Sunny-as2: Enhancing SUNNY for Algorithm Selection (Extended Abstract) *

Tong Liu[1] , **Roberto Amadini**[2] , **Maurizio Gabbrielli**[2] , **Jacopo Mauro**[3]
[1]Meituan, Beijing, China
[2]Department of Computer Science and Engineering, University of Bologna, Italy
[3]Department of Mathematics and Computer Science, University of Southern Denmark, Denmark
lteu@icloud.com, {roberto.amadini,maurizio.gabbrielli}@unibo.it, mauro@imada.sdu.dk

Abstract

SUNNY is a k-nearest neighbors based Algorithm Selection (AS) approach that schedules and runs a number of solvers for a given unforeseen problem. In this work we present sunny-as2, an enhancement of SUNNY for generic AS scenarios that advances the original approach with wrapper-based feature selection, neighborhood-size configuration and a greedy approach to speed-up the training phase. Empirical evidence shows that sunny-as2 is competitive w.r.t. state-of-the-art AS approaches.

1 Introduction

A meta-algorithmic way to face the disparate nature of combinatorial problems and speed-up their resolution is to use a *portfolio* of different algorithms (or solvers) to be selected on different problem instances. The task of identifying suitable algorithm(s) for specific instances of a problem is known as per-instance *Algorithm Selection* (AS).

The SUNNY [Amadini *et al.*, 2014] portfolio approach was originally developed for Constraint Programming (CP). Given an unforeseen CP problem instance i, SUNNY retrieves the k-nearest neighbors of i and then selects the best solver(s) for these k instances, by assigning to them a time slot proportional to the number of solved instances. Finally, the selected solvers are sorted by average solving time and then executed on i.

Afterwards, SUNNY has been extended to handle generic AS scenarios (sunny-as) without much luck: the default configuration of SUNNY did not generalize well outside the CP field. The extensions detailed in [Liu *et al.*, 2021] significantly improve sunny-as thanks to the synergistic use of wrapper-based feature selection, neighborhood-size configuration and a greedy approach to speed-up the training phase. The empirical evaluations in [Liu *et al.*, 2021] show that this new version, called sunny-as2, outperforms sunny-as as well as other state-of-the-art AS methods.

2 Preliminaries

Every AS problem instance is embedded in a given context or *scenario*. Formally, we can define an AS scenario as a triple $(\mathcal{I}, \mathcal{A}, m)$ where: $\mathcal{I}$ is a set of *instances*; $\mathcal{A}$ is a set (or portfolio) of *algorithms* (or solvers) with $|\mathcal{A}| > 1$; and $m : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$ is a *performance metric* that we assume w.l.o.g. to be minimized. An (algorithm) *selector* s is a total mapping $s : \mathcal{I} \to \mathcal{A}$ that aims to return the best algorithm $A \in \mathcal{A}$, according to m, for any instance $i \in \mathcal{I}$. The AS problem consists in determining a selector s minimizing $\sum_{i\in\mathcal{I}} m(i, s(i))$. This definition can be easily extended to selectors that, like SUNNY, schedule more than one solver.

What makes hard AS is that the performance metric m on $\mathcal{I}$ is only *partially* known. The goal is hence to define a selector able to *estimate* the value of m for the instances $i \in \mathcal{I}$ where $m(i, A)$ is unknown. A common practice is to partition $\mathcal{I}$ into a training set $\mathcal{I}_{tr}$, used to build a selector s, and a test set $\mathcal{I}_{ts}$ used to evaluate the performance of s.

A further complication arises from the (NP-)hardness of the instances in $\mathcal{I}$. Typically a *timeout* τ is set and m is possibly extended with criteria to penalize a selector not finding any solution before τ occurs. A common practice is to use the *Penalized Average Runtime* (PAR) score with penalty $\lambda > 1$, penalizing unsolved instances with $\tau \times \lambda$. One drawback of PAR is that the values of τ and m can greatly change across different AS scenarios, thus making the absolute value of PAR hardly indicative for heterogeneous scenarios. In these cases *relative* metrics are better [Amadini *et al.*, 2022].

In the AS competitions 2015 and 2017 [Lindauer *et al.*, 2019] the *closed gap* score is used to measure how much a selector improves the *single best solver* (SBS) of the scenario w.r.t. the *virtual best solver* (VBS) with $m = \mathsf{PAR}_{10}$. The SBS is the best individual solver available, and its scenario performance is $m_{\text{SBS}} = \min\{\sum_{i\in\mathcal{I}} m(i, A) \mid A \in \mathcal{A}\}$. The VBS is a virtual selector always picking the best solver for a given instance, and its scenario performance is $m_{\text{VBS}} = \sum_{i\in\mathcal{I}} \min\{m(i, A) \mid A \in \mathcal{A}\}$. The closed gap for a selector s is $\frac{m_{\text{SBS}} - m_s}{m_{\text{SBS}} - m_{\text{VBS}}}$ with $m_s = \sum_{i\in\mathcal{I}} m(i, s(i))$. A good selector has m_s close to m_{VBS}, hence the closed gap close to 1. If instead m_s is near to m_{SBS} the closed gap tends to 0 or less.

AS scenarios typically characterize each instance $i \in \mathcal{I}$ with a corresponding *feature vector* $\mathcal{F}(i) \in \mathbb{R}^n$, so the algorithm selection for i is actually performed according to $\mathcal{F}(i)$. The *feature selection* (FS) process allows one to consider smaller feature vectors $\mathcal{F}'(i) \in \mathbb{R}^m$, derived from $\mathcal{F}(i)$ by projecting $m \leq n$ of its features. The goal is reducing

*The full version was published in the JAIR [Liu et al., 2021].

	x_1	x_2	x_3	x_4	x_5
A_1	τ	τ	**3**	τ	**278**
A_2	τ	**593**	τ	τ	τ
A_3	τ	τ	**36**	**1452**	τ
A_4	τ	τ	τ	**122**	**60**

Table 1: Runtime (in seconds). τ means the solver timeout.

the search space for fitting a model, diminishing the noise of misleading features and improving the prediction accuracy.

FS approaches can be classified in *filters*, *wrappers*. Filter methods select the features regardless of the model, trying to suppress the least interesting ones. These methods are efficient and robust to overfitting. In contrast, wrappers evaluate subsets of features possibly detecting interactions. They can be more accurate than filters, but also more exposed to overfitting and can have a much higher computational cost.[1]

2.1 SUNNY

The SUNNY portfolio approach was firstly introduced by Amadini et al. [2014]. SUNNY relies on a number of assumptions: *(i)* a small portfolio is usually enough to achieve a good performance; *(ii)* solvers either solve a problem quite quickly, or cannot solve it in reasonable time; *(iii)* solvers perform similarly on similar instances; *(iv)* a too heavy training phase is often an unnecessary burden.

Given a test instance $x \in \mathcal{I}_{ts}$, SUNNY produces a sequential schedule $\sigma = [(A_1, t_1), \ldots, (A_n, t_n)]$ where the algorithm $A_i \in \mathcal{A}$ runs for t_i seconds on x and $\sum_{i=1}^{n} t_i = \tau$. The schedule is obtained as follows. First, SUNNY employs k-NN to select from $\mathcal{I}_{tr}$ the subset I_k of the k instances closest to the feature vector $\mathcal{F}(x)$ according to the Euclidean distance. Then, it uses three heuristics to compute σ: *(i)* H_{sel}, for *selecting* the most effective algorithms $\{A_1, \ldots, A_n\} \subseteq \mathcal{A}$ in I_k; *(ii)* H_{all}, for *allocating* to each $A_i \in \mathcal{A}$ a certain runtime $t_i \in [0, \tau]$ for $i = 1, \ldots, n$; *(iii)* H_{sch}, for *scheduling* the sequential execution of the algorithms according to their performance in I_k.

The heuristics H_{sel}, H_{all}, and H_{sch} are based on the performance metric, and depend on the application domain. E.g., for CSPs H_{sel} selects the smallest set of solvers $S \subseteq \mathcal{A}$ that "solves" the most instances in I_k, breaking ties with runtime; H_{all} allocates to each $A_i \in S$ a time t_i proportional to the instances that S solves in I_k, by using a special *backup solver* covering the instances of I_k not solvable by any solver; finally, H_{sch} sorts the solvers by increasing solving time in I_k.

Example 1 *Let x be a CSP, $\mathcal{A} = \{A_1, A_2, A_3, A_4\}$ a portfolio, A_3 the backup solver, $\tau = 1800s$ the timeout, $I_k = \{x_1, ..., x_5\}$ the $k = 5$ neighbors of x, and the runtime of solver A_i on problem x_j defined as in Tab. 1. In this case, the smallest set of solvers that solve most instances in $\mathcal{I}_k$ are $\{A_1, A_2, A_3\}$, $\{A_1, A_2, A_4\}$, and $\{A_2, A_3, A_4\}$. The heuristic H_{sel} selects $S = \{A_1, A_2, A_4\}$ because these solvers are faster in solving the instances in I_k. Since A_1 and A_4 solve 2 instances, A_2 solves 1 instance and x_1 is not solved by any solver, the time window $[0, \tau]$ is partitioned in $2 + 2 + 1 + 1 = 6$ slots: 2 assigned to A_1 and A_4, 1 slot to A_2, and 1 to the backup solver A_3. Finally, H_{sch} sorts in ascending order the solvers by average solving time in I_k. The final schedule produced by SUNNY is, therefore, $\sigma = [(A_4, 600), (A_1, 600), (A_3, 300), (A_2, 300)]$.*

SUNNY aims to avoid *overfitting* w.r.t. the performance of the solvers in the selected neighborhood, i.e., it tries to not be too tied to the strong assumption that the runtimes in the neighborhood faithfully reflect the runtime on the instance to solve. Clearly, the design choices of SUNNY have pros and cons. For example, the schedule in Example 1 cannot solve the instance x_2 although x_2 is actually in the neighborhood.

The sunny-as [Amadini *et al.*, 2015] tool implements SUNNY algorithm to handle generic AS scenarios of *ASlib library* [Bischl *et al.*, 2016]. In its optional pre-processing phase, sunny-as can perform only filter-based feature selection and select a pre-solver to be run for a short time. At runtime, it produces the schedule of solvers by following the approach explained above. The AS challenge 2015 [Lindauer *et al.*, 2019] underlined some issues of sunny-as, especially in SAT scenarios. One reason is that sunny-as does not learn any parameter according to the input AS scenario, but it only uses default values (e.g., the neighborhood size is set to square root of its training set, rounded to the nearest integer).

3 sunny-as2

sunny-as2 is the evolution of sunny-as and its preliminary prototype attended the 2017 AS competition.[2] It introduces an *integrated* approach where features and k-value of the underlying k-NN are *co-learned* during the training step. In particular, *wrapper*-based FS is performed. This makes sunny-as2 "less lazy" than the original SUNNY, which only scaled the features in $[-1, 1]$ without performing any actual training.

To improve the configuration accuracy and robustness, and to assess the quality of a parameters setting, sunny-as2 uses *nested cross-validation* [Loughrey and Cunningham, 2005]. The original dataset is split into five folds thus obtaining five pairs $(T_1, S_1) \ldots, (T_5, S_5)$ where the T_i are the *outer* training sets and the S_i are the (outer) *test sets*, for $i = 1, \ldots, 5$. For each T_i we then perform an inner 10-fold CV to get a suitable parameter setting. We split each T_i into further ten sub-folds $T'_{i,1}, \ldots, T'_{i,10}$, and in turn for $j = 1, \ldots, 10$ we use a sub-fold $T'_{i,j}$ as *validation set* to assess the parameter setting computed with the inner *training set*, which is the union of the other nine sub-folds $\bigcup_{k \neq j} T'_{i,k}$. We then select, among the 5 configurations obtained, the one for which SUNNY achieves the best PAR10 score on the corresponding validation set. The selected configuration is used to run SUNNY on the paired test set S_i.

Before explaining how sunny-as2 learns features and k-value, we first describe greedy-SUNNY, the "*greedy variant*" of SUNNY introduced to speed-up the solvers' selection. SUNNY computes the smallest portfolio that maximizes the number of solved instances in a neighborhood $\mathcal{N}$. The worst-case time complexity is $O(2^m)$, where m is the number of

[1]Embedded methods integrating feature selection into the learning algorithm also exist.

[2]The 2017 AS competition was named OASC challenge, while the 2015 AS competition was called ICON challenge.

available solvers. greedy-SUNNY instead starts from $S = \emptyset$ and adds to S one solver at a time by selecting the one solving the most instances in $\mathcal{N}$. These instances are then removed from $\mathcal{N}$ and the process is repeated until $|S| = \lambda$ or $\mathcal{N} = \emptyset$, where λ is an external threshold.[3] According to empirical experiments, it is reasonable to set a small value for λ (e.g., 3) as also suggested by the experiments in [Lindauer *et al.*, 2016]. If λ is a constant, the worst-case time complexity of greedy-SUNNY is $O(m \cdot |\mathcal{N}|)$.

3.1 Selecting Features and Neighborhood Size

sunny-as2 uses greedy-SUNNY to learn the features and/or the k-value for a given AS scenario. The user can choose among three different flavors, namely:

1. **sunny-as2-k**. All the features are used and only k-configuration is performed by varying k in the range $[1, maxK]$ where $maxK$ is a user-defined parameter.
2. **sunny-as2-f**. The k-value is fixed to its default and FS is performed. Starting from $F = \emptyset$, the feature decreasing the most the PAR_{10} is added, until PAR_{10} increases or $|F| = maxF$ where $maxF$ is a user-defined parameter.
3. **sunny-as2-fk**. The k-value *and* the features are configured together by running **sunny-as2-f** with different values of k in $[1, maxK]$.

Algorithm 1 shows through pseudo-code how sunny-as2-fk selects the features and the k-value. LEARNFK takes as input the available algorithms $\mathcal{A}$, the maximum schedule size λ for greedy-SUNNY, the set of training instances $\mathcal{I}$, the maximum neighborhood size *maxK*, the original set of features $\mathcal{F}$, and the upper bound *maxF* on the maximum number of features to be selected. It returns the learned value $bestK \in [1, maxK]$ for the neighborhood size and the learned set of features $bestF \subseteq \mathcal{F}$ having $|bestF| \leq maxF$.

After the i-th iteration of the outer *for* loop (Lines 7–17) the current set of features $currFeat$ consists of exactly i features. Each time $currFeat$ is set, the inner *for* loop is executed n times to evaluate different values of k on dataset $\mathcal{I}$. The evaluation is performed by GETSCORE, which returns the score (the higher, the better) of a particular SUNNY setting. By default, GETSCORE relies on greedy-SUNNY instead of the original SUNNY to have a faster solver selection.

At the end of the outer *for* loop, if adding a new feature could not improve the score of the previous iteration (i.e., with $|currFeat| - 1$ features) the learning process halts. Otherwise, both the features and the k-value are updated and a new iteration begins, until the score cannot be further improved or the maximum number of features *maxF* is reached.

If $d = \min(maxF, |\mathcal{F}|)$, $n = \min(maxK, |\mathcal{I}|)$ and the worst-case time complexity of GETSCORE is γ, then the overall worst-case time complexity of LEARNFK is $O(d^2 n \gamma)$.

From LEARNFK one can easily deduct the algorithms for learning either the k-value (for sunny-as2-k) or the selected features (for sunny-as2-f): in the first case, the outer *for* loop is omitted because features do not vary; in the second case, the inner loop is skipped because the value of k is constant.

[3]As one can expect, greedy-SUNNY does not guarantee that S is the *minimal* set of solvers solving the most instances of $\mathcal{N}$.

Algorithm 1 Configuration procedure of sunny-as2-fk.

1: **function** LEARNFK($\mathcal{A}$, λ, $\mathcal{I}$, *maxK*, $\mathcal{F}$, *maxF*)
2: $bestF \leftarrow \emptyset$
3: $bestK \leftarrow 1$
4: $bestScore \leftarrow -\infty$
5: **while** $|bestF| <$ *maxF* **do**
6: $currScore \leftarrow -\infty$
7: **for** $f \in \mathcal{F}$ **do**
8: $currFeat \leftarrow bestF \cup \{f\}$
9: **for** $k \leftarrow 1, \ldots,$ *maxK* **do**
10: $s \leftarrow$ GETSCORE$(\mathcal{A}, \lambda, \mathcal{I}, k, currFeat)$
11: **if** $s > currScore$ **then**
12: $currScore \leftarrow s$
13: $currFeat \leftarrow f$
14: $currK \leftarrow k$
15: **end if**
16: **end for**
17: **end for**
18: **if** $currScore \leq bestScore$ **then**
19: **break** ▷ Cannot improve best score
20: **end if**
21: $bestScore \leftarrow currScore$
22: $bestF \leftarrow bestF \cup \{currFeat\}$
23: $bestK \leftarrow currK$
24: $\mathcal{F} = \mathcal{F} - \{currFeat\}$
25: **end while**
26: **return** $bestF$, $bestK$
27: **end function**

4 Experiments

In this section we report part of the experiments we conducted over several configurations of sunny-as2. We omit here the sensitivity evaluations of the parameters that sunny-as2 cannot learn, i.e., the split modes for cross-validation and the limits on the numbers of features, training instances, and schedule size. Also, we skip the analysis on the performance variability of sunny-as2 and other insights on SUNNY. All this information can be found in [Liu *et al.*, 2021].

In the following we will show how SUNNY can benefit from learning the k-value and/or the features, how greedy-SUNNY can improve the original SUNNY, and some comparisons of sunny-as2 against other AS approaches.[4]

We first compared sunny-as2-f, sunny-as2-k, and sunny-as2-fk against the original version of sunny-as on 12 ASlib scenarios. Tab. 2 shows the average closed gap of each approach. Interestingly, there is not a dominant configuration. As also shown in Lindauer et al. [2016], a proper k-configuration is crucial for SUNNY—indeed, sunny-as2-k achieves the peak performance in 7 scenarios out of 12. However, sunny-as2-fk has the best average performance across

[4]All the experiments were run on Linux machines with Intel Corei5 3.30GHz processors and 8 GB of RAM. We used a time cap of 24 hours for learning the parameters. All the ASlib scenarios are publicly available at https://github.com/coseal/aslib_data

Approach	Caren	Mira	Magnus	Monty	Quill	Bado	Svea	Sora	ASP	CPMP	GRAPHS	TSP	Average
sunny-as	−0.0517	−0.1289	0.6343	0.4291	0.6976	0.7854	0.6458	0.1781	0.6674	0.7488	0.6968	−0.4457	0.4047
sunny-as2-f	−0.0603	−0.1649	0.4425	0.4489	0.6854	0.7695	0.5783	0.2459	0.7717	0.7771	0.5663	−0.1058	0.4129
sunny-as2-k	0.1611	0.0276	0.6352	0.5830	0.7361	0.7976	0.6915	0.3591	0.7193	0.7273	0.6504	−0.8822	0.4338
sunny-as2-fk	0.0845	−0.1891	0.4458	0.5846	0.7139	0.7590	0.6643	0.3428	0.7454	0.7885	0.5614	−0.0343	0.4556

Table 2: Closed gap of sunny-as and sunny-as2 variants over different ASlib scenarios

Approach	Caren	Mira	Magnus	Monty	Quill	Bado	Svea	Sora	All
sunny-as2	0.7855	0.0291	0.5833	0.8450	0.8414	0.9057	0.6077	0.4059	0.6255
sunny-as2-fk-OASC	0.9099	0.4320	0.5723	0.9102	0.5691	0.8444	0.6578	0.0084	0.6130
ASAP.v2	0.3238	0.5053	0.4979	0.8331	0.6981	0.7573	0.6765	0.2150	0.5634
AutoFolio	0.5995	0.0846	0.6707	0.6923	0.5165	0.8089	0.6585	0.3479	0.5474
*Zilla	0.6356	0.4761	0.4932	0.4194	0.8001	0.7322	0.5850	0.1754	0.5396
ASAP.v3	0.3276	0.5091	0.4963	0.7631	0.5797	0.8048	0.6881	0.0639	0.5291
sunny-as2-k-OASC	0.6440	−0.0137	0.4924	0.6318	0.8495	0.7441	0.5789	0.0021	0.4911
SUNNY-original	0.7687	−0.8996	0.5859	0.4025	0.7697	0.7687	0.4866	0.1899	0.3841
Random Forest	0.1952	0.4892	0.2037	−1.4422	−0.4737	0.7913	0.5966	0.0934	0.0567
AS-RF	−1.0617	0.4947	−1.0521	−6.8992	−0.3280	0.8331	0.5853	−0.3700	−0.9747

Table 3: Closed gap of different AS approaches over the instances of the 2017 AS competition

all the scenarios. The main reason is arguably the poor performance of sunny-as2-k in the TSP scenario. sunny-as is clearly less promising than any other variant of sunny-as2, but it is the best approach for GRAPHS. What we can conclude from Tab. 2 is that most of the performance improvement is due to the selection of the right neighborhood size k. However, feature selection also gives a positive contribution.

We recall that by default greedy-SUNNY is used to compute k-value and/or features on the training sets. Then, sunny-as2 sets the corresponding SUNNY parameters to the computed values when it runs on the test sets. The rationale is to speed-up the training time, so that for scenarios with a high number of algorithms (e.g., Svea) can exceed 24 hours of computation. Interestingly, we later realized that training sunny-as2 with SUNNY instead of greedy-SUNNY does not bring any substantial benefit. Surprisingly, in 8 scenarios out of 12 the performance deteriorates. We also noted that the peak performance in any scenario is achieved when SUNNY is used for testing. Using greedy-SUNNY on an unforeseen instance might therefore be useful in time-sensitive contexts where exponential-time scheduling is not acceptable but, in general, SUNNY provides a more precise scheduling.

Table 3 shows the hypothetical performance of the improved sunny-as2 in the 2017 AS competition.[5] In addition to the original competitors (viz., *Zilla, ASAP, AS-RF and the preliminary versions of sunny-as2 called sunny-as2-fk-OASC and sunny-as2-k-OASC in Tab. 3) we added 3 more baselines: AutoFolio [Lindauer *et al.*, 2015], the original SUNNY approach [Amadini *et al.*, 2014], and an off-the-shelf Random Forest approach.

Tab. 3 shows that sunny-as2 has the highest average closed gap, and it is the best approach in Bado and Sora scenarios. Unsurprisingly, its performance is quite close to the one of sunny-as2-fk-OASC. ASAP.v2 does not attain the best score in any scenario, but in general its performance is robust and effective—this confirms what reported in [Gonard *et al.*, 2019]. AutoFolio is slightly behind ASAP.v2, nevertheless it achieves good results and it is the best approach for the Magnus scenario. As sunny-as2, also AutoFolio suffers in scenarios like Caren and Mira having a small number of instances. *Zilla and ASAP.v3 also close more than 50% of the gap between the SBS and the VBS. sunny-as2-k-OASC is instead slightly below this threshold: the performance difference w.r.t. sunny-as2-fk-OASC denotes the importance of a proper feature selection. The original SUNNY approach is even worse: this confirms the effectiveness of the improvements introduced by sunny-as2.

At the bottom of the table we find the AS approaches based on Random Forest. However, it is crucial to highlight that the chosen performance metric plays a fundamental role as clearly shown in [Liu *et al.*, 2021]. For example, replacing the closed gap score with the one adopted in the MiniZinc Challenge [Stuckey *et al.*, 2014] literally overturns the closed-gap ranking. An in-depth discussion of this issue can be found in Amadini et al. [2022].

5 Conclusions

We experimentally learned that wrapper-based feature selection and k-configuration are quite effective for SUNNY, and perform better when integrated. Moreover, a sub-optimal greedy approach for solver selection enables a more robust, fast and effective training w.r.t. the schedule generation procedure of the original SUNNY approach. These three ingredients significantly improved the SUNNY performance for generic AS scenarios, making sunny-as2 a state-of-the-art AS approach for runtime minimization.

A natural future direction for SUNNY is the study of alternative AI-driven solver selection mechanisms, and the extension of sunny-as2 to optimization problems, for which the solution(s) quality must be taken into account.

[5]The submitted version was prototypical and slightly different.

References

[Amadini *et al.*, 2014] Roberto Amadini, Maurizio Gabbrielli, and Jacopo Mauro. SUNNY: a Lazy Portfolio Approach for Constraint Solving. *TPLP*, 14(4-5):509–524, 2014.

[Amadini *et al.*, 2015] Roberto Amadini, Fabio Biselli, Maurizio Gabbrielli, Tong Liu, and Jacopo Mauro. SUNNY for algorithm selection: a preliminary study. In *Proceedings of the 30th Italian Conference on Computational Logic, Genova, Italy, July 1-3, 2015.*, pages 202–206, 2015.

[Amadini *et al.*, 2022] Roberto Amadini, Maurizio Gabbrielli, Tong Liu, and Jacopo Mauro. On the evaluation of (meta-)solver approaches. *arXiv preprint arXiv:2202.08613*, 2022.

[Bischl *et al.*, 2016] 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.

[Gonard *et al.*, 2019] François Gonard, Marc Schoenauer, and Michèle Sebag. Algorithm selector and prescheduler in the icon challenge. In *Bioinspired Heuristics for Optimization*, pages 203–219. Springer, 2019.

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

[Lindauer *et al.*, 2016] Marius Lindauer, Rolf-David Bergdoll, and Frank Hutter. An Empirical Study of Per-instance Algorithm Scheduling. In *LION*, volume 10079 of *LNCS*, pages 253–259. Springer, 2016.

[Lindauer *et al.*, 2019] Marius Lindauer, Jan N. van Rijn, and Lars Kotthoff. The algorithm selection competitions 2015 and 2017. *Artificial Intelligence*, 272:86–100, 2019.

[Liu *et al.*, 2021] 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.

[Loughrey and Cunningham, 2005] John Loughrey and Pádraig Cunningham. Overfitting in wrapper-based feature subset selection: The harder you try the worse it gets. In *Research and development in intelligent systems XXI*, pages 33–43. Springer, 2005.

[Stuckey *et al.*, 2014] Peter J. Stuckey, Thibaut Feydy, Andreas Schutt, Guido Tack, and Julien Fischer. The MiniZinc Challenge 2008-2013. *AI Magazine*, 35(2):55–60, 2014.