SMTgazer: Learning to Schedule SMT Algorithms via Bayesian Optimization

Chuan Luo*, Shaoke Cui*, Jianping Song*, Xindi Zhang†, Wei Wu‡, Chanjuan Liu§, Shaowei Cai†, Chunming Hu*
*School of Software, Beihang University, Beijing, China
†Key Laboratory of System Software (Chinese Academy of Sciences) and State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Science, Beijing, China
‡School of Computer Science and Engineering, Central South University, Changsha, China
§School of Computer Science and Technology, Dalian University of Technology, Dalian, China
{chuanluo, cuisk, jpsong, hucm}@buaa.edu.cn, {zhangxd, caisw}@ios.ac.cn,
william.third.wu@gmail.com, chanjuanliu@dlut.edu.cn

***Abstract*—Satisfiability Modulo Theories (SMT) plays a critical role in various software engineering applications, including program verification, symbolic execution, and automated test generation. Over the years, a wide range of SMT solvers has been developed, typically designed for general purposes or tailored to specific background theories, such as bit-vectors or nonlinear arithmetic. Due to the diversity and complexity of SMT instances, no single solver consistently outperforms others across all problem domains. This motivates the need for algorithm selection strategies that can adaptively choose solvers based on the characteristics of the instances.**

To overcome the limitations of single-solver selection, solving SMT as a scheduling problem, enabling a more fault-tolerant and effective use of multiple solvers in sequence. We model algorithm scheduling as a hyperparameter optimization problem, enabling efficient black-box search over solver sequences while treating the dataset as a whole, thus achieving globally optimized and robust scheduling strategies. The resulting scheduler called *SMTgazer*. To further enhance scheduling efficiency and solver performance, we propose two optimizations: leveraging unsupervised X-means clustering to create semantically coherent instance groups for localized model training, and augmenting the Bayesian optimization surrogate with boosting and bagging ensembles to improve generalization and mitigate overfitting, thereby yielding more reliable performance predictions for the sequential portfolio scheduler.

Extensive experiments are conducted to evaluate the performance of *SMTgazer*, utilizing six SMT benchmarks derived from real-world applications. It shows that our approach consistently outperforms current state-of-the-art methods. Particularly, *SMTgazer* achieves a 44.65% reduction in PAR-2 score and 69.11% decrease in the number of unsolved instances, compared to the strongest competitor, *Sibyl*, demonstrating the effectiveness of formulating SMT algorithm scheduling as a hyperparameter optimization problem. We further analyze the generated scheduling sequences to uncover the design principles that explain the success of our method. Finally, we also empirically show that our approach is both robust and generalizable, and the proposed strategies are effective.

***Index Terms*—SMT, solver scheduling, Bayesian optimization**

I. Introduction

Satisfiability Modulo Theories (SMT) is a fundamental decision problem that extends Boolean satisfiability (SAT) by incorporating first-order background theories such as bit-vectors, linear and nonlinear arithmetic, arrays, uninterpreted functions, or their combinations. The ability to reason about complex logical formulas makes SMT a central enabling technology in many areas of software engineering. Notably, SMT solvers [1] are widely used in program verification [2], symbolic execution [3], model checking [4], automated test generation [5], and software synthesis [6]. Their capacity to encode and solve rich semantic constraints has significantly contributed to advancements in automated reasoning and high-assurance software development.

As the scale and complexity of SMT problems continue to increase in industrial applications, there is a growing demand for high-performance SMT solving techniques. In response, researchers have developed efficient SMT solvers [1], [7]–[10] over the past two decades and have proposed various effective algorithms tailored to different background theories. For instance, modern SMT solvers primarily rely on eager bit-blasting techniques to address the bit-vector (BV) background theory [7], while they typically employ the lazy CDCL(T) [1], [11] framework or Model-Constructing Satisfiability Calculus (MCSAT) [12] to manage complex integer arithmetic (IA) theories. Recently, novel paradigms such as local search [13], [14] and learning [15], [16] have emerged in the domain of SMT solving.

There is no universally superior SMT algorithm, and algorithms often exhibit complementary strengths. To fully leverage the capabilities of various SMT solvers, certain application software integrates multiple solvers, allowing users to make informed selections [17]–[19]. However, choosing the appropriate SMT algorithm is inherently challenging. It requires a thorough understanding of the structural and semantic features of the given SMT formula, as well as the ability to model the performance landscape of each solver across different types of problems.

Algorithm selection is introduced to address this issue. It is a methodology designed to leverage the advantages of a diverse set of algorithms to effectively solve a given problem. Numerous studies have been conducted on algorithm selection

* Xindi Zhang is the corresponding author.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00109

related to SMT solving.

Typically, (single-algorithm) selectors identify the empirically best solver by employing feature engineering. This process involves extracting feature vectors from given SMT queries and training models to predict the performance or ranking of the solvers. MachSMT [20] is a recently advanced SMT selector that employs traditional feature engineering methods. The features are defined and tuned by SMT experts utilizing domain knowledge. Sibyl [21] represents the current state-of-the-art selector, which relies on graph attention networks (GATs). It constructs graph-based embeddings of queries, and the feature vectors are generated using representation learning techniques to predict solver rankings.

However, selectors cannot consistently choose the best or even a relatively superior solver [21]. If an ineffective solver is selected, it may exceed the time limit for the SMT query, indicating a lack of robustness.

Parallel portfolios offer a straightforward solution. The performance depends on the effectiveness of the best solver within the selected subset. While this approach is often effective, the cost of computational resources increases with the number of selected solvers, making it less feasible in resource-constrained environments.

Algorithm scheduling is a flexible and cost-effective alternative, often referred to as the sequential portfolio. In order to maximize the likelihood of quickly solving a given instance, a scheduler involves not only selecting and ranking of complementary solvers or configurations, but also the allocation of resources across multiple solvers.

A representative scheduler is SATZilla [22], and its underlying methods remain competitive today [23] for SAT solving. Recently, there have been studies focusing on SMT algorithm scheduling [24], [25], among which MedleySolver [25] serves as a representative. It employs multi-armed bandit techniques for online algorithm selection. Given a specific set of SMT solvers and queries, MedleySolver learns algorithm preference policies during the solving process and allocates time resources across a sequence of solvers using a lightweight timeout prediction scheme. This method improves the overall runtime; however, due to the absence of pre-training knowledge, its performance cannot compete with that of the current leading algorithm selector, Sibyl.

Contributions The primary contribution of this paper is the development of a high-performance SMT algorithm scheduling tool, referred to as *SMTgazer*. The second and third contributions are two optimization strategies aimed at enhancing scheduling efficiency: Fine-grained cluster-specific scheduling (*clustering*) and a hybrid surrogate model for Bayesian optimization (*HyModel*).

SMTgazer: This scheduler formulates the algorithm scheduling problem as a hyperparameter optimization task. It considers both the ranking of the scheduled solvers and the allocated time as hyperparameters, while the time required to solve a given instance is treated as the target cost function. By leveraging the power of Bayesian optimization [26], [27] with the assistance of *SMAC3* [28], *SMTgazer* efficiently explores the scheduling space and selects the solver scheduling solution with minimal costs. *SMTgazer* enables more precise scheduling and allows for iterative refinement based on performance feedback during training.

Clustering: Hyperparameter optimization typically operates on a dataset and yields a single solution. However, the SMT benchmark set often comprises instances generated from various applications. For this issue, we apply the *X-means clustering* to partition the large benchmark into several semantically coherent smaller groups. Each cluster is then associated with a separately scheduling solution.

Hybrid-Model: Bayesian optimization is widely used for hyperparameter tuning, often employing bagging-based (e.g. random forests) surrogate model. However, in regression tasks, bagging and boosting provide complementary advantages—bagging enhances generalization, while boosting improves predictive accuracy. To leverage both, we integrate a LightGBM-based boosting model with SMAC's default bagging surrogate, forming a hybrid model. This integration improves surrogate accuracy and enhances overall performance on small datasets with poor clustering.

Evaluation: This paper conducts extensive experiments to evaluate our scheduler. We selected six SMT benchmarks, each derived from real-world application scenarios, and further collected the corresponding candidate SMT solvers for each set. *SMTgazer* outperforms existing competitive methods, achieving a 44.65% lower PAR-2 score [29] and 69.11% fewer unsolved instances than *Sibyl*, the most recent competitor, thereby validating the effectiveness of formulating SMT scheduling as a hyperparameter optimization problem. In addition to overall performance, we provide an in-depth analysis of the learned schedules from multiple perspectives, including solving behavior, cross-validation consistency, and solver dominance, to better understand the nature of the scheduling strategy. An ablation study further demonstrates both the effectiveness and the complementarity of the proposed approach.

II. Preliminaries

This section provides the necessary preliminaries for this paper.

A. Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) extends propositional satisfiability (SAT) by allowing formulas over background theories such as arithmetic, arrays, or bit-vectors. Formally, the SMT problem asks whether a first-order logic formula is satisfiable under specific theories.

The SMT-LIB[1] initiative provides a standardized format and benchmark suite for SMT solvers, serving as the basis for the annual SMT Competition (SMT-COMP)[2]. SMT-COMP has become the standard platform for evaluating SMT solver performance across a range of background theories.

[1]https://smt-lib.org/

[2]https://smt-comp.github.io/

The benchmarks are categorized based on the underlying theories. There are three representative families that encompass the largest number of instances, and QF is an abbreviation for 'quantifier-free'.

- *Bit-Vectors (QF-BV)*: Models fixed-width binary representations and supports bitwise arithmetic operations.
- *Nonlinear Integer Arithmetic (QF-NIA)*: Extends linear arithmetic constraints $\sum_i a_i x_i < k$ by allowing multiplicative terms such as xy or x^n.
- *Equality with Uninterpreted Functions and Linear Arithmetic (ELA)*: Combines equality reasoning over uninterpreted functions with linear integer or rational arithmetic.

Example. Consider the following SMT formula that combines multiple theories with a disjunctive constraint:

$$f(x) = y \wedge (x+y < 5 \vee x \cdot y = 6) \wedge \texttt{bvadd}(a, \texttt{0b0001}) = \texttt{0b0010}$$

This formula involves uninterpreted functions, linear and nonlinear integer arithmetic, and bit-vector operations. This formula is *satisfiable* with one possible model:

$$x = 2, \quad y = 3, \quad f(2) = 3, \quad a = \texttt{0b0001}$$

B. Algorithm Selection and Scheduling

An SMT *query*, or SMT formula, represents a set of first-order logic constraints defined within specific background theories, which are to be resolved by SMT solvers.

Let $\mathcal{Q} = \{q_1, q_2 ..., q_m\}$ be a set of SMT queries, and $\mathcal{S} = \{s_1, s_2 ..., s_n\}$ be the candidate SMT solvers.[3]

Algorithm selection learns a mapping $f_{\text{select}} : \mathcal{Q} \to \mathcal{S}$ that assigns each query $q \in \mathcal{Q}$ to the solver $s_i \in \mathcal{S}$ expected to solve it q most efficiently.

Algorithm scheduling instead learns a mapping $f_{\text{scheduling}} : \mathcal{Q} \to \Sigma$. For each query $q \in \mathcal{Q}$, $\sigma(q) \in \Sigma$ is a *scheduling sequence*, where

$$\sigma(q) = [(s_{i_1}, t_1), (s_{i_2}, t_2), \ldots, (s_{i_k}, t_k)],$$

and a time budget t_j is specified for each solver. Solvers are executed sequentially to minimize overall runtime and mitigate the effects of suboptimal selection decisions.

The solvers in $\sigma(q)$ are executed sequentially according to the specified order. The execution halts when either a solver returns a solution, *satisfiable* or *unsatisfiable*, or when the cumulative time exceeds a global timeout T_{max}.

If q is solvable under the scheduling sequence $\sigma(q)$, and it is first solved by the i-th solver s_i, we denote this index as $\ell(q) = i$. Let $t_f \leq t_{s_i}$ be the actual time taken by s_i to solve q. The total runtime for solving q following the scheduling sequence $\sigma(q)$, denoted as $T_\sigma(q)$, is defined as:

$$T_\sigma(q) = \sum_{j=1}^{\ell(q)-1} t_{s_j} + t_f$$

[3]Noting that SMT covers diverse background theories, and solvers differ in their support and optimization for each. As a result, candidate solvers are selected based on the specific theory or benchmark, yielding different solver pools across query sets.

If none of the solvers in $\sigma(q)$ successfully solve q within the global cutoff T_{max}, then $T(q) = k \times T_{\text{max}}$, where k is the penalty factor (PAR-k score).

The *objective* of algorithm scheduling is to find the optimal $f_{\text{scheduling}}$ that minimizes the total runtime.

$$\min_{\sigma \in \Sigma} \sum_{q \in \mathcal{Q}} T_\sigma(q)$$

C. Hyperparameter Optimization Problem

Hyperparameter Optimization (HPO) aims to find a configuration of hyperparameters that minimizes a black-box objective function. Given a configuration space Λ and an objective function $f : \Lambda \to \mathbb{R}$, the problem is defined as:

$$\lambda^* = \arg\min_{\lambda \in \Lambda} f(\lambda)$$

Here, λ is a candidate configuration, and $f(\lambda)$ evaluates its performance. Due to the expensive and non-differentiable nature of f, methods like *Bayesian Optimization* (BO) are commonly employed. The following section will illustrate BO methods in the context of HPO.

D. Bayesian Optimization

Bayesian Optimization is an effective strategy for optimizing expensive black-box functions, frequently employed in hyperparameter tuning and algorithm configuration. [30]

At each iteration t, BO fits a *surrogate model* using existing *observations* and selects the next promising configuration by minimizing an *acquisition function*.

The formal definitions are as follows.

Let $\Lambda \subset \mathbb{R}^d$ be a bounded configuration space, where each $\lambda = (\lambda_1, ..., \lambda_m) \in \Lambda$ denotes a *configuration* (e.g., a set of algorithm parameters), and $f : \Lambda \to \mathbb{R}$ be a black-box objective function over Λ. BO aims to identify the optimal configuration that incurs the lowest cost.

A piece of the *observation* is a recorded input–output pair (λ, y), where $y = f(\lambda) + \epsilon$, obtained by evaluating the objective function at configuration λ. ϵ represents the observation *noise*, typically i.i.d. Gaussian.

The observations collected prior to the time frame t are denoted as $\mathcal{D}_t = \{(\lambda_i, y_i)\}_{i=1}^{t-1}$.

A *surrogate model* is a probabilistic approximation $p(f|\mathcal{D}_t)$ of the objective function f, constructed using the observed data $\mathcal{D}_t$, typically with a Gaussian Process.

For each iteration t, an *acquisition function* $\alpha : \Lambda \to \mathbb{R}$ guides the next query:

$$\lambda_t = \arg\max_{\lambda \in \Lambda} \alpha(\lambda; p(f \mid \mathcal{D}_t)).$$

After evaluating $f(\lambda_t)$, the observed dataset are updated $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{\lambda_t, f(\lambda_t)\}$, and the surrogate model is refined accordingly. This process repeats until a time budget t_{max} is reached or convergence is achieved.

SMAC3 [28] is a prominent hyperparameter optimization algorithm that combines Bayesian optimization with decision-tree-based surrogate models. It provides a robust and flexible framework for efficiently finding high-performing configurations across a wide range of machine learning tasks.

E. Ensemble Learning

Ensemble Learning is a fundamental machine learning paradigm that combines multiple base learners (or models) to produce a more accurate and robust learner than any single learning alone. Common ensemble methods can be divided into two categories:

Bagging: The core idea of bagging (bootstrap aggregating) is to train each base learner independently on different bootstrap samples of the training data, which are generated by random sampling with replacement. The outputs are then combined, typically through majority voting or averaging, to produce a final prediction. The primary advantage of bagging lies in its ability to enhance stability and reduce overfitting in high-variance models. A representative method of bagging is the Random Forest [31].

Boosting: The idea is to sequentially training a series of models, where each new model is trained to focus on the instances that previous models misclassified, often by adjusting the sample weights. This approach effectively reduces bias and builds a strong learner from many weak learners, with each model concentrating more on the hard-to-predict examples. XGBoost [32] and LightGBM [33] are two representative boosting algorithms.

In the domain of HPO, the default surrogate model of SMAC3 is a random forest [28], which leverages the strengths of bagging to provide robust predictions and uncertainty estimates for efficient Bayesian optimization.

III. Our Proposed *SMTgazer* Approach

In this section, we present the core methodology behind *SMTgazer*, our effective SMT algorithm scheduling framework. The central idea of *SMTgazer* is to reformulate the scheduling problem as a HPO task over a given set of SMT queries. Additionally, we elaborate on two more contributions, *Clustering* and *Hybrid-Model*, that significantly enhance the performance of *SMTgazer*.

A. Top-level Framework of SMTgazer

The top-level framework of *SMTgazer* is illustrated in Fig. 1. As a learning approach for SMT algorithm scheduling, *SMTgazer* comprises both the training component (Fig. 1a) and the testing component (Fig. 1b).

1) Modeling

Given a set of SMT queries $\mathcal{Q}$ and a set of candidate n SMT solvers $\mathcal{S}$. The scheduling sequence for query $q \in \mathcal{Q}$ is $\sigma(q) = [(s_i, t_i)]_{i=1}^{n}$.

There are two types of configurations for scheduling: solver-related parameters and time-related parameters.

Solver-related Parameters. Integers $j \in \{1, ..., n\}$ uniformly represent an SMT solver in $\mathcal{S}$; specifically, the categorical hyperparameter $s_i = j$ denotes the i-th solver in $\sigma(q)$ and is assigned to j.

Time-related Parameters. The continuous hyperparameters $t_i \in [0, T_{max}]$ represents the allocated time for the i-th solver in $\sigma(q)$.

With n candidate SMT solvers, $\sigma(q)$ can be formulated to a configuration $\lambda = (s_1, t_1, s_2, t_2, ..., s_n, t_n)$. There are two additional constraints that the configuration must adhere to.

Time-allocation Constraint. All solvers should be allocated a specific amount of runtime, where $s_i = 0$ indicates that the i-th solver should not be used in this portfolio.

$$\sum_{i=0}^{n} t_i = T_{max}$$

All-different Constraint. This constraint enforces that all variables within a specified set must assign distinct values.

$$\text{all-different } (s_1, s_2, ..., s_n)$$

The goal (target function) is to find the optimal configuration λ^*.

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \sum_{q \in Q} T_{\sigma[\lambda]}(q),$$

where $T_{\sigma[\lambda]}(q)$ denotes the runtime for $q \in \mathcal{Q}$ following $\sigma[\lambda]$, and $\sigma[\lambda]$ indicates the scheduling sequence corresponding to configuration λ.

2) Training

The training process is described as follows:

In the modeling stage, each SMT query is transformed into a 189-dimensional feature vector utilizing human-engineered features.

Noting that *SMTgazer* employ the same feature extraction approach as MachSMT [20][4], where the majority of the features are designed to capture the syntactic characteristics of the input formulas, such as the occurrence frequencies of Boolean constructs, integer terms, and real-valued expressions.[5]

Then, *SMTgazer* clusters the SMT queries into several groups based on the feature vectors, which will be discussed in Sec. III-B.

In the PO stage, *SMTgazer* treats each group of queries as an individual HPO process. Subsequently, for each group, it employs Bayesian optimization techniques to obtain a promising configuration λ^*. *SMTgazer* uses the BO implementation of SMAC3 [28], and is improved with a hybrid surrogate model, which will be discussed in Sec. III-C.

Finally, we get the scheduling sequences $\sigma[\lambda^*]$ for each group of SMT queries.

3) Testing

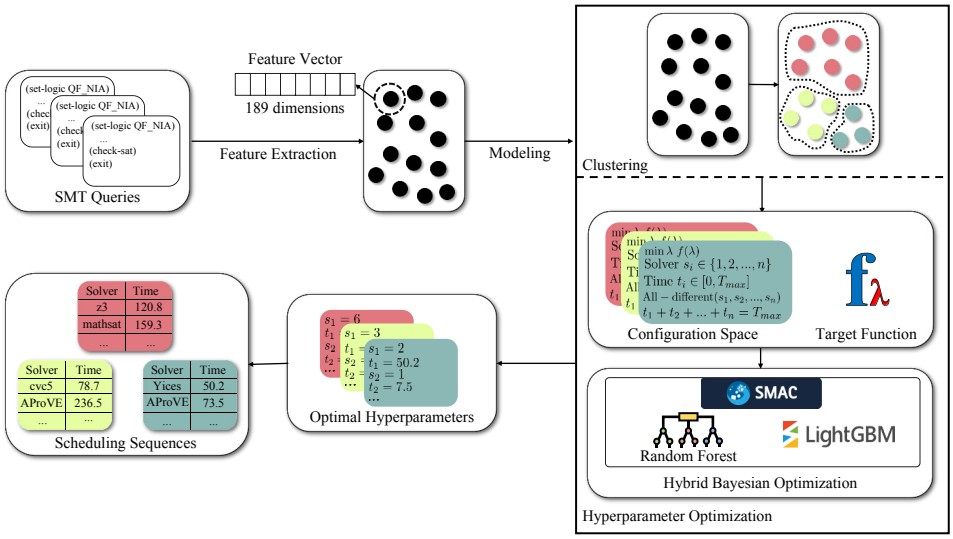
Given a new SMT query q, *SMTgazer* first extracts its feature vector, classifies q into the cluster with the nearest centroid, and then applies the scheduling sequence corresponding to this cluster.

B. Fine-grained Cluster-specific Scheduling

One limitation of HPO is its inability to fine-tune decisions at the instance level, as it operates on global performance across the dataset. Existing studies have shown that no single solver consistently achieves optimal performance across all

[4]https://github.com/MachSMT/MachSMT

[5]For a complete description of each feature, readers can refer to the literature [20] for more details.



(a) *SMTgazer* training component.

(b) *SMTgazer* testing component.

Fig. 1: Top-level framework of our *SMTgazer* method.

TABLE I: A motivating example of further clustering.

Name	Solver					
	Bitwuzla	*STP*	*Yices*	*cvc5*	*mathsat*	*z3*
bench_5632	3.82	3.40	2.37	TO	217.20	TO
bench_8458	174.66	511.75	303.90	TO	TO	TO

SMT logics. [20], [21] Even within the same logic, it remains challenging for a solver to perform optimally on all queries.

Motivating Example. In the QF_BV benchmark, there are two instances from the Sage2 subset, `bench_5632.smt2` and `bench_8458.smt2`, belonging to the same application domain. However, according to the average results of 10 runs shown in Table I, *Yices* is optimal for `bench_5632.smt2`, while *Bitwuzla* performs best on `bench_8458.smt2`. With the help of the clustering method, *SMTgazer* assigns these queries to different clusters, cluster 15 for `bench_5632.smt2` and cluster 0 for `bench_8458.smt2`. This demonstrates that clustering is an effective method for distinguishing their differing solver preferences.

To address this issue, *SMTgazer* employs a clustering-based approach. Since the exact number of distinct SMT query types in the dataset is unknown and difficult to determine in advance, we adopt an unsupervised learning strategy to automatically discover latent structure among queries.

In practice, *SMTgazer* employs the *X-Means* algorithm [34] for clustering. This process automatically determines the number of subcategories and stores the corresponding centroids, enabling efficient assignment of new SMT queries to the appropriate clusters.

C. Hybrid Surrogate Models for Bayesian Optimization

During hyperparameter optimization, *SMTgazer* constructs a unified configuration space for each cluster and applies Bayesian optimization to identify the optimal settings.

For each BO iteration t, we assume that the configuration is λ_t and the corresponding scheduling sequence for this iteration is $\sigma[\lambda_t]$. The total runtime across the training set is evaluated by calculating $f(\lambda_t) = \sum_q T_{\sigma[\lambda_t]}(q)$, where *SMTgazer* employs the PAR-2 score. Subsequently, an observation $(\lambda_t, f(\lambda_t) + \epsilon_t)$ is incorporated into the observation records $\mathcal{D}_t$, where the noise ϵ_t is determined by the standard deviation across 10 runs with random seed.

To generate a legal configuration λ_t for the subsequent iteration, the acquisition function must adhere to both the time-allocation constraint and the all-different constraint. Since SMAC3 does not support sum-equality constraints, *SMTgazer* employs the proportional allocation method to simulate the time-allocation constraint.

The surrogate model of SMAC3 is a random forest, which is a typical bagging-based methods. Bagging and boosting are known to offer complementary advantages: while bagging reduces variance by aggregating predictions from multiple

independent learners, boosting reduces bias through sequential training that emphasizes hard-to-predict instances. In the context of algorithm scheduling, capturing fine-grained performance patterns is crucial. Boosting methods like LightGBM [33] excel at modeling such patterns due to their bias-reduction capabilities.

To leverage the complementary strengths of both approaches, we construct a hybrid ensemble by combining a random forest (bagging) $p_1(f|\mathcal{D}_t)$ and a LightGBM model (boosting) $p_2(f|\mathcal{D}_t)$ in a 1:1 ratio by

$$p(f|\mathcal{D}_t) = \frac{p_1(f|\mathcal{D}_t) + p_2(f|\mathcal{D}_t)}{2}.$$

This design enables us to balance variance reduction and bias mitigation, resulting in more accurate performance predictions and, ultimately, more effective scheduling decisions.

Optimization Strategy. In practice, *SMTgazer* follows an incremental scheduling process. Starting from an empty list $L = []$, *SMTgazer* iteratively appends a new solver s_i from the remaining solvers to maximize the overall performance, forming $L = L \cup [s_i, t_i]$ along with its allocated time t_i. For each iteration, BO is applied to jointly optimize the time parameters and the newly added solver, while keeping the order of the previously added solvers. This process continues until L contains a predefined number of solvers n.

IV. Experimental Design

This section introduces the experimental settings employed to evaluate the performance of *SMTgazer* against the current state-of-the-art SMT algorithm selectors and schedulers. The experiments are designed to address the following three research questions (RQs).

A. Research Questions

RQ1 *:* How does SMTgazer perform in comparison to the current state-of-the-art algorithm selectors or schedulers?

Answer: SMTgazer demonstrates significant performance improvements compared to its competitors, and the results confirm both effectiveness and stability.

RQ2*:* What are the characteristics of the solver schedules generated by SMTgazer?

Answer: SMTgazer generates adaptive, front-loaded scheduling sequences that not only solve the majority of instances early but also exhibit strong scheduling policy-level effectiveness and generalize well across datasets with varying structures.

RQ3*:* Are the proposed *clustering* and *hybrid-model* strategies in this work effective?

Answer: The proposed two strategies are efficient and collaborate effectively when used together.

B. Benchmarks

Based on the existing studies [20], [21], we selected datasets from four application domains of SMT solvers: competition (SMT-COMP), bounded model checking (BMC), symbolic execution (SymEx), and syntax-guided synthesis (SyGuS). For the competition domain, we selected the three SMT logics with the highest query counts; for the remaining three application domains, we adopted the same datasets used by *Sibyl* [21]. The detailed description of the datasets is as follows.

SMT-COMP. SMT-COMP was established to spark advances in SMT, especially for verification. Empirical results across successive SMT-COMPs show that the performance of solvers improves substantially over that of previous competitions. The SMT-COMP 2021 single query track consists of 18 logic divisions [35]. From the published research [21], learning methods require substantial data for training and evaluation, so we selected the three logic divisions with the highest number of queries, *i.e.,* QF_BV, QF_NIA, and ELA.

BMC. *ESBMC* [36] is a C bounded model checker that leverages an SMT solver backend. It generates SMT queries to establish that a specification holds up to a given bound and employs K-Induction to extend verification beyond that bound. These queries encode both the program's semantics and the induction conditions required to validate the inductive step. BMC dataset was generated by executing *ESBMC* on 11,369 C files from the SV-COMP'22 dataset [37]. Consistent with SV-COMP, for each C file, *ESBMC* was run with a 15-minute timeout, yielding nearly 700,000 SMT queries. From these, 100,000 queries were randomly sampled to form the dataset. All SMT queries in BMC dataset are in either the QF_BV or QF_Equality+BV logic division.

SymEx. *KLEE* [38] is a symbolic execution engine that relies on an SMT solver backend to determine the reachability of program execution states. When encountering interesting states, *KLEE* generates an SMT query encoding the path condition leading to that state to verify its reachability. SymEx dataset was generated by running *KLEE* version 2.2 on 20 GNU coreutils and 5 real world programs which *KLEE* is often evaluated on [17], [38], [39]. All SMT queries in SymEx dataset are in either the QF_BV or QF_Equality+BV logic division.

SyGuS. Syntax-guided synthesis is a program synthesis technique that generates candidate programs from a user-defined formal grammar and employs a constraint solver, typically an SMT solver, to validate each candidate's correctness [40]. SyGuS-COMP is the competition that evaluates program synthesis tools. SyGuS dataset was generated by a program synthesis tool competed in SyGuS-COMP'19 called *DryadSynth* [41]. Executed on the benchmarks from the Invariant Synthesis and Conditional Linear Integer Arithmetic categories from SyGuS-COMP'19, *DryadSynth* generated 2,000,000 SMT queries, in which 100,000 queries were randomly selected to construct the dataset. All SMT queries in SyGuS dataset are in the ELA logic division.

C. Candidate SMT solvers

Most solvers are designed for specific logics; therefore, we select the corresponding set of solvers based on the logics present in the dataset, in accordance with prior studies [21]. For each dataset, Table II presents the total number of SMT

TABLE II: Statistics of all datasets.

Dataset	#Total	#Testing	Candidate Solvers
QF_BV	8,748	1,737	[42]–[47]
QF_NIA	9,112	1,767	[42]–[45], [48], [49]
ELA	12,549	2,495	[42], [43], [45], [50]–[53]
BMC	100,000	65,003	[42]–[47]
SymEx	93,996	73,391	[42]–[47]
SyGuS	100,000	79,981	[42], [43], [45], [50], [53]

queries (#Total), the number of instances selected for testing (#Testing)[6], and the references for the candidate SMT solvers.

D. Competitors

This paper compares *SMTgazer* with three state-of-the-art SMT algorithm selectors and schedulers, *i.e., MachSMT* [20], *MedleySolver* [25], and *Sibyl* [21].[7]

- *MachSMT*. It leverages machine learning to construct empirical hardness models and pairwise ranking comparators of solvers for algorithm selection.
- *MedleySolver*. It models the solver scheduling problem as a multi-armed bandit problem and selects solvers through feature engineering and reinforcement learning.
- *Sibyl*. It is a selector for SMT solvers based on graph attention networks, a variant of graph neural networks. For a given SMT query, *Sibyl* generates scores for solvers for selection.

To facilitate a better understanding of the performance and behavior of SMT solvers, we also compare *SMTgazer* with two static selectors and the (VBS).

- *Frequently Fastest Static Selector (FFSS).* It selects the solver that is most frequently identified as the optimal choice for individual cases across the dataset.
- *Overall Fastest Static Selector (OFSS).* It selects the solver that takes the least total time on the entire training set.
- *Virtual Best Solver (VBS).* It is a hypothetical algorithm that is not implementable in practice. It is theoretically able to select the best solver from a given set of solvers for each SMT query, thus providing valuable bounds for evaluating performance.

E. Environmental Setups and Implementation Details

In this paper, all experiments are conducted on a cluster equipped with two AMD EPYC 7763 CPUs @ 2.45 GHz, totaling 128 physical cores, and 1 TB of RAM, running the Ubuntu 20.04 LTS (64-bit).

As with its competitors, *SMTgazer* is written in Python and primarily utilizes third-party software from SMAC3 [28] and LightGBM [33].

[6]The method for splitting the training and test sets is consistent with that of *Sibyl* [21]: 80:20 for SMT-COMP and 20:80 for the other dataset. Instances that none of the candidate solvers could solve were removed, since they provide no useful signal for the scheduling problem.

[7]Noting that there was an SMT scheduler before MedleySolver [24]. However, since its source codes and executable file is not publicly available (as confirmed via email with the authors of Sibyl [21]), we did not compare against Hůla et.al in this paper.

Training and evaluation require runtime labels for each SMT query. The runtime is labeled using the PAR-2, with a cutoff set at 1200 seconds. We employ the labels from *Sibyl*'s repository[8], which were produced by execution on the Starexec cluster [54].

In this paper, all selectors and schedulers are retrained using 10 distinct random seeds to ensure statistical reliability.

V. Evaluation

This section addresses the three research questions through comprehensive and extensive experiments.

A. Performance Evaluation

We compare *SMTgazer* with two state-of-the-art algorithm selectors, *MachSMT* and *sibyl*, one algorithm scheduler, *MedleySolver*, two static selectors, FFSS and OFSS, and the VBS selector. All the selectors and schedulers run ten times with different seeds. Since the unsolvable instances are filtered like sibyl, #Unsolved of VBS is always zero.

The results are presented in Table III. For each dataset, we report the average PAR-2 time (PAR-2) and the number of unsolved instances (#UNK) for each method in the corresponding table. Additionally, we specify the SMT solver selected by FFSS and OFSS for each dataset, and the best results in each category are underlined in the table for emphasis.

- *SMTgazer* solves the most instances across all six benchmarks; and achieves the best PAR-2 scores on five of them, with only one exception where the difference is just 0.11 seconds. On average, compared to *MachSMT*, *MedlySolver*, and *Sibyl*, *SMTgazer* reduces the number of unsolved instances by 81.27%, 94.71%, and 69.11%, and the PAR-2 scores by 48.70%, 91.22%, and 44.65%, respectively.
- *SMTgazer* consistently outperforms FFSS and OFSS, both of which select two fixed solvers for all instances. This highlights the strength of our method. Designing a good selecting method is not easy, as the other three competing methods do not always achieve better performance than FFSS and OFSS on all benchmarks.

To provide a more detailed comparison of *SMTgazer* against other strategies at the instance level, we include a scatter plot visualization. Figure 2 uses different colors and markers to compare *SMTgazer* with its three competitors, where each point represents a pair of average runtimes for a single instance. Points above the diagonal indicate instances where *SMTgazer* performs better (i.e., has a lower runtime), while points below indicate the opposite. Due to space constraints, we include all instances from the six benchmarks in this single figure for illustration. A more detailed version with per-benchmark breakdowns is available in our code repository.

This reveals that *SMTgazer* is able to solve many instances that its competitors fail to handle, and its primary advantage lies in effectively tackling the more challenging cases.

[8]https://github.com/will-leeson/sibyl

TABLE III: Comparing *SMTgazer* to its competitors on all datasets.

(a) QF_BV

Method	PAR2	#UNK
SMTgazer	33.12	12.8
MachSMT	50.64	28.1
MedleySolver	329.60	210.0
Sibyl	49.66	28.4
FFSS (*Yices* [44])	41.16	18.0
OFSS (*Yices* [44])	41.16	18.0
VBS	9.63	-

(b) QF_NIA

Method	PAR2	#UNK
SMTgazer	119.50	52.8
MachSMT	369.91	260.7
MedleySolver	329.47	215.1
Sibyl	257.33	164.8
FFSS (*Yices* [44])	440.39	312.0
OFSS (*z3* [42])	274.49	159.0
VBS	18.57	-

(c) ELA

Method	PAR2	#UNK
SMTgazer	4.91	3.0
MachSMT	26.87	25.4
MedleySolver	1361.48	1408.0
Sibyl	1539.63	1598.0
FFSS (*cvc5* [43])	28.03	26.0
OFSS (*cvc5* [43])	28.03	26.0
VBS	0.70	-

(d) BMC

Method	PAR2	#UNK
SMTgazer	1.67	9.4
MachSMT	2.79	39.7
MedleySolver	55.74	1123.0
Sibyl	1.68	16.2
FFSS (*Bitwuzla* [46])	2.78	24.0
OFSS (*Yices* [44])	2.18	18.0
VBS	0.58	-

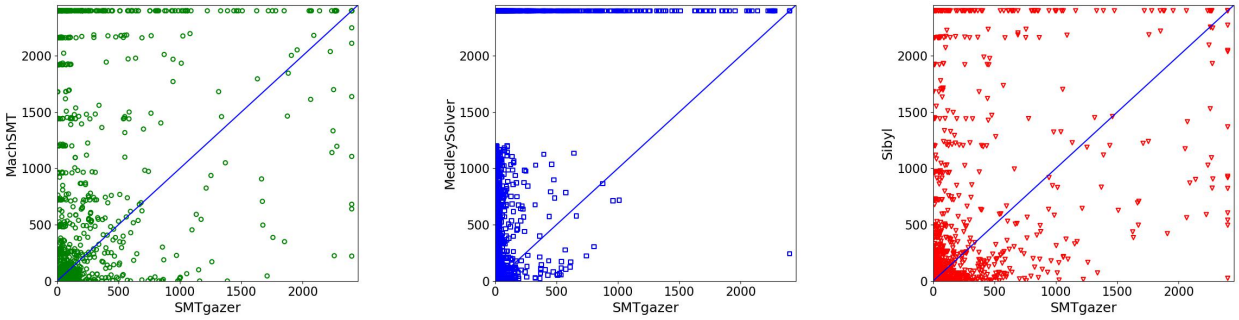
(e) SymEx

Method	PAR2	#UNK
SMTgazer	0.32	0.1
MachSMT	0.33	0.9
MedleySolver	18.34	275.0
Sibyl	0.27	0.2
FFSS (*STP* [47])	10.10	188.0
OFSS (*Yices* [44])	0.40	0.0
VBS	0.23	-

(f) SyGuS

Method	PAR2	#UNK
SMTgazer	0.028	0.0
MachSMT	0.080	2.0
MedleySolver	2.94	97.2
Sibyl	2.13	70.1
FFSS (*UE*[1][45])	2.93	97.0
OFSS (*cvc5* [43])	1.60	52.0
VBS	0.019	-

[1] To save space, "UE" denotes *UltimateEliminator+MathSAT*.



(a) *SMTgazer* vs *MachSMT* (b) *SMTgazer* vs *MedleySolver* (c) *SMTgazer* vs *Sibyl*

Fig. 2: Scatter plots of *SMTgazer* against its three competitors on all the six benchmarks. Noting that the average runtime may exceed 1200 seconds due to PAR-2 penalty and averaging across 10 runs.

We employ two methods to ensure the results are robust and generalizable of *SMTgazer*. Firstly, we conduct the experiment 10 times using different seeds and all the metrics use in this paper are the average numbers, which have already been discussed in the previous text.

Secondly, we adopt 5-fold cross-validation to evaluate the robustness of the training model. Specifically, each dataset is randomly partitioned into five equal-sized folds. In each round, four folds are used for training and the remaining one for testing. This process is repeated five times, with each fold used exactly once as the test set. The average standard deviation of PAR-2 across the five folds is 0.33. On average, the results differ by only 16.69% from those reported in Table III, indicating the reliability of our evaluation.

This evaluation protocol reduces the variance caused by random train-test splits and yields a more robust estimate of the model's performance. The results confirm the consistency and stability of *SMTgazer* across different subsets of the data.

RQ1: *SMTgazer* outperforms state-of-the-art SMT algorithm selectors and schedulers, and it is experimentally proven to be robust, demonstrating the effectiveness of modeling algorithm scheduling as a hyperparameter optimization problem.

B. Scheduling Behavior Analysis

Since *SMTgazer* is a scheduling algorithm, it raises several questions about the scheduling behavior (RQ2), such as:

- What kinds of schedules does it produce?
- How does it compare to other methods at the policy level?
- Does it generalize across datasets?

We design a series of experiments that focus on these three aspects.

Firstly, to analyze the behavior of the scheduling sequence, we note that different random seeds lead to varying solver orders. As a result, we cannot precisely determine the final scheduling sequence used for each dataset. Therefore, we conducted an additional experiment to record the position in

TABLE IV: The average number of SMT queries that the k-th solver successfully solved on all datasets.

k	Dataset					
	QF_BV	QF_NIA	ELA	BMC	SymEx	SyGuS
1	1634.6	1399.5	2429.1	63967.3	73336.1	68273.0
2	23.4	217.1	48.9	850.5	54.8	11659.0
3	55.0	57.2	7.3	151.4	0.0	49.0
4	11.2	40.4	6.7	24.4	0.0	0.0

the schedule, i.e., the index of the solver, that ultimately solved each instance.

The results are summarized in Table IV, where we report, for each instance set, the number of instances solved by the k-th solver in the schedule. Most of the benchmarks are solved by the first solver, achieving an average percentage of 92.39%. Meanwhile, the average percentages of instances solved by the 2nd, 3rd, and 4th solvers are 5.25%,1.17%, and 0.54%, respectively.

On the other hand, the average time allocated to the first 4 solvers are 219.86, 198.12, 355.17, and 426.85 seconds, respectively.

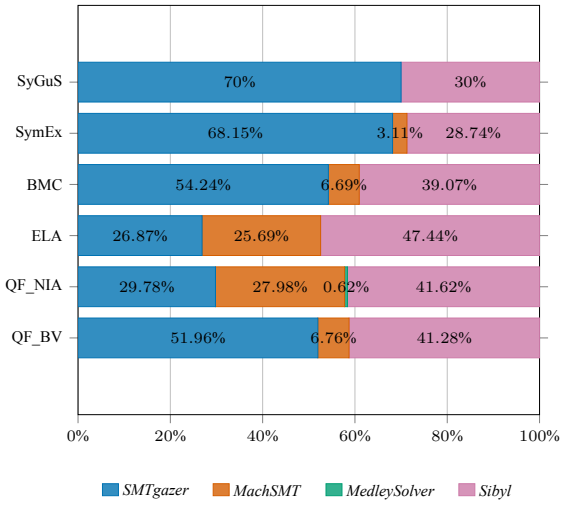


Fig. 3: The optimal performance ratio of *SMTgazer* and its competitors on all datasets.

Secondly, we evaluate the scheduling policy quality, Figure 3 presents a horizontal stacked bar chart illustrating the percentage of instances where each method achieves the best performance across six benchmark datasets.

From the result, *SMTgazer* outperforms a high win rate. For example, it achieves 70% on SyGuS and 68.15% on SymEx, significantly outperforming all baselines. Even on more diverse datasets like BMC and QF_BV, *SMTgazer* maintains a clear lead.

In contrast, Sibyl performs relatively well on structurally diverse datasets such as ELA and QF_NIA. *SMTgazer*, however, solves more instances on these datasets through its adaptive scheduling policy, highlighting a complementary relationship between the two methods.

TABLE V: Generalization evaluation via cross-dataset testing

Training	Testing	PAR2	#UNK
QF_BV	BMC	2.88	13.7
QF_BV	SymEx	3.92	0.0
BMC	QF_BV	43.27	17.8
BMC	SymEx	4.06	0.4
SymEx	BMC	8.03	5.0
SymEx	QF_BV	73.12	16.4

Finally, we evaluate the generalization ability of the sequences generated by *SMTgazer* through cross-dataset validation. Table V presents all possible combinations where the training and testing datasets differ, excluding cases where the candidate solvers are not the same. For each setting, the table reports the training and testing datasets, the resulting PAR-2 scores, the number of unsolved instances.

The results show that, when trained and tested on different datasets, the number of unsolved instances changes by 4.3, -0.1, 5.0, 0.3, -4.4, and 2.6, respectively, demonstrating the robustness and generalizability of our approach.

RQ2: *SMTgazer* generates effective, front-loaded schedules with strong generalization across datasets.

C. Ablation Studies

We perform ablation studies on *SMTgazer* to address the final research question.

This paper proposed two strategies, *Clustering* and *Hybrid-Model*. To evaluate the contribution of individual components in *SMTgazer*, we compare the full *SMTgazer* system with two ablated variants.

- *Alt-1*: removes the clustering step and schedules solvers on the whole dataset directly.
- *Alt-2*: uses only the default model in SMAC3, random forest, by removing the boosting model.

The results are summarized in Table VI. We present the PAR-2 score along with the average number of unsolved instances (#UNK) for each benchmark.

As shown in the table, each of the two strategies led to performance improvements on 5 out of the 6 datasets, with the sets of benefited datasets being partially distinct, demonstrating the effectiveness of combining both strategies.

To further illustrate the impact of each strategy, we present a bar chart of the relative PAR-2 score differences between the full *SMTgazer* system and its ablated variants (Figure 4). Each bar shows the percentage increase or decrease in PAR-2 score for a variant compared to *SMTgazer*, across the six benchmarks.

We observe that the clustering strategy demonstrates solid performance across most benchmarks, with an average improvement of 11.45% over the baseline without clustering.

TABLE VI: Comparing *SMTgazer* and its two variants.

Method	QF_BV		QF_NIA		ELA		BMC		SymEx		SyGuS	
	PAR2	#UNK	PAR2	#UNK	PAR2	#UNK	PAR2	#UNK	PAR2	#UNK	PAR2	#UNK
SMTgazer	33.12	12.8	119.50	52.8	4.91	3.0	1.67	9.4	0.32	0.1	0.028	0.0
Alt-1	29.51	9.7	140.78	66.3	5.13	3.0	2.07	12.4	0.40	0.0	0.036	0.0
Alt-2	33.45	13.0	121.14	56.7	7.05	3.0	1.70	9.6	0.31	0.1	0.029	0.0

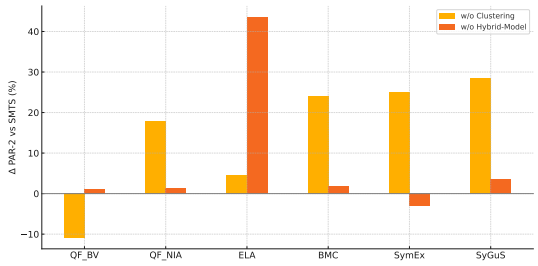


Fig. 4: Percentage Change in PAR-2 Compared to *SMTgazer*.

However, its effectiveness is limited on QF_BV and ELA, where the performance gain is relatively modest or even slightly negative.

This behavior can be attributed to the characteristics of these two benchmarks: both have limited training data and high intra-domain diversity. For instance, QF_BV was partitioned into 20 clusters, which is largely due to data sparsity leading to unstable clustering. The small dataset size further hinders the ability to train accurate predictors for each cluster.

ELA was grouped into only two clusters, indicating that the manually designed features fail to effectively separate the underlying task types. As previously discussed, the boosting-based model offers more precise predictions and demonstrates complementary behavior to the random forest approach. Experimental results confirm this synergy, showing that the combination of both models leads to improved overall performance.

Noting that, although QF_NIA contains diverse instances from SMT-COMP, most of the instances are generated from software termination verification problems. This results in more homogeneous structures, which enhances the effectiveness of clustering.

> **RQ3:** Both *Clustering* and *HybridModel* are effective and applicable to different scenarios, with complementary strengths.

D. Threats to Validity

There is a potential threat that bugs or implementation errors in our system may affect the correctness of the scheduling algorithm and solver results. Although we conducted thorough testing, we have also performed multi-person code reviews and made the code open-source to increase transparency and reduce the risk of unnoticed errors. However, such issues cannot be completely ruled out and might still impact the reliability of our findings.

For *SMTgazer* and its competitors, overfitting is a general threat. To address the threat, as claimed in Section V, we adopt 5-fold cross-validation to evaluate the robustness of *SMTgazer*. The average standard deviation of PAR-2 across the five folds is 0.33, indicating the reliability of our evaluation. Moreover, we have evaluated the generalization ability of *SMTgazer* through cross-dataset validation. The results in Table V demonstrate the robustness and generalizability of *SMTgazer*.

Another threat lies in the limited number and variety of solvers used in our experiments, which may not fully represent all existing SMT solvers. To address this, we have collected as many solver samples as possible and conducted cross-validation on multiple datasets to demonstrate the generalizability and scalability of our approach. Nevertheless, the benchmark datasets may still not fully capture the complexity of all real-world scenarios.

VI. Related Work

This section reviews related work on algorithm selection and scheduling, organized accordingly to reflect these two aspects, while highlighting how our approach differs from existing methods throughout the discussion.

Algorithm Selection. *MachSMT* [20] extracts the features of SMT queries and predicts the solving time of SMT queries given a solver through a machine learning model. *Sibyl* [21] employs representation learning techniques to generate feature vectors of SMT queries to predict solver rankings. Different from their work, our *SMTgazer* method constructs scheduling sequences containing several solvers rather than selecting a single solver. Moreover, we extract features of SMT queries for clustering instead of using them to predict the performance of solvers on SMT queries.

Algorithm Scheduling. *YicesLS* [55] first runs *Yices* for 100 seconds, then runs stochastic local search for 500 seconds. If the given instance is still not solved, *YicesLS* will run *Yices* for the remaining time. *FastSMT* [56] combines learning and synthesis techniques to find the right solution strategy for the solver. The difference between our approach and theirs is that we focus on scheduling solvers rather than the solving strategies inside solvers. *SATZilla* [23] uses machine learning to predict the time it will take for solvers on the given queries, thereby obtaining a subset of solvers that are expected to perform best. Unlike *SATZilla*, our approach focuses on SMT rather than SAT, which covers a wider range of logic divisions. And *SATZilla* does not set a suitable stop time

for each solver. It will only replace the next solver when the solver crashes. *MedleySolver* [25] employs multi-armed bandit techniques to predict the solver sequence and the sequence of time-allocations. Different from *MedleySolver*, we use Bayesian optimization to build the scheduling sequences instead of dynamically adjusting them based on SMT queries. Hůla *et al.*'s work [24] uses GNN to predict the runtimes of individual solvers and creates scheduling sequences based on the predictions. Our approach differs from their work in that they select the order of solvers based on the predictions without further exploring the time allocated to solvers.

VII. Conclusion and Future Works

This paper introduces *SMTgazer*, an algorithm scheduling based method for SMT solving. By formulating scheduling problem as a hyperparameter optimization problem, *SMTgazer* learns robust solver sequences that outperform existing selectors and schedulers. To improve generality and performance, *SMTgazer* integrates cluster-aware scheduling and a hybrid surrogate model. Extensive experiments demonstrate substantial gains over prior work, including a 44.65% reduction in PAR-2 and a 69.11% decrease in timeout instances compared to currently best competitor. Our findings highlight the potential of optimization-driven scheduling in SMT and offer insights into effective solver combinations.

In the future, we plan to extend it to small and diverse datasets, exploring finer-grained scheduling, and enabling data-driven control over the internal algorithms of SMT solvers. Moreover, we intend to integrate *SMTgazer* into client tools such as model checkers and symbolic executors to further enhance their performance.

VIII. Data availability

The implementation of *SMTgazer*, all adopted subjects, and detailed experimental results are publicly available at our repository: **https://github.com/Bazoka13/SMTgazer**.

IX. Acknowledgement

This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFB3307503, in part by the National Natural Science Foundation of China under Grants 62172072, 62202025, 62302528, 62422203, and 62522201, in part by Beijing Natural Science Foundation under Grant L241050, in part by the Young Elite Scientist Sponsorship Program by CAST under Grant YESS20230566, in part by CCF-Huawei Populus Grove Fund under Grant CCF-HuaweiFM2024005, and in part by the Fundamental Research Fund Project of Beihang University.

References

[1] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, Springer, 2008.

[2] N. S. Bjørner, K. L. McMillan, and A. Rybalchenko, "Program verification as satisfiability modulo theories.," *SMT@ IJCAR*, vol. 20, pp. 3–11, 2012.

[3] C. Cadar, D. Dunbar, D. R. Engler, *et al.*, "Klee: unassisted and automatic generation of high-coverage tests for complex systems programs.," in *OSDI*, vol. 8, pp. 209–224, 2008.

[4] D. Kroening and M. Tautschnig, "Cbmc–c bounded model checker: (competition contribution)," in *Tools and Algorithms for the Construction and Analysis of Systems: 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings 20*, pp. 389–391, Springer, 2014.

[5] R. Peña, J. Sánchez-Hernández, M. Garrido, and J. Sagredo, "Smt-based test-case generation and validation for programs with complex specifications," in *Analysis, Verification and Transformation for Declarative Programming and Intelligent Systems: Essays Dedicated to Manuel Hermenegildo on the Occasion of His 60th Birthday*, pp. 188–205, Springer, 2023.

[6] D. Ahmed, A. Peruffo, and A. Abate, "Automated and sound synthesis of lyapunov functions with smt solvers," in *Tools and Algorithms for the Construction and Analysis of Systems: 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings, Part I 26*, pp. 97–114, Springer, 2020.

[7] A. Niemetz and M. Preiner, "Bitwuzla," in *International Conference on Computer Aided Verification*, pp. 3–17, Springer, 2023.

[8] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, *et al.*, "cvc5: A versatile and industrial-strength smt solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 415–442, Springer, 2022.

[9] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The mathsat5 smt solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 93–107, Springer, 2013.

[10] B. Dutertre, "Yices 2.2," in *International Conference on Computer Aided Verification*, pp. 737–744, Springer, 2014.

[11] X. Zhang, B. Li, and S. Cai, "Deep combination of cdcl (t) and local search for satisfiability modulo non-linear integer arithmetic theory," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13, 2024.

[12] L. De Moura and D. Jovanović, "A model-constructing satisfiability calculus," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pp. 1–12, Springer, 2013.

[13] H. Li, B. Xia, and T. Zhao, "Local search for solving satisfiability of polynomial formulas," in *International Conference on Computer Aided Verification*, pp. 87–109, Springer, 2023.

[14] S. Cai, B. Li, and X. Zhang, "Local search for smt on linear integer arithmetic," in *International Conference on Computer Aided Verification*, pp. 227–248, Springer, 2022.

[15] Z. Lu, "Alphasmt: A reinforcement learning guided smt solver," Master's thesis, University of Waterloo, 2023.

[16] F. Jia, Y. Dong, M. Liu, P. Huang, F. Ma, and J. Zhang, "Suggesting variable order for cylindrical algebraic decomposition via reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 76098–76119, 2023.

[17] H. Palikareva and C. Cadar, "Multi-solver support in symbolic execution," in *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings* (N. Sharygina and H. Veith, eds.), vol. 8044 of *Lecture Notes in Computer Science*, pp. 53–68, Springer, 2013.

[18] H. Rocha, R. Menezes, L. C. Cordeiro, and R. S. Barreto, "Map2check: Using symbolic execution and fuzzing - (competition contribution)," in *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part II* (A. Biere and D. Parker, eds.), vol. 12079 of *Lecture Notes in Computer Science*, pp. 403–407, Springer, 2020.

[19] M. Y. R. Gadelha, F. R. Monteiro, J. Morse, L. C. Cordeiro, B. Fischer, and D. A. Nicole, "ESBMC 5.0: an industrial-strength C model checker," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018* (M. Huchard, C. Kästner, and G. Fraser, eds.), pp. 888–891, ACM, 2018.

[20] J. Scott, A. Niemetz, M. Preiner, S. Nejati, and V. Ganesh, "Machsmt: A machine learning-based algorithm selector for SMT solvers," in *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021,*

Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II (J. F. Groote and K. G. Larsen, eds.), vol. 12652 of *Lecture Notes in Computer Science*, pp. 303–325, Springer, 2021.

[21] W. Leeson, M. B. Dwyer, and A. Filieri, "Sibyl: Improving software engineering tools with SMT selection," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pp. 2185–2197, IEEE, 2023.

[22] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: Portfolio-based algorithm selection for SAT," *J. Artif. Intell. Res.*, vol. 32, pp. 565–606, 2008.

[23] H. Shavit and H. H. Hoos, "Revisiting satzilla features in 2024," in *27th International Conference on Theory and Applications of Satisfiability Testing (SAT 2024)*, pp. 27–1, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[24] J. Hula, D. Mojzísek, and M. Janota, "Graph neural networks for scheduling of SMT solvers," in *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021*, pp. 447–451, IEEE, 2021.

[25] N. Pimpalkhare, F. Mora, E. Polgreen, and S. A. Seshia, "Medleysolver: Online SMT algorithm selection," in *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings* (C. Li and F. Manyà, eds.), vol. 12831 of *Lecture Notes in Computer Science*, pp. 453–470, Springer, 2021.

[26] J. Mockus, "The application of bayesian methods for seeking the extremum," *Towards global optimization*, vol. 2, p. 117, 1998.

[27] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[28] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter, "Smac3: A versatile bayesian optimization package for hyperparameter optimization," *Journal of Machine Learning Research*, vol. 23, no. 54, pp. 1–9, 2022.

[29] S. Cai, C. Luo, X. Zhang, and J. Zhang, "Improving local search for structured SAT formulas via unit propagation based construct and cut initialization (short paper)," in *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021* (L. D. Michel, ed.), vol. 210 of *LIPIcs*, pp. 5:1–5:10, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[30] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning - Methods, Systems, Challenges* (F. Hutter, L. Kotthoff, and J. Vanschoren, eds.), The Springer Series on Challenges in Machine Learning, pp. 3–33, Springer, 2019.

[31] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[32] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016* (B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, eds.), pp. 785–794, ACM, 2016.

[33] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 3146–3154, 2017.

[34] D. Pelleg, A. Moore, *et al.*, "X-means: Extending k-means with e cient estimation of the number of clusters," in *ICML'00*, pp. 727–734, Citeseer, 2000.

[35] "Smt-comp 2021." https://smt-comp.github.io/2021/.

[36] M. Y. R. Gadelha, H. I. Ismail, and L. C. Cordeiro, "Handling loops in bounded model checking of C programs via k-induction," *Int. J. Softw. Tools Technol. Transf.*, vol. 19, no. 1, pp. 97–114, 2017.

[37] D. Beyer, "Progress on software verification: SV-COMP 2022," in *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part II* (D. Fisman and G. Rosu, eds.), vol. 13244 of *Lecture Notes in Computer Science*, pp. 375–402, Springer, 2022.

[38] C. Cadar, D. Dunbar, and D. R. Engler, "KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs," in *8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008, December 8-10, 2008, San Diego, California, USA, Proceedings* (R. Draves and R. van Renesse, eds.), pp. 209–224, USENIX Association, 2008.

[39] J. He, G. Sivanrupan, P. Tsankov, and M. T. Vechev, "Learning to explore paths for symbolic execution," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021* (Y. Kim, J. Kim, G. Vigna, and E. Shi, eds.), pp. 2526–2540, ACM, 2021.

[40] R. Alur, R. Bodík, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa, "Syntax-guided synthesis," in *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pp. 1–8, IEEE, 2013.

[41] K. Huang, X. Qiu, P. Shen, and Y. Wang, "Reconciling enumerative and deductive program synthesis," in *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020* (A. F. Donaldson and E. Torlak, eds.), pp. 1159–1174, ACM, 2020.

[42] L. M. de Moura and N. S. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings* (C. R. Ramakrishnan and J. Rehof, eds.), vol. 4963 of *Lecture Notes in Computer Science*, pp. 337–340, Springer, 2008.

[43] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar, "cvc5: A versatile and industrial-strength SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I* (D. Fisman and G. Rosu, eds.), vol. 13243 of *Lecture Notes in Computer Science*, pp. 415–442, Springer, 2022.

[44] B. Dutertre, "Yices 2.2," in *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings* (A. Biere and R. Bloem, eds.), vol. 8559 of *Lecture Notes in Computer Science*, pp. 737–744, Springer, 2014.

[45] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The mathsat5 SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings* (N. Piterman and S. A. Smolka, eds.), vol. 7795 of *Lecture Notes in Computer Science*, pp. 93–107, Springer, 2013.

[46] A. Niemetz and M. Preiner, "Bitwuzla at the SMT-COMP 2020," *CoRR*, vol. abs/2006.01621, 2020.

[47] R. Michel, A. Hubaux, V. Ganesh, and P. Heymans, "An smt-based approach to automated configuration," in *10th International Workshop on Satisfiability Modulo Theories, SMT 2012, Manchester, UK, June 30 - July 1, 2012* (P. Fontaine and A. Goel, eds.), vol. 20 of *EPiC Series in Computing*, pp. 109–119, EasyChair, 2012.

[48] M. Brockschmidt, F. Frohn, C. Fuhs, J. Giesl, J. Hensel, P. Schneider-Kamp, T. Ströder, and R. Thiemann, "Aprove at smt-comp 2020," 2020.

[49] F. Corzilius, G. Kremer, S. Junges, S. Schupp, and E. Ábrahám, "SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving," in *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings* (M. Heule and S. A. Weaver, eds.), vol. 9340 of *Lecture Notes in Computer Science*, pp. 360–368, Springer, 2015.

[50] J. Christ, J. Hoenicke, and A. Nutz, "Smtinterpol: An interpolating SMT solver," in *Model Checking Software - 19th International Workshop, SPIN 2012, Oxford, UK, July 23-24, 2012. Proceedings* (A. F. Donaldson and D. Parker, eds.), vol. 7385 of *Lecture Notes in Computer Science*, pp. 248–254, Springer, 2012.

[51] G. Reger, M. Suda, and A. Voronkov, "Instantiation and pretending to be an SMT solver with vampire," in *Proceedings of the 15th International Workshop on Satisfiability Modulo Theories affiliated with the International Conference on Computer-Aided Verification (CAV 2017), Heidelberg, Germany, July 22 - 23, 2017* (M. Brain and L. Hadarean, eds.), vol. 1889 of *CEUR Workshop Proceedings*, pp. 63–75, CEUR-WS.org, 2017.

[52] K. Korovin, "iprover - an instantiation-based theorem prover for first-order logic (system description)," in *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings* (A. Armando, P. Baumgartner, and G. Dowek, eds.), vol. 5195 of *Lecture Notes in Computer Science*, pp. 292–298, Springer, 2008.

[53] T. Bouton, D. C. B. D. Oliveira, D. Déharbe, and P. Fontaine, "verit: An open, trustable and efficient smt-solver," in *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings* (R. A. Schmidt, ed.), vol. 5663 of *Lecture Notes in Computer Science*, pp. 151–156, Springer, 2009.

[54] A. Stump, G. Sutcliffe, and C. Tinelli, "Starexec: A cross-community infrastructure for logic solving," in *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings* (S. Demri, D. Kapur, and C. Weidenbach, eds.), vol. 8562 of *Lecture Notes in Computer Science*, pp. 367–373, Springer, 2014.

[55] S. Cai, B. Li, and X. Zhang, "Yicesls: A local-search solver for smt." SMT-COMP 2021 Solver Description, 2021. Winner of QF_IDL track at SMT-COMP 2021.

[56] M. Balunovic, P. Bielik, and M. T. Vechev, "Learning to solve SMT formulas," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 10338–10349, 2018.