Feature Selection for SUNNY: a Study on the Algorithm Selection Library

Roberto Amadini, Fabio Biselli, Maurizio Gabbrielli, Tong Liu, Jacopo Mauro

Feature Selection for SUNNY: a Study on the Algorithm Selection Library

Roberto Amadini*†, Fabio Biselli*, Maurizio Gabbrielli*†, Tong Liu* and Jacopo Mauro‡
*Department of Computer Science and Engineering, University of Bologna, Italy.
†FOCUS team, INRIA, France.
‡Department of Informatics, University of Oslo, Norway.

***Abstract*—Given a collection of algorithms, the Algorithm Selection (AS) problem consists in identifying which of them is the best one for solving a given problem. The selection depends on a set of numerical features that characterize the problem to solve. In this paper we show the impact of feature selection techniques on the performance of the SUNNY algorithm selector, taking as reference the benchmarks of the AS library (ASlib). Results indicate that a handful of features is enough to reach similar, if not better, performance of the original SUNNY approach that uses all the available features. We also present `sunny-as`: a tool for using SUNNY on a generic ASlib scenario.**

I. Introduction

The *Algorithm Selection* (AS) problem aims at selecting the best algorithm for solving a given problem among a collection of available algorithms. Initially proposed in 1976 by Rice [30], in the last decade AS has attracted increasing attention and a variety of AS approaches[1] have been proposed.

Given a set (or portfolio) of algorithms $\{A_1, \ldots, A_m\}$, the prediction of the "best" algorithm A_k for a new, unseen problem x clearly depends on a given notion of performance measure (e.g., the runtime or the solution quality) and it is usually performed on the basis of the *feature vector* of x, i.e., a collection $F_x = (f_1, \ldots, f_d)$ of $d \geq 1$ numerical attributes that characterize the input problem x. Unfortunately, not all the features are equally important: very often some features are more informative than others. Determining how many and which features to use is crucial for the performance of AS. This process, well-known also in related fields like Machine Learning and statistics, is called *Feature Selection* (FS).

SUNNY [2] is an algorithm selector originally tailored for Constraint Satisfaction Problems (CSPs) and later on adapted for dealing with Constraint Optimization Problems (COPs) [4]. SUNNY is also the algorithm that underpins `sunny-cp` [3], a constraint solver exploiting a portfolio of different constituent solvers for solving both CSPs and COPs encoded in MiniZinc language [27]. In this paper we show the impact of FS techniques on the performance of the SUNNY algorithm selector. We use the benchmarks of the *Algorithm Selection library* (ASlib) [6], a recently introduced repository which encodes different AS scenarios in a standardized format with the aim of sharing and comparing different AS approaches. Note that the contribution of this paper is not to compare the performance of SUNNY with other AS approaches as done in the preliminary study of [1]. Here instead we fix the SUNNY original approach as a baseline, and we evaluate and compare different FS methods on all the heterogeneous scenarios of the ASlib. The goal is to assess if and how SUNNY can take advantage of a reduced set of features in order to reach a similar, or maybe better, performance w.r.t. its original version which uses all the available features.

We introduce new normalized metrics for evaluating well-known FS methods over different ASlib scenarios by also varying the number of selected features. Empirical results indicate that —although there is not a dominant FS method— a handful of features is often enough for SUNNY to reach similar, if not better, performance of the default approach exploiting all the given features. This confirms the findings reported in [6], [7], [18], [25] showing that only few features can be enough to describe and discriminate the structure of the problem to be solved. Unfortunately, it seems that no pattern exists to discriminate *ex-ante* the best features for a given scenario: even scenarios having the same feature space have indeed different best features.

Based on these findings we also developed `sunny-as`: a tool for applying SUNNY on a given ASlib scenario. `sunny-as` enables to train and test an AS scenario conforming to ASlib specifications, allowing us to apply different FS methods.

Paper Structure. In Section II we provide some background notions. Section III presents the methodology we followed for conducting the experiments. In Section IV we show the empirical results while in Section V we introduce the `sunny-as` tool. Section VI discusses the related work. Section VII concludes the paper presenting also some future directions.

II. Background

In this section we give some preliminary notions about the ASlib library, the SUNNY algorithm, and the FS algorithms we used in our experiments.

A. Algorithm Selection Library

The Algorithm Selection library (ASlib) [6] provides a standardized format for representing AS scenarios and a repository

Supported by the EU project FP7-644298 *HyVar: Scalable Hybrid Variability for Distributed, Evolving Software Systems*.

[1]The original notion of AS has been afterwards extended by the definition of *Algorithm Portfolio* [10] [19], [24], [33].

Scenario	Algorithms (m)	Problems (n)	Features (d)	Timeout (τ)
ASP-POTASSCO	11	1294	138	600
CSP-2010	2	2024	86	5000
MAXSAT12-PMS	6	876	37	2100
PREMARSHALLING-ASTAR-2013	4	527	16	3600
PROTEUS-2014	22	4021	198	3600
QBF-2011	5	1368	46	3600
SAT11-HAND	15	296	115	5000
SAT11-INDU	18	300	115	5000
SAT11-RAND	9	600	115	5000
SAT12-ALL	31	1614	115	1200
SAT12-HAND	31	767	115	1200
SAT12-INDU	31	1167	115	1200
SAT12-RAND	31	1362	115	1200

TABLE I
ASlib SCENARIOS.

that contains a growing number of datasets from the literature. Its goal is to enable researches and practitioners to effectively share and compare different AS approaches.

Each ASlib scenario contains at least: an algorithm space $\mathcal{A} = \{A_1, \ldots, A_m\}$; a problem space $\mathcal{X} = \{x_1, \ldots, x_n\}$; a feature space $\mathcal{F}_d = \{F_1, \ldots, F_n\}$ where $F_j \in \mathbb{R}^d$ is the feature vector of the problem x_j; a performance space $\mathcal{P}_\tau = \{P_{1,1}, \ldots, P_{m,n}\}$ where $P_{i,j} \in \mathbb{R}$ measures the performance of algorithm A_i on problem x_j within a timeout of τ seconds. Optionally, a scenario may contain a file indicating the cost of extracting a (group of) features. Indeed, feature computation contributes to the total runtime. However, since it is usually a minor overhead, when the feature cost is not specified it is assumed to be 0.

As of today, ASlib contains 13 heterogeneous scenarios[2] that ranges from Answer-Set Programming to Constraint Satisfaction Problems, from Quantified Boolean Formula to Boolean Satisfiability, and so on. As summarized in Table I, the scenarios also vary in the number of algorithms m, problems n, features d, and in the time limits τ. The most represented domain is SAT, with seven scenarios. In all the scenarios, the runtime is the only performance measure. In the rest of the section we briefly describe the ASlib scenarios. For more details, we refer the interested reader to [6].

1) ASP-POTASSCO: This scenario contains a set of Answer Set Programming (ASP) problems and a set of algorithms automatically constructed by an adapted version of Hydra [36]. The features were generated by a lightweight version of clasp, including static and probing features organized into feature groups; they were previously used by claspfolio solver [15].

2) CSP-2010: This scenario contains Constraint Satisfaction Problems and only two solvers: one that employs lazy learning [8] and one that does not [9]. The dataset is heavily biased towards the non-learning solver. Improving on this scenario is a challenging task, since the single best algorithm is very good and both solvers share a common core.

3) MAXSAT12-PMS: MaxSAT aims to maximize the number of satisfied clauses for a SAT problem. In particular this scenario focuses on the partial MaxSAT (PMS) problem [26]. It consists of a collection of random, crafted, and industrial instances from the 2012 MaxSAT Evaluation. It contains only six solvers with very different characteristics.

4) PREMARSHALLING-ASTAR-2013: This scenario contains different real-world and time-sensitive Container Pre-Marshalling (CPM) problems. Built from from two recent A^* and IDA^* approaches [35], it contains four highly homogeneous algorithms. The features provided are new and not as well tested as in the other scenarios.

5) PROTEUS-2014: The PROTEUS scenario [16] includes an extremely diverse mix of well-known CSP solvers alongside competition-winning SAT solvers that can accept different conversions of CSP into SAT. The features include both the SAT and CSP features for a given instance. This potentially provides additional information to AS approaches.

6) QBF-2011: This scenario contains a Quantified Boolean Formula (QBF) dataset coming from the main, small hard, 2QBF and random tracks of QBF Solver Evaluation 2010. Features and solvers are taken from the AQME system and described in more detail in [28]. Although the QBF scenario includes only five algorithms, this set is highly diverse.

7) SAT:* ASlib contains seven scenarios of SAT problems derived from three tracks of SAT competitions 2011/12: industrial (INDU), crafted (HAND) and random (RAND). The SAT scenarios are characterized by a high level of maturity and diversity in terms of solvers, features, and instances.

B. SUNNY

Originally tailored for Constraint Programming only, the SUNNY [2] algorithm can be easily generalized to Algorithm Selection. Fixed a solving timeout τ and a portfolio $\mathcal{A}$ of algorithms, SUNNY exploits instances similarity to produce a sequential schedule $\sigma = [(A_1, t_1), \ldots, (A_h, t_h)]$ where algorithm $A_i \in \mathcal{A}$ has to run for t_i seconds and $\sum_{i=1}^{h} t_i = \tau$. For any input problem x, SUNNY uses a *k-Nearest Neighbours* (k-NN) algorithm to select from a training set of known instances the subset $N(x, k)$ of the k instances closer to the feature vector F_x of x according to the Euclidean distance. Starting from the $N(x, k)$ instances SUNNY relies on three heuristics to compute the schedule σ: *(i)* H_{sel}, for *selecting* the most promising algorithms $\{A_1, \ldots, A_h\} \subseteq \mathcal{A}$ to run; *(ii)* H_{all}, for *allocating* to each $A_i \in \mathcal{A}$ a certain runtime $t_i \in [0, \tau]$

[2]In this paper we consider the 1.0.1 version of ASlib, i.e., the same used in the ongoing evaluation available at http://www.coseal.net/aslib/.

for $i = 1, \ldots, h$; *(iii)* H_{sch}, for *scheduling* the sequential execution of the algorithms according to their presumed speed. The heuristics H_{sel}, H_{all}, and H_{sch} depend on the application domain. For CSPs, H_{sel} selects the smallest sub-portfolio $S \subseteq \mathcal{A}$ that solves the most instances in $N(x, k)$, by using the runtime for breaking ties. H_{all} allocates to each $A_i \in S$ a time t_i proportional to the instances that S can solve in $N(x, k)$, by using a special *backup solver* for covering the instances of $N(x, k)$ not solvable by any solver. Finally, H_{sch} sorts the solvers by increasing solving time in $N(x, k)$. For COPs the approach is similar, but different evaluation metrics are used. Example 1 shows a running example of how SUNNY works on a given CSP; for more details about SUNNY we refer the interested reader to [2], [4].

Example 1 *Let x be a CSP, $\mathcal{A} = \{A_1, A_2, A_3, A_4\}$ a portfolio, A_3 the backup solver, $\tau = 1800$ seconds the solving timeout, $N(x, k) = \{x_1, ..., x_5\}$ the $k = 5$ neighbours of x, and the runtimes of solver A_i on problem x_j defined as in Table II. In this case, the smallest sub-portfolios that solve the most instances (4 to be precise) in $N(x, k)$ 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 $N(x, k)$. 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 the solvers by increasing solving time. The final schedule produced by SUNNY is therefore $\sigma = [(A_4, 600), (A_1, 600), (A_3, 300), (A_2, 300)]$.*

	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 II
RUNTIMES (IN SECONDS). τ MEANS THE SOLVER TIMEOUT.

C. Feature Selection

Features are of paramount importance for characterizing a given problem instance and building accurate predictions models. For this reason, there is an extensive literature concerning the problem of selecting a suitable subset of the most informative features [18], [25]. This process, typically referred as Feature Selection (FS), has the purpose of removing those features that are redundant, irrelevant, and potentially harmful. Selecting a minimal set of features has the advantage of simplifying the prediction model, lowering the training and feature extraction costs, reducing the potential overfitting and hopefully improving the prediction accuracy [29].

FS techniques [11] consists basically of a combination of two ingredients: a search technique for finding good subsets of features, and a scoring metric to evaluate such subsets. Since exploring all the possible subsets of features is computationally intractable for non-trivial feature spaces, heuristics are employed to guide the search of the best subsets. Greedy search strategies usually come in two flavors: forward selection and backward elimination. In forward selection, features are progressively incorporated into larger and larger subsets. Conversely, in backward elimination features are progressively removed starting from all the available features. Combination of these two techniques, genetic algorithms, or local search algorithms such as simulated annealing are also used.

FS approaches can be distinguished in mainly two categories: wrappers and filters. Filter methods select the features regardless of the model, trying to suppress the least interesting ones. These methods are particularly efficient and robust to overfitting. In contrast, wrappers evaluate subsets of features possibly detecting the interactions between features. Wrappers methods can be more accurate than filters, but have two main disadvantages: they are more exposed to the overfitting risk, and they have a much higher computational cost.

In this work we chose to focus on filter methods only. We made this decision since we wanted to test general and lightweight FS approaches, trying to be as free as possible from overfitting and high training costs. This choice also met the rules of the ICON Challenge on Algorithm Selection [20] that impose a maximal training time of 12 CPU hours.[3] In particular, we used two standard search methods: *BestFirst* and *Ranker*. BestFirst searches the feature subsets by greedy hill climbing augmented with a backtracking facility. In this paper we tested both the canonical forward selection and the backward selection, where backtracking is used if necessary. Ranker instead ranks the features by their individual evaluation. For BestFirst we used the *Correlation-based Feature Selection* (CFS) evaluator, which computes the worth of a subset of features by considering the individual predictive ability of each feature, along with the degree of redundancy between them [13]. For Ranker we used the following well-known evaluators:

- *Information Gain* (InfoGain): Evaluates the worth of an attribute by measuring the information gain with respect to the class:[4]

$$\text{InfoGain}(\text{Class}, \text{Feat}) = H(\text{Class}) - H(\text{Class} \mid \text{Feat})$$

  where $H(\text{Class})$ and $H(\text{Feat})$ are the entropies based on the probability associated with each class and feature respectively, and $H(\text{Class} \mid \text{Feat})$ is the entropy of Class conditioned on Feat.
- *Gain Ratio* (GainR): Measures the gain ratio w.r.t. the class:

$$\text{GainR}(\text{Class}, \text{Feat}) = \frac{\text{InfoGain}(\text{Class}, \text{Feat})}{H(\text{Feat})}$$

- *Symmetrical Uncertainty* (SymmU): Measures the symmetrical uncertainty w.r.t. the class:

$$\text{SymmU}(\text{Class}, \text{Feat}) = 2 \times \frac{H(\text{Class}) - H(\text{Class} \mid \text{Feat})}{H(\text{Class}) + H(\text{Feat})}$$

[3] We noticed that for some ASlib scenarios the use of a wrapping method required days of computation, even with the simplest greedy search strategies.

[4] In our case, the class is the best available algorithm for a given instance, i.e., the one having the lower runtime.

- *Relief*: Evaluates the worth of an attribute by repeatedly sampling an instance and considering the value of the given attribute for the nearest instance of the same and different class. For more details, see [22], [23], [31]
- *OneR*: Evaluates the worth of an attribute by means of 1R, a classifier using the minimum-error attribute for predictions, discretizing numeric attributes [14].

All the above mentioned approaches have been implemented by means of WEKA [12], a well-known tool for Machine Learning and data mining tasks.

III. Methodology

In this Section we explain the methodology we followed for adapting SUNNY to the different ASlib scenarios and for evaluating the FS approaches above introduced.

A. SUNNY Adaptation

Adapting SUNNY to ASlib scenarios was rather straightforward. Fixed a training set $\mathcal{X}_{tr} \subseteq \mathcal{X}$ of known instances and a corresponding feature space $\mathcal{F}_{tr}$, we normalized every feature vector by removing all the constant features of $\mathcal{F}_{tr}$ and by scaling each of them in the range $[-1, 1]$. Then, for each unknown problem $x \notin \mathcal{X}_{tr}$, SUNNY computes the neighborhood $N(x, k) \subseteq \mathcal{X}_{tr}$ and the resulting schedule $\sigma = [(a_1, t_1), \ldots, (a_h, t_h)]$ exactly as explained in Section II-B. In particular, since the only performance measure of ASlib scenarios is the runtime, the methodology is the same of Example 1. Following the methodology of [5], we set $k = \sqrt{|\mathcal{X}_{tr}|}$ and the backup solver as the algorithm of $\mathcal{A}$ having the lower average runtime in $\mathcal{X}_{tr}$.

Each scenario of the ASlib is evaluated with a *10-fold cross validation*: the dataset $\mathcal{X}$ is partitioned in 10 disjoint subsets $\mathcal{X}_1, \ldots, \mathcal{X}_{10}$ called folds, treating in turn a fold $\mathcal{X}_i$ as the test set and the union $\bigcup_{j \neq i} \mathcal{X}_j$ of other folds as the training set.

B. Performance Evaluation

Different metrics can be used for evaluating the performance of an AS approach on a given scenario. The *Fraction of Solved Instances* (FSI) of an approach is the ratio between the number of instances it solves and all the instances of the scenario. This metric is commonly used for comparing different AS approaches due to its simplicity and significance. However, it does not take into account the time needed to solve a problem. To capture also the timing aspects of the resolution, the *Penalized Average Runtime* (PAR) measure is often used. Fixed an AS scenario and a positive integer k, $\mathsf{PAR}_k(S)$ represents the average time taken by AS approach S to solve all the problems of the scenario, where a penalization of $k \times \tau$ seconds is given for the instances not solved by S within the timeout τ.

Unfortunately, despite FSI and PAR_k can be suitable metrics for measuring the AS quality, when more than one scenario is considered their average values may be no longer significant. For example, putting together and averaging the PAR_k values of ASP-POTASSCO and CSP-2010 is not meaningful because the different time limits (τ is 600 and 5000 respectively) also implies a different penalization $k \times \tau$ for the instances not solved within τ seconds. Moreover, even if the FSI value is always in the $[0, 1]$ range, comparing the average FSI of different scenarios may be not fair. For example, a FSI of 0.5 can be very good or very bad depending on the algorithms and the problems of a scenario since almost every scenario of ASlib contains instances which are not solvable by any algorithm. A normalization is therefore needed to keep the performance of different scenarios on the same scale.

Let us define the *Virtual Best Solver* (VBS) of a scenario as the oracle selector that for every instance $x \in \mathcal{X}$ always chooses the algorithm $A \in \mathcal{A}$ having the lower runtime on x. If $\mathsf{FSI}(S)$ is the fraction of solved instances of an AS approach S on a given scenario, we define the *Normalized Fraction of Solved Instances* $\overline{\mathsf{FSI}}$ as:

$$\overline{\mathsf{FSI}}(S) = \max\left\{0,\ 1 - \frac{\mathsf{FSI}(VBS) - \mathsf{FSI}(S)}{\mathsf{FSI}(VBS) - \mathsf{FSI}(SBS)}\right\}$$

where $\mathsf{FSI}(SBS)$ is the FSI of the *Single Best Solver* (SBS), i.e., the algorithm $A \in \mathcal{A}$ that solves the most instances of the dataset $\mathcal{X}$. In this way, for any given scenario and AS approach S, $\overline{\mathsf{FSI}}(S)$ is always a value in $[0, 1]$ linearly proportional to the performance quality of S. In particular, $\overline{\mathsf{FSI}}(S) = 0$ indicates a performance worse than or equal to the SBS, while $\overline{\mathsf{FSI}}(S) = 1$ if S reaches the perfect performance (i.e, that of the VBS). Moreover, the relative scenario performance is always preserved: for every distinct approaches S, S' we have that $\mathsf{FSI}(S) \leq \mathsf{FSI}(S') \iff \overline{\mathsf{FSI}}(S) \leq \overline{\mathsf{FSI}}(S')$.

Analogously, set the SBS as the algorithm $A \in \mathcal{A}$ having the lower average runtime over all the instances of $\mathcal{X}$, we define the *Normalized Penalized Average Runtime* $\overline{\mathsf{PAR}}_k$ of an AS approach S on a given scenario as:

$$\overline{\mathsf{PAR}}_k(S) = \max\left\{0,\ 1 - \frac{\mathsf{PAR}_k(VBS) - \mathsf{PAR}_k(S)}{\mathsf{PAR}_k(VBS) - \mathsf{PAR}_k(SBS)}\right\}$$

Recalling that it is outside the scope of this paper to provide a comparison of SUNNY against other AS approaches, in Table III we however show the $\overline{\mathsf{PAR}}_{10}$ score of SUNNY against other state-of-the-art AS approaches (viz., ISAC, SNNAP, aspeed, claspfolio, claspfolio-pre, zilla, and LLAMA). These values are obtained from the PAR_{10} results of [1]. As can be seen, SUNNY appears to be promising for non-SAT scenarios (in particular QBF and Proteus) while for SAT problems there is a clear performance gap w.r.t. the best approaches. Curiously, by considering the average $\overline{\mathsf{PAR}}_{10}$ over all the scenarios, the best approach turns out to be claspfolio with pre-solving, even if this approach is never the best one for any given scenario.

C. Feature Selection

As mentioned in Section II-C, the FS algorithms we tested in the experiments have been evaluated by exploiting their implementation in WEKA. We used in particular the `AttributeSelection` class, a supervised attribute filter that allows to combine search and evaluation methods. Concerning BestFirst method, we used the class `BestFirst` for

Scenario	ISAC	SNNAP	aspeed	claspfolio	claspfolio-pre	zilla	LLAMA	SUNNY
ASP-POTASSCO	0.472	0.644	0.353	**0.819**	0.800	0.710	0.773	0.690
CSP-2010	0.062	0.000	0.045	0.805	0.793	0.828	**0.857**	0.681
MAXSAT12-PMS	0.640	0.588	0.648	0.891	0.716	**0.939**	0.864	0.871
PREMARSHALLING-ASTAR-2013	0.166	0.000	**0.744**	0.292	0.680	0.564	0.350	0.706
PROTEUS-2014	0.713	0.644	0.865	0.789	0.848	0.808	0.790	**0.877**
QBF-2011	0.590	0.197	0.803	0.858	0.892	0.873	0.894	**0.896**
SAT11-HAND	0.189	0.474	**0.729**	0.624	0.712	0.655	0.711	0.516
SAT11-INDU	0.000	0.184	0.000	**0.356**	0.346	0.000	0.251	0.092
SAT11-RAND	0.777	0.646	0.776	0.926	0.930	**0.950**	0.892	0.534
SAT12-ALL	0.000	0.536	0.095	0.693	0.675	0.728	**0.740**	0.581
SAT12-HAND	0.000	0.458	0.473	0.706	0.702	0.759	**0.779**	0.572
SAT12-INDU	0.000	0.449	0.000	**0.586**	0.446	0.409	0.410	0.005
SAT12-RAND	0.257	0.000	0.000	**0.380**	0.274	0.160	0.305	0.000
Average all scenarios	0.297	0.371	0.426	0.671	**0.678**	0.645	0.663	0.540
Average non-SAT scenarios	0.441	0.345	0.576	0.742	**0.788**	0.787	0.755	0.787
Average SAT scenarios	0.175	0.392	0.296	**0.610**	0.584	0.523	0.584	0.329

TABLE III
PENALIZED AVERAGE RUNTIME.

search and class `CfsSubsetEval` as evaluator. Regarding Ranker method, we used `Ranker` class for search and the following classes for the evaluation:

- `InfoGainAttributeEval`
- `GainRatioAttributeEval`
- `SymmetricalUncertAttributeEval`
- `ReliefFAttributeEval`
- `OneRAttributeEval`

Each WEKA algorithm has been evaluated by using its default parameters configuration, possibly tuning the `-N` parameter for varying the number of selected features (see Section IV).

All the experiments have been performed on Intel Dual-Core 2.93GHz computers with 3 MB of CPU cache, 2 GB of RAM, and Debian 3.2.65 operating system. The source code and the results of the experiments are publicly available at https://github.com/lteu/sunny-fs.

IV. RESULTS

In this section we present the experimental results we obtained following the methodology described in Section III. In particular, we measured the performance of the different FS approaches in terms of the $\overline{\mathsf{FSI}}$ and $\overline{\mathsf{PAR}}_{10}$ metrics.

We first show the performance of BestFirst method with Forward and Backward heuristics. As can be seen from Table IV, improvements are minimal both in terms of $\overline{\mathsf{PAR}}_{10}$ and $\overline{\mathsf{FSI}}$. Indeed, with the sole exception of PREMARSHALLING scenario, the Forward method is always worse than the original SUNNY approach exploiting all the available features. Slightly better is the Backward approach, which improves the original performance also in the QBF and SAT11-INDU scenarios. Not surprisingly, due to the pronounced anti-correlation between FSI and PAR_{10}, the corresponding normalised values $\overline{\mathsf{FSI}}$ and $\overline{\mathsf{PAR}}_{10}$ are proportional: the higher is the $\overline{\mathsf{FSI}}$, the higher is the $\overline{\mathsf{PAR}}_{10}$ (and viceversa).

Better results are instead achieved by using Ranker. Table V depicts the peak $\overline{\mathsf{PAR}}_{10}$ performance of the FS approaches introduced in Section II-C against the original SUNNY approach.[5] Here we evaluated every method by varying the number of selected features in $\{1, 2, \ldots, 8\} \cup \{16, 32, 64\}$. Several interesting insights can be observed. Firstly, for all the considered scenarios there is always a FS method able to outperform the original SUNNY approach exploiting all the available features. The best feature evaluator in this regard appears to be Relief, which reaches the best $\overline{\mathsf{PAR}}_{10}$ for 7 scenarios out of 13. However, there are no dominated evaluators since every method reaches the peak performance for at least one scenario. The "# Features" column represents the number of selected features for the best approach in the form $d \rightarrow d'$ where d is the original number of features and d' the resulting number of selected features. We can note that for almost all the scenarios less than eight features are enough for outperforming the original SUNNY approach, with a reduction factor which is often above 90%. The relative performance gain[6] can be considerable (sometimes also beyond 100%).

Table VI reports the average $\overline{\mathsf{PAR}}_{10}$ performance of each FS approach over all the scenarios by varying the number of selected features in $\{1, 2, \ldots, 8\} \cup \{16, 32, 64\}$. The results point out that there is not a clearly dominant FS method. Relief is still the approach that reaches the peak performance the most number of times (i.e., with $1, 2, 4, 16$, and 32 features). However, the overall best $\overline{\mathsf{PAR}}_{10}$ score (0.605) is reached by InfoGain with just 5 features.

We conclude this section with some considerations about how many and which features to select for a given scenario. Unfortunately, as shown in Table V, there is not a clear pattern indicating what is the number of features to select. Although in general SUNNY needs only few features (often no more than five, see "# Features" column) for reaching a good performance, we cannot identify a clear trend in the number of selected features even when the application domain is the same (e.g., see the number of features for SAT scenarios).

[5] We do not report the $\overline{\mathsf{FSI}}$ results since, as mentioned above, $\overline{\mathsf{FSI}}$ results are strongly correlated to $\overline{\mathsf{PAR}}_k$ results.

[6] In Table V the relative performance gain is defined as $100 \times \frac{P' - P}{P}$ where P and P' are respectively the $\overline{\mathsf{PAR}}_{10}$ of the original approach and the $\overline{\mathsf{PAR}}_{10}$ of the best FS approach.

	PAR_{10}		$\overline{\mathsf{PAR}}_{10}$			FSI		$\overline{\mathsf{FSI}}$		
Scenario	*VBS*	*SBS*	Original	Forward	Backward	*VBS*	*SBS*	original	Forward	Backward
ASP-POTASSCO	400.2	880.5	0.690	0.676	0.671	0.859	0.937	0.692	0.679	0.667
CSP-2010	6344.3	7201.6	0.681	0.659	0.598	0.858	0.875	0.706	0.647	0.588
MAXSAT12-PMS	3127.2	4893.1	0.871	0.828	0.871	0.769	0.853	0.869	0.833	0.869
PREMARSHALLING-ASTAR-2013	227.6	7002.9	0.706	**0.752**	**0.752**	0.812	1	0.729	**0.777**	**0.777**
PROTEUS-2014	4105.9	13443.4	0.877	0.833	0.874	0.628	0.887	0.892	0.865	0.892
QBF-2011	8337.1	15330.2	0.896	0.835	**0.907**	0.577	0.77	0.917	0.876	**0.927**
SAT11-HAND	13360.7	25649.1	0.516	0.489	0.494	0.497	0.74	0.514	0.486	0.486
SAT11-INDU	8187.5	14605.9	0.092	0.066	**0.109**	0.717	0.843	0.103	0.079	**0.127**
SAT11-RAND	9186.4	19916.4	0.928	0.907	0.896	0.603	0.82	0.931	0.908	0.899
SAT12-ALL	241.3	3079.9	0.581	0.473	0.552	0.753	0.988	0.596	0.485	0.562
SAT12-HAND	3662.2	6338.9	0.572	0.484	0.558	0.477	0.701	0.585	0.496	0.567
SAT12-INDU	2221.5	3266	0.052	0	0	0.736	0.821	0.082	0	0
SAT12-RAND	2872.8	3271.1	0	0	0	0.731	0.764	0	0	0
Average	4790.362	9606.077	0.574	0.539	0.560	0.694	0.846	0.586	0.548	0.566

TABLE IV
BESTFIRST FORWARD AND BACKWARD PERFORMANCE.

Scenario	Original	SymmU	GainRatio	InfoGain	Relief	OneR	# Features	Reduction [%]	Gain [%]
ASP-POTASSCO	0.690	0.700	0.700	0.700	**0.710**	0.611	138 → **5**	**96.377**	2.888
CSP-2010	0.681	**0.795**	**0.795**	0.717	0.739	0.785	86 → **5**	**94.186**	16.770
MAXSAT12-PMS	0.871	0.874	0.874	0.874	**0.913**	0.831	37 → 16	56.757	4.835
PREMARSHALLING-ASTAR-2013	0.706	0.781	0.766	0.785	**0.794**	0.781	16 → **2**	87.500	12.458
PROTEUS-2014	0.877	**0.895**	0.852	0.893	0.886	0.876	198 → 7	**96.465**	2.080
QBF-2011	0.896	0.910	0.910	0.914	**0.922**	0.897	46 → **3**	**93.478**	2.942
SAT11-HAND	0.516	**0.602**	0.588	**0.602**	0.596	0.524	115 → **5**	**95.652**	16.724
SAT11-INDU	0.092	0.244	0.244	0.244	0.220	**0.256**	115 → **2**	**98.261**	**178.022**
SAT11-RAND	0.928	0.931	0.932	0.932	**0.939**	0.924	115 → 16	86.087	1.222
SAT12-ALL	0.581	**0.610**	0.557	0.602	0.601	0.586	115 → 32	72.174	5.002
SAT12-HAND	0.572	**0.623**	0.598	**0.623**	0.593	0.597	115 → **5**	**95.652**	8.956
SAT12-INDU	0.052	0.226	0.228	0.224	**0.238**	0.181	115 → **3**	**97.391**	**359.510**
SAT12-RAND	0	0	0.010	0.024	**0.090**	0	115 → 7	**93.913**	∞

TABLE V
PEAK $\overline{\mathsf{PAR}}_{10}$ PERFORMANCE FOR EACH RANKER SELECTOR.

FS Approach	1	2	3	4	5	6	7	8	16	32	64
SymmU	0.486	0.546	**0.583**	0.582	0.599	0.591	**0.599**	0.571	0.567	0.568	0.480
GainRatio	0.412	0.472	0.521	0.549	0.572	0.581	0.587	**0.577**	0.575	0.553	0.476
InfoGain	0.496	0.548	0.566	0.569	**0.605**	**0.595**	0.595	0.574	0.569	0.575	**0.482**
Relief	**0.531**	**0.550**	0.563	**0.587**	0.598	0.581	0.574	0.565	**0.587**	**0.590**	0.492
OneR	0.518	0.570	0.560	0.547	0.570	0.572	0.571	0.573	0.576	0.576	0.579
Maximum	0.531	0.550	0.583	0.587	**0.605**	0.595	0.599	0.577	0.587	0.590	0.492

TABLE VI
AVERAGE $\overline{\mathsf{PAR}}_{10}$ PERFORMANCE OF RANKER OVER ALL THE SCENARIOS.

If we focus instead on which features allow to reach the best performance, the results obviously vary across the scenarios. Table VII lists the features selected by the best tested FS method reported in Table V. We report these features for two reasons. On the one hand, these reduced sets of features enable researchers to further assess and compare the performance of other AS and FS approaches over the different scenarios. On the other hand, such features might help a domain expert to deepen the study of the most informative features for a particular scenario. For more details about scenarios and features we refer the reader to [6] (and to the papers describing the scenarios listed in Section II-A).

V. `SUNNY-AS`

Based on the results shown in Section IV, we developed `sunny-as`: an AS utility tool that uses the SUNNY algorithm for evaluating scenarios conforming to the ASlib format specifications. According to the `cv.arff` file contained in every scenario, `sunny-as` allows to split the original scenario into $h \times k$ training/testing scenarios, where h is the number of repetitions and k the number of folds of the corresponding k-cross validation. `sunny-as` can be used in three different modes: training, pre-solving and testing.

In the training phase, `sunny-as` extracts some basic information from the training scenario (e.g., the portfolio, the backup solver, the timeout) and builds the knowledge base needed by SUNNY for computing the algorithms schedule as explained in Section II-B. The knowledge base is automatically created by processing the meta-information of the scenario (e.g., runtimes, feature values, feature costs, etc.). Basically, it is a map that associates to every training instance its normalized feature vector (where constant features are removed and values are scaled in the range $[-1, 1]$) and the runtimes of each solver of the portfolio on such instance. `sunny-as` also allows to set different options, e.g., the range of the feature values or the timeout for feature extraction. Note that —since SUNNY is a lazy approach— this phase is more a setup than a training, since no prediction model is explicitly learned.

Optionally, after the training phase the user can define

Scenario	Features
ASP-POTASSCO	Frac_Body-Body_Equivalences, Frac_Other_Equivalences, Tight, Frac_Atom-Atom_Equivalences, Literals_in_Conflict_Nogoods-1
CSP-2010	stats_sums_count, stats_var_bool, stats_edge_density, stats_branchingvars, stats_varcount
MAXSAT12-PMS	pnr_var_max, pnr_cls_std, pnr_var_mean, numClauses, pnr_cls_mean, pnr_var_spread, horn, binary, pnr_var_min,var_clauses_ratio, perc_soft, trinary, unary, pnr_var_std, vcg_var_std, vcg_cls_min
PREMARSH.	container-density, tiers
PROTEUS-2014	csp_perten_avg_predshape, csp_perten_avg_predsize, csp_sqrt_max_domsize, csp_sqrt_avg_domsize, csp_percent_dec_predicate, directorder_POSNEG-RATIO-CLAUSE-mean, directorder_reducedVars
QBF-2011	NEG_HORN_CLAUSE, TERMORE_CLAUSE, OCCFP_OCCFN
SAT11-HAND	BINARYp, horn_clauses_fraction, SP_bias_q25, VCG_CLAUSE_coeff_variation, lobjois_mean_depth_over_vars
SAT11-INDU	SP_unconstraint_q10, VCG_CLAUSE_coeff_variation
SAT11-RAND	cl_num_q90, cl_num_max, cl_num_q75, cl_num_mean, cl_num_q50, cl_num_q25, cl_num_min, cl_num_q10, VG_max, DIAMETER_entropy, cl_size_q10, cl_size_q25, VG_mean, DIAMETER_coeff_variation, cl_num_coeff_variation, cl_size_mean
SAT12-ALL	SP_unconstraint_q25, vars_clauses_ratio, SP_unconstraint_mean, POSNEG_RATIO_CLAUSE_entropy, VCG_VAR_coeff_variation, SP_unconstraint_q75, SP_bias_coeff_variation, SP_bias_mean, BINARYp, reducedVars, SP_unconstraint_max, VCG_CLAUSE_entropy, POSNEG_RATIO_CLAUSE_coeff_variation, cl_num_q75, cl_num_q50, cl_num_mean, cl_num_q90, SP_unconstraint_q90, cl_num_q25, VG_coeff_variation, SP_bias_q25, SP_bias_q50, VCG_CLAUSE_coeff_variation, nclausesOrig, HORNY_VAR_coeff_variation, SP_unconstraint_coeff_variation, gsat_FirstLocalMinRatio_Mean, VCG_VAR_entropy, DIAMETER_max, nvarsOrig, POSNEG_RATIO_CLAUSE_min, nclauses
SAT12-HAND	SP_bias_coeff_variation, reducedClauses, horn_clauses_fraction, SP_unconstraint_max, POSNEG_RATIO_CLAUSE_min
SAT12-INDU	BINARYp, vars_clauses_ratio, SP_bias_q50
SAT12-RAND	cl_num_max, cl_num_q90, cl_num_q75, cl_num_mean, cl_num_q50, cl_num_q25, cl_num_q10

TABLE VII
FEATURES SELECTED BY THE BEST FS METHODS.

also a presolving phase in which it is possible to filter the initial portfolio, set a static schedule, and in particular perform a selection over all the original features. For example, by defining the corresponding `sunny-as` options the user can apply all the FS algorithms listed in Section III-C.

Finally, the testing scenario is used for computing and evaluating the SUNNY predictor. A script is available for printing (or redirecting) the SUNNY schedule for every test instance, according to the ASlib specifications. The arguments used by SUNNY are those computed in the previous steps. However, `sunny-as` allows to override them by modifying for example the static schedule, the neighborhood size, the portfolio, the backup solver, the timeout, and even the knowledge base itself. `sunny-as` also provides a utility for making statistics on the predictions (e.g., computing the PAR_{10} and FSI measures).

`sunny-as` is completely written in Python and publicly available at https://github.com/CP-Unibo/sunny-as. It participated in the first ICON Challenge on Algorithm Selection [20] using InfoGain to select the best 5 features. This version was the best algorithm selector for PREMARSHALLING scenario, but it globally achieved rather poor results. We are investigating the reasons behind such performance, especially for SAT scenarios.

VI. RELATED WORK

The interest in algorithm selection is quite general and growing. For more comprehensive surveys about AS problem and related paradigms, we refer the interested reader to [19], [24], [32]. In this context we report just the AS related works dealing with feature selection.[7]

A preliminary investigation on ASlib is performed in [6]. In particular, forward selection is used to select the best features and algorithms for each scenario. In contrast to our approach, the authors use a wrapper with a random regression forest to evaluate the features/algorithms subset. Their findings are however similar to ours: usually the number of required features is very small if compared to the complete feature set (at most five features per scenario are selected). Interestingly, also the number of algorithms is substantially reduced in most scenarios. The possibility of considering just a small number of algorithms may lead to performance improvements also for SUNNY, especially when it comes to compute the best sub-portfolios. However, due to the fact that the use of wrapper methods for performing the algorithm selection was far more computationally expensive for SUNNY, in this paper we focus only in feature selection and not in algorithm filtering.

In [7] forward feature selection is used for predicting the runtime of planners. In this case too, authors show that only 4-6 features (among 311 features in total) may lead to a performance similar to that obtained by using all the available features. In [25] feature filtering has been applied to the Instance-Specific Algorithm Configuration (ISAC) tool, proving that in SAT and CSP domains the number of features used by ISAC can be reduced to less than a quarter, often providing significant performance gains. In [18] empirical analyses on SAT, MIP, and TSP problems show that just two key algorithm parameters or, similarly, few instance features suffice to predict the most salient algorithm performance measures.

VII. CONCLUSIONS

In this paper we examined the impact of Feature Selection (FS) techniques on the performance of the SUNNY algorithm selector. We use the benchmarks of the *Algorithm Selection library* (ASlib) [6], a repository including different AS scenarios encoded in a standardized format. In particular, we evaluated whether and how different off-the-shelf FS approaches applied to SUNNY algorithm can improve the original SUNNY approach (which uses by default all the available features). To the best of our knowledge, except for the preliminary investigation of [6], no similar analysis on the ASlib scenarios have been

[7] An updated list of papers dealing with algorithm selection is available at 4c.ucc.ie/~larsko/assurvey/

performed yet.

We evaluated and compared the different FS methods on all the heterogeneous scenarios of the ASlib in terms of two normalized metrics, $\overline{\mathsf{FSI}}$ and $\overline{\mathsf{PAR}}_{10}$, that we introduced for avoiding biases when considering scenarios with different characteristics. Empirical results indicate that —although there is not a clearly dominant FS method— a handful of features is often enough for SUNNY to reach a similar if not better performance of the default approach. Unfortunately we are not currently able to extrapolate meaningful and general patterns for detecting *ex-ante* the number and the best features for a given scenario. Finally, we also developed `sunny-as`: a tool for applying SUNNY on a given ASlib scenario.

From a qualitative point of view, understanding and analyzing which are the most significant features is a major challenge that deserves to be explored in the future. However, we believe that even quantitative and empirical investigations —as those described in this paper— are worth to follow for avoiding the use of redundant information, focusing on a restricted but still meaningful set of features.

As an extension of this work, additional scenarios (e.g., optimization and planning problems), performance measures, and FS approaches may be tested. The `sunny-as` tool we developed should facilitate the evaluation and the comparison of SUNNY using different AS scenarios and FS methods. For example, we are interested in comparing the off-the-shelf FS approaches of this paper with the best ones of the ICON challenge [20] and ASlib [6].

Furthermore, by tuning the parameters of `sunny-as` it is also possible to try and test different SUNNY configurations (e.g., by adding a static schedule, by varying the portfolio composition, or by changing the neighborhood size). In this context, automatic configurators [17], [21] can be used for tuning the parameters.

An interesting future direction concerns the development of dynamic AS approaches. Indeed, since for the majority of the scenarios just few static or probing features are enough to reach the best results, following approaches such as [34] it might be possible to define dynamic AS approaches that only rely on the observed runtime behavior of the algorithms.

REFERENCES

[1] R. Amadini, F. Biselli, M. Gabbrielli, T. Liu, and J. Mauro. SUNNY for Algorithm Selection: A Preliminary Study. In *CILC*, 2015. Available at: http://www.cs.unibo.it/~amadini/cilc_2015.pdf.

[2] R. Amadini, M. Gabbrielli, and J. Mauro. SUNNY: a Lazy Portfolio Approach for Constraint Solving. *TPLP*, 14(4-5):509–524, 2014.

[3] R. Amadini, M. Gabbrielli, and J. Mauro. A multicore tool for constraint solving. In *IJCAI*, pages 232–238. AAAI Press, 2015.

[4] R. Amadini, M. Gabbrielli, and J. Mauro. Portfolio approaches for constraint optimization problems. *AMAI*, pages 1–18, 2015.

[5] R. Amadini, M. Gabbrielli, and J. Mauro. SUNNY-CP: a sequential CP portfolio solver. In *SAC*, pages 1861–1867. ACM, 2015.

[6] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, and J. Vanschoren. Aslib: A benchmark library for algorithm selection. *arXiv preprint arXiv:1506.02465*, 2015.

[7] C. Fawcett, M. Vallati, F. Hutter, J. Hoffmann, H. H. Hoos, and K. Leyton-Brown. Improved features for runtime prediction of domain-independent planners. In *ICAPS*. AAAI, 2014.

[8] I. P. Gent, C. Jefferson, L. Kotthoff, I. Miguel, N. C. A. Moore, P. Nightingale, and K. E. Petrie. Learning when to use lazy learning in constraint solving. In *ECAI*, pages 873–878, 2010.

[9] I. P. Gent, C. Jefferson, and I. Miguel. Minion: A fast scalable constraint solver. In *ECAI*, pages 98–102, 2006.

[10] C. P. Gomes and B. Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.

[11] I. Guyon and A. Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[12] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1), Nov. 2009.

[13] M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[14] R. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

[15] H. Hoos, M. T. Lindauer, and T. Schaub. claspfolio 2: Advances in algorithm selection for answer set programming. *TPLP*, 14(4-5):569–585, 2014.

[16] B. Hurley, L. Kotthoff, Y. Malitsky, and B. O'Sullivan. Proteus: A Hierarchical Portfolio of Solvers and Transformations. In *CPAIOR*, volume 8451 of *LNCS*, pages 301–317. Springer, 2014.

[17] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION*, volume 6683 of *LNCS*, pages 507–523. Springer, 2011.

[18] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Identifying Key Algorithm Parameters and Instance Features Using Forward Selection. In *LION*, volume 7997 of *LNCS*, pages 364–381. Springer, 2013.

[19] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm Runtime Prediction: The State of the Art. *CoRR*, abs/1211.0906, 2012.

[20] ICON Challenge on Algorithm Selection. http://iconchallenge.insight-centre.org/challengeas.

[21] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney. ISAC - Instance-Specific Algorithm Configuration. In *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2010.

[22] K. Kira and L. A. Rendell. A practical approach to feature selection. In D. H. Sleeman and P. Edwards, editors, *9th International Workshop on Machine Learning*, pages 249–256. Morgan Kaufmann, 1992.

[23] I. Kononenko. Estimating attributes: Analysis and extensions of relief. In F. Bergadano and L. D. Raedt, editors, *ECML*, pages 171–182. Springer, 1994.

[24] L. Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.

[25] C. Kroer and Y. Malitsky. Feature Filtering for Instance-Specific Algorithm Configuration. In *ICTAI*, pages 849–855. IEEE, 2011.

[26] C. M. Li and F. Manyà. Maxsat, hard and soft constraints. In *Handbook of Satisfiability*, pages 613–631. 2009.

[27] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. MiniZinc: Towards a Standard CP Modelling Language. In *CP*, 2007.

[28] L. Pulina and A. Tacchella. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, 14(1):80–116, 2009.

[29] J. Reunanen. Overfitting in Making Comparisons Between Variable Selection Methods. *Journal of Machine Learning Research*, 3:1371–1382, 2003.

[30] J. R. Rice. The Algorithm Selection Problem. *Advances in Computers*, 15:65–118, 1976.

[31] M. Robnik-Sikonja and I. Kononenko. An adaptation of relief for attribute estimation in regression. In D. H. Fisher, editor, *ICML*, pages 296–304. Morgan Kaufmann, 1997.

[32] K. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1), 2008.

[33] K. A. Smith-Miles. Towards insightful algorithm selection for optimisation using meta-learning concepts. In *IJCNN*, pages 4118–4124. IEEE, 2008.

[34] K. Stergiou. Heuristics for dynamically adapting propagation in constraint satisfaction problems. *AI Commun.*, 22(3):125–141, 2009.

[35] K. Tierney and Y. Malitsky. An algorithm selection benchmark of the container pre-marshalling problem. In *LION 9*, pages 17–22, 2015.

[36] L. Xu, H. Hoos, and K. Leyton-Brown. Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In *AAAI*, 2010.