2015 IEEE 27th International Conference on Tools with Artificial Intelligence

Portfolio Methods for Optimal Planning: an Empirical Analysis

Mattia Rizzini*, Chris Fawcett†, Mauro Vallati‡, Alfonso E. Gerevini* and Holger H. Hoos†
*Department of Information Engineering, University of Brescia, Italy
†Department of Computer Science, University of British Columbia, Canada
‡School of Computing and Engineering, University of Huddersfield, United Kingdom

***Abstract*—Combining the complementary strengths of several algorithms through portfolio approaches has been demonstrated to be effective in solving a wide range of AI problems. Notably, portfolio techniques have been prominently applied to suboptimal (satisficing) AI planning.**

Here, we consider the construction of sequential planner portfolios for (domain-independent) optimal planning. Specifically, we introduce four techniques (three of which are dynamic) for per-instance planner schedule generation using problem instance features, and investigate the usefulness of a range of static and dynamic techniques for combining planners. Our extensive experimental analysis demonstrates the benefits of using static and dynamic sequential portfolios for optimal planning, and provides insights on the most suitable conditions for their fruitful exploitation.

***Keywords*-automated planning; optimal planning; sequential portfolio; per-instance portfolio generation.**

I. Introduction

Automated planning is a prominent AI challenge. Within the area of automated planning, cost-optimal (hereinafter, optimal) planning deals with finding optimal plans, i.e., plans that reach a given goal state through an ordered set of actions with minimum total cost. Optimal plans are desirable in many applications.

In recent years, there has been considerable progress in developing powerful domain-independent planners, in no small part spurred on by the International Planning Competitions (IPCs). However, none of these systems clearly dominates all others in terms of performance over a broad range of planning domains. Furthermore, it has been observed that if a planner does not solve a given problem quickly, it will likely not solve it at all [1, 2]. These observations motivate the exploitation of portfolio approaches in planning. In particular, much work has been done in the area of sequential portfolios, where selected algorithms are executed sequentially on a single CPU. Portfolio approaches have been successfully exploited in several other areas, notably SAT solving and answer set programming (ASP) [3, 4].

There are planner portfolio configuration systems mainly designed to generate domain-specific planners, such as PbP and ASAP [5, 6], as well as a range of domain-independent portfolio planners. Among the latter, we can identify two main classes: static portfolios, which run the same schedule of planners on every given problem instance, and portfolios based on per-instance planner schedules. Cedalion [7] and StoneSoup [8] are well-known examples of static portfolio-based planners, while IBaCoP selects the best planner schedule on a per-instance basis [9]. Here, we introduce a third class, that of *dynamic portfolios*, comprised of planners in which the schedule is created dynamically, during execution, based on performance data from earlier runs of the given planners as well as on features of the planning instance to be solved.

Interestingly, we observe that most of the existing work on portfolio-based planners is focused on satisficing planning. However, we note that in IPC-14, some static portfolios (MIPlan and DPMPlan), as a well as two algorithm selection approaches (Nucelar and AllPaca) have been submitted to the optimal track [9]. Furthermore, [10] mined the results of IPC 2011 by using mixed-integer programming to construct sequential portfolios of optimal planners; this turned out to be helpful for assessing the usefulness of different sets of training instances, and for better understanding the performance of planners that took part in IPC 2011.

In the following, we consider the automatic construction of sequential planner portfolios for domain-independent optimal planning. In particular, we introduce four new techniques: two similarity-based approaches and two model-based approaches. Similarity-based approaches select the algorithms to run by considering performance on training instances similar to the given testing instance. Model-based systems generate a model of the performance of each considered component planner, which is then exploited for the selection process. Three of the proposed methods are dynamic portfolio approaches.

The sequential portfolios thus obtained are then compared with static planner portfolios and with PlanZilla, an out-of-the-box application of the SATZilla algorithm selection approach to planning. In an extensive empirical analysis we demonstrate the usefulness of portfolio approaches for optimal planning. In particular, we find that (i) our new model-based and instance-based approaches are more robust in that they generalise better to new domains of planning problems than the static portfolios and PlanZilla; (ii) when the training set is representative of testing problems, our model-based approaches consistently outperform static portfolios.

II. Portfolio-based Optimal Planning

In this section, we provide a description of the sequential planner portfolio approaches considered in our investigation – approaches from the literature as well as four new per-instance approaches. Every portfolio approach considered here requires as input a set of planning algorithms A, a set of training instances I, and performance measurements for the planners in A on I. Here, we measure performance as CPU time required to produce an optimal plan and assign a penalty value if no optimal plan was produced. The Penalised Average Runtime (PAR score) is a real number which counts (i) runs that crash or do not find an optimal plan as ten times the cutoff time (PAR10) and (ii) runs that find an optimal plan as the actual runtime. PAR scores are commonly used in automated algorithm configuration, algorithm selection and portfolio construction, because using them allows runtime to be considered while still placing a strong emphasis on high instance set coverage.

In this work we are considering optimal planners, i.e., planners that only produce optimal solutions to a given planning problem.

A. Static Portfolios

Static portfolios are determined based on the performance of the given set of potential component algorithms on the instances in the training set. They are defined by: (i) the subset of component planners that will be run; (ii) the order in which those planners are to be executed; and (iii) the runtime allocated to each planner. Once configured for a given training set, a static portfolio is not adjusted in any way to the problem instances to be solved by it after training is complete.

In this work, we consider two classes of static portfolios. First, in an approach we will simply refer to as "static portfolio" from here on, we used the Fast Downward Stone Soup hill-climbing technique [8]. With a target of k planner components ("Static k") that can be included in the portfolio, out of a larger set of candidates, and a given limit on the total runtime allocated to the portfolio components, we greedily construct a portfolio: starting with an empty portfolio, in each iteration, we either add a new planner component or we extend the allocated CPU time of an existing planner component; from all such modifications, we choose the one that maximally increases the number of solved problems within the given training set. This process continues until no more than k planner components have been added and the time limit has been reached.

For our second static portfolio approach, we use the greedy schedule construction heuristic of Streeter and Smith [11]. This approach starts with an empty portfolio and iteratively adds the $\langle \mathsf{planner}, \mathsf{runtime} \rangle$ pair that maximises the ratio between additional instances solved and runtime spent. This can be computed efficiently using only the runtimes at which a component planner solved a training instance as a potential runtime choice. We will refer to this approach as "Streeter-style".

B. Problem Instance Features

We now turn our attention towards per-instance portfolio approaches, which depend on a vector $\mathbf{f}$ of *features* to be computed for any given problem instance i. Each feature in $\mathbf{f}$ is a numeric value that reflects a specific property of i, such as the average number of out-edges in i's causal graph, or whether i has action costs. These features are designed to succinctly describe important aspects of the instance, such that similar instances have similar feature vectors.

In this work, we use the feature set and extraction algorithm introduced by [12]. This set contains 311 problem instance features, including simple properties of the PDDL representation, features obtained by translating the problem instance to SAT and using the SAT feature extractors of [3] and [13], and more. To the best of our knowledge, this is the most comprehensive set of features available for planning instances.

C. PlanZilla

PlanZilla is an adaptation of the well-known model-based algorithm selection procedure SATzilla [3] to optimal planning. This and all the following per-instance and dynamic portfolios implement the same general structure, composed of four separate stages: pre-solving, feature extraction, main and backup solving. The pre-solving stage is essentially a greedily-selected static portfolio with a very short runtime cutoff (in this case 1.11% of the total), aimed at solving the easiest problem instances very quickly without expending runtime to compute problem instance features. If the problem instance is not solved by the pre-solving stage, a model is evaluated that uses a very simple reduced set of instance features to predict whether the full set of features will be computable within the given time. If so, the complete feature set is extracted from the problem instance and PlanZilla proceeds to the main stage. Otherwise, PlanZilla switches to the backup solving stage. The PlanZilla main stage makes use of predictive models (in this case Empirical Hardness Models, or EHPs [13]) in order to predict the single best component planner to run on a given problem instance (using the extracted features for that instance), and also predicts the runtime required for that planner to solve the instance. In our case, PlanZilla will execute this selected planner for the entirety of the remaining runtime. If for any reason execution terminates early without producing an optimal plan, PlanZilla switches to the backup solving stage. Finally, the PlanZilla backup solving stage consists of running the single best component planner, as determined by training set PAR10 score, for the remaining available runtime.

Our implementation of PlanZilla uses the default configuration of an early version of a new, general-purpose Java implementation of SATZilla called the *Zilla framework, which was generously provided by the SATzilla team. We do not consider PlanZilla itself to be a contribution of our work presented here. It should also be noted that PlanZilla is not, according to our definition, a dynamic portfolio approach.

D. Pre- and Backup Solvers

In addition to their use in PlanZilla, each of our four per-instance portfolio approaches also makes use of pre-solving and backup solving stages to complement the main portfolio construction stage(s). Pre-solving is performed identically to that of PlanZilla, with 1.11% of the total runtime allocated to a greedily-constructed static portfolio executed before feature extraction is performed. As with PlanZilla, training set instances that are solved by the pre-solving stage are removed from the training set used for the main portfolio construction and backup solving stages.

The backup solving stage, however, is not the same as that used by PlanZilla. The *Zilla backup solving mechanism was designed with the assumption that any failure necessitating the use of the backup solver would come early in any given run (e.g., due to failure to extract features). In the case of planner portfolios this is no longer the case, as a failure can happen during execution of any individual portfolio component. We have therefore extended the backup solving mechanism to also take into account the runtime remaining at the time a backup solver is required. This is done by using incremental runtime cutoffs (with one minute increment) and determining, for each of them, the component planner with the best PAR10 score on the training set when given that runtime cutoff. If the backup solving stage is required during a subsequent run, the selected component planner is that associated to the cutoff closest to the remaining runtime.

In the following subsections, we describe only the different main stages for each of our new portfolio approaches.

E. Similarity-based approaches

Given a planning instance to solve, it is reasonable to select a schedule of planners that performed well on training set instances similar to the given instance. In all four of our approaches, this similarity is determined based on the features extracted from this instance, with two model-free (this subsection) and two model-based (next subsection) approaches. Our two model-free (or "similarity-based") approaches are dynamic portfolios and make use of a notion of *distance* between problem instances in feature space, in this case Euclidean distance after feature normalisation to $[0, 1]$. Our two model-free approaches proceed iteratively, and operate as follows:

Instance-set-core-based. This approach first prunes instances out of the training set that are further than a given boundary cutoff value from the problem instance under consideration. In our experiments, we use an empirically-determined distance cutoff of 4.5, chosen to maximise performance on the considered training set. In each iteration, this approach selects the component planner with the best performance on the remaining training set instances (the *core*), and executes that planner for a runtime t maximising the ratio $\frac{n}{t}$ where n is the number of core instances solved using runtime t (analogous to the procedure for the Streeter-style schedules). After each run of a selected planner P that fails to solve the test instance under consideration, we remove all n training instances from the core that were solved by P in the selected runtime t. If at any point, the core set is empty, the approach proceeds to the backup solving stage.

Weight-based. This approach does not perform any initial pruning of the training set. Instead, it assigns a weight to each training set instance, equal to the distance between that instance and the test instance under consideration. In each iteration, the performance of every component planner is computed as the weighted sum of the PAR10 scores for that planner on each training instance (using instance weights). The planner with the best performance is selected, and as in the instance-set-core-based approach this planner is executed for a runtime maximising the ratio of instances solved to runtime spent. After each failed run, we once again remove all problem instances from the training set that were solved by the selected planner in the selected runtime. If the remaining training instance set becomes empty, the approach proceeds to the backup solving stage.

Both model-free approaches run until the test instance under consideration has been solved or the runtime budget has been exhausted.

F. Model-based approaches

For our model-based approaches, the choices of the next component planner to execute and its runtime are made using empirical hardness models learned from training data, using the *Zilla framework. We implemented a *simplified model-based* per-instance approach and a *full model-based* dynamic portfolio.

Simplified model-based. Given a planning instance to be solved, this approach iteratively selects the next planner to run, and its runtime, based on performance predictions obtained from trained *Zilla models. We train a random decision forest classification model to perform algorithm selection (the next planner to run), and a separate regression forest model for each planner that predicts the runtime required to find an optimal plan for the given instance. After each run of a component planner, that planner is removed as an option from our classification model, to prevent a duplicate selection in the next iteration. We run the selected component planner for the predicted runtime (or the remaining runtime, if that is smaller), until the runtime budget has been exhausted.

Full model-based. This approach uses the same regression models for component planner runtime prediction as for the simplified approach, but the planner selection process is

extended by adding a second classification model (using the same learning techniques as the simplified approach) trained on a feature set that has been extended to take previous failed component planner runs into account. The extended feature set adds a Boolean feature and a real-valued feature for each component planner, indicating whether that planner has already been unsuccessfully run on the given problem instance, and for what runtime. This second model is used after each failed selected planner run for a given test instance to decide the next planner to run, considering the planners already tried and their runtime (the first selected planner is decided using the same "base" classification model of the simplified approach described above).
In order to train this modified classification model, we have to simulate training data for component planner runs, in order to give the model a wide variety of values for the new features. For each component planner a and instance i in the training set, we produce a set of *failure runtimes*. These failure runtimes consist of (i) a short runtime if a crashes and (ii) the runtime t required for a to find an optimal plan for i, plus runtimes slightly below and above t, if a is able to solve i. (Values greater than t are used to take into account randomisation of the planner affecting t.) We then generate the full cross product of component planner combinations (up to a small dimension, given as a parameter of the training process) with their failure runtimes on i and add the resulting features to the training data.

The full model-based approach allows us to utilise information from incorrect predictions to inform subsequent planner selections, but due to the greatly-increased size of the training data requires a significantly greater amount of time for training the classification model. In our experiments reported in the following, we considered combinations of up to 2 planners chosen from the overall best 5, resulting in an overall training time of 1 CPU week.

III. Experimental Analysis

We report the results from a large-scale experimental study, in which we examined the effectiveness of the described portfolio approaches, as well as the performance of the individual planners used as portfolio components.

A. Settings

We considered all of the planners that took part in the optimal track of the 2014 International Planning Competition (IPC-14), namely: AllPaca, cGamer, DPMPlan, Dynamic-Gamer, Gamer, Fast Downward Cedalion, hflow, hpp, hpp-ce, Metis, MIPlan, NuCeLaR, RIDA, Rational Lazy A*, SPM&S, SyMBA*-1, SymBA*-2. (Detailed descriptions of these planning systems can be found in [9].) Hereinafter, we will refer to the participants of the competition as "individual planners", regardless of the approach they exploit for solving planning problems. In our analysis, they are used as basic solver components.

For the sake of readability, we will use the following terminology: **Model-B** refers to our full model-based approach; **S-Model** indicates the simplified version of our model-based approach; **Sim-I** and **Sim-W** refer to the instance-set-core-based and weight-based similarity approaches, respectively. **Static** X denotes the static portfolio using up to X planners.

We focused our study on domains that have been used in the optimal track of IPC-14: Barman, Cave-Diving, ChildSnack, Citycar, Floortile, GED, Hiking, Maintenance, Openstacks, Parking, Tetris, Tidybot, Transport and Visitall. For each domain with a randomised problem instance generator, we generated 200 instances using the same generator parameter setting distribution as in the IPC.

Instances were divided into training, validation and testing sets. For each domain, this was done by randomly partitioning a given instance set as follows: the *training* set was sized to include a sufficient number of instances such that performance varied by at most 10% of the performance on the entire instance set; the *validation* set was chosen to contain 10% of the generated instances; and the *testing* set contains all of the remaining problem instances. Hereinafter, when we refer to the training, validation and testing sets, we mean the sets including the corresponding instances from all of the considered domains. The GED domain unfortunately lacks a random instance generator, so the IPC-14 GED instances have been included in our testing set only.

All of our experiments were run using single cores of a cluster utilising 2.66 Ghz Intel Xeon X5650 CPUs. Memory was limited to 8 GB per planner. For both training and testing purposes, a cutoff time of 1800 CPU seconds was used, as in the optimal track of IPC-14. In some experiments, we also considered a shorter cutoff time of 300 CPU seconds, for the purposes of examining the efficacy of our approaches when runtime is more tightly limited.

In the optimal track of the IPC, planners are usually evaluated by considering only instance set coverage (i.e., number of solved test instances). In our analysis, we evaluate also runtime performance by considering the PAR10 score.

B. Individual Planners and Static Portfolios

It is well known that portfolio approaches exploit the combination of complementary strengths of the available component algorithms. Evidently, there is little point in combining solvers with very similar performance. In order to understand the complementarity of planners that participated in the optimal track of IPC-14, and hence their suitability as components within portfolio approaches, we analysed the performance of individual planners on our complete set of 2620 IPC-14 instances (the union of our previously-mentioned training, validation and testing sets, including the 20 instances from the GED domain). Table I shows the results of this complementarity analysis in terms of coverage and PAR10 scores. Many planners are able to provide high performance on different sets of instances,

Table I
INDIVIDUAL PLANNER PERFORMANCE ON THE 2620 TESTING INSTANCES. THE TOTAL NUMBER OF SOLVED PROBLEMS (SOLVED), THE NUMBER OF PLANNING DOMAINS FOR WHICH THAT PLANNER HAD THE BEST COVERAGE (COLUMN 3), PAR10 SCORE (COLUMN 4), AND THE NUMBER OF PROBLEM INSTANCES FOR WHICH THAT PLANNER EITHER ACHIEVED THE BEST PAR10 SCORE (COLUMN 5) OR WAS WITHIN 1 CPU SECOND OF THE BEST PAR10 SCORE (COLUMN 6).

Planner	Solved	Coverage (dom)	PAR10 (dom)	PAR10 (inst)	1s in PAR10 (inst)
Metis	911	3	1	201	370
D-Gamer	880	1	1	76	80
SyMBA1	862	1	1	119	344
SyMBA2	861	1	0	141	372
MIPlan	812	2	1	45	194
DPMPlan	805	2	0	67	215
Cgamer	791	4	4	289	341
Nucelar	746	2	1	41	49
Gamer	728	0	0	4	7
RlazyA	720	2	0	134	237
All-paca	679	2	0	0	28
SPM&S	651	1	1	45	172
Cedalion	637	2	0	11	122
RIDA	553	2	2	28	35
hflow	484	2	2	191	251
hpp-ce	58	0	0	18	22
hpp	54	0	0	3	18

with all but hpp-ce and hpp solving more than 400 problem instances. Moreover, we observe that there is also a very good distribution of performance at the domain level: nearly all planners have one domain on which they are "best". We note further that this property extends even to instances within the same domain, as several domains have different best planners depending on the problem instance. Therefore, the individual planners considered here are very suitable for combination by means of portfolio approaches. For instance, the Metis planning system, which provides the best overall instance set coverage, provides the best PAR10 results in one domain only. In contrast, hflow does not provide good overall instance set coverage, but it is the best choice for two of the IPC-14 domains. Similar observation can be obtained on a per-instance basis. Gamer shows a peculiar behaviour: it is able to solve a large number of instances, but does not perform particularly well on any domain; it also tends to have high runtimes on instances it manages to solve. This is due to the approach it exploits: Gamer spends half of the CPU time in creating a heuristic through symbolic search. If a solution is then found, it is immediately reported. Otherwise, an abstraction is generated and used for the remainder of the time budget.

Given this promising complementarity between the individual planners under consideration, we evaluated the performance of static portfolios combining these planners. We tested portfolios executing 2, 3, 4 and 5 planners, selected via the mechanism outlined in the previous section. We also generated a Streeter-style portfolio using these planners. The planners included in the largest static portfolio, ordered according to their allocated CPU time, are: Cgamer, Metis, DPMPlan, SyMBA1 and hflow.

Smaller static portfolios include subsets of those 5 planners. From Table II, which shows the performance of all portfolios, the virtual best planner (VBP, representing an oracle which always selects the best solver for the given instance) and individual planners on our testing set, it is clear that all of the static portfolios and the Streeter-style portfolio achieve better results than the individual planners they use as components. The Streeter-style portfolio outperforms the static portfolios with 2, 3 and 4 component planners, but does not reach the performance of the static portfolio of size 5. Unsurprisingly, the larger the number of planners included in a static portfolio, the higher the performance of the portfolio. This is due to the fact that the cutoff time of 1800 seconds allows all of the included planners to run at least for 5 minutes, which is usually enough to optimally solve most of the test instances. We empirically observed that performance does not further increase for even larger static planner portfolios. The static portfolio generation techniques recognise that additional component planners will have insufficient time to increase overall portfolio performance and therefore do not include them.

Table II also shows the performance of the static and Streeter-style portfolios with a shorter cutoff time of 300 CPU seconds. In this setting, static portfolios are still able to provide better performance than the individual planners. However, using this lower cutoff time, the static portfolio generation always selects 2 planners for all of the static portfolio sizes, namely Metis and SyMBA1. This is due to the fact that adding more planners further reduces the already limited runtime available for each component. Unlike when using a 1800 CPU second cutoff, with a 300 CPU second cutoff, the Streeter-style portfolio greatly outperforms the static portfolios, solving 76 instances more.

C. Per-instance Portfolio Performance

In order to evaluate the performance of our four per-instance approaches, as well as that of PlanZilla, we trained each approach using our IPC-14 training set and evaluated the result on the corresponding held-out test set. The runtime cutoff for solving each problem instance was 1800 CPU seconds. Instance set coverage and PAR10 scores for each portfolio approach are reported in Table II, showing that the model-based approaches substantially outperform the static portfolios and similarity-based approaches. In this scenario, the 5-planner static portfolio outperforms the instance-set-core similarity method, and the weight-based similarity method is further outperformed by the 4-planner static portfolio and the Streeter-style schedule. However, even the similarity-based approaches perform better than all of the individual planners. The fact that the training and test sets were sampled from the same underlying distribution is to PlanZilla's advantage, as its single planner selection is likely

Table II
NUMBER OF INSTANCES SOLVED AND PAR10 SCORES FOR THE PLANNING SYSTEMS CONSIDERED IN OUR STUDY, EVALUATED WITH 1800 AND 300 CPU SECOND RUNTIME CUTOFFS ON OUR IPC-14 TEST SET. VBP INDICATES THE PERFORMANCE OF THE VIRTUAL BEST PLANNER, AND GREY ROWS INDICATE PORTFOLIO-BASED PLANNERS. PLANNERS ARE LISTED IN THE ORDER OF INCREASING PAR10.

1800 Second Timeout			300 Second Timeout		
System	Sol.	PAR10	System	Sol.	PAR10
VBP	706	9355.3	VBP	609	1757.6
PlanZilla	677	9725.3	PlanZilla	542	1901.6
S-Model	650	10051.2	S-Model	515	1952.8
Model-B	632	10269.8	Streeter	484	2006.1
Static 5	610	10527.5	Model-B	482	2018.4
SIM-I	603	10639.1	SIM-I	455	2068.8
Streeter	586	10786.6	SIM-W	422	2134.4
Static 4	582	10847.6	Static 5	408	2155.4
SIM-W	575	10997.7	Static 4	408	2155.4
Static 3	558	11132.7	Static 3	408	2155.4
Static 2	499	11831.1	Static 2	408	2155.4
Metis	441	12534.3	SyMBA2	364	2239.6
D-Gamer	410	12914.7	SyMBA1	363	2241.7
MIPlan	384	13223.2	Metis	343	2285.0
SyMBA1	378	13236.2	Cgamer	318	2335.0
SyMBA2	378	13236.4	DPMPlan	315	2339.2
DPMPlan	378	13264.1	D-Gamer	289	2396.5
Cgamer	378	13266.2	MIPlan	273	2424.6
Nucelar	350	13620.4	RlazyA	254	2458.8
RlazyA	331	13840.3	Nucelar	255	2461.8
Gamer	322	13961.3	SPM&S	242	2482.0
All-paca	306	14133.1	Cedalion	243	2482.7
SPM&S	297	14234.8	All-paca	237	2492.7
Cedalion	292	14293.8	Gamer	220	2527.5
RIDA	256	14741.2	hflow	191	2582.2
hflow	239	14922.4	RIDA	180	2615.7
hpp-ce	23	17487.4	hpp-ce	19	2922.7
hpp	20	17523.7	hpp	14	2932.3

to be correct and the selected planner exploits the large runtime.

To investigate the performance of our per-instance approaches when given a much smaller runtime cutoff, we performed another set of experiments with the same training and test sets, but using a 300 CPU second runtime cutoff. We observed similar results as for the 1800 CPU second cutoff, but in this case, the similarity-based approaches now outperformed the static portfolios. Interestingly, the Streeter-style schedule performs very well in this case, and its performance was only exceeded by that of PlanZilla and our simplified model-based approach. We believe that the high performance of the Streeter-style schedule is due to training and test set being drawn from the same distribution, and that by design, this approach performs short runs of many planners. Given a reasonably good selection of planners, and considering the fact that most of the benchmarks can be solved quickly, the observed performance of the Streeter-style approach is not surprising.

D. Generalisation Performance

In order to test the generalisation of all considered approaches to instances dissimilar from those found in the training set, we performed two additional experiments. Because of its high training time (which would have added up to several months of computation), we excluded the full model-based approach from this part of our study.

Our first generalisation experiment involved removing all instances from one domain at a time from our IPC-14 training set, training each approach using this new training set, and then evaluating the result on all problem instances from the held-out domain. As before, we used 1800 CPU seconds as the runtime cutoff for solving each instance. We call this experiment "leave-one-domain-out". In Table III, we present the resulting per-domain instance set coverage.

The Streeter-style schedule performed best in this scenario, followed by the two similarity-based approaches and the static portfolios. Our simplified model-based approach and PlanZilla both fail to generalise as well, and are outperformed by the two SyMBA planners and Metis, respectively; we believe that this is due to the models becoming overly specialised for the given training set.

Our next generalisation experiment took the "leave-one-domain-out" approach further and used a training set containing no problem instance from any of the IPC-14 domains. Instead, we used domains from the optimal tracks of IPC 2008 and 2011 that were *not* used in IPC-14. We trained all portfolio-based planners using this new training set and evaluated the result on our IPC-14 test set. This was done with an 1800 CPU second runtime cutoff as well as with a 300 CPU second cutoff. The resulting test set coverage and PAR10 scores are summarised in Table IV.

First, we note that the Metis planner now outperforms all other approaches on the test set. After further investigation, we determined that the Metis planner was frequently not the best (or even a good) planner on the domains of our training set, leading to Metis not being selected often for problem instances in our test set. This is, of course, a known downside to having a test set that is greatly dissimilar from the instance set used for training.

We note that on this scenario, the static portfolios drop in performance, while the performance of the similarity-based approaches increases. Moreover, our proposed approaches seem to have better generalisation performance than PlanZilla, likely due to having multiple attempts at selecting the "right" planner for each problem instance. From these generalisation experiments, it appears that different portfolio approaches work best under different circumstances: the model-based approaches are often best in situations where the test instances are likely to be largely similar to those used for training. The similarity-based approaches often perform better when the test set contains many instances from domains not used during training.

E. Importance of Pre- and Backup Solvers

Dynamic portfolio approaches use the pre- and backup solvers for two distinct purposes. Pre-solving aims at quickly

Table III

Number of instances solved per domain by each of the approaches considered in our study, in the "leave-one-domain-out" scenario. This allows for a rudimentary analysis of generalisation performance. Domains, from left: Barman, Cave-Diving, ChildSnack, Citycar, Floortile, Hiking, Maintenance, Openstacks, Parking, Tetris, Tidybot, Transport and Visitall.

Planner	BM	CD	CS	CC	FT	H	M	OS	P	T	TB	TP	VA	Total
VBP	34	35	77	126	200	118	20	152	51	166	44	98	97	1218
Streeter	10	**35**	41	98	186	106	16	121	1	**154**	21	82	69	940
SIM-I	8	**35**	40	84	185	104	7	143	7	140	24	77	62	916
SIM-W	10	**35**	41	90	186	105	16	119	1	144	18	70	59	894
Static 2	10	**35**	52	73	186	107	0	138	3	140	30	72	16	862
Static 3	10	**35**	52	73	186	107	0	138	3	140	30	72	16	862
Static 4	10	**35**	52	73	186	107	0	138	3	140	30	72	16	862
Static 5	10	**35**	52	73	186	107	0	138	3	140	30	72	16	862
SyMBA1	12	16	19	90	186	110	0	143	1	126	23	86	15	827
SyMBA2	12	16	20	89	186	110	0	143	0	126	21	**87**	15	825
S-Model	12	33	33	81	155	102	0	122	12	119	25	81	29	804
Metis	0	**35**	67	**126**	90	101	0	30	9	135	34	74	15	716
PlanZilla	4	16	19	110	95	105	0	123	8	117	24	73	14	708
Cgamer	**34**	16	40	0	**200**	**112**	0	**152**	0	0	0	29	**97**	680
DPMPlan	3	**35**	10	53	70	100	16	54	44	141	22	83	15	646
D-Gamer	0	16	**77**	81	134	109	6	101	0	0	3	82	24	633
MIPlan	1	**35**	1	53	55	98	16	14	45	143	24	67	14	566
RlazyA	0	**35**	0	107	62	69	0	18	9	126	31	67	32	556
All-paca	0	**35**	0	105	66	69	0	23	4	122	29	63	13	529
Gamer	0	16	11	92	122	102	4	95	0	0	4	71	8	525
SPM&S	9	16	10	13	168	**112**	0	54	0	100	4	17	21	524
Nucelar	0	16	4	0	60	105	0	17	**51**	138	**35**	**87**	10	523
Cedalion	0	**35**	0	89	58	74	0	12	8	132	30	66	18	522
hflow	0	1	0	0	49	33	0	0	1	**154**	1	43	**97**	379
RIDA	0	0	0	100	0	65	**20**	0	0	52	32	85	9	363
hpp-ce	0	0	0	26	0	14	0	0	0	0	0	0	0	40
hpp	0	0	0	22	0	17	0	0	0	0	0	0	0	39

solving easy instances, for which extracting features would be wasteful. Backup solvers are used if the feature extraction process is believed to be infeasible within the given amount of time (or extraction fails), or in case of failure of the main selected solver(s). From this perspective, the purpose of backup solvers can be understood as minimising the impact of poor algorithm selection, while pre-solvers are used as an optimisation for improving the overall runtime.

Table V shows the percentage of problems solved by each stage of our dynamic portfolio approaches; namely pre-, main and backup solving, when using a 1800 CPU second cutoff on our testing set. We observe that the backup solver is rarely exploited, and only the model-based systems use it successfully. This is possibly due to the fact that the backup solver is run only in exceptional cases, such as when feature computation or model evaluation fails. On the other hand, Table V clearly shows that pre-solvers are extremely important and responsible for solving a significant percentage of the instances. Given the very limited CPU time available for the pre-solver (1.11% of the cutoff time, around 20 CPU seconds in our experiments), this result is a clear indication that a large number of the benchmarks can be solved in a short amount of time by a single solver. We note that this is especially the case for the Floortile domain.

In order to investigate the contribution of the pre-solving stage to the performance of the approaches considered here, we re-ran all of our approaches with the pre-solving mechanism disabled. Interestingly, we observed that the impact on instance set coverage was much smaller than expected. The most affected system is our full model-based approach (Model-B), for which disabling pre-solving results in 1.2% fewer instances solved. Apparently, the main solver stage is generally able to solve most of the instances usually solved by the pre-solving stage; we also noticed an increase in the exploitation of backup solvers, which are now used in up to 10% of the solved instances. This suggests that pre-solving is not fundamental in terms of coverage, but is useful for improving the runtime of a portfolio approach. Evaluating the usefulness of the backup solver is straightforward: since it is used only when other steps fail or are believed to be infeasible, its impact can be measured directly as the percentage of instances solved by the backup stage, which was very low in our experiments.

IV. Conclusions

In this paper we introduced four new per-instance portfolio techniques exploiting the largest set of planning features currently available [12]. Two of our approaches are model-free and based on similarity metrics in instance feature space. The other two techniques are model-based and iteratively select the next solver to run by considering instance features as well as information about previous failed selections. We compared the performance of these new approaches with that of several static portfolio methods

Table IV
NUMBER OF INSTANCES SOLVED AND PAR10 SCORES FOR THE PLANNING SYSTEMS CONSIDERED IN OUR STUDY, TRAINED ON THE IPC 2011 BENCHMARKS, ON THE 2014 INSTANCES. GREY INDICATES THE INVESTIGATED PLANNER PORTFOLIOS, WHILE VBP INDICATES THE PERFORMANCE OF THE VIRTUAL BEST PLANNER. SYSTEMS ARE LISTED FOLLOWING INCREASING PAR10 ORDER.

1800 Second Timeout			300 Second Timeout		
System	Sol.	PAR10	System	Sol.	PAR10
VBP	652	8021.9	VBP	557	1338.0
Metis	535	9638.2	Metis	535	1418.2
S-Model	526	9779.2	S-Model	447	1563.5
Sim-I	507	10041.8	Streeter	438	1571.7
Sim-W	508	10051.2	Sim-W	435	1583.4
Streeter	504	10062.2	Sim-I	432	1589.6
PlanZilla	501	10121.1	PlanZilla	427	1600.9
Static 4	472	10517.7	DPMPlan	407	1652.8
Static 5	470	10543.0	Static 5	395	1660.7
Static 3	441	10934.9	Static 3	395	1660.7
Static 2	420	11221.8	Static 2	395	1660.7
DPMPlan	407	11408.8	Static 4	395	1660.7
MIPlan	407	11420.1	MIPlan	407	1664.1
RlazyA	389	11664.8	SyMBA*1	381	1687.9
D-Gamer	392	11679.4	SyMBA*2	380	1689.6
SyMBA*1	381	11755.9	RlazyA	389	1692.7
SyMBA*2	380	11769.6	AllPaca	374	1718.0
Cedalion	380	11799.5	Cedalion	380	1719.5
AllPaca	374	11870.0	D-Gamer	392	1743.4
RIDA	351	12246.5	RIDA	351	1818.5
Nucelar	318	12664.3	Nucelar	318	1840.3
SPM&S	307	12816.6	SPM&S	307	1860.6
Gamer	285	13137.5	Gamer	285	1917.5
hflow	230	13880.7	hflow	230	2000.7
Cgamer	185	14508.5	Cgamer	185	2088.5
hpp-ce	58	16281.1	hpp-ce	58	2337.1
hpp	54	16333.1	hpp	54	2341.1

Table V
PERCENTAGES OF INSTANCES SOLVED BY THE PRE-SOLVING, MAIN, AND BACKUP STAGES OF OUR FOUR PER-INSTANCE PORTFOLIO APPROACHES, AS WELL AS BY PLANZILLA. WE ALSO INCLUDE THE PERCENTAGE OF INSTANCES LEFT UNSOLVED BY EACH APPROACH.

	Pre	Main	Backup	Unsolved
PlanZilla	15.0	31.0	0.0	54.0
Model-B	14.0	28.0	1.0	57.0
SIM-I	20.0	21.0	0.0	59.0
SIM-W	20.0	19.0	0.0	61.0
S-Model	20.0	24.0	1.0	56.0

and with the performance of PlanZilla, an out-of-the-box application of the SATzilla algorithm selection system [3].

The results of our extensive empirical analysis showed that: (i) the planners from the optimal track of IPC-14 have an high level of complementarity and can thus be fruitfully combined using portfolio approaches; (ii) if the training instances are representative of testing instances, portfolio-based planners achieve better performance than any individual planner; (iii) when training and testing sets include problem instances taken from the same distribution, the newly model-based approaches consistently outperform the static portfolios, while the similarity-based approaches match the performance of the static portfolios; (iv) when the testing set includes multiple domains not found in the training sets, both the new model-based and similarity-based approaches outperform the static portfolios; and (v) our model-based and similarity-based approaches appear to generalise better to previously unseen domains than PlanZilla.

We see several avenues for future work. Firstly, we are interested in further investigating the generalisation performance of the methods considered in our study. Secondly, we see promise in studying less expensive training techniques for the full model-based approach. Finally, we plan to apply our new portfolio approaches to other areas of planning.

REFERENCES

[1] A. Howe and E. Dahlman, "A critical assessment of benchmark comparison in planning," *JAIR*, vol. 17, pp. 1 – 33, 2002.

[2] M. Helmert, G. Röger, and E. Karpas, "Fast downward stone soup: A baseline for building planner portfolios," in *Proceedings of the PAL workshop*, 2011, pp. 28–35.

[3] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: portfolio-based algorithm selection for sat," *JAIR*, pp. 565–606, 2008.

[4] H. Hoos, M. Lindauer, and T. Schaub, "claspfolio 2: Advances in algorithm selection for answer set programming," *Theory and Practice of Logic Programming*, vol. 14, no. 4-5, pp. 569–585, 2014.

[5] A. Gerevini, A. Saetti, and M. Vallati, "Planning through automatic portfolio configuration: The pbp approach," *JAIR*, vol. 50, pp. 639–696, 2014.

[6] M. Vallati, L. Chrpa, and D. E. Kitchin, "ASAP: an automatic algorithm selection approach for planning," *IJAIT*, vol. 23, no. 6, 2014.

[7] J. Seipp, S. Sievers, M. Helmert, and F. Hutter, "Automatic configuration of sequential planning portfolios," in *Proceedings of AAAI*, 2015.

[8] J. Seipp, M. Braun, J. Garimort, and M. Helmert, "Learning portfolios of automatically tuned planners," in *Proceedings of ICAPS*, 2012, pp. 369 – 372.

[9] M. Vallati, L. Chrpa, and T. L. McCluskey, "The 2014 IPC: Description of Participating Planners of the Deterministic Track," 2014.

[10] S. Núñez, D. Borrajo, and C. L. López, "Performance analysis of planning portfolios." in *Proceedings of SOCS*, 2012.

[11] M. Streeter and S. Smith, "New techniques for algorithm portfolio design," in *Proceedings of UAI*, 2008, pp. 519–527.

[12] 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 *Proceedings of ICAPS*, 2014.

[13] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79 – 111, 2014.