covering arrays. For an overview, see [73]. In general, there has been significant progress in determining upper bounds for the minimum size of the covering arrays. Furthermore, many algorithms [67, 33, 42] and tools [41] have been developed for the construction of covering arrays. In our framework, we utilize the AETG covering array generation system [41].

With covering arrays, we can define seeds, or combinations of locations that must be covered by the covering array. We can define avoids, or interactions that do not need to be covered. We can define constraints, which are interactions that should not be covered. We can also define the strength of some factors over others. All this allows for a very flexible framework when conducting conformance testing.

5.7 Case Study: Toyota Engine Controller

We consider two models of varying fidelity of an internal combustion engine. The models are academic, but of industrial complexity, and are provided by the Toyota Technical Center [1] . The first model Σ_1 is a high-fidelity engine plant and controller while the second one Σ_2 is a simplified, polynomial approximation of Σ_1. In this case, with conformance testing, we aim to explore how close the two models are and whether we can utilize Σ_2 for model predictive control [37, 38, 84, 85]. Both Σ_1 and Σ_2 are modeled in Matlab/Simulink. Σ_1 has 1878 blocks, including 10 integrator blocks, 47 lookup tables, 19 saturation blocks, 27 switch blocks, and 44 subsystem blocks. The model of the plant has 11 continuous and 68 discrete states. The controller is defined in a function block and contains $\approx$ 500 lines of code with multiple if-else conditional statements. Model Σ_2 has 1858 blocks, including 47 lookup tables, 17 saturation blocks, 27 switch blocks and 49 subsystem blocks with no continuous states and 60

[1] We note that due to confidentiality agreements, we have removed the unit measurements of the y-axes from the figures presented in this section.

discrete states. Both Σ_1 and Σ_2 have two inputs and one output each. The inputs are *Fuel Level* and the number of engine revolutions (Ne). The output is the pressure intake manifold (Pin). In the following, we run several experiments with different optimization functions. We also provide partial coverage of the input space as well as branch coverage for the controller code.

5.7.1 Experimental Design Parameters

Simulated Annealing

The search space over system inputs is defined over control points which are interpolated to generate an input signal as defined in Section 2.4. Here, both inputs have 10 control points. The search space also includes timing of the control points. Since the initial and terminal timing control points are fixed at the start and end of the simulation, respectively, for each input, we have an additional 8 search variables . The signals are interpolated through a piecewise constant interpolation function. An example input signal is presented in Fig. 5.5 (a) (Left). In total, we have 36 search variables. The stochastic optimizer utilized is a simulated annealing algorithm [2]. After 1000 tests, we found the system inputs that generate the outputs in Fig. 5.5. In the left figure, the resulting system inputs are presented. Figure 5.5 (a) (Middle) shows the system outputs, where the red out is from the complex model Σ_1, and the blue line is from the simplified model Σ_2. The figure on the right is the Pareto front over (δ, ϵ) which illustrates the ϵ difference over the δ range. The ϵ value at $\delta = 0$ represents the instantaneous difference between the outputs of the two models at the same time t, while the ϵ value at $\delta = 0.2$ represents the largest difference between the outputs of the two models while comparing the values of the two signals in a moving window of width 0.2s. Note that the Pareto Front is over 1000 tests and is therefore

an under-approximation of the *true* Pareto Front over all system behaviors. We can guarantee that (δ, ϵ) is at least as large as shown in Fig. 5.5 (a) (Right).

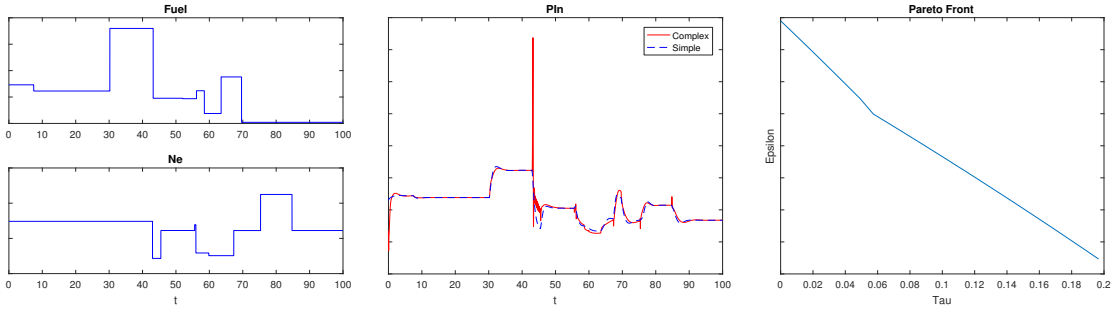
Grid Search

One of the goals in the conformance testing process is to provide a level of input search space coverage. To do so, we developed a grid search algorithm which divides the input search space in a grid. Formally, search is conducted over the following set $S = \{(x, y) : x \in [min(U_1) : \frac{range(U_1)}{g-1} : max(U_1)]\ and\ y \in [min(U_2) : \frac{range(U_2)}{g-1} : max(U_2)]\}$, where U_1, U_2 are input signals for inputs 1 and 2, respectively. Here, g is the granularity of the grid. The results of the grid search algorithm are presented in Fig. 5.5 (b). The middle figure shows highly non-conformant behavior between Σ_1 and Σ_2, indicating a possible singularity in either model at this particular input. Here, since the maximal difference between the two trajectories is constant over a period of $2 \times \delta$, the Pareto Front on the right figure is flat.

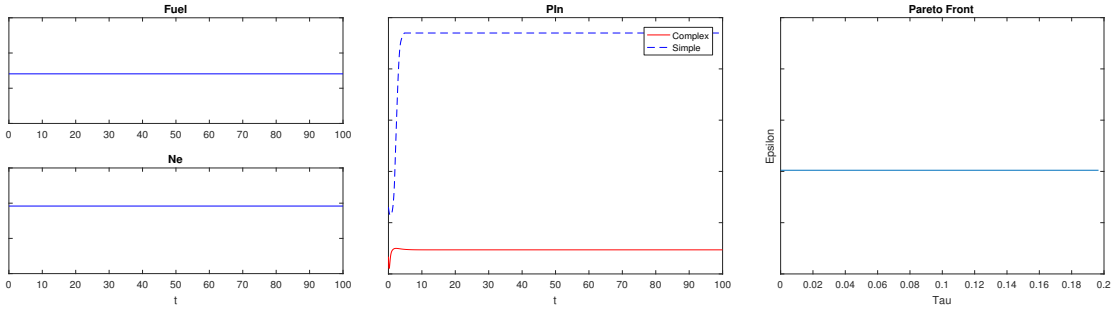
In the future, we plan to implement a grid search algorithm which includes several grids with respect to time and the input signals are interpolated linearly between them. This way, we can also study switching behaviors of the system.

Controller Branch Coverage

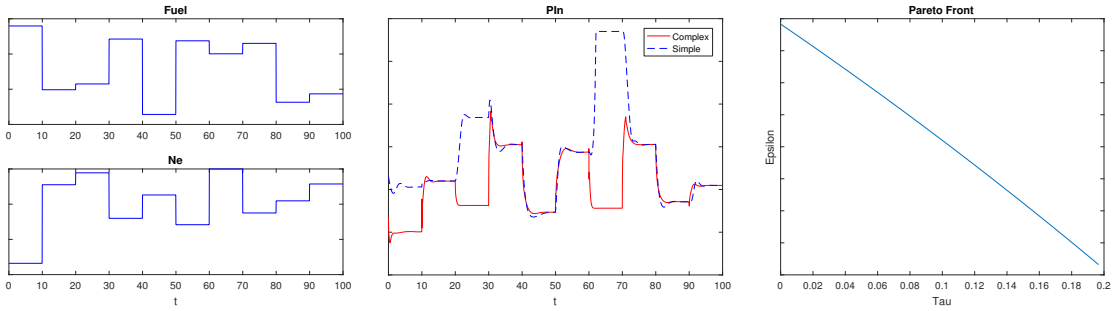
The next step in the analysis is to conduct conformance testing while making sure that we have controller branch coverage. We will consider three if-else blocks to be of particular importance. Namely M_1 with 12 branches, M_2 with 2 branches, and M_3 with 4 branches. These if-else blocks are instrumented automatically from the Simulink model. The instrumentation process extracts the branch information from the Embedded Matlab code function block and passes it to S-TaLiRo for conformance testing. We follow the approach presented in Sect. 5.4 which is related to the



(a) Simulated Annealing



(b) Grid Search



(c) Controller Branch Coverage

Figure 5.5: Experimental results under various conformance testing methods. Left: System inputs for *Fuel* and *Ne*. Middle: System output for *Pin*. Right: Pareto front over all tested system behaviors that illustrates the ε difference over the δ range.

testing approach in [55].

After running our algorithm, we found non-conformant behavior in locations $(6, 1, 4)$ for M_1, M_2, and M_3. The results are presented in Fig. 5.5 (c). The target locations $(6, 1, 4)$ were reached after 803 tests. After reaching the target locations, the stochastic optimizer was focused on maximizing the (δ, ε) metric between trajectories.

Following our testing results, Ken Butts, an Executive Engineer from the Powertrain Control Department at Toyota Technical Center provided the following statement:

"*Their tool has pointed out where the high-fidelity model is fragile and producing erroneous results. It is good to know that the polynomial model performs well at these cases.*"

5.8 Related Works

Conformance notions have been studied in the past. Tretmans [137] defined an Input-Output conformance (IOCO) notion for discrete labeled transition systems. This notion is defined as a relationship where the implementation does not generate an output that is not producible by the specification. Also, the implementation always produces an output when it is required by the specification. Later, Van Osch [114] extended IOCO to hybrid transition systems (HTS) by incorporating continuous-time inputs.

In [141], the authors extend the work by [137] where the implementation is a black box that generates trajectories. In this framework, the specification is represented as a timed automaton. Here, every trajectory in the language of the implementation needs to satisfy the timed automaton representing the specification.

In the work by Brandl et al. [28] the authors present a method for conformance checking through qualitative reasoning techniques. The method utilzes mutation-