MLPP-DCS: Machine Learning-based Performance Prediction for Discrete Controller Synthesis

1st Takumi Ikeda
Waseda University
Tokyo, Japan
ti-2236@fuji.waseda.jp

2nd Takanori Hirano
Waseda University
Tokyo, Japan
cincischio@ruri.waseda.jp

3rd Takuto Yamauchi
Waseda University
Tokyo, Japan
takuto.yamauchi@aoni.waseda.jp

4th Jialong Li
Waseda University
Tokyo, Japan
lijialong@fuji.waseda.jp

5th Kenji Tei
Tokyo Institute of Technology
Tokyo, Japan
tei@c.titech.ac.jp

***Abstract*—Discrete Controller Synthesis (DCS) is an automated technology designed to synthesize controllers that ensure safety within specified environments. It utilizes environment and monitor models described by Labelled Transition Systems. However, DCS faces a significant challenge: the exponential growth of computational space as the increase in the number of environment models. This paper introduces Machine Learning-based Performance Prediction for DCS (MLPP-DCS), aimed at predicting the computation time and maximum memory usage required for DCS. MLPP-DCS generates predictors using machine learning (ML)-based algorithms that take the characteristics of the environment and monitor models as inputs. Our predictor generation algorithms leverage four ML techniques. Evaluation results demonstrate its effectiveness, achieving a coefficient of determination of up to 0.814 for predicting computation time and 0.536 for predicting maximum memory usage.**

***Index Terms*—Discrete Controller Synthesis, Labelled Transition System, Machine Learning**

I. Introduction

A Discrete Event System (DES) is a dynamic system characterized by state changes that occur in response to external events. Common examples of such events include machine failures or task completions within a manufacturing system.

P. Ramadge et al. [1] introduced Discrete Controller Synthesis (DCS), a technique for automatically generating controllers that guarantee safety properties within DES environments. Here, safety properties are defined as ensuring that "bad things do not happen" during the execution of the controller [2]. DCS operates by utilizing an environment model, representing the system's surroundings, and a monitor model, defining the safety properties that need to be guaranteed. From these models, DCS can automatically synthesize a controller that guarantees safety within the specified environment.

However, DCS faces significant challenges due to the exponential growth in computational space as more environment models are added. This results in prohibitive computation times and maximum memory usage, particularly when dealing with numerous environment models, thereby constraining the broad application of DCS [3]. Addressing these computational challenges is crucial, as their management can influence decision-making during development and potentially broaden the applicability of DCS.

To this end, this paper proposes Machine Learning-based Performance Prediction for DCS (MLPP-DCS), aimed at predicting performance parameters, specifically computation time and maximum memory usage required for DCS. In MLPP-DCS, predictors are generated using machine learning (ML)-based algorithms that take the characteristics of the environment and monitor models as inputs. The ML-based predictor generation algorithms leverage four ML techniques.

The contributions of this paper are as follows:

- Proposal of ML-based performance predictor generation algorithms for DCS (ML-based DCS predictor generation algorithms)
- Evaluation of the algorithms through the generation and validation of 8 DCS predictors (2 performance parameters $\times$ 4 ML-based DCS predictor generation algorithms)

The structure of the rest of the paper is as follows: Section II and III explain DCS and ML algorithms as the background technology, respectively. Section IV describes MLPP-DCS. Section V presents the evaluation of MLPP-DCS. Section VI discusses related research, and finally, Section VII provides the conclusion and future work.

II. Discrete Controller Synthesis

DCS takes as input environment models $e \in E$ and monitor models $r \in R$, which set by the developer, and output a controller c that represents operational specifications to ensure safety within the environment models. The environment model e represents the operational context of the system, while the monitor model r specifies the safety properties that must be ensured in the environment models. The elements e, r and c in DCS are represented using the Labelled Transition System (LTS) defined as $m = (S_m, A_m, \Delta_m, s_m^0)$ where:

- S_m is a finite set of states.
- $A_m = A_m^+ \cup A_m^-$ is a set of actions, where A_m^+ and A_m^- represent controllable and uncontrollable actions by the system, respectively.

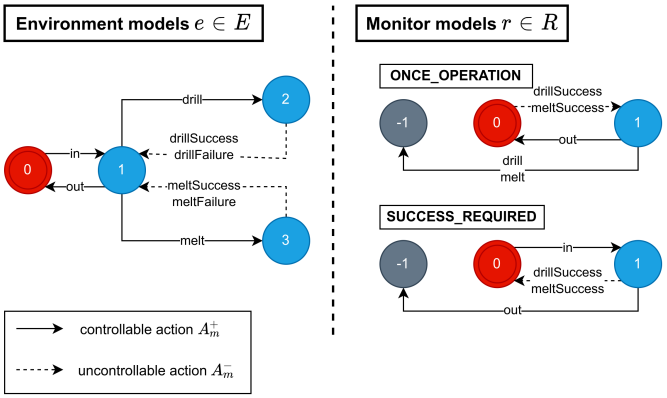

Fig. 1. Environment and monitor models for chocolate manufacturing system

- $(s, a, s') \in \Delta_m$ is a transition relation, meaning that state $s \in S_m$ transitions to state $s' \in S_m$ by action $a \in A_m$.
- $s_m^0 \in S_m$ is the initial state.

We explain the DCS algorithm using an example of a chocolate manufacturing system. The steps of the system are as follows: First, the materials used for chocolate manufacturing are added. Next, the materials are either melted or drilled. Finally, if the melting or drilling operation is successful, the finished product is produced, and the process returns to the first step. The environment and monitor models of the system are shown in Fig. 1. The monitor models are defined by "ONCE_OPERATION" and "SUCCESS_REQUIRED", representing the safety properties "perform the operation only once" and "do not produce if unsuccessful", respectively. The violation state in the monitor model is indicated by -1.

The steps of DCS algorithm are described as follows: First, the environment models E are synthesized to generate the monolithic environment model m (**Parallel Composition**). Second, the game space g is synthesized from the monolithic environment model m and the monitor models R (**Modified Parallel Composition**). Third, since the generated game space g has multiple violation states, these are compressed into a single violation state s_{err} (**Error State Abstraction**). The game space generated in this step is denoted as the abstracted game space g'. The abstracted game space g' for the chocolate manufacturing example is shown in Fig. 2. Finally, by inputting the abstracted game space g' into the **Safety Game Solver**, the controller c is output. The Safety Game Solver uses two-player game theory to explore the transitions in g' through backward propagation from s_{err}, deriving a controller composed only of transitions that do not lead to s_{err}.

III. Machine Learning Algorithms

In this paper, we employ four different ML algorithms: Gradient Boosting Decision Tree (GBDT), Random Forest (RF), Decision Tree (DT), and Logistic Regression (LR). These well-known ML algorithms have been applied in various fields, including remote vehicle control [4], network intrusion detection systems [5], computer vision [6], and robot sensing systems [7].

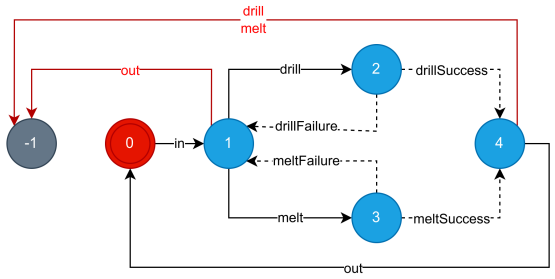

Fig. 2. Abstracted game space g' for chocolate manufacturing system

IV. MLPP-DCS: ML-based Performance Prediction for DCS

The overview of MLPP-DCS is shown in Fig. 3. This figure illustrates the case where the performance parameter is computation time T_1, but the same methodology applies when the performance parameter is maximum memory usage T_2. In MLPP-DCS, DCS predictors are generated using the ML-based DCS predictor generation algorithms.

The ML-based DCS predictor generation algorithms A_{GBDT}, A_{RF}, A_{DT}, and A_{LR} use the ML algorithms GBDT, RF, DT, and LR, respectively. The training dataset used as input for these algorithms consists of pairs of explanatory variables $V_{E_1}, V_{E_2}, \cdots, V_{E_{10}}$, derived from the features of the environment models E and monitor models R input to DCS, and target variables (V_{T_1} for computation time or V_{T_2} for maximum memory usage) obtained from the actual execution of DCS. The explanatory variables are as follows, totaling ten:

- Regarding the environment models: number of models, maximum number of states, sum of number of transitions, and transitions by controllable/uncontrollable actions.
- Regarding the monitor models: number of models, sum of number of states, transitions, and transitions by controllable/uncontrollable actions.

The maximum number of states regarding the environment models E is computed as $\prod_{e \in E} S_e$, considering $\forall e \in E$. This refers to the maximum number of states in the monolithic environment model m generated by Parallel Composition.

We constructed these explanatory variables based on the following rationale. When synthesizing a controller c using DCS, intermediate results such as the monolithic environment model m, game space g, and abstracted game space g' are generated. In this paper, these are referred to as the computational space in DCS. It is known that the number of states in this computational space increases exponentially with the number of environment models E used as inputs to the system [8]. The increase in the number of states in the computational space leads to higher computational time and maximum memory usage for DCS. Therefore, based on this understanding, we constructed these explanatory variables that are hypothesized to significantly impact the number of states in the computational space.

The DCS predictor (P_{T_1} or P_{T_2}) takes pairs of the environment and monitor models $\langle E', R' \rangle$, which are the inputs

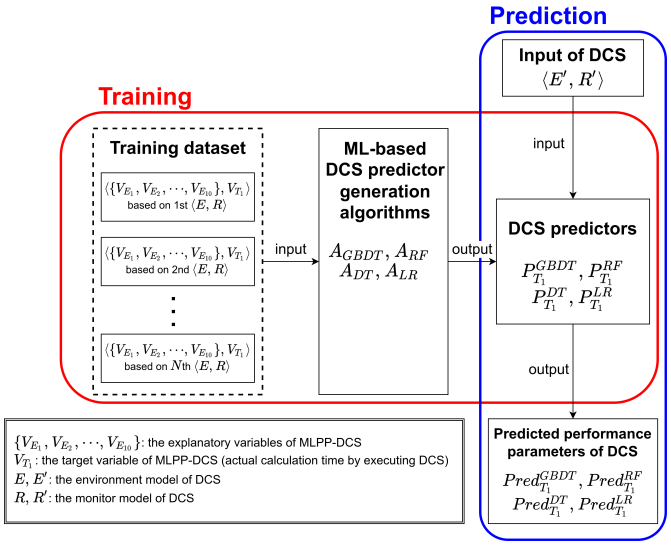

Fig. 3. Overview of MLPP-DCS (performance parameter: calculation time)

to DCS, and outputs the predicted performance parameters of DCS ($Pred_{T_1}$ or $Pred_{T_2}$). For each performance parameter, there are four predictors generated using four different algorithms: A_{GBDT}, A_{RF}, A_{DT}, and A_{LR}.

V. Evaluation

Based on the following research questions (RQs), we evaluated the DCS predictors generated in MLPP-DCS:

RQ1 How does the accuracy change with the different generation algorithms used?

RQ2 How does the accuracy change based on the performance parameter predicted?

A. The Experimental Settings

The platform used in this experiment is outlined as follows:

1) **[DCS platform]** CPU: Intel(R) Xeon(R) W-2265, RAM: 256GB, OS: Windows 10 Pro, 64-bit
2) **[ML platform]** CPU: Intel(R) Core(TM) i7-12700KF, RAM: 32GB, OS: Ubuntu 22.04.4 LTS
3) The Modal Transition System Analyser (MTSA) [9]: Tool for executing DCS
4) Scikit-learn [10]: Modules for creating ML models

More than 50 pairs of environment and monitor models for evaluation were created based on five scenarios used in related research [11]–[13]. Each scenario includes parameter N to vary the number of environment models and parameter K to vary the number of states within each environment model. Various environment models with different numbers of states and models were prepared by manipulating N and K. These were used as inputs to execute the DCS. During this process, pairs that took excessively long to execute or exceeded the memory capacity of the experimental environment were excluded. As a result, DCS outcomes were obtained for 49 pairs. These pairs were randomly divided into training and testing datasets in an 80:20 ratio. For each performance parameter, four DCS predictors are generated using four different ML-based generation algorithms, with the training dataset as input. However, these ML-based generation algorithms can yield different results depending on the random seed values. Therefore, seed values ranging from 0 to 2999 were considered. Finally, the DCS predictors were validated using the test dataset, with the coefficient of determination R^2 as the main metric.

TABLE I
Result of DCS predictors for computation time

Predictor	Algorithm	Seed	R^2_{max}	V_{R^2}	T_{pred}
$P_{T_1}^{GBDT}$	A_{GBDT}	810	0.814	0.0203	75.1
$P_{T_1}^{RF}$	A_{RF}	611	0.543	0.00704	147
$P_{T_1}^{DT}$	A_{DT}	508	0.814	0.129	7.86
$P_{T_1}^{LR}$	A_{LR}	0	−0.137	0	15.4

TABLE II
Result of DCS predictors for maximum memory usage

Predictor	Algorithm	Seed	R^2_{max}	V_{R^2}	T_{pred}
$P_{T_2}^{GBDT}$	A_{GBDT}	2990	0.523	0.0269	74.5
$P_{T_2}^{RF}$	A_{RF}	118	0.536	0.00564	146
$P_{T_2}^{DT}$	A_{DT}	1621	−0.219	0.0134	7.65
$P_{T_2}^{LR}$	A_{LR}	0	0.145	0	15.1

B. Experimental Results

The experimental results are presented in Tables I and II. In these tables, "Seed" denotes the seed value at the best accuracy, R^2_{max} denotes the coefficient of determination at best accuracy, V_{R^2} denotes the variance of coefficient of determination due to seed values, and T_{pred} denotes the total execution time (in seconds) of the ML-based DCS predictor generation algorithm across all seed values for the experimental subjects. The closer R^2_{max} is to 1, the better the accuracy, and the closer V_{R^2} is to 0, the smaller the variance in accuracy due to seed values.

From Table I, the predictor $P_{T_1}^{GBDT}$, which has the highest R^2_{max} and a smaller V_{R^2} compared to the predictor $P_{T_1}^{RF}$, demonstrates the best accuracy. From Table II, the predictor $P_{T_2}^{RF}$, which has the highest R^2_{max} and the second smallest V_{R^2}, demonstrates the best accuracy.

Furthermore, a comparison between the predicted values and the actual values at the best accuracy is presented in Fig. 4. In this figure, the x-axis represents the predicted values, and the y-axis represents the actual values. The closer the data points are distributed around the line $y = x$, the better the accuracy, indicating that the predicted values closely match the actual values.

C. Discussion

1) Impact of Different Generation Algorithms (RQ1): The experimental results show that for predicting computation time, the predictor $P_{T_1}^{GBDT}$ achieved the highest accuracy, while for predicting maximum memory usage, the predictor $P_{T_2}^{RF}$ achieved the best accuracy. As the objective of this

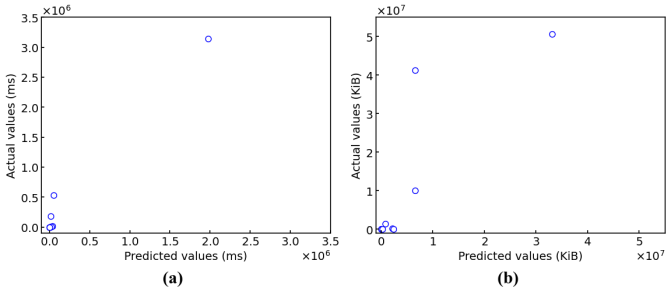

Fig. 4. The predicted and actual values at best accuracy: (a) calculation time predicted by $P_{T_1}^{GBDT}$, (b) maximum memory usage predicted by $P_{T_2}^{RF}$

study was to determine trends in the performance parameters from the explanatory variables rather than performing binary classification, algorithms based on decision trees were more suitable than LR. Furthermore, the performance parameters in this study have the characteristic of increasing exponentially. In such cases, ensemble learning methods such as RF and GBDT are likely to achieve higher accuracy.

2) Impact based on Performance Parameter Predicted (RQ2): The experimental results indicate that the accuracy of predicting computation time was higher than that of predicting maximum memory usage. One possible reason for this could be that the explanatory variables were not well-suited for predicting the performance parameter. It is believed that maximum memory usage is more directly related to the computational space than computation time. Since predicting maximum memory usage requires capturing more detailed trends of computational space, the accuracy for this prediction was lower. Strictly predicting computation space requires information about actions, specifically how actions relate between the environment and the monitor model. However, in this study, this information was not included in the explanatory variables, likely contributing to the decreased prediction accuracy.

3) Threats to Validity: For result parameters other than T_{pred} in this experiment, there are no external factors that threaten internal validity. While background processes during the experiment might impact T_{pred}, their influence on the scale of T_{pred} is minor and can be considered negligible. Although only 49 datasets were provided for training with the ML-based algorithm, which is generally considered insufficient, this issue was mitigated by using models from related research. This approach helped avoid the problem of arbitrary dataset utilization and addressed threats to external validity.

VI. Related Work

Similar to DCS, model checking (MC) is another field that faces the challenge of exponential growth in computation space. In a related study, W. Zhu et al. [14] addressed this issue by replacing the non-polynomial problems in MC with binary classification problems using ML.

DCS has been practically applied to various products and applications related to consumer electronics. Specifically, S. A. Zudaire et al. [15] synthesized flight plans for drones using DCS and demonstrated its application in forest fire suppression missions with physical drones. Similarly, A. Borda et al. [12] and J. Li et al. [16] explore the use of DCS in smart city applications, including cyber-physical security systems, smart building control, and flood monitoring systems.

VII. Conclusion and Future Work

This paper proposes MLPP-DCS, a method for predicting computation time and maximum memory usage of DCS based on features derived from the environment and the monitor models. As future work, we plan to stringify LTS to incorporate information about the types of actions into the DCS predictor.

References

[1] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[2] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distributed Computing*, vol. 2, no. 3, pp. 117–126, Sep. 1987.

[3] J. Li and K. Tei, "Done is better than perfect: Iterative adaptation via multi-grained requirement relaxation," in *2022 IEEE 30th International Requirements Engineering Conference (RE)*, 2022, pp. 288–294.

[4] M. H. L. Louk and B. A. Tama, "Dual-ids: A bagging-based gradient boosting decision tree model for network anomaly intrusion detection system," *Expert Systems with Applications*, vol. 213, p. 119030, 2023.

[5] M. McClelland and M. Campbell, "Probabilistic modeling of anticipation in human controllers," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 4, pp. 886–900, 2013.

[6] G. Yu, J. Yuan, and Z. Liu, "Unsupervised random forest indexing for fast action search," in *CVPR 2011*, 2011, pp. 865–872.

[7] C. Shen, Y. Chen, G. Yang, and X. Guan, "Toward hand-dominated activity recognition systems with wristband-interaction behavior analysis," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 7, pp. 2501–2511, 2020.

[8] K. Aizawa, K. Tei, and S. Honiden, "Analysis space reduction with state merging for ensuring safety properties of self-adaptive systems," in *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, 2019, pp. 1363–1370.

[9] N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel, "Mtsa: The modal transition system analyser," in *23rd IEEE/ACM International Conference on Automated Software Engineering*, 2008, pp. 475–476.

[10] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] T. Delgado, M. S. Sorondo, V. Braberman, and S. Uchitel, "Exploration policies for on-the-fly controller synthesis: A reinforcement learning approach," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 33, no. 1, pp. 569–577, Jul. 2023.

[12] A. Borda, L. Pasquale, V. Koutavas, and B. Nuseibeh, "Compositional verification of self-adaptive cyber-physical systems," in *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, 2018, pp. 1–11.

[13] T. Hirano, K. Tei, K. Aizawa, and S. Honiden, "Differential controller synthesis at runtime using changed parts of environment model," in *2021 IEEE 8th International Conference on Industrial Engineering and Applications (ICIEA)*, 2021, pp. 91–100.

[14] W. Zhu, H. Wu, and M. Deng, "Ltl model checking based on binary classification of machine learning," *IEEE Access*, vol. 7, pp. 135 703–135 719, 2019.

[15] S. A. Zudaire, L. Nahabedian, and S. Uchitel, "Assured mission adaptation of uavs," *ACM Trans. Auton. Adapt. Syst.*, vol. 16, no. 3–4, jul 2022.

[16] J. Li *et al.*, "Employing discrete controller synthesis for developing systems-of-systems controllers," in *Proceedings of the 12th ACM/IEEE International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems*, 2024, pp. 1–8.