A Portfolio Approach to Algorithm Selection

Kevin Leyton-Brown and **Eugene Nudelman** and **Galen Andrew** and **Jim McFadden** and **Yoav Shoham**
{kevinlb;eugnud;galand;jmcf;shoham}@cs.stanford.edu, Stanford University, Stanford CA 94305

1 Introduction

Although some algorithms are better than others on average, there is rarely a best algorithm for a given problem. Instead, it is often the case that different algorithms perform well on different problem instances. Not surprisingly, this phenomenon is most pronounced among algorithms for solving $\mathcal{NP}$-Hard problems, because runtimes for these algorithms are often highly variable from instance to instance. When algorithms exhibit high runtime variance, one is faced with the problem of deciding which algorithm to use; in 1976 Rice dubbed this the "algorithm selection problem" [8]. In the nearly three decades that have followed, the issue of algorithm selection has failed to receive widespread attention, though of course some excellent work does exist. By far, the most common approach to algorithm selection has been to measure different algorithms' performance on a given problem distribution, and then to use only the algorithm having the lowest average runtime. This "winner-take-all" approach has driven recent advances in algorithm design and refinement, but has resulted in the neglect of many algorithms that, while uncompetitive on average, may offer excellent performance on particular problem instances. Our consideration of the algorithm selection literature, and our dissatisfaction with the winner-take-all approach, has led us to ask the following two questions. First, what general techniques can we use to perform per-instance (rather than per-distribution) algorithm selection? Second, once we have rejected the notion of winner-take-all algorithm evaluation, how ought novel algorithms to be evaluated? We offer the following answers:

1. Algorithms with high average running times can be combined to form an algorithm portfolio with low average running time when the algorithms' easy inputs are sufficiently uncorrelated.
2. New algorithm design should focus on problems on which the current algorithm portfolio performs poorly.

Readers familiar with the boosting paradigm in machine learning [9] will recognize that boosting uses similar ideas: combining weak classifiers into a much stronger ensemble by iteratively training new classifiers on data on which the ensemble performs poorly.

2 Algorithm Portfolios

In our previous work [6] we demonstrated that statistical regression can be used to learn surprisingly accurate algorithm-specific models of the empirical hardness of given distributions of problem instances. In short, the method proposed in that work is:

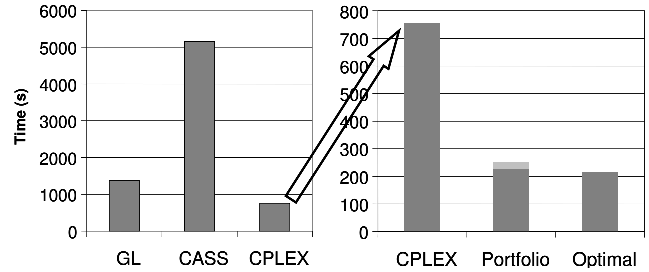

Figure 1: Algorithm and Portfolio Runtimes

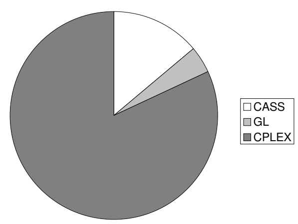

Figure 2: Optimal

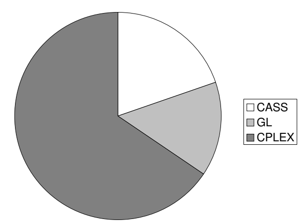

Figure 3: Selected

1. Use domain knowledge to select features of problem instances that might be indicative of runtime.
2. Generate a set of problem instances from the given distribution, and collect runtime data for the algorithm on each instance.
3. Use regression to learn a real-valued function of the features that predicts runtime.

Given this existing technique for predicting runtime, we now propose building portfolios of multiple algorithms as follows:

1. Train a model for each algorithm, as described above.
2. Given an instance:
   (a) Compute feature values
   (b) Predict each algorithm's running time
   (c) Run the algorithm predicted to be fastest

2.1 WDP Case Study: Past Work

Our case study is based on the data collected in our previous work [6]. In that work we have constructed models for predicting the running time of the CPLEX solver on the winner determination problem (WDP), which is an $\mathcal{NP}$-Complete combinatorial optimization problem formally equivalent to weighted set-packing. We have also created models for two other WDP algorithms: GL (Gonen-Lehmann) [3], a simple branch-and-bound algorithm with CPLEX's LP solver as its heuristic, and CASS [1], a more complex branch-and-bound algorithm with a non-LP heuristic. The data set consists of 4500 instances of a fixed size (256 goods and 1000 non-dominated bids), sampled uniformly from CATS [7] instance generator. Since our methodology relies on machine learning, we split the data into training, validation, and test sets. We report our portfolio runtimes only on the test set that was never used to train or evaluate models. Runtime data was collected on 550 MHz Pentium Xeons, running Linux 2.2.

2.2 WDP Case Study: Portfolios

We now describe new results. Fig. 1 compares the average runtimes of our three algorithms to that of the portfolio. Note that CPLEX would be chosen under winner-take-all algorithm selection. The "optimal" bar shows the performance of an ideal portfolio where algorithm selection is performed perfectly and with no overhead. The portfolio bar shows the time taken to compute features (light portion) and the time taken to run the selected algorithm (dark portion). Despite the fact that CASS and GL are much slower than CPLEX on average, the portfolio outperforms CPLEX by more than a factor of 3. Further, neglecting the cost of computing features, our portfolio's selections take only 5% longer to run than the optimal selections. Figs. 2 and 3 show the frequency with which each algorithm is selected in the ideal portfolio and in our portfolio. They illustrate the quality of our algorithm selection and the relative value of the three algorithms. While our portfolio does not always make the right choice, most of the mistakes occur when algorithms have very similar running times. These mistakes are not very costly, which explains why our portfolio's choices have a running time so close to optimal. These results show that our portfolio methodology can work very well even with a small number of algorithms, and even when one algorithm has superior average performance.

3 Focused Algorithm Design

Once it has been decided to select algorithms from a portfolio on a per-instance basis, it is necessary to reexamine the way we design and evaluate algorithms. Since the purpose of designing new algorithms is to reduce the time that it will take to solve problems, designers should aim to produce new algorithms that complement an existing portfolio given a distribution D reflecting problems that will be encountered in practice. The instances with the greatest potential for improvement will be hard for the portfolio, common in D, or both. The full version of this paper describes a technique for using rejection sampling to automatically generate such instances. In Figures 4 and 5 we show how our techniques are able to automatically skew two of the easiest CATS instance distributions towards harder regions. In fact, for these two distributions we generated instances that were (respectively) 100 and 50 times harder than anything we had previously seen! Moreover, the *average* runtime for the new distributions was greater than the observed *maximum* running time on the original distribution.

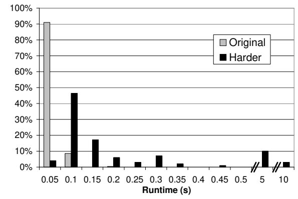

Figure 4: Matching

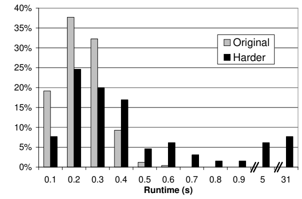

Figure 5: Scheduling

4 Conclusions

Learned runtime models may be used to create algorithm portfolios that outperform their constituent algorithms. These models can also be used to induce harder benchmark distributions for use in the development and evaluation of new algorithms. Our case study on combinatorial auctions demonstrates that a portfolio composed of CPLEX and two older—and generally *much* slower—algorithms outperforms CPLEX alone by about a factor of 3. In the full version of this paper we describe our methodology in more detail, and also:

- Show how to reduce the time spent computing features without substantially degrading portfolio performance;
- Demonstrate ways of using response variable transformations to focus portfolios on metrics other than average running time;
- Explain how to induce distributions with characteristics other than hardness (e.g. realism);
- Compare our approach to previous work that executes algorithms in parallel [2]; uses classification instead of regression [4]; or considers the problem as an MDP [5].

References

[1] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI*, 1999.

[2] C. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[3] R. Gonen and D. Lehmann. Linear programming helps solving large multi-unit combinatorial auctions. Technical Report TR-2001-8, Leibniz Center for Research in Computer Science, April 2001.

[4] E. Horvitz, Y. Ruan, C. Gomes, H. Kautz, B. Selman, and M. Chickering. A Bayesian approach to tackling hard computational problems. In *UAI*, 2001.

[5] M. Lagoudakis and M. Littman. Algorithm selection using reinforcement learning. In *ICML*, 2000.

[6] K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *CP*, 2002.

[7] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM EC*, 2000.

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

[9] R. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.