Learning Portfolios of Automatically Tuned Planners

Jendrik Seipp
Manuel Braun
Johannes Garimort
Albert-Ludwigs-Universität Freiburg
Freiburg, Germany
{seipp,braun,garimort}@informatik.uni-freiburg.de

Malte Helmert
Universität Basel
Fachbereich Informatik
Bernoullistrasse 16
4056 Basel, Switzerland
malte.helmert@unibas.ch

Abstract

Portfolio planners and *parameter tuning* are two ideas that have recently attracted significant attention in the domain-independent planning community. We combine these two ideas and present a portfolio planner that runs automatically configured planners. We let the automatic parameter tuning framework ParamILS find fast configurations of the Fast Downward planning system for a number of planning domains. Afterwards we learn a portfolio of those planner configurations. Evaluation of our portfolio planner on the IPC 2011 domains shows that it has a significantly higher IPC score than the winner of the sequential satisficing track.

Introduction

Planning domains are inherently different from one another, and no planner outperforms all others on all domains (Roberts and Howe 2009). Moreover, typical planners either solve a problem quickly or not at all (Howe and Dahlman 2002; Helmert, Röger, and Karpas 2011). These observations motivate the idea of running multiple planners sequentially – a *sequential portfolio* of planners – to achieve better overall performance than any individual planner.

Portfolios have been successfully applied to a number of combinatorial search domains, most notably the satisfiability problem for propositional logic. Petrik and Zilberstein (2006) provide a good general discussion of the portfolio learning problem, including sample complexity results.

One of the central questions in building portfolio planners is which planning systems to include in the portfolio. Our hypothesis is that if we want a portfolio to generalize to unknown domains (i.e., domains not used for training the portfolio), *diversity* of components is critical. In this work, we achieve this diversity by taking a number of domains from previous planning competitions and finding a good planner for each of them. Although in general we could include completely different planners in the search, we limit ourselves to the configuration space of the Fast Downward planning system (Helmert 2006). This system provides abundant search types and heuristics by itself, many of which are state-of-the-art mechanisms for automatic planning. For searching well-performing configurations we use the off-the-shelf parameter configuration framework ParamILS (Hutter et al. 2009).

Our ideas are similar to the methods used in Helmert, Röger, and Karpas (2011). There the authors handpick a set of configurations for the Fast Downward planning system and evaluate their performance on previous International Planning Competition (IPC) domains. Afterwards they use hill-climbing search to form a planner portfolio by assigning each configuration a relative amount of time that it is allowed to try solving the problem. A major difference to our work is that we do not rely on the researcher to decide which configurations to examine, but find them automatically by tuning the planner's parameters in different domains.

One of the first portfolio systems for planning is described by Howe et al. (1999). The authors implemented a system called BUS that runs six planners in a round-robin scheme until one of them finds a solution. In the portfolio the planners are ordered based on a linear regression model of their success and runtime. Contrary to our work the assigned time slices may change during portfolio execution and a planner may be restarted multiple times with different time slices.

Similarly Roberts and Howe (2006) let a decision tree learn the success rates of 23 planners on 1839 problem instances based on 57 domain and instance features. For each new problem they form a portfolio of planners by ordering them by decreasing probability of success and run them sequentially with increasing timeouts in each round until a solution is found. The major difference between our work and the two approaches above is that we learn our portfolio offline, i.e. the ordering and the time limits of the contained planners are fixed before we encounter a new problem.

While the portfolio planners above are domain-independent, Gerevini, Saetti, and Vallati (2009) introduced PbP, a planner that learns a portfolio for a specific domain. PbP incorporates seven planners it can choose from. It lets them learn macro-actions for the domain and runs up to three best-performing ones in round-robin fashion with learned time slots. What differentiates our approach from PbP, in addition to PbP being domain-specific, is the fact that we allow using more planners in the portfolio (up to 21) and never start the same planner twice. Obviously our domain-independent portfolio cannot use macro-actions.

Gerevini, Saetti, and Vallati (2011) recently implemented PbP2 which extends PbP by adding the planners LAMA

(Richter and Westphal 2010) and ParLPG (Vallati et al. 2011) to the set of available planners. This addition mixes portfolios with parameter tuning, ParLPG being a framework for automatically finding good parameters for LPG (Gerevini and Serina 2002) given a planning domain.

ParLPG finds the LPG parameters using the ParamILS framework (Hutter et al. 2009). The same technique has also been applied to the Fast Downward planning system (Fawcett et al. 2011). Both systems have been tuned once for speed and once for plan quality, while tuning for the latter is apparently much harder, because the runtimes cannot be capped as efficiently. We avoid this by tuning for speed exclusively and improve on plan quality by running many obtained fast planners sequentially.

A different approach to parameter configuration can be found in the work of Vrakas et al. (2003). The authors extract a number of domain- and problem-dependent features from a set of benchmark domains and learn a rule-based classification model that decides about the parameter settings for each problem based on the interaction of the features and parameter values for some training problems.

The remainder of the paper is organized as follows. In the next section we present the parameter tuning method and the results on the training domains. Afterwards we explain our portfolio generation methods and evaluate their performance on the domains of the IPC 2011 sequential satisficing track.

Tuning Planners

We used the FocusedILS variant of ParamILS (Hutter et al. 2009) to tune the parameters of the Fast Downward planning system separately for 21 domains from the IPC 1998–2006 challenges. If multiple equivalent formulations of the same domain were available, we only picked one of them. IPC 2008 and 2011 domains were excluded, because these form our distinct test set, on which we evaluate the portfolios introduced in the next section.

Fast Downward consists of many highly parametrized search algorithms and heuristics that can be combined in potentially infinite ways. We therefore restricted the configuration space to a smaller subset as described for FD-Autotune.s in Fawcett et al. (2011). Even with this restriction the parameter space includes 2.99×10^{13} configurations.

We tuned for speed rather than quality since we want to run multiple fast planner configurations sequentially in the same amount of time a single planner is given in the IPC. Moreover, it is easier to tune for speed than quality as this allows to cap inefficient configurations early during the parameter tuning process, which is not possible when tuning for quality, because one always has to wait for the planner to terminate before finding out the quality it will achieve. As the objective measure for tuning we used mean runtime.

Although Fast Downward does not have a default configuration, ParamILS has to start its search in the configuration space somewhere. We use the same initial configuration as Fawcett et al. (2011).

For each domain we let the set of training instances be the problems that the initial configuration solves in under 30 minutes and over 0.1 seconds. In order to speed up the tuning process, we do not use harder problems and set a timeout of 180 seconds for individual planner runs. Easier problems are also excluded, because they can be solved by all configurations without any measurable difference in speed and are therefore not useful for tuning.

The FocusedILS algorithm contains some randomization allowing us to start multiple tunings (in our experiments five) and compare the discovered configurations. We say that a tuning *converged* when all tuning runs yield the same best configuration, i.e. use the same heuristics and search options and differ only in minor parameters.

The randomized FocusedILS tuning algorithm converged after at most 40 CPU hours in all 21 domains. Details on the configurations found in each domain are provided in a technical report (Seipp et al. 2012); noteworthy trends include:

- preferred operators are almost always used (19x)
- lazy search is almost always used (20x)
- most configurations use one (10x) or two (9x) heuristics
- five different heuristics are used: h^{FF} (12x), landmarks (11x), h^{cg} (6x), h^{cea} (4x), h^{add} (1x)

In order to validate the efficiency of our tuning, we ran all new planner configurations on all 21 domains. Table 1 shows the number of solved problems in each domain for each planner. In keeping with our assumptions, in almost all domains the coverage is highest for the configuration that was trained on the respective domain. For some domains the performance of other configurations is similar or even better. This indicates that planners tuned for one domain can also perform well on others and that some domains share inherent problem structure. If all configurations failed on all domains except the one they where tuned on, we probably would not be able to build a portfolio that generalizes to unseen problems.

Portfolios

In this section we present seven different portfolio generation methods. We use them to build sequential portfolios containing the domain-wise trained configurations from the previous section. However, these methods could in general be used for all kinds of planning algorithms or planner configurations.

A portfolio generator requires a set of planning algorithms $\mathcal{A}$, a set of training instances $\mathcal{I}$ and the corresponding performance measures. For each $a \in \mathcal{A}$ and $i \in \mathcal{I}$ we measure the *runtime* $t(a, i)$, the *cost* $c(a, i)$ needed to solve i with a and the *quality* $q(a, i)$ of the calculated plan. We define the quality analogously to the IPC 2011 settings:

$$q(a, i) = \frac{\min_{a' \in \mathcal{A}} c(a', i)}{c(a, i)}$$

If algorithm $a \in \mathcal{A}$ cannot solve problem $i \in \mathcal{I}$, we set $t(a, i) = \infty$ and $q(a, i) = 0$. Moreover we need to specify a *score* s according to which the generator optimizes the portfolio and a *total time limit* the portfolio is allowed to run.

Then a (sequential) portfolio is a mapping $P : \mathcal{A} \to \mathbb{R}_0^+$ which assigns a *time limit* to each algorithm. Time limits of 0 imply that the portfolio does not contain the corresponding algorithms. The sum of all component time

	airport	depot	driverlog	freecell	grid	logistics00	miconic-full	mprime	optical-t	pathways	philosophers	pipes-nt	pipes-t	psr-large	rovers	satellite	schedule	storage	tpp	trucks	zenotravel
airport (50)	47	42	**48**	25	42	40	35	42	37	44	26	36	35	37	46	26	43	27	47	34	28
depot (22)	18	**19**	18	12	18	17	**19**	**19**	16	16	13	**19**	**19**	14	17	12	18	17	16	15	12
driverlog (20)	**20**	**20**	**20**	16	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	18	18
freecell (80)	78	78	**80**	**80**	**80**	77	79	76	77	**80**	66	**80**	**80**	76	**80**	70	77	**80**	**80**	77	43
grid (5)	**5**	**5**	**5**	3	**5**	**5**	**5**	**5**	**5**	**5**	4	**5**	**5**	3	**5**	4	**5**	4	**5**	4	**5**
logistics00 (28)	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**
miconic-full (150)	0	0	0	0	0	**139**	138	**139**	138	0	137	0	138	138	0	134	0	0	0	137	134
mprime (35)	**35**	**35**	**35**	21	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**	**35**	34	34	**35**	**35**	**35**	34	**35**	**35**
optical-t (48)	0	0	0	0	0	4	2	4	**21**	0	2	0	3	1	0	1	0	0	0	2	2
pathways (30)	23	29	29	5	29	28	29	28	22	**30**	7	**30**	29	14	**30**	7	**30**	11	**30**	6	7
philosophers (48)	0	0	0	0	0	**48**	45	**48**	**48**	0	**48**	0	45	**48**	0	**48**	0	0	0	5	5
pipes-nt (50)	43	43	43	29	43	42	44	42	41	44	30	44	44	32	43	23	**45**	35	43	43	28
pipes-t (50)	25	37	30	17	40	34	41	33	26	39	21	40	**42**	20	38	17	38	17	38	38	20
psr-large (50)	0	0	0	0	0	31	15	31	26	0	33	0	20	**36**	0	33	0	0	0	19	31
rovers (40)	39	**40**	**40**	18	**40**	**40**	**40**	**40**	34	**40**	32	**40**	**40**	32	**40**	32	**40**	32	**40**	23	28
satellite (36)	**36**	**36**	**36**	11	**36**	**36**	**36**	**36**	**36**	**36**	**36**	34	**36**	**36**	**36**	**36**	**36**	**36**	**36**	35	27
schedule (150)	143	**150**	144	61	**150**	144	148	127	145	**150**	61	**150**	149	65	146	62	**150**	**150**	149	99	26
storage (30)	18	18	19	19	19	19	20	20	20	**21**	17	18	20	17	**21**	17	20	**21**	**21**	17	19
tpp (30)	25	28	**30**	10	**30**	29	**30**	26	24	**30**	27	**30**	**30**	26	**30**	26	**30**	26	**30**	8	10
trucks (30)	15	16	13	7	16	16	20	16	16	13	12	18	20	18	13	11	15	9	13	**23**	10
zenotravel (20)	**20**	**20**	**20**	15	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**

Table 1: Coverage (number of solved problems) of all tuned planner configurations (columns) on all domains (rows). Best values are highlighted in bold, diagonal entries are underlined.

limits, $\sum_{a \in \mathcal{A}} P(a)$ must be smaller or equal to the total time limit of a portfolio P. P *solves* a problem i if i can be solved by any algorithm within its time limit, i.e. $t(a, i) \leq P(a)$ for any a. The according *quality* is the maximum quality achieved by all algorithms solving i, i.e. $q(P, i) = \max\{q(a, i) \mid a \in \mathcal{A}, t(a, i) \leq P(a)\}$. These definitions follow Helmert, Röger, and Karpas (2011).

Portfolio Generators

The generators get the following information:

- $\mathcal{A}$ consists of the 21 Fast Downward configurations we found with the methods from the previous section.
- $\mathcal{I} = D_1 \cup \ldots \cup D_{21}$ is the union of all 21 domains from the last section (with each domain viewed as a set of problems) and consists of 1002 planning problems.
- Naturally, the score s should be based on solution quality. However, due to the nature of our approach we have to make two adjustments. To avoid favoring configurations that excel in domains with many problems, we normalize the scores by the number of tasks in a domain:

$$s(P) = \sum_{j=1}^{21} \frac{1}{|D_j|} \sum_{i \in D_j} q(P, i)$$

  Moreover, to ensure that the resulting portfolio generalizes well to new domains, we only count the performance of configurations on domains they have not been trained on. This is achieved by setting $q(a, i) = 0$ if algorithm a was trained on the domain to which problem i belongs.
- Following the rules of the IPC satisficing track the generated portfolios have a total time limit of 30 minutes.

Stone Soup. The Stone Soup portfolio generator (Helmert, Röger, and Karpas 2011) tries to find portfolios maximizing the score by hill-climbing. We include the algorithm here to evaluate the impact of the domain tuning on the portfolio performance.

The algorithm starts with a portfolio of total time 0 for each configuration. Then in each iteration it generates a set of possible successor portfolios: separately, the total time of each configuration is increased by a small time portion g resulting in one new portfolio per configuration. The successor with the maximum score is selected as the new portfolio. The algorithm proceeds until the total time limit is reached.

Uniform. The uniform portfolio runs all 21 configurations for $\left\lfloor \frac{\textit{total time limit}}{21} \right\rfloor = 85$ seconds.

Selector. For each subset size $\{1, \ldots, 21\}$ this brute force approach computes the best subset of configurations with equal time limits $\left\lfloor \frac{\textit{total time limit}}{\textit{subset size}} \right\rfloor$ and returns the one with the highest score.

Cluster. We use the k-means clustering algorithm (MacKay 2003, p. 284–289) to form k clusters of configurations with similar scores for the individual problems. Afterwards, from each cluster we select the configuration with the best score and give it the runtime $\left\lfloor \frac{\textit{total time limit}}{k} \right\rfloor$.

Increasing Time Limit. The generator iteratively increases the portfolio time limit by t seconds. In each iteration it chooses the problems that can be solved within the time limit ignoring the problems the portfolio already solves. If no such problems are found, it increases the time limit and continues. Otherwise, the time limit of the configuration maximizing the score for the selected problems is set to the maximum runtime the configuration needs to solve the

Quality	LAMA	FD-Autotune		FD Stone Soup		Stone Soup		Uniform	Selector	Cluster		Inc. Time Limit		DW	RIS
	2011	1	2	1	2	g=100	g=110			k=16	k=12	t=10s	t=5s		
Training Score	–	–	–	–	–	19.39	19.39	19.22	19.28	19.18	19.17	18.73	18.74	19.07	19.49
barman (20)	17.62	13.23	5.40	6.31	5.60	17.06	18.27	19.02	18.96	17.05	**19.14**	19.02	19.01	18.88	19.08
elevators (20)	11.11	10.63	14.37	11.82	13.48	15.18	15.71	15.27	14.96	15.70	15.80	15.25	14.90	15.30	**16.38**
floortile (20)	4.81	6.56	**8.87**	5.87	6.37	5.93	5.80	5.93	6.01	5.93	5.63	5.18	5.22	5.20	5.95
nomystery (20)	10.81	10.38	16.52	12.33	12.51	18.77	18.75	17.73	17.76	18.75	17.76	16.65	16.65	16.67	**18.82**
openstacks (20)	**18.71**	12.99	17.06	12.48	12.35	11.11	11.26	16.33	10.72	16.76	15.36	15.29	14.49	14.14	11.41
parcprinter (20)	19.51	19.48	13.61	19.12	18.17	19.56	19.54	19.88	18.82	**19.89**	19.74	19.87	19.86	19.88	**19.89**
parking (20)	11.48	3.74	3.49	15.23	14.85	17.16	**17.40**	16.89	16.98	17.05	16.20	15.57	15.57	13.32	14.79
pegsol (20)	**20.00**	19.28	19.72	17.04	14.61	19.50	19.38	19.54	19.38	19.64	19.29	19.19	19.19	19.53	19.43
scanalyzer (20)	17.89	16.71	15.87	18.65	17.38	19.12	18.37	19.19	18.92	19.00	18.75	18.77	18.83	19.25	**19.69**
sokoban (20)	16.68	17.16	10.51	17.31	15.55	18.59	18.55	17.26	17.37	17.35	17.25	18.26	18.22	**18.64**	17.55
tidybot (20)	14.13	13.86	12.51	14.69	14.63	15.08	15.03	**16.40**	15.87	16.21	14.49	15.60	15.58	15.29	15.88
transport (20)	12.64	9.48	8.49	9.29	9.59	15.94	15.88	17.55	**17.60**	16.00	17.11	16.74	16.95	17.12	15.58
visitall (20)	15.55	1.71	3.29	3.98	0.92	**19.90**	**19.90**	**19.90**	**19.90**	**19.90**	**19.90**	**19.90**	**19.90**	**19.90**	**19.90**
woodworking (20)	14.65	14.72	10.25	**19.99**	18.43	15.92	15.82	15.94	15.88	15.92	15.71	15.80	15.79	15.77	15.91
Sum (280)	205.59	169.94	159.94	184.12	174.45	228.82	229.67	**236.84**	229.14	235.17	232.15	231.10	230.17	228.89	230.28

Table 2: The qualities of our portfolios (right) compared to IPC 2011 competitors (left) on the IPC 2011 sequential satisficing track domains. Best planners in each domain are highlighted in bold. The first row shows the training scores of our portfolios.

problems. The algorithm continues in this fashion until the total time limit is reached or all problems are solved.

Domain-wise. Iteratively we retrieve the domain with the highest improvement potential, i.e. the biggest difference between the currently achieved score and the score it could reach if all configurations had the maximum time limit of 30 minutes. We add the configuration that improves the domain's score the fastest to the portfolio and set its time limit to the amount of time needed for the improvement. Problems solved by the current portfolio are ignored in later score calculations. The iteration continues until the total time limit is reached or no more domains can be improved.

Randomized Iterative Search. This generator can use any existing portfolio as initialization and iteratively improves it using a randomized local search method. At each step it evaluates successors of the current portfolio in a random order. If it finds a better successor, it instantly commits to it and continues the iteration. As a successor it considers any portfolio that can be obtained from the current one by swapping a fixed runtime amount between two algorithms or by taking an equal amount of runtime from all algorithms and assigning it to a single one. We end the iteration when the score does not increase for 20000 successor evaluations.

In our experiments we used the uniform portfolio as initialization. Starting with other portfolios did not improve the training score and the variance of the achieved score was very small across multiple runs.

Experiment

We evaluated all generated portfolios on the domains of the IPC 2011 sequential satisficing track to investigate transferability to unseen domains. For comparison we also included some of the best performers from the last IPC competition in the experiment. We used a time limit of 30 minutes and a memory limit of 2 GB. The results in Table 2 clearly show that not only do we beat all versions of the other portfolio planner Stone Soup (Helmert, Röger, and Karpas 2011) and the tuned version of Fast Downward FD-Autotune (Fawcett et al. 2011), but we even clearly outperform the competition winner LAMA 2011 (Richter, Westphal, and Helmert 2011).

The most striking result however, is that the trivial uniform portfolio method performs best.[1] This reinforces our initial assumption that problems tend to be solved fast or not at all, and diversity is key to building strong portfolios. The technical report (Seipp et al. 2012) discusses additional experiments with shorter timeouts, which show that the best portfolios already outperform LAMA 2011 when only given half as much time, and that with significantly shorter timeouts the uniform portfolio is no longer the best choice.

Conclusions and Future Work

We have shown that it is very helpful to automatically find diverse planners for a range of known domains to be able to build portfolios that perform well on unknown domains. Of course our idea is not limited to automated planning. We think that portfolio approaches from all kinds of research areas might benefit from automatically widening the scope of algorithms the portfolios can choose from.

In the future we will try to use our approach for optimal planning. Recent improvements in Fast Downward have introduced many parameters for the optimizing configurations, making our methods a natural fit in this area.

Additionally, we will experiment with adaptively selecting the next planner during portfolio execution.

We note that our work naturally benefits from improvements in the underlying configurations. Furthermore, expanding our set of available planners with planners from other research teams would probably increase the number of domains we can perform well on.

[1] The difference between the uniform portfolio and LAMA 2011 is highly statistically significant ($p = 0.0002$ with $\chi^2 = 13.76$ using McNemar's test when comparing the set of tasks solved).

References

Fawcett, C.; Helmert, M.; Hoos, H.; Karpas, E.; Röger, G.; and Seipp, J. 2011. FD-Autotune: Domain-specific configuration using Fast Downward. In *ICAPS 2011 Workshop on Planning and Learning*, 13–17.

Gerevini, A., and Serina, I. 2002. LPG: A planner based on local search for planning graphs with action costs. In Ghallab, M.; Hertzberg, J.; and Traverso, P., eds., *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002)*, 13–22. AAAI Press.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: PbP. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 350–353. AAAI Press.

Gerevini, A.; Saetti, A.; and Vallati, M. 2011. PbP2: Automatic configuration of a portfolio-based multi-planner. In *IPC 2011 planner abstracts*.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, 28–35.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Howe, A. E., and Dahlman, E. 2002. A critical assessment of benchmark comparison in planning. *Journal of Artificial Intelligence Research* 17:1–33.

Howe, A. E.; Dahlman, E.; Hansen, C.; Scheetz, M.; and von Mayrhauser, A. 1999. Exploiting competitive planner performance. In Biundo, S., and Fox, M., eds., *Recent Advances in AI Planning. 5th European Conference on Planning (ECP 1999)*, volume 1809 of *Lecture Notes in Artificial Intelligence*, 62–72. Heidelberg: Springer-Verlag.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36:267–306.

MacKay, D. J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

Petrik, M., and Zilberstein, S. 2006. Learning parallel portfolios of algorithms. *Annals of Mathematics and Artificial Intelligence* 48:85–106.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39:127–177.

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, 50–54.

Roberts, M., and Howe, A. E. 2006. Directing a portfolio with learning. In Ruml, W., and Hutter, F., eds., *AAAI 2006 Workshop on Learning for Search*, 129–135.

Roberts, M., and Howe, A. E. 2009. Learning from planner performance. *Artificial Intelligence* 173:536–561.

Seipp, J.; Braun, M.; Garimort, J.; and Helmert, M. 2012. Learning portfolios of automatically tuned planners: Detailed results. Technical Report 268, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Vallati, M.; Fawcett, C.; Gerevini, A.; Holger, H.; and Saetti, A. 2011. ParLPG: Generating domain-specific planners through automatic parameter configuration in LPG. In *IPC 2011 planner abstracts*.

Vrakas, D.; Tsoumakas, G.; Bassiliades, N.; and Vlahavas, I. P. 2003. Learning rules for adaptive planning. In Giunchiglia, E.; Muscettola, N.; and Nau, D., eds., *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, 82–91. AAAI Press.