Exploring Machine Learning for Faster Mapping and Scheduling of Automotive Applications on ADAS Platforms

Rafael Sterzinger, Wolfgang Koch, and Ralph Hoch
Institute of Computer Technology
TU Wien
Vienna, Austria
{firstname.lastname}@tuwien.ac.at

***Abstract*—In this work, we address the challenge of efficient task mapping and scheduling in Advanced Driver Assistance Systems (ADAS), which are becoming increasingly complex. For this, we explore Mixed-Integer Programming (MIP) combined with Machine Learning (ML) techniques as an alternative to previous work using heuristic algorithms such as simulated annealing or genetic algorithms. Our key contributions include: (1) A simplified MIP formulation of the problem with a novel load-balancing objective. (2) Employing Bayesian optimization to expedite the solving process by finding a better MIP solver configuration, reducing worst-case solving time by 65%. (3) ML-driven decision variable prediction via a Graph Convolutional Network, which is able to fix decision variables with an average precision of up to 77%, allowing for better branching decisions during the solving process. Experimental results demonstrate the potential for faster solving times, highlighting the value of further integrating machine learning with MIP for advanced ADAS scheduling.**

***Index Terms*—Scheduling, Automotive Applications, Machine Learning, Mixed-Integer Programming, Bayesian Optimization**

I. Introduction

Motivation. Advanced Driver Assistance Systems (ADASs) are a collection of electronic technologies designed to aid drivers in the driving process, enhancing safety and improving the overall driving experience. ADAS employs sensors, cameras, radar, and other data sources to monitor the vehicle's surroundings and provide critical information to the driver or even take automatic actions to prevent accidents. Currently, tasks within an ADAS platform are scheduled in advance and cannot be migrated during runtime. However, a migration at runtime is favorable as it would yield better resource utilization [1]. Moreover, the already complex ADAS systems are expected to grow even further in complexity, resulting in thousands of tasks with multiple task dependencies. Based on this trend, faster mapping and scheduling of tasks on ADAS platforms is required, motivating the exploration of Machine Learning (ML) techniques to expedite this process.

This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) in the "ICT of the Future" program and administered by the FFG as project #887474 with the short name JOptim. More information can be found at https://projekte.ffg.at/projekt/4138805.

Related Work. In the past, optimization algorithms such as simulated annealing or genetic algorithms (cf. [1], [2]) have been employed to heuristically solve the mapping and scheduling task in ADAS platforms. However, in this work, we want to explore exact Mixed-Integer Programming (MIP) as an option, motivated by the active research in the combinatorial optimization community that combines ML with MIP to reduce solving time. Here, the goal is to exploit statistical regularities within a problem, e.g., when they are solved repeatedly with only minor changes in their formulation (see [3], and therein mentioned references).

MIP is typically solved using a variant of a Branch-and-Bound (B&B) method, which involves recursively constructing a search tree, with the root node representing the original problem [4]. During the solving process, the tree is expanded by adding constraints (branching) to divide the feasible solution space or reduced (pruned) by employing bounds stemming from the current best-known solution. Additionally, so-called cuts can further tighten the feasible region by eliminating non-integral or suboptimal solutions [5].

In recent years, the combination of MIP and ML has already proven beneficial in various practical applications. For instance, MIP in combination with ML is used in the works by Sterzinger et al. [6] to expedite the scheduling of power generation units or, according to Paulus et al. [4], to speed up server load balancing or neural network verification.

Contribution. Our contributions are as follows:

- Proposing a simplified MIP formulation for the mapping and scheduling problem of automotive applications on ADAS platforms, which includes a new optimization objective for load balancing among cores.
- Employing Bayesian optimization to efficiently find optimal MIP-Solver configuration with only a handful of samples, which can aid in reducing worst-time complexity by around 65%.
- Utilizing ML classifiers to predict the probability of a decision variable belonging to the optimal solution, where decision variables in the context of ADAS can be fixed with an average precision of up to 77%.

II. Automotive Applications on ADAS Platforms

System & Application Model: ADAS is a complex platform, hosting different CPU and GPU hardware setups of varying clock speeds that run different Operating Systems (OS) depending on safety and performance requirements [1]. In order to provide a common execution environment for these systems, a middleware layer is running on top of them, guaranteeing the portability of software functions based on their execution and safety needs. Additionally, it enables tasks to be executed according to a table-driven pre-computed schedule, independent of the underlying OS dispatching mechanisms, ensuring temporal isolation. Exploring the option of ML in conjunction with MIP to find such schedules faster is the goal of this paper.

Software functions, potentially stemming from different vendors, must be integrated and deployed on top of this middleware. A software function comprises multiple tasks that are either already pre-assigned to cores, e.g., if specific hardware requirements exist, or assigned at a later point by the scheduling algorithm [7].

A task gives rise to a set of periodic (reoccurring) executions that can be preempted on macrotick granularity, i.e., stopping a task and continuing it at a later point in time. Furthermore, each task has the following requirements/attributes:

- Core (Optional)
- Periodicity
- Worst-Case Computation Time
- Deadline
- Earliest Release-Time/Activation
- Maximum Jitter

Timing Constraints: Besides the timing constraints stemming from a task's definition, such as its periodicity, worst-case execution time, defined deadline, and possible jitter constraints that restrict the variance of execution of consecutive instances [1], tasks may also have requirements related to temporal dependencies between them, defined in so-called task chains. A task chain specifies that instances of different tasks must execute in a specified order within a maximum end-to-end latency [7]. A task chain consists of a source, processing, and sink task, each with potentially different periodicity (cf. Fig. 1; four tasks with periodicity 10, 15, 15, and 5). In more detail, for each source task, there needs to be a correct sequence of the remaining tasks defined in the task chain. However, processing and sink tasks can merge inputs stemming from multiple preceding tasks.

III. Problem Formulation as Mixed-Integer-Program

In this section, we formulate the mapping and scheduling task of automotive applications on ADAS platforms as a MIP. Note that we omit the option of preemption as well as jitter constraints (we only verify the adherence to task dependency chains once) to simplify solving for this initial exploration. However, we hypothesize that accomplished speed-ups would translate to the complete problem formulation and will be tested extensively in future work.

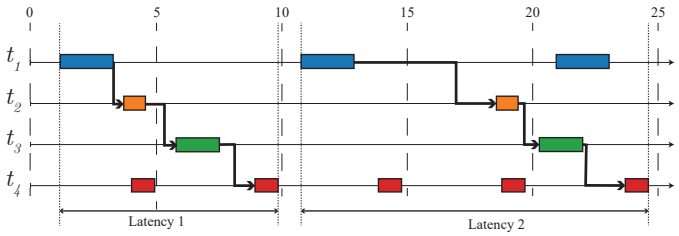

Fig. 1. An example of a task dependency chain – consisting of four tasks (t_1 to t_4) with periodicity, 10, 15, 15, 5 – illustrating its execution order and resulting different latencies.

Our proposed formulation is as follows:

A. Sets and Parameters

- J: Set of tasks
  - p_j: Period of task j
  - d_j: Deadline of task j
  - a_j: Earliest-Activation of task j
  - w_j: Worst-Case Computation Time of task j
- C: Set of cores
  - m_c: Macro-Tick of core c
- H: Set of chains;
  - b_h: Budget for chain h

B. Decision Variables

- $x_{j,c}$: 1 if task j is assigned to core c, 0 otherwise (binary)
- $k_{j,h}$: Periodicity of task j in chain h (integer)
- s_j: Start time of task j (integer)

C. Objective Function

We minimize the Mean Absolute Deviation of task assignments per core to encourage load balancing:

$$\text{Minimize } \frac{1}{|C|}\sum_{c\in C}\Big|\sum_{j\in J} x_{j,c} - \mu\Big|, \tag{1}$$

where μ is the mean amount of tasks per core, i.e., $\frac{|J|}{|C|}$.

D. Constraints

Only one core per task:

$$\sum_{c\in C} x_{j,c} = 1 \quad \forall j \in J \tag{2}$$

Task finishes before its deadline:

$$s_j + x_{j,c}\cdot\left\lceil\frac{w_j}{m_c}\right\rceil \le d_j \quad \forall j \in J, \forall c \in C \tag{3}$$

Start time is after the earliest activation:

$$s_j \ge a_j \quad \forall j \in J \tag{4}$$

No overlap of two tasks assigned to the same core:

$$s_i + \left\lceil\frac{w_i}{m_c}\right\rceil \le s_j \vee s_j + \left\lceil\frac{w_j}{m_c}\right\rceil \le s_i \tag{5}$$

Here, we employ the so-called Big-M method to formulate that if tasks $i, j \in J$ are scheduled on the same core $c \in C$, they *must not* overlap.

Tasks are in order of task dependency chain: For two consecutive tasks j_m and j_n in a task dependency chain $h \in H$ with $h \subseteq J$ and $j_m \prec_h j_n$, we create constraints of the form:

$$s_{j_m} + k_{j_m,h} \cdot p_{j_m} + \sum_{c \in C} x_{j_m,c} \cdot \left\lceil \frac{w_{j_m}}{m_c} \right\rceil \leq s_{j_n} + k_{j_n,h} \cdot p_{j_n} \quad (6)$$

Budget of task dependency chain is not exceeded: Let j_a be the source task and j_z be the sink task of h, the latency budget is constrained as follows:

$$\Big(k_{j_z,h} \cdot p_{j_z} + s_{j_z} + \sum_{c \in C} x_{j_z,c} \cdot \left\lceil \frac{w_{j_z}}{m_c} \right\rceil \Big) - \Big(k_{j_a,h} \cdot p_{j_a} + s_{j_a}\Big) \leq b_h \quad (7)$$

IV. Learning Solver Configuration

Similar to Valentine et al. [8], we first want to efficiently learn good settings to configure the MIP-Solver – SCIP [9] – considering only the three emphasis parameter presets for *presolving*, *heuristics*, and *separation* (each with parameters: default, off, fast, aggressive) as well as a general solver *emphasis* setting (default, counter, cpsolver, easycip, feasibility, hardlp, optimality, phasefeas, phaseimprove, phaseproof). Such presets guide the more specific solver settings at a higher level, which is necessary since optimizing individual parameters is infeasible due to the presence of approximately 2,500 binary, integer, and continuous parameters [10]. By using these high-level presets, the number of possible configurations, denoted as $\mathcal{C}$, is reduced to $|\mathcal{C}| = 4^3 \times 10$, making sufficient exploration more manageable. However, evaluating each possible $c \in \mathcal{C}$ is still prohibitively expensive

Hence, we opt for Bayesian optimization to systematically sample configurations from $\mathcal{C}$ and to improve our understanding of the relation between them incrementally and consequently solving time; given a training set of problem instances D_{train}, we want to find a good parameter configuration for the solver to ideally achieve optimal solving time on a test set D_{test}. At a high level, the procedure consists of sampling configurations, evaluating their performance, and updating the expectation over solving time. By doing so, we end up with an approximate optimal configuration for D_{train} while needing much fewer evaluations.

In more detail, we employ a Gaussian Process (GP) to learn a regressor of solving time, based on configurations, from noisy continuous data and to systematically model the uncertainty of unseen ones [11]. To systematically select the next sample, we require a cost-effective acquisition function. For this, we use the probability of improvement [12]:

$$PI(c) = P\Big(f(c) \geq f(c^+) + \kappa\Big),$$

where c^+ is the best configuration observed so far, f represents the MIP solver, and κ is a parameter that balances exploration and exploitation.

Our procedure is thus as follows: (1) we split the problem instances into training- and test-set, (2) we conduct an initial so-called burn-in phase of m steps, using a Latin-Hypercube sequence, (3) we perform n sampling steps based on PI and evaluate each sample for the entire set D_{train}, picking the maximum solving time to approximate *worst-case* solving time of D_{train}, (4) we pick the top ten predicted configurations and aggregate their parameter settings, where the majority of each emphasis setting dictates our *optimal* configuration.

V. Learning Primal Heuristics

Next, we want to learn so-called primal heuristics to not only speed up solving by selecting a configuration beforehand but also the solving process itself. Primal heuristics are crucial for finding good, feasible solutions early on, allowing for more aggressive pruning of the B&B tree during the solving process, which significantly enhances the performance [13].

Based on the works by Shen et al. [13], we explore the possibility of using machine learning to automatically learn effective primal heuristics in the context of ADAS platforms. For this, we train a machine learning model using a dataset of small-scale problem instances that have been solved to optimality to learn the label of the decision (binary) variables, i.e., to which core a task is assigned.

Here, the model at hand is a Graph Convolutional Network (GCN), which takes a linkage graph of a MIP problem instance as input. In this graph representation, each node represents a decision variable, and edges are added between nodes if their corresponding decision variables are linked by a constraint.

After training, when presented with a new problem instance, the trained GCN, using the mentioned graph representation, can effectively estimate the probability of each decision variable being active in the optimal solution. Based on these estimations, the solver can make more educated decisions about the branching direction during the solving process, which expedites finding good primal solutions, given that a strong solution is actually present in that part of the search tree.

VI. Experiments & Results

We evaluate the proposed methodologies on 150 problem instances of different complexity: ADAS 100% to ADAS 500%, with 30 instances each; a higher percentage means more complex, i.e., ADAS 500% has five times more resources and tasks to schedule than ADAS 100%. Our experiments were conducted on a shared distributed computing cluster equipped with an AMD EPYC 9654 96-core processor with a clock speed of 2.4 GHz. Hence, to reduce measurement variability of solving time due to fluctuating load, we run all evaluations on the same machine four times and report test set means.

A. Results: Learning Solver Configurations

In order to learn optimal solver configuration, we perform Bayesian optimization on a training set consisting of 10 out of 30 total instances of ADAS 100% to 300%. With respect to

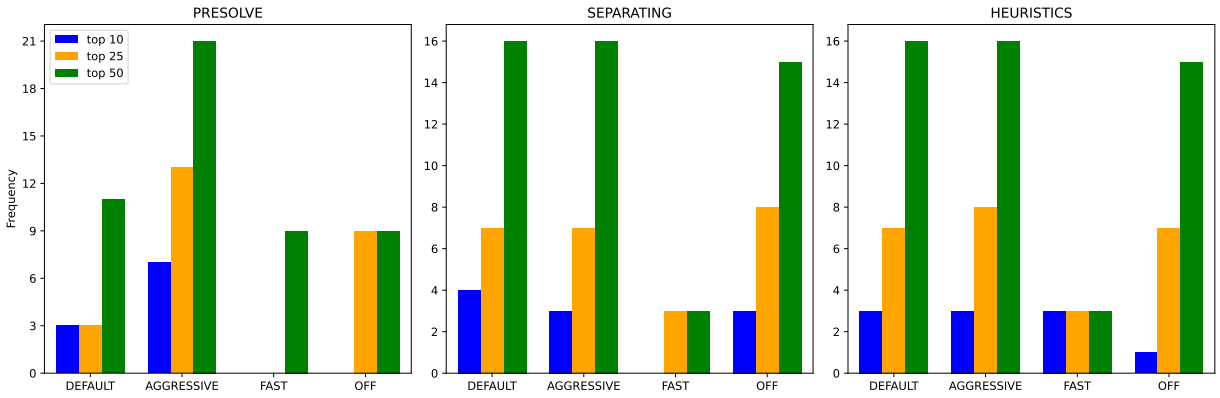

Fig. 2. Comparing parameter distributions of predicted top 10, 25, and 50 configurations (highest probability of improvement over c^+) for ADAS 200%.

the hyperparameters, we perform $m = 10$ steps for the burn-in phase and $n = 50$ steps for actual sampling. Given the computational resources required to evaluate a configuration, we enforce a smaller subset of task dependency chains during the fitting phase. We increase the amount when evaluating solving times to see if speed-ups translate to more complex instances (exact numbers are denoted in Table I).

First, we report predictive performance: Table I presents the estimated worst-case solving time predicted by the fitted GP, alongside the actual solving time of the MIP-Solver in its *default* configuration and our learned *optimal* one. Note that the dependency chains must be the same as during the fitting phase for evaluating predictive performance. A few key takeaways: estimating worst-case solving time for the default configuration is hard, with significant discrepancies observed; the learned configuration achieves better worst-case solving time and is easier to estimate.

TABLE I
COMPARISON OF ACTUAL AND PREDICTED WORST-CASE SOLVING TIMES USING THE FITTED GAUSSIAN PROCESS MODEL

Test Case	Default		Optimal		#Chains
	$\hat{y}_{\max}$	$y_{\max}$	$\hat{y}_{\max}$	$y_{\max}$	Fit/Eval.
ADAS 100%	115.55	201.53	61.30	74.88	5/5
ADAS 200%	18.66	3.93	6.46	3.76	6/20
ADAS 300%	67.66	12.30	9.68	12.16	7/30

Next, depicted in Fig. 2, we illustrate the distribution over parameters of predicted top 10, 25, and 50 configurations with the highest probability of improvement over c^+ for ADAS 200% instances. Notably, the impact of *presolve* is significant, with a clear preference for the *aggressive* setting. Additionally, regarding settings for *separating* and *heuristics*, solving time appears to be indifferent between the *default*, *aggressive*, and *off* settings, with only the *fast* option being unfavorable.

Finally, in Fig. 3, we report the measured average, median, 90th percentile, and worst-case solving time of each of the remaining 20 test instances evaluated with the predicted *optimal* solver configuration with a computing budget of 300, 600, and 1200 seconds respectively as well as a *higher* amount of task dependency chains enforced (cf. Table I). Although

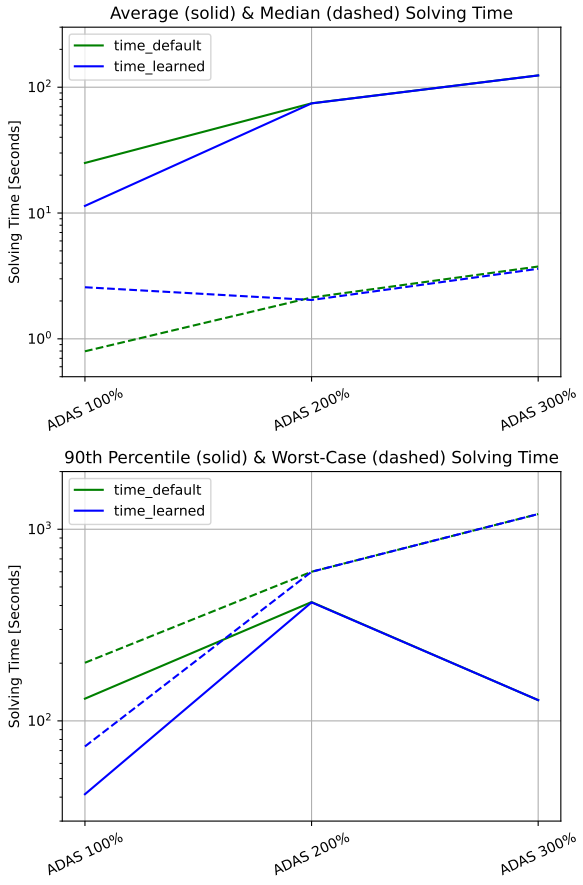

Fig. 3. Evaluating average, median, 90th percentile, and worst-case solving times for learned solver configurations: less complex instances benefit greatly, reducing the worst-case solving time by around 65%.

for instances of ADAS 200% and ADAS 300%, solving time has not been improved, instances from ADAS 100% benefited greatly, improving worst-case solving time by around 65%.

B. Results: Learning Primal Heuristics

For the next experiment, we consider three different ML models: the previously mentioned GCN, a Logistic Regression (LR) classifier, as well as XGBoost (XGB). For the GCN, we use 20 layers, while the hidden vector dimension is set to 32, and train it for 40 epochs. LR and XGB use default hyperparameters from Scikit-learn [12]. All models use a 57-dimensional feature vector for each decision variable, normalized to the range $[0, 1]$. Each model is trained using 60 optimally-solved instances in total, stemming from ADAS 100% and ADAS 200%.

We assess the classification performance of ML models using two *more difficult* test datasets: ADAS 300% and ADAS 400%, with 30 instances each and employing all task dependency chains. Performance is measured by Average Precision (AP) which accumulates the product of precision and changes in recall across ranked variables. AP is preferred over Accuracy (AC) in this context because it accounts for the ranking of decision variables, making it more suitable for imbalanced data, typical in NP-hard problem predictions [13].

TABLE II
PREDICTIVE PERFORMANCE OF DIFFERENT ML-CLASSIFIERS

Test Case	**ADAS 400%**		**ADAS 500%**	
	AP	AC	AP	AC
GCN	**76.39**	**73.47**	**76.70**	**73.52**
LR	63.51	71.86	64.81	72.12
XGB	61.60	57.30	60.94	57.25

Results of this experiment are denoted in Table II. Concerning the two evaluated datasets, there is a clear ranking in performance: GCN $>$ LR $>$ XGB. Considering the performance of GCN, which attains an AP of around 77%, ML is a vital option in accurately predicting decision variables to allow for improved branching within the solving process.

VII. CONCLUSION & FUTURE WORK

In this work, we addressed the challenge of efficient task mapping and scheduling of automotive applications on ADAS platforms, exploring MIP combined with ML techniques. We proposed a simplified MIP formulation and a new load balancing objective, as well as employed Bayesian optimization to configure MIP-Solvers to reduce worst-case solving time by up to 65%. Moreover, we used ML classifiers to accurately predict decision variables with an AP of 77%. In summary, our experimental results showed improvements in solving times and predictive accuracy, demonstrating the potential of integrating ML with MIP for advanced ADAS scheduling, motivating future work in this direction. Anticipated future work includes expanding the problem formulation to encompass the complete ADAS scheduling problem. Additionally, we plan to learn configurations on a per-instance basis following [10] and incorporating the predicted decision variables within the solving process to better guide branching within the B&B tree.

REFERENCES

[1] S. D. McLean, S. S. Craciunas, E. Alexander Juul Hansen, and P. Pop, "Mapping and Scheduling Automotive Applications on ADAS Platforms using Metaheuristics," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 329–336, IEEE.

[2] S. D. McLean, E. A. Juul Hansen, P. Pop, and S. S. Craciunas, "Configuring adas platforms for automotive applications using metaheuristics," *Frontiers in Robotics and AI*, vol. 8, Jan. 2022.

[3] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, "Exact Combinatorial Optimization with Graph Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 32*, 2019.

[4] M. Paulus and A. Krause, "Learning to dive in branch and bound," in *Advances in Neural Information Processing Systems* (A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds.), vol. 36, pp. 34260–34277, Curran Associates, Inc., 2023.

[5] M. B. Paulus, G. Zarpellon, A. Krause, L. Charlin, and C. J. Maddison, "Learning to Cut by Looking Ahead: Cutting Plane Selection via Imitation Learning," July 2022.

[6] R. Sterzinger, J. Poland, M. B. Paulus, and D. Chételat, "Learning to predict security constraints for large-scale unit commitment problems," in *2023 IEEE PES Innovative Smart Grid Technologies Europe (ISGT EUROPE)*, pp. 1–5, 2023.

[7] S. D. McLean, E. A. Juul Hansen, P. Pop, and S. S. Craciunas, "Configuring ADAS Platforms for Automotive Applications Using Metaheuristics," vol. 8, p. 762227.

[8] R. Valentin, C. Ferrari, J. Scheurer, A. Amrollahi, C. Wendler, and M. B. Paulus, "Instance-wise algorithm configuration with graph neural networks," Feb. 2022.

[9] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig, "The SCIP Optimization Suite 8.0," Dec. 2021.

[10] R. Valentin, C. Ferrari, J. Scheurer, A. Amrollahi, C. Wendler, and M. B. Paulus, "Instance-wise algorithm configuration with graph neural networks," Feb. 2022. arXiv:2202.04910 [cs, math].

[11] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning, Cambridge, Mass: MIT Press, 2006.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] Y. Shen, Y. Sun, A. Eberhard, and X. Li, "Learning Primal Heuristics for Mixed Integer Programs," in *IJCNN 2021 - International Joint Conference on Neural Networks, Proceedings*, IEEE, Institute of Electrical and Electronics Engineers, July 2021.