SUNNY for Algorithm Selection: A Preliminary Study

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

SUNNY for Algorithm Selection: A Preliminary Study

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

Department of Computer Science and Engineering
University of Bologna, Italy.

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. In this paper we show how we adapted the algorithm selector SUNNY, originally tailored for constraint solving, to deal with general AS problems. Preliminary investigations based on the AS Library benchmarks already show some promising results: for some scenarios SUNNY is able to outperform AS state-of-the-art approaches.

1 Introduction

Given a collection of algorithms, the *Algorithm Selection* (AS) problem basically consists in identifying which of them is the best one for solving a given problem. Initially proposed by Rice in 1976 [9], in the last decade AS has attracted some attention [7, 10]. In particular, the original notion of AS has been extended by the definition of *Algorithm Portfolio* (AP) [6]. In a nutshell, AP approaches exploit a portfolio $\{A_1, \ldots, A_m\}$ of different algorithms to get a globally better algorithm. They go beyond the original notion of AS introduced by Rice since APs perform the algorithm selection case-by-case instead of in advance. When a new, unseen problem p comes, an AP approach tries to predict which is (or which are) the best constituent algorithm(s) $A_{i_1}, A_{i_2}, \ldots, A_{i_k}$, with $1 \leq i_j \leq m$, for solving p and then runs such algorithm(s) on p. Scheduling $k > 1$ algorithms can reduce the risk of selecting only one algorithm —maybe the wrong one— and possibly enables the knowledge sharing between the scheduled algorithms. However, note that the boundary between AS and AP is fuzzy: these two related problems are often considered as equivalent. For this reason, with a little abuse of notation, in the following we will only use the AS notation for indicating both AS and AP problems.

SUNNY is an algorithm selector tailored for *Constraint Programming* (CP), where the algorithms to be selected correspond to different constraint solvers. Originally conceived for solving Constraint Satisfaction Problems (CSPs) only [1], it has been later on adapted for dealing with Constraint Optimisation Problems (COPs) [3]. SUNNY is also the algorithm that underpins `sunny-cp` [4], a constraint solver exploiting a portfolio of different constituent solvers for solving both CSPs and COPs.

In this paper we present a preliminary evaluation of SUNNY on different AS benchmarks taken from the *Algorithm Selection library* (ASlib) [5]. We show that SUNNY can be applied also outside the CP domain, reaching promising performance in different fields such as Answer-Set Programming (ASP), Quantified Boolean Formula (QBF), or the Container Pre-marshalling Problem. Conversely, for the Boolean Satisfiability (SAT) problems of ASlib there is still a performance gap with the best AS approaches.

2 SUNNY

The SUNNY [1] algorithm was originally introduced for constraint solving. 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* of x according to the Euclidean distance. Basically, the feature vector of x is a collection $F(x) \in \mathbb{R}^d$ of numerical attributes that characterise x (e.g., statistics over the variables or the constraints of x). Starting from the $N(x, k)$ instances SUNNY relies on three heuristics to compute the schedule σ: H_{sel}, for *selecting* the most promising algorithms $\{A_1, \ldots, A_h\} \subseteq \mathcal{A}$ to run; H_{all}, for *allocating* to each $A_i \in \mathcal{A}$ a certain runtime $t_i \in [0, \tau]$ for $i = 1, \ldots, h$; H_{sch}, for *scheduling* the sequential execution of the algorithms according to their presumed speed. The heuristics H_{sel}, H_{all}, and H_{sch} depends 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. We conclude the section by showing an example of how SUNNY works on a given CSP; for more details about SUNNY we refer the interested reader to [1, 3].

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 1. 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 1. Runtimes (in seconds). τ means the solver timeout.

Scenario	m	n	d	τ	*Domain*
ASP	11	1294	138	600	Answer-Set Programming
CSP	2	2024	86	5000	Constraint Satisfaction Problem
MAXSAT	6	876	37	2100	Maximum Satisfiability Problem
PREMARSH	4	527	16	3600	Container Pre-marshalling Problem
PROTEUS	22	4021	198	3600	CSP, with possible encoding into SAT
QBF	5	1368	46	3600	Quantified Boolean Formula
SAT11-HAND	15	296	115	5000	SAT 2011 Competition – Handcrafted problems
SAT11-INDU	18	300	115	5000	SAT 2011 Competition – Industrial problems
SAT11-RAND	9	600	115	5000	SAT 2011 Competition – Random problems
SAT12-ALL	31	1614	115	1200	SAT Challenge 2012 – All problems
SAT12-HAND	31	767	115	1200	SAT Challenge 2012 – Handcrafted problems
SAT12-INDU	31	1167	115	1200	SAT Challenge 2012 – Industrial problems
SAT12-RAND	31	1362	115	1200	SAT Challenge 2012 – Random problems

Table 2. ASlib Scenarios.

3 Evaluation

To evaluate SUNNY on different scenarios we exploited the Algorithm Selection library (ASlib). ASlib provides standardised format and data for representing AS scenarios allowing the comparison of different AS approaches. Each ASlib scenario contains: 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. ASlib contains 13 heterogeneous scenarios[1] as summarised in Table 2. The scenarios differ in the number of algorithms m, problems n, features d, and in the time limits τ. For every scenario, the runtime is used as performance measure: if algorithm A solves problem x in $t < \tau$ seconds the runtime $\texttt{RunTime}(A, x)$ of A on x is t. Otherwise, $\texttt{RunTime}(A, x) = \tau$. Each scenario of the ASlib is evaluated with a *10-fold cross validation*: $\mathcal{X}$ is partitioned in 10 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 the other folds as the training set.

Adapting SUNNY to ASlib scenarios was rather straightforward. Fixed a training set $\mathcal{X}_{tr} \subseteq \mathcal{X}$ and a corresponding feature space $\mathcal{F}_{tr}$, we normalised the feature vectors by removing all the constant features of $\mathcal{F}_{tr}$ and scaling them in the range $[-1, 1]$. Then, for each unknown problem $x \notin \mathcal{X}_{tr}$, SUNNY computes the neighbourhood $N(x, k) \subseteq \mathcal{X}_{tr}$ and the resulting schedule $\sigma = [(a_1, t_1), \ldots, (a_h, t_h)]$ exactly as explained in Section 2. Following the methodology of [4], we set $k = \sqrt{|\mathcal{X}_{tr}|}$ and the backup solver as the algorithm of $\mathcal{A}$ having the lower average $\texttt{RunTime}$ in $\mathcal{X}_{tr}$.

Table 3 shows for each scenario the *Fraction of Solved Instances* (FSI) of SUNNY. As the name underlines, the FSI of an AS approach is the ratio between the number of instances it solves and all the instances of the scenario. SUNNY is compared against the state-of-the-art AS approaches reported in [5] (viz., ISAC, SNNAP, aspeed, claspfolio, claspfolio-pre, zilla, and LLAMA) and two additional baselines: the *Single Best Solver* (SBS), i.e., the algorithm in $\mathcal{A}$ with highest FSI, and the *Virtual Best Solver* (VBS), i.e., the oracle approach that for every $x \in \mathcal{X}$ always select the algorithm $A \in \mathcal{A}$ for

[1] We considered the 1.0.1 version of ASlib. For more details, we refer the reader to [5].

Scenario	*VBS*	*SBS*	ISAC	SNNAP	aspeed	claspfolio	claspfolio-pre	zilla	LLAMA	SUNNY
ASP	0.937	0.859	0.896	0.910	0.890	**0.923**	0.923	0.915	0.920	0.913
CSP	0.875	0.858	0.859	0.858	0.862	0.872	0.872	0.872	**0.873**	0.870
MAXSAT	0.853	0.769	0.823	0.818	0.845	0.844	0.844	**0.848**	0.841	0.842
PREMARSH	1	0.812	0.843	0.753	**0.956**	0.867	0.945	0.918	0.879	0.949
PROTEUS	0.887	0.628	0.812	0.794	**0.867**	0.832	0.855	0.838	0.835	0.859
QBF	0.77	0.577	0.692	0.615	0.745	0.744	0.753	0.746	0.751	**0.754**
SAT11-HAND	0.74	0.497	0.541	0.611	**0.676**	0.649	0.672	0.655	0.669	0.622
SAT11-INDU	0.843	0.717	0.710	0.740	0.710	**0.763**	0.763	0.717	0.750	0.730
SAT11-RAND	0.82	0.603	0.773	0.743	0.777	0.805	0.807	**0.810**	0.797	0.805
SAT12-ALL	0.988	0.753	0.752	0.880	0.778	0.917	0.916	0.926	**0.929**	0.893
SAT12-HAND	0.701	0.477	0.467	0.580	0.587	0.636	0.638	0.649	**0.653**	0.608
SAT12-INDU	0.821	0.736	0.735	0.777	0.719	**0.788**	0.779	0.775	0.775	0.743
SAT12-RAND	0.764	0.731	0.740	0.730	0.724	**0.744**	0.743	0.737	0.742	0.727

Table 3. Fraction of Solved Instances.

which $\texttt{RunTime}(A, x)$ is minimal. We can see that SUNNY is the best approach for the QBF scenario, and that for all the non-SAT scenarios it is rather close to the best performance. Conversely, for the SAT benchmarks its performance is quite poor.

The FSI 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 *Penalised Average Runtime* (PAR) measure is often used. PAR_k represents the average time taken to solve the problems by giving a penalisation of $k \times \tau$ seconds for the instances not solved within the timeout τ.

Table 4 shows the results considering the average PAR_{10} score. In this case the *SBS* is the single algorithm having the lower PAR_{10} score. Not surprisingly, PAR_{10} is strongly anti-correlated to FSI and the results of 4 somehow reflect what observed in Table 3. However, some differences arise. For instance, in addition to QBF, by considering PAR_{10} SUNNY is the best approach also for the PROTEUS scenarios. This means that in this scenario aspeed solves few instances more than SUNNY, but SUNNY is on-average faster.

Scenario	*VBS*	*SBS*	ISAC	SNNAP	aspeed	claspfolio	claspfolio-pre	zilla	LLAMA	SUNNY
ASP	400.2	880.5	653.6	571.2	711.2	**487.2**	496.0	539.5	509.1	549.0
CSP	6344.3	7201.6	7148.6	7201.6	7163.0	6511.2	6521.4	6491.5	**6466.7**	6617.8
MAXSAT	3127.2	4893.1	3763.1	3855.7	3748.4	3320.4	3629.3	**3234.7**	3367.6	3354.9
PREMARSH	227.6	7002.9	5880.8	9042.1	**1964.1**	5025.0	2395.7	3179.1	4634.2	2221.5
PROTEUS	4105.9	13443.4	6782.5	7430.3	5363.4	6075.1	5525.0	5900.4	6066.6	**5254.4**
QBF	8337.1	15330.2	11201.3	13954.0	9714.3	9333.6	9089.8	9222.5	9075.0	**9064.8**
SAT11-HAND	13360.7	25649.1	23325.1	19820.4	**16688.4**	17975.7	16897.8	17602.4	16906.5	19308.5
SAT11-INDU	8187.5	14605.9	14968.9	13426.8	15008.2	**12322.7**	12383.9	14621.2	12996.6	14014.9
SAT11-RAND	9186.4	19916.4	11575.1	12984.4	11589.0	9982.4	9936.2	**9719.6**	10341.2	9960.262
SAT12-ALL	241.3	3079.9	3101.0	1558.6	2810.2	1113.0	1163.2	1014.7	**980.3**	1429.8
SAT12-HAND	3662.2	6338.9	6466.3	5112.3	5071.9	4450.4	4459.4	4306.3	**4252.9**	4808.3
SAT12-INDU	2221.5	3266.0	3306.2	2796.8	3499.9	**2653.5**	2800.2	2838.4	2837.4	3211.891
SAT12-RAND	2872.8	3271.1	3168.8	3289.9	3382.1	**3119.6**	3161.9	3207.6	3149.6	3327.9

Table 4. Penalised Average Runtime.

4 Conclusions

In this work we presented an evaluation of SUNNY algorithm on different Algorithm Selection (AS) scenarios coming from the Algorithm Selection library (ASlib). Despite SUNNY is tailored for constraint solving, its adaptation to AS appears to be promising also in other fields such as Answer-Set Programming (ASP), Quantified Boolean Formula (QBF), or the Container Pre-marshalling Problem. Conversely, for the Boolean Satisfiability (SAT) problems there is still a performance gap with the best approaches.

We would like to remark that in this evaluation we used the default SUNNY approach without leveraging its settings to fit the different scenarios. As a future work we would like to try to improve the performance of SUNNY by using well-known techniques like pre-solving, parameters tuning, and feature selection. It would be interesting to consider also different scenarios, like optimisation and planning problems. Indeed, the ASlib currently contains a limited number of scenarios for which the only metric is the runtime. It would be nice also to perform a deeper study to better understand the SUNNY performance (and in particular why SUNNY is not so good for the SAT benchmarks).

We strongly encourage the submission of new scenarios and new algorithm selectors to the ASlib in order to foster the study and the comparison of new and better AS approaches. For instance, since SUNNY turns out to be the best approach for QBF, it would be interesting to consider a comparison with the multi-engine solver AQME [8].

We are currently implementing SUNNY as an automated algorithm selector for ASlib scenarios, with the aim of enrolling it to the next ICON Challenge on Algorithm Selection. Moreover, we are also interested in studying how SUNNY can be optimally parallelised to run its algorithms simultaneously on multiple cores. A preliminary investigation on the SUNNY parallelisation for CSPs and COPs is presented in [2].

References

1. R. Amadini, M. Gabbrielli, and J. Mauro. SUNNY: a Lazy Portfolio Approach for Constraint Solving. *TPLP*, 14(4-5):509–524, 2014.
2. R. Amadini, M. Gabbrielli, and J. Mauro. A Multicore Tool for Constraint Solving. In *IJCAI*, 2015. Pre-print available at: `http://arxiv.org/abs/1502.03986`.
3. R. Amadini, M. Gabbrielli, and J. Mauro. Portfolio approaches for constraint optimization problems. *AMAI*, pages 1–18, 2015.
4. R. Amadini, M. Gabbrielli, and J. Mauro. SUNNY-CP: a Sequential CP Portfolio Solver. In *SAC*, 2015. Available at `http://www.cs.unibo.it/~amadini/sac_2015.pdf`.
5. Algorithm Selection Library - coseal. `https://code.google.com/p/coseal/wiki/AlgorithmSelectionLibrary`.
6. C. P. Gomes and B. Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.
7. L. Kotthoff. Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3):48–60, 2014.
8. L. Pulina and A. Tacchella. A self-adaptive multi-engine solver for quantified boolean formulas. *Constraints*, 14(1):80–116, 2009.
9. J. R. Rice. The Algorithm Selection Problem. *Advances in Computers*, 15:65–118, 1976.
10. K. A. Smith-Miles. Towards insightful algorithm selection for optimisation using meta-learning concepts. In *IJCNN*, pages 4118–4124. IEEE, 2008.