Directing a Portfolio with Learning *

Mark Roberts and Adele E. Howe
Computer Science Dept., Colorado State University
Fort Collins, Colorado 80523
mroberts,howe@cs.colostate.edu
http://www.cs.colostate.edu/meps

Abstract

Algorithm portfolios are one approach designed to harness algorithm bias to increase robustness across a range of problems. We conjecture that learning based on the previous performance of a suite of *planning systems* can direct a portfolio strategy at each of three stages: *selecting* which planners to run; *ranking* the order to run the selected planners; and *allocating* computational time to them. Our specific focus in this paper is to examine ways to inform portfolio strategy by exploiting learned performance models. We cull the planners to the non-dominated subset and rank them based on the models. Allocation is especially challenging for hard problems because the run-time distributions typically exhibit heavy-tails and high variance, both of which lower confidence for run-time prediction. We examine a round robin allocation strategy based on the run-time distributions of the planners. Our results show that the portfolio can solve more problems than any single planner and that it is faster than the average successful planner.

Introduction

Understanding previous performance is often a key to gaining insight into algorithm design for hard problems such as MAX-3SAT or classical planning. Algorithms typically exploit problem-specific structure for such problems through a well-founded or ad-hoc intuition about what will or won't work. For example, Tabu search is (in part) designed to overcome plateaus in a search space. Meta-search techniques that perform localized or elitist restarts implicitly leverage the proximity of local and global minima ('big valleys'). But, it is well known that a problem-specific approach can lead to an algorithm that does well on one problem but fails on other problems with a markedly different structure. No free lunch proofs help us understand this observation, at least for the discrete case (Wolpert & Macready 1997).

Portfolios are one way to transcend algorithm bias and maintain robustness across a range of problems (Huberman, Lukose, & Hogg 1997; Gomes & Selman 1997). A sequential portfolio controls the run time of a suite of algorithms with a strategy that *selects* which algorithms to run, *ranks* them, and *allocates* computational time to them. An ideal strategy maximizes success and minimizes total computation, so the strategy must be accurate and fast.

Our general outline for constructing a portfolio strategy based on learning from previous performance is: 1) Construct models to predict *success* using features extracted from the domain, problem instance, and state space topology. 2) Use these models to rank them. 3) Allocate time based on the run-time performance of the algorithms on a training set.

We followed this strategy to construct a portfolio of planning systems. The International Planning Competitions (IPCs) encourage a variety of planners that are mostly made publicly available. Although some planners excel in the competition, no single planner can solve all benchmark problems. This argues for a portfolio approach. In this paper, we examine each stage of the portfolio strategy by addressing two key questions:

1. The 57 features we included for modeling have computational costs varying by at least four orders of magnitude. Can we select features that minimize total computational cost but still maintain overall model accuracy?
2. We note that the run-time distributions (RTDs) for classical planners are heavy tailed. In spite of this, can we use the RTDs to relate portfolio quitting time with confidence of success?

These questions focus on ways to improve the learning accuracy and to better leverage the learned models; we empirically assess them in the sections that follow. A long-term goal of this work is to use these analyses to drive deeper insight into the behavior of planning systems. So we also present some results that aren't directly used in the portfolio but provide a foundation for this goal.

Related Work

Portfolio learning was originally called the algorithm selection problem (Rice 1976). Most portfolio strategies use models of run-time that are statically constructed (off-line) prior to use in the portfolio, but some models are dynamically constructed (on-line) as the portfolio learns a good strategy. These models are constructed with one or more features that highlight domain knowledge, problem instance characteristics, or structural properties of the search space.

*This research was sponsored by the National Science Foundation under grant number IIS-0138690.
Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Some features are more costly to compute than others because they construct specialized data structures or they directly probe the search space. Finally, there is a difference among allocation strategies. Some portfolios select the best algorithm in a winner-take-all approach, while others dynamically reallocate computational resources based on solution quality measurements. Early portfolio-like approaches such as (Gratch & Chien 1996) and (Minton 1996) learned more effective application of existing search control heuristics.

Two portfolios in the literature incorporate an on-line strategy. An early formulation used a measure of the expected gain to select among three commitment strategies in search for partial plans (Fink 1998). More recently, Beck and Freuder examined a "low-knowledge" approach to algorithm selection that uses three simple first-order features of solution quality to reallocate the remaining time of the portfolio (2004).

Most portfolios use off-line analysis to relate features to runtime. One of the first formulations used the run-time distribution of a Las Vegas algorithm to calculate the best ratio for allocating time two two independent instances of that algorithm (Huberman, Lukose, & Hogg 1997). Gomes and Selman (1997) explicitly computed the value of multiple restarts of a stochastic algorithm for the Quasigroup Completion Problem. Baptista and Silva (2000) followed observations from (Gomes & Selman 1997), but studied selection among *different* randomized algorithms to solve *unsatisfiable* instances of SAT. The Bus meta-planner (Howe *et al.* 1999) used linear regression models of computation time and success to rank a set of classical planners; it allocated run-time using a round robin scheme.

Lagoudakis, Littman and Parr examined a recursive MDP formulation to select among three algorithms for the canonical sorting problem (2000) . Lagoudakis and Littman expanded this work to select an optimal policy of branching rules in search for SAT (Lagoudakis & Littman 2001). Horvitz et al. (2001) as well as Guo and Hsu (2004) both used inductive learning over a Bayesian network of the feature space to guide algorithm selection.

Leyton-Brown et al. examine algorithm selection for Combinatorial Auctions (2003a,2003b) and find that linear regression of the log time variable predicted runtime quite well; they also capped runs that took longer than a maximum time. Nudelman et al. extend this work by creating models that predict runtime for uniform random 3-SAT problems and assembling a SAT solver, SATzilla, for the 2003 and 2004 SAT competitions (2004).

Guerri and Milano (2004) extend work of (Leyton-Brown, Nudelman, & Shoham 2002) to select between Integer Programming or Constraint Programming for the Bid Evaluation Problem. Gebruers et al. (2004) extend this work to use model free learning but it is unclear how the results compare to their earlier work. Gebruers et al. also examine case based learning as applied to the Social Golfer Problem (2005) .

Choosing appropriate features is a key issue for algorithm selection; it is preferable to identify linear or, at worst, polynomial features for a problem. Most of the above strategies model simple features of the specific problem instance or the general problem family; these features are often selected for their known (or suspected) correlation with search cost. Some, such as (Gebruers *et al.* 2005; **?**), use wrapper functions to select subsets of relevant features. One dynamic strategy incorporated first and second order runtime features to measure solution progress (Beck & Freuder 2004). (Nudelman *et al.* 2004) used search probing to generate search space features. This work, along with (Leyton-Brown, Nudelman, & Shoham 2002) also examines using more sophisticated combinations of features for regression.

Planner	Authors [Date]
AltAlt-1.0	Nguyen,Kambhampati, Nigenda [2002]
BlkBox-4.2	Kautz, Selman [1999]
CPT-1.0	Vidal, Geffner [2004]
FF-2.3	Hoffmann [2001]
HSP-2.0	Bonet, Geffner [2001]
IPP-4.1	Koehler, Nebel [1997]
LPG-1.1	Gerevini, Serina [2002]
LPG-1.2	Gerevini, Saetti, Serina [2003]
LPG-TD-1.0	Gerevini, Saetti, Serina [2005]
Metric-FF-02	Hoffman, [2003]
MIPS-3	Edelkamp [2003]
Optop-1.6.19	McDermott, [2005]
Prodigy	Veloso et al. [1995]
SystemR	Lin [2001]
SAPA-2	Do, Kambhampati [2003]
Satplan04	Kautz, Selman [1999]
SGP-1.0h	Weld, Anderson, Smith [1998]
SGPlan-04	Chen, Hsu, Wah [2004]
SimPlan-2.0	Sapena, Onaindia [2002]
SNLP-1.0	McAllester, Rosenblitt [1991]
STAN-4	Long, Fox [1999]
UCPOP-4.1	Penberthy, Weld [1992]
VHPOP-2.2	Younes, Simmons [2003]

Table 1: The planners used, their authors and dates of their publications.

Experiment Design

We study 23 classical planners, denoted in Table 1, which are composed of IPC competitors plus non-IPC planners to diversify the set of approaches represented. None of these planners was intended for use in a portfolio, though Prodigy and BLACKBOX-4.2 are portfolio-like in their approach.

Classical planners work in a variety of search paradigms that obscure straightforward measurement of solution progress. Many of the planners, though not all, sacrifice completeness. Also, some planners can run for an exceedingly long time without producing a solution or proving one doesn't exist. Based on these observations, we construct two classification models: *success* and *time*. Success predicts a binary variable while time predicts computation time needed for a given planner to complete a given problem. Together, these off-line models will form part of the portfolio strategy.

Statistics	Characteristics	Description
num	operators	
num	predicates	
min,mu,max	params-pred	arity for predicates
min,mu,max	prec-oper	predicates in precond.
min,mu,max	post-oper	predicates in effects
min,mu,max	neg-post-oper	negations in effects
num,ratio	neg-post-oper	actions over negative effects
req	adl	req. ADL
req	cond-effects	req. conditional effects
req	derived-pred	req. derived predicates
req	disjunctive-prec	req. disjunctive precond.
req	domain-axioms	req. domain axioms
req	equality	req. equality
req	exist-prec	req. existential precond.
req	fluents	req. fluents
req	quant-pred	req. quantified precond.
req	safety-const	req. safety constraints
req	strips	req. strips
req	typing	req. typing
req	univ-prec	req. universal precond.
num	goals	# of goals
num	objects	# of objects
num	inits	# of inits
min,mu,max	tail-actions	operators unifying to goals
num	toggles	pairs of clobbering actions
num	invariant-pred	disunified predicates
num,mu-p	minima	#,mean %
num,mu-p,mu-dia,mu-sz	benches	#, mean %, diameter, size
num,mu-p,mu-dia,mu-sz	contours	#, mean %, diameter, size
num,mu-p,mu-dia,mu-sz	solutions	#, mean %, diameter, size
num	plains-to-min	plateaus leading to minima
num	plains-to-ben	plateaus leading to benches
mu-dia,mu-sz	lm	mean diameter/size of local min
mu-p	bench-exits	% of bench exits
mu-med	benches	maximal edit distance

Table 2: The feature set: first two columns form the names we use in the paper; last column briefly describes the feature.

The models are trained using observed planner performance on benchmark problems. We run all planners on 3959 STRIPS PDDL problems from 77 domains. The problems are taken from Hoffmann's dataset (Hoffmann 2004), the UCPOP Strict benchmark, IPC sets (IPC1, IPC2, IPC3 Easy Typed and IPC4 Strict Typed) and 37 other problems from two domains (Sodor and Stek) that have been made publicly available. Following typical time and memory constraints from the community, each planner is allowed 30 minutes and 768 Meg on 22 identically configured Pentium 4 3.4Ghz computers. A planner is considered to have failed if it reaches 30 minutes. No planner solves all the problems in the set; the planner that solves the most problems achieves 73.9% success.

Type	Min	μ	Median	Max
Domain & Instance	0	0.00028	0	.035
Interaction	0	5.203	0	5052
Hoffmann (1839)	0	2.187	0	1650

Table 3: Computation costs (in seconds) for features by type.

We use 57 features (Table 2) that are automatically extracted from problem and domain definitions. The set starts with features from (Howe *et al.* 1999) and (Hoffmann 2001) and adds others. As shown in Table 2, we divide the features into four categories of increasing knowledge and computational cost: domain specific, instance specific, action interaction and Hoffmann's state space topology[1] (Hoffmann 2001). In the table, 'mu' refers to the mean value; the required ('req') features are taken from PDDL requirements annotations from the domain files. We use the requirements as defined in the original PDDL specification (Ghallab *et al.* 1998). In practice, not all domains correctly specify their requirements (Howe *et al.* 1999). However, enough do that it may provide useful information since some planners cannot solve problems with certain requirements (such as ADL).

We use the WEKA data mining package (Witten & Frank 2005) to build the models. We tried several different models from WEKA; to begin with, we focused our work on two simple models that worked well: OneR and J48. OneR selects the single feature that yields the highest prediction value on the training set. It provides two important details for us: first, it provides a very simple evaluation of the most important feature for the data. This will be useful in later analysis of which features distinguish planner performance. Second, it gives us a baseline against which to judge more sophisticated machine learning techniques. J48 is a simple decision tree based on Quinlan's C4.5; decision trees were used for prediction in (Gebruers *et al.* 2005; Guo & Hsu 2004; Guerri & Milano 2004; Horvitz *et al.* 2001). By default, we use 10-fold cross validation to train and test the models.

Feature Selection

Our first question relates to identifying a subset of features that is cheap to compute. As shown in Table 3, the four types of features vary considerably in their cost to compute over all problems. We computed times for Domain/Instance and Interaction features over all problems, which meant that some of the Interaction times were longer than 30 minutes. Calculating the topological features involves analyzing the full planning graph induced by the h^+ heuristic, so we include only the subset of 1839 problems for which this analysis is possible. We call the domain and instance features the 'fast' features.

Restricting the models to only fast features significantly extends the number of problems that can be modeled, but also reduces the accuracy. To assess the impact on accuracy, we computed two sets of J48 models: one for the full feature set using the subset of 1839 problems accessible to all

[1]We thank Jörg Hoffmann for supplying the code.

Problems/ Metric	Feature Set: Full μ	Full Sd	Fast μ	Fast Sd	Fast-req μ	Fast-req Sd
Subset						
succ	96.63	3.42	96.03	4.08	95.85	4.08
time	96.16	5.80	95.97	6.17	95.98	6.14
All						
succ	–	–	95.56	1.85	95.47	1.85
time	–	–	94.30	3.07	94.30	3.12

Table 4: Comparing average accuracy (% correct) for models with all features (Full), all fast features (Fast), and fast without the language requirements (Fast-req).

```
req-typing = t
|   req-adl = t: nil (1169.0/6.0)
|   req-adl = nil
|   |   num-objects <= 29
|   |   num-objects > 29: nil (614.0/7.0)
req-typing = nil
|   mu-prec-oper <= 6
|   |   num-inits <= 185
|   |   num-inits > 185: nil (25.0)
|   mu-prec-oper > 6
|   |   min-params-pred <= 0: nil (202.0)
|   |   min-params-pred > 0
```

Figure 1: The first three levels of the decision tree for CPT-1.0. This tree had 103 nodes with 52 leaves.

features and another for the fast features only using all problems. Table 4 compares the accuracy results for the models generated for the two metrics given the computable subset for training when using either the full feature set or only the fast features. The last two lines in the table also show accuracy of the fast on the full problem set.

Table 4 shows only minor degradation in performance when only the fast features are used: slightly lower mean and higher standard deviation. A paired sample T-test comparing classification accuracies for Full and Fast on the *Subset* is insignificant ($p = 0.16$) for time, but highly significant for success ($p < .001$) with an average difference of 0.6%. Thus, it does not appear that the expensive features are necessary to model time and marginally informative for success. A paired sample T-test between Fast and Fast-req for all planner models was insignificant for both metrics ($p = 0.09$ for success and $p = 0.40$ for time). Thus, the evidence suggests that adding the language requirements also lacks significant impact on prediction accuracy.

The decision trees using the 'req' features are qualitatively different even though accuracy did not significantly increase. Nine of the planner models use a requirement as the root feature (5 were `req-typing`, 2 were `req-strips` and 2 were `req-adl`). In general, it seemed to be that language features distinguished problems that were further refined by the other features. Figure 1 shows an example of this trend for the first three levels for CPT-1.0. The features `req-typing` and `req-adl` distinguish problems in the first and second layer, but then other fast features show up lower in the tree. The lack of statistically significant change in accuracy with the strong use of these features suggest that the they may be strongly correlated with other features. The resulting decision trees appear to have stronger explanatory power for our next stage of analyzing the trees to better understand what features distinguish planner performance.

Quitting Time

Our second key question is whether the amount of time allocated to each planner can be predicted with sufficient accuracy to support the portfolio. We propose using the RTDs to arrive at a more informed allocation strategy that will make good use of extra time when justified. Table 5 (left) shows the log distributions of time-to-success for the non-dominated planners. It is clear that we do indeed see heavy tails in most of the planner RTDs. 84.5% of the runs finish in less than one second while only 0.24% finish in greater than 1000 seconds. Failures take much longer; 73.9% take less than a second and 10.7% take longer than 1000 seconds. Over all runs, 77.8% finish in under a second and 6.9% finish in over 1000 seconds.

A simple and intuitive approach to using these data for allocation is to predict time with decision trees similar the success models. Using log-sized bins, J48 classifies all problems (regardless of success) across all planners with an average accuracy of 94.3% (sd of 3.07). Though these predictions have high accuracy, the larger bins are not informative enough to predict exact run-time in the portfolio.

The predictions may also be useless for reasons related to using classification. Recent work from Leyton-Brown et al. points out that regression techniques are preferable to classifiers because they penalize large misclassification more than small ones (according to the strength of the distance metric used for regression) and because they don't rely on arbitrary binning boundaries (2002). But previous research showed simple regression models were not particularly well suited to predicting run-time for automated planners (Howe *et al.* 1999). They used weighted regression over five standard domain and problem features. Those regression models failed to adequately explain variance for runtime; the average R^2 value for the six planners was 0.42 and values ranged from 0.19 to 0.76. We also looked at the extent to which the classifier was off by one bin or more than one bin. 3% of the classifications were off by one bin, 1% by two bins, 0.6 % by three bins, and 0.7effect of large misclassification has less of an impact than the artificial binning boundaries.

Other issues appear to make regression less appropriate for our application. It isn't readily clear which features (or weighting of feature combinations) correlate well with runtime; a scatter-plot of the data did not reveal discernible relationships. Regression over single features does not appear justified because of strong deviations from the normality and variance assumptions. (Nudelman *et al.* 2004) propose two potentially valuable approaches of using a feature space that is the pairwise product of the original features and using logistic models (we were already considering exponential

	Time-to-success						Time Percentiles							
	1	10	10^2	10^3	10^4	TOTAL	0.8	0.85	0.9	0.95	0.96	0.97	0.98	0.99
SGPlan-04	2839	52	9	24	2	2926	0.0	0.1	0.2	0.4	0.5	1.0	3.0	15.3
IPP-4.1	2211	42	41	38	1	2333	0.0	0.0	0.1	1.3	4.2	18.6	50.2	173.7
LPG-TD-1.0	1858	144	111	46	4	2163	0.3	0.8	4.3	32.3	46.7	73.8	142.6	442.9
Satplan04	1064	129	66	62	7	1328	1.0	2.8	10.7	113.5	179.5	283.9	358.9	556.0
FF-2.3	2709	100	46	27	4	2886	0.0	0.1	0.3	1.9	3.9	6.6	27.0	104.7
Prodigy	876	68	28	13	1	986	0.7	0.7	1.2	7.0	11.3	24.7	48.4	203.8
UCPOP-4.1	730	156	109	99	15	1109	10.2	31.8	106.3	328.2	426.1	581.4	822.5	1187.3
BlkBox-4.2	1187	27	104	11	2	1331	0.0	0.1	12.0	13.9	15.8	23.8	41.9	90.1
Metric-FF-02	2598	145	53	18	2	2816	0.0	0.1	0.4	2.5	3.8	6.4	18.9	61.7
ALL	27878	2139	2176	711	80	32984	0.5	1.2	6.6	22.4	35.6	65.5	143.4	378.4

Table 5: Time-to-success (left side) across the non-dominated planners; the planners are ordered according to selection by the Greedy-Set-Cover. The right side shows the percentiles for the planners.

modeling). We plan to examine this approach for a better prediction of run time.

A simple observation leads to a remarkably successful allocation strategy that does not use either of the above analyses or models. As shown in Table 5 (right), all but one achieve the 80th percentile at 10 seconds. At 100 seconds, one planner each achieves the 89th, 94th, and 97th percentiles, five reach the 98th percentile, and three achieve the 99th percentile. At 200 seconds, six have reached the 99th percentile; each increase of 100 seconds raises the percentile for all planners run. It follows that we can use a stepped approach for allocating time and avoid any cost of calculating an optimal run time for each planner. Each planner starts at a point of higher confidence than the median, and gradually reaches higher confidence for each additional round.

The Portfolio

Our portfolio strategy uses offline analyses to model success with decision tree classifiers and to select a reasonable cut-off for run-time. In contrast to recent research that uses more sophisticated regression or Bayesian techniques, we find that this approach guides the portfolio to solve more problems than any single planner while at the same time being significantly faster than the average planner time.

As mentioned, we pruned the planners to a non-dominated subset based on successful performance so as to eliminate the portfolio wasting time on a subsumed planner. It is easy to see this is a set covering problem. We implemented Greedy-Set-Cover from (Cormen *et al.* 2003) and found that at least 14 planners could be removed.

The portfolio begins with the set of planners under consideration and ranks them according to the success models. Ideally, we would obtain from the success model an estimate of P(solution found|problem, planner). Since the decision trees do not provide this directly, we estimate it using counts from the predicted leaf node of the tree. For example, if the planner success model returns a leaf node indicating success, we take the ratio of successes to the number of instances at that leaf. In effect, we are measuring the confidence of the leaf node over its training examples. An obvious drawback to this approach is that the leaves are actually poor estimators due to pruning. Though it works well in this application, it is admittedly a rather ad hoc way to estimate the probability; we do plan to incorporate more sophisticated probability estimation techniques in future work. We rank the planners in decreasing probability of success then increasing probability of failure. This ensures that the portfolio tries the planner most likely to succeed first and the planner most likely to fail last.

As in (Howe *et al.* 1999), the portfolio allocates time in a series of round robin stages. The first stage tries the first five planners for ten seconds each; we chose the first five because it is half of the planners. The second stage starts at the top of the ranking and runs all planners up to 100 seconds. Every stage thereafter adds 100 seconds. The portfolio stops when 1) a planner succeeds, 2) no planners are alive to run, or 3) max-time is exceeded, where max-time is the same max run-time (30 minutes) of any individual planner.

Portfolio Performance

Our long-term goal in examining the portfolio performance is to understand *why* the portfolio selected one algorithm over another for a particular problem; that is, what features and performance indicators distinguish planner performance for particular problems. In this paper, however, we start by examining the raw number of problems solves by the portfolio and its robustness on unseen problems as compared to the best and average planner performance. We randomly selected 90% (3565) of the problems for training the portfolio models; the remaining 10% (394) of the problems are used for testing the portfolio. Of this 394 problems, 371 (94.2%) were solved by at least one planner. The best single planner, SGPlan-04 solved 291 (73.9%) of these problems.

To examine the impact of using the culled set, we compare the performance of the portfolio using all planners ($\mathcal{A}_{port}$) against the portfolio using the non-dominated (unique) planners ($\mathcal{U}_{port}$). In terms of robustness, $\mathcal{A}_{port}$ solved 307 (77.9%) problems while $\mathcal{U}_{port}$ solved 325 (82.5%). According to a paired sample t-test, both portfolios perform significantly faster than the average planner run time (by about 6 seconds); the average planner run-time is the mean time-to-success for all planners that solved that problem. A comparison of the RTDs for $\mathcal{U}_{port}$ and the average planner for problems that the portfolio solved is shown Figure 2. The average run time of $\mathcal{A}_{port}$ was higher by about half a second

than $\mathcal{U}_{port}$, but this difference was not significant ($p = .41$) according to a paired t-test of the 300 problems that both portfolios solved. The mean ratio of successful planners included in $\mathcal{U}_{port}$ was 0.70 (sd of 0.18). Though it is difficult to say with certainty without more testing, there is a trend for $\mathcal{U}_{port}$ to be much more robust.

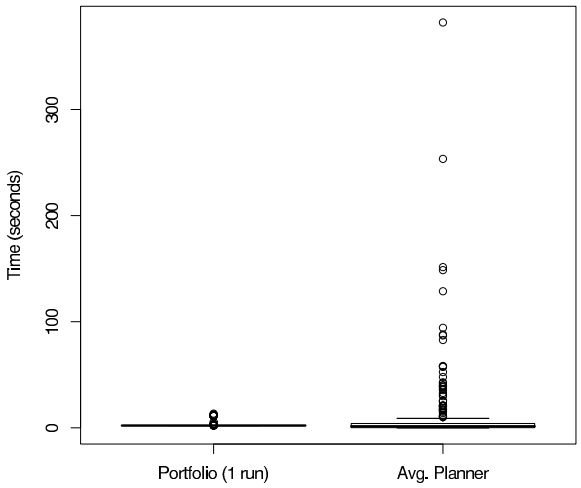

Figure 2: Comparison of the portfolio (left) to the average planner performance (right) for the 325 problems that the portfolio solved.

Closing

We posed two questions at the outset with the aim of improving portfolio robustness. We found that:

1. We can use the subset of 'fast' features only without significantly impacting classification accuracy for time and significantly, but only marginally (by 0.6% on average), impacting accuracy for success. Using 'req' features did not significantly change the prediction accuracy, though it did qualitatively change the models in a way that may prove useful for other explanations.
2. The run-time distributions show heavy-tails, as expected. Using log-sized bins provides the highest accuracy; decision trees achieve 94.3% average prediction accuracy. Instead of relying on prediction for allocation, the portfolio allocates time in a stepped round robin strategy.

Incorporating learning from prior experience into a portfolio yielded some promising results. The portfolio solved 82.5% problems solved out of a possible best of 94.2%; this was also more than the best single planner, which solved 73.9%. The time to solution was lower by 6 seconds over the average planner; this gap may widen as we incorporate larger problems into the research. We present some follow-up directions for this research.

Planner Selection: The set covering algorithm returned *one* possible minimum set of planners for the portfolio. But the algorithm disregards the quality of individual planners beyond raw counts of the number of problems solved. One might expect that the planners of a particular strategy (POCL, SAT, etc.) subsume other planners of a similar strategy or that newer planners subsume older variants, but we found counter-examples of both of these expectations. We hope to further explore which planners subsume one another to arrive at clues for explaining planner behavior.

Problem Set: Some of our results suggest a floor effect in the problem set; a single planner solved almost 80% of the problems. One way to overcome the floor effect is to generate more challenging problems; the two previous International Planning Competitions provided such tools. The planning competition at ICAPS this year will likely produce newer problems that we can use to distinguish planner performance. We also hope to extend the problem set to include the recent temporal and probabilistic extensions to PDDL.

Solution Variety: An added benefit of using a suite of algorithms is that the planners may produce different solutions. Currently, we stop processing once the first solution is achieved. A recent focus in the planning community is to produce plans that optimize metrics other than plan length. The special case of multi-objective optimization suggests a returning a set of Pareto-optimal solutions. Some of the planners in the set (such as SAPA-2) are able to return multiple solutions and can work with multi-objective metrics.

Confidence Bounds on Quitting: The staged round robin strategy incrementally increases the confidence that the planner will not solve the problem; if a planner achieves the 99th percentile, then we can be 99% sure that the planner would not have solved the problem in the time it was run. The portfolio could state the percentiles of each algorithm as an empirical confidence bound over its training examples. Alternatively this method provides a hard limit for quitting in the absence of a time limit. For example, the portfolio could be instructed to run each algorithm up to a specific percentile. In advance, the portfolio can estimate the run time for the percentile.

Predicting Runtime: As mentioned in the body, we plan to examine the use of more sophisticated learning for predicting run-time.

Ranking: We intend to explore other options for ranking. One method would be to incorporate the ranking strategy from (Howe *et al.* 1999) where the planners are ranked according to $\frac{P(success|problem,A_j)}{T(A_j|problem)}$, where $T(A_j|problem)$ is the expected time (at a specific percentile) of algorithm j given the problem. This ratio minimizes the expected cost of trying n algorithms until one works (Simon & Kadan 1975). Using this ratio also solves the issue of which value to use from time predictions in the log scale since the denominator simply moves the decimal by the size of the bin.

We have informed each stage of the portfolio strategy using *learning* prior performance of the algorithms on a large collection of problems. The trends in our results indicate that we have identified some key interactions within each stage of portfolio construction. But the larger task now lies

in *explaining* the results that we see. There are two directions along which we intend to pursue such explanations. First, we hope to examine more closely the interactions between the portfolio stages to determine which stages are critical to a successful portfolio. Understanding the critical components of a portfolio help us refine the distinctions between the algorithms. Second, we hope to link dependencies between the models learned and the systems they model to highlight specific search behaviors. One simple way to do this is to test specific hypotheses about planner behavior with respect to the learned models using specially generated problems. Ultimately, we hope to provide better explanations of the algorithm behavior with an eye toward better algorithm design.

Acknowledgments

We thank the anonymous reviewers for their comments.

References

Baptista, L., and Silva, J. P. M. 2000. Using randomization and learning to solve hard real-world instances of satisfiability. In *Principles and Practice of Constraint Programming*, 489–494.

Beck, J. C., and Freuder, E. C. 2004. Simple rules for low-knowledge algorithm selection. In *Proc. of 1st CPAIOR*.

Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. 2003. *Introduction to Algorithms*. MIT press, Cambridge, MA, second edition.

Fink, E. 1998. How to solve it automatically: Selection among problem solving methods. In *Proc. of 4th AIPS*, 128–136.

Gebruers, C.; Guerri, A.; Hnich, B.; and Milano, M. 2004. Making choices using structure at the instance level within a case based reasoning framework. In *CPAIOR*, 380–386.

Gebruers, C.; Hnich, B.; Bridge, D. G.; and Freuder, E. C. 2005. Using CBR to select solution strategies in constraint programming. In *ICCBR*, 222–236.

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins., D. 1998. PDDL : the planning domain defnition language. Technical report, Yale Center for Computational Vision and Control.

Gomes, C. P., and Selman, B. 1997. Algorithm portfolio design: Theory vs. practice. In *Proc. of 13th UAI*. Linz, Austria.: Morgan Kaufman.

Gratch, J., and Chien, S. 1996. Adaptive problem-solving for large-scale scheduling problems: A case study. *JAIR* 4:365–396.

Guerri, A., and Milano, M. 2004. Learning techniques for automatic algorithm portfolio selection. In *Proc. of 16th ECAI*, 475–479.

Guo, H., and Hsu, W. H. 2004. A learning-based algorithm selection meta-reasoner for the real-time mpe problem. In *Australian Conference on Artificial Intelligence*, 307–318.

Hoffmann, J. 2001. Local search topology in planning benchmarks: An empirical analysis. In *Proc. of 17th IJCAI*, 453–458.

Hoffmann, J. 2004. *Utilizing Problem Structure in Planning: A local Search Approach*. Berlin, New York: Springer-Verlag.

Horvitz, E.; Ruan, Y.; Gomes, C. P.; Kautz, H.; Selman, B.; and Chickering, D. M. 2001. A bayesian approach to tackling hard computational problems. In *Proc. of 17th UAI*, 235–244.

Howe, A. E.; Dahlman, E.; Hansen, C.; von Mayrhauser, A.; and Scheetz, M. 1999. Exploiting competitive planner performance. In *Proc. of 5th ECP*.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An economics approach to hard combinatorial problems. *Science* 275:51–54.

Lagoudakis, M. G., and Littman, M. L. 2000. Algorithm selection using reinforcement learning. In *Proc. 17th ICML*, 511–518.

Lagoudakis, M. G., and Littman, M. L. 2001. Learning to select branching rules in the dpll procedure for satisfiability. In *LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*. Boston, MA: Electronic Notes in Discrete Mathematics (ENDM), Vol. 9,.

Leyton-Brown, K.; Nudelman, E.; Andrew, G.; McFadden, J.; and Shoham, Y. 2003a. Boosting as a metaphore for algorithm design. In *Proc. of CP*.

Leyton-Brown, K.; Nudelman, E.; Andrew, G.; McFadden, J.; and Shoham, Y. 2003b. A portfolio approach to algorithm selection. In *Proc. of 18th IJCAI*.

Leyton-Brown, K.; Nudelman, E.; and Shoham, Y. 2002. Learning the empirical hardness of optimization problems: the case of combinatorial auctions. In *Principles and Practices of Constraint Programming (CP-2002)*.

Minton, S. 1996. Automatically configuring constraint satisfaction programs: A case study. *Constraints* 1(1/2):7–43.

Nudelman, E.; Leyton-Brown, K.; Hoos, H.; Devkar, A.; and Shoham, Y. 2004. Understanding random sat: Beyond the clauses-to-variables ratio. In *Priciples and Practices of Constraint Programing (CP-2004)*.

Rice, J. R. 1976. The algorithm selection problem. *Advances in Computers* 15:65–118.

Simon, H., and Kadan, J. 1975. Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence* 6:236–247.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical machine learning tools and techniques*. Number ISBN 0-12-088407-0. San Francisco: Morgan Kaufmann, 2nd edition.

Wolpert, D. H., and Macready, W. G. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1(1):67–82.