# Classification of Driving Behaviors Using STL Formulas: A Comparative Study

Ruya Karagulle[1(✉)], Nikos Aréchiga[2], Jonathan DeCastro[2], and Necmiye Ozay[1]

[1] University of Michigan, Ann Arbor, MI 48109, USA
{ruyakrgl,necmiye}@umich.edu
[2] Toyota Research Institute, Cambridge, MA, USA
{nikos.arechiga,jonathan.decastro}@tri.global

**Abstract.** In this paper, we conduct a preliminary comparative study of classification of longitudinal driving behavior using Signal Temporal Logic (STL) formulas. The goal of the classification problem is to distinguish between different driving styles or vehicles. The results can be used to design and test autonomous vehicle policies. We work on a real-life dataset, the Highway Drone Dataset (HighD). To solve this problem, our first approach starts with a formula template and reduces the classification problem to a Mixed-Integer Linear Program (MILP). Solving MILPs becomes computationally challenging with increasing number of variables and constraints. We propose two improvements to split the classification problem into smaller ones. We prove that these simpler problems are related to the original classification problem in a way that their feasibility imply that of the original. Finally, we compare our MILP formulation with an existing STL-based classification tool, LoTuS, in terms of accuracy and execution time.

**Keywords:** driving behavior · STL classification · formal methods

## 1 Introduction

A key factor in developing high-quality and robust policies for autonomous driving is strong understanding of the driving environment. An essential component to understanding this environment is high-quality prediction models of driving behaviors. In this paper, we compare methods to classify driving behaviors as exhibited in real-world data. We use the HighD dataset to work on naturalistic vehicle trajectories [13]. The dataset comprises 110500 vehicle tracks recorded on German highways. As our case study, we consider the task of distinguishing the longitudinal driving behavior of cars and trucks in this dataset.

Existing work in time series classification provides a variety of methods that can be used to classify behaviors. However, time-series classifiers such as Long Short-term Memory (LSTM) [11] or classification using Dynamic Time Warping [4] frequently lack interpretability. In the early stages of system development,


S. Bogomolov and D. Parker (Eds.): FORMATS 2022, LNCS 13465, pp. 153–162, 2022.
https://doi.org/10.1007/978-3-031-15839-1_9

human engineers are heavily involved in interacting with prediction models, as well as debugging erroneous conditions that may arise in either simulation or real-world testing. Engineers would greatly benefit from interpretable classification and prediction models, which would help them to better understand the root cause of failures as well as possible solutions. Temporal Logic (TL) [17] is extensively used to describe the behavior of cyber-physical systems [1,6–8]. The symbolic nature of temporal logic specifications make them a good candidate for interpretable classifiers. Moreover, they can be used for runtime decision-making.

Signal Temporal Logic (STL) [14] is a variant of temporal logic that can be used to reason about continuous, discrete, or even hybrid signals. In addition, several different quantitative semantics have been proposed for STL. The quantitative semantics can be used to compute the *robustness* metric, which is a measure of the degree of satisfaction of signals for a given STL formula and a signal trace. Robustness is a sound and non-smooth function [19].

TL formula inference problem tries to learn a TL formula from the data and it has been studied before in the literature [5,9,12,15]. [5,9,12] use another variant of STL called Parametric STL (PSTL) where numerical values of the formula are interpreted as unknown parameters. In [3], the authors approach the two-class classification problem as a statistical learning problem. Given the template formula, they explore the parameters using statistical model checking. The paper [12] defines a directed acyclic graph (DAG) of formula templates and searches over this graph, after proving the partial ordering of formulas. Their loss function is the number of misclassifications and the length of the formula (the number of linear predicates that appear in the formula). In [5], they propose a decision tree approach for both online and offline learning using STL formula. They incrementally update the binary tree linked to the STL formula decision and search for the smallest formula that can be constructed from a set of primitives. In this paper, we use their toolbox LoTuS for comparison. [15] solves a series of satisfiability problems in Boolean logic to obtain the smallest linear temporal logic formula possible.

Learning from formula templates can be used for car-truck classification as well. It is known that trucks tend to go slower than cars since they are heavier and tend to maintain a longer distance from lead vehicles since their deceleration rates are slower than cars. By considering this knowledge and taking inspiration from Adaptive Cruise Control (ACC) driving specifications [16], we develop an STL formula template. We recast the classification problem to an appropriate MILP problem in parameters of the STL formula template, and we optimize the robustness of the STL formula to find the optimal parameters. Since MILP problems can blow up in execution time with the number of variables and constraints, we propose two separation methods to improve execution time. The first separation is over the formula template and the second one is over the data. Although optimality is not preserved, we prove that the parameters found after improvements are also in the feasible domain for the original problem. We compare the MILP formulation with [5] and provide quantitative results for test error and for execution times. Limitations of both approaches are discussed.

## 2 Preliminaries

STL is defined with the syntax $\phi := \top \mid \pi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathcal{U}_I \phi_2$. The boolean true is $\top$, and $\pi$ is a predicate of the form $f(s(t)) \leq \mu$ where $f : \mathbb{R}^n \to \mathbb{R}$, $\pi = \top$ when inequality holds. The logical not is $\neg$, and $\wedge$ is the conjunction operator. The temporal operator "Until" with the bounded interval I is shown as $\mathcal{U}_I$. Always $\square_I$, eventually $\Diamond_I$, disjunction $\vee$, are defined as $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\Diamond_I \phi = \top \mathcal{U}_I \phi$, and $\square_I \phi = \neg\Diamond_I \neg\phi$.

If a signal $s$ satisfies a formula $\phi$, it is shown as $s \models \phi$, and if it violates, it is shown as $s \not\models \phi$. Satisfaction rules are given by the qualitative semantics:

$$
\begin{array}{ll}
s(t) \models \pi & \iff \pi \\
s(t) \models \neg\phi & \iff \neg(s(t) \models \phi) \\
s(t) \models \phi_1 \wedge \phi_2 & \iff ((s(t) \models \phi_1) \wedge (s(t) \models \phi_2)) \\
s(t) \models \phi_1 \mathcal{U}_{[a,b]} \phi_2 & \iff \exists t' \in [t+a, t+b] \text{ s.t. } (s(t') \models \phi_2) \wedge (\forall t'' \in [t, t'](s(t'') \models \phi_1))
\end{array}
$$

In addition to their boolean semantics, STL formulas have a quantitative semantics, which quantify the degree to which a formula is satisfied or falsified. The quantitative semantics are defined with a metric called *robustness*. Given the signal and the formula, if the robustness metric is positive, then the signal satisfies the formula and vice versa. The robustness of the logical truth is $\rho(s, \top, t) = +\infty$ and the robustness of the predicate $\phi = f(s) \leq \mu$ is $\rho(s, \phi, t) = \mu - f(s(t))$. The robustness of a formula is defined recursively as follows:

$$
\begin{array}{ll}
\rho(s, \neg\phi, t) & = -\rho(s, \phi, t) \\
\rho(s, \phi_1 \wedge \phi_2, t) & = \min(\rho(s, \phi_1, t), \rho(s, \phi_2, t)) \\
\rho(s, \phi_1 \mathcal{U}_{[a,b]} \phi_2, t) & = \max_{t' \in [t+a, t+b]} (\min(\rho(s, \phi_2, t'), \min_{t'' \in [t,t']} \rho(s, \phi_1, t'')))
\end{array}
$$

Robustness of derived operators can be found by re-writing the operator in terms of the primitive operators. In STL formulas, the predicate values $\mu$ and time intervals $I$ are known. In [2], authors define another variant of STL, called Parametric Signal Temporal Logic (PSTL), where those values can be defined as parameters. A PSTL formula $\phi_{\boldsymbol{\mu}}$ will become an STL formula $\phi$ with a corresponding valuation of parameters $\boldsymbol{\mu}$, where $\boldsymbol{\mu}$ is the set of both scale and time parameters. With slight abuse of notation, we use $\varphi_\mu$ for corresponding valuation vector $\mu$ of parameters $\boldsymbol{\mu}$. We define an *STL formula template* as a PSTL formula with a set of unknown parameters.

## 3 Specifying Longitudinal Driving Behavior with STL

For the longitudinal driving scenario considered in this paper, the sample signal is two-dimensional, speed in $[km/h]$ and time headway in $[s]$, that is $s_i = \begin{bmatrix} v_i & w_i \end{bmatrix}^\top$, where $v_i \in \mathbb{R}^{T_i}$ and $w_i \in \mathbb{R}^{T_i}$ are the time series of speed and time headway values, respectively, with signal duration $T_i$. We want to find the formula that

separates trucks from cars. From the intuition that trucks keep different speed and time headway than cars, we propose the following STL formula template:

$$\begin{aligned}\varphi = \square\{\square_{[0,\tau_1]}(w \geq w_\epsilon) \implies \lozenge_{[0,\tau_2]}(\square(v \leq (1+\epsilon)v_{des}) \wedge \square(v \geq (1-\epsilon)v_{des}) \vee w < w_\epsilon) \\ \wedge \square_{[0,\tau_1]}(w < w_\epsilon) \implies \lozenge_{[0,\tau_2]}(\square(w \leq (1+\epsilon)w_{des}) \wedge \square(w \geq (1-\epsilon)w_{des}) \vee w \geq w_\epsilon)\},\end{aligned} \tag{1}$$

where $w_\epsilon$ is the time headway threshold, $\tau_1$ is the cause time interval, $\tau_2$ is the effect time interval, $\epsilon$ is the acceptance threshold for desired values, and $v_{des}$ and $w_{des}$ are desired speed and desired time headway, respectively. This formula comprises of two subformulas. The first subformula $\varphi_1 = \square_{[0,\tau_1]}(w \geq w_\epsilon) \implies \lozenge_{[0,\tau_2]}(\square(v \leq (1+\epsilon)v_{des} \wedge v \geq (1-\epsilon)v_{des}) \vee w > w_\epsilon)$ semantically represents that if there is no lead vehicle in close distance (time headway more than $w_\epsilon$) for $\tau_1$ seconds, the ego vehicle eventually reaches it desired speed $v_{des}$ or lead vehicle happens to be in ego vehicle's close distance. This is similar to ACC set speed mode. The second subformula represents ACC time gap mode. $\varphi_2 = \square_{[0,\tau_1]}(w < w_\epsilon) \implies \lozenge_{[0,\tau_2]}(\square(w \leq (1+\epsilon)w_{des} \wedge w \geq (1-\epsilon)w_{des}) \vee w \geq w_\epsilon)$ means if there is a lead vehicle in front of the ego vehicle in time interval $[0, \tau_1]$, ego vehicle eventually reaches its desired time headway $w_{des}$ in time interval $[0, \tau_2]$ or the time headway increases. Here in this formula, it is assumed that $w_\epsilon, \epsilon, \tau_1$ and $\tau_2$ are known values. $\boldsymbol{\mu} = \boldsymbol{\mu_1} \cup \boldsymbol{\mu_2} = \{v_{des}, w_{des}\}$ are unknown parameters. Note that the formula can be seen as $\varphi_{\boldsymbol{\mu}} = \square(\varphi_{1,\boldsymbol{\mu_1}} \wedge \varphi_{2,\boldsymbol{\mu_2}})$.

## 4 Methods

Let $\mathcal{S} = \{(s_i, y_i)\}_{i=1}^N$ be the signal set. Let $s_i$ be the $\mathrm{i}^{\mathrm{th}}$ sample signal, and $y_i \in \{0, 1\}$ be its label. We assume there are two classes of signals that we wish to classify: Class 0 signals are denoted as $\mathcal{S}_0 := \{s_i \in \mathcal{S} : y_i = 0\}$, and class 1 signals are denoted as $\mathcal{S}_1 := \{s_i \in \mathcal{S} : y_i = 1\}$. The problem this paper addresses is to find an STL formula satisfied by class 0 signals and violated by class 1.

We can attempt to solve this problem by proposing a PSTL template and then solving an optimization problem over the formula parameters to maximize the robustness of the formula over class 0 while minimizing the robustness over class 1. This method can be useful when domain-specific knowledge allows proposing a suitable PSTL template, but specific parameters are unknown. Assuming that signal classes are separable with the formula above, the problem:

$$\begin{aligned} \max_{r,\mu} \quad & r \\ \text{s.t.} \quad & \rho(s_i, \varphi_\mu, 0) \geq r \quad \forall s_i \in \mathcal{S}_0 \\ & \rho(s_i, \varphi_\mu, 0) \leq -r \; \forall s_i \in \mathcal{S}_1 \\ & r \geq 0 \end{aligned} \tag{2}$$

returns the optimal parameter set. In particular, Problem (2) tries to find the parameter set that maximizes the margin $r$ between two classes.

The classes might not be separable in reality, at least for two reasons. First, in real-life scenarios, it is likely to have outliers. Second, the formula template is heuristically selected. It is not guaranteed that experts are competent enough

to separate the dataset with respect to the template or this separation may not be that obvious to human observation. Hence, we also consider a soft margin version of Problem (2):

$$\begin{aligned} \max_{r,\mu}\ & r - \theta(\textstyle\sum_i \zeta_i^+ + \sum_i \zeta_i^-) \\ \text{s.t.}\ & \rho(s_i, \varphi_\mu, 0) + \zeta_i^+ \geq r \quad \forall s_i \in \mathcal{S}_0 \\ & \rho(s_i, \varphi_\mu, 0) - \zeta_i^- \leq -r \ \forall s_i \in \mathcal{S}_1 \\ & r, \zeta_i^+, \zeta_i^- \geq 0, \end{aligned} \tag{3}$$

where $\zeta_i^+$ and $\zeta_i^-$ are slack variables, $\theta$ is a weight that penalizes the violation of margins. In this formulation, class 0 signals can have negative robustness and class 1 signals can have positive robustness.

Both problems (2) and (3) can be converted to MILP using standard encodings of robustness metrics [18]. The performance of a MILP depends on the number of variables and the number of constraints. Here, each signal adds new sets of variables and constraints. In addition, the number of constraints increases with increasing formula complexity, i.e., the number of operators in the formula. With large datasets and complex formulas such as HighD and the formula above, Problems (2) and (3) may not be solved in reasonable time. To address this issue, we propose two improvements and one formula template relaxation. First improvement uses the structure of the formula (1).

**Proposition 1 (Formula Separation).** *Given any PSTL formula of the form* $\varphi_{\boldsymbol{\mu}} = \square(\varphi_{1,\boldsymbol{\mu}_1} \wedge \varphi_{2,\boldsymbol{\mu}_2})$*, with* $\boldsymbol{\mu}_1$*,* $\boldsymbol{\mu}_2$ *disjoint, consider the following optimization problems:*

$$\begin{aligned} \max_{r_1,\mu_1}\ & r_1 \\ s.t.\ & \rho(s_i, \varphi_{1,\mu_1}, 0) \geq r_1 \quad \forall s_i \in \mathcal{S}_0 \\ & \rho(s_i, \varphi_{1,\mu_1}, 0) \leq -r_1 \ \forall s_i \in \mathcal{S}_1 \\ & r_1 \geq 0, \end{aligned} \tag{4}$$

$$\begin{aligned} \max_{r_2,\mu_2}\ & r_2 \\ s.t.\ & \rho(s_i, \varphi_{2,\mu_2}, 0) \geq r_2 \quad \forall s_i \in \mathcal{S}_0 \\ & \rho(s_i, \varphi_{2,\mu_2}, 0) \leq -r_2 \ \forall s_i \in \mathcal{S}_1 \\ & r_2 \geq 0 \end{aligned} \tag{5}$$

*If Problems (4) and (5) are feasible, then Problem (2) is feasible. Specifically, if* $\mu_1^*$*,* $\mu_2^*$*,* $r_1^*$*,* $r_2^*$ *are the optimizers of Problem (4) and (5), then* $\tilde{\mu} = \begin{bmatrix}\mu_1^* & \mu_2^*\end{bmatrix}$ *and* $\tilde{r} = \min(r_1^*, r_2^*)$ *is a feasible solution for Problem (2).*

Proof of Proposition 1 can be found in [10]. Since the formula (1) satisfies the conditions in Proposition 1, we are able to halve the number of variables and number of constraints for each subproblem.

Next, we propose a relaxation of the formula template (1), inspired by [16]. Assuming that trucks move slower than cars, instead of using $\varphi_{1,\boldsymbol{\mu}_1}$, we can remove the upper bound and range acceptance threshold $\epsilon$. The new formula becomes $\tilde{\varphi}_{1,\boldsymbol{\mu}_1} = \square_{[0,\tau_1]}(w \geq w_\epsilon) \implies \Diamond_{[0,\tau_2]}(\square(v \leq \mu_1) \vee w < w_\epsilon)$. Note that this formula is to be satisfied by trucks. Therefore, trucks are set to class 0. Formula $\tilde{\varphi}_{1,\boldsymbol{\mu}_1}$ tries to find limit speed that distinguishes two classes. Same approach can be applied for time headway. Since trucks are keeping longer time

headways, we can remove the lower bound. Second subformula will be $\varphi_{2,\boldsymbol{\mu_2}} = \square_{[0,\tau_1]}(\square w < w_\epsilon) \implies \Diamond_{[0,\tau_2]}(\square(w \geq \mu_2) \vee w \geq w_\epsilon)$. Finally, we obtain

$$\tilde{\varphi}_{\boldsymbol{\mu}} = \square(\tilde{\varphi}_{1,\boldsymbol{\mu_1}} \wedge \tilde{\varphi}_{2,\boldsymbol{\mu_2}}). \tag{6}$$

Note that we can apply Proposition 1 to template (6). With the help of formula (6), we are able to discard two sets of constraints per each input. However, number of variables and constraints still depends on number of inputs and large input sets are still not solvable in reasonable time even with these improvements. Instead of solving the large input set as a whole, we can divide it into smaller chunks and handle them separately.

**Proposition 2 (Data Separation).** *Take one of the formulas $\tilde{\varphi}_{i,\boldsymbol{\mu_i}}$, $i \in \{1,2\}$ from (6). Consider a dataset $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1$, where $\mathcal{S}_0$ denotes trucks and $\mathcal{S}_1$ denotes cars. Partition $\mathcal{S}_0$ and $\mathcal{S}_1$ to form $B$ batches $\{\mathcal{S}^i = (\mathcal{S}_0^i, \mathcal{S}_1^i)\}_{i=1}^B$. If Problem (4) solved for $\mathcal{S}$ is feasible, then Problem (4) is feasible for every batch $\mathcal{S}^i$. Conversely, let $\mu_{1,i}^*$ and $r_i^*$ be the optimizers of Problem (4) solved for $\mathcal{S}^i$ for each $i$. Then, $\tilde{\mu}_1 = 1/B \sum_{i=1}^B \mu_{1,i}^*$ gives a feasible solution with $0 \leq \tilde{r} \leq \min_i(r_i^*)$ for Problem (4) for $\mathcal{S}$.*

Proof can be found in [10]. This proposition depends on class definitions, satisfaction rules posed in the problem setup, and formula template. Although we are losing optimality, we can still recover feasible STL formulas that are satisfied by class 0 and violated by class 1 signals. We use Proposition 2 for soft margin problem as well. Soft margin problem is not a feasibility problem, in fact every solution is feasible in that formulation. By dividing the dataset into smaller parts, we are finding optimal decision boundaries for each subproblem. Our intuition is by taking the mean of optimal parameters, we approach minimum error in total.

## 5 Experiments

HighD dataset consists of vehicle trajectory data from different highway sections. Since we are interested in longitudinal driving behavior, we discard vehicles that change lanes. Speed and time headway signals have different orders of magnitudes, effecting robustness differently. To avoid this, we normalize the data for training, and then denormalize to report the learned parameter values. Among all vehicles in the dataset, 79.24% of them of them are cars, the rest are trucks.

Initially, we conduct a baseline experiment. We use one of the well-known and interpretable classifiers, Support Vector Machines (SVM). The accuracy of SVM over HigD dataset is 80.59%. It is slightly more than assuming that all vehicles are cars. Therefore, it is clear that we need different approaches. Additional information for baseline experiment can be found in [10]. Next, we execute four different MILP instances: (M1) Soft margin MILP approach (Problem (3)) with formula template (1) without any improvement, (M2) using Proposition 1, (M3) Problem (3) with formula template (6) using only Proposition 1 and (M4) Problem (3) with formula template (6) using Proposition 1 and Proposition 2.

Known values of templates (1) and (6) are as follows: $w_\epsilon = 3$, $\tau_1 = 2$, $\tau_2 = 3$ and $\epsilon = 0.2$. We randomly partitioned data into batches of four inputs. MILP method does not require equal signal lengths but LoTuS needs equal signal lengths. In the first part of the experiments we compare only MILP instances with full data length. In the second part, for the sake of fair comparison with LoTuS, we truncate the data. There are two comparison metrics: test error and execution time. Training set is balanced with equal number of cars and trucks. However, the test set is not balanced. For all MILP instances, we use an off-the-shelf optimization solver, Gurobi. We set Gurobi's solving time limit to one hour per parameter and optimality gap to 5%. When time limit is reached, Gurobi returns its incumbent solution if there exists any. The optimality of the incumbent solution is not guaranteed. In tables below, "∼" means that Gurobi cannot find a feasible solution within the time limit. For tests, a Macbook Pro with 2 GHz Quad-Core Intel Core i5 processors and 16 GB RAM is used.

Results for full length can be found in Table 1. The first column represents the number of training inputs. Methods (M1)–(M4) represent methods that are described in the beginning of this section. The minimum test error occurs when formula template (6) with MILP instance (M4) is trained with 2000 inputs, and $\tilde{\varphi} = \Box\{\Box_{[0,2]}(w \geq 3) \implies \Diamond_{[0,3]}(\Box(v \leq 98.95) \vee w < 3) \wedge \Box_{[0,2]}(w < 3) \implies \Diamond_{[0,3]}(\Box(w \geq 2.047) \vee w \geq 3)\}$ is the STL formula. When shorter execution time is an important criterion, formula template (6) trained with 1000 inputs gives comparable results. The test error increases by 0.04% but execution times decreases almost 2.5 times. Optimal parameters for this instance are $\mu_{1000} = \begin{bmatrix}97.13 & 1.99\end{bmatrix}$. Execution times between MILP methods decrease significantly with each improvement.

**Table 1.** Comparison among MILP methods with full time length

| Inputs | (M1) | | (M2) | | (M3) | | (M4) | |
|---|---|---|---|---|---|---|---|---|
| | Error % | Time [s] | Error % | Time [s] | Error % | Time [s] | Error % | Time [s] |
| 8 | 18.09 | 697 | 14.11 | 4011 | 14.29 | 11.71 | 13.18 | 3.057 |
| 20 | 25.07 | 7201 | 25.14 | 7201 | 12.87 | 119.7 | 13.19 | 4.991 |
| 40 | ∼ | ∼ | ∼ | ∼ | 13.87 | 725.3 | 13.37 | 31.24 |
| 200 | ∼ | ∼ | ∼ | ∼ | 15.02 | 7204 | 12.15 | 145.34 |
| 1000 | ∼ | ∼ | ∼ | ∼ | 43.95 | 7214 | **11.35** | **675.6** |
| 2000 | ∼ | ∼ | ∼ | ∼ | 79.83 | 7218 | **11.31** | **1507** |

Table 2 shows comparative results of each MILP instance with LoTuS on truncated data. When data is truncated, we lose information during truncation, and hence, test error increases. The minimum test error is obtained with LoTuS when trained with 200 inputs. The obtained formula is

$\varphi_{\text{LoTuS}_{2000}} = ((\square_{[1e-06,0.5]} v < 90.4 \wedge (\square_{[8.33,24]} v < 87.5 \vee (\Diamond_{[8.33,24]} v > 87.5 \wedge \Diamond_{[15.3,24]} w > 1.74))) \vee (\Diamond_{[1e-06,0.5]} v > 90.4 \wedge (\square_{[0.369,19.7]} v < 97.8 \wedge \Diamond_{[3.73,20]} w > 2.08)))$.

**Table 2.** Comparison among MILP methods and LoTuS with truncated data

| Inputs | (M3) | | (M4) | | LoTuS | |
|---|---|---|---|---|---|---|
| | Error % | Time [s] | Error % | Time [s] | Error % | Time [s] |
| 8 | 16.88 | 3.011 | 17.37 | 1.894 | 17.00 | 7.098 |
| 20 | 16.76 | 10.80 | 21.86 | 5.607 | 16.22 | 5.319 |
| 40 | 13.17 | 152.5 | 16.89 | 10.20 | 15.15 | 9.749 |
| 200 | 14.03 | 7213 | 18.79 | 40.09 | **13.69** | **15.51** |
| 1000 | 30.06 | 7234 | 18.50 | 177.9 | 15.72 | 43.72 |
| 2000 | 79.83 | 7219 | 18.65 | 353.5 | 14.66 | 51.7 |

This formula is not as interpretable as the formula templates used in MILP. Besides, interpretability decreases as the number of inputs increases. E.g., the STL formula that LoTuS finds for eight inputs is $\square_{[8.52,11.3]} v < 115$ whereas, for 2000 inputs is $\varphi_{\text{LoTuS}_{2000}}$. We can say that the test error is lower when using MILP with full length.

## 6 Discussion and Conclusions

In this paper, we considered STL-based classification of driving behaviors. First, we came up with an STL formula template and recast the classification problem with a template as a MILP. We observed MILP-based solutions suffer from computation complexity. We proposed two improvements to address the scalability issue. This approach was compared with the toolbox LoTuS in terms of accuracy and computation time. Both methods have some drawbacks. MILP approach requires domain knowledge to come up with a template and it cannot search for time parameters effectively. LoTuS has a set of primitives to search over (limited to eventually always and always eventually) and it connects them with disjunction or conjunction, not allowing further nesting. This restricts possible formula types that can be obtained. In addition, LoTuS loses interpretability as the training dataset grows. Further research is needed to find a better balance between scalability, interpretability, and accuracy on real datasets. For scalability, one can try satisfiability modulo convex optimization approaches, which have been shown to improve practical performance compared to MILP on some STL problems in recent years. For interpretability, using two different formula templates, one per class, might increase flexibility.

**Acknowledgements.** Toyota Research Institute provided funds to support this work.

## References

1. Abbas, H., Hoxha, B., Fainekos, G., Ueda, K.: Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In: 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems, pp. 1–6. IEEE-CYBER 2014, October 2014. https://doi.org/10.1109/CYBER.2014.6917426

2. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric identification of temporal properties. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 147–160. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_12
3. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 23–37. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10512-3_3
4. Berndt, D., Clifford, J.: Using dynamic time warping to find patterns in time series. In: Workshop on Knowledge Knowledge Discovery in Databases, vol. 398, pp. 359–370 (1994). http://www.aaai.org/Papers/Workshops/1994/WS-94-03/WS94-03-031.pdf
5. Bombara, G., Belta, C.: Offline and online learning of signal temporal logic formulae using decision trees. ACM Trans. Cyber-Phys. Syst. **5**(3), 1–23 (2021). https://doi.org/10.1145/3433994
6. Chou, G., Ozay, N., Berenson, D.: Explaining multi-stage tasks by learning temporal logic formulas from suboptimal demonstrations. In: Proceedings of Robotics: Science and Systems. Corvalis, Oregon, USA, July 2020. https://doi.org/10.15607/RSS.2020.XVI.097
7. Fainekos, G.E., Girard, A., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for dynamic robots. Automatica **45**(2), 343–352 (2009). https://doi.org/10.1016/j.automatica.2008.08.008, https://linkinghub.elsevier.com/retrieve/pii/S000510980800455X
8. Garg, K., Panagou, D.: Control-lyapunov and control-barrier functions based quadratic program for spatio-temporal specifications. In: 2019 IEEE 58th Conference on Decision and Control (CDC), pp. 1422–1429 (2019). https://doi.org/10.1109/CDC40024.2019.9029666
9. Jones, A., Kong, Z., Belta, C.: Anomaly detection in cyber-physical systems: a formal methods approach. In: Proceedings of the IEEE Conference on Decision and Control 2015-February (February), pp. 848–853 (2014). https://doi.org/10.1109/CDC.2014.7039487
10. Karagulle, R., Aréchiga, N., Decastro, J., Ozay, N.: Classification of driving behaviors using STL formulas: a comparative study (2022). https://doi.org/10.7302/4872, https://deepblue.lib.umich.edu/handle/2027.42/173041
11. Karim, F., Majumdar, S., Darabi, H., Chen, S.: LSTM Fully Convolutional Networks for Time Series Classification. IEEE Access **6**, 1662–1669 (2017). https://doi.org/10.1109/ACCESS.2017.2779939
12. Kong, Z., Jones, A., Ayala, A.M., Gol, E.A., Belta, C.: Temporal logic inference for classification and prediction from data. In: HSCC 2014 - Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control (Part of CPS Week), pp. 273–282 (2014). https://doi.org/10.1145/2562059.2562146, http://dx.doi.org/10.1145/2562059.2562146
13. Krajewski, R., Bock, J., Kloeker, L., Eckstein, L.: The highD dataset: a drone dataset of naturalistic vehicle trajectories on German highways for validation of highly automated driving systems. In: 2018 IEEE 21st International Conference on Intelligent Transportation Systems (ITSC) (2018). https://doi.org/10.1109/ITSC.2018.8569552
14. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12

15. Neider, D., Gavran, I.: Learning linear temporal properties. In: Proceedings of the 18th Conference on Formal Methods in Computer-Aided Design, FMCAD 2018, pp. 148–157, January 2019. https://doi.org/10.23919/FMCAD.2018.8603016
16. Nilsson, P., et al.: Correct-by-construction adaptive cruise control: two approaches. IEEE Trans. Control Syst. Technol. **24**(4), 1294–1307 (2016). https://doi.org/10.1109/TCST.2015.2501351
17. Pnueli, A.: The temporal logic of programs. In: Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS, vol. 1977-October, pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/sfcs.1977.32
18. Raman, V., Donzé, A., Maasoumy, M., Murray, R.M., Sangiovanni-Vincentelli, A., Seshia, S.A.: Model predictive control with signal temporal logic specifications. In: 53rd IEEE Conference on Decision and Control, pp. 81–87 (2014). https://doi.org/10.1109/CDC.2014.7039363
19. Varnai, P., Dimarogonas, D.V.: On robustness metrics for learning STL tasks. In: Proceedings of the American Control Conference 2020-July, pp. 5394–5399 (2020). https://doi.org/10.23919/ACC45564.2020.9147692