Exploiting Portfolio Strategy to Explore the Interaction of Problems and Algorithms in AI Planning *

Mark Roberts
Computer Science Department
Colorado State University
Fort Collins, Colorado 80523
mroberts@cs.colostate.edu

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 ((Wolpert & Macready 1997) consider the discrete case). Portfolios are a general way to overcome algorithm bias and maintain robustness across a range of problems. A portfolio controls the run time of a suite of algorithms with a strategy consisting of: *selecting* which algorithms to run; *ranking* the selected algorithms; and *allocating* computational time to them. An ideal strategy maximizes success and minimizes total computation. This means the strategy must be accurate and fast.

Previous portfolios used a variety of strategies. One of the first formulations calculated the risk of selecting an algorithm (Huberman, Lukose, & Hogg 1997). Gomes and Selman (1997) explicitly computed the value of multiple restarts of the same algorithm. The Bus meta-planner (Howe *et al.* 1999) used a round robin allocation based on a trade-off between a simple model of success and expected cost. Other portfolio approaches (e.g., (Gratch & Chien 1996; Minton 1996; Fink 1998; Baptista & Silva 2000; Leyton-Brown *et al.* 2003; Lagoudakis & Littman 2000)) used easily extracted features of the problems and solution progress and rely, for their portfolio strategy, on models generated off-line from problem instances. Most approaches also leverage statistical run-time distribution information (e.g., (Horvitz *et al.* 2001; Beck & Freuder 2004)) while some, such as (Guerri & Milano 2004), focus on extracting features that uncover *structure* of the problem instance.

The goodness or utility of a portfolio strategy can be measured in different ways. For example, we can examine the raw number of problems it solves and examine its robustness on unseen problems. We can compare portfolio performance against the individual or aggregate performance of the suite of algorithms. We can examine the selection, ranking, and allocation strength of the portfolio strategy against the best it could have done. Finally, *we can study the portfolio strategy for clues about* why *one algorithm is favored over another and what led the portfolio to make the distinction.* Ultimately, it is this last measure that will foster deeper insight and explanation.

*This research was sponsored by the National Science Foundation under grant number IIS-0138690.

My research analyzes portfolio performance on planning problems from the International Planning Competitions to uncover algorithmic and problem structure dependencies through:

1. Modeling planner performance of 23 planners across almost 4000 benchmark problems.
2. Constructing portfolio strategies using a principled methodology based on analysis of and learning from previous off-line performance.
3. Measuring portfolio strategies to test specific hypotheses about what constitutes effective selection, ranking, and allocation.
4. Examining planner performance, models of the performance, features used to build those models, domain information, and dependencies between these sets to develop and test specific hypotheses leading to stronger explanations of search performance in planning.

What follows are highlights of some of my findings with regard to questions (1) and (2) that form the core of a recently submitted AAAI workshop paper. My dissertation research will address all four of these questions.

Modeling Planner Performance

We study 23 classical planners, and for each planner, we construct two models: *success* and *time*. Success estimates $P(\text{solution found}|\text{problem, planner})$. Time predicts computation time needed for a given planner to complete a given problem.

Each problem instance is defined by 57 features that can be automatically extracted from problem and domain definitions. The set starts with features from (Howe *et al.* 1999) and (Hoffmann 2001) and adds others. 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). Based on the amount of computation time to compute features, we designate the domain and instance-specific as 'fast' features and the action interaction and topological features as 'expensive' features.

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,

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

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. Each planner is allowed 30 minutes and 768 Meg. We used 22 identically configured Pentium 4 3.4Ghz computers.

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, while J48 is a simple decision tree based on Quinlan's C4.5. By default, we use 10-fold cross validation.

We distinguish our models based on the data we use to build them: *all* data and *old* data (all but IPC4). The *time3* model divides 30 minutes of possible time into 10 equal sized (three minute) bins. The *logTime* model uses 5 bins of time based on the division (in seconds): $\{1, 10, 100, 1000, 10000\}$.

Predicting Success The distribution of results in the raw data values tends to be skewed. 35.4% of the runs succeed over all planners and problems. 97.1% (successful) and 96.4% (failure) of runs complete in under 3 minutes. The distributions do vary across planners from a range of 5%-70% for success and 65.5%-100% for % successful runs under 3 minutes and 72.6%-100% for % failed runs under 3 minutes. We found that J48 predicts success for *old* with 96.7% average accuracy (sd of 3.2) and for *all* with 96.8% average accuracy (sd of 2.12).

Predicting Time The run-time distributions (RTDs) are heavily skewed for each planner. Over all planners, 77.8% of the runs finish in less than one second and 6.9% finished in greater than 1000 seconds[2]. Given these heavy tails, we found that *logTime* predictions are much more accurate than *time3* predictions. The average prediction accuracy using J48 with *binned data* was 93.52% (sd of 6.55) for *logTime*.

Which features are informative? To examine if specific features are informative we constructed OneR models, which rely on a *single* feature for classification. For success on the old problems, the `average number of negations in effects` was the best predictor for nine of the planners; the `predicate arity` was best for another four. The first feature may indicate where the often used h^+ heuristic may have trouble; the second roughly influences branching in the search space.

When we examined the features used for the *old* data, we found that fast features had been selected for predicting success for 16 of the 23 planners and for predicting time for 5 of the 12 planners. All of the expensive features selected were from Hoffmann's set. The mean accuracy for the success models was 91.8% using fast features and 93.6% using expensive features. The mean accuracy for time models was 96.2% using fast features and 78.2% using expensive features; however, the skews on the distribution for models using the expensive features tended to be more extreme.

[2]We did not construct time models for nine planners that either always finished in less than 1 second or that too rarely succeeded in less than 30 minutes.

Do we need all of the features? We noted a considerable (100 to 1000-fold) difference in computing the feature cost for the fast features and the expensive features. We wondered if we could use only the fast features without sacrificing accuracy. On a subset of the data for which we had all feature information, a paired sample T-test of using or removing the expensive features was not significant ($p < 0.61$ for success and $p < 0.49$ for *time3*); it appears that expensive features are not necessary for accurate models.

Do models for older problems generalize? To test this question, we trained models using *old* and tested with cross-validation as well as with IPC4. A paired sample T-test comparing accuracy of each of the planner models when tested on *old* versus IPC4 is highly significant ($P < .0001$); the models are statistically significantly more accurate for the old problems. This suggests that the IPC4 problems must be classified differently than the older problems.

Portfolio Construction

We construct our portfolio in a principled manner from the model analyses. To date, we have tried several variants that are increasingly informed by deeper insight.

Pruning with static allocation We developed and tested a simple model of portfolio allocation by applying the RTDs. We noted that the highest median planner time was 5 seconds. Our first strategy selected only those planners to those that were predicted to succeed, ranked that pruned set by probability of success, and allocated ten seconds (5 seconds rounded to the next largest bin) to each planner. We found that this strategy was not very robust, though it did significantly outperform the average planner performance. In hindsight, a simple explanation is that the planners were stopped too soon; we effectively stopped some planners somewhere between their 50th and 80th percentiles for time-to-success.

Using unique planners We first examined how we might reduce the number of algorithms that the portfolio uses. Let $\mathcal{A}$ be the entire set of algorithms under consideration. In general, we seek to find the minimum covering of unique planners, $\mathcal{U} \subset \mathcal{A}$, such that all problems solved by $\mathcal{A}$ remain solved by $\mathcal{U}$. A minimum covering is equivalent to a set covering, which is $\mathcal{NP}$-Complete, but has a known polynomial algorithm, Greedy-Set-Cover, that has an approximation ratio $\rho(n) = \ln|\mathcal{A}|+1$ (Cormen *et al.* 2003). We implemented Greedy-Set-Cover and found that at least 14 planners could be removed.

Quiting with confidence A closer look at the RTDs showed us that we could set a reasonable pausing time for the planners on each iteration of a round robin portfolio. In this way, we start with a higher confidence than the median, and we gradually increase the confidence in each pass of the algorithm. For the unique planners, all but one achieve the 80th percentile at 10 seconds. At 100 seconds, one planner achieves each of 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. This observation suggests a stepped approach for allocating time.

The current portfolio begins with the set of planners under consideration (either $\mathcal{A}$ or $\mathcal{U}$); it does not itself perform any

selection. For these planners, the portfolio ranks the planners according to the learned models in decreasing probability of success then in 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.

The portfolio allocates time to the planners 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 $|\mathcal{U}|$. 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.

We trained the models for this portfolio using a random sampling of 90% of the problems and tested it on the remaining 10% (394). 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 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%). Both portfolios significantly perform faster than the average planner run time (by about 6 seconds) according to a paired sample T-test.

Future Work

Incorporating learning from prior experience into a portfolio yielded some promising results. But there still remains much work toward the primary goal of *linking* the search bias of various planners with their performance on specific problems. In this section, we present several key points of continuing work that we hope will reveal deeper insight into understanding specific planning approaches.

Features To date, we have examined three basic feature types: domain/problem instance, action interaction, and topological. The OneR and J48 models provide some evidence linking particular features to performance prediction. Already, we have begun to examine other features for impact on performance prediction. We expect to identify new features that help explain less direct action interactions as we perform richer domain analysis. Further, we hope to examine the feature set for overlap and uniqueness similar to what we have done for planners.

Planners We began our research with the widest set of planners possible. We plan to examine the data for interactions based on planner type (such as SAT-based, POCL-base, Graphplan, Relaxed Graphplan, or Hybrid). We use the default settings of all planners; our work can easily be extended to include some limited parameter tuning of the planners by viewing each change in parameters as a new planner. Most of the planners we used came from the IPCs. This coming IPC5 will provide another set of planners to extend our work. Attending ICAPS during an IPC year will also provide direct interaction (and possible collaboration with) the authors of these planners to aid in gaining deeper insight.

Problems This work is based in the classical planning paradigm. Recent extensions to PDDL were intended to push the community toward more realistic problems (Hoffmann & Edelkamp 2005); our work indicates that there is a significant difference between the old and new problems. My hope is to fully develop a principled methodology for portfolio construction and then extend it to the newer Temporal and Probabilistic tracks of the IPC. Another potential area for understanding the planner behavior is through the problem generators from the IPCs - these could be used to test specific hypotheses about the dependencies we notice.

Classifiers We mined our data with two simple classifiers to predict success and run time. We conjecture that there is a wealth of information beyond this utilitarian approach, so we hope to extend these analyses by incorporating more sophisticated classification techniques (such as Bayesian or k-nearest neighbor clustering) to mine relationships between the features, the problems, and the algorithms.

Unified Model Work by (Kambhampati, Knoblock, & Yang 1995) and (Kautz & Selman 1999) unifying planning approaches under the same framework provides a foundation for exploring deeper analysis of planners.

On-line learning Our current system uses prior, off-line learning to inform the portfolio strategy. It seems reasonable to consider that we could extend the models into an on-line learning paradigm.

Computational Trade-offs Our preliminary research indicates we may be able to use only fast features to predict success and time. But we need to examine this hypothesis in light of newer domains and problems (especially since the models built on old models didn't generalize). If we do find that expensive features are necessary, we expect to examine in more detail the question of determining the appropriate level of computational effort for a strategy. Classifiers that incorporate feature cost into classification would be helpful determining an appropriate trade-off point.

The focus of incorporating these extensions is to identify dependencies between the domains, heuristics, algorithms, and run time dynamics in classical planning. The main directions of this research are 1) in providing a principled methodology for portfolio construction and 2) in providing a framework for exploiting the portfolio strategy to refine our understanding of planning approaches.

Acknowledgments

I want to thank Adele Howe, my dissertation adviser, and the reviewer of this paper who both provided feedback and ideas for further directions for this work. Finally, a thanks to all the authors who make their planners available for study.

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.

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.

Hoffmann, J., and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *JAIR* 24:519–579.

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.

Kambhampati, S.; Knoblock, C. A.; and Yang, Q. 1995. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence* 76(1-2):167–238.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proc. IJCAI-99*.

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

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

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

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 Trans./ on Evolutionary Comp.* 1(1):67–82.