An Empirical Study of Per-Instance Algorithm Scheduling

Marius Lindauer, Rolf-David Bergdoll, and Frank Hutter

University of Freiburg

Abstract. Algorithm selection is a prominent approach to improve a system's performance by selecting a well-performing algorithm from a portfolio for an instance at hand. One extension of the traditional algorithm selection problem is to not only select one single algorithm but a schedule of algorithms to increase robustness. Some approaches exist for solving this problem of selecting schedules on a per-instance basis (e.g., the *Sunny* and *3S* systems), but to date, a fair and thorough comparison of these is missing. In this work, we implement *Sunny*'s approach and dynamic schedules inspired by *3S* in the flexible algorithm selection framework *flexfolio* to use the same code base for a fair comparison. Based on the algorithm selection library (*ASlib*), we perform the first thorough empirical study on the strengths and weaknesses of per-instance algorithm schedules. We observe that on some domains it is crucial to use a training phase to limit the maximal size of schedules and to select the optimal neighborhood size of k-nearest-neighbor. By modifying our implemented variants of the *Sunny* and *3S* approaches in this way, we achieve strong performance on many *ASlib* benchmarks and establish new state-of-the-art performance on 3 scenarios.

Keywords: Algorithm Selection, Algorithm Schedules, Constraint Solving

1 Introduction

A common observation in many areas of AI (e.g., SAT or CSP solving) and machine learning is that no single algorithm dominates the performance of all others. To exploit this complementarity of algorithms, algorithm selection systems [11, 6, 8] are used to select a well-performing algorithm for a new given instance. Algorithm selectors, such as *SATzilla* [12] and *3S* [7], demonstrated in several SAT competitions that they can outperform pure SAT solvers by a large margin (see, e.g., the results of the SAT Challenge 2012[1]).

An open problem in algorithm selection is that the machine learning model sometimes fails to select a well-performing algorithm, e.g., because of uninformative instance features. An extension of algorithm selection is to select a schedule of multiple algorithms at least one of which performs well.

[1] `http://baldur.iti.kit.edu/SAT-Challenge-2012/`

To date, a fair comparison of such algorithm schedule selectors is missing, since every publication used another benchmark set and some implementations (e.g., *3S*) are not publicly available (because of license reasons). To study the strengths and weaknesses of such schedulers in a fair manner, we implemented well known algorithm schedule approaches (i.e., *Sunny* [1] and dynamic schedules inspired by *3S* [7]) in the flexible framework of *flexfolio* (the successor of *claspfolio 2* [5]) and studied them on the algorithm selection library (*ASlib* [3]).

2 Per-Instance Algorithm Scheduling

Similar to the per-instance algorithm selection problem [11], the per-instance algorithm scheduling problem is defined as follows:

Definition 1 (Per-Instance Algorithm Scheduling Problem). *Given a set of algorithms $\mathcal{P}$, a set of instances $\mathcal{I}$, a runtime cutoff κ, and a performance metric $m : \Sigma \times \mathcal{I} \to \mathbb{R}$, the* per-instance algorithm scheduling problem *is to find a mapping $s : \mathcal{I} \to \Sigma$ from an instance $\pi \in \mathcal{I}$ to a (potentially unordered) algorithm schedule $\sigma_\pi \in \Sigma$ where each algorithm gets a runtime budget $\sigma_\pi(\mathcal{A})$ between 0 and κ such that $\sum_{\mathcal{A} \in \mathcal{P}} \sigma_\pi(\mathcal{A}) \leq \kappa$ and $\sum_{\pi \in \mathcal{I}} m(s(\pi), \pi)$ will be minimized.*

The algorithm scheduler *aspeed* [4] addresses this problem by using a static algorithm schedule; i.e., *aspeed* applies the same schedule to all instances. The schedule is optimized with an answer set programming [2] solver to obtain a timeout-minimal schedule on the training instances. The scheduler *aspeed* either uses a second optimization step to determine a well-performing ordering of the algorithms or sorts the algorithms by their assigned times, in ascending order (such that a wrongly selected solver does not waste too much time).

Systems such as *3S* [7], *SATzilla* [12] and *claspfolio 2* [5] combine static algorithm schedules (also called pre-solving schedules) and classical algorithm selection. All these systems run the schedule for a small fraction of the runtime budget κ (e.g., *3S* uses 10% of κ), and if this pre-solving schedule fails to solve the given instance, they apply per-instance algorithm selection to run an algorithm predicted to perform well. *3S* and *claspfolio 2* use mixed integer programming and answer set programming solvers, respectively, to obtain a timeout-minimal pre-solving schedule. *SATzilla* uses a grid search to obtain a pre-solving schedule that optimizes the performance of the entire system.

The algorithm scheduler *Sunny* [1] determines the schedule for a new instance π by first determining the set of k training instances $\mathcal{I}_k$ closest to π in instance feature space, and then assigns each algorithm a runtime proportional to the number of instances in $\mathcal{I}_k$ it solved. The algorithms are sorted by their average PAR10 scores on $\mathcal{I}_k$, in ascending order (which corresponds to running the algorithm with the best expected performance first).

3 Instance-specific Aspeed (*ISA*)

Kadioglu et al. [7] proposed a variant of *3S* that uses per-instance algorithm schedules instead of a fixed split between static pre-solving schedule and algorithm

selection. In order to evaluate the potential of per-instance timeout-optimized scheduling, we developed the scheduler *ISA*, short for *instance-specific aspeed*. Inspired by Kadioglu et al. [7], our implementation uses k-nearest neighbor (k-NN) to identify the set $\mathcal{I}_k$ of training instances closest to a given instance π and then applies *aspeed* to obtain a timeout-minimal schedule for them.

During offline training, we have to determine a promising value for the neighborhood size k. In our experiments, we evaluated different k values between 1 and 40 by running cross-validation on the training data and stored the best performing value to use online. We chose this small upper bound for k to ensure a feasible runtime of the scheduler[2] (in our experiments less than 1 second). Furthermore, to optimize the runtime of the scheduler, we reduced the set of training instances, omitting all instances that were either solved by every algorithm or solved by none within the cutoff time.

For each new instance, *ISA* first computes the k nearest neighbor instances from the reduced training set. This instance set is passed to *aspeed* [4], which returns a timeout-minimal unordered schedule for the neighbor set. The schedule is finally aligned by sorting the time slots in ascending order.

4 Trained Sunny (*TSunny*)

To offer a form of scheduling with less overhead in the online stage than *ISA*, we implemented a modified version of *Sunny* [1] by adding a training phase. For a new problem instance *Sunny* first selects a subset of k training instances $\mathcal{I}_k$ using k-NN. Then time slots are assigned to each candidate algorithm: Each solver gets one slot for each instance of $\mathcal{I}_k$ it can solve within the given time. Additionally, a designated backup solver gets one slot for each instance of $\mathcal{I}_k$ that cannot be solved by any of the algorithms. Having this slot assignment, the actual size of a single time slot is computed by dividing the available time by the total number of slots. Finally, the schedule is aligned by sorting the algorithms by their average PAR10 score on $\mathcal{I}_k$, thereby running the most promising solver first.

Preliminary experiments for our implementation of this algorithm produced relatively poor results. Examining the schedules, we found that *Sunny* tends to employ many algorithms per schedule, which we suspected to be a weakness. Thus, we enhanced the algorithm by limiting the number of algorithms used in a single schedule to a specified number λ.

Originally, *Sunny* is defined as lazy, i.e. not applying any training procedures after the benchmark data is gathered. However, to obtain better values for our new parameter λ, and also to improve the choice of the neighborhood size k, we implemented a training process for *Sunny*. Similar to *ISA*, different configurations for λ (range 1 to the total number of solvers) and k (range 1 to 100) are evaluated by cross-validation on the training data. To distinguish this enhanced algorithm from the original *Sunny*, we dubbed this trained version *TSunny*.

[2] Optimizing a schedule is NP-hard; thus the size of the input set, defined by k, must be kept small to make the process applicable during runtime.

5 Empirical Study

To compare the different algorithm scheduling approaches of *ISA* and *Sunny*, we implemented them in the flexible algorithm selection framework *flexfolio*[3] and compared them to various other systems: The static algorithm scheduling system *aspeed* [4], the default configuration of *flexfolio* (which is similar to *SATzilla* [12] and *claspfolio 2* [5] and includes a static-presolving schedule), as well as the per-instance algorithm selector *AutoFolio* [9] (an automatically-configured version of *flexfolio* without consideration of per-instance algorithm schedules). If not mentioned otherwise, we used the default parameter values of *flexfolio*. The comparison is based on the algorithm selection library (*ASlib* [3]), which is specifically designed to fairly measure the performance of algorithm selection systems. Version 1.0 of *ASlib* consists of 13 scenarios from a wide range of different domains (SAT, MAXSAT, CSP, QBF, ASP and operations research).

	flexfolio	*AutoFolio*	*aspeed*	*Sunny*	*TSunny*	*ISA*
ASP-POTASSCO	0.78*	0.80*	0.34	0.69	**0.81***	0.72
CSP-2010	**0.80***	0.75*	0.05	0.68	0.77*	0.74*
MAXSAT12-PMS	0.67	0.90*	0.65	0.87	0.93*	**0.94***
PREMAR-2013	0.70	0.74*	0.74*	0.71	0.62	**0.78***
PROTEUS-2014	0.82	0.87	0.87	0.88	**0.94***	0.91
QBF-2011	0.90	0.91	0.80	0.90	**0.94***	0.92
SAT11-HAND	0.73*	0.71*	**0.74***	0.54	0.52	0.69*
SAT11-INDU	0.29*	0.36*	0.06	0.19	0.37*	**0.43***
SAT11-RAND	0.93*	**0.95***	0.80	0.59	0.87	0.95*
SAT12-ALL	0.69*	0.69*	0.10	0.58	0.69*	**0.71***
SAT12-HAND	0.68	0.71	0.46	0.57	0.72	**0.78***
SAT12-INDU	0.39	0.46*	−0.22	0.01	0.53*	**0.54***
SAT12-RAND	0.17	0.24*	−0.28	−0.14	**0.32***	0.12
Average	0.66	0.7	0.39	0.54	0.69	0.71
Equal to Best	6	10	2	0	9	9

Table 1: Gap metric on PAR10: 1.0 corresponds to a perfect oracle score and 0.0 corresponds to the single best score. The best score for each scenario is highlighted with bold face and all system performances have a star that are not significantly worse than the best system (permutation test with 100 000 random permutations and $\alpha = 0.05$; "Equal to Best"). All systems are implemented in *flexfolio*, except *Sunny* which is the original version.

Table 1 shows the performance of the systems as the fraction of the gap closed between the static single best algorithm and the oracle (i.e., the performance of

[3] The source code and all benchmark data are available at `http://www.ml4aad.org/algorithm-selection/flexfolio/`.

an optimal algorithm selector), using performance metric PAR10[4]. As expected, the static schedules of *aspeed* performed worse on average (except on SAT11-HAND, where – as observed previously [4] – *aspeed* performed best). *Sunny* performed better than *aspeed* but still not as well as *flexfolio*.[5] The trained *Sunny* version, *TSunny*, performed better than *Sunny* and even better than *flexfolio*, but slightly worse than *AutoFolio*. On average, *ISA* was the best system with a score of 0.71, but it was only amongst the best systems on 9/13 scenarios. *AutoFolio* was more robust, being amongst the best systems on 10/13 scenarios, but it had a slightly worse average (with a score of 0.7). We also note that training *ISA* was much cheaper (only some CPU hours for each scenario) than *AutoFolio*, for which we spent several CPU weeks per scenario to obtain a similar performance. According to the on-going evaluation on *ASlib*, *ISA* establishes new state-of-the-art performance on PREMAR-2013 (short for PREMARSHALLING-ASTAR-2013) and *TSunny* on PROTEUS-2014 and QBF-2011.

Table 2 gives more insights into our systems' performance. It also includes our implemented version of *Sunny* without training, dubbed *Sunny*'. *Sunny* (and also *Sunny*') sets the neighborhood size k as the square root of the number of instances, whereas *TSunny* optimizes k on the training instances. The reason for *TSunny*'s better performance in comparison to *Sunny* is probably its much smaller values for k on all scenarios except on SAT12-RAND. Also *TSunny*'s average schedule size was smaller on nearly all scenarios (except CSP-2010).

Comparing the static *aspeed* and the instance-specific *aspeed* (*ISA*), the average schedule size of *aspeed* is rather large since *aspeed* has to compute a single static schedule that is robust across all training instances and not only on a small subset. Surprisingly, the values of k for *ISA* and *TSunny* differ a lot, indicating that the best value of k depends on the scheduling strategy.

6 Conclusion and Discussion

We showed that per-instance algorithm scheduling systems can perform as well as algorithm selectors and even establish new state-of-the-art performance on 3 scenarios of the algorithm selection library [3]. Additionally, we found that the performance of the algorithm schedules strongly depends on the adjustment of their parameters for each scenario, here the neighborhood size of the k-nearest neighbor and the maximal size of the schedules.

In our experiments we did not tune all possible parameters of *Sunny* and *ISA* in the flexible *flexfolio* framework; e.g., we fixed the pre-processing strategy of the instance features. Therefore, a future extension of this line of work would be to extend the search space of the automatically-configured algorithm selector *AutoFolio* [9] to also cover per-instance algorithm schedules. Another extension could be to allow communication between the algorithms in the schedule [10].

[4] PAR10 is the penalized average runtime where timeouts are counted as 10 times the runtime cutoff.

[5] We note that *SATzilla* also has an average gap score of 0.66 according to the on-going evaluation on *ASlib* (see `www.aslib.net`).

	aspeed		*ISA*			*Sunny'*			*TSunny*		
	$\varnothing\lvert\sigma\rvert$	$\varnothing suc$	k	$\varnothing\lvert\sigma\rvert$	$\varnothing suc$	k	$\varnothing\lvert\sigma\rvert$	$\varnothing suc$	k	$\varnothing\lvert\sigma\rvert$	$\varnothing suc$
ASP-POTASSCO	5.9	1.96	14.4	1.6	1.07	34.0	10.7	1.15	19.6	1.0	1.01
CSP-2010	2.0	1.2	5.8	1.1	1.0	43.0	1.9	1.01	12.8	1.9	1.0
MAXSAT12-PMS	3.0	1.98	7.3	1.2	1.02	28.0	5.4	1.04	6.4	3.0	1.01
PREMAR-2013	4.0	1.75	32.6	2.3	1.3	22.0	4.0	1.22	9.0	3.6	1.21
PROTEUS-2014	18.3	7.27	30.6	3.2	1.41	60.0	13.9	1.77	26.6	12.5	1.5
QBF-2011	4.9	2.2	27.8	1.9	1.26	35.0	4.5	1.1	14.1	3.3	1.06
SAT11-HAND	5.9	2.96	27.8	3.1	1.92	16.0	13.5	1.6	10.2	1.7	1.02
SAT11-INDU	4.6	2.82	3.8	1.3	1.03	16.0	16.5	1.55	4.2	1.4	1.02
SAT11-RAND	3.8	1.94	14.4	1.8	1.13	23.0	7.8	1.04	18.3	1.5	1.02
SAT12-ALL	12.6	5.24	8.8	1.6	1.12	38.0	24.4	1.72	4.2	1.0	1.0
SAT12-HAND	10.9	5.45	4.8	1.5	1.09	26.0	26.2	1.68	4.6	1.0	1.01
SAT12-INDU	6.2	3.64	6.1	1.2	1.04	32.0	22.5	1.75	4.3	1.0	1.0
SAT12-RAND	5.2	2.27	18.3	1.8	1.07	35.0	15.7	1.12	67.2	1.0	1.0

Table 2: Statistics of schedules: neighborhood size k, average size $\varnothing\lvert\sigma\rvert$ of schedules, average position $\varnothing suc$ of successful solver in schedule for our systems *aspeed*, *ISA*, *Sunny*' (a reimplementation of the lazy version of *Sunny*), and *TSunny* (the non-lazy trained version of *Sunny*')

References

1. Amadini, R., Gabbrielli, M., Mauro, J.: SUNNY: a lazy portfolio approach for constraint solving. TPLP 14(4-5), 509–524 (2014)
2. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
3. Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Frechétte, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K., Vanschoren, J.: ASlib: A benchmark library for algorithm selection. AIJ (2016), to appear
4. Hoos, H., Kaminski, R., Lindauer, M., Schaub, T.: aspeed: Solver scheduling via answer set programming. TPLP 15, 117–142 (2015)
5. Hoos, H., Lindauer, M., Schaub, T.: claspfolio 2: Advances in algorithm selection for answer set programming. TPLP 14, 569–585 (2014)
6. Huberman, B., Lukose, R., Hogg, T.: An economic approach to hard computational problems. Science 275, 51–54 (1997)
7. Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm selection and scheduling. In: Proc. of CP'11. pp. 454–469 (2011)
8. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. AI Magazine pp. 48–60 (2014)
9. Lindauer, M., Hoos, H., Hutter, F., Schaub, T.: Autofolio: An automatically configured algorithm selector. Journal of Artificial Intelligence 53, 745–778 (2015)
10. Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Boosting sequential solver portfolios: Knowledge sharing and accuracy prediction. In: Proc. of LION'13. pp. 153–167 (2013)
11. Rice, J.: The algorithm selection problem. Advances in Computers 15, 65–118 (1976)
12. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. JAIR 32, 565–606 (2008)