# Online Planning for F1 Race Strategy Identification

**Diego Piccinotti*, Amarildo Likmeta†*, Nicolo Brunello*, Marcello Restelli***
Politecnico di Milano*, Università di Bologna†
diego.piccinotti@mail.polimi.it, amarildo.likmeta2@unibo.it, nicolo.brunello@mail.polimi.it, marcello.restelli@polimi.it

**Abstract**

Formula 1 (F1) racing is one of the most competitive racing competitions involving high-performance single-seater racing vehicles. The result of a race is determined by vehicle and driver performance, as well as by the tire and pit-stop strategy employed in the race. In this work, we consider the problem of deciding when to pit-stop and which compound to use as a sequential decision-making problem and we investigate the application of online planning algorithms to tackle it. The availability of high-accuracy race and vehicle simulators presents a perfect opportunity to apply planning algorithms, which require a model of the environment to search for the best policies to apply. To this end, we investigate the feasibility of applying online planning to the specific problem of race-strategy identification and propose an open-loop approach that combines Monte Carlo sampling and Temporal Difference (TD) updates to identify whether to perform a pit-stop at each race lap and which tire compound to employ. Furthermore, we perform an evaluation of different planning algorithms using a simulator based on (Heilmeier et al. 2020a), which we modify to be consistent with a planning application.

## 1 Introduction

In circuit motorsport events, participating race cars have to complete a defined amount of laps around a closed circuit. Formula 1 (F1) is an international category of motorsport racing, in which open-wheel, single-seater racing cars compete. In this competition, points are currently awarded after each race to the top-ten classified drivers and their respective teams; therefore, the goal for each driver is to consistently achieve the best possible final placement. This final placement does not depend solely on their skill or the car performance, but can be significantly influenced by their *tire strategy*.

We call *tire strategy* the sequence of compounds fitted to each car, together with the number of laps used for each set of tires.

As F1 regulations often change quickly, in this work we focus on the so-called "turbo-hybrid era" (2014 - present) cycle. In this setting, the manufacturer chooses, from a hardness spectrum, the most suitable compounds for the track, mainly in relation to the asphalt's parameters (e.g., temperature and abrasion) and the subsequent expected life of the tire. Table 2 reports the relative hardness and denomination of Pirelli tire sets across the years, as described in (Heilmeier et al. 2020b).

The possibility of fitting fresh tires, also called *pit-stop*, represents one of the most strategic opportunities in Formula 1, allowing either to become faster than direct competitors or to overtake them through time gap gains instead of overtaking directly on the track. In order to devise a good tire strategy, it is essential to balance the benefit of fitting fresh tires with the cost of stopping the car to fit them. Therefore, the problem of deciding a tire strategy during the race can be seen as a sequential decision-making problem, in which the teams are tasked with deciding, at each race lap, whether to pit-stop and which tire compound to choose.

In this work, we model this problem as a Markov Decision Process (MDP) (Puterman 1994), with a continuous state-space containing information about the race position and performance of all the drivers and a discrete action-space, representing the decision to perform a pit-stop or not, and in the case a pit-stop is necessary, which of the available tire compounds to use. One of the most commonly used methods for solving this kind of sequential decision-making problem is *online planning* (Munos 2014).

In online planning, given a (possibly approximate) model of the environment, an autonomous agent is tasked with finding the best future strategy given a finite budget, usually expressed as calls to the model or time. The Monte Carlo Tree Search (MCTS) family of algorithms uses tree-search algorithms, together with Monte Carlo (MC) sampling, to iteratively build a search tree, starting from the current state of the environment, which allows the agent to estimate the value of each possible action and choose the optimal one. The Upper Confidence Tree (UCT) (Kocsis and Szepesvári 2006) is the most used algorithm of the MCTS family, because of its simplicity and ease of use. UCT builds a search-tree, iteratively and asymmetrically, focusing the tree building on the most promising regions, making it ideal for use in problems with large search spaces, such as race strategy identification, where building the full search tree is infeasible.

Race strategy identification presents the perfect scenario for the application of planning algorithms as accurate models of the environment are often available to racing teams, and the decision whether to change tires has to be made at

the end of each lap, giving the whole duration of the lap as a possible budget to perform the planning activity. Nonetheless, planning in this setting presents some challenges like large continuous state spaces and stochasticity of the environment coupled with a large planning horizon.

In this work, we investigate the application of planning algorithms to the race strategy identification problem and address some of the issues that arise when applying these algorithms to this specific decision problem. We employ an *open-loop* planning strategy coupled with Q-learning (QL) TD updates to tackle the high variance of the returns. To evaluate the algorithms, we modify the simulator described in (Heilmeier et al. 2020a) and use it as an evaluation environment as well as a forward planning model. The open-loop setting allows tackling the added complexity of stochastic transitions in a continuous state-space, without significant losses in performance while still outperforming the true strategies applied by the racing teams. The QL update strategy, taken from the Reinforcement Learning (RL) literature, allows lowering the variance of the back-up updates generated from the noisy values coming from the rollout phase of the MTCS algorithm. Finally, we compare this back-up strategy with other TD update strategies applied to UCT coming to the MCTS literature and observe an improved performance of the QL updates compared to SARSA($\lambda$) (Vodopivec, Samothrakis, and Šter 2017) and Power Mean (Dam et al. 2020) updates.

## 2 Planning in Continuous Stochastic Environments

In this section, we review the literature on planning algorithms used to address particular problems also faced in the race strategy identification problem, such as large continuous spaces and stochastic transitions. While most of the research on MCTS focuses on deterministic transitions and discrete actions, work on stochastic transitions also exists. A stochastic transition model means that the branching factor of the search tree is no longer fixed but depends on the number of possible next states in each node, which can be infinite in the case of continuous state spaces. One of the earliest MCTS algorithms tackling MDPs with stochastic transitions, providing performance guarantees in terms of the value of the recommended action, is Sparse Sampling (Kearns, Mansour, and Ng 1999). However, sparse sampling only considers the case where the number of next states is limited by $B < \infty$, which is not the case with continuous state spaces and is very computationally inefficient in practice. The setting of considering bounded next-state space cardinality is often seen in the literature. StOP (Szörényi, Kedenburg, and Munos 2014) also provides theoretical bounds on the sample complexity in this setting, but it does so by explicitly storing and searching the space of possible policies, which makes it a theoretical algorithm since it would be infeasible to apply to anything larger than small toy problems. MDP-GapE (Jonsson et al. 2020) also tackles this setting, but it does so by employing UGapE (Gabillon, Ghavamzadeh, and Lazaric 2012) a *Best Arm Identification* algorithm for the planning setting.

The main method applied to tackle continuous stochastic environments is *progressive widening*. In DPW-UCT (Couëtoux et al. 2011) the authors extend UCT to continuous stochastic environments by progressively adding new nodes to the search tree, depending on the number of visits to the node. In practice, the number of children of the node is limited by a sublinear function of the node visits. When no children can be added to the nodes, one of the current children is sampled and revisited. This also requires tuning the hyperparameters of this bounding function of the children of the nodes, which is problem-dependent. Furthermore, in practice, DPW-UCT not only requires a forward model of the environment, but also requires to be able to save the state of the simulator at each node of the tree in order to restart the simulation from that node. In some cases, such as with complex racing simulators, this greatly increases the memory requirements for the planner.

The *open-loop* setting is another way to tackle stochasticity in the environment. In this setting, the search is conducted on the set of possible sequences of actions rather than on the mappings from states to actions. Each node in the search tree represents a sequence of actions instead of a single state. This makes the search tree simpler, at the expense of lower performance. OLOP (Bubeck and Munos 2010) provides near-optimal guarantees on the expected simple regret in this setting by constructing a bandit problem on the set of possible action sequences. OLOP carefully constructs valid high-probability upper bounds on the value of each action sequence and chooses the one with the highest upper bound. Being that it has to select a complete action sequence from an exponential number of sequences at each iteration, it is impractical to apply except for small horizon problems. UCT has also been extended to open-loop settings. In (Lecarpentier et al. 2018) the authors extend UCT to open-loop planning and prove that it retains the same convergence guarantees as the original algorithm while preserving the anytime and the asymmetry properties.

## 3 Preliminaries

### Markov Decision Processes

In this work, we adopt a discrete-time Markov Decision Process (MDP) definition (Sutton and Barto 1998) given by the 6-tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathscr{P}(\mathcal{S})$[1] is the transition model, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which provides the reward signal $\mathcal{R}(s, a)$ when the agent is in state $s \in \mathcal{S}$ and performs action $a \in \mathcal{A}$, $\gamma \in [0, 1]$ is the discount factor and $\mu \in \mathscr{P}(\mathcal{S})$ is the initial state probability distribution.

An agent acting in the MDP, at each time $t = 1, \ldots, T$, observes the current state $s_t \in \mathcal{S}$, executes an action $a_t \in \mathcal{A}$ and receives from the environment a reward $r_t = \mathcal{R}(s_t, a_t)$ and next state $s_{t+1} \sim \mathcal{P}(s_t, a_t)$. The behavior of an agent is modeled by means of a policy $\pi : \mathcal{S} \to \mathscr{P}(\mathcal{A})$ which maps states to probability distributions over actions. We define the (policy-dependent) state value $V^\pi(s) =$

[1] We denote by $\mathscr{P}(\mathcal{X})$ the space of probability distributions over $\mathcal{X}$

$\mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s, a_t \sim \pi(s_t) \right]$ and state-action value $Q^\pi(s,a) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s, a_0 = a, a_t \sim \pi(s_t) \right]$, respectively. The goal of an agent is to find the policy $\pi^*(s) = \max_\pi V^\pi(s), \quad \forall s \in \mathcal{S}$.

### Monte Carlo Tree Search

In this section, we also present a brief introduction to the Monte Carlo Tree Search (MCTS) family of algorithms. In particular, we focus on the most popular algorithm in the MCTS family, the upper confidence bound for trees (UCT) (Kocsis and Szepesvári 2006).

MCTS algorithms combine tree-search algorithms with Monte Carlo sampling to iteratively build a search-tree. They can be summarized by four general phases (Browne et al. 2012):

1. **Selection**: from the root of the search tree, a selection policy (also called tree policy) is recursively applied up to reaching an unexpanded leaf node. A node is expandable if it represents a nonterminal state and has unvisited children.
2. **Expansion**: one or more successors to the previously found unexpanded node are generated according to the actions available in the node.
3. **Simulation**: from the newly generated node(s), a simulation (also called *rollout*) is run following a default policy up to a terminal state.
4. **Backpropagation**: a "back-up" of the rewards collected during the simulation is performed at the nodes which have been visited in the trajectory from the root, updating their statistics.

The UCT algorithm uses as selection policy the bandit algorithm Upper Confidence Bound (UCB1) (Auer, Cesa-Bianchi, and Fischer 2002). UCB1 chooses the next action to perform, $a_n$, according to

$$a_n = \arg\max_{i=1..K} B(a_i) = \overline{R}_{i,T_i(n-1)} + C\sqrt{\frac{2\log n}{T_i(n-1)}},$$

where $K$ is the number of actions, $C$ is a constant which regulates the exploration-exploitation tradeoff, $T_i(n-1)$ is the number of times action $i$ has been played up to time $n-1$ and $\overline{R}_{i,T_i(n-1)}$ is the average payoff observed from arm $i$. UCT considers the selection process at each decision node as a separate bandit problem, where the payoff is the return of the trajectories starting from that node. In addition, during the back-up phase, UCT recursively updates the values of the nodes from the leaf to the root of the tree. The algorithm is proved to be *consistent*, i.e., it converges to the optimal policy in the limit. In addition to the convergence guarantees of UCT, the algorithm also enjoy some other properties that have made it highly popular in the literature. First, the algorithm is *anytime*, which means it does not need to know the planning budget in advance and can return its best guess at any moment. This makes it particularly attractive for real-time scenarios like race-strategy identification. The algorithm is also *asymmetric*, which means that it iteratively builds the search tree by favoring the most promising regions of the tree while still giving a probability of selection to all branches. This is extremely important in some applications where the search tree is extremely large. This, in fact, is also one of the reasons why UCT has no theoretical guarantees on its *sample complexity*. This asymmetric tree building is based on the confidence bound kept from the UCB algorithm employed in each node, which, in turn, are not valid in high probability and might delay the discovery of the optimal paths if the reward function is misleading in the top levels of the tree.

### Open Loop Planning

In the open-loop setting, the problem is shifted from finding the optimal policy (mapping from states to actions) to finding the optimal sequence of actions to perform starting from the current state, regardless of the intermediate states visited, and instead averaging between them. Clearly, when the MDP transitions are deterministic, the open-loop and closed-loop settings are equivalent.

More specifically, given a starting state $s \in \mathcal{S}$ and a sequence of actions $\tau = (a_0, a_1, \ldots, a_m), a_i \in \mathcal{A}$, the value of the sequence $\tau$ starting from the state $s$ is defined as:

$$V_{OL}(s,\tau) = \mathbb{E}\left[ \sum_{t=0}^{m} \gamma^t r_t \,\middle|\, s_0 = s, a_t \in \tau \right]. \qquad (1)$$

We note that, $\tau$ can be an infinite sequence if $\gamma < 1$. Accordingly, the open-loop optimal value is given by maximizing over the sequences of actions:

$$V^*_{OL}(s) = \max_\tau V_{OL}(s,\tau). \qquad (2)$$

We also define, the open-loop action-value of a state-action pair, $(s,a)$, as the maximizer over the possible action sequences $\tau$ that start with $a$, denoted with $\tau_a$:

$$Q^*_{OL}(s,a) = \max_{\tau_a} V_{OL}(s,\tau_a). \qquad (3)$$

Planning in an open-loop setting leads to a loss of performance, since $Q^*_{OL}(s,a) < Q^*(s,a)$, but it simplifies the planning problem by limiting the size of the search-tree and may be beneficial in cases with small search budgets.

## 4 Simulation Environment

To evaluate our planners, we build on the race simulator described in (Heilmeier et al. 2020a). This simulator provides a parameterized and probabilistic description of the race allowing control over probabilistic race events such as accidents, Safety Cars (SC), or Virtual Safety Cars (VSC). Each of the simulator's internal events relies on probabilistic models, whose parameters have been optimized to provide a plausible replica of past races, using publicly available data (Heilmeier et al. 2020a) on tire strategy and events.

The race is modeled using a lap-by-lap approach, in which the race time of each driver in the simulation is computed at each lap. Since the decision space is whether to pit stop on each lap, a lap-by-lap simulator allows us to model temporal transitions between states in an MDP setting. Lap times are

computed based on the sum of multiple contributions such as, for instance, tire wear, fuel consumption, vehicle performance, driver performance, race events, and others. Each of these contributions is stochastic and is sampled from distributions fitted on the dataset of previous races. For further details on the various contributions and the dataset used to fit the probabilistic models, see (Heilmeier et al. 2020a).

The simulator is also able to consider driver retirements and Full Course Yellow (FCY) flag events. In Formula 1, when exposed by the marshals, a simple yellow flag forbids overtaking and requires drivers to lift the throttle pedal only in the affected track sector, whereas FCYs affect the whole track. An FCY is usually triggered by accidents on the track that force the race marshals to order the drivers to reduce their speed to maintain safety on the racecourse. There are two types of FCY, Safety Car (SC) and Virtual Safety Car (VSC), which both correspond to a delta with respect to a target lap, set by the marshals before each race, that drivers must respect. VSC is a digital safety measure, which requires drivers to autonomously increase their lap times to 140% of a reference lap. Instead, SC is a physical car that is sent on track to be followed by the drivers. Usually, the lap times of drivers behind the Safety Car increase to 160% of the reference lap time. Because all drivers are slowed down, a pit-stop may become particularly appealing as the time lost to change tires reduces during an FCY. This happens because the FCY deployment lowers the speed difference between the main straight and the pit-lane, thus reducing the amount of time lost with respect to competitors when performing the pit-stop.

Slight changes have been applied to the simulator to allow a high-level control of the drivers' strategy. First, we modified the way the Full Course Yellow (FCY) events are generated and handled. It is clear that the simulation of FCY events is crucial to obtain useful recommendations from the agent within this context. The simulator in (Heilmeier et al. 2020a) handles the generation of random events according to real race data, stored inside the simulator itself and loaded at the beginning of each simulation. From the planning agent's point of view, even though this information is not included in the state, in different simulations of the same race performed during planning, the events would always occur at the same time, consistently. This would mean that the planner would be able to "predict the future" and prepare for the events beforehand. Therefore, we modify this behavior by generating events during the race either with a stochastic model or adding the real race's events only at runtime. In particular, we allowed the simulator to accept new FCY events to be added at runtime one lap after they have been requested. This way, we ensure that FCYs in Monte-Carlo simulations appear only when they would be "visible" in the race. The duration of the FCY event can either be pre-specified, assuming a human operator evaluates the race situation and enters the expected duration into the system, or it can be generated at random, to simulate multiple scenarios during the search.

Secondly, the race simulator would load strategies for all drivers from a configuration file and then run the full event simulation with these fixed strategies. In order to comply with our planning simulation requirements, we extend the simulator to support also lap-by-lap manual control of the drivers. We consider a discrete action space where actions represent either performing a pit stop to fit one of the available compounds (one action for each compound) or a "stay-on-track action". At the beginning of each race, one or more drivers can be specified as controlled drivers, whose actions can be specified at each lap. The other drivers are considered as part of the environment and are controlled by default policies. These default policies can be customized for each driver, e.g., the lap strategy of the actual race, or they can be specified at the beginning of the race. In the experiments we performed, each non-controlled driver employed the real-race tire strategy.

On top of this simple framework, the environment ensures, mostly by modifying actions available at each timestep, that the constraints related to the F1 race strategy problem are satisfied. Firstly, we start the race episodes at a specified race lap, whereas before that lap, the agent-controlled drivers take actions according to the strategy used in the real race. This behavior models the fact that, barring accidents or sudden weather changes, a pit-stop is not required in the first laps of a race. Secondly, to simplify the decision space, drivers controlled by an agent cannot perform more than two pit stops in a single race, with the exception of one extra pit stop allowed each time an FCY is deployed, but only while the FCY stays active on the track. This constraint is reasonable, as most of the races in the period under consideration could have been completed with at most two pit stops unless accidents or weather changes occurred. Furthermore, after a driver has made a pit-stop, we remove the pit-stop possibility for the next five laps. For instance, if a driver pits at lap 18, he will be unable to pit again until lap 24. Since this is a reasonable behavior even in real races (all compounds typically last more than 10 laps), it is enforced in our environment to reduce the planning complexity.

Each driver can choose from a finite number of tire sets for each compound. In real races, each team reserves a specific number of tire-sets for each compound to the race. As such data is not easily obtainable for historic races, we considered two possible scenarios. For races in which only two compounds were available or had been used in the race, we assumed three sets of tires available for the softer compound, whereas the harder compound had two. For races where more than two dry compounds were used, we assumed the availability as follows: two sets of tires for the softer compound, two for the medium, and one for the harder. We made these default tire availability assumptions based on the empirical observation that softer compounds are generally preferred unless extreme track temperature conditions are present. As per F1 regulations, drivers must change at least one compound type during the race. If this does not happen, the environment forces the offending driver(s) to make a pit stop at the penultimate lap of the race to satisfy such rule and avoid assigning penalties.

After the actions for all controlled drivers have been received, the computation of the lap times is tasked to the underlying simulator (Heilmeier et al. 2020a).

### MDP Modelling

We now present details on the state and reward representation employed in this environment. The state-space is fairly hard to model for this problem, as many factors are needed to be taken into account to fully identify the racing situations. First, to represent the state of the race, we keep track of the remaining laps until the end of the race. We also consider two flags, which represent whether a Safety Car (SC) or Virtual Safety Car (VSC) (types of FCY) event exists in the current lap. Then, for each driver in the race, we track features related to his performance, including their previous lap time, cumulative race time, tire compound currently employed, current tire age (as a measure of tire degradation). For each driver, we also track features related to the employed tire compounds and the remaining tire compounds. This is important for strategic reasoning as well as for fulfilling the condition of changing the compound at least once during the race. To this end, for each driver, we keep a flag variable, which represents whether the agent has satisfied the condition on changing its compound type, as well as information on how many tire sets for each remaining compound type the driver has left. For instance, if we have three compound types, we include three variables for each driver, each of which tracks the number of tire sets of each compound the driver has left. By combining the latter two, we do not need to store information about previous tire changes, as this gives us enough information on which tire compound to apply next in case we need to make a pit stop.

To summarize, the state is composed of 3 variables which represent the state of the race together with $K * (R + D)$ driver-related variables, where $K$ is the number of drivers in the race and $D$ is the number of tire compounds, and $R$ is the number of variables tracking driver performance and compound change, which in our setting equals to 5.

Finally, we discuss the reward function employed. Every driver's goal in F1 is to maximize his final position to accumulate as many points as possible. Nonetheless, having a reward function represented by the number of positions earned at each lap would be too sparse a reward function, which is harder to optimize. In this work, we employ a reward function represented by the negative lap time. In this way, each driver aims to minimize the cumulative race time. In certain situations, a driver may choose to sacrifice lap time in order to keep his position or gain positions, e.g., when preventing a competitor from overtaking or delaying a pit stop strategically. A more principled approach would be to employ a reward function that weighs two different objectives, minimizing time and gaining positions, but accurately weighing these two contributions would not be straightforward and is beyond the scope of this work.

## 5 Open-Loop Planning for Race Strategy

In this work, we employ an open-loop approach to tackle the race strategy problem, searching at every lap of the race for the best race strategy to use for the rest of the race. We choose an open-loop strategy instead of the alternative progressive-widening (PW) (Couëtoux et al. 2011) because of the additional memory constraints required to employ PW by copying the internal state of the simulator to each node of the search tree. In fact, for the evaluation campaign presented in the next section, we considered including an evaluation of a PW-UCT algorithm but quickly ran out of memory during the tree search, even for small search budgets.

We start from an open-loop setting of UCT. We denote by $\mathcal{T}$ a planning tree and by $\mathcal{N}_{d,i}$ the $i$-th node at depth $d \geq 0$ for $i \in \mathbb{N}$. $\mathcal{N}_{0,0}$ contains a single state, $s_0 \in \mathcal{S}$, from which we want to perform planning. Nodes $\mathcal{N}_{d,i}$, with $d > 0$, at deeper levels of the tree, represent the distribution of states given the sequence of actions from the root of the tree to $\mathcal{N}_{d,i}$. More specifically, given a sequence of $d$ actions, $\tau_{d,i} = (a_1, a_2, \ldots, a_d)$, this sequence identifies exactly the node $\mathcal{N}_{d,i}$ in our tree $\mathcal{T}$, representing the distribution of states $s \in \mathcal{S}$ reachable by executing the sequence of actions $\tau_{d,i}$ starting from the root state $s_0$. We define the value of a node $\mathcal{N}_{d,i}$ as

$$\mathcal{V}\left(\mathcal{N}_{d,i}\right) = \mathop{\mathbb{E}}_{s\sim\mathcal{P}(\cdot|s_0,\tau_{d,i})}\left[V^*_{OL}(s)\right], \tag{4}$$

and the value of an action $a \in \mathcal{A}$ in a node as:

$$\mathcal{Q}\left(\mathcal{N}_{d,i}, a\right) = \mathop{\mathbb{E}}_{s\sim\mathcal{P}(\cdot|s_0,\tau_{d,i})}\left[Q^*_{OL}(s,a)\right] \tag{5}$$

$$= \mathop{\mathbb{E}}_{s\sim\mathcal{P}(\cdot|s_0,\tau_{d,i})}\left[r(s,a)\right] + \gamma\mathcal{V}\left(\mathcal{N}_{d+1,j}\right), \tag{6}$$

where $\tau_{d+1,j} = (\tau_{d,i}|a)$ is the sequence of action derived from concatenating $\tau_{d,i}$ with $a$ and $\mathcal{N}_{d+1,j}$ is the corresponding node in the tree.

The goal of our planner is to estimate the open-loop values of the actions in the root of the tree by employing a UCT-like tree policy, which considers action selection at each node as a separate bandit problem and selects the action that optimizes an upper bound for $\mathcal{Q}$ values of the actions in the node.

The second modification we add to the standard UCT scheme is the tree back-up strategy. UCT employs MC updates recursively up the tree after receiving the leaf-value estimates from the rollout policy. Since the rollout policy is suboptimal by definition (if we had an optimal policy for rollout, we would not need planning) and the selection policy in the tree includes the exploration and exploitation of intermediate value estimates, the back-up values include the evaluation of suboptimal policies, which change with each search iteration. This usually makes the value estimates in the tree very noisy, which is a problem in our race strategy problem as the margins for selecting a good pit-stop strategy are small compared to the race duration. For this reason, we employ a TD operator, namely the Q-learning operator (Watkins 1989) as follows:

$$\mathcal{Q}_t\left(\mathcal{N}_{d,i}, a\right) = (1-\alpha_t)\mathcal{Q}_t\left(\mathcal{N}_{d,i}, a\right) \tag{7}$$

$$+ \alpha_t\left(r_t + \gamma\max_{a'}\mathcal{Q}_t\left(\mathcal{N}_{d+1,j}, a'\right)\right), \tag{8}$$

where $r_t$ is the reward observed in the current search pass at node $\mathcal{N}_{d,i}$ and $\alpha_t$ is the learning rate. We include a comparison with other TD update strategies (Vodopivec, Samothrakis, and Šter 2017; Dam et al. 2020) in the experimental setting.

**Algorithm 1** Q-Learning Open Loop Planning

**procedure** OLSEARCH($s_0$)
  Create root node $\mathcal{N}_{0,0}$ from state $s_0$
  **while** within computational budget **do**
    $\mathcal{N}_{d,i}, s \leftarrow$ TREEPOLICY($\mathcal{N}_{0,0}$)
    $\mathcal{V}(\mathcal{N}_{d,i}) \leftarrow$ ROLLOUT($\mathcal{N}_{d,i}, s$)
    BACKUP($\mathcal{N}_{d,i}$)
  **end while**
  **return** BESTCHILD($\mathcal{N}_{0,0}$)
**end procedure**
**procedure** TREEPOLICY($\mathcal{N}$)
  **while** $\mathcal{N}$ not terminal **do**
    **if** $\mathcal{N}$ not fully expanded **then**
      **return** EXPAND($\mathcal{N}$)
    **else**
      $\mathcal{N} \leftarrow$ BESTCHILD($\mathcal{N}, C_p$)
    **end if**
  **end while**
  **return** $\mathcal{N}$
**end procedure**
**procedure** ROLLOUT($\mathcal{N}$, s)
  $\Delta \leftarrow 0$
  **while** s is non-terminal **do**
    Choose $a \in A(s)$ according to rollout strategy
    Generate next state $s'$ and reward $r$
    $\Delta \leftarrow \gamma\Delta + r$
    $s \leftarrow s'$
  **end while**
  **return** $\Delta$
**end procedure**
**procedure** EXPAND($\mathcal{N}$)
  Choose $a \in$ untried actions from $\mathcal{N}$
  $s \leftarrow$ SIMULATEUNTIL($\mathcal{N}$)
  Execute $a$ in $s$ generating $s'$ and $r$
  Add a new child $\mathcal{N}'$ to $\mathcal{N}$
  $\mathcal{N}'.n \leftarrow 0$
  $\mathcal{N}'.r = r$ ▷ Store the reward obtained during first visit
  **return** $\mathcal{N}', s'$
**end procedure**
**procedure** BESTCHILD($\mathcal{N}, c$)
  $C(\mathcal{N})$ denotes children nodes of $\mathcal{N}$
  $C(\mathcal{N}, a)$ denotes the child of $\mathcal{N}$ corresponding to action $a$
  **return** $\arg\max_a \mathcal{Q}(\mathcal{N}, a) + c\sqrt{\frac{2\ln \mathcal{N}.n}{C(\mathcal{N},a).n}}$
**end procedure**

**procedure** BACKUP($\mathcal{N}, V$)
  $C'(\mathcal{N})$ denotes explored children nodes of $\mathcal{N}$
  $\mathcal{N}' \leftarrow$ parent of $\mathcal{N}$
  $\mathcal{N}.n \leftarrow \mathcal{N}.n + 1$
  **while** $\mathcal{N}'$ is not null **do**
    **if** $\mathcal{N}$ is leaf **then**
      $\Delta \leftarrow V$
    **else**
      $\Delta \leftarrow \max_{a' \in C'(\mathcal{N})} Q(\mathcal{N}, a')$
    **end if**
    $Q(\mathcal{N}', a) \leftarrow Q(\mathcal{N}', a) \quad +$
      $\alpha(\mathcal{N}'.r + \gamma\Delta - Q(\mathcal{N}', a))$
    $\mathcal{N}'.n \leftarrow \mathcal{N}'.n + 1$
    $\mathcal{N} \leftarrow \mathcal{N}'$
    $\mathcal{N}' \leftarrow$ parent of $\mathcal{N}$
  **end while**
**end procedure**

Algorithm 1 shows the pseudocode of the planner employed. At each lap in the race, we perform multiple iterations of planning until we reach the planning budget. At each search iteration, we perform the UCB selection at each node of the tree until we reach a leaf-node, from which we perform a rollout, using one of the rollout policies discussed below. The rollout gives us an initial (noisy) estimation of the node value. Next, we recursively employ QL updates up the tree, updating the node and action values and counts. This means that the initial noisy back-up value given by the rollout, even though it is stored in the leaf node, might not make its way up to the root, since at each node we employ the max operator to define the target value, as shown in the BACKUP procedure. When all the children of a node have not been explored yet, for the Q-learning update of Equation 7, we apply the max operator only on the visited nodes, disregarding the unexplored actions.

For space considerations, we show the implementation of all procedures, except for SimulateUntil. This procedure takes as input a node of the search tree and simulates all the transitions from the root state to the input node by executing the sequence of actions that identify that node. In the end, it returns the state of the environment resulting from the sequence execution, which is used later as a starting point for the rollout.

## 6 Rollout and Opponent Policies

Since, during the planning phase, we do not desire an agent to be "clairvoyant" over the opponents' strategies, we need to specify a planning configuration for the simulator to use when performing the tree search. In our experimental setting, we implemented this goal by considering all drivers as controlled by the planner and applying a rollout policy for the opposing drivers. The rollout policy is a crucial component of MCTS-like algorithms, as it provides an initial estimate for the value of leaf nodes, which is then used during the back-up phase of such algorithms. Using a random policy would generate extremely noisy value estimates in the tree nodes; therefore, careful consideration of the rollout is needed.

During our experiments, we modeled two types of default strategies. The first approach we considered was a simple stochastic strategy: the considered driver, at each lap, has a 0.9 probability of staying on the track and a 0.1 probability of making a pit stop by randomly choosing one of the compounds available. As the strategies provided by this simple baseline were, for the most part, unreasonable for both the track situation and the tire degradation status in the simulator, we moved to a more realistic strategy definition.

We then focused on the predicted race strategies publicly available in sports articles (see Table 3) taken from the motorsport opinion website ESPN.[2] Since the Monte-Carlo agents explore different and sometimes unreasonable strategic options, it would have been hard to automatically identify and re-map the agent's strategy to one of the predicted ones. To be independent of such mapping, we computed a plausible amount of laps that each compound would

[2]https://www.espn.com/f1

| Season | Track | ESPN | True | VSE | Sarsa UCT | Power UCT | OL UCT | QL-OL UCT | Ranking Gain |
|---|---|---|---|---|---|---|---|---|---|
| 2015 | Japan | 4576.01±1.0 | 4577.52±1.0 | 4575.34±1.3 | 4575.36±1.2 | 4583.25±2.4 | 4577.99±1.1 | **4570.35±1.0*** | 0.4 |
| 2016 | Japan | 4507.85±1.0 | 4507.54±0.7 | 4549.01±1.3 | 4508.90±0.8 | 4524.45±1.2 | 4519.03±1.3 | **4505.35±0.9*** | 0.1 |
| 2017 | Australia | 4470.39±1.7 | 4466.22±1.9 | 4477.29±1.3 | 4466.56±2.9 | 4474.12±2.2 | 4479.90±2.2 | **4459.71±2.4*** | -1.3 |
| 2017 | Spain | 5202.42±1.4 | 5209.89±1.3 | 5207.94±1.1 | 5196.38±2.1 | 5200.83±2.0 | 5211.05±1.1 | **5188.05±1.3*** | 0.1 |
| 2017 | Austria | 4525.88±1.2 | 4430.84±1.7* | 4491.66±1.9 | 4476.43±2.8 | **4444.38±2.4** | 4484.14±1.8 | 4465.85±2.9 | -2.4 |
| 2017 | Belgium | 4265.4±0.7 | 4256.44±1.0 | 4236.0±0.6* | 4255.98±0.7 | 4259.52±0.7 | 4260.24±1.0 | **4246.09±0.7** | 2.9 |
| 2017 | Russia | 4419.98±1.3 | 4412.87±1.2* | 4428.62±2.4 | 4425.10±2.1 | 4437.00±1.3 | 4430.93±1.7 | **4421.54±1.3** | 0.0 |
| 2018 | China | 5140.7±0.9 | 5134.01±1.0 | 5095.34±2.0* | 5099.33±1.5 | 5128.63±0.9 | 5113.31±2.6 | 5098.93±1.5 | 4.2 |
| 2018 | Italy | 3909.37±1.9 | 3898.38±1.9* | 3943.95±1.9 | 3907.22±1.4 | 3918.42±1.3 | 3911.24±1.3 | **3903.67±1.5** | -0.3 |
| 2018 | Brazil | 4678.24±2.1* | 4700.36±2.1 | 4711.61±1.7 | 4692.7±3.1 | 4699.25±1.7 | 4706.94±1.5 | **4686.32±2.9** | 2.0 |

Table 1: Cumulative return comparison for the experimental setting, lower is better. Bold stands for best performance among planners, star stands for best overall race time.

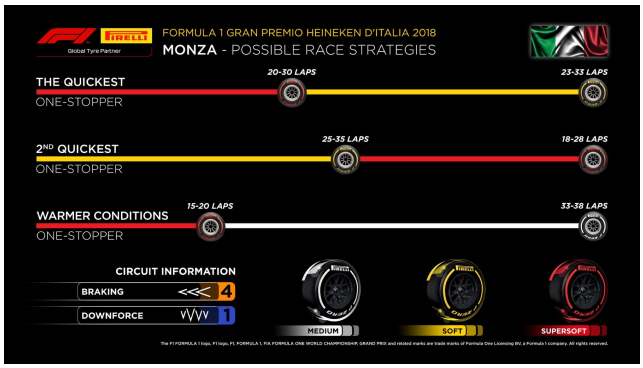


Figure 1: A sample of strategies for the 2018 Italian GP predicted by Pirelli, taken from https://twitter.com/pirellisport/status/1036169705634054144

last in a specific race, which we call compound durability, which would still yield some of ESPN's predicted strategies. Figure 1 shows an example of the manufacturer's predicted strategies, proposing different options to cover the race distance. Note that the same compound can be used in stints of different lengths, depending on the sequence of tires included in the strategy and the position of the stint in the sequence. This may be due to the fact that, in the real world, tire degradation is higher in the early stages of the race since less rubber has settled on the asphalt, and the cars have more weight due to the almost entirely unburned fuel mass. In practice, this means that the same compound will last longer if fitted towards the end of the race, being less stressed during its working period. Furthermore, a compound usually wears out faster if the driver pushes for faster lap times, leading to a shorter stint duration.

To compute tire durability, we first took the average duration for stints with uncertain duration, and then we averaged between stints with the same compound and different strategies that had different durations. After applying the second averaging, however, some of the original strategies were no longer obtainable: one or more compounds would have reduced their maximum durability and therefore left a part of the race uncovered, requiring a further pit stop. To address this problem, we extended the durability of the hardest compounds to cover the missing laps, relying on the empirical observation that they present a slower lap-time degradation than the softer ones and, therefore, extending a stint for a few laps would have less impact than using a softer compound.

The rollout policy that takes advantage of these durabilities works as follows. When the current tire set has reached its expected duration, the pit stop is performed with a probability of 0.9, or the decision is deferred by one lap with a probability of 0.1. The choice of the next compound to use is deterministic: if there are any compounds whose durability would cover the remainder of the race, the policy suggests fitting the softest one among them. If there is no such compound, the softest compound available is fitted instead. Finally, to meet F1 regulations, the policy ensures that each driver switches to a different compound at their first pit stop.

## 7 Experiments

Our experiments focused on a specific driver, Sebastian Vettel, and on a specific time frame, from 2015 to 2018. This decision was backed up by the fact that Vettel, who was driving for Scuderia Ferrari at the time, was a close contender for the title in 2017 and 2018 and had a good performance in the remaining years. Furthermore, we could recall that Scuderia Ferrari made some strategic mistakes in some races during this period, so the goal of our experiments for these races was to check if an online planning agent would have avoided them. As the race simulator presents some deviations from the historical data, we simulated each race 1000 times and averaged each driver's final race time to generate the baseline performance of the true strategies employed by the drivers.

For our experiments, we considered those races where the average gap between Vettel and the driver in front was less than 10 seconds and we selected a sample of 9 races (see Table 4). This selection criterion was aimed at finding races in which, with better strategic decisions, it would have been possible to bring the driver to the front. As the authors are Ferrari fans, the 2017 Spanish GP race was added to the race list, as it hosted a spectacular battle between Hamilton and Vettel, and we feel that the Scuderia's strategy could have been stronger on that event, bringing the number of competitions considered to 10.

For each race, we performed 100 experiments with each of the following planners: Sarsa UCT (Vodopivec, Samothrakis, and Šter 2017), Open-Loop UCT (Lecarpentier et al.

2018), PowerUCT (Dam et al. 2020), and our agent Q-learning OL UCT. Furthermore, we performed the same amount of experiments using VSE (Heilmeier et al. 2020b), a neural-network-based agent designed specifically for automated F1 pit-stop decision. For our QL OL-UCT we employed an exponentially decaying learning rate. We started the strategy planning from lap 8 and set the discount factor $\gamma$ to 1 to make the agent focus on the cumulative reward, as, in F1 racing, points are awarded based on the final standing. The computational budget of our experimental setting represents the maximum number of samples that the agent can take from the environment. We set the computational budget to 10,000 samples. The time horizon for the problem is variable between each race, as it corresponds to the total number of laps prescribed for each racing event minus the starting lap. Table 4 reports the selected races and the respective number of prescribed laps.

Each planner's hyperparameters were tuned on the 2017 Australian GP race using the Bayesian optimization framework Mango (Sandha et al. 2020) and employed the same hyper-parameters in each race. We consider the 2017 Australian GP a suitable choice for hyperparameter tuning because of two main factors. First, the real race was a close fight between Vettel and Hamilton, which ultimately Vettel won. Secondly, we selected this race because it had no accidents and no subsequent FCYs so that the optimizer would not look for parameters correlated to specific racing situations.

Table 1 reports the results for our experiments, comparing the undiscounted cumulative return for both the planners mentioned at the beginning of this section and two baselines. The ESPN baseline value was obtained by applying the *default* strategy prescribed by the environment, whereas the *true* baseline corresponds to the result obtained by applying the strategy used by the driver in the real race. We show in bold, the best planner (between the UCT variants) in each race. We use bold, only when the difference between the best and second best satisfies a statistical significance test. Only in China 2018 we do not have a best planner since the performance of QL-OL UCT and SARSA UCT are close compared to their confidence interval. Moreover, with the symbol $*$, we denote the best strategy between planners and baselines (with statistical significance).

The results we obtained show that in most races, our proposed planning algorithm performs better than other planners. Furthermore, the planner is able to improve the average race times on most occasions with respect to the real strategy performed by the drivers. All planners fail to do so in three races: Italy 2018, Russia 2017, and Austria 2017. In particular, the Austria 2017 race shows the larger gap between real strategy and planner performance. Our a-posteriori analysis found that the predicted ESPN strategy was highly inconsistent with the real strategies applied during the race: the article considered more tire wear than in the race, almost halving some of the durability of the compounds. We attribute this performance loss to the suboptimality of the rollout policy, which can be addressed by increasing the computational budget: in this specific race, the performance of our proposed algorithm improved by 15 seconds by setting the computational budget to 100,000, whereas a parameter tuning for the race did not provide significant performance gains.

A final remark is that, in most of the considered races, autonomous agents are able to improve or maintain the final position of the real driver. The last column, labeled *Ranking Gain* shows the average ranking improvement achieved by our planner compared to the position achieved by the true race strategies of the race. An interesting case where this does not happen is represented by the 2017 Australian GP, in which our proposed algorithm is able to improve the cumulative race time but loses positions in the final placement. This is most likely an artifact generated by the reward structure: since the reward promotes behaviors that minimize the cumulative race time, the agent does not consider it penalizing to take actions that allow its competitors to overtake it, as long as this allows to maximize the reward, suggesting the need for a carefully thought-out reward function that incentivizes fast laps and position gains, while still providing a dense reward signal to the agent.

## 8 Conclusions and Future Works

In this work, we investigated how MCTS algorithms can be used to design an automatic race strategy identification system. We employed an open-loop search strategy to tackle the large, continuous, stochastic state transition model and employed TD updates to address the high variance of the returns observed in the search tree. We empirically demonstrated, using a racing simulator, that open-loop planning can be used to improve the performance of hand-crafted race strategies, represented by the ESPN rollout policies employed during the search. We believe that online-planning algorithms can be a resourceful tool, able to provide race strategy recommendations to the strategists of Formula 1 teams during the race, especially when race situations differ from the predictions made before the race.

Nevertheless, we observe that the performance of planners strongly depends on the rollout policies employed, as they are used for initial evaluations of the tree nodes, and therefore affect the regions of the tree explored during the search. This can, in turn, be tackled with higher search budgets.

Several future extensions of this work are possible. First, to decrease search times and to allow generalization across races, we can employ an AlphaZero (Silver et al. 2017) planning strategy, where function approximators are used to give an initial bias to the actions to explore in the tree and to evaluate the leaf nodes. The extension of this algorithm to an open-loop setting is not straightforward and can present an interesting research problem. Second, a multi-agent framework can be employed to model situations where the controlled driver is "dueling" with other drivers. In this case, the dueling driver would not be considered as part of the environment, but a multi-player (double in the simple case of one rival) MCTS could be employed, which allows the agent to consider the adversarial behavior of the opponent.

## References


Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-Time Analysis of the Multiarmed Bandit Problem. *Mach. Learn.* 47(2–3): 235–256. ISSN 0885-6125. doi:10.1023/A:1013689704352. URL https://doi.org/10.1023/A:1013689704352.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1): 1–43. doi:10.1109/TCIAIG.2012.2186810.

Bubeck, S.; and Munos, R. 2010. Open Loop Optimistic Planning. In *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel*. URL https://www.microsoft.com/en-us/research/publication/open-loop-optimistic-planning/.

Couëtoux, A.; Hoock, J.-B.; Sokolovska, N.; Teytaud, O.; and Bonnard, N. 2011. Continuous Upper Confidence Trees. In Coello, C. A. C., ed., *Learning and Intelligent Optimization*, 433–445. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-25566-3.

Dam, T.; Klink, P.; D'Eramo, C.; Peters, J.; and Pajarinen, J. 2020. Generalized Mean Estimation in Monte-Carlo Tree Search. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 2397–2404. International Joint Conferences on Artificial Intelligence Organization. doi:10.24963/ijcai.2020/332. URL https://doi.org/10.24963/ijcai.2020/332. Main track.

Gabillon, V.; Ghavamzadeh, M.; and Lazaric, A. 2012. Best Arm Identification: A Unified Approach to Fixed Budget and Fixed Confidence. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems*, volume 25, 3212–3220. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper/2012/file/8b0d268963dd0cfb808aac48a549829f-Paper.pdf.

Heilmeier, A.; Graf, M.; Betz, J.; and Lienkamp, M. 2020a. Application of Monte Carlo Methods to Consider Probabilistic Effects in a Race Simulation for Circuit Motorsport. *Applied Sciences* 10(12). ISSN 2076-3417. doi:10.3390/app10124229. URL https://www.mdpi.com/2076-3417/10/12/4229.

Heilmeier, A.; Thomaser, A.; Graf, M.; and Betz, J. 2020b. Virtual Strategy Engineer: Using Artificial Neural Networks for Making Race Strategy Decisions in Circuit Motorsport. *Applied Sciences* 10(21). ISSN 2076-3417. doi:10.3390/app10217805. URL https://www.mdpi.com/2076-3417/10/21/7805.

Jonsson, A.; Kaufmann, E.; Ménard, P.; Domingues, O. D.; Leurent, E.; and Valko, M. 2020. Planning in Markov Decision Processes with Gap-Dependent Sample Complexity.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 1999. A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, 1324–1331. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based Monte-Carlo Planning. In *In: ECML-06. Number 4212 in LNCS*, 282–293. Springer.

Lecarpentier, E.; Infantes, G.; Lesire, C.; and Rachelson, E. 2018. Open Loop Execution of Tree-Search Algorithms. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, 2362–2368. International Joint Conferences on Artificial Intelligence Organization. doi:10.24963/ijcai.2018/327. URL https://doi.org/10.24963/ijcai.2018/327.

Munos, R. 2014. From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning. *Foundations and Trends® in Machine Learning* 7(1): 1–129. doi:10.1561/2200000038. URL https://doi.org/10.1561%2F2200000038.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc. ISBN 0471619779.

Sandha, S. S.; Aggarwal, M.; Fedorov, I.; and Srivastava, M. 2020. Mango: A Python Library for Parallel Hyperparameter Tuning. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3987–3991. IEEE.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T. P.; Simonyan, K.; and Hassabis, D. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR* abs/1712.01815. URL http://arxiv.org/abs/1712.01815.

Sutton, R. S.; and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. A Bradford book. Bradford Book. ISBN 9780262193986.

Szörényi, B.; Kedenburg, G.; and Munos, R. 2014. Optimistic Planning in Markov Decision Processes Using a Generative Model. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, 1035–1043. Cambridge, MA, USA: MIT Press.

Vodopivec, T.; Samothrakis, S.; and Šter, B. 2017. On Monte Carlo Tree Search and Reinforcement Learning. *J. Artif. Int. Res.* 60(1): 881–936. ISSN 1076-9757.

Watkins, C. J. C. H. 1989. *Learning from delayed rewards*. Ph.D. thesis, King's College, Cambridge.

| Season | A1 | A2 | A3 | A4 | A5 | A6 | A7 |
|---|---|---|---|---|---|---|---|
| 2014 | Hard | Medium | Soft | Supersoft | - | - | - |
| 2015 | Hard | Medium | Soft | Supersoft | - | - | - |
| 2016 | Hard | Medium | Soft | Supersoft | Ultrasoft | - | - |
| 2017 | Hard | Medium | Soft | Supersoft | Ultrasoft | - | - |
| 2018 | Superhard | Hard | Medium | Soft | Supersoft | Ultrasoft | Hypersoft |
| 2019 | - | C1 | C2 | C3 | - | C4 | C5 |

Table 2: Overview of available tire compounds in the seasons 2014 to 2019. Taken from (Heilmeier et al. 2020b)

| Season | Track | Source |
|---|---|---|
| 2015 | Japan | https://www.espn.co.uk/f1/story/_/id/13768566/japanese-grand-prix-strategy-briefing |
| 2016 | Japan | https://www.espn.co.uk/f1/story/_/id/17750221/japanese-grand-prix-strategy-guide |
| 2017 | Australia | https://www.espn.com/f1/story/_/id/19005027/australian-grand-prix-race-strategy-guide |
| 2017 | Spain | https://www.espn.co.uk/f1/story/_/id/19379342/spanish-grand-prix-strategy-guide |
| 2017 | Austria | https://www.espn.com/f1/story/_/id/27087391/austrian-grand-prix-strategy-guide |
| 2017 | Belgium | https://www.espn.com/f1/story/_/id/20473131/belgian-grand-prix-strategy-guide |
| 2017 | Russia | https://www.espn.com/f1/story/_/id/19273668/russian-grand-prix-strategy-guide |
| 2018 | China | https://www.espn.com.au/f1/story/_/id/23177650/chinese-grand-prix-strategy-guide |
| 2018 | Italy | https://www.espn.com/f1/story/_/id/24553859/italian-grand-prix-strategy-guide |
| 2018 | Brazil | https://www.espn.com/f1/story/_/id/25242197/brazilian-grand-prix-strategy-guide-race-pace |

Table 3: ESPN source articles used for building fictitious tire durabilities.

# A Dynamic events

Here we present details on the generation of FCY events employed during the search phase. We allowed the simulator to accept new FCY events to be added at runtime one lap after they have been requested. This way, we ensure that FCYs in Monte-Carlo simulations appear only when they would be "visible" in the race. The FCY event's length can either be pre-specified, in the assumption that a human operator would be evaluating the race situation and inputting the predicted duration to the system, or be generated at random, to simulate multiple scenarios during the search. We ensure that the following constraints are not violated:

- When requiring at lap $l$ the generation of an event, the simulator generates an actual VSC or SC only at lap $l + 1$. As the event's starting point is represented by a cumulative race time, this constraint is necessary to ensure that the FCY event is generated in the future and started simultaneously for all drivers. If the constraint were not satisfied, lapped or slower drivers, which have a larger cumulative race time than non-lapped ones but have covered less of the race distance, could already have passed the FCY starting timestamp, leading to inconsistencies in drivers that are subject to the FCY.
- A new event can only be generated at least one lap of distance from a previous one.
- When the FCY duration is generated randomly, if an event is already running at lap p and a new FCY event is requested, the current event duration is extended by a number of laps sampled randomly, to simulate the merging of the two events.

| Season | Track | Laps | SC | VSC |
|---|---|---|---|---|
| 2015 | Japan | 53 | No | No |
| 2016 | Japan | 53 | No | No |
| 2017 | Australia | 58 | No | No |
| 2017 | Spain | 66 | No | Yes |
| 2017 | Austria | 71 | No | No |
| 2017 | Belgium | 44 | Yes | No |
| 2017 | Russia | 53 | No | Yes |
| 2018 | China | 56 | Yes | No |
| 2018 | Italy | 53 | Yes | No |
| 2018 | Brazil | 71 | No | No |

Table 4: List of the races used for evaluating planners' performance in the experiments. The laps column reports the real number of laps for each race