Frugal Algorithm Selection

Erdem Kuş
School of Computer Science, University of St Andrews, UK

Özgür Akgün
School of Computer Science, University of St Andrews, UK

Nguyen Dang
School of Computer Science, University of St Andrews, UK

Ian Miguel
School of Computer Science, University of St Andrews, UK

Abstract

When solving decision and optimisation problems, many competing algorithms (model and solver choices) have complementary strengths. Typically, there is no single algorithm that works well for all instances of a problem. Automated algorithm selection has been shown to work very well for choosing a suitable algorithm for a given instance. However, the cost of training can be prohibitively large due to running candidate algorithms on a representative set of training instances. In this work, we explore reducing this cost by choosing a subset of the training instances on which to train. We approach this problem in three ways: using active learning to decide based on prediction uncertainty, augmenting the algorithm predictors with a timeout predictor, and collecting training data using a progressively increasing timeout. We evaluate combinations of these approaches on six datasets from ASLib and present the reduction in labelling cost achieved by each option.

2012 ACM Subject Classification Theory of computation → Active learning; Theory of computation → Constraint and logic programming

Keywords and phrases Algorithm Selection, Active Learning

Digital Object Identifier 10.4230/LIPIcs.CP.2024.38

Category Short Paper

Supplementary Material *Other (Source code, data, experimental results and appendix)*: `https://doi.org/10.5281/zenodo.13294528` [24]

Funding The experiments made use of Cirrus, a UK National Tier-2 HPC Service at EPCC (`http://www.cirrus.ac.uk`) funded by the University of Edinburgh and EPSRC (EP/P020267/1). *Ian Miguel*: Ian Miguel is funded by EPSRC grant EP/V027182/1.

1 Introduction

Solving combinatorial optimisation problems is a challenging task, where often multiple approaches compete to offer the most effective solution. In many cases, these problems require large amounts of computational resource. Typically, different algorithms (model and solver choices) are better suited for different problem instances and new potential approaches are continuously developed. Identifying the most suitable algorithm for a particular problem instance has the potential to provide significant efficiency gains.

Machine learning (ML) has emerged as a powerful tool for automating algorithm selection, effectively learning to predict which algorithm is most likely to perform well on a given problem instance. By analysing the features of various problem instances and the corresponding performance of different algorithms, ML models can be trained to recommend the most suitable algorithm for a specific problem [23, 21]. This approach has proven highly effective,

often leading to significant improvements in computational efficiency and overall problem-solving capability (e.g., [45, 46, 5]). Despite the success of ML-based algorithm selection techniques, there is a notable drawback: the high computational cost associated with running candidate algorithms on a large and representative set of training instances. In order to generate a robust model, it is necessary to evaluate the performance of each algorithm on numerous problem instances. However, this process can be time-consuming and resource-intensive, limiting the scalability and practicality of current algorithm selection methods.

Herein we propose to select instances for training AS models iteratively. This setting is called Active Learning (AL) [8], and is popular for cost-effective training in ML. We focus on AS scenarios that aim at optimising the runtime of the predicted algorithm on a specific instance, a common setting in CP and SAT domains. In these scenarios, each run of an algorithm on a problem instance is limited by a typically large cutoff time. Full information about performance data of timeout cases is expensive to collect but not necessarily useful.

In a typical AL scenario, the *labelling cost* (in our context the cost of running an algorithm to solve a selected instance) is assumed to be uniform. Our situation is further complicated by the fact that, within the cutoff, there is typically significant variance in the cost of running an algorithm. Therefore, we propose two strategies within the active learning setting to reduce labelling cost, thus improving learning efficiency. We evaluate the proposed strategies in combination with two methods for selecting new instances in the active learning setting: an uncertainty-based and a naive random selection method (see Section 3.1 for details). Experimental results on six scenarios from ASLib [6], the standard benchmarking library for algorithm selection, are presented, showing the effectiveness of applying active learning setting to the AS context: in most cases, we can achieve 100% of the predictive power of an AS method that uses all of the training data, while requiring less than 60% of total labelling cost. This is further reduced to as low as 10% of the labelling cost in some scenarios, thanks to a combination of using a timeout predictor and dynamic timeouts.

The primary contributions of our work are:

1. Novel use of timeout predictors and dynamic timeouts for cost-effective algorithm selection.
2. A thorough empirical evaluation across six benchmarks from ASLib, validating the effectiveness of the proposed timeout aware active learning approach.

2 Background

This section provides background for our work, covering three key areas: Automated Algorithm Selection, where the primary challenge is to optimally select an algorithm from a portfolio based on specific instance characteristics and desired performance outcomes; ASLib, a pivotal resource for benchmarking algorithm selection; and Active Learning, a method for organising the training process by strategically choosing informative data points to label.

2.1 Automated Algorithm Selection

The automated algorithm selection (AS) problem can be defined as follows. Given a portfolio of n algorithms $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ with *complementary* strengths, an instance distribution $\mathcal{D}_\mathcal{I}$ where each instance $i \sim \mathcal{D}_\mathcal{I}$ is described as a feature vector $v(i)$, a metric $c(i, a_k)$ that measures the performance of algorithm $a_k \in \mathcal{A}$ on instance i, the AS problem involves building an automated selector $f(v(i))$ to select the best algorithm $a \in \mathcal{A}$ for an instance i. More formally, we want to find f such that $E_{i \sim \mathcal{D}_\mathcal{I}}[c(i, f(v(i))]$ is optimised. There is sometimes an extra cost associated with the extraction of instance features, which must be taken into account in our optimisation objective. In practice, we are often given in advance

a set of training instances, their feature vectors, the cost of extracting such features, and the performance of each algorithm in the portfolio on each given instances. This data is used for training a machine learning model to predict the best algorithm for an unseen instance.

The first AS problem was described in 1976 by Rice [35]. Over the last two decades, several AS approaches have been proposed and the applications of such techniques have provided a significant boost in state-of-the-art performance across a wide range of computational areas, such as Constraint Programming [32, 39], propositional satisfiability (SAT) solving [45, 46], AI planning [42, 36, 37], and combinatorial optimisation [22]. A wide range of machine learning-based techniques were adopted in those AS approaches. Some examples include empirical performance models [45], k-nearest neighbours [9], clustering-based methods [20, 5], and cost-sensitive pairwise classification approaches [46]. In addition to building a machine learning model to predict the best performing algorithm, modern AS systems often adopt an extra component called a *pre-solving schedule* [46, 17, 25, 15], a static schedule of algorithms run for a small amount of time before (expensive) feature extraction and algorithm selection are conducted. Another related approach is algorithm scheduling, where instead of selecting a single algorithm for given instance, we build a *schedule* of algorithms [3, 4, 26]. For detailed overviews of AS approaches and their applications, we refer to [21, 23].

In constraint programming, alternative algorithms typically arise from different models for the same problem or from the use of various solvers. Alternative models can be created manually or generated using automated modelling tools such as Conjure [2]. Tools like Savile Row [31, 13] and Minizinc [30] can automatically target multiple alternative solvers.

2.2 The Algorithm Selection Library (ASLib)

The Algorithm Selection Library [6] (ASLib) is a widely-used benchmarking library for automated algorithm selection. It provides a standard format for representing algorithm selection scenarios across a wide range of algorithms and problems, currently consisting of 44 datasets from multiple application domains, including SAT, Quantified Boolean Formula (QBF), Maximum Satisfiability (MAX-SAT), Constraint Satisfaction Problems, Answer Set Programming (ASP), and combinatorial optimisation problems. In these scenarios, the performance metric is either the algorithm running time (for solving a given instance) or the solution quality obtained within a time limit.

The process of data labelling, i.e. obtaining the target output labels for all training data points, is a critical task in machine learning as it is an essential component for building effective models. However, this process is not free of cost, and it can be a time-intensive endeavour, often requiring more resources than training the machine learning model itself. In automated algorithm selection, the cost of collecting performance data can be substantial. As an example, several scenarios in ASLib require more than 100 CPU days to collect the full performance data on the given instance sets (some may require up to 3 CPU years). Our hypothesis is that we can significantly reduce this cost by only collecting partial information about the performance data without sacrificing algorithm selection quality.

2.3 Active Learning for querying informative instances

Active learning is a methodology that maintains machine learning accuracy with fewer labelled data points by allowing incremental labelling of training data [38, 19]. This iterative process comprises training a model on a small set of labelled data, using the model to identify and query the most uncertain unlabelled data points, obtaining labels for these queried points, and then retraining the model with the newly labelled data.

Several methods exist for querying in active learning [47, 44, 10, 33]. Herein, we use pool-based sampling (we maintain a list of candidate training data points that can be queried) and two methods for querying from this pool: uncertainty-based and random instance selection. These are widely regarded as effective methods for active learning, particularly suitable for our context. See Appendix B for more about the behaviour of alternative query methods.

3 Frugal Algorithm Selection

In this section, we first explain the underlying AS model. We then describe how to use active learning to select a subset of instances for labelling during the training (Section 3.1) and the two additional strategies we propose, namely *timeout predictors* (Section 3.2) and *dynamic timeout* (Section 3.3), to reduce labelling cost during the instance selection process.

The AS model used in our experiment follows a pairwise classification approach, as it has been shown to be effective for several AS scenarios (e.g. [46]). The approach is a collection of binary classifiers, each designed to compare a pair of algorithms to determine which one is faster for a given instance. The algorithm with the highest number of votes across all classifiers is chosen as the best option. Following previous work [46], we use a random forest for each classifier. This setup is also advantageous for our goal of minimising resource usage, as it allows selective training of classifiers on specific subsets of instances. The sequence diagram overview of our approach can be found in Appendix A.

Passive learning. an AS model trained using the entire training set. This is the baseline for investigating the effectiveness of our frugal AS methods: we want the frugal AS to achieve the same performance as this passive learning model, while using a significantly less amount of training data.

Frugal methods. We explore three configuration options, each offering two alternatives, to create a range of strategies for frugal algorithm selection. The first configuration option is instance selection, which involves comparing uncertainty-based selection (focusing on potentially informative instances) with random selection. The second configuration option is whether we use timeout predictors and the third configuration option is whether we use dynamic timeouts. In the rest of this section we explain these three configuration options. In Section 4 we empirically evaluate all 8 combinations of these configuration settings.

3.1 Selecting Instances: Uncertainty-based vs random

In our frugal algorithm selection methods, we begin by training all machine learning classifiers on a small number of randomly selected instances. The remaining instances in the training set are kept in a pool of candidates. For each classifier we maintain a separate pool of candidates: this allows us to run an instance on a subset of the algorithms instead of necessarily running it on all algorithm options. This flexibility can be particularly useful when some algorithms tend to timeout very often and hence take up a lot of resources unnecessarily.

At each step of our algorithm, we select N samples from the available unlabelled set of instances. One option for the selection process is employing an uncertainty-based query strategy. Based on an "informativeness" measure, this strategy aims to prioritise the instances that are most likely to yield valuable insights when annotated. We perform uncertainty-based querying by using the predictive model we have partially trained so far. We predict a class for each candidate instance in the pool. The machine learning predictor returns a confidence level in addition to a class prediction. We then query the instances where the predictor is

least confident. Uncertainty-based querying is based on the premise that, by focusing on the data points where the model's predictions are least confident, the model is expected to learn from the most uncertain data points. We also allow different machine learning models to make different numbers of queries, based on the confidence levels. For each predictive model we create a table of requested data points and the associated confidence level. We then combine these tables into a single table, sort the table by confidence level and select top N requests. This approach enables predictive models with a high level of uncertainty to query more instances in comparison to those with very low levels of uncertainty.

Uncertainty-based querying can be expensive because it requires feature extraction for all instances in the candidate pool at the start of the procedure. It also only considers informativeness without taking labelling cost into account. To evaluate whether uncertainty-based querying is effective in our work, we compare it with a random query order.

3.2 Timeout Predictor

Instances that time out with a given algorithm are particularly costly in automated algorithm selection. This is partly because if an instance cannot be solved by two algorithms within the timeout, we spend considerable time running these algorithms but gain no new information. Furthermore, when one algorithm solves an instance quickly and another times out, we gain no additional information by allowing the slower algorithm to run to completion. The binary classifiers are provided information only about which algorithm is faster.

We enhance our base machine learning architecture of binary classifiers for all algorithm pairs with dedicated timeout predictors: additional random forest classifiers, one per algorithm, whose task is to predict whether an unseen instance will time out for a specific algorithm. The hypothesis is that training a timeout predictor is a simpler learning task, and this classifier can be trained without requiring additional data. We adjust our voting mechanism to take the timeout predictors into account. If an algorithm is predicted to time out, we exclude it from the options before calculating the votes using the pairwise predictors. An exception is in instances where all algorithms are predicted to time out; in such cases, we do not eliminate any of the options and continue to use pairwise predictors for the entire set. The timeout predictors are also used during the instance selection process: any pair of algorithms and instances predicted to be timeout will be pushed to the end of selection queue, ensuring that we focus on instances that are more likely to provide useful information.

3.3 Dynamic Timeout

The dynamic timeout strategy begins with an initially defined timeout period and incrementally increases it up to a maximum of one hour. After the initial training phase in active learning, the algorithms selected for querying are executed on the selected data within the current time limit. An algorithm that fails to solve the example within this specified time is classified as a timeout for the active time limit. This approach is intended to minimise labelling costs by initially running instances with a short time limit. Hence, resources are not wasted on instances that both algorithms are likely to fail to solve in the early stages. In single timeout cases, where one of the two algorithms can solve the instance, samples can be labelled at a lower cost. The condition for increasing the timeout is determined based on the performance of the model predictions on the validation set. If the model predictions reach a plateau on the validation set, we increase the timeout at the specified rate.

When timeout predictors and dynamic timeouts are used in combination we train the timeout predictors with respect to the particular timeout value at a given moment.

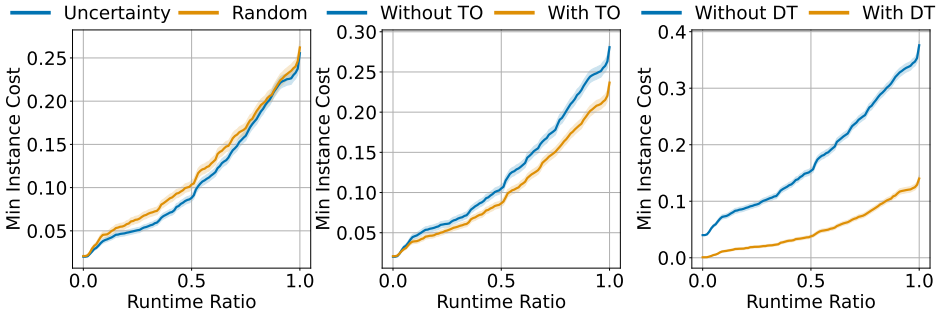

Figure 1 Results aggregated by configuration option. Instance selection (random vs uncertainty-based) does not make a big difference, timeout predictor (TO) improves runtime ratio slightly, dynamic timeout (DT) improves runtime ratio significantly.

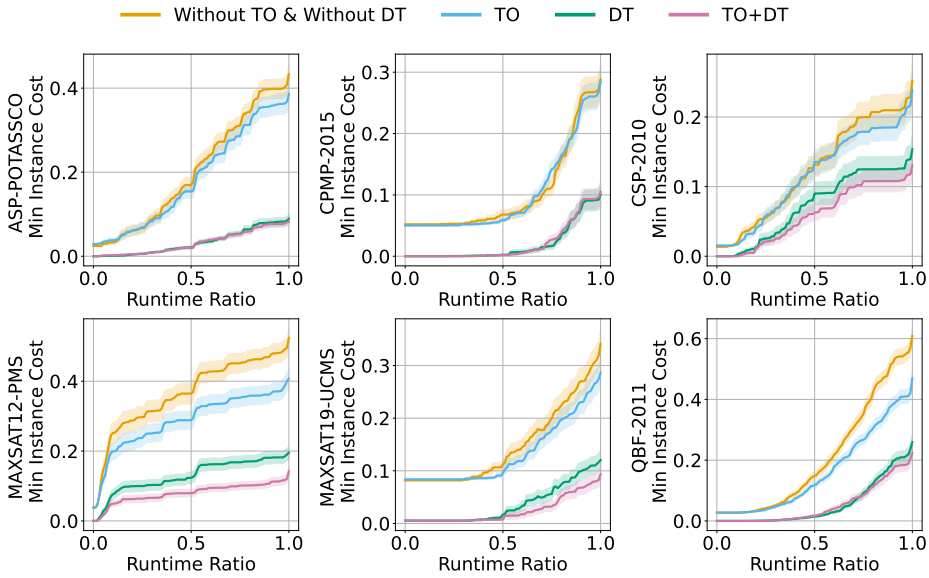

Figure 2 Performance of different timeout configurations (aggregated by instance selection). Notably, the combination of timeout predictor and dynamic timeouts (TO+DT) and the standalone dynamic timeout option exhibit significantly better performance.

4 Experimental Results

We evaluate the performance of all eight configurations of the frugal algorithm selection methods using six datasets from ASLib, chosen for their diverse characteristics. These datasets include various problem-solving domains: one from Answer Set Programming (ASP), two from Constraint Programming (CP), two from propositional satisfiability (SAT), and one from Quantified Boolean Formula (QBF) solving. The datasets vary significantly in complexity and size, with between 2 to 11 algorithm options, 22 to 138 features, and 527 to 2024 instances. Appendix E presents detailed descriptive statistics of the selected datasets.

Experimental setup. To evaluate our methodology we split each dataset into three subsets: training, validation, and test. We allocate 10% of the instances to the test set and perform 10-fold cross-validation to ensure thorough evaluation, running each fold five times with

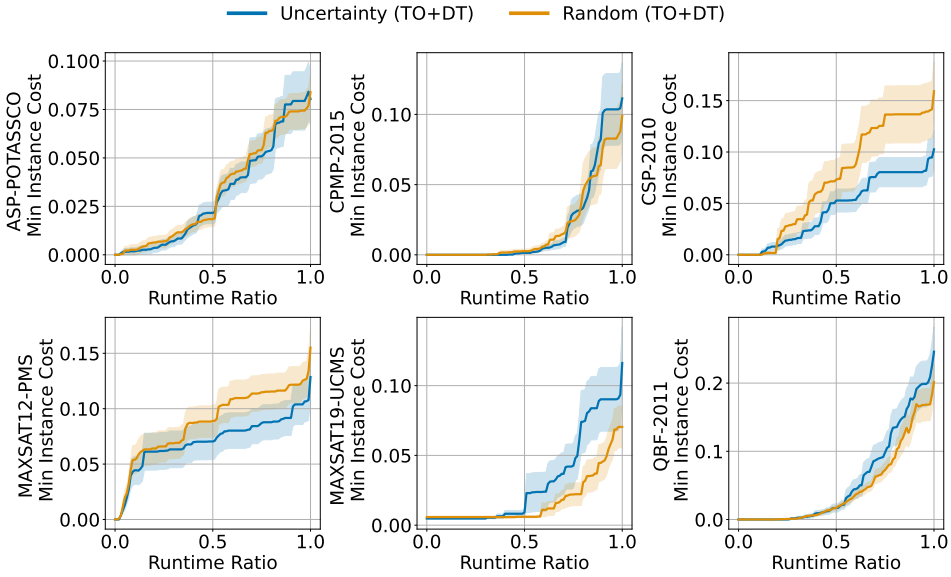

Figure 3 TO+DT options (best in Figure 2) and disaggregating by approach. No clear winner.

different random seeds. An additional 10% of the training set is used as the validation set, used to decide when to increase the timeout (discussed in Section 3.3). At each step, we label 1% of the training data. Appendix D provides full details of the experimental setup.

We present our results in a series of plots designed to compare the configurations at different aggregation levels. All have the same structure. The horizontal axis is the ratio of the performance of passive learning to that of the frugal method, where performance is measured as the total runtime of the predicted algorithms on all test instances. This metric serves as a proxy for predictive performance, indicating how closely the frugal method approaches the benchmark established by passive learning. The vertical axis is the minimum amount of training data (as a ratio of the entire training set) required to achieve the performance indicated on the horizontal axis. The representation highlights the efficiency of the frugal method in terms of data utilisation. We plot the mean and a ribbon showing standard error.

All source code, data, experimental results and the appendix are available at `https://github.com/stacs-cp/CP2024-Frugal`

Key findings. Even the worst configuration of frugal algorithm selection is able to reduce the labelling cost without sacrificing predictive performance relative to passive learning (Figure 2). In several cases training effort is reduced to 10% of the labelling cost of passive learning. It is clear that our frugal approaches are able to reduce the training cost independent of configuration.

Figure 1 presents an overview of the entire set of experiments. The results are aggregated one configuration option at a time, combining results of all configurations that share, for example, Uncertainty as the instance selection method in the first plot. Overall, instance selection strategy does not make a big difference, using timeout predictors (TO) improves performance slightly, and using dynamic timeouts (DT) improves performance significantly.

Since instance selection strategy does not make a big difference and we observe an interesting interaction between TO and DT, we aggregate over instance selection strategy and plot 4 options in Figure 2. Using TO slightly improves performance, while DT improves performance significantly. Moreover the best configurations use TO and DT together.

In Figure 3 we focus on the configurations that include TO+DT and compare the effect of instance selection of the best configuration setting found in Figure 2. We further validate that there is not a clear winner among uncertainty-based instance selection and random instance selection. This finding is consistent across all combinations of strategies, as detailed in Appendix C, which includes raw data plots for all eight configurations.

The observation that uncertainty-based instance selection is not better than random may be unexpected, but selecting instances purely based on informativeness (via prediction uncertainty) does not take the cost of running an instance on a particular algorithm into account. Where there is a uniform cost across all candidates, uncertainty-based selection may perform better than random. In our setting, however, sample cost varies. Hence, we wish to maximise how much information is gained per time spent. Therefore having explicit timeout predictors and a dynamic timeout strategy makes a more significant contribution.

5 Related work

Selecting a representative subset of benchmark instances from a large pool for a reliable and cost-effective comparison of algorithms has been investigated across different domains, including SAT [18, 27, 14], CP [28], combinatorial optimisation [29], evolutionary computation [7], and machine learning [34]. While a majority focuses on selecting a subset of instances in a static setting (i.e., all instances are chosen at once), some recent work has proposed selecting instances iteratively. Matricon et al. [28] present a statistical-based method to incrementally select instances for comparing two solvers. Fuchs et al. [14] propose an active learning-based approach for cost-effective benchmark instance selection. The key difference between the approaches above and our work is that they focus on identifying the algorithm with the best *overall performance* across a given problem instance distribution, while our work focuses on algorithm selection, where the aim is to predict the best *instance-specific* algorithm.

The closest work to ours is Volpato and Song [43], where three commonly-used active learning techniques were evaluated in an AS scenario for SAT[1]. However, they did not consider the significantly varying labelling costs among algorithms and instances, a common characteristic of SAT scenarios. Consequently, the effectiveness of active learning for instance selection was reported based on the percentage of labelled data being saved only.

Although the majority of active learning techniques assume uniform labelling cost, there is work on non-uniform labelling cost settings (e.g., [38, 40, 41]). A common approach is to predict the labelling cost and to strive for a balance between informativeness and the predicted cost of a new data point. We adopt a similar technique in our work, where timeout predictors are used for identifying costly (unlabelled) data points. It can be considered a "softened" version of the cost estimation approach, as predicting the precise runtime of algorithms in the domain of combinatorial optimisation is often difficult: most AS techniques focus on learning the ranking among algorithms instead of trying to predict runtime directly [16].

Effective algorithm selection relies on the availability of a representative set of instances for a problem. When such instances are not available, automated methods can generate discriminating instances [1, 11]. These methods aim to find instances that one algorithm can solve quickly while another algorithm struggles, and vice versa.

[1] These three methods result in the same set of selected instances when the prediction model is binary classification as in our work – see Appendix B.

6 Conclusion

We have proposed and evaluated several approaches to *frugal* algorithm selection, an active learning method that reduces labelling cost by using only a subset of the training data, together with a dynamic timeout strategy that uses incomplete information about the performance of algorithms in the portfolio. Our results confirm the utility of the proposed approach and our analysis offers insights into the contribution of each of its components. Interestingly, the standard active learning data selection technique contributes little to performance, while our proposed dynamic timeout mechanism results in significant savings.

In future, we plan to incorporate enhancement techniques in AS into the proposed active learning framework, including the use of a pre-solving schedule [45] and cost-sensitive pairwise classification AS models [46, 25]. Other important avenues include investigating the impact of hyper-parameter tuning in the active learning setting, and developing an early-stopping mechanism to terminate the learning process once diminishing returns are observed.

References

1 Özgür Akgün, Nguyen Dang, Ian Miguel, András Z Salamon, Patrick Spracklen, and Christopher Stone. Discriminating instance generation from abstract specifications: A case study with cp and mip. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research: 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21–24, 2020, Proceedings 17*, pages 41–51. Springer, 2020.

2 Özgür Akgün, Alan M Frisch, Ian P Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Conjure: Automatic generation of constraint models from problem specifications. *Artificial Intelligence*, 310:103751, 2022.

3 Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. Sunny: a lazy portfolio approach for constraint solving. *Theory and Practice of Logic Programming*, 14(4-5):509–524, 2014.

4 Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. Sunny-cp: a sequential cp portfolio solver. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1861–1867, 2015.

5 Carlos Ansótegui, Joel Gabas, Yuri Malitsky, and Meinolf Sellmann. Maxsat by improved instance-specific algorithm configuration. *Artificial Intelligence*, 235:26–39, 2016.

6 Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, 2016. doi:10.1016/j.artint.2016.04.003.

7 Gjorgjina Cenikj, Ryan Dieter Lang, Andries Petrus Engelbrecht, Carola Doerr, Peter Korošec, and Tome Eftimov. Selector: selecting a representative benchmark suite for reproducible statistical comparison. In *Proceedings of The Genetic and Evolutionary Computation Conference*, pages 620–629, 2022.

8 David Cohn. *Active Learning*, pages 10–14. Springer US, Boston, MA, 2010. doi:10.1007/978-0-387-30164-8_6.

9 Marco Collautti, Yuri Malitsky, Deepak Mehta, and Barry O'Sullivan. Snnap: Solver-based nearest neighbor for algorithm portfolios. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III 13*, pages 435–450. Springer, 2013.

10 Nguyen Viet Cuong, Wee Sun Lee, Nan Ye, Kian Ming A. Chai, and Hai Leong Chieu. Active learning for probabilistic hypotheses using the maximum gibbs error criterion. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, pages 1457–1465, Red Hook, NY, USA, 2013. Curran Associates Inc.

11 Nguyen Dang, Özgür Akgün, Joan Espasa, Ian Miguel, and Peter Nightingale. A framework for generating informative benchmark instances. *arXiv preprint arXiv:2205.14753*, 2022.

12 Tivadar Danka and Péter Horváth. modal: A modular active learning framework for python. *CoRR*, abs/1805.00979, 2018. arXiv:1805.00979.

13 Ewan Davidson, Özgür Akgün, Joan Espasa, and Peter Nightingale. Effective encodings of constraint programming models to smt. In *Principles and Practice of Constraint Programming: 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7–11, 2020, Proceedings 26*, pages 143–159. Springer, 2020.

14 Tobias Fuchs, Jakob Bach, and Markus Iser. Active learning for sat solver benchmarking. In *Tools and Algorithms for the Construction and Analysis of Systems: 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22–27, 2023, Proceedings, Part I*, pages 407–425. Springer, 2023.

15 François Gonard, Marc Schoenauer, and Michèle Sebag. Algorithm selector and prescheduler in the icon challenge. *Bioinspired heuristics for optimization*, pages 203–219, 2019.

16 Jonas Hanselle, Alexander Tornede, Marcel Wever, and Eyke Hüllermeier. Hybrid ranking and regression for algorithm selection. In *German Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 59–72. Springer, 2020.

17 Holger Hoos, Marius Lindauer, and Torsten Schaub. claspfolio 2: Advances in algorithm selection for answer set programming. *Theory and Practice of Logic Programming*, 14(4-5):569–585, 2014.

18 Holger H Hoos, Benjamin Kaufmann, Torsten Schaub, and Marius Schneider. Robust benchmark set selection for boolean constraint solvers. In *Learning and Intelligent Optimization: 7th International Conference, LION 7, Catania, Italy, January 7-11, 2013, Revised Selected Papers 7*, pages 138–152. Springer, 2013.

19 Sheng-Jun Huang, Rong Jin, and Zhi-Hua Zhou. Active learning by querying informative and representative examples. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(10):1936–1949, 2014. doi:10.1109/TPAMI.2014.2307881.

20 Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, and Kevin Tierney. Isac–instance-specific algorithm configuration. In *ECAI 2010*, pages 751–756. IOS Press, 2010.

21 Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, 27(1):3–45, 2019.

22 Pascal Kerschke, Lars Kotthoff, Jakob Bossek, Holger H Hoos, and Heike Trautmann. Leveraging tsp solver complementarity through machine learning. *Evolutionary computation*, 26(4):597–620, 2018.

23 Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. *Data mining and constraint programming: Foundations of a cross-disciplinary approach*, pages 149–190, 2016.

24 Erdem Kuş. stacs-cp/CP2024-Frugal. Other, version 1.1. (visited on 2024-08-20). URL: https://doi.org/10.5281/zenodo.13294528.

25 Marius Lindauer, Holger H Hoos, Frank Hutter, and Torsten Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

26 Tong Liu, Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. sunny-as2: Enhancing sunny for algorithm selection. *Journal of Artificial Intelligence Research*, 72:329–376, 2021.

27 Norbert Manthey and Sibylle Möhle. Better evaluations by analyzing benchmark structure. *Proc. PoS*, 2016.

28 Théo Matricon, Marie Anastacio, Nathanaël Fijalkow, Laurent Simon, and Holger H Hoos. Statistical comparison of algorithm performance through instance selection. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

29 Mustafa Mısır. Benchmark set reduction for cheap empirical algorithmic studies. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 871–877. IEEE, 2021.

30 Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack. Minizinc: Towards a standard cp modelling language. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–543. Springer, 2007.

31 Peter Nightingale, Özgür Akgün, Ian P Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. Automatically improving constraint models in savile row. *Artificial Intelligence*, 251:35–61, 2017.

32 Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent, and Barry O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Irish conference on artificial intelligence and cognitive science*, pages 210–216, 2008.

33 Mijung Park and Jonathan W. Pillow. Bayesian active learning with localized priors for fast receptive field characterization. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pages 2348–2356, Red Hook, NY, USA, 2012. Curran Associates Inc.

34 João Luiz Junho Pereira, Kate Smith-Miles, Mario Andrés Muñoz, and Ana Carolina Lorena. Optimal selection of benchmarking datasets for unbiased machine learning algorithm evaluation. *Data Mining and Knowledge Discovery*, 38(2):461–500, 2024.

35 John R Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

36 Mattia Rizzini, Chris Fawcett, Mauro Vallati, Alfonso E Gerevini, and Holger H Hoos. Portfolio methods for optimal planning: an empirical analysis. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 494–501. IEEE, 2015.

37 Mattia Rizzini, Chris Fawcett, Mauro Vallati, Alfonso E Gerevini, and Holger H Hoos. Static and dynamic portfolio methods for optimal planning: An empirical analysis. *International Journal on Artificial Intelligence Tools*, 26(01):1760006, 2017.

38 Burr Settles. Active learning literature survey, 2009. URL: `https://api.semanticscholar.org/CorpusID:324600`.

39 Patrick Spracklen, Nguyen Dang, Özgür Akgün, and Ian Miguel. Automated streamliner portfolios for constraint satisfaction problems. *Artificial Intelligence*, 319:103915, 2023.

40 Katrin Tomanek and Udo Hahn. A comparison of models for cost-sensitive active learning. In *Coling 2010: Posters*, pages 1247–1255, 2010.

41 Yu-Lin Tsou and Hsuan-Tien Lin. Annotation cost-sensitive active learning by tree sampling. *Machine Learning*, 108(5):785–807, 2019.

42 Mauro Vallati, Lukáš Chrpa, and Diane Kitchin. Asap: an automatic algorithm selection approach for planning. *International Journal on Artificial Intelligence Tools*, 23(06):1460032, 2014.

43 Riccardo Volpato and Guangyan Song. Active learning to optimise time-expensive algorithm selection. *arXiv preprint arXiv:1909.03261*, 2019.

44 Liantao Wang, Xuelei Hu, Bo Yuan, and Jianfeng Lu. Active learning via query synthesis and nearest neighbour search. *Neurocomputing*, 147:426–434, 2015. Advances in Self-Organizing Maps Subtitle of the special issue: Selected Papers from the Workshop on Self-Organizing Maps 2012 (WSOM 2012). `doi:10.1016/j.neucom.2014.06.042`.

45 Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

46 Lin Xu, Frank Hutter, Jonathan Shen, Holger H Hoos, and Kevin Leyton-Brown. Satzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge*, pages 57–58, 2012.

47 Jingbo Zhu, Huizhen Wang, Benjamin K. Tsou, and Matthew Ma. Active learning with sampling by uncertainty and density for data annotations. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1323–1331, 2010. `doi:10.1109/TASL.2009.2033421`.

A Appendix A: Sequence Diagram Overview

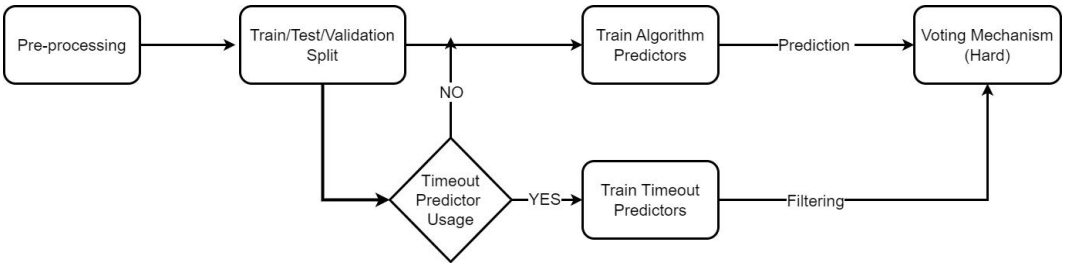

Figure 4 The diagram depicts the process of passive learning, which includes preprocessing, splitting the dataset, and training multiple binary Random Forest (RF) algorithm classifiers and timeout classifiers, followed by a hard voting mechanism to finalize predictions. Timeout predictors are used for filtering algorithm predictors in the voting mechanism. All models are included in the voting mechanism where the timeout predictor configuration is not applied.

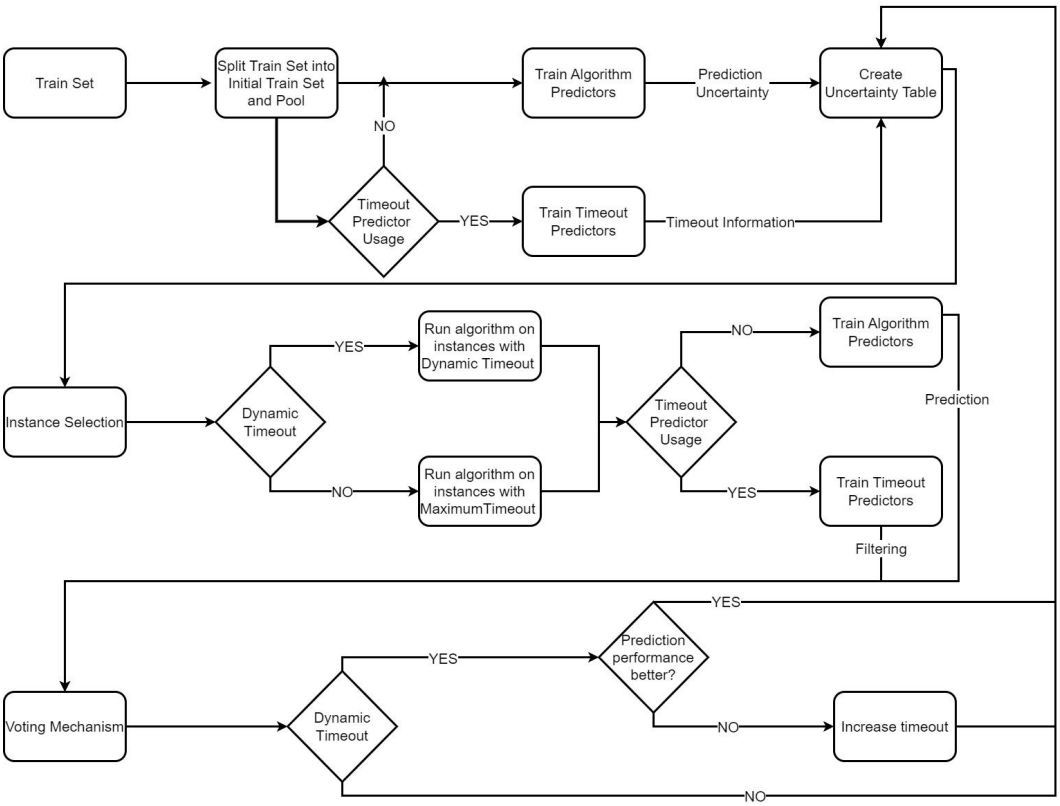

Figure 5 The diagram illustrates the steps in the proposed approach. It starts with splitting the training set into an initial training set and a pool for each model. Multiple binary Random Forest (RF) algorithm and timeout classifiers are then trained. The instance selection process involves identifying the most uncertain data points across all models and excluding instances predicted to time out by the timeout predictors. A dynamic timeout is applied during the labeling process, which is increased when there is no performance enhancement on the validation set. After labeling, the iterative process begins again, continuously refining the models.

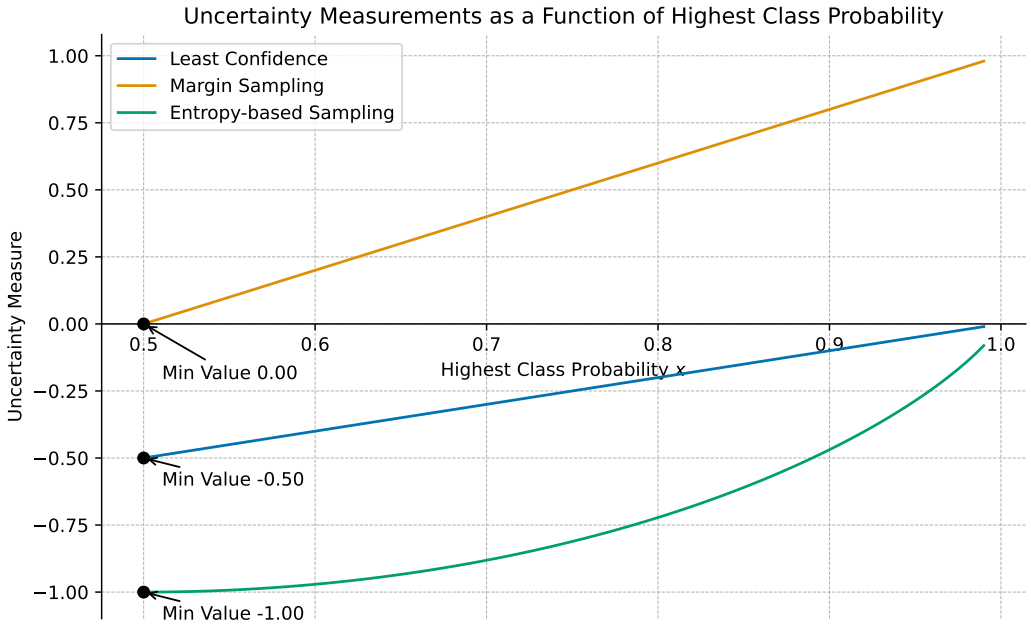

Figure 6 Uncertainty measurements as a function of the highest class probability. The red curve represents the Least Confidence uncertainty (LC) calculated as $LC = x - 1$, the green curve denotes Margin Sampling (MS) using the formula $MS = x - (1 - x)$, and the blue curve illustrates the Entropy-based method ($H(x) = -[x \log_2(x) + (1-x) \log_2(1-x)]$). Critical minimum values for each method are marked with black circles and annotated to emphasise the points where the uncertainty function is minimised.

B Appendix B: Analysis of Uncertainty Measurement Behaviours in Active Learning for Binary Classification

There are three main approaches for uncertainty sampling in active learning. However, in a binary classification setting (which is what we use) these approaches perform identically to each other. We explain the different approaches here. Figure 6 shows the behaviour of these uncertainty sampling methods graphically.

We implement "Least Confidence" in our approach.

- *Least Confidence*: for a given input x and an output label $\hat{y}$, we can measure the posterior probability $P(\hat{y}|x;\theta)$ of observing $\hat{y}$ given x via the current model (parameterised by θ). The Least Confidence method selects data points x^* with the smallest maximum posterior probability across all labels:

$$x^* = \underset{x}{\operatorname{argmin}} \max_{\hat{y}} P(\hat{y}|x;\theta) \tag{1}$$

- *Margin-based*: this approach takes the two highest posterior probability values for each input data point x and calculates their difference. The smaller the difference, the less certain the model is about its prediction and vice versa. More formally, let $\hat{y_1}$ and $\hat{y_2}$ the output labels with the highest and second-highest posterior probabilities for a given input x, respectively, the queried points x^* are chosen as:

$$x^* = \underset{x}{\operatorname{argmin}}\, P(\hat{y_1}|x;\theta) - P(\hat{y_2}|x;\theta) \tag{2}$$

- *Entropy-based*: this approach takes into account the posterior probability values across *all* output classes. The idea is to select the data points x^* where there is a high entropy among the predicted output labels:

$$x^* = \underset{x}{\text{argmax}} - \sum_i P(\hat{y}|x;\theta) \log P(\hat{y}|x;\theta) \tag{3}$$

C Appendix C: Performance of 8 individual configurations

Figure 7 illustrates a side-by-side comparison of the following eight active learning strategies in binary classification without aggregation across configurations:

- Uncertainty Sampling without Timeout Predictor & without Dynamic Timeout (NO TO & NO DT)
- Uncertainty Sampling with Timeout Predictor (TO)
- Uncertainty Sampling with Dynamic Timeout (DT)
- Uncertainty Sampling with Timeout Predictor and Dynamic Timeout (TO+DT)
- Random Sampling without Timeout Predictor & without Dynamic Timeout (NO TO & NO DT)
- Random Sampling with Timeout Predictor (TO)
- Random Sampling with Dynamic Timeout (DT)
- Random Sampling with Timeout Predictor and Dynamic Timeout (TO+DT)

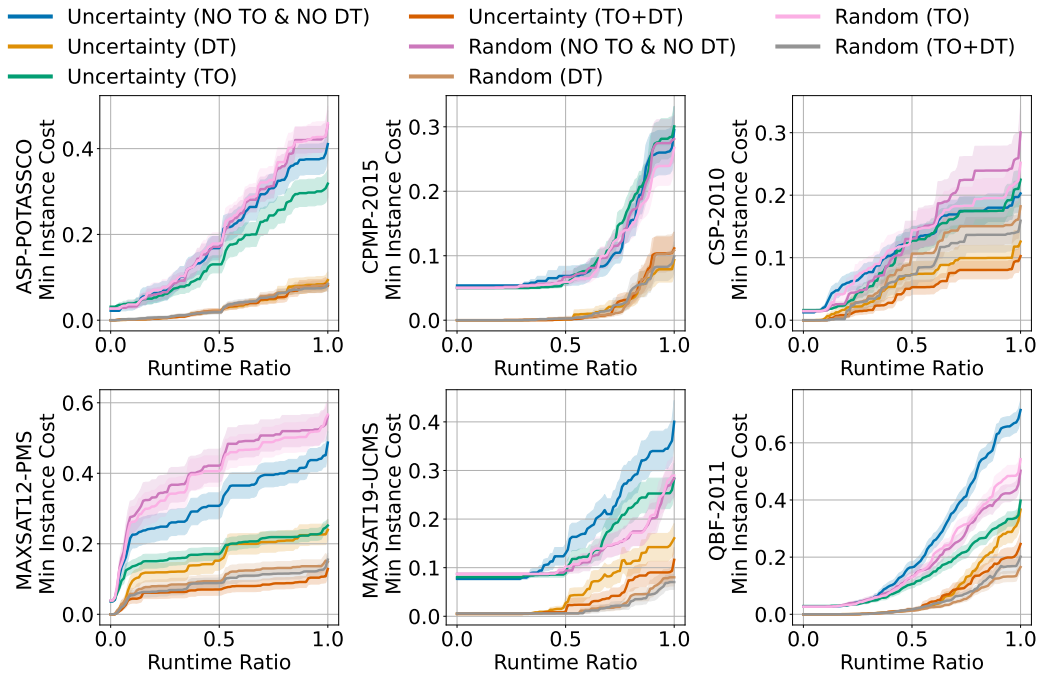

Figure 7 Comparison of performance across eight configurations as described in the paper. Each configuration was normalised according to the passive learning prediction performance ratio.

D Appendix D: Experimental Setup

This study used a Random Forest classifier configured with 100 estimators and the Gini impurity measure to determine the best splits. Each tree is limited to using up to the square root of the number of features, and the depth of the decision trees is practically unlimited (with a maximum depth set to 2^{31}). Nodes require at least two samples before splitting, and bootstrapping is enabled for sampling data when building each decision tree. These settings were determined through experimentation in the passive learning setup and were consistently used throughout the study.

We also addressed missing data by removing features where more than 20% of the instances had missing values and applied a median imputer to fill the remaining gaps.

We employed a cross-validation approach with 10 splits to validate the robustness of our study. To ensure reproducibility, we used 5 distinct seeds (7, 42, 99, 123, 12345) across our experiments, ensuring consistent generalization across multiple runs.

To determine when to increase the timeout in configurations where dynamic timeout is used, 10% of the training set was allocated as the validation set. Throughout the experiments, timeout values were scaled by a factor of 10, following the PAR10 measure.

Additional Parameters and Configurations.

Timeout Predictor Usage: This parameter determines whether the timeout predictor is used on the system.

Timeout Limit: Sets the initial time for the dynamic timeout. We used an initial timeout of 100 seconds when employing dynamic timeout, and a fixed timeout of 3600 seconds when not using dynamic timeout.

Timeout Increase Rate: Adjusts the dynamic timeout when there is no improvement in prediction performance on the validation set. We set this rate to increase by 100 seconds when no improvement was observed.

Initial Train Size: Determines the size of the initial training set for uncertainty selection. The initial training set was created by randomly selecting 20 data points from the overall training set.

Query Size: Refers to the percentage of the dataset queried in each iteration. We set this to 1%, meaning 1% of the total pool of candidates was queried in each iteration of our experiments.

For active learning, we utilized the modAL framework [12], which facilitated the implementation of uncertainty sampling and other active learning strategies in our experiments.

E Appendix E: Description Table of Selected Datasets

Table 1 shows key information about the datasets used in this study. It includes the time it took for the algorithms to run, the Virtual Best Solver(VBS) representing the best algorithm for each problem, and the Single Best Solver (SBS) as the best overall algorithm. While VBS is the hypothetical best, SBS serves as a benchmark for comparison against other algorithms.

Table 1 Descriptive statistics of selected datasets. Times rounded to the nearest whole number.

Dataset	Instances	Algorithms	Features	Total Time	VBS	SBS
ASP-POTASSCO	1294	11	138	2,085h	8h	112h
CPMP-2015	527	4	22	682h	33h	134h
CSP-2010	2024	2	86	435h	49h	82h
MAXSAT12-PMS	876	6	37	1,472h	8h	85h
MAXSAT19-UCMS	572	7	54	545h	20h	52h
QBF-2011	1368	5	46	352h	28h	300h

F Appendix F: Timeout (TO) Configuration Impact on Passive Learning

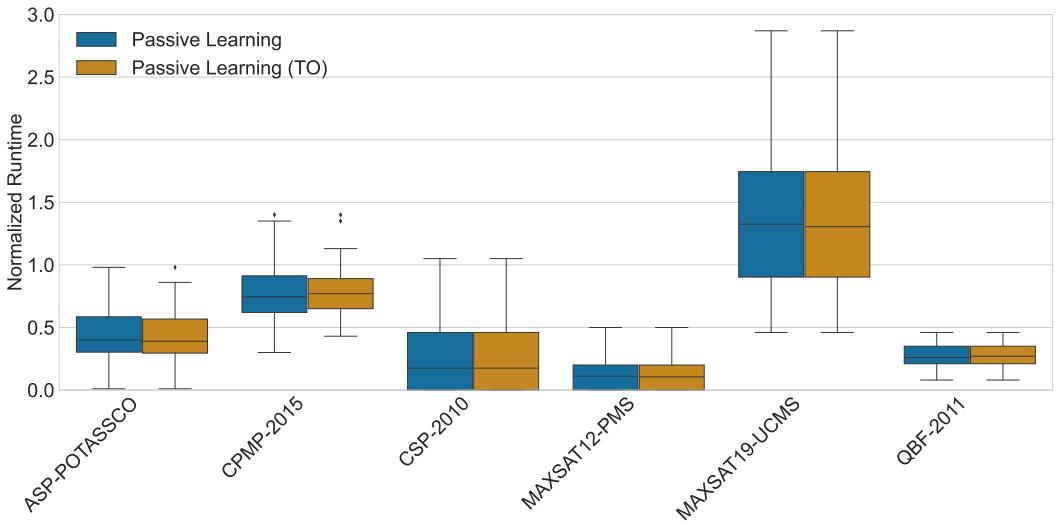

Figure 8 Comparison of Timeout (TO) Configuration Impact on Passive Learning: The graph illustrates that implementing the TO configuration in passive learning on the test set does not significantly enhance performance, yet importantly, it does not compromise prediction accuracy either.