

Statistical Comparison of Algorithm Performance Through Instance Selection

Théo Matricon, Mario Anastacio, Nathanaël Fijalkow, Laurent Simon, Holger H Hoos

▶ **To cite this version:**

Théo Matricon, Mario Anastacio, Nathanaël Fijalkow, Laurent Simon, Holger H Hoos. Statistical Comparison of Algorithm Performance Through Instance Selection. International Conference on Principles and Practice of Constraint Programming (CP 2021), Oct 2021, Montpellier, France. 10.4230/LIPIcs.CP.2021.43 . hal-03800492

HAL Id: hal-03800492
https://cnrs.hal.science/hal-03800492

Submitted on 6 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Statistical Comparison of Algorithm Performance Through Instance Selection

Théo Matricon
Univ. Bordeaux, CNRS, LaBRI, UMR 5800, F-33400, Talence, France

Marie Anastacio
Leiden Institute of Advanced Computer Science, Leiden, The Netherlands

Nathanaël Fijalkow
CNRS, LaBRI, Bordeaux, France,
The Alan Turing Institute of data science, London, UK

Laurent Simon
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France

Holger H. Hoos
Leiden Institute of Advanced Computer Science, Leiden, The Netherlands
University of British Columbia, Vancouver, Canada

Abstract

Empirical performance evaluations, in competitions and scientific publications, play a major role in improving the state of the art in solving many automated reasoning problems, including SAT, CSP and Bayesian network structure learning (BNSL). To empirically demonstrate the merit of a new solver usually requires extensive experiments, with computational costs of CPU years. This not only makes it difficult for researchers with limited access to computational resources to test their ideas and publish their work, but also consumes large amounts of energy. We propose an approach for comparing the performance of two algorithms: by performing runs on carefully chosen instances, we obtain a probabilistic statement on which algorithm performs best, trading off between the computational cost of running algorithms and the confidence in the result. We describe a set of methods for this purpose and evaluate their efficacy on diverse datasets from SAT, CSP and BNSL. On all these datasets, most of our approaches were able to choose the correct algorithm with about 95% accuracy, while using less than a third of the CPU time required for a full comparison; the best methods reach this level of accuracy within less than 15% of the CPU time for a full comparison.

2012 ACM Subject Classification General and reference → Evaluation; Theory of computation → Automated reasoning; Theory of computation → Constraint and logic programming

Keywords and phrases Performance assessment, early stopping, automated reasoning solvers

Digital Object Identifier 10.4230/LIPIcs.CP.2021.43

Supplementary Material The source code can be found at:
Software (Source Code): `https://github.com/Theomat/PSEAS`
archived at `swh:1:dir:0cec861beb644ac8df9cd2f195457cbaabd16773`

1 Introduction

The amount of computational resources required to assess empirically whether a new automated reasoning algorithm exceeds state-of-the-art performance is growing as our ability to run experiments on challenging benchmark instances expands. From the evaluation of early algorithms against the human ability to solve given instances by hand [7] to extensive competitions requiring CPU years to determine a winner [10, 24, 30], the demands for computational power have grown along with the ability of state-of-the-art solvers to tackle larger instances. Moreover, each published idea is often the result of a number of unsuccessful


© Théo Matricon, Marie Anastacio, Nathanaël Fijalkow, Laurent Simon, and Holger H. Hoos;
licensed under Creative Commons License CC-BY 4.0



attempts, which developers either evaluated on a small set of instances, without a principled way of knowing how representative this evaluation has been, or in a more extensive way, implying days of CPU times.

This growth comes at a cost. By requiring large amounts of computational resources for compelling performance comparisons, the community restricts the ability of researchers with limited access to such resources to evaluate their ideas and publish their work. The energy consumption of such computations is also an increasing concern. Our work addresses this issue by proposing principled statistical methods to decide earlier when to stop running a less promising solver.

We introduce the per-set efficient algorithm selection problem (PSEAS): Given two algorithms, an incumbent A_{inc} and a challenger A_{ch}, and a set of problem instances $\mathcal{I}$, how can we minimise the computational resources (here: CPU time) required to determine, at a required level of confidence, whether A_{ch} performs better than A_{inc} on $\mathcal{I}$?

We are not aware of any prior work on this fundamental problem, but methods for addressing variants of it are used in several contexts. This includes, for example, general-purpose algorithm configurators, which compare the performance of several configurations of a single algorithm. While most configurators, such as SMAC [14] or ParamILS [15], simply look at the difference between their objective functions, the racing-based configurator `irace` [20], inspired by prior racing procedures from machine learning [22], addresses the problem of statistical confidence using a statistical test. However, all of them sample the instances uniformly at random. Other related work comes from the area of per-instance algorithm selection, for which Gent et al. [8] introduced a discrimination measure that we adapted to our context.

We describe five methods for selecting on which instances to run the competing algorithms, and three methods for deciding when to stop the evaluation. We compare the 15 resulting approaches on four benchmarks for classic computational problems: the propositional satisfiability problem (SAT), the constraint satisfaction problem (CSP) and Bayesian network structure learning (BNSL). On these datasets, our approaches can determine the better-performing algorithm with up to 98% accuracy, while using less than a third of the CPU time required for a full comparison, and the best methods achieve this level of accuracy within less than 15% of the CPU time for an exhaustive comparison.

The remainder of this paper is organised as follows: In Section 2, we introduce the PSEAS problem and present related work, and in Section 3, we describe methods for solving it. Section 4 presents implementation details and the datasets used in our experiments. Section 5 explains some of our design choices with experimental results, and the results from our main series of experiments are reported in Section 6. Finally, in Section 7, we draw some general conclusions and discuss future work.

2 Background and related work

The PSEAS problem formalises the following question: How to compare a new solver against the state of the art with as little computational power as possible? More specifically, how to select on which instances to run the new algorithm to do this comparison and on which criterion can this comparison be stopped if one algorithm performs significantly better than the other? The PSEAS problem or small variations of it also appears in competitions, where it could be used to disqualify low performing algorithms, or in algorithm configurators to abandon less promising configurations faster. For simplicity, we consider an algorithm as a method or solver with fixed parameter values and the state of the art as a single algorithm.

To answer these questions, we suppose that we have prior knowledge about the performance of the state-of-the-art algorithm, features describing the problem instances and performance samples or performance distributions on those instances.

Definition of the per-set efficient algorithm selection problem (PSEAS)

We let $\mathcal{I}$ denote the set of instances, $T_{cut} \in \mathbb{R}^+$ the cutoff threshold, m the performance metric that evaluates an algorithm on an instance, and c the cost function that evaluates the cost of running an algorithm on an instance. We consider two algorithms: the incumbent A_{inc} and the challenger A_{ch}, and assume that the cost $c(A_{inc}, I)$ and performance $m(A_{inc}, I)$ of running A_{inc} on an instance I is known for all instances, whereas these quantities are unknown on all instances for A_{ch}. This assumption is consistent with the fact that A_{inc} represents the state of the art, hence can be assumed to have been evaluated on many problems. The problem is to determine which of the two algorithms performs best according to $\sum_{I\in\mathcal{I}} m(A_{ch}, I)$ while running A_{ch} only on a subset $\mathcal{I}_{run} \subset \mathcal{I}$ that minimises the cost $\sum_{I\in\mathcal{I}_{run}} c(A_{ch}, I)$.

Scope of this work

We pose $\mathcal{A}_+ = \mathcal{A} \bigcup \{A_{ch}\}$ where $\mathcal{A}$ is a set of algorithms containing A_{inc} and providing background knowledge. Unless stated otherwise, we write I for an instance in $\mathcal{I}$ and A for an algorithm in $\mathcal{A}$. For simplicity we consider the algorithms to be deterministic, hence for an algorithm $A \in \mathcal{A}_+$, we define the running time as $rt(A, I) \in [0, T_{cut}]$ for an instance I. We define $m(A, I) = c(A, I) = rt(A, I)$: the running time of an algorithm is considered as a proxy for the energy cost of running it.

The performance of an algorithm is the sum on all instances of the running times bounded by a fixed cutoff time. It is consistent with the typical performance metric used in programming competitions, the Penalised Average Running time (PAR), which penalises algorithms that do not solve an instance before the cutoff time by assigning them the score of α times the cutoff time, for a constant α.

Some methods we describe rely on background knowledge about the set of instances. The required knowledge varies from one to another but is similar to the one used in the algorithm selection problem and thus readily accessible. We consider the following ways of specifying the background knowledge:

- *Sample-based*: for each instance I and algorithm A we have the running time $rt(A, I)$ of A on I.
- *Feature-based*: for each instance I we have a feature vector f_I.
- *Statistics-based*: for each instance I we have a prior in the form of a probability distribution δ_I over $[0, T_{cut}]$, expressing that $\delta_I(t)$ is the probability that A_{ch} solves the instance I at time t. In practice, we obtain this prior by fitting a distribution to the running times of A.

Note that above, $A \neq A_{ch}$, the background knowledge that is based on other algorithms. The implicit assumption is that running times of algorithms from $\mathcal{A}$ and feature vectors of the instances are both predictive of the running times of A_{ch}: for instance, if all algorithms in $\mathcal{A}$ solve an instance I very quickly, then so should A_{ch}. In other words A_{ch} is expected to have similar behaviour as the algorithms in $\mathcal{A}$. Similarly, if two feature vectors f_I and $f_{I'}$ are close for two instances I, I', then their running times should be close. These assumptions are prominently made in running-time prediction such as in Hutter et al. [16] and per-instance algorithm configuration (see e.g. Kerschke et al. [17]). In other words, the key insights and mathematical formalisation of our work are using the background knowledge described above to evaluate the expected performance of A_{ch}.

Related work

To the best of our knowledge, the problem we address has not been studied as such in the literature. However, similar questions appear in other settings.

In the early SAT Competitions (see e.g. the 2002 competition [27]), the competition consisted of two stages: first they ran all the solvers on a subset of instances to extract the top solvers, then the latter were run on all instances. The selection was done by experts: we propose to address this problem in an automated manner on a solver pair basis.

In the context of instance generation for CP problems, Gent et al. [8] propose a way to define how discriminating an instance is in order to generate instances for model selection based on samples of running times. This method does not answer our aim to reduce the running time but could lead to choose relevant instances. Thus, we included it in our experiments with a minor change to account for running time minimisation.

When we decide on which instance to run our algorithm, a score is assigned to each instance (see Section 3) which relates to the fitness functions used in evolutionary algorithms. At the time of writing, we found no published method that could be easily applied to our problem.[1]

For algorithm configuration (see e.g. Hoos [11]), which tries to find a set of parameter that optimises the performances of a configurable algorithm, comparing the performance of two configurations is a key element. SMAC and ROAR [14], as well as `irace` [20], pick uniformly at random the instances on which they run it, without considering prior knowledge they gathered. Racing procedures like `irace` are based on prior work from Maron and Moore [22] which aimed at comparing many machine learning models on a subset of test points to estimate their accuracy with a certain statistical confidence. In this line of work, `irace` requires evidence in the form of a statistical test to decide when to stop running a less promising configuration. SMAC and ROAR on the other hand compare the raw performance metric. We included the statistical test from `irace` in our experiments.

Our problem is also related to the per-instance algorithm selection problem (see e.g. Kerschke et al. [17]) in which one tries to know on which algorithm a specific instance should be run to be solved with the best possible performance. There are key differences that prevent us from using selection algorithms; typically their problem comes with prior knowledge in the form of instances features, that we do not always assume to have, and running time of the algorithms on other instances, which we do not have available for the new algorithm. Also, our main goal is to reduce the time needed to evaluate which one is the best.

Finally, there is a significant link with problems tackled by active learning methods [29], in particular the pool-based selective sampling problem, that tries to decide which instance among a set of unlabeled instances should be evaluated next. Those methods are aimed at a machine learning model and the choice of an instance is based on the impact it may have on the model (e.g. reducing its variance or expected error). In this work we limit our investigation to model-free methods.

3 Instance Selection and Discrimination Methods for PSEAS

Our goal is to define a *strategy* that sequentially chooses the instances on which to run A_{ch} and decides if the evidence so far gives sufficient confidence to stop the comparative evaluation. Algorithm 1 formalises this iterative process using a score-based approach: each

[1] We cover the recently published work of Bossek & Wagner [5] in Appendix B.

instance is assigned a score, which may be updated along the comparison to – intuitively – reflect the interest in running this instance. There are two main components in this algorithm: one for score computation (lines 2, 4, 7, see Section 3.1) and one for confidence (lines 1, 3, 6, see Section 3.2). The score enables to choose the best instance to run whereas the confidence tells when to stop the comparison. These two components will be explained in more details later.

Algorithm 1 Determine which of A_{inc}, A_{ch} performs best on $\mathcal{I}$ with a confidence threshold of C_{thres}; $C_{current}$ is the current confidence and depends on A_{inc}, $\mathcal{I}_{torun}$ are the instances on which A_{ch} has not been run.

1: set $\mathcal{I}_{torun} = \mathcal{I}$ and $C_{current} = 0$
2: compute score(I) for all $I \in \mathcal{I}$
3: **while** $C_{current} < C_{thres}$ **do**
4: pick $I^* \in \text{argmax}_{I \in \mathcal{I}_{torun}}\ score(I)$ and remove I^* from $\mathcal{I}_{torun}$
5: evaluate $rt(A_{ch}, I^*)$
6: update $C_{current}$
7: update score(I) for $I \in \mathcal{I}_{torun}$
8: **end while**
9: **return** best performing algorithm from (A_{inc}, A_{ch})

Strategy evaluation

We consider two metrics for evaluating strategies: the *cost* and the *accuracy*.

We measure the computational effort (which we want to minimise) as the ratio of the total running time for instances in $\mathcal{I}_{run}$, the set of instances on which A_{ch} has been run by the strategy, over the total running time over all instances; this results in a number between 0 and 1. Note that the goal is not to minimise the *number* of instances A_{ch} is run on, but rather the total running time of A_{ch} on these instances. To evaluate our strategy, we determine this cost over many ordered pairs of algorithms (A_{inc}, A_{ch}) and consider the median. Formally, for a set of ordered pairs $\mathcal{P}$:

$$cost(\mathcal{P}) = median\left[\left(\frac{\sum_{I \in \mathcal{I} \setminus \mathcal{I}_{torun}} rt(A_{ch}, I)}{\sum_{I \in \mathcal{I}} rt(A_{ch}, I)}\right)_{(A_{inc}, A_{ch}) \in \mathcal{P}}\right],$$

where $\mathcal{I}_{torun}$ are the instances that have not been run by the strategy during its execution, as defined in Algorithm 1. We note that $cost(\mathcal{P})$ only depends on A_{ch}, since A_{inc} is assumed to have already been run.

We measure the *accuracy* of a strategy (which we want to maximise), as the ratio of correct guesses made by the strategy when deciding which algorithm from an ordered pair of algorithms (A_{inc}, A_{ch}) performs best. Formally, for a set of ordered pairs $\mathcal{P}$:

$$accuracy(\mathcal{P}) = \frac{\sum_{(A_{inc}, A_{ch}) \in \mathcal{P}} 1_{\{\hat{A}_{best} = A_{best}\}}}{|\mathcal{P}|},$$

where A_{best} is the true best performing algorithm in (A_{ch}, A_{inc}), and $\hat{A}_{best}$ is the best algorithm given by the strategy. Our definition of *accuracy* uses the mean, since the median over the results of the indicator function would produce too limited a range of results to be useful for comparing strategies.

We note that the choice made in line 4 of Algorithm 1 aims at balancing the effects of two contradicting goals. The *instance selection component* tries to minimise the computational effort by deciding on which instances to run A_{ch}, based on a score given to each instance. The *discrimination component* decides, based on the data gathered so far, whether the expected accuracy, or confidence, is high enough to stop the comparison.

3.1 The instance selection component

With the aim of minimising the overall computational effort, our algorithm iteratively chooses the most relevant instance, according to a score (lines 2 and 7 in Algorithm 1). Instances with the highest score are expected to be the most relevant ones (i.e. intuitively giving the most information at the lowest cost).

Baseline: Uniform random sampling

As a baseline, we use a random sampling approach. In our algorithm, this corresponds to giving the same score to all instances, and thus to a uniform random choice at each iteration.

The discrimination-based selection method

This sample-based method is inspired by Gent et al. [8]; they developed it as a way to find optimal parameters of instances in an instance selection method for automated constraint model selection. The intuition is to choose the most discriminating instances first. Let $\rho > 1$ be a constant; an algorithm A is *ρ-dominated* on an instance I if there exists another algorithm A' such that $rt(A', I) \leq \rho \cdot rt(A, I)$. The *discrimination quality* of an instance I, denoted $G(I)$, is the fraction of algorithms that are ρ-dominated on this instance. Using this measure as-is would not take into account our goal of minimising the running time, so we divide the discrimination quality by the mean running time of the instance. The obtained score only needs to be computed once:

$$score(I) = \frac{G(I)}{mean[(rt(A, I))_{A \in \mathcal{A}}]}.$$

The variance-based selection method

This statistics-based method uses the intuition that the most interesting instances are the ones most likely to have very different running times for A_{inc} and A_{ch}. For each instance I we have a prior δ_I, which is the running time distribution of A_{ch}. We want to choose an instance with the highest variance $\text{argmax}_{I \in \mathcal{I}_{torun}} \mathtt{V}(\delta_I)$. As for the discrimination-based selection method, since we want to minimise the running time we divide by the mean running time of the instance. The obtained score only needs to be computed once:

$$score(I) = \frac{\mathtt{V}(\delta_I)}{\mathbb{E}[\delta_I]}.$$

The information-based selection method

This statistics-based method is based on a similar intuition as the previous method. We are interested in instances from which we gain as much information as possible; the variance is only one (natural) indicator of this information. Following this approach, we can also estimate

the information gained from a specific instance. The concrete information we are after[2] is given by the discrete random variable stating that A_{ch} is better than A_{inc}, formally defined as $sign(\Delta_{tot})$. Let Δ_I be the random variable defined as $\Delta_I := rt(A_{ch}, I) - rt(A_{inc}, I)$; we compute the expected information brought by Δ_I; hence the information gain is defined as follows for $I \in \mathcal{I}$:

$$IG_I(sign(\Delta_{tot})) := \mathbb{E}_{e_I \sim \Delta_i}[D_{\mathrm{KL}}(P_{+i} \mid\mid P)] \text{ with}$$
$$P = sign(\Delta_{tot}) | \forall J \in \mathcal{I}_{run}, \Delta_J = e_J$$
$$P_{+i} = sign(\Delta_{tot}) | \forall J \in \mathcal{I}_{run}, \Delta_J = e_J, \Delta_I = e_I$$

where D_{KL} is the Kullback–Leibler divergence, with the e_J being realizations of the Δ_J since the difference for the instances in $\mathcal{I}_{run}$ is known.

As for the previous method, to balance information and running time, we divide by the expected running time, and therefore use the following score function, which we update at each iteration:

$$score(I) = \frac{IG_I(sign(\Delta_{tot}))}{\mathbb{E}[\delta_I]}.$$

The feature-based selection method

In this feature-based and statistics-based method, we assume that for each instance I, we have a feature vector $f_I \in \mathbb{R}^n$ in some dimension n. The implicit assumption is that features are predictive of the running times of A_{ch}. We proceed in two steps:

- Constructing a distance metric $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, such that if $d(f, f')$ is small, then two instances with features f and f' have similar running times.
- Assigning a score to each instance $I \in \mathcal{I}_{torun}$.

Constructing a distance metric. The objective is to define a distance predictive of the running times; to this end, we introduce a weight for instance features, represented by a weight vector $\theta \in \mathbb{R}^n$. Let us consider distances of the form:

$$d_\theta(f_I, f_J) = \sqrt{\sum_{x=1}^{n} (\theta(x) \cdot (f_I(x) - f_J(x)))^2}.$$

Intuitively, for a feature x, the parameter $\theta(x)$ determines the importance of x in predicting the running times. Let us write m_I for the median time over all algorithms on instance I. We optimise over θ by considering:

$$\theta^* \in \operatorname*{argmin}_{\theta \in \mathbb{R}^n} \sum_{I,J \in \mathcal{I}} \left(d_\theta(f_I, f_J)^2 - |m_I - m_J|\right)^2;$$

i.e., d_{θ^*} is the best distance in this family for predicting differences in median running time. The parameter vector θ^* is the solution of a non-negative ordinary least square optimisation problem and can therefore be computed efficiently [18]. Note that the space complexity is quadratic in the number of instances and linear in the feature space dimension.

[2] See Section 3.2 for one possible expression of this concrete information: the variable Δ_{tot}.

Assigning a score. Given a distance metric d, we now define a score for a given problem instance. Here, it is convenient to minimise rather than maximise the following quantity with respect to d:

$$S(I) = \sum_{J \notin \mathcal{I}_{torun}} \frac{rt(A_{ch}, J)}{d(f_I, f_J)} + \sum_{J \in \mathcal{I}_{torun}} \frac{\mathbb{E}[\delta_J]}{d(f_I, f_J)} \quad \text{and} \quad score(I) = \frac{1}{S(I)}.$$

The score is updated at each iteration. In all previous methods, the score of an instance I only uses the information on I; the strength of this method is to gather and weight information over all instances. Indeed, the score of I is a weighted average over all running time predictions, meaning $\mathbb{E}[\delta_J]$ when $J \in \mathcal{I}_{torun}$ and $rt(A_{ch}, J)$ otherwise, and the prediction for J contributes to the prediction of I up to the multiplicative factor $\frac{1}{d(f_I, f_J)}$.

3.2 The discrimination component

The discrimination component aims at estimating the accuracy of the current decision of which among A_{inc} and A_{ch} performs best. However, this measure can never be accessed, since the complete data is not available. Hence we introduce the expected accuracy, or *confidence*, as a proxy for accuracy. We note that this is not an expectation in the statistical sense. The confidence has a different meaning and is computed differently for each discrimination method, as explained later. It provides a measure of the current state of the strategy. When the confidence reaches a threshold C_{thres} (line 3 of Algorithm 1), the strategy stops and returns the algorithm evaluated as being the best.

Baseline: Subset method

As a baseline, we use a fixed-size subset of instances: we fix $\gamma \in [0; 1]$ and decide to stop when A_{ch} has been run on $\lfloor \gamma |\mathcal{I}| \rfloor$ instances[3]. The confidence for this method is 0 until all instances of the subset have been executed then the confidence is 1.

Wilcoxon test

There is a large body of literature on statistical tests, and many of them can be used in the context of racing algorithms [3]. For instance, the F-Race [2] algorithm uses a Friedman two-way analysis of variance by ranks. However, this test concerns a family of candidates, while here, we are interested in an ordered pair of algorithms. When only two configurations remain, the F-Race algorithm switches to a *Wilcoxon matched-pairs signed-ranks test* [6], because it is more powerful and data-efficient than the Friedman test in that scenario [26].

The test we want to apply should satisfy the following requirements: it should be nonparametric, it should be applicable to paired data. Such a test would *not need any background knowledge*. We chose the Wilcoxon test because it satisfies our requirement while exploiting other properties of our data: data is measured on an interval scale, the differences (between running times) are symmetric and the magnitudes of the differences between our paired data is exploited. This test assumes that running times are independent and the two samples are mutually independent, that is not truly the case; however, we find that assuming independence is a good first approximation. This test is only based on observed data, it does not take into account the remaining instances. Through hypothesis testing we can find out

[3] Note that this does not ensure a ratio of γ for the total running time, as the total running time of A_{ch} over all instances is not available and therefore cannot be used for discrimination.

when there is enough evidence to stop, at which point the best algorithm is the one with the lowest mean running time. In this case, our confidence threshold C_{thres} is compared to the p-value of the alternative two-sided hypothesis. Let us note that other statistical tests, such as the Mann-Whitney U test, the permutation test, the Kolmogorov-Smirnov test, or the paired t-test do not satisfy our assumptions.

The distribution-based discrimination method

This method requires statistics-based background knowledge. Let us consider the following random variable computing the difference in performances:

$$\Delta_{tot} = \underbrace{\sum_{i \in \mathcal{I}_{run}} rt(A_{ch}, I) - rt(A_{inc}, I)}_{\text{constant}} + \sum_{I \in \mathcal{I}_{torun}} \underbrace{rt(A_{ch}, I)}_{\text{random variable}} - \underbrace{rt(A_{inc}, I)}_{\text{constant}}.$$

We are interested in determining the sign of Δ_{tot}, meaning which of the two algorithms performs best. For a fixed confidence threshold $C_{thres} = 1 - \varepsilon$, we estimate $\mathbb{P}(\Delta_{tot} > 0)$ and stop if:

- $\mathbb{P}(\Delta_{tot} > 0) \geq 1 - \varepsilon$, in which case A_{ch} performs worse than A_{inc},
- or $\mathbb{P}(\Delta_{tot} > 0) \leq \varepsilon$, meaning $\mathbb{P}(\Delta_{tot} \leq 0) \geq 1 - \varepsilon$, i.e. A_{ch} performs better than A_{inc}.

The confidence is $\mathbb{P}(\Delta_{tot} > 0)$ for the former case and $1 - \mathbb{P}(\Delta_{tot} > 0)$ for the latter. Looking at the definition of the random variable Δ_{tot}, its probability law can be described using translations and convolutions of the distributions $(\delta_I)_{I \in \mathcal{I}_{torun}}$. In practice, many natural classes of distributions (for instance Gaussian and Cauchy distributions) are closed under translations and convolutions, so $\mathbb{P}(\Delta_{tot} > 0)$ can be effectively computed or approximated.

Because running times are positive and algorithms are stopped when they reach the cutoff time T_{cut}, the running times are bounded. A distribution matching this behaviour would be a truncated distribution, but most are not closed under convolution, which we have stated above as a necessary property, so they cannot be used directly. Nevertheless, the sum of the bounds on individual running times can be used as bounds for Δ_{tot}, which we can model as a truncated distribution. For heavy-tailed distributions, such as the Cauchy distribution, the confidence is higher with a truncated distribution than without, as impossible cases are not taken into account, enabling to stop earlier.

4 Experimental setup

To empirically evaluate our approaches, we implemented them and ran them on all ordered pairs of algorithms from well-known benchmark scenarios.

4.1 Datasets

We use ASlib [4], a benchmark library for algorithm selection that contains datasets from competitions for various challenging problems, including Boolean satisfiability and constraint programming. It provides very relevant data on which our strategies can be tested, because such problems are the typical use-case scenario that we envisioned.

From ASlib, we use three datasets: the CSP MiniZinc 2016 (20 algorithms, 100 instances) dataset, which comprises performance data from the 2016 MiniZinc Challenge ('Free Search' Category) [19, 28]; the BNSL 2016 (8 algorithms, 1178 instances) dataset [21] from Bayesian Network structure learning; the SAT18 (37 algorithms, 353 instances) dataset, which consists of performance data from the EXP track of the 2018 SAT Competition [10]; and, to account

Table 1 Discrimination difficulty of our datasets according to our metric.

Dataset	CSP MiniZinc	BNSL	SAT 18	SAT 20
Mean	59.74	9.363	2458	78.88
Median	3.28	1.15	9.65	5.51
Top-3 mean	24.7	77.5	47.7	49.9

for more recent advances in SAT, we created the SAT20 (67 algorithms, 400 instances) dataset from the results of the main track of the 2020 SAT Competition [1]. Those datasets were chosen to cover a broad range of prominent problems and instance sets.

For our feature-based approaches, we decided to replace missing features by the mean value, as done by Hutter et al. [16]. Hence, no information can be extracted from such instances from a distributional point of view.

Discrimination difficulty

To get a sense of how difficult it is to discriminate between the algorithms from each dataset, we introduce a measure of difficulty based on how different the algorithms behave on our set of instances. We propose to use the following ratio: $\mathcal{D}_{discr}(A_{inc}, A_{ch}) = \frac{\sum_{I\in\mathcal{I}} median[(rt(A,I))_{A\in\mathcal{A}}]}{\left|\sum_{I\in\mathcal{I}} rt(A_{ch},I) - rt(A_{inc},I)\right|}$. This measure has been chosen, because it grows when the two algorithms have similar performance, and it is invariant under scaling, so that the difficulty remains the same if running times are multiplied by a constant factor. It is also symmetric: exchanging A_{inc} and A_{ch} leads to the same result.

In Table 1, we report the mean difficulty, the median difficulty over all pairs and the mean difficulty of the subset of the best 3 algorithms for all of our datasets. Based on this measure, we expect it to be easy to discriminate between algorithms from BNSL, while SAT18 should provide a bigger challenge. The large discrepancy between the mean and median value, seen for SAT18 in particular, is caused by small groups of algorithms with very close performances. Pairs of algorithms from those groups usually have very high difficulty, reaching up to a million for SAT18, which affects the mean.

4.2 Implementation details

Our implementation is available on GitHub (see supplementary materials).

To estimate the parameter of running time distributions, we use maximum likelihood estimation; and we use a Cauchy distribution for the distribution-based discrimination method, as motivated in Section 5.2. For the timeout correction, the seed was set to 0.

For the random instance selection method, the seed was also set to 0. The parameter ρ for the discrimination-based selection method was set to 1.2. For the information-based method, we use the expression of Δ_{tot} defined for the distribution-based method, and to compute the expected value, which is an integral, we use Simpson's rule.

For the Wilcoxon discrimination method, Conover [6] recommends at least 20 samples; however, this would represent up to 20% of our instance for the CSP Minizinc dataset. Thus, we decided to follow `irace` [20], which requires 5 samples in a context similar to ours. We found no significant performance change between different methods for managing zero differences, when paired data from both population is equal, as such we report the performance using Pratt's method [23].

5 Estimation of the running time distribution

Our approach relies heavily on our ability to estimate the distribution of running times of algorithms on the instances. This distribution is used in 3 out of the 5 instance selection methods and one of our 3 discrimination methods. As such, the choice of the distribution could significantly impact the performance of those strategies. Fitting a distribution on our data requires us to decide how to handle the cutoff time and which distribution to use.

We note that when predicting a running time, a log transformation is typically used [12, 16]. This transformation showcases better performance for predicting running times, because running times distributions tend to be heavy-tailed as shown in the work of Gomes et al. [9]. Since in our case we are mostly interested in predicting the mean or the sum over instances, we do not apply this log transformation.

5.1 Handling censored running times

As explained in the problem definition, after a given cutoff time T_{cut}, the given algorithm is stopped. Running times are thus right-censored, which limits our ability to estimate the true distribution.

Our method for handling time-outs is based on the one proposed by Hutter et al. [13], which itself is based on a prior work from Schmee & Hahn [25]. The resulting algorithm is Algorithm 2 for instance I, with parameters $M \in \mathbb{N}$ and $t_{max} \in \mathbb{R}_+$.

Algorithm 2 Correcting timeouts for a sample $(t_{I,A})_{A\in\mathcal{A}}$.

fit $\mathcal{D}istribution$ on $(t_{I,A})_{A\in\mathcal{A}}$ without the timeouts
set N to the number of timeouts in $(t_{I,A})_{A\in\mathcal{A}}$ and n to 0
while not converged **do**
    set S to $M \cdot N + n$ samples from $\mathcal{D}istribution$ then increment n
    for $k = 1$ to N **do**
        set q_k to quantile $\frac{k}{N+1}$ of S
        replace timeout k with $\min(q_k, t_{max})$ in $(t_{I,A})_{A\in\mathcal{A}}$
    end for
    fit $\mathcal{D}istribution$ on $(t_{I,A})_{A\in\mathcal{A}}$
end while
return $\mathcal{D}istribution$ and $(t_{I,A})_{A\in\mathcal{A}}$

There is a slight difference from the original algorithm, in the fact that at each iteration, we increment the number of samples used to enable convergence when there is a majority of timeouts on an instance. The parameter M enables to reduce the sampling variance; it is most important on instances with many timeouts. The parameter t_{max} prevents overly large variations of the samples. There are two steps in this algorithm: first we estimate the parameters of the distribution, second we replace the timeouts in the sample. They are repeated until convergence, when the estimated parameters of the distribution are stable. We decided to stop, when the squared difference between the parameters between two iterations is less or equal to 1. Schmee & Hahn [25] use the mean instead of the quantiles of a sample; however, heavy-tailed distributions such as the Cauchy distribution have an undefined mean. We chose to use the sampling approach used by Hutter et al. [13] which enabled them to translate the incertitude and improved the likelihood for their random forest models.

Table 2 Median log likelihood of Maximum Likelihood Estimation for Levy and Cauchy distributions over the instances of each dataset. The highest likelihood for each dataset is shown in boldface.

	CSP MiniZinc	BNSL	SAT 18	SAT 20
Levy	-129.6	**-58.08**	-299.7	-573.5
Cauchy	**-107.5**	-62.88	**-183.8**	**-364.9**

5.2 Choosing a distribution

What is the distribution satisfying the imposed constraints that gives the best performances?

In practice, since only a set of running times are provided, the distribution parameters must be estimated. We explained how the parameters were estimated in practice in Section 4.2, where here, we explain our choice of distribution. This choice can be motivated by choosing the best candidate distribution that has the lowest error on the set of all instances.

Since many running time distributions are heavy-tailed, we tested two heavy tailed distributions on our four datasets. We report on Table 2 the median log likelihood for each distribution; the parameters of these distributions were estimated using maximum likelihood estimation. The Cauchy distribution provides a clear advantage over the Levy distribution. The only case in which the Levy distribution yields a higher likelihood shows a much smaller difference between the two distributions.

6 Experiments

We designed and conducted extensive experiments, in order to answer the following questions:
Q1 – Can our strategies reduce the CPU time required for evaluating a new algorithm?
Q2 – How do the selection methods affect the accuracy of the strategies?
Q3 – Can our strategies discriminate well between top ranking algorithms?

A run consists of selecting an ordered pair (A_{ch}, A_{inc}) and running the strategy. On each run, all strategies have to compare the same A_{ch} and A_{inc}. In all of our experiments, we ran all of our strategies on each ordered pair of a given dataset.

6.1 General Performance Comparison

To answer Q1, we plotted our strategies in Figure 1, with a target confidence threshold $C_{thres} = 0.95$ (see Algorithm 1). For each of them, the y-axis shows accuracy (in percent) and the x-axis the median time used over all ordered pairs of algorithms, as defined in Section 3. As this corresponds to a multi-objective setup, we highlight the Pareto fronts induced by our results. This does not imply that we can produce a strategy that follows the Pareto front between points; however, by changing the confidence threshold C_{thres}, we can obtain local curves around the performance of each strategy (see Section 6.2). Note that while we show the performance of our strategies without applying a penalty for timeouts, using penalty coefficients from $[|1; 10|]$ did not affect our findings.

On all datasets, we observe that our random baseline (random sampling a subset of 20% of the instances) shows rather strong performance, with 89% to 100% accuracy for about 20% running time. Further investigation (see Section 6.2) shows that the accuracy of the random baseline increases steeply as we add more instances, until reaching about 20% of the instances, after which the increase in accuracy is substantially slower. Thus increasing the amount of instances does not lead to significantly higher accuracy. Moreover, more than half of the time, this strategy takes 17 to 22% of the running time, which means that the running

times of the instances follow a distribution such that they are as many easy instances as hard ones.We expect that this behaviour is linked to the nature of the competition datasets we are using; instances were gathered by experts to be representative and to show various levels of difficulty. We also note that the BNSL dataset, which is the one that gives the largest advantage to the random baseline, contains very few instances that are not solved within the cutoff time. Choosing these instances incurs a high penalty, because they offer no new information for deciding between the two algorithms, while using up a large amount of running time (see Appendix A).

On all datasets we see that the Wilcoxon method is superior and reaches the desired accuracy in less than 15% of the time; it thus represents the left-hand side of our Pareto front. The subset baseline uses consistently around 20% of the time but hardly reaches 90% accuracy on the hardest dataset; it contributes to the Pareto front only for BNSL. The distribution-based method tends to be more conservative and run longer but often reaches higher accuracies than the desired C_{thres} and thus marks the right-hand side of our Pareto front on our two SAT scenarios; however, it performs very poorly on BNSL, which is the scenario with the least background knowledge due to its low number of algorithms.

The instance selection methods do not show such a clear pattern. We notice, however, that the information-based method lies near or on the Pareto front when combined with Wilcoxon, whereas the discrimination-based and variance-based methods show strong performance when used in combination with distribution-based discrimination.

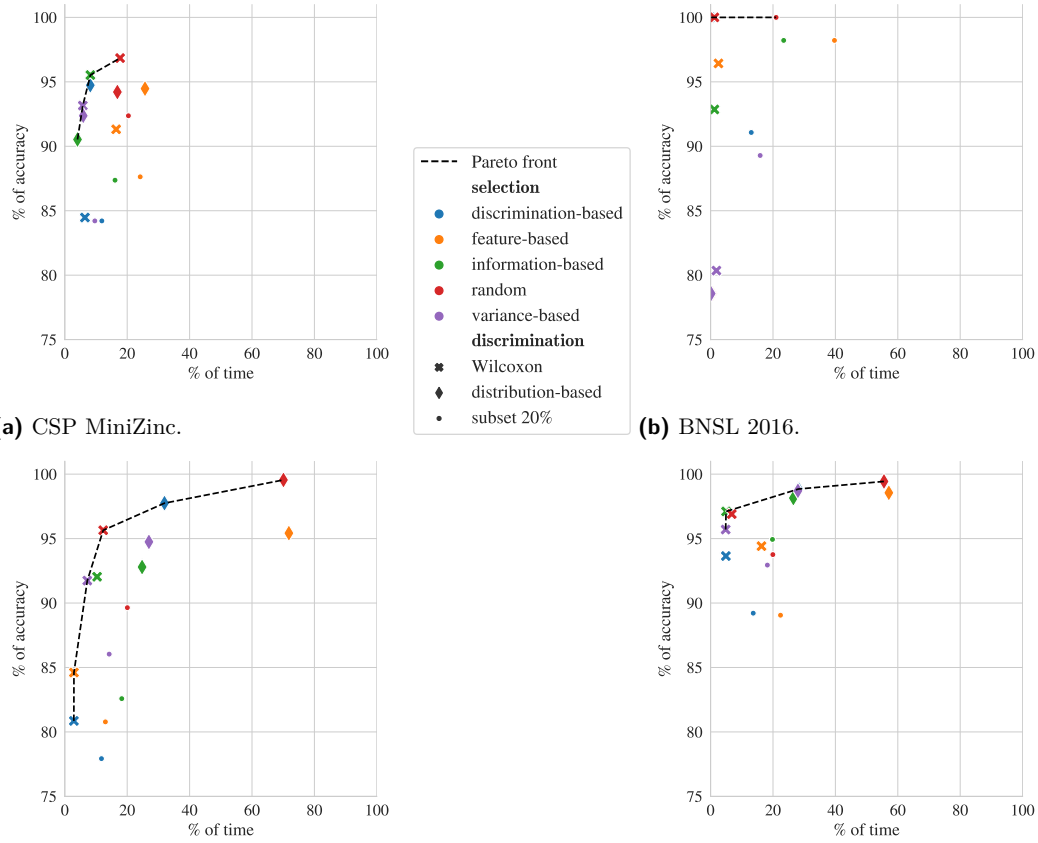
The evaluated strategies reach up to 95.5% accuracy using 8.21% of the time on the MiniZinc dataset, 95.6% accuracy using 12.3% of the time on SAT18, and 97.1% accuracy using 4.96% of the time on SAT20. For the BNSL dataset, we observed a surprising 100% accuracy while using only 0.0001% of the time using the discrimination-based selection with Wilcoxon discrimination that is hidden behind on Figure 1b, running a median number of 6 instances. The observed performance of our strategies is consistent with the ranking of the datasets according to our difficulty metric (see Table 1 in Section 4) for the distribution-based methods, but not for Wilcoxon, where SAT20 should have been harder than MiniZinc. Overall, in the worst-case, we manage to save 87.6% of CPU time while being 95.6% accurate and in the best case, we saved 95.0% of CPU time while being 97.1% accurate.

6.2 Accuracy over time

To answer Q2, we ran our strategies without stopping criterion, measuring regularly the percentage of accuracy and the time spent running A_{ch}. Figure 2 shows the accuracy (in percent) of the Wilcoxon and distribution-based discrimination methods on all our datasets.

Unlike Figure 1, which did not show any clear pattern regarding the instance selection methods, this analysis reveals two groups of methods. On all but the BNSL dataset, the information-based, variance-based and discrimination-based selection methods lead to a very high accuracy after 55 to 60% running time. This is consistent with the ratio of instances for which most algorithms time out, thus providing little discriminatory power. The feature-based method shows the lowest accuracy, and the random sampling comes in second to last after 40% of the running time.

The BNSL dataset is different, due to a low number of timeouts and large performance differences between the algorithms. In this case, randomly sampling instances offers very good accuracy after a few instances. None of the selection methods offers a clear advantage, because all instances provide evidence towards the algorithm performing best. This suggests that on easy datasets, the random method is a good choice, while on harder datasets containing instances that are not solved within the given cutoff time, more sophisticated selection methods can save running time.



(a) CSP MiniZinc.

(b) BNSL 2016.

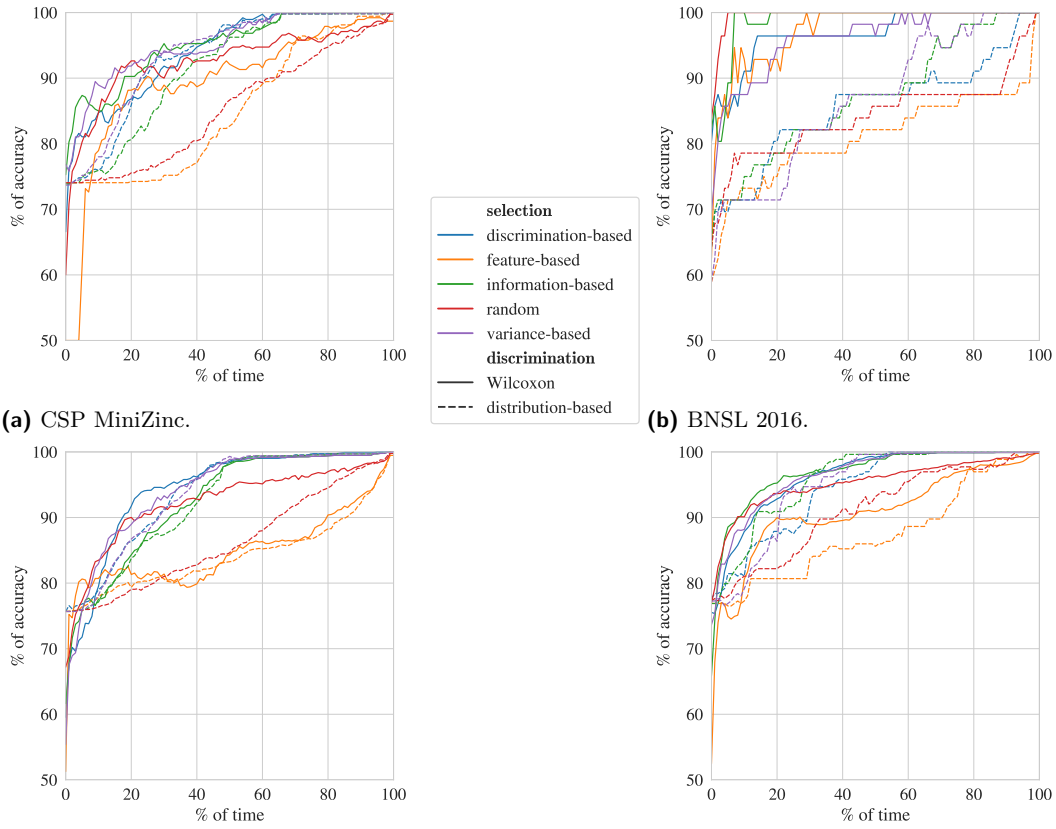
(c) SAT 18.

(d) SAT 20.

Figure 1 Accuracy over median running time. *y-axis*: percentage over all ordered pairs of algorithms in the dataset. *x-axis*: the time spent running the new algorithm.

6.3 Top ranking

To answer our last question, we decided to keep the top 10 ranking algorithms of the SAT20 dataset and use our strategies on this new dataset; this reflects that fact that often, the primary interest is in discriminating between top-ranking algorithms, be it to compare a new algorithm to the state of the art or to discriminate between the winners of a competition. As per our difficulty measure introduced in Section 4.1, the mean difficulty of the dataset thus obtained is 163, and the median is 22, which is higher than for any of our other datasets. Furthermore, the number of algorithms is reduced, which should reduce the performance of our methods based on prior knowledge. We report the results in Figure 3 analogous to what was done in Section 6.1; for comparative purposes, we also plot the performances on the full SAT20 dataset. The performance of the subset method decreases by more than 10% in accuracy. The distribution-based discrimination method requires more time for this subset, and the discrimination-based selection method drops out of the Pareto front. Because they require prior knowledge, these methods encounter difficulties with this more challenging dataset. The Wilcoxon method is least affected, since it is does not depend on prior knowledge; consequently, 3 out of the 4 strategies on the Pareto front use this method. The selection methods in combination with the Wilcoxon test are affected in different ways.



(a) CSP MiniZinc.

(b) BNSL 2016.

(c) SAT 18.

(d) SAT 20.

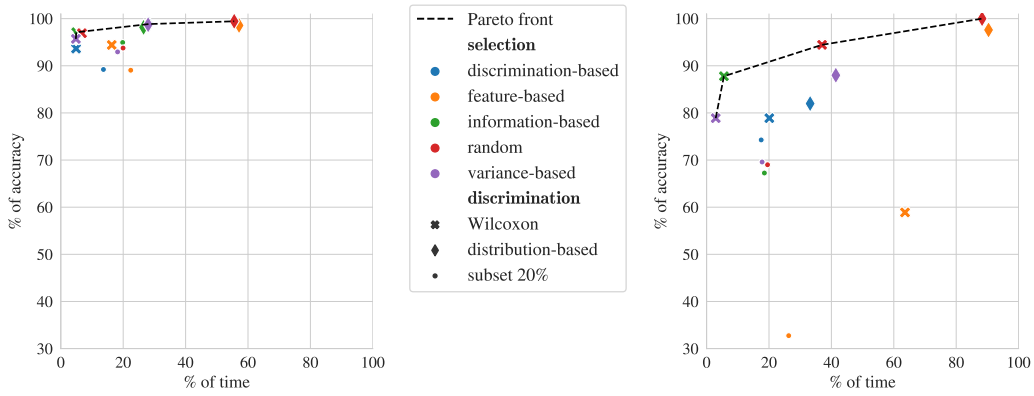
Figure 2 Accuracy over running time used. *y-axis*: percentage over all ordered pairs of algorithms in the dataset. *x-axis*: time spent running the new algorithm.

The information-based and variance-based approaches lead to a quick but less accurate decision, while random sampling leads to a slower decision, achieving 94.4% accuracy for 37.0% of running time.

In this experiment, which compares algorithms with similarly good performance, the information-based method using the Wilcoxon test suffers less than the other strategies, both in terms of cost and accuracy. All other methods lead to either high cost or poor accuracy.

7 Conclusions and future work

In this work, we have investigated methods for reducing the computational effort required for comparing the performance of two automated reasoning algorithms, while gathering sufficient statistical evidence to correctly identify the solver that performs better on a given set of problem instances. We defined the per-set efficient algorithm selection problem (PSEAS) in Section 2. We studied the case in which the performance of a given algorithm is evaluated based on its running time on a set of instances. We described a set of strategies in Section 3, inspired by related problems from the literature and by novel considerations, and tested these on four datasets covering SAT, CSP and structure learning in Bayesian networks. Our experimental evaluation in Section 6 shows that on these datasets, some of our strategies consistently return the correct answer with at least 95% accuracy, while using less than 15%



(a) SAT 20, full Dataset, 67 algorithms.

(b) SAT 20, top-10 algorithms.

Figure 3 Accuracy over running time used for the full and reduced SAT20 datasets. *y-axis*: percentage over all ordered algorithms' pairs in the dataset. *x-axis*: time spent running the new algorithm.

of the CPU time it would take to run the full comparison. In particular, using a Wilcoxon test to decide when to stop, while deciding the next instance to run based on the expected amount of information it can provide, is consistently near or among the best-performing approaches.

A finer-grained analysis of our instance selection methods (see Section 6.2) provides additional insights. We found that deciding on which instance to run based on its discrimination power, following the work of Gent et al. [8]), or simply on a notion of running time variance, has the potential to reduce the time required to take a decision when a significant fraction of the given instances are difficult.

Furthermore, we tested our methods on a smaller but more challenging set of algorithms, keeping the 10 best algorithms of the SAT20 competition. While the overall performance is lower than on the full dataset, the Wilcoxon method still reaches an accuracy of 94.4% in 37.0% of the overall running time. Overall, we found that for easy datasets, which discriminate very different algorithms on instances that can be solved quickly, random sampling offers good performance, but when facing hard instances or comparing well performing algorithms, it is beneficial to use more sophisticated methods.

In future work, it would be interesting to consider randomised algorithms. Incorporating empirical performance models [16] such as the ones used in algorithm configuration [14] and algorithm selection [31] could also open new avenues, e.g., involving the use of active learning methods. Finally, while the scope of our work presented here has been limited to comparing two algorithms, one interesting area of future work is focused extensions to many algorithms, in order to devise principles mechanisms for running competitions and other large-scale performance comparisons more efficiently.

References

1 Tomáš Balyo, Nils Froleyks, Marijn J.H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors. *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*. Department of Computer Science Report Series B. Department of Computer Science, University of Helsinki, Finland, 2020. URL: `http://hdl.handle.net/10138/318450`.

2 Mauro Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer Publishing Company, Incorporated, 1st ed. 2005. 2nd printing edition, 2009. doi:10.1007/978-3-642-00483-4.

3 Mauro Birattari, Thomas Stützle, Luis Paquete, and Klaus Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, GECCO'02, pages 11–18, San Francisco, CA, USA, January 2002. Morgan Kaufmann Publishers Inc. URL: http://gpbib.cs.ucl.ac.uk/gecco2002/AAAA223.pdf.

4 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.

5 Jakob Bossek and Markus Wagner. Generating instances with performance differences for more than just two algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '21, page 1423–1432, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3449726.3463165.

6 William Jay Conover. *Practical nonparametric statistics*, volume 350. John Wiley & Sons, 1998. URL: https://www.math.ttu.edu/~wconover/book.html.

7 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960. doi:10.1145/321033.321034.

8 Ian P. Gent, Bilal Syed Hussain, Christopher Jefferson, Lars Kotthoff, Ian Miguel, Glenna F. Nightingale, and Peter Nightingale. Discriminating instance generation for automated constraint model selection. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming*, pages 356–365, Cham, 2014. Springer International Publishing. doi:10.1007/978-3-319-10428-7_27.

9 Carla Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, January 2000. doi:10.1023/A:1006314320276.

10 Marijn Heule, Matti Järvisalo, and Martin Suda. Sat competition 2018. *Journal on Satisfiability, Boolean Modeling and Computation*, 11:133–154, September 2019. doi:10.3233/SAT190120.

11 Holger H. Hoos. Automated algorithm configuration and parameter tuning. In Youssef Hamadi, Eric Monfroy, and Frédéric Saubion, editors, *Autonomous Search*, pages 37–71. Springer, 2012. doi:10.1007/978-3-642-21434-9_3.

12 Barry Hurley and Barry O'Sullivan. Statistical regimes and runtime prediction. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, page 318–324. AAAI Press, 2015. URL: https://www.ijcai.org/Proceedings/15/Papers/051.pdf.

13 Frank Hutter, Holger Hoos, and Kevin Leyton-brown. Bayesian optimization with censored response data. In *In NIPS workshop on Bayesian Optimization, Sequential Experimental Design, and Bandits*, 2011. arXiv:1310.1947.

14 Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-25566-3_40.

15 Frank Hutter, Thomas Stützle, Kevin Leyton-Brown, and Holger H. Hoos. Paramils: An automatic algorithm configuration framework. *CoRR*, abs/1401.3492, 2014. arXiv:1401.3492.

16 Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014. doi:10.1016/j.artint.2013.10.003.

17 Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019. doi:10.1162/evco_a_00242.

18 Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. Society for Industrial and Applied Mathematics, 1995. doi:10.1137/1.9781611971217.

19 Marius Lindauer, Jan N. van Rijn, and Lars Kotthoff. Open algorithm selection challenge 2017: Setup and scenarios. In Marius Lindauer, Jan N. van Rijn, and Lars Kotthoff, editors, *Proceedings of the Open Algorithm Selection Challenge*, volume 79 of *Proceedings of Machine Learning Research*, Brussels, Belgium, September 11–12, 2017. PMLR. URL: http://proceedings.mlr.press/v79/lindauer17a.html.

20 Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. doi:10.1016/j.orp.2016.09.002.

21 Brandon Malone, Kustaa Kangas, Matti Järvisalo, Mikko Koivisto, and Petri Myllymäki. Empirical hardness of finding optimal bayesian network structures: algorithm selection and runtime prediction. *Machine Learning*, 107:247–283, January 2018. doi:10.1007/s10994-017-5680-2.

22 Oded Maron and Andrew W. Moore. The racing algorithm: Model selection for lazy learners. *Artif. Intell. Rev.*, 11(1-5):193–225, 1997. doi:10.1023/A:1006556606079.

23 John W. Pratt. Remarks on zeros and ties in the wilcoxon signed rank procedures. *Journal of the American Statistical Association*, 54:655–667, 1959. doi:10.1080/01621459.1959.10501526.

24 Luca Pulina and Martina Seidl. The 2016 and 2017 qbf solvers evaluations (qbfeval'16 and qbfeval'17). *Artificial Intelligence*, 274:224–248, 2019. doi:10.1016/j.artint.2019.04.002.

25 Josef Schmee and Gerald J. Hahn. A simple method for regression analysis with censored data. *Technometrics*, 21(4):417–432, 1979. URL: http://www.jstor.org/stable/1268280.

26 Sidney Siegel and N John Castellan Jr. *Nonparametric statistics for the behavioral sciences*. Mcgraw-Hill Book Company, 1988. doi:10.2307/2332896.

27 L. Simon, Daniel Leberre, and E. Hirsch. The SAT 2002 Competition. *Annals of Mathematics and Artificial Intelligence*, vol.43, Issue 1-4:307–342, 2005. URL: https://hal.archives-ouvertes.fr/hal-00022662.

28 Peter James Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The minizinc challenge 2008-2013. *AI Magazine*, 35(2):55–60, 2014. doi:10.1609/aimag.v35i2.2539.

29 Li-Li Sun and Xi-Zhao Wang. A survey on active learning strategy. In *2010 International Conference on Machine Learning and Cybernetics*, volume 1, pages 161–166, 2010. doi:10.1109/ICMLC.2010.5581075.

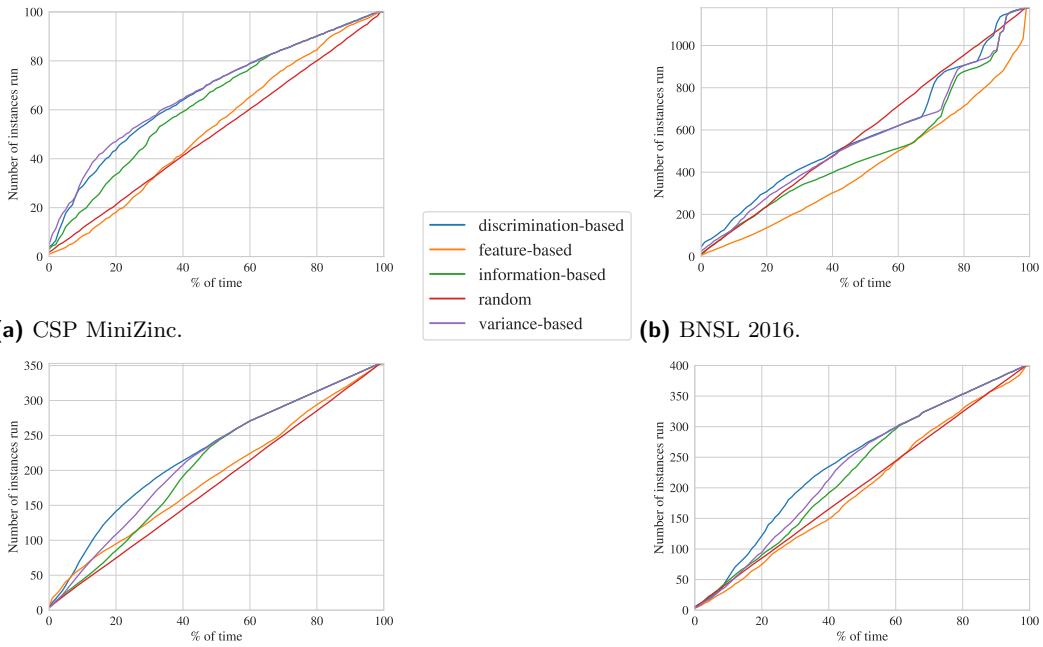
30 G. Sutcliffe. Proceedings of the 10th ijcar atp system competition (casc-j 10). *IJCAR*, 2020. URL: http://www.tptp.org/CASC/J10/Proceedings.pdf.

31 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. doi:10.1613/jair.2490.

A Choice of instances

To obtain deeper insight into the instances chosen by our selection methods, we performed some additional empirical analyses. In particular, we plotted the mean number of instances run for each percentage of overall running time used, averaging both quantities over all ordered pairs of algorithms from a given dataset. Figure 4 shows these plots for all our selection methods combined with the distribution-based discrimination method.

We notice that most selection methods give precedence to instances with low running times, as intuitively expected. The feature-based method, however, does not exhibit this behaviour. As in most of our experiments, our methods show significantly different behaviour on BNSL compared to the other datasets (see Figure 4b); here, our methods tend to select instances with running times similar to or higher than instances selected uniformly at random.



(a) CSP MiniZinc.

(b) BNSL 2016.

(c) SAT 18.

(d) SAT 20.

Figure 4 Number of instances run over running time used. *y-axis*: number of instances run. *x-axis*: time spent running the new algorithm.

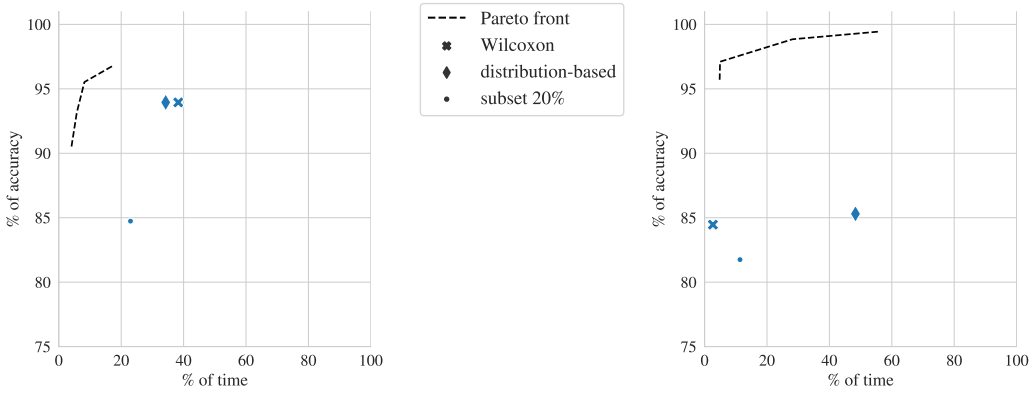
B The ranking-based selection method

B.1 The instance selection method

The ranking-based selection method is a sample-based method, inspired by Bossek & Wagner [5], who developed the explicit-ranking method as a fitness function for evolutionary algorithms, in order to generate instances that follow a given ranking. Their ranking is a lexicographical order, maximising three criteria in sequence: first, the similarity between the algorithms' ranking on the instance and the overall ranking, then two quantities that describe how different the running times of the algorithms are on this instance. The intuition is that given the samples, there is a ranking of algorithms over all instances; we want instances that are good at predicting this ranking – that is, instances on which the algorithms have a ranking closest to the overall ranking. Furthermore, we would also like that given two instances that have the same ranking, the instance that has the highest variance in running times between algorithms is chosen first.

In our case, the desired ranking is the ranking of algorithms with respect to their total performance. We associate each algorithm of $\mathcal{A}$ with an integer, and introduce the desired ranking π, such that $\pi(j)$ is the jth best performing algorithm. Then, for a given instance I, we can define the good pairs as $G_I = \{(j, j+1) \mid rt(A_{\pi(j)}, I) \leq rt(A_{\pi(j+1)}, I)\}$ and the bad pairs as $B_i = \{(j, j+1) \mid rt(A_{\pi(j)}, I) > rt(A_{\pi(j+1)}, I)\}$. The order in which the instances are run is the lexicographical order of the scoring function over instances:

$$score(I) = (|G_I|, \frac{f_B(I)}{median[(rt(A, I))_{A \in \mathcal{A}}]}, \frac{f_G(I)}{median[(rt(A, I))_{A \in \mathcal{A}}]}),$$



(a) CSP MiniZinc.

(b) SAT 20.

Figure 5 Percentage of accuracy with over median running time for the ranking-based selector. The Pareto front represents the front obtained from our results in Section 6.1. *y-axis*: percentage over all ordered pairs of algorithms in the dataset. *x-axis*: time spent running the new algorithm.

where

$$f_B(I) = \sum_{(j,j+1)\in B_I} (rt(A_{\pi(j+1)}, I) - rt(A_{\pi(j)}, I))$$

$$f_G(I) = \begin{cases} \sum_{(j,j+1)\in G_I} (rt(A_{\pi(j+1)}, I) - rt(A_{\pi(j)}, I)) \text{ if } |G_I| > 0 \\ -\infty \text{ else.} \end{cases}$$

As a normalisation step in the context of running time minimisation, we divided the original f_B and f_G by the median running time, which was not done by Bossek & Wagner [5]. The score is computed once in the beginning of Algorithm 1 and does not need to be updated at line 7.

B.2 Experimental results

Figure 5 shows the performance of the ranking-based selection methods combined with the three discrimination methods described in Section 3.2 (Wilcoxon, distribution-based and subset) on the CSP MiniZinc and on the SAT20 datasets. The results are presented the same manner as those in Section 6.1 and compared to the Pareto front obtained from the results of the methods presented there. As seen in the figure, in all but one case, the ranking-based approach is dominated in terms of performance by our methods discussed earlier. The one exception was observed on the SAT20 dataset, where combined with the Wilcoxon discrimination method, the ranking-based approach exhibits very short running time, but low accuracy, and thus shows performance to the lower left of our previously observed Pareto front. On the CSP MiniZinc dataset, the same combination of methods achieves good accuracy at the cost of high running time.

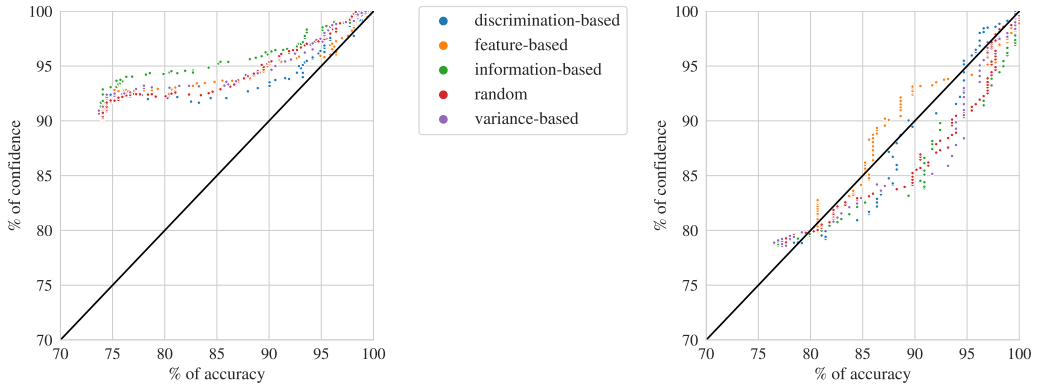
C Bias analysis

To better understand the behaviour of the discrimination component, we investigated the confidence achieved by our discrimination strategy. This should be correlated with the accuracy of the outcome (as described in Section 3.2). To test this, we ran our strategies,

without stopping criterion, measuring regularly the percentage of accuracy with respect to the confidence level of the discrimination component. Figure 6 shows the percentage of confidence over the accuracy for the distribution-based discrimination method. The black line indicates the desired behaviour where confidence is equal to accuracy. Points above this line represent overconfidence, while points below the line reflect underconfidence.

The discriminator starts, after one instance, with a confidence level of 90% for MiniZinc and 78% for SAT20, while the accuracy is around 75% in both cases. On CSP MiniZinc, it stays highly overconfident until the end, though the gap between confidence and accuracy diminishes. On SAT 20, confidence changes with accuracy, although we observe a tendency for underconfidence, except for the feature-based instance selector. This confidence discrepancy is clearly dependent on the dataset and does not seem correlated with our difficulty measure of the dataset.

Compared to the general performance from Figure 1, the closer the strategies are to correctly estimating the accuracy around our criterion of 95% of confidence, the better they perform.



(a) CSP MiniZinc.

(b) SAT 20.

Figure 6 Percentage of confidence with respect to percentage of accuracy of the distribution-based discrimination for all instance-selection methods. *y-axis*: percentage of confidence over all ordered pairs of algorithms in the dataset. *x-axis*: percentage of accuracy over all ordered pairs of algorithms in the dataset.