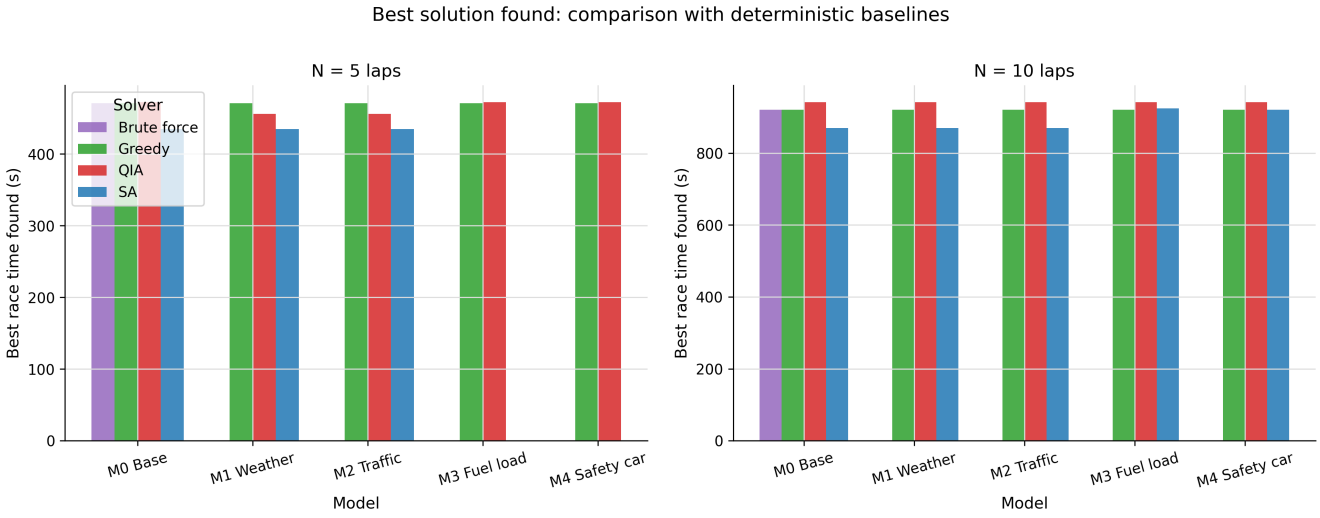

Figure 3: Best solution found compared to brute-force and greedy baselines for $N = 5$ and $N = 10$.

At $N = 5$, SA rarely produces feasible solutions and is therefore not comparable. QIA finds feasible solutions in most runs. Its best solutions are close to the optimum, while the average remains worse, indicating convergence to near-optimal configurations.
At $N = 10$, both solvers can be compared directly. SA achieves lower race times than QIA, both on average and in the best case. Once feasibility is reached, SA is more effective at minimizing the objective.
At $N = 20$, both solvers achieve high feasibility for Models 0–4. SA consistently produces lower race times than QIA, both in average and best solutions.

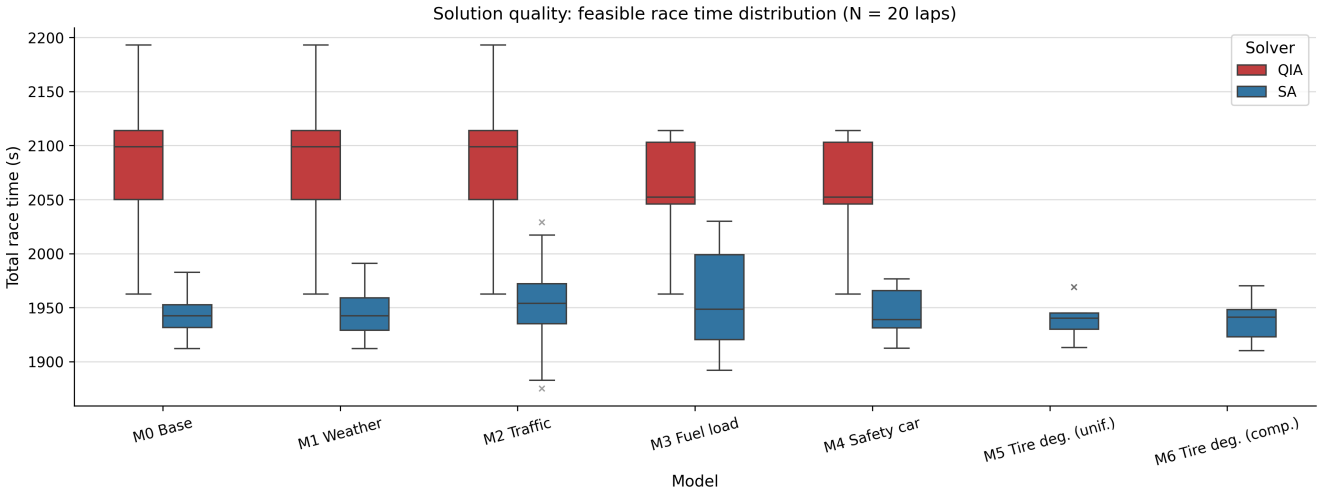

Figure 4: Distribution of feasible race times at $N = 20$ for all models.

At $N = 20$, both solvers produce solutions that are on average worse than the greedy strategy. One possible explanation is that the large penalty weights required to enforce feasibility flatten the objective landscape around feasible solutions, reducing the energy difference between strategies of different quality and making it harder for annealing methods to distinguish between them. However, this §12w2qexplanation is not directly verified by the experiments.
For Models 5 and 6, solution quality can only be evaluated for SA. At $N = 10$, the average race time increases due to degradation costs. At $N = 20$, the average is similar to earlier models, suggesting a balance between degradation effects and pit-stop decisions.
Overall, SA outperforms QIA in solution quality whenever feasible solutions are available.

6.3 Runtime and Scalability

This section evaluates the computational cost of the solvers and how it scales with problem size. For Models 0–4, SA shows near-linear scaling with respect to the number of laps. Runtime increases gradually as the number of variables grows, while the number of sweeps remains at its lower bound. QIA follows a similar scaling trend but with consistently higher runtime. The relative overhead compared to SA remains stable across instance sizes.

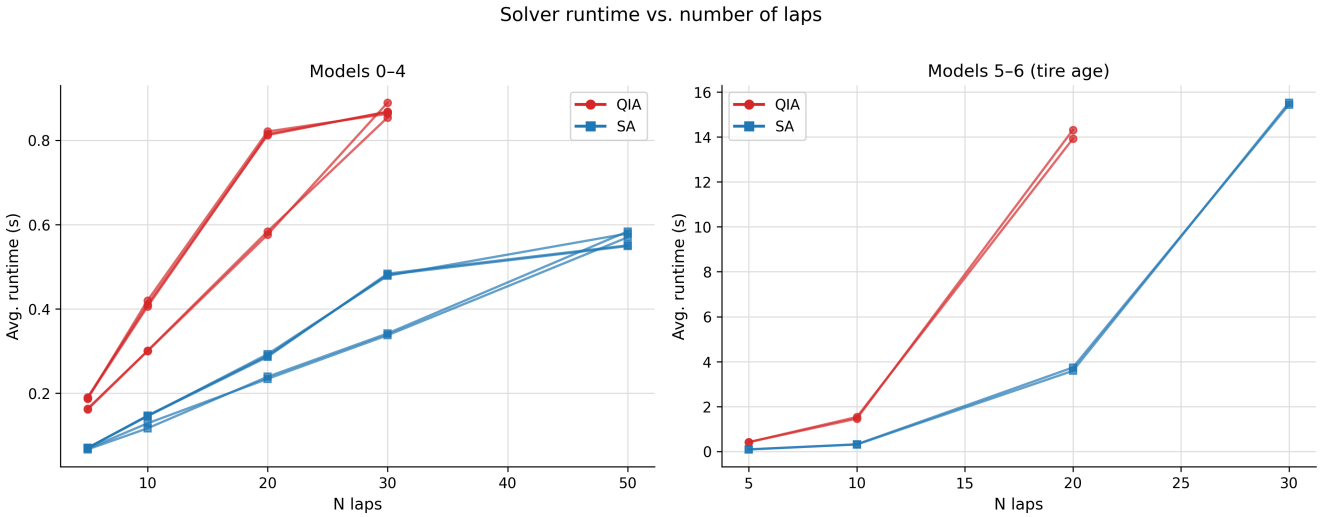

Figure 5: Runtime scaling of SA and QIA for Models 0–4 and Models 5–6.

The runtime ratio between QIA and SA remains approximately constant for Models 0–4, indicating predictable relative performance.

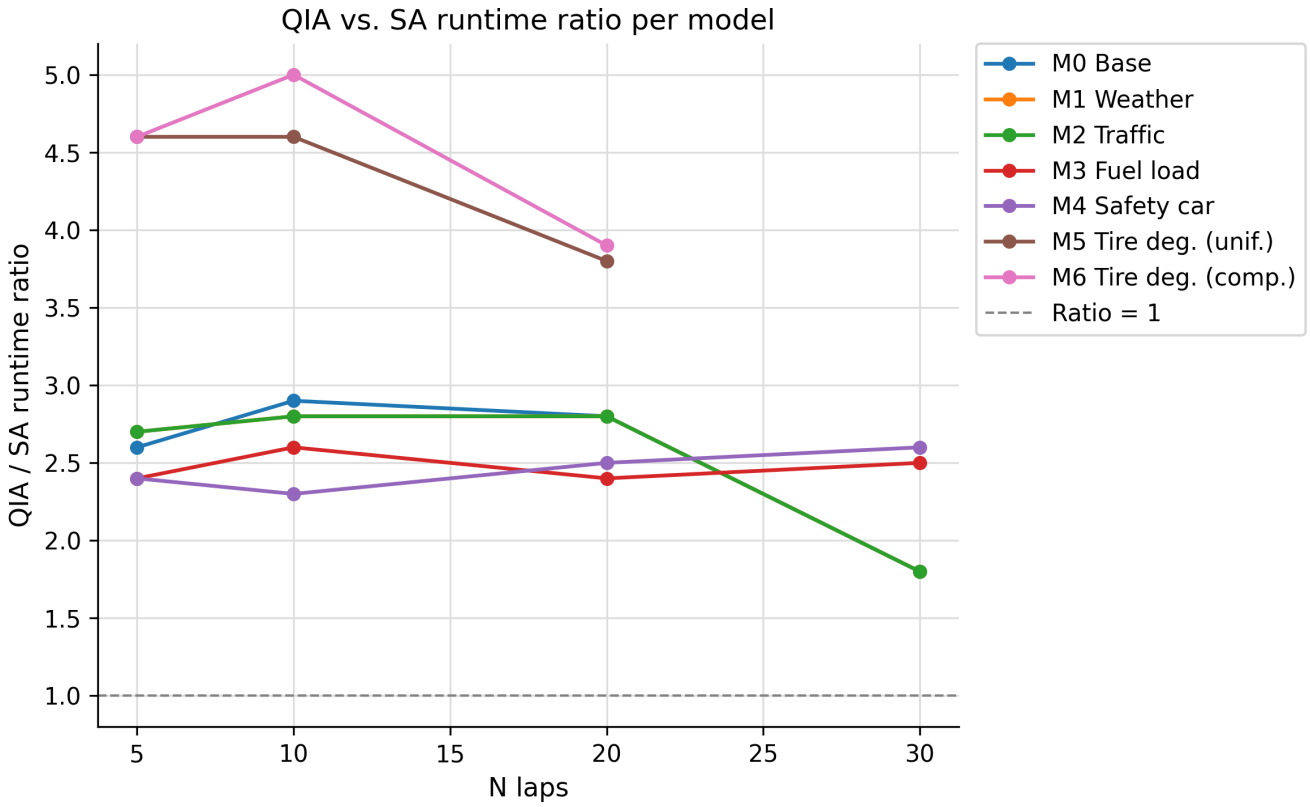

Figure 6: Runtime ratio between QIA and SA across models and instance sizes.

A significant change occurs in Models 5 and 6. Tire-age tracking substantially increases the number of variables and quadratic interactions.

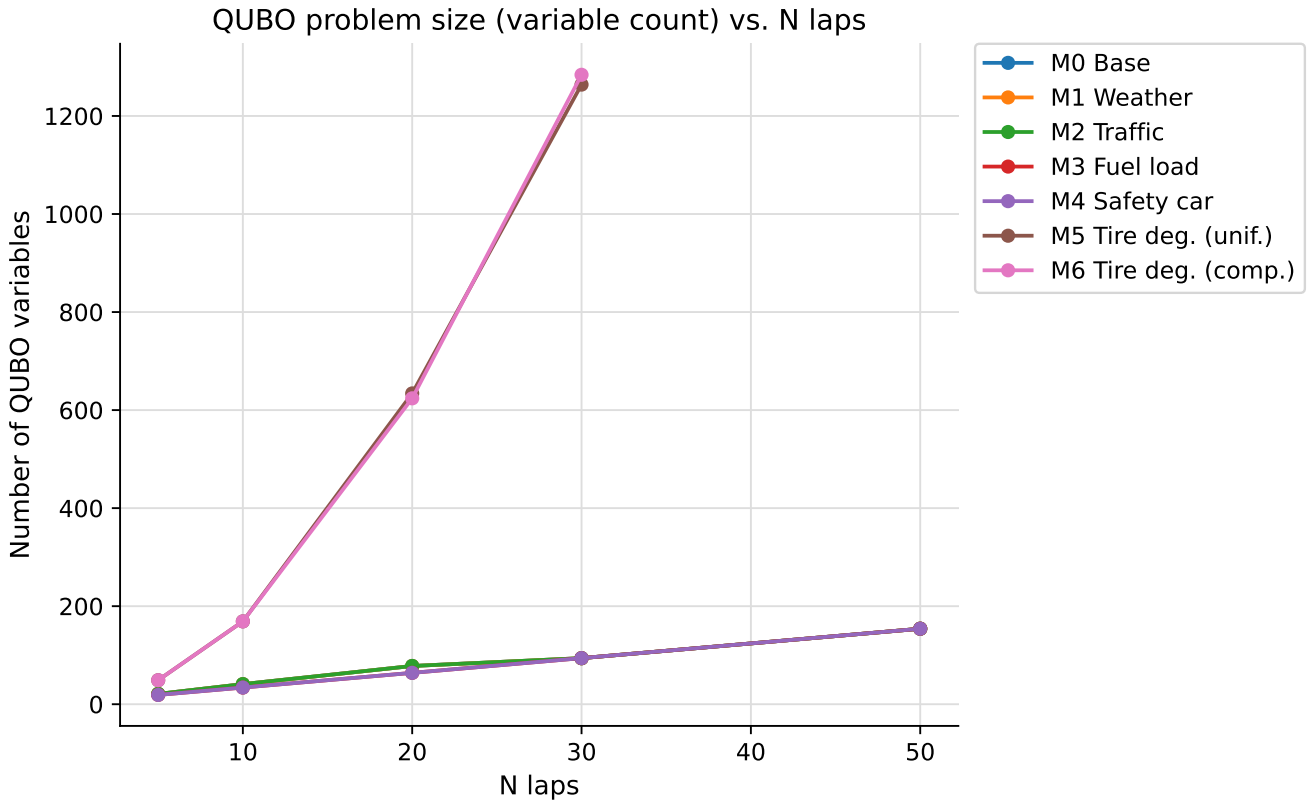

Figure 7: Growth in QUBO variable count as a function of the number of laps.

For SA, this results in a sharp increase in runtime due to adaptive sweep scaling. QIA shows an even stronger increase, with a growing runtime gap compared to SA.
The difference can be explained by the solver structure. SA operates on a single configuration, while QIA maintains multiple coupled replicas, increasing the cost per update step.
Overall, both solvers scale well for Models 0–4. For Models 5–6, SA remains usable with higher cost, while QIA becomes significantly slower and less practical.

6.4 Sensitivity Analysis

To assess whether the chosen penalty weights produce stable results, a sensitivity analysis was conducted for the minimum-compound penalty P_C. The remaining penalties were held fixed at their baseline values ($P_A = 1000$, $P_B = 5000$, $P_D = 2000$), and P_C was varied across the values $\{1000, 2000, 5000, 8000, 10000\}$. The analysis was run at $N = 5$ on Models 0–4.

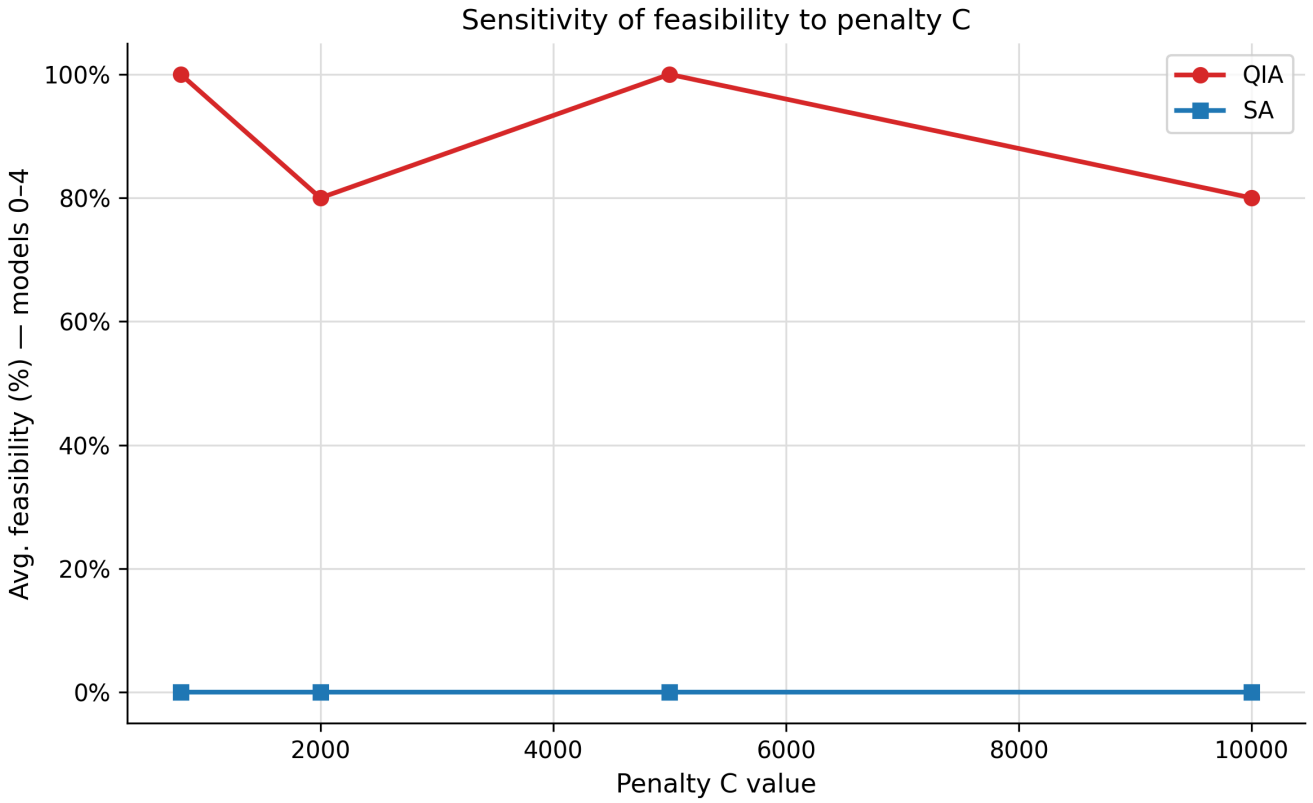

Figure 8: Average feasibility rate across Models 0–4 at $N = 5$ as a function of P_C, with all other penalty weights held at their baseline values.

The results are shown in Figure 8. SA produces no feasible solutions at any tested value of P_C. This is consistent with the main experimental results: as reported in Section 6, SA fails to achieve feasibility at $N = 5$ regardless of model variant, because the energy landscape at this problem size consistently favors a single-compound strategy that violates the compound-use constraint. The P_C penalty governs precisely this constraint, yet varying it across an order of magnitude does not change SA's behavior. This indicates that SA's failure at $N = 5$ is not caused by an insufficient value of P_C, but by the structure of the energy landscape at small instance sizes.

QIA shows a different pattern. Feasibility ranges from 80% to 100% across the tested values, with no monotonic relationship between P_C and feasibility rate. The variation is moderate rather than systematic, suggesting that QIA's performance at $N = 5$ is not highly sensitive to the specific value of P_C within the tested range. The dip at $P_C = 2000$ and the recovery at $P_C = 5000$ do not follow a clear trend and are likely due to the limited number of runs at this problem size rather than a meaningful dependence on the penalty value.

Overall, the sensitivity analysis provides two useful observations. First, the baseline value $P_C = 800$ used in the main experiments is not the cause of SA's small-instance failures. Second, QIA's feasibility at $N = 5$ is broadly stable across a wide range of P_C values, which means the comparison between solvers in the main experiments is not an artifact of the specific penalty chosen. The analysis does not extend to larger N or to the other penalty parameters, which is a limitation: P_B in particular has a stronger influence on the objective landscape and would warrant a similar check in future work.

6.5 Discussion

This section brings together the experimental results in relation to the research question, which asks how effectively the F1 pit stop strategy problem can be formulated as a QUBO model and how SA and QIA perform on that formulation.

QUBO as a modeling framework. The results show that the penalty-based encoding captures the structure of the problem in a consistent way. All seven model variants lead to well-defined quadratic programs, and for Models 0–4 at least one solver finds feasible solutions at every tested problem size. The progression from M0 to M6 illustrates that the formulation is compositional: each additional effect can be included by extending the existing Q matrix, without changing the overall structure. This makes it practical to build the model step by step.

The chain encoding used in Models 5 and 6 is the main structural extension. It allows tire age and compound-specific degradation to be represented within a binary framework, without introducing integer variables. However, this comes at a clear cost. The number of variables grows as $\mathcal{O}(N \cdot |C| \cdot K_{\max})$, compared to $\mathcal{O}(N \cdot |C|)$ for the base model. This is not an artifact of the implementation, but follows directly from tracking time-dependent state in a QUBO setting. At $N = 20$, Models 5 and 6 already reach around 630 variables, with a corresponding increase in interaction density.

A related limitation is the choice of penalty weights. These were selected manually, and their effect is only partly understood. The sensitivity analysis shows that varying P_C over an order of magnitude does not change feasibility in a meaningful way, which suggests that this parameter is not the main bottleneck. In contrast, $P_B = 5000$ was required to dominate the objective at larger N, but was not analysed in the same systematic way. Such large penalties tend to flatten the landscape around feasible solutions, making it harder for solvers to distinguish between strategies of different quality.

Solver comparison. The comparison between SA and QIA shows a clear difference that depends on both problem size and model complexity. For small instances ($N = 5$–10) on Models 0–2, QIA achieves higher feasibility than SA (87–93% versus 17–60%). This advantage is visible, but limited in practice. On feasible runs, QIA consistently produces higher race times than SA under the same settings. In other words, QIA finds feasible solutions more often at small N, while SA finds better ones when it does.

At $(M0, N = 20)$, for example, SA reaches an average race time of 1942 s, compared to 2082 s for QIA. Part of this difference comes from the solver setup. SA performs 200 independent restarts and returns the best result, while QIA follows a single annealing trajectory. This gives SA a broader coverage of the feasible region once the problem is large enough that feasibility is regularly reached. For Models 5 and 6, QIA does not produce any feasible solutions across all 130 runs. This does not seem to be only a matter of tuning. The dense constraint structure leads to strong inter-replica couplings $J_\perp$, which dominate thermal fluctuations and cause the system to freeze before it can explore the feasible region. SA handles this partly through adaptive sweep scaling, reaching 66–80% feasibility on Model 5 at $N = 10$–20, although with a higher runtime (3.5–15.5 s per run). The failure of QIA here points to a structural limitation of SQA-style methods on densely constrained QUBOs.

Solution quality and the role of the greedy baseline. Neither solver consistently reaches global optima at larger sizes. The greedy baseline outperforms both methods in terms of average race time across all models and instance sizes. This can be traced back to the use of large penalty weights. In particular, $P_B = 5000$ enforces feasibility but also reduces the relative differences between feasible solutions. As a result, once a solver reaches feasibility, it becomes difficult to further improve the objective value. This shows that feasibility alone is not sufficient as a performance metric, and that penalty design directly affects solution quality.

Implications. Overall, the results indicate that QUBO is a workable modeling framework for this problem at moderate race lengths. At the same time, solver performance depends strongly on the specific model variant. SA is the more reliable option across the tested range. QIA is competitive only for small and simple instances, and breaks down on the more complex models. It remains open whether improved tuning or access to quantum annealing hardware would change this picture, but the variable counts of Models 5 and 6 (634–1 284 at $N = 20$–30) already approach the limits of current systems.

7 Conclusions and Further Research

This chapter answers the research question by summarising both the modeling approach and the observed solver behaviour, and by outlining the main limitations and possible extensions.

7.1 Conclusions

This thesis addressed the following research question:

> *How can the Formula 1 pit stop strategy problem be effectively formulated as a QUBO model, and how do simulated annealing and quantum-inspired annealing methods perform when applied to this formulation?*

On the QUBO formulation. The pit stop strategy problem can be expressed as a QUBO across several levels of realism. The penalty-based encoding enforces the relevant constraints, and the formulation remains structurally consistent as new components are added. This makes it possible to extend the model without redefining it from scratch.
The chain-based representation of tire age in Models 5 and 6 is the main modeling contribution. It captures sequential effects within a binary framework, but leads to a variable count of $\mathcal{O}(N{\cdot}|C|{\cdot}K_{\max})$. This growth is inherent to the approach. In practice, it limits the range of instance sizes that can be handled efficiently, as already seen at $N = 20$.

On solver performance. SA performs more consistently across the tested settings. It reaches full feasibility for Models 0–4 at $N \geq 20$, scales up to $N = 50$, and produces lower race times than QIA when feasible solutions are found. For the more complex models, it remains usable but requires more runtime due to the increased number of sweeps.
QIA shows higher feasibility at small instance sizes, but this does not translate into better overall performance. Its solution quality is lower on feasible runs, and it fails completely on Models 5 and 6.

The sensitivity analysis suggests that SA's behaviour at small N is linked to the structure of the landscape rather than to insufficient penalties.

Overall answer. QUBO provides a flexible way to model the pit stop strategy problem, but the choice of solver has a strong impact on the outcome. SA is the most practical option within the tested setup. An open question is whether improved QIA implementations or hardware-based quantum annealing could combine the feasibility advantages at small N with better performance at larger scales.

7.2 Limitations

Several points limit how far these results can be generalised.
The model is fully deterministic. External factors such as weather, traffic, fuel load, and safety cars are fixed in advance. This excludes adaptive strategies that respond to changes during the race.
The comparison between SA and QIA is not entirely balanced. SA relies on a well-optimised library with multiple restarts, while QIA is implemented in Python with a single trajectory. Part of the performance gap may come from this difference.
Penalty weights were chosen manually and only partly validated. The analysis focuses on P_C at $N = 5$, while $P_B = 5000$ has a strong effect at larger sizes and was not tested systematically.
Finally, the model considers only a single car. Interactions such as undercuts, overcuts, and traffic effects are not included, which limits the realism of the resulting strategies.

7.3 Future Research

Penalty weight tuning. A more systematic approach to selecting penalty weights, for example through adaptive scaling or Lagrangian methods, could improve both feasibility and solution quality. In particular, the role of P_B deserves a more detailed analysis across different instance sizes.

Multi-car strategy. Extending the model to multiple cars would make it more realistic. Important effects include undercuts and overcuts, where strategies depend on the expected behaviour of competitors. This would increase the number of variables, but the current structure provides a starting point.

Hardware quantum annealing. The experiments are based on a classical simulation of quantum annealing. Running the model on actual hardware, such as D-Wave systems, would give more insight into the practical performance of QUBO formulations. While Models 0–4 may be feasible at moderate sizes, the more complex models would likely exceed current hardware limits.

Validation on real race data. The current evaluation uses synthetic instances. Applying the model to historical race data would allow a comparison with real strategies and help identify where the formulation can be improved, for example in the degradation model or pit-stop assumptions.

References

[AT24] Felipe Aguad and Charles Thraves. Optimizing pit stop strategies in formula 1 with dynamic programming and game theory. *European Journal of Operational Research*, 319(3):908–919, 2024.

[CHT23] Oscar F. Carrasco Heine and Charles Thraves. On the optimization of pit stop strategies via dynamic programming. *Central European Journal of Operations Research*, 31(1):239–268, 2023.

[Fé25] Fédération Internationale de l'Automobile. Formula one sporting regulations: 2026 season. Official FIA Regulations Document, December 2025. Issue 01, published 10 December 2025.

[GKD22] Fred Glover, Gary Kochenberger, and Yu Du. A tutorial on formulating and using qubo models. *Annals of Operations Research*, 314(1):141–183, 2022.

[Iov25] Gerardo Iovane. Quantum-inspired algorithms and perspectives for optimization. *Electronics*, 14(14):2839, 2025.

[JAG+11] Mark W. Johnson, Mohammad H. S. Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris, Andrew J. Berkley, Jan Johansson, Paul Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KHG+14] Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, and Yang Wang. The unconstrained binary quadratic programming problem: A survey. *Journal of Combinatorial Optimization*, 28(1):58–81, 2014.

[Luc14] Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014.

[Pap98] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1998.

[VVWV10] Nico Vandaele, Tom Van Woensel, and An Verbruggen. A queueing based traffic flow model. *Transportation Research Part D: Transport and Environment*, 15(6):356–363, 2010.

A Source Code Listings

The full source code used in this thesis is available at: https://github.com/yourusername/f1-pitstop-qubo
The listings below present the most relevant components of the implementation.

A.1 Variable Type Definitions and Index Allocation

```python
@dataclass(frozen=True)
class XVar:
    lap: int
    compound: str

@dataclass(frozen=True)
class UVar:
    compound: str

@dataclass(frozen=True)
class WVar:
    """w[l,c] = x[l,c] * x[l+1,c]  (same compound on consecutive laps)."""
    lap: int
    compound: str

@dataclass(frozen=True)
class RVar:
    """Chain run variable: R[l,c,q] = 1 iff compound c ran at least q+1
    consecutive laps ending at lap l (tire age >= q).
    R[l,c,1] is aliased to w[l-1,c] -- no extra variable needed.
    R[l,c,q] = w[l-1,c] * R[l-1,c,q-1]  for q >= 2."""
    lap: int
    compound: str
    run_length: int   # q >= 2

# Inside VariableIndex.__init__ -- index allocation for the chain model

# x[l,c]: N_laps * |compounds| variables
for l in self.laps:
    for c in self.compounds:
        var = XVar(l, c)
        self.var_to_idx[var] = idx;  idx += 1

# u[c]: one per compound
for c in self.compounds:
    var = UVar(c)
    self.var_to_idx[var] = idx;  idx += 1

if enable_chain_model:
    # w[l,c] = x[l,c] * x[l+1,c]  -- (N_laps - 1) * |compounds| variables
    for l_idx in range(len(self.laps) - 1):
        for c in self.compounds:
            var = WVar(self.laps[l_idx], c)
            self.var_to_idx[var] = idx;  idx += 1

    # R[l,c,q] for q >= 2, l >= q  (R[l,c,1] reuses the WVar above)
    for c in self.compounds:
```

```
        K_c = self.max_chain_age_per_compound[c]
        for q in range(2, K_c + 1):
            for l_idx in range(q, len(self.laps)):
                var = RVar(self.laps[l_idx], c, q)
                self.var_to_idx[var] = idx;  idx += 1
```

Listing 1: Variable type definitions (`XVar`, `UVar`, `WVar`, `RVar`) and their sequential index allocation inside `VariableIndex` for the chain encoding used in models M5 and M6 (`variable_index.py`).

A.2 Chain Constraint Builder

```
def add_chain_constraints(Q, var_index, compounds, penalty,
                          max_chain_age_per_compound):
    """
    Encodes constraint D3a  (w[l,c] = x[l,c] * x[l+1,c])  and the chain
    recursion  R[l,c,q] = w[l-1,c] * R[l-1,c,q-1]  for q >= 2,  using
    the standard product linearisation for binary variables x, y:

        penalty * (3z + xy - 2xz - 2yz)    where z = xy

    Age reset on a pit stop is automatic: w[l-1,c] = 0 forces all
    downstream R[l,c,q] = 0 by the chain recursion, so no separate
    reset constraint is required.
    """
    N_laps = var_index.N_laps

    # D3a: linearise  w[l,c] = x[l,c] * x[l+1,c]
    for lap in range(N_laps - 1):
        for c in compounds:
            w  = var_index.w(lap, c)
            x0 = var_index.x(lap,     c)
            x1 = var_index.x(lap + 1, c)
            add_qubo_coeff(Q, w,  w,   3.0 * penalty)
            add_qubo_coeff(Q, x0, x1,        penalty)
            add_qubo_coeff(Q, x0, w,  -2.0 * penalty)
            add_qubo_coeff(Q, x1, w,  -2.0 * penalty)

    # Chain: linearise  R[l,c,q] = w[l-1,c] * R[l-1,c,q-1]  for q >= 2
    for c in compounds:
        K_c = max_chain_age_per_compound[c]
        for q in range(2, K_c + 1):
            for l_idx in range(q, N_laps):
                l      = var_index.laps[l_idx]
                r_lq   = var_index.r(l,     c, q)       # RVar(l,   c, q)
                w_fac  = var_index.r(l,     c, 1)       # w[l-1, c]
                r_prev = var_index.r(l - 1, c, q - 1)   # R[l-1, c, q-1]
                add_qubo_coeff(Q, r_lq,   r_lq,   3.0 * penalty)
                add_qubo_coeff(Q, w_fac,  r_prev,       penalty)
```

```
                add_qubo_coeff(Q, w_fac,  r_lq,  -2.0 * penalty)
                add_qubo_coeff(Q, r_prev, r_lq,  -2.0 * penalty)
```

Listing 2: Chain constraint builder implementing constraint D3a and the chain recursion for tire age tracking in models M5 and M6. Each binary product is linearised using the standard three-term penalty $P(3z + xy - 2xz - 2yz)$ where $z = xy$ (`model_common.py`).

A.3 Simulated Quantum Annealing Core Loop

```
for sweep in range(num_sweeps):
    progress = sweep / max(num_sweeps - 1, 1)

    # Linear inverse-temperature schedule: hot start -> cold finish
    beta = beta_start + progress * (beta_end - beta_start)
    T    = 1.0 / beta

    # Exponential transverse-field schedule: strong tunneling -> classical
    log_gamma = (math.log(gamma_start)
                 + progress * (math.log(gamma_end) - math.log(gamma_start)))
    gamma = math.exp(log_gamma)

    # Inter-replica coupling strength (Suzuki-Trotter decomposition)
    # J_perp = -0.5 * T * ln(tanh(gamma / (P * T)))
    x = gamma / (num_replicas * T)
    J_perp = (-0.5 * T * math.log(math.tanh(x))
              if 1e-10 < x <= 20 else 0.5 * T * x)

    for k in range(num_replicas):
        for i in range(num_vars):
            current_bit = replica_system.replicas[k][i]

            # Classical QUBO energy change for flipping bit i in replica k
            delta_E_classical = delta_energy_fast(
                replica_system.replicas[k], i, adj_neighbors, adj_diag)

            # Inter-replica coupling energy change (ring topology)
            k_prev = (k - 1) % num_replicas
            k_next = (k + 1) % num_replicas
            s      = 2 * current_bit - 1                          # {0,1} -> {-1,+1}
            s_prev = 2 * replica_system.replicas[k_prev][i] - 1
            s_next = 2 * replica_system.replicas[k_next][i] - 1
            delta_E_coupling = 2.0 * J_perp * s * (s_prev + s_next)

            # Effective energy change and Metropolis acceptance
            delta_E_total = delta_E_classical / num_replicas + delta_E_coupling
            accept = (delta_E_total <= 0 or
                      rng.random() < math.exp(-beta * delta_E_total))

```

```
            if accept:
                replica_system.replicas[k][i] = 1 - current_bit
                replica_system.energies[k]   += delta_E_classical
                if replica_system.energies[k] < best_energy:
                    best_energy = replica_system.energies[k]
                    best_state  = replica_system.replicas[k].copy()
```

Listing 3: Core annealing loop of the custom Simulated Quantum Annealing (SQA) solver. Each sweep updates all bits in all P replicas. The inter-replica coupling strength $J_\perp$ is derived from the Suzuki-Trotter decomposition; acceptance follows the Metropolis criterion applied to the combined classical and coupling energy change (`quantum_inspired_annealing.py`).

A.4 QUBO Solution Decoder

```
def decode_solution(sample, var_index):
    # --- Compound per lap ---
    compounds_per_lap = {}
    for l in range(var_index.N_laps):
        chosen = [c for c in var_index.compounds
                  if int(sample.get(var_index.idx(l, c), 0)) == 1]
        compounds_per_lap[l] = (chosen[0]  if len(chosen) == 1 else
                                "NONE"     if len(chosen) == 0 else "MULTI")

    # --- Pit stop locations (compound change between consecutive laps) ---
    pit_laps = [l for l in range(var_index.N_laps - 1)
                if compounds_per_lap[l] != compounds_per_lap[l + 1]]

    # --- Stint structure ---
    stints  = []
    current = {'start_lap': 0, 'compound': compounds_per_lap.get(0, 'NONE')}
    for l in range(1, var_index.N_laps):
        if compounds_per_lap[l] != compounds_per_lap[l - 1]:
            current['end_lap'] = l - 1
            current['length']  = l - current['start_lap']
            stints.append(current)
            current = {'start_lap': l, 'compound': compounds_per_lap[l]}
    current['end_lap'] = var_index.N_laps - 1
    current['length']  = var_index.N_laps - current['start_lap']
    stints.append(current)

    # --- Tire age via chain variables (models M5/M6 only) ---
    tire_ages = None
    if var_index.enable_chain_model:
        tire_ages = {}
        for l_idx in range(var_index.N_laps):
            l   = var_index.laps[l_idx]
            age = 0
            for c in var_index.compounds:
```

```
                if l_idx >= 1:                          # q=1: R[l,c,1] = w[l-1,c]
                    age += int(sample.get(var_index.r(l, c, 1), 0))
                K_c = var_index.max_chain_age_per_compound.get(
                          c, var_index.max_tire_age)
                for q in range(2, K_c + 1):             # q>=2: RVar(l,c,q)
                    if l_idx >= q:
                        age += int(sample.get(var_index.r(l, c, q), 0))
            tire_ages[l] = age

    return {'compounds': compounds_per_lap, 'pit_laps': pit_laps,
            'num_stops': len(pit_laps), 'stints': stints,
            'tire_ages': tire_ages}
```

Listing 4: QUBO solution decoder. Translates a raw bitstring (variable index $\rightarrow \{0, 1\}$) into a human-readable race strategy: compound per lap, pit stop laps, stint boundaries, and tire age per lap for chain models (`decode.py`).