# Harnessing Algorithm Bias in Classical Planning

**Mark Roberts**
mroberts@cs.colostate.edu
http://www.cs.colostate.edu/~mroberts
Computer Science Department, Colorado State University
Fort Collins, CO, 80521

A planning system's performance is biased due to many factors related to its design. For example, the representation, decision points, search control, memory usage, heuristic guidance, and stopping criteria all can have implications for performance. Problem instance characteristics also impact system performance. The interaction of the design choices with the problem instance makes it difficult to select the most efficient system from the array of choices. It seems natural to apply learning to aid in allocating computational resources among a *portfolio* of planners that may have complementing (or competing) search technologies. Such selection is called the *portfolio strategy*.

My thesis is that *we can study a portfolio of planning systems for clues about* ***why*** *one algorithm is favored over another*. A secondary thesis is that *we can uncover algorithmic and problem structure dependencies by examining algorithm performance on specific instances*. This research focuses on a series of questions, roughly addressed in the order:

1. Can we model planner performance on benchmark problems with simple features?
2. Is a portfolio based on analysis of and learning from previous performance competitive with existing planners?
3. Which portfolio strategies constitute effective selection, ranking, and allocation?
4. What can we learn from the data and/or models that will aid us in developing and testing specific hypotheses leading to stronger explanations of search performance in planning?
5. Can we construct a meta-planner based on our findings?
6. How can the knowledge we have gained supplement the current benchmark problems?
7. Do our findings hold under relaxed classic assumptions?

What follows are highlighted findings for items (1) and (2) as well as the current state of items (3) and (4); my dissertation will provide detailed analyses for all of these items.

This paper presents a several extensions to the results presented in the 2006 Doctoral Consortium. We added 9 more planners (but two were not reliable) as well as 767 more challenging problems from the IPC5 Propositional and IPC4 hard-typed domains. We also added 30 more advanced learning models to our original two[1]. Finally, we modified the portfolio to use a variety of ranking and allocation strategies (including random baselines) and the ability to run algorithms either serially (in order by ranking) or in a round-robin fashion until success or time runs out. In terms of examining the data, we have included recent work that takes steps toward analyzing question 4 above.

[1]Another graduate student, Landon Flom, did this work

## Collecting Performance Data

We have gone through several major revisions of data collection. Our current configuration consists of:

**4726 STRIPS PDDL Problems** from 385 domains that are taken from Hoffmann's dataset (2004), the UCPOP Strict benchmark, IPC sets (IPC1, IPC2, IPC3 Easy Typed, IPC4, IPC5) plus 37 other publicly available problems from two domains (Sodor and Stek).

**27 Planners** from our complete list of 86 planner instances that are designed to sample the variety of historical approaches. Each planner is allowed 30 minutes and 768 MB memory. We used 30 identically configured machines.

**38 Features** automatically extracted from problem and domain definitions; we included features from (Howe *et al.* 1999) plus many others. We divide the features into three categories of increasing knowledge and computational cost: domain specific, instance specific, action interaction. In early work, we examined 20 features from Hoffmann's (2004) state space taxonomy[2]. But these features are very costly and are excluded from modeling. We label the domain and instance-specific features as 'fast' and the action interaction and topological features as 'expensive.'

## Performance Models

For each planner, we constructed two models: *success* and *runtime*. Success models output a binary decision and may also estimate the probability of finding a solution given a problem instance and a planner ($P(\text{solution found}|\text{problem, planner})$). Runtime models predict computation time needed for a given planner to complete a given problem instance.

We build all models with the WEKA data mining package (Witten & Frank 2005); to start, we used two models that worked well that we could explain: OneR and J48. OneR selects the single feature that yields the highest prediction value on the training set, while J48 is a decision tree method

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

based on Quinlan's C4.5. The models are distinguished based on the training data used to build them: *preIPC4* data (all but IPC4 and IPC5), *preIPC5* (all but IPC5), and *challenge3* (problems for which one to three planners succeeded and the median completion time was over one second). Unless mentioned otherwise, results are reported for ten-fold cross-validation. For these two models we found that:

- When predicting success, J48 achieved 96.7% average accuracy (sd of 3.2) for *preIPC4* and 96.8% average accuracy (sd of 2.12) for *preIPC5*.
- When predicting time, the average accuracy using J48 on *log binned data* was 95.0% (sd of 2.49). Over all planners, 84.2% of the runs finish in less than one second and 5% finished in greater than 1000 seconds.
- Expensive features do improve accuracy, but not enough to justify their computational cost.
- In the success OneR models, 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.

Our most recent work extends the model set to 30 more techniques. Some preliminary findings with respect to these models are:

- The expanded model set improves accuracy over the initial models; especially for runtime. The most common model was KStar – a variant of K-Nearest Neighbors.
- Models trained on *preIPC4* and tested on IPC4 did not generalize well, but models trained on *preIPC5* and tested on IPC5 generalized well.

## The Portfolio Architecture

The portfolio should contain the most accurate models (that are not overfit). We split *challenge3* into an 80% training and 20% testing sets, where the testing set included half the problems from IPC4 and IPC5 (to further focus the learning on the most challenging problems). We evaluated all models on the testing data and selected the most accurate success and runtime models.

The other three decision making components (selection, ranking, and allocation) can be configured into many different combinations. The components and their options are:

**Selection** restricts the set of possible planners to only those needed to cover the problem set. We use a single strategy that computes a "cover" from the 28 planners using a greedy set covering approximation algorithm (Cormen *et al.* 2003) on the 80% *challenge3* data. We excluded from the final set any planner that only solved one unique problem in the training set (4 planners total). The reduced "cover" consists of 10 planners.

**Ranking** orders the execution of the planners. The ranking strategies we have examined include:

*random* ranks the planners in an arbitrary order,
*cover* uses the set covering order,
*pSuccess* prunes those planners predicted to fail and orders the rest by decreasing predicted probability of success,
*predTime* orders by increasing predicted time, and
*SimonKadane* orders in decreasing $\frac{pSuccess}{predTime}$, which minimizes expected cost of success in serial search (1975) .

**Allocation** determines the runtime for each planner within one of two execution paradigms:

**Serial execution** runs each planner to its allocated time and quits at the first success or after all planners have spent their time. For serial execution we applied two allocation strategies:

*avgPlanner* uses the average time to succeed for the planner, and
*predTime* computes the predicted time for the problem from the model.

**Round-robin execution** cycles through the queue of planners until a single planner indicates success, all planners stop with failure, or the portfolio exceeds the experimental time limit (30 minutes). On each iteration, the planner begins where previously left off; it does not restart from scratch. We support one allocation strategy:

*confInt* uses the runtime distribution of successful training runs to estimate the quantiles $q = \{25, 50, 75, 80, 85, 90, 95, 97, 99\}$. For example, if the actual successful runtimes of a planner are $\{0.1, 0.2, 0.3, 0.4, 0.5, 10, 100, 1000\}$, then *confInt* will return an allocation strategy of $\{0.28, 0.45, 32.5, 64.0, 95.5, 370.0, 685.0, 1000.0\}$.

We also implemented a *random* allocation scheme for both execution models as a baseline.

## How does the portfolio compare to existing planners?

We have compared this simple portfolio to the most successful and average planners and found that:

- a simple allocation strategy (paired with *pSuccess*) that uses the run-time distributions can produce performance better than the average planner performance,
- the best ranking strategies employ the extended models,
- we need even more accurate time models; under any ranking strategy random allocation performed as well as any other allocation strategy,
- performance trends suggest that round-robin allocation is more successful than serial allocation,
- the best portfolio lags 60 seconds on average behind an 'oracle' selection strategy that uses perfect knowledge,
- the best portfolio is 10 seconds faster on average than the best planner (SGPlan-2006), and
- out of 244 testing problems from *challenge3*, the best portfolio solves 51 more problems than the best planner.

## Learning from the Data: Hoffmann's Taxonomy

We applied Hoffmann's topology taxonomy to see if it helps explain the performance of two groups of planners: Heuristic Search (HS) planners and Non-heuristic search (NHS) planners. Using statistical analyses on the *challenge3* data we have found that:

- The performance of HS and NHS planners is distinct from one another regardless of the taxonomy,
- The performance of NHS planners appears to be insensitive to the taxonomy,

- The performance of HS planners is sensitive to the taxonomy,
- More work needs to be done to account for problem difficulty in the existing taxonomy.

The results, though limited, suggest that it is useful to mine the data to uncover specific performance dependencies and that this analysis can extend existing knowledge.

## Summary of status

With respect to the questions posed in the introduction, we have made the following contributions:

**1. Modeling Performance** We can model planner performance on benchmark problems with simple features extracted from problems and planners. Feature cost and feature selection remain important to further enhancing the portfolio models. Given the relatively poor performance of the time models, we may need to look into meta-learning techniques in machine learning for better models.

**2. Basic Portfolio** Our proof-of-concept portfolio is competitive with existing planners as of IPC4 but including problems up to the most recent IPC5.

**3. Portfolio Strategies** We have begun exploring the space of portfolio strategies and found that round-robin strategies that use Simon/Kadane ranking appear to be the most successful. However, more work needs to be done on modeling runtime so that the portfolio can better allocate time. There are also many more strategies from the literature that we could employ in our study.

**4. Linking Performance** We have shown that both the models and the raw performance data are rich with information that can lead to explanations of search performance in classical planning. The models provide some evidence linking particular features to performance prediction. But there remains much work to *link* the search bias of various planners with their performance on specific problems. We expect to identify new features that help explain indirect action interactions as we perform richer domain analysis. A reasonable place to start may be including some features from HAP (Vrakas *et al.* 2005), TIM (Fox & Long 1998), and Londex (Chen, Zhao, & Zhang 2007). A more sophisticated approach would be to use the analysis structures from (Helmert 2006) in generating features and identifying problem specific bias[3].

**5. The meta-planner** As it is, the portfolio is unfit for inclusion as a competition planner. But we expect to make key insights into which components of planners are critical for successful performance and implement a planner that combines these components. We also hope to extend our analysis to modeling dynamic features wherein we may include sampled state-space features that may change planner behavior on-line.

**6. Extending the Benchmark Problems** Generating specific problems with IPC problem generators will be useful for testing specific hypotheses about the dependencies we notice. We have converted (but not yet thoroughly studied) two new problem sets[4], which we hope will extend the benchmarks to other realistic domains. A key issue is to expand the problem set in ways that are both challenging and realistic (Watson *et al.* 1999).

**7. Relaxing Assumptions** Finally, this work is based in the classical planning paradigm and numerous extensions to PDDL relax classical assumptions. We hope to fully develop a principled methodology for portfolio construction and then extend it to these relaxed assumptions.

**Acknowledgments** The author would like to thank his advisor Adele Howe, the students in the AI group at CSU, the ICAPS community, as well as all the authors of the planning systems.

Portions of this work appear in other venues, and the reader is referred to these or the authors website for more detail: (Roberts & Howe 2006), (Roberts, Howe, & Flom 2007), (Roberts & Howe 2007).

[3]A thanks to David Smith for this suggestion.

[4]Christina Williams added these problems.

## References

Chen, Y.; Zhao, X.; and Zhang, W. 2007. Long distance mutual exclusion for propositional planning. In *International Joint Conference on Artificial Intelligence (IJCAI-07).*

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

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *JAIR* Volume 9:367–421.

Helmert, M. 2006. New complexity results for classical planning benchmarks. In *ICAPS 2006*, 52–61.

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

Howe, A.; Dahlman, E.; Hansen, C.; vonMayrhauser, A.; and Scheetz, M. 1999. Exploiting competitive planner performance. In *Proc. of ECP-99*.

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

Roberts, M., and Howe, A. 2007. Local search topology: Implications for planner performance. In *ICAPS 2007, Workshop on Heuristics for Domain-independent Planning*, to appear.

Roberts, M.; Howe, A.; and Flom, L. 2007. Learned models of performance for many planners. In *ICAPS 2007, Workshop AI Planning and Learning*, to appear.

Simon, H., and Kadane, J. 1975. Optimal problem-solving search: All-or-none solutions. *Artificial Intelligence* 6:235–247.

Vrakas, D.; Tsoumakas, G.; Bassiliades, N.; and Vlahavas, I. 2005. *Intelligent Techniques for Planning*. Idea Group. chapter Machine Learning for Adaptive Planning, 90–120.

Watson, J.; Barbulescu, L.; Howe, A.; and Whitley, L. 1999. Algorithm performance and problem structure for flow-shop scheduling. In *Proc. of AAAI-99*.

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