MedleySolver: Online SMT Algorithm Selection

Nikhil Pimpalkhare[1], Federico Mora[1], Elizabeth Polgreen[1 2], and Sanjit A. Seshia[1]

[1] University of California, Berkeley
[2] University of Edinburgh

Abstract. Satisfiability modulo theories (SMT) solvers implement a wide range of optimizations that are often tailored to a particular class of problems, and that differ significantly between solvers. As a result, one solver may solve a query quickly while another might be flummoxed completely. Predicting the performance of a given solver is difficult for users of SMT-driven applications, particularly when the problems they have to solve do not fall neatly into a well-understood category. In this paper, we propose an *online* algorithm selection framework for SMT called MedleySolver that predicts the relative performances of a set of SMT solvers on a given query, distributes time amongst the solvers, and deploys the solvers in sequence until a solution is obtained. We evaluate MedleySolver against the best available alternative, an *offline* learning technique, in terms of pure performance and practical usability for a typical SMT user. We find that with no prior training, MedleySolver solves 93.9% of the queries solved by the virtual best solver selector achieving 59.8% of the par-2 score of the most successful individual solver, which solves 87.3%. For comparison, the best available alternative takes longer to train than MedleySolver takes to solve our entire set of 2000 queries.

1 Introduction and Motivation

State-of-the-art Satisfiability Modulo Theory (SMT) solvers employ highly optimized and unique techniques to efficiently solve queries. One example of differentiation between solvers is in quantifier reasoning, where the number of different algorithms implemented is reflected in the wide spectrum of literature on the subject, e.g. [7, 16, 23, 27, 34]. In the same vein, solvers use very different techniques for different theories; for example, there are various techniques that can be used for bit-precise reasoning, e.g. [6, 9, 10, 14, 15, 17, 20, 22, 30, 31].

SMT solvers are becoming more widely used across various applications including verification, automated software testing, and policy verification, e.g. [4,8,24], making them particularly useful to industry practitioners and non-SMT researchers. Given the performance differential between solvers, a key question for such practitioners wishing to apply SMT solving to a problem in a specific domain is "which solver should I use?" In this work, we endeavor to provide a simple answer to this question: "Let MedleySolver choose for you!"

MedleySolver frames the problem of choosing an SMT-solver as a modified Multi-Armed Bandit (MAB) problem, a classic reinforcement learning formulation in which an agent must repeatedly pick from several different choices with

unknown reward distributions, minimizing overall regret. This agent must trade-off between exploitation (choosing a solver that is already believed to be fast) and exploration (testing out other solvers). For a given SMT query, MedleySolver selects a sequence of solvers to run, running the solver it believes is most likely to solve the query first and the solver that is least likely last. MedleySolver also predicts the time it should spend running each solver before it should give up and move onto the next solver in the sequence.

We apply classic algorithm selection techniques from the domain of Multi-Armed Bandit problems to the order selection problem. In this paper, we highlight Thompson Sampling and k-Nearest-Neighbor (k-NN) classification. We select these two as high-performing instances of a non-contextual and a contextual algorithm respectively but perform a more extensive comparison with a variety of other Multi-Armed Bandit algorithms for completeness. These algorithms traditionally predict one optimal action, but we use them to *rank* SMT solvers on a given query based on the behavior observed on previous queries and, in the case of the contextual bandit algorithms, a feature vector. This ranking allows the algorithm to explore and exploit in a single round.

Non-contextual multi-armed bandit algorithms have been directly applied to selecting search heuristics and variable orderings for constraint satisfaction problems [39,43], and to implementing co-operative sharing of clauses in parallel SAT solving [25]. Our use of a contextual multi-armed bandit framework to select a sequence of SMT solvers for a given SMT query is novel.

We pair these order selection algorithms with two runtime prediction algorithms. The first runtime prediction algorithm estimates the time each solver should be run by modeling its performance as exponential distributions with a parameter that is updated dynamically. The second fits a linear model using stochastic gradient descent (SGD). Runtime estimation helps reduce the cost of exploratory solver choices that do not produce rewards by stopping solvers when we are confident they will not finish before the overall timeout.

Our work is inspired by recent work showing machine learning techniques can be used to solve SMT queries faster. For example, FastSMT [5] is a tool that uses machine learning to find an optimal sequence of tactics, or query transformations, for SMT solvers to use on queries from a given domain. One issue with such approaches is that the complexity of the learning methods leads to training times grossly larger than the time spent solving. In this paper, we achieve comparable performance boosts with no pre-training or additional burden on the SMT end-user. To meet this goal, we approached the problem of algorithm selection for SMT solvers in a dynamic, or "online," manner. This ensures the cost of our training remains small, proportional, and justified by how SMT practitioners use SMT solvers. For example, techniques such as counterexample-guided inductive synthesis (CEGIS) [38] produce long sequences of similar SMT queries that are not easy to obtain prior to solving for offline training.

Contributions The key contributions of this work are:

1. An adaptation of standard regression techniques to predicting when a given solver will timeout on a query, and a novel approach for the same time

allocation problem that models runtime as exponential distributions and estimates timeouts dynamically and with context.
2. A framing of the SMT solver selection problem as a Multi-Armed Bandit (MAB) problem combined with a timeout prediction scheme. Specifically, we extend the MAB problem to selecting sequences of solvers per query instead of a single solver and use the timeout prediction scheme to allocate time to each solver in the sequence. This interaction lets us use lightweight techniques for both problems that do not require pre-training while retaining comparable performance to pre-trained techniques.
3. An empirical evaluation on a set of 2000 benchmarks representing a typical user's workload. Our approach solves 1813 queries on this set; 128 more than the next best solver in $3/5th$ of the time, with no pre-training.

2 Related Work

MedleySolver is most related to algorithm selection techniques for SAT and SMT. Our motivation, however, is similar to portfolio-based approaches.

Algorithm/Solver Selection Early approaches to learning-based algorithm selection in solvers included picking between different encodings of SMT to SAT in the UCLID solver [11, 36] and selecting input parameters for SAT solvers [19]. **SatZilla** [44] used empirical-hardness models to map queries to SAT solvers. Models are learned offline, then combined with a fixed order of "pre-solvers"—solvers that are called before featurization with a short timeout—when online. MedleySolver differs from SATZilla in that it targets SMT, learns solver orders, distributes time among solvers, and does not require training. **ArgoSmArT k-NN** [32] applies a pre-trained k-Nearest-Neighbor algorithm to portfolio SAT solving. Given a query, they deploy the most successful solver on the k nearest neighbors. Although one algorithm we apply is k-NN, we use it to select sequences of solvers and apply it in combination with the time-prediction algorithm. **MachSMT** [35] is a pre-trained tool like SatZilla but for SMT. Like SatZilla, MachSMT pre-trains to learn an empirical hardness model, then used to predict solving time for a given query. This is related to our timeout estimation, however, our version requires no pre-training. We achieve similar performance to MachSMT without pre-training by decoupling solver choice from time allocation and allowing for mistakes by selecting a sequence of solvers to run, instead of a single solver. **Where4** [18] is a portfolio-based SMT solver that uses regression models to select which solver to run. It extracts features from WhyML programs rather than SMT queries and does not allocate time between solvers. **CPHydra** [33] does allocate time between solvers but does so by solving an NP-Hard problem (knapsack). CPHydra also ignores solver order, requires offline training, and is aimed at CSP, which is related to, but different from, SMT. **FastSMT** [5] is a pre-trained learning tool for speeding up the Z3 SMT solver that works by selecting algorithmic "tactics" or strategies inside the solver itself. FastSMT is interesting because it produces an interpretable strategy that can often be significantly faster than Z3 out-of-the-box. MedleySolver differs from FastSMT in

that it learns to combine solvers, rather than to combine the tactics of a single solver. FastSMT also requires significant training time.

Parallel Portfolio Solvers Parallel portfolio solvers execute sets of solving processes in parallel for each query. Our approach is complementary in that we focus on speeding up sequential computation: different configurations of MedleySolver could be run in parallel with a portfolio approach. Nevertheless, portfolio approaches share a similar goal, so we highlight related works in this space. **PAR4** [40] is a basic portfolio parallel SMT solver that won several tracks in the SMT-competition in 2019. Wintersteiger et al. [42] implement **Parallel Z3**, a portfolio solver for SMT with the additional feature that learned clauses are shared between processes. Menouer and Baarir [28] combine search space-splitting with portfolio solving by using EXP3 to dynamically allocate cores to each search space splitting solver from a set. Our approach differs in that we focus on SMT, not SAT; we allocate time, not cores; and we use bandit algorithms (like EXP3) to pick the order of solvers, not the number of cores.

3 Problem Statement and Approach Overview

SMT users rarely aim to solve a single query in isolation and usually care about resource consumption. For example, verification engines generate many verification queries for one verification problem and aim to solve these queries in the least amount of time. As such, we define the *practical* SMT problem as that of taking a set of SMT queries $Q = \{q_1, ..., q_m\}$, a set of SMT algorithms (usually solvers) S, and producing a set of answers $A = \{a_1, ..., a_m\}$, where each a_i corresponds to the matching q_i, while using the least computational resources.

Our approach to the practical SMT problem, MedleySolver, is a program that takes Q, S, and a timeout T per query, and aims to maximize the number of instances solved while minimizing the cumulative time spent. We decompose our approach into two parts. For each query $q \in Q$, we predict 1. which solvers are most likely to solve a given query and return a list of solvers ordered by chance of success; and 2. the time each solver is likely to take to solve a given query and distribute the timeout T to the solvers in the sequence accordingly.

Given a query q_i, MedleySolver generates a sequence of solvers σ, and a sequence of time-allocations $t_1, ..., t_n$, so that the solver in σ_1 is run for t_1 seconds and so on. If a solver in the sequence successfully solves the query, we do not run the rest of the solver sequence on that query and instead move onto the next query. For the remainder of the paper, we use σ to denote a sequence of solvers and σ_i to denote the i^{th} solver in the sequence σ. We process each query $q_1, ..., q_m$ in order and our solver selection algorithms learn as we go, so the solver selection for q_m uses information from queries $q_1, ..., q_{m-1}$. In practice, this historical information can be reset whenever and in our experimental results, we reset it when confronting a new set of queries (for instance, a new category of the SMT competition). An overview of our approach is shown in Figure 1. We describe the three main components of Figure 1: the solver selector, the timeout predictor, and the featurization of queries, in detail in the next two sections.

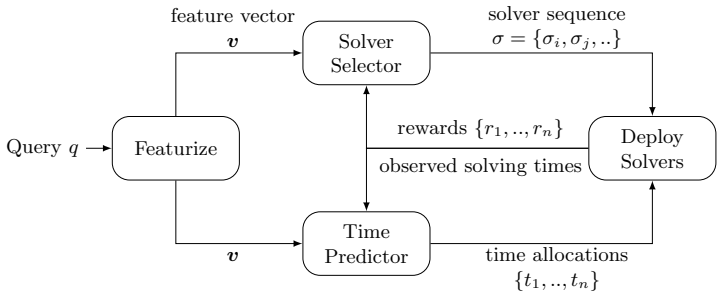


Fig. 1: Overview of one iteration of our approach.

4 Dynamic Solver Selection

In the **multi-armed bandit (MAB)** problem [13] an agent sequentially selects between choices with unknown associated reward distributions, aiming to maximize the reward achieved over time. The agent must trade off exploration (trying new actions and learning about them) and exploitation (deploying actions we know have the potential for high reward).

We frame the solver selection problem as a MAB problem. The agent is selecting the solver to use and the payout is based on successfully solving queries. We assume that running a solver for a randomly selected SMT query is equivalent to sampling from some unknown distribution that we seek to approximate. Contextual MABs extend the problem by giving agents access to a feature vector before each round. This allows us to add information about the characteristics of the SMT query we are trying to solve in each round, as described in Section 4.2.

We modify the MAB problem in one key way: we select a sequence of solvers to run (with corresponding time allocations that we consider later), instead of selecting a single solver. This contribution allows solver selection to perform exploration on each query until it observes a reward, has tried all solvers, or reaches the time-out per query. We also use a time-prediction algorithm, described in Section 5 which predicts the time it is worth running a solver on a given query, allowing us to perform "partial exploration" instead of committing to running a single solver until time-out or termination. Both of these extensions to the MAB algorithms allow the solver selection to correct incorrect solver choices, reducing the cost of exploration vs exploitation. In the following subsections, we adapt one non-contextual algorithm and one contextual algorithm (i.e., algorithms that use the feature vector) from the literature to our domain. We choose the algorithms based on their popularity in the classic literature for MAB, and the compatibility of the assumptions the algorithm makes with our domain (for instance, we omit algorithms such as LinUCB [26], which assumes that the reward is linearly correlated with the feature vector). In Section 6 we evaluate our choices against other MAB algorithms for completeness.

Rewards We use a binary reward structure where a solver receives a reward of 1 if it is observed solving a query and 0 if it is observed failing to solve a query,

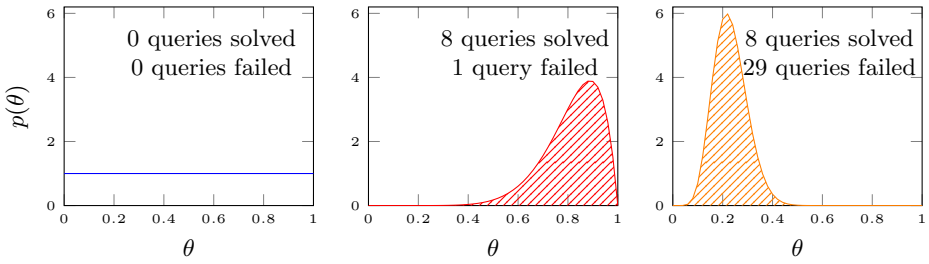


Fig. 2: Updating the distribution of θ according to Bayes' rule. A distribution for a solver that has failed on more queries than it has solved will have a $\theta < 0.5$.

which decouples solving time from rewards. We also explored an exponential reward structure where a solver receives a reward of $(1 - t/T)^4$ if it solves a query in time t, but found the binary reward more effective which we believe is due to its ability to differentiate more clearly between benchmarks that are slow to solve and benchmarks that are not solved.

4.1 Thompson Sampling

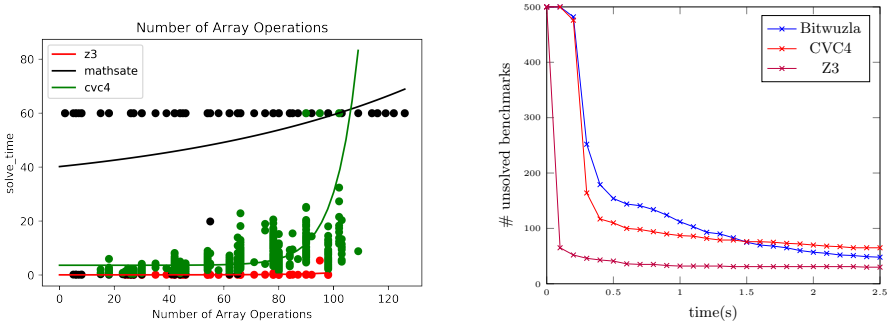
Thompson Sampling [1, 2] uses Bayes' rule to choose an action, or arm in the MAB problem, that maximizes the expected reward. Each round of the MAB in this context is picking a random query from the set of queries and trying to solve it with a specific solver σ_i. To adapt non-contextual Thompson Sampling to our SMT solver selection problem, we model the outcome of an experiment with a Bernoulli distribution where the solver solves the query with a probability θ_i and fails to solve it within the time-out with a probability $1 - \theta_i$. In Thompson Sampling, the agent does not know the value of each θ_i but begins with some prior belief over each one. These priors are beta-distributed: the prior for θ_i is

$$p(\theta_i) = \frac{\Gamma(\alpha_i + \beta_i)}{\Gamma(\alpha_i)\Gamma(\beta_i)} \theta_i^{\alpha_i - 1}(1 - \theta_i)^{\beta_i - 1}.$$

We initially take this distribution to be uniform i.e., $\alpha_i = \beta_i = 1$. That is, we assume a prior that, for a random query, each solver has a 50% chance of solving the query and a 50% chance of failing to solve the query within the timeout.

To select a solver to deploy, Thompson Sampling takes a sample from each distribution $p(\theta_i)$ corresponding to a solver. Note that, because Thompson Sampling takes a sample from the distributions $p(\theta_1)...p(\theta_n)$, it is more likely to pick solvers that we are uncertain about instead of simply returning the solvers in order of the $p(\theta)$ with the highest mean, allowing exploration. Conventional Thompson Sampling returns the solver with the highest valued sample. Our algorithm returns a sequence of solvers in descending order of these sampled values i.e., the solver with the θ closest to 1 is first.

After deploying the solvers and observing the results, the distributions over $\theta_1, .., \theta_n$ are updated according to Bayes's rule. Each time a solver σ_i is run, a



(a) Solver performance on Uclid5 queries as the number of array operations increases.

(b) Three solver's runtime over a sample of bit-vector queries.

Fig. 3: Empirical intuition for features and time predictors.

reward r_i is observed. The posterior distribution for the beta distribution [12] is obtained by adding the reward r_i to α_i and $1-r_i$ to β_i, as illustrated in Figure 2.

Thompson Sampling assumes events are independent, i.e., the probability of a solver being able to solve a query is independent of all queries the solver has seen before. This could be true if our time-out prediction algorithm is perfect so that if a solver can solve a query within the timeout T, the solver will also always be able to solve that query within the time t allocated to the solver.

4.2 Features for Contextual Approaches

Contextual approaches depend on the assumption that queries with similar characteristics will cause SMT solvers to perform similarly. We capture the characteristics of each SMT query q_i in a feature vector $\boldsymbol{v}_i \in V$. We use these feature vectors both in contextual bandit algorithms (described next) and in our contextual time-prediction (described in Section 5).

We identify a list of 24 features that are quick to extract and that we believe correlate with solving time for specific solvers. These features include context-free qualities like counts for specific operators (e.g. array store operators), the maximum value of literals, the sum of literal values, and so on. The features also include context-sensitive qualities like quantifier nesting and alternations, as well as the size of a given queries' abstract-syntax-tree as a minimal graph (we refer to this representation as a term graph). All feature extraction procedures run in $\mathcal{O}(n)$ where n is the size of the term graph. The term graph construction is efficient and the term graph itself is often exponentially smaller than the input query. Therefore, the cost of extracting features is relatively small.

As a heuristic, we try to build features that can differentiate solver performance on their own. For example, Figure 3a shows the performance of three different SMT solvers as the number of array operations in the term graph increases accompanied by an exponential regression fit. For this example, intuitively, we would want to favor the use of Z3 as the number of array operations

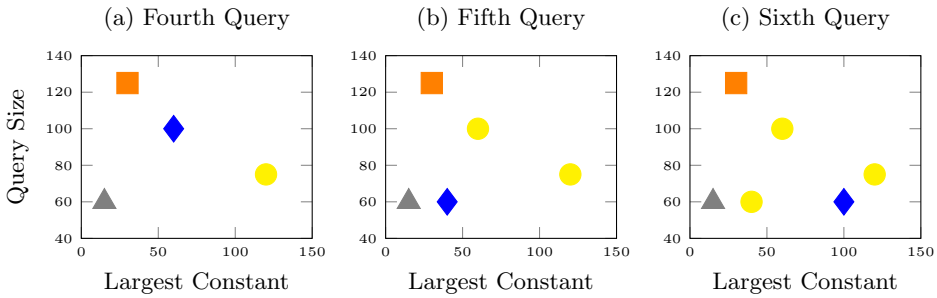


Fig. 4: Example k-NN run with $k = 1$, solvers A, B, C, and 2 features. A, B, and C are represented by a square, a triangle, and a circle, respectively; the new query is represented by a diamond. A and B fail on the new query in (a); solver B fails on the new query in (b); and solver C succeeds on both.

increases. While we did not do any empirical feature selection, we did use our prior knowledge of SMT solvers to decide which feature extractors to build, and we evaluate the impact of our decisions in Section 6.3.

4.3 *k*-Nearest-Neighbor

The k-Nearest-Neighbor algorithm(k-NN) is a simple contextual approach. Viewed from the perspective of a MAB problem, given solvers $s_1, ..., s_n$, k-NN classifies SMT queries into n classes: queries where solver s_1 is the best choice, queries where solver s_2 is the best choice, and so on. We extend the standard k-NN algorithm to return a sequence of solvers.

Given a query, q, the basic k-NN algorithm looks at the k nearest queries to q, tallies the number solved by each solver, and orders the solvers by their tallies. We calculate the distance between two queries—how "near" two queries are—by computing the Euclidean distance between their feature vectors. The idea behind this algorithm is that if a solver succeeded on many queries similar to q, then it is likely to succeed on q. If any solvers in the solver set are not included in the neighbors, we randomly shuffle these solvers and append them to the end of the sequence—we make an exception for the $k = 1$ case: when $k = 1$ we return the solver that solved the nearest neighbor, followed by the solver that solved the next nearest neighbor and so on, without replacement.

Once an order is selected, we run the solvers in sequence. If a solver s in this sequence succeeds, we add the feature vector of that query to our data-set of previously solved queries along with the label s. See Figure 4 for three steps of a hypothetical 1-nearest neighbor example.

5 Runtime prediction

The second component of our approach comprises time predictors, which we use to split the per-query timeout T into sub-timeouts per solver $t_1 \dots t_n$. We train a

time-predictor for each solver. Our goal is to find a time t_i such that we can stop running the i^{th} solver in the sequence and be highly confident it was unlikely to solve the current query after this point. Formally, we are trying to find the minimum t_i such that $P(t_i < u_i < T) \leq \delta$, where u_i is the true runtime of σ_i on q_i and δ is the accepted error probability. We consider this event the only relevant error scenario because it implies if we had allocated more time to solver σ_i we could have solved the current query.

To calculate each t_i, we model each solver's runtime as an exponential distribution, justified by our experimental observations illustrated in Figure 3b: solvers usually succeed early or not at all. We employ Maximum Likelihood Estimation (MLE) [29] to fit an exponential distribution to the runtime samples which we have gathered up to that point. In MLE, we find $\min_\lambda P(u_1 \dots u_m|\lambda)$, where $u_1 \dots u_m$ are the observed runtime samples we have seen, which we assume are drawn from $Exponential(\lambda)$. We use the exponential's probability distribution function as a measure of likeliness, so this problem is equal to $\min_\lambda n \ln \lambda - \lambda(\sum_i u_i)$, leading to the following minimizer:

$$\lambda^* = \frac{n}{\sum_i u_i}$$

Applying the cumulative distribution function and using the memoryless property of the exponential distribution, we can calculate t_i as follows:

$$t_i = \frac{-\ln(\delta + e^{-\lambda^* T})}{\lambda^*}$$

We split T into sub-timeouts greedily; we use the above process to allocate time for solvers starting from the beginning of our ordering and stop once we reach the overall timeout, allocating zero time to the remaining solvers in the order. If we reach the end of the ordering and still have time remaining, we give the remaining timeout to the last solver.

***k*-NN Runtime Prediction** We present a contextual runtime prediction system based on the k-NN algorithm. Instead of using every past sample point to estimate λ^*, we limit our estimation scheme to the k nearest data-points. As with the k-NN based solver selection, the distance between two queries is the Euclidean distance between their feature vectors. The rest of the estimation scheme remains identical to the non-contextual scheme.

Linear Regression Runtime Prediction Finally, we present a contextual runtime prediction system that finds a linear relationship between our feature vector and the associated runtime. To do so, it minimizes the L2-regularized squared loss of the linear model using stochastic gradient descent, solving:

$$\min_{w,b} \sum_i (w^T x_i + b - u_i) + \alpha ||w||_2,$$

where w is the learned weight of our features, b is a learned coefficient, x_i is the feature vector of the i^{th} query, u_i is the true runtime of the i^{th} query, and α is a regularization constant.

6 Empirical Evaluation

We implemented a prototype of MedleySolver in Python.[3] The input is a directory of queries, and the output is the result, solver used, and time elapsed per query. In this section, we evaluate this prototype and aim to answer the following research questions: 1. How does MedleySolver compare to individual solvers on the practical SMT problem? 2. How does MedleySolver compare to the best available alternatives on the practical SMT problem? 3. How do the individual components of MedleySolver affect the overall performance?

Subjects and Methods We equip MedleySolver with six SMT solvers (CVC4 v1.8 [6], MathSAT v5.6.3 [9], Z3 v4.8.7 [14], Boolector v2.4.1 [31], Bitwuzla v.0.9999 [30] and Yices v2.6.2 [15]) and run on four benchmark sets, each with 500 queries. Some SMT solvers do not support all needed syntax. For example, the BV set includes quantifiers that Boolector cannot handle. We expect MedleySolver to learn to avoid solvers that fail on specific kinds of queries.

The benchmark queries simulate a typical user's workload in that they are similar in nature, i.e. use related logical theories and come from similar applications, but are diverse enough to expose issues a normal user will encounter, i.e. deviations in SMT-LIB conformance. We selected a random sample of 500 queries from an existing benchmark set, Sage2, derived from a test generation tool; 500 queries from 140 Uclid5 [37] verification tasks; and 500 queries each from the BV and QF_ABV theory SMT-COMP tracks, respectively.

We ran every individual solver with a timeout of 60 seconds for every query on a Dell PowerEdge C6220 server blade equipped with two Intel Xeon 10-core Ivy Bridge processors running Scientific Linux 7 at 2.5 GHz with 64 GB of 1866 Mhz DDR3 memory. We saved these results and used them to simulate runs of MedleySolver. This helped ensure results are deterministic, reproducible, and lowered our carbon emissions. The overhead of running MedleySolver on all 2000 queries varies between learning algorithms and features used but is always less than two minutes for the full set of queries, and is therefore negligible.

6.1 RQ1: Comparison With Individual Solvers

To evaluate the utility of MedleySolver for a typical user, we ran k-NN and Thompson with the three timeout predictors on every set individually and then over the combined set. The combined set represents a realistic combination of queries a typical user might want to solve.

[3] Code and data: `https://github.com/uclid-org/medley-solver/tree/SAT2021`

	Benchmark Set					
Solver	BV	QFABV	Sage2	Uclid5	Split	Combined
10-NN	**2081.8**	1208.2	14855.4	5386.9	**23532.3**	29111.2
Expo	484	492	396	457	1829	1799
10-NN	2857.7	1039.7	17558.2	5386.9	26842.5	**25133.8**
10-Nearest-Expo	477	493	367	457	1794	1813
10-NN	4229.1	**824.4**	**13789.0**	8136.4	26978.9	87898.4
Linear	468	496	409	452	1825	1308
Thompson	2986.4	1308.5	15323.1	5540.9	25158.9	51757.3
Expo	480	491	401	456	1828	1676
Thompson	2840.8	1105.1	17949.7	5536.0	27431.6	27555.9
10-Nearest-Expo	479	493	365	456	1793	1816
Thompson	4291.1	1474.1	15661.3	**4344.6**	25771.1	45267.7
Linear	466	489	392	473	1820	1658
Boolector	60000.0	1408.7	31502.3	60000.0		152911.1
	0	491	265	0		756
Bitwuzla	3872.3	**822.9**	22316.4	60000.0		87011.7
	471	496	349	0		1316
CVC4	3332.1	5874.1	49161.5	7395.5		65763.3
	477	459	117	459		1512
MathSat	11455.0	1724.0	34159.4	50783.3		98121.7
	406	488	232	77		1203
Yices	7244.3	922.1	**13544.4**	60000.0		81710.9
	442	494	411	0		1347
Z3	**2888.6**	1202.0	35279.2	**2637.2**		**42007.1**
	477	492	232	484		1685
Virtual Best	964.6	530.8	9006.2	2192.1		12786.5
	493	497	453	476		1931

Table 1: Par-2 score (lower is better) and the number of queries solved for each solver across benchmarks. MedleySolver configurations are selectors (e.g. 10-NN) over time predictors (e.g. Linear time prediction). Learning algorithms use binary reward and every query is given a 60-second timeout. 'Split' refers to the sum over all individual benchmark sets where the learning algorithms are restarted between sets, while 'Combined' refers to the aggregated set with no resets. Individual sets contain 500 queries; 'Combined' and 'Split' contain 2000.

Table 1 reports the results of our experiment in terms of Par-2 score,[4] where "virtual best" is calculated by using the best-performing individual solver for each query. On individual sets where one solver dominates, like Z3 on Uclid5, MedleySolver approaches the best solver but does not reach it. Conversely, on sets where no one solver is close to the virtual best, like BV, we find the MedleySolver can exploit this performance differentiation and approach the virtual best solver. The combined set, which is less uniform than BV but less dominated

[4] sum of all runtimes for solved instances + 2∗timeout for unsolved instances [41].

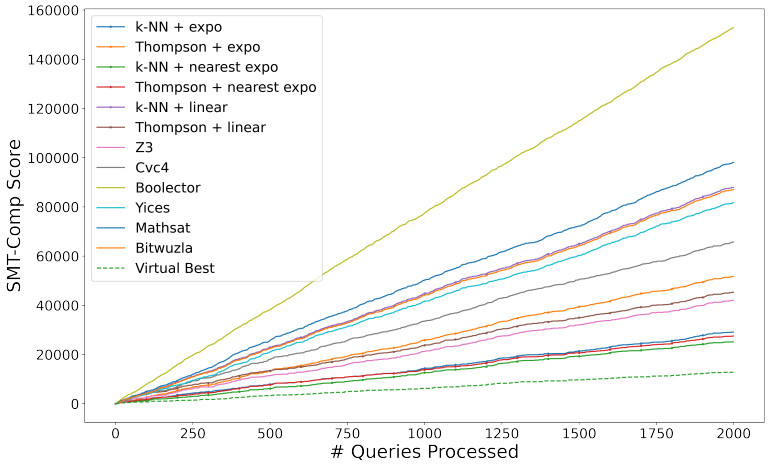


Fig. 5: Par-2 score over queries solved in the combined set.

than Uclid5, demonstrates the power of MedleySolver: with no training, MedleySolver solves 94.5% of the queries solved by the virtual best using only 72.3% of the time taken by the most successful individual solver, Z3, which solves 87.3%. Figure 5 shows the performance of every solver over the number of queries processed, and visually depicts the proximity of MedleySolver to the virtual best. Together, Table 1 and Figure 5 answer RQ3: MedleySolver outperforms every individual solver on the practical SMT problem.

The Thompson MAB selector generally does better when summing up individual sets than when running on the combined set, while the k-NN selector is the opposite. This suggests contextual approaches can effectively carry over lessons between sets, and non-contextual approaches benefit from being used in the context of a benchmark.

6.2 RQ2: Comparison With State-of-the-Art

We now compare MedleySolver to alternative portfolio approaches, including those based on pre-trained machine learning techniques. Parallel portfolio solvers like PAR4 [40] run multiple solvers in parallel and stop all solvers when the first one solves the query. We can calculate the hypothetical performance of such a solver by multiplying the virtual best solver time by the number of solvers we run. This would give a Par-2 score of 65481, in comparison to 10-NN's better score of 32632 over the combined benchmark set.

MachSMT [35] uses a neural network to select which solver to run on a given query. Table 2 shows the performance of MachSMT and MedleySolver on the same benchmarks as in Table 1 but with 2/5 of the queries set aside for MachSMT to train on per set. Although MachSMT slightly outperforms MedleySolver on the test sets, the training time required by MachSMT is orders of magnitude larger than the time required to solve, particularly because, to have

Solver	Benchmark Set BV	QFABV	Sage2	Uclid5	Combined
MedleySolver	1638.7	310.5	9245.3	4248.0	18565.5
	N/A	N/A	N/A	N/A	N/A
MachSMT	1458.3	919.2	8516.1	2430.9	12539.1
	33895.5s	4498.9s	55115.5s	276419.8s	300072.8s
Virtual Best	801.7	184.3	5204.2	1464.7	6746.0
	N/A	N/A	N/A	N/A	N/A

Table 2: Par-2 score (lower is better) and training time for MedleySolver, MachSMT, and the Virtual Best of all individual solvers. Individual benchmarks consist of 200 training queries and 300 test queries; 'Combined' consists of 800 training queries and 1200 test queries. Only MachSMT uses the training queries.

training data on which to train, MachSMT must run all the solvers on all the queries in the training set which takes a considerable amount of time. So, on the practical SMT problem, MedleySolver achieves similar results with significantly less resource consumption and we argue the cost of training is not worth it if online learning can achieve competitive performance. Preiner et al. do make a trained model available that could eliminate training time for a user. However, this pre-trained model is trained on specific versions of a specific set of SMT solvers running on a specific system. In practice and in our example, the user's specifics do not match and local training is required.

FastSMT [5] is an offline approach that synthesizes strategies for the Z3 SMT solver. FastSMT requires significant training time and needs to be trained per benchmark set. We can run the pre-trained FastSMT model on the Sage2 benchmark set, where it solves 358 queries in 12766s (par-2 score of 29806). This is a substantial performance gain over Z3, but all MedleySolver configurations still outperform this without any pre-training. FastSMT improves on Z3 but it is limited to one single solver and, unlike MedleySolver, is not able to take advantage of the range of different SMT solvers implementing different heuristics.

6.3 RQ3: Impact of Individual Components

In this section, we evaluate the performance of the individual pieces of MedleySolver. Specifically, we aim to answer the following questions. 1. How do our learning algorithms compare to other well-known MAB algorithms? 2. How well do our order selectors perform? 3. What is the impact of selecting an order instead of a single solver on performance? 4. Which query features are most responsible for MedleySolver's performance?

Performance of Other Multi-Armed Bandit Algorithms We have highlighted the results from Thompson Sampling and k-NN but we also adapted and evaluated the following Multi-Armed Bandit algorithms: the classic non-contextual epsilon-greedy bandit algorithm [21]; LinUCB [26], an upper-confidence bound algorithm that assumes a linear relationship between the rewards and the feature vector; Exp3 [3]: an adversarial bandit algorithm; and an adaptation of a

neural network classification based bandit [45]. All of these bandits performed comparably or better than the best individual solver, but no non-contextual algorithm performed as well as Thompson Sampling, and no contextual algorithm performed as well as k-NN.

Order Selection Accuracy To better understand MedleySolver's performance, we measure how frequently it selects the best solver as the first solver to try. Over our five case-study benchmarks, the best performing selector is k-Nearest-Neighbor with $k = 10$ (10-NN), which correctly picks the best solver 74.3% of the time. The highest success rate is on the BV benchmark, where 10-NN is 92.0% accurate. The lowest success rate is on the QF_ABV benchmark, where 10-NN is 52.0% accurate. This difference demonstrates MedleySolver's accuracy is proportional to the cost of mistakes: solvers are much better overall on the QF_ABV category so MedleySolver can frequently pick a sub-optimal solver that will still terminate quickly; on the other hand, in BV, picking the wrong solver will often lead to a timeout.

Timeout Prediction Impact To better understand MedleySolver's performance, we measure the impact of selecting an order of solvers instead of a single solver. To do this we run MedleySolver without timeout prediction, giving the entire time per query to the first solver in the sequence. We find, all else equal, on the combined set, selecting a single solver produces a par-2 score 350% worse than our best MedleySolver configuration. This difference is due to the direct cost of mispredictions and because, without the time prediction, MedleySolver is unable to learn from mistakes on a given query.

Feature Evaluation In this section, we aim to interpret what our results tell us about SMT solvers and optimize performance through feature analysis. To better understand the SMT solvers we use, we measure how well each feature correlates with solver performance differentiation. Specifically, for every feature f, for every pair of solvers $(s_i, s_j) \in S \times S$, we measure the Pearson correlation coefficient between f and $\text{time}(s_i) - \text{time}(s_j)$. Using this technique we found, on the BV benchmark set, the most distinguishing features were the number of universal quantifiers for Bitwuzla and MathSat5; the number of free variables for Yices and Boolector; the number of bound variables for Boolector and Bitwuzla; and the size of the term graph for Bitwuzla and Z3.

To optimize performance, we search for the subset of features that induce the best performance from MedleySolver. Specifically, we use backward step-wise feature selection (BSFS) to iteratively remove features from our feature vector whose removal does not negatively affect performance. Using this technique we found that, all else equal, using only three features on the BV benchmark (number of quantifiers, number of variables, and term graph size) improves the par-2 score of the best configuration of MedleySolver by 30%. We observe a similar reduction in feature vector size across benchmarks and an average par-2 performance improvement of 11%. Interestingly, BSFS often removes the feature with the smallest correlation score, as described above.

6.4 Threats To Validity

We evaluated MedleySolver on a set of benchmarks and a combination of solvers we believe represent a real user's SMT workload, but we understand these results may not generalize. To mitigate this possibility, we evaluated the SMT-Competition data curated by the MachSMT authors. On the QF_UFBV benchmark, the best individual solver was Bitwuzla with a score of 2614.6 while MedleySolver scored 2600.5; on the QF_LIA benchmark, the best individual solver was Yices with a score of 45871.2 while MedleySolver scored of 25737.2; on the QF_BVFPLRA benchmark, the best individual solver was MathSAT5 with a score of 3015.2 while MedleySolver scored 567.7; and on the NRA benchmark, the best individual solver was Z3 with a score of 1455.6 while MedleySolver scored 1068.6. In all cases, MedleySolver outperformed every individual solver while using no pre-training–often by margins greater than those observed in our case study–suggesting our results do generalize.

The queries MedleySolver has seen in the past affect the prediction MedleySolver makes for the current query, and thus the order MedleySolver receives the queries could affect the overall performance on the full dataset. MAB algorithms such as Thompson Sampling use random sampling and choice of random seed could also affect the results. To gain confidence in our claims, we repeated our experiments with 20 different random seeds and found the standard deviation in MedleySolver's overall par-2 score to be approximately 1% of its average score. The margins between MedleySolver and any individual solver are significantly larger than 1% and so MedleySolver is consistently comfortably better than any individual solver in our evaluation regardless of deviation.

7 Conclusions and Future Work

We presented MedleySolver, an online learning algorithm for SMT that uses a novel application of multi-armed bandits to predict the best order in which to deploy a sequence of SMT solvers, in combination with a novel time-prediction algorithm, allowing the solver selection to recover from mistakes. Our approach solves more queries in less time per query than any individual solver on a set of benchmarks taken from the SMT-competition and verification tasks. Unlike offline techniques, MedleySolver requires no pre-training.

In the future, we intend to explore white-box techniques for solver termination prediction. We hypothesize that monitoring a solver's execution can help identify when the solver is unlikely to terminate. We are also interested in exploring online feature selection techniques.

Acknowledgments: This work was supported in part by NSF grants CNS-1739816 and CCF-1837132, by the DARPA LOGiCS project under contract FA8750-20-C-0156, by the iCyPhy center, and by gifts from Intel, Amazon, and Microsoft.

References

1. Agrawal, S., Goyal, N.: Analysis of Thompson sampling for the multi-armed bandit problem. In: COLT. JMLR Proceedings, vol. 23, pp. 39.1–39.26. JMLR.org (2012)
2. Agrawal, S., Goyal, N.: Thompson sampling for contextual bandits with linear payoffs. In: ICML (3). JMLR Workshop and Conference Proceedings, vol. 28, pp. 127–135. JMLR.org (2013)
3. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: The nonstochastic multi-armed bandit problem. SIAM J. Comput. **32**(1), 48–77 (2002)
4. Backes, J., Bolignano, P., Cook, B., Dodge, C., Gacek, A., Luckow, K.S., Rungta, N., Tkachuk, O., Varming, C.: Semantic-based automated reasoning for AWS access policies using SMT. In: FMCAD. pp. 1–9. IEEE (2018)
5. Balunovic, M., Bielik, P., Vechev, M.T.: Learning to solve SMT formulas. In: NeurIPS, pp. 10338–10349 (2018)
6. Barrett, C.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6806, pp. 171–177. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_14, `https://doi.org/10.1007/978-3-642-22110-1_14`
7. Barth, M., Dietsch, D., Fichtner, L., Heizmann, M.: Ultimate eliminator: a quantifier upgrade for smt solvers at smt-comp 2019 (2019)
8. Bjørner, N.: SMT solvers for testing, program analysis and verification at microsoft. In: SYNASC. p. 15. IEEE Computer Society (2009)
9. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The mathsat4 smt solver. In: Gupta, A., Malik, S. (eds.) Computer Aided Verification. pp. 299–303. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
10. Bryant, R.E., Kroening, D., Ouaknine, J., Seshia, S.A., Strichman, O., Brady, B.: Deciding bit-vector arithmetic with abstraction. In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science, vol. 4424, pp. 358–372. Springer (March 2007)
11. Bryant, R.E., Lahiri, S.K., Seshia, S.A.: Modeling and verifying systems using a logic of counter arithmetic with lambda expressions and uninterpreted functions. pp. 78–92. LNCS 2404 (July 2002)
12. Castillo, E.F., Hadi, A.S., Solares, C.: Learning and updating of uncertainty in dirichlet models. Mach. Learn. **26**(1), 43–63 (1997)
13. Cesa-Bianchi, N., Lugosi, G.: Prediction, learning, and games. Cambridge university press (2006)
14. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. p. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008)
15. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) Computer-Aided Verification (CAV'2014). Lecture Notes in Computer Science, vol. 8559, pp. 737–744. Springer (July 2014)
16. Ge, Y., De Moura, L.: Complete instantiation for quantified formulas in satisfiabiliby modulo theories. In: International Conference on Computer Aided Verification. pp. 306–320. Springer (2009)
17. Hansen, T.: A constraint solver and its application to machine code test generation. Ph.D. thesis, University of Melbourne, Australia (2012), `http://hdl.handle.net/11343/37952`

18. Healy, A., Monahan, R., Power, J.F.: Predicting SMT solver performance for software verification. In: F-IDE@FM. EPTCS, vol. 240, pp. 20–37 (2016)
19. Hutter, F., Babic, D., Hoos, H.H., Hu, A.J.: Boosting verification by automatic tuning of decision procedures. In: 7th Intl. Conference on Formal Methods in Computer-Aided Design (FMCAD). pp. 27–34 (2007)
20. Jha, S., Limaye, R., Seshia, S.A.: Beaver: engineering an efficient smt solver for bit-vector artithmetic. In: Proc. 21st International Conference on Computer-Aided verification (CAV). Lecture Notes in Computer Science, vol. 5643, pp. 668–674 (June 2009)
21. Johnson, J.D., Li, J., Chen, Z.: Reinforcement learning: An introduction: R.S. sutton, A.G. barto, MIT press, cambridge, MA 1998, 322 pp. ISBN 0-262-19398-1. Neurocomputing **35**(1-4), 205–206 (2000)
22. Jonáš, M., Strejček, J.: Solving quantified bit-vector formulas using binary decision diagrams. In: Creignou, N., Le Berre, D. (eds.) Theory and Applications of Satisfiability Testing – SAT 2016. pp. 267–283. Springer International Publishing, Cham (2016)
23. Kovács, L., Robillard, S., Voronkov, A.: Coming to terms with quantified reasoning. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. pp. 260–270 (2017)
24. Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View, Second Edition. Texts in Theoretical Computer Science. An EATCS Series, Springer (2016)
25. Lazaar, N., Hamadi, Y., Jabbour, S., Sebag, M.: BESS: Bandit Ensemble for parallel SAT Solving. Research Report RR-8070 (Sep 2012), `https://hal.inria.fr/hal-00733282`
26. Li, L., Chu, W., Langford, J., Schapire, R.E.: A contextual-bandit approach to personalized news article recommendation. In: Proceedings of the 19th international conference on World Wide Web. pp. 661–670. ACM (2010)
27. Löding, C., Madhusudan, P., Peña, L.: Foundations for natural proofs and quantifier instantiation. Proceedings of the ACM on Programming Languages **2**(POPL), 1–30 (2017)
28. Menouer, T., Baarir, S.: Parallel learning portfolio-based solvers **108**, 335–344 (2017)
29. Myung, I.J.: Tutorial on maximum likelihood estimation. J. Math. Psychol. **47**(1), 90–100 (Feb 2003). https://doi.org/10.1016/S0022-2496(02)00028-7, `https://doi.org/10.1016/S0022-2496(02)00028-7`
30. Niemetz, A., Preiner, M.: Bitwuzla at the SMT-COMP 2020. CoRR **abs/2006.01621** (2020), `https://arxiv.org/abs/2006.01621`
31. Niemetz, A., Preiner, M., Biere, A.: Boolector 2.0. J. Satisf. Boolean Model. Comput. **9**(1), 53–58 (2014). https://doi.org/10.3233/sat190101, `https://doi.org/10.3233/sat190101`
32. Nikolic, M., Maric, F., Janicic, P.: Simple algorithm portfolio for SAT. Artif. Intell. Rev. **40**(4), 457–465 (2013)
33. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: Irish conference on artificial intelligence and cognitive science. pp. 210–216 (2008)
34. Reynolds, A., Deters, M., Kuncak, V., Tinelli, C., Barrett, C.: Counterexample-guided quantifier instantiation for synthesis in smt. In: International Conference on Computer Aided Verification. pp. 198–216. Springer (2015)

35. Scott, J., Niemetz, A., Preiner, M., Nejati, S., Ganesh, V.: Machsmt: A machine learning-based algorithm selector for SMT solvers. In: Groote, J.F., Larsen, K.G. (eds.) 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. Lecture Notes in Computer Science, vol. 12652, pp. 303–325. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_16, https://doi.org/10.1007/978-3-030-72013-1_16
36. Seshia, S.A.: Adaptive Eager Boolean Encoding for Arithmetic Reasoning in Verification. Ph.D. thesis, Carnegie Mellon University (May 2005)
37. Seshia, S.A., Subramanyan, P.: UCLID5: integrating modeling, verification, synthesis and learning. In: MEMOCODE. pp. 1–10. IEEE (2018)
38. Solar-Lezama, A., Tancau, L., Bodík, R., Seshia, S.A., Saraswat, V.A.: Combinatorial sketching for finite programs. In: ASPLOS. pp. 404–415. ACM (2006)
39. Wattez, H., Koriche, F., Lecoutre, C., Paparrizou, A., Tabary, S.: Learning variable ordering heuristics with multi-armed bandits and restarts. In: ECAI. Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 371–378. IOS Press (2020)
40. Weber, T.: Par4 system description. https://smt-comp.github.io/2019/system-descriptions/Par4.pdf
41. Weber, T., Conchon, S., Déharbe, D., Heizmann, M., Niemetz, A., Reger, G.: The SMT competition 2015-2018. J. Satisf. Boolean Model. Comput. **11**(1), 221–259 (2019)
42. Wintersteiger, C.M., Hamadi, Y., de Moura, L.M.: A concurrent portfolio approach to SMT solving. In: CAV. Lecture Notes in Computer Science, vol. 5643, pp. 715–720. Springer (2009)
43. Xia, W., Yap, R.H.C.: Learning robust search strategies using a bandit-based approach. In: AAAI. pp. 6657–6665. AAAI Press (2018)
44. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. **32**, 565–606 (2008)
45. Zhou, D., Li, L., Gu, Q.: Neural contextual bandits with upper confidence bound-based exploration. CoRR **abs/1911.04462** (2019)