Boosting as a Metaphor For Algorithm Design

Kevin Leyton-Brown Eugene Nudelman James McFadden Galen Andrew Yoav Shoham

Department of Computer Science, Stanford University, USA

The Combinatorial Auction Winner Determination Problem

- Find revenue-maximizing non-conflicting allocation of submitted bids

$$\text{maximize:} \quad \sum_{i=1}^{m} x_i p_i$$

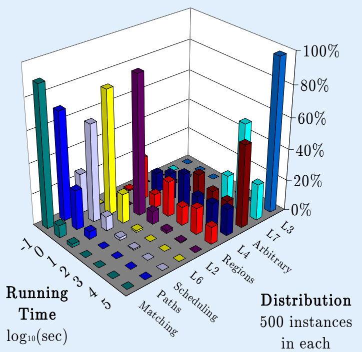
$$\text{subject to:} \quad \sum_{i|g \in S_i} x_i \le 1 \qquad \forall g$$

$$x_i \in \{0,1\} \qquad \forall i$$

- Complete heuristic search algorithms we used:
  - CPLEX [ILOG Inc.]
  - CASS [Leyton-Brown et.al]
  - GL [Gonen and Lehman]

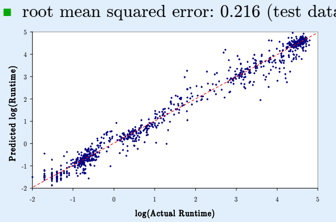
Data

- We generated instances from:
  - Weighted Random (L2), Uniform (L3), Decay (L4) [Sandholm]
  - Exponential (L6), Binomial (L7) [Fujishima]
  - CATS: Regions, Arbitrary, Matching, Scheduling [Leyton-Brown et al.]
- Randomly sampled generator's parameters for each instance
- Took more than **3 years of CPU time** just to collect CPLEX runtimes



Empirical Hardness Models

- In past work, we found that quadratic regression can yield very accurate models
  - predicting $\log_{10}$ of CPLEX runtime
  - root mean squared error: 0.216 (test data)

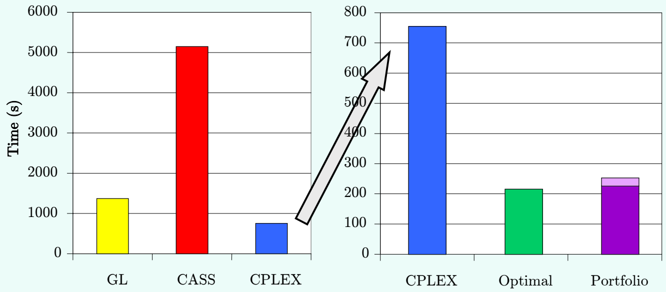


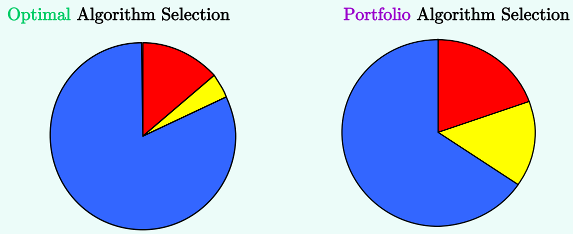
Boosting

1. Combine uncorrelated weak classifiers into a stronger aggregate
2. Train new classifiers on instances that are hard for the aggregate

1. Algorithm Portfolios

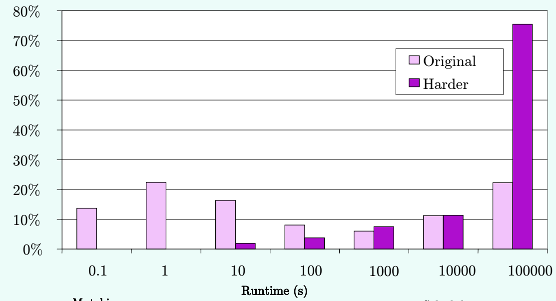
- Hardness models can be used to **select an algorithm** to run on a per-instance basis

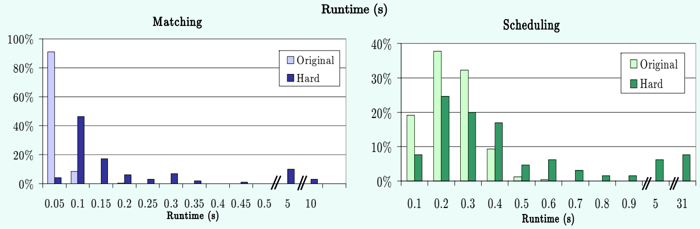




2. Distribution Induction

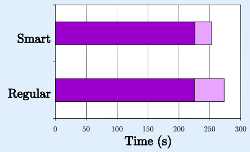
- To evaluate new algorithms, use portfolio hardness model as a PDF and generate problems **in proportion to the time our portfolio spends** on them
  - D: original distribution of instances; H_f: model of portfolio runtime (h_f normalized)
  - Generate instances from $D \times h_f$ using rejection sampling



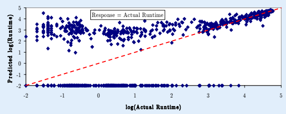


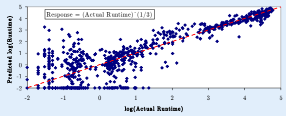
Extensions

1. Trade off time taken to compute features and time taken to run the selected algorithm



2. Transform response variable to achieve tradeoffs between absolute and relative prediction error





3. Cap runtimes to significantly reduce the amount of time required for collecting data

Discussion: Other Approaches

- Algorithm selection has received some previous study; e.g., [Rice], [Lobjois & Lemaitre]
- Classification [Horvitz et al.]
  - error measure often inappropriate
  - class boundary effects
- Run n algorithms in parallel [Gomes & Selman]
  - running time always $n \cdot$ min-time
  - in our case study, we did much better ($1.05 \cdot$ min-time)
- Sequential algorithm selection using MDP formalism [Lagoudakis & Littman]
  - algorithms must be reimplemented
  - computing a good value function at every recursive branch can be very expensive (our value function averaged 27 secs)

Future Directions

- Apply these ideas to other $\mathcal{NP}$-hard problems such as SAT
  - our preliminary SAT portfolio ("SATzilla") showed very encouraging results at the SAT-2003 competition (2nd on random data; 3rd on "handmade" data)
- Study the use of SVM regression rather than least squares regression
  - our initial results show that SVMs outperform least-squares models, albeit by a fairly small "margin"

We would like to acknowledge Ryan Porter, Carla Gomes and Bart Selman for their assistance. This work was supported by DARPA grant F30602-00-2-0598 and a Stanford Graduate Fellowship.