Algorithm Portfolios

- Motivation: No one algorithm dominates all others on all problems…
  Basically the NFL theorem in practice.
- Solution:
  - Construct a framework
  - Select which algorithm to run for a particular problem instance

Types of Portfolios I

Parallel:

- Run *n* algorithms simultaneously
- Stop when one finds an acceptable solution or return the best solution when they run out of time

- Benefits:
  - Makes excellent use of multi-core or distributed platforms
  - Works well with high variance in performance
  - No need to model the algorithms
- Weakness:
  - May require *n* times more CPU time than necessary

Parallel portfolios

- Allocate copies of a set of *m* stochastic algorithms to *n* processors (assume $m<n$)
  - Given a probability distribution of algorithm performance, can construct portfolio to minimize time at which first is done [Gomes & Selman 1997]

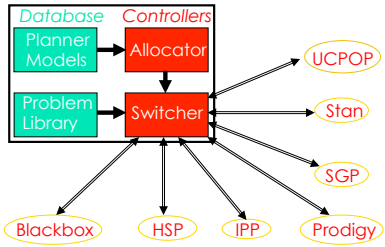
Types of Portfolios II

- Round Robin Strategy:
  - Order algorithms by "best" and try one at a time
  - Assuming you can stop and re-start processes, can allocate time slices
- Benefit:
  - Can be very efficient and effective
- Weakness:
  - Requires strategy for selecting algorithms and allocating time slices

Example: Round Robin

Bus: a planner portfolio (circa 2000)

- Learned multiple linear regression models of likelihood of success and time to succeed
- Ordered by: P(suc)/Expected-Time
- Time slice = Expected-time
- Then each add'l planner got that much more time in Round Robin until solved



Types of Portfolios III

- Sequential:
  - Select the best algorithm to run and amount of time to run
  - Repeat until problem is solved or time runs out
- Benefit:
  - Can be very efficient and effective
- Weakness:
  - Requires selection strategy for algorithm and time

Constructing a Sequential Portfolio

- By hand:
  - For each solver and instance, collect data on whether instance is solved and how long it takes
  - Look for combination that solves the most instances
  - Identify best order and then time required for each and crossover point where most are solved, e.g., allocate 60% of time to first solver and 40% to second

Example: SAT competition

Rules: 5000 second, return UNSAT or solution if SAT; given set of 150 problem instances some carry over from prior year; winner solves most instances correctly, ties broken by CPU time

Methodology:

1. Collect data (return, CPU time) on 150 instances from 2013 for 10 solver configurations
2. Determine minimal solver set that covers all solvable instances
3. Compute union of pairs of these solvers
4. Sort solvers in each viable pair (solved > some threshold) by CPU
5. Identify time allocation by looking at instances solved by only 1

Constructing a Sequential Portfolio (cont.)

- Using ML off-line:
  - For each solver and instance, collect data on whether instance is solved and how long it takes
  - For each instance, collect features
  - Construct model(s) to predict which solver is best for each instance and/or required time slice

Constructing a Sequential Portfolio (cont.)

- Using ML on-line:
  - Construct model(s) to predict how to extrapolate from early performance
  - Run each solver for a short time on a new instance
  - Select the one predicted to perform the best

Example: SATzilla

- 3-of-n SAT portfolio: executed 2 pre-solvers followed by one main solver
- a-of-n portfolio: "a set of n algorithms and a procedure for selecting among them with the property that if no algorithm terminates early, exactly a algorithms will be executed "
- Algorithm selection based *empirical hardness model*: inexpensive predictor of an algorithm's runtime based on features of a problem instance and the algorithm's past performance

SATzilla: offline portfolio construction

[Xu, Hutter, Hoos & Leyton-Brown JAIR 2008]

1. Identify a target distribution/set of problem instances.
2. Select a set of solvers that have relatively uncorrelated runtimes and should perform well on some instances.
3. Identify features that characterize problem instances and are fast to compute.
4. On a training set of problem instances, compute these features and run each algorithm to determine its running times.
5. Identify one or more solvers to use for pre-solving (run for a short time to solve easy instances)

SATzilla: offline portfolio construction (cont.)

6. Using a validation data set, identify a backup solver which achieves the best performance for instances that are not solved by the pre-solvers and on which the feature computation times out.
7. Construct an empirical hardness model for each algorithm in the portfolio, which predicts the runtime of the algorithm for each instance, based on the instance's features.
   - Feature selection: forward selection which greedily adds one feature at a time, minimizing cross-validation error
   - Learning: ridge regression (a multiple non-linear regression model)
   - Treat censored values (e.g., failures) as lower bounds
8. From all given solvers, select a subset for which the respective portfolio (which uses the empirical hardness models learned in the previous step) achieves the best performance on the validation set.

SATzilla: online portfolio execution

1. Run each pre-solver until a predetermined fixed cutoff time is reached.
2. Compute feature values.
3. If features cannot be computed for some reason, run the backup solver.
4. Else, predict each algorithm's runtime using the empirical hardness models.
5. Run the algorithm predicted to be the best. If a solver fails without solving, run the next best algorithm.

Related Idea: Automated Algorithm Configuration

Find best parameter settings given a set of tuning instances.

- irace: [Lopez-Ibanez et al.]

1. sample new configurations according to a particular distribution,
2. select the best configurations from the newly sampled ones by means of racing (remove those that perform statistically worse),
3. update the sampling distribution in order to bias the sampling towards the best configurations.