Application-Specific Algorithm Selection

Tim Roughgarden (Stanford)

joint work with Rishi Gupta

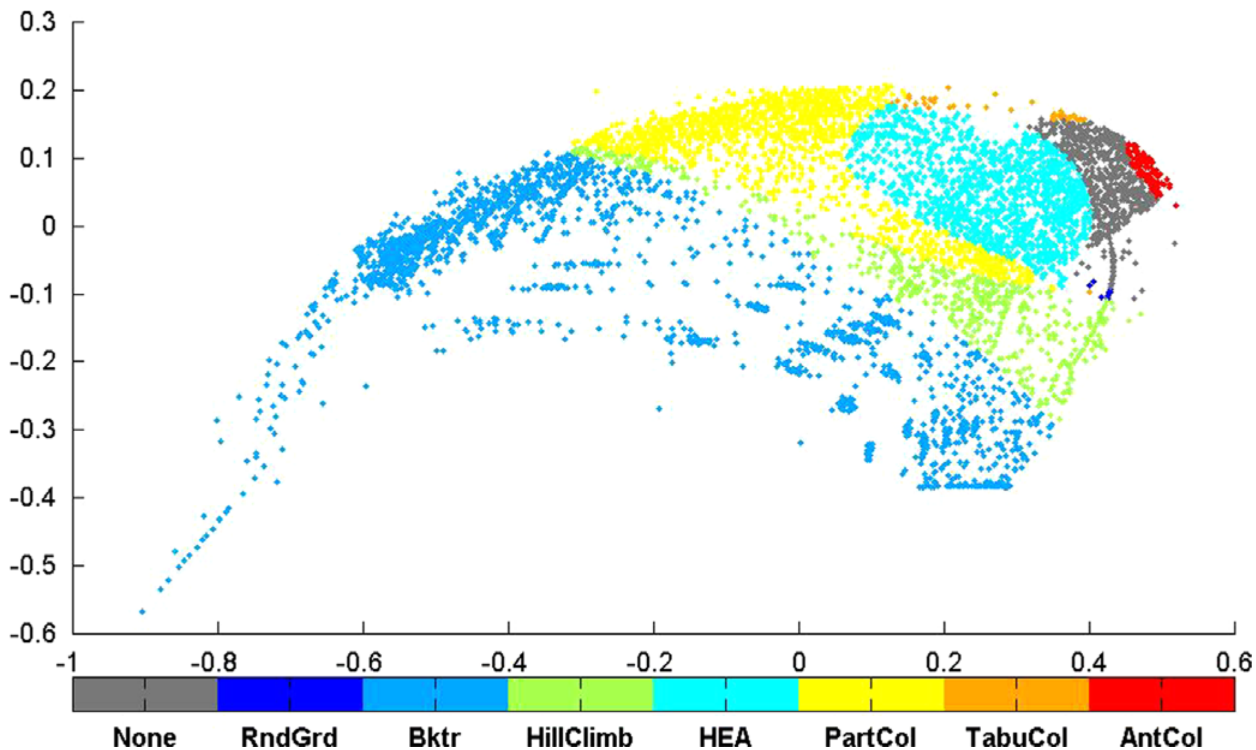
Algorithm Selection

"I need to solve problem X. Which algorithm should I use?"

Answer usually depends on the details of the application (i.e., the instances of interest).

- for most problems, no "silver bullet" algorithm

Graph Coloring



(from Smith-Miles/Baatar/Wreford/Lewis (2014))

Example #1: SATzilla

SAT competition: enter your best SAT solver, will be run on instances from diverse domains.

Bold idea: [Xu/Hutter/Hoos/Leyton-Brown] design "meta-algorithm" for smartly deploying a portfolio of existing solvers.

- uses coarse features of an instance to select a solver

(spoiler: won multiple SAT competitions)

Leyton-Brown

Example #1: SATzilla

[Xu/Hutter/Hoos/Leyton-Brown]

- Portfolio = 7 SAT solvers
  - widely varying performance
- Identify coarse features of SAT instances
  - clause/variable ratio, Knuth's search tree estimate, …
- Use regression to learn good "empirical performance models (EPMs)," mapping input features to predicted solver running time.
- Run solver predicted to be fastest by EPMs.

Example #2: FCC Auctions

Broadcast Television Incentive Auction (ongoing):

- Reverse Auction: buy TV broadcast licenses
  - CBO estimate: $15 billion cost
- Forward Auction: sell 4G wireless licenses.
  - CBO estimate: $40 billion revenue.
- Revenue to cover auction costs, fund a new first responder network, reduce the deficit (!)
  - "Middle Class Tax Relief and Job Creation Act"

Reverse Auction Algorithm

Question: which stations stay on the air?

[Milgrom/Segal 14] use a greedy algorithm ("descending clock auction")

- good: higher value for broadcasting
- bad: more interference
- scoring rule: rank by (value)/(# conflicting stations)$^{1/2}$
- a la [Lehmann/O'Callaghan/Shoham 02]

Milgrom

Segal

On Parameter Tuning

Case Study #1: machine learning.

- e.g., choosing the step size in gradient descent
- e.g., choosing a regularization parameter

Case Study #2: CPLEX. (LP/IP solver).

- 135 parameters! (221-page reference manual)
- manual's advice: "you may need to experiment with them" (gee, thanks…)

Example #3: Self-Improving Algorithms

Model: receive sequence of inputs drawn independently from unknown input distribution F.

Goal: quickly converge to a near-optimal algorithm (w.r.t. F). [using small space]

- sorting
  - [Ailon/Chazelle/Liu/Seshadhri 06]
- Delaunay triangulations
  - [Clarkson/Seshadhri 08]
- convex hulls
  - [Clarkson/Mulzer/Seshadhri 10]

Seshadhri

A Theory of Algorithm Selection?

Question: what would a theory of "application-specific algorithm selection" look like?

- need to go "beyond worst-case analysis"

Worst-Case Analysis

Worst-case analysis: $\mathrm{cost}(A) := \sup_z \mathrm{cost}(A,z)$

- cost(A,z) = performance of algorithm A on input z

Pros of WCA: universal applicability (no input assumptions)

- countless killer applications
- relatively analytically tractable

Cons of worst-case analysis: overly pessimistic

- can rank algorithms inaccurately (LP, paging)
- no data model (rather: "Murphy's Law" model)

A Theory of Algorithm Selection?

Question: what would a theory of "application-specific algorithm selection" look like?

- need to go "beyond worst-case analysis"

Idea: model as a learning problem.

- algorithms play role of concepts/hypotheses
- algorithm performance acts as loss function
- two models: offline (batch) learning and online learning (i.e., regret-minimization)

Formalism

Given: a class C of algorithms for some problem π.

- could be finite (coloring, SAT) or infinite (parameter-tuning)
- no single "silver bullet" algorithm

Given: a cost function cost(A,z) of algorithm A on input z (running time, solution quality, etc.) (range = [0,H])

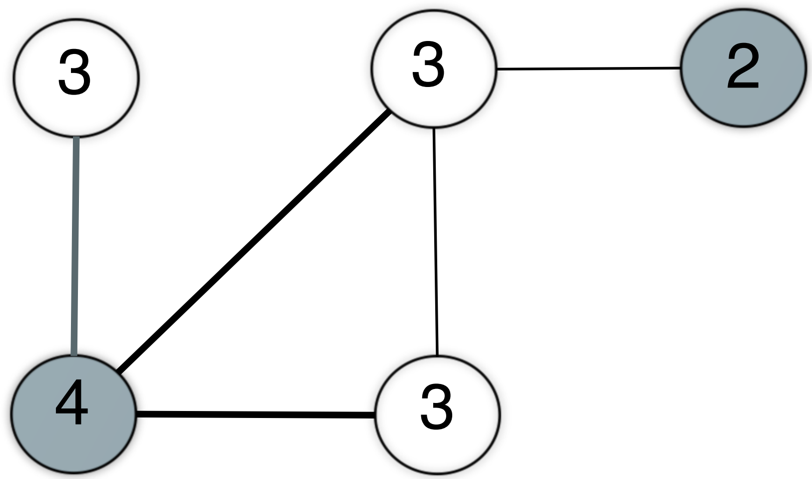
Perspective: think of each algorithm A as a real-valued function:

$$z \rightarrow \text{cost}(A,z)$$

input performance of A on input

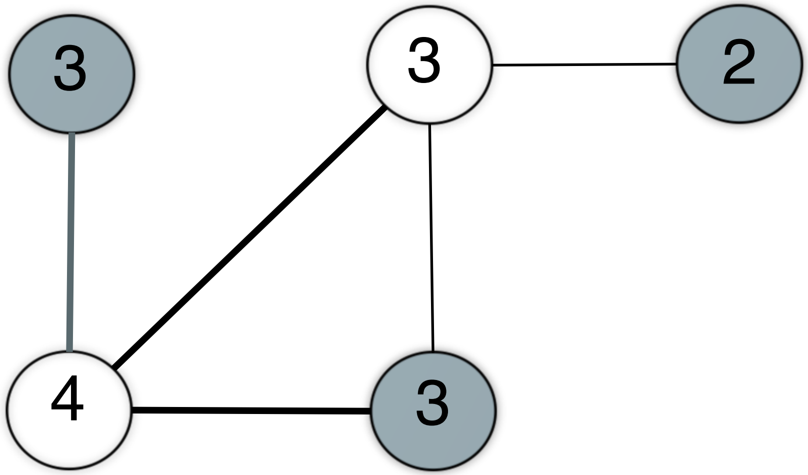
Example: Independent Set

(no adjacent vertices allowed)



Greedy algorithm #1: process vertices in decreasing order of w_v.

Example: Independent Set

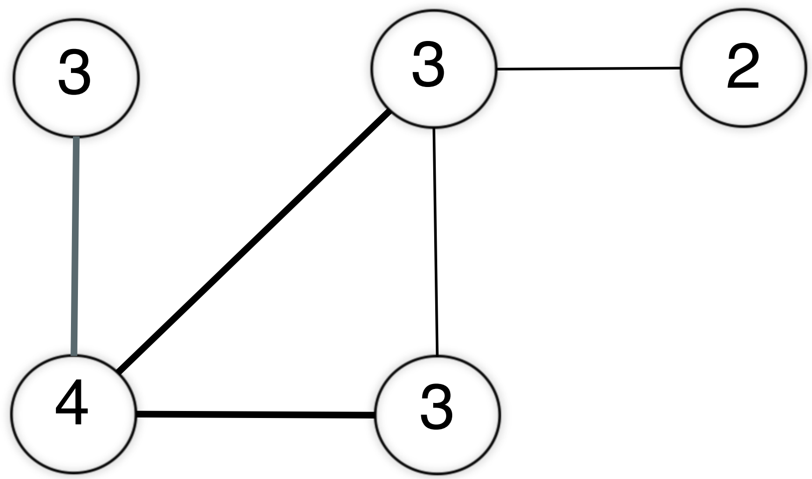


(no adjacent vertices allowed)

Greedy algorithm #2: process vertices in decreasing order of $w_v/(1+deg(v))$.

Example: Independent Set

(no adjacent vertices allowed)



Example class C of algorithms: all greedy algorithms that rank by $w_v/(1+\deg(v))^p$ for a parameter $p \geq 0$.

- can be adaptive or non-adaptive

Model #1: Unknown Distribution

Offline ("Batch") Learning Model: ($\approx$ PAC learning)

- unknown distribution F over inputs z of problem π
- receive s i.i.d. samples $z_1,\ldots,z_s$ from F
- based on sample, choose an algorithm A of C to use on all future inputs
  - extension: choose mapping from instance features to algorithms (a la SATZilla)

Goal: identify A^* that (approximately) minimizes

$$E_{z\sim F}[\text{cost}(A,z)] \quad \text{(over A in C)}$$

High-Level Plan

Lesson from learning theory: sample complexity scales with "complexity" of the "hypothesis class."

- e.g., VC dimension

Corollary: the best "simple" hypothesis can be learned from a modest amount of data.

Proposed simplicity measure of a class C of algorithms: *pseudodimension* of the real valued functions (from inputs to performance) induced by C.

Bounding the Sample Complexity

Theorem: [Haussler 92], [Anthony/Bartlett 99] if C has low *pseudodimension*, then it is easy to learn from data the best algorithm in C.

- obtain $s = \tilde{\Omega}(H^2 \varepsilon^{-2} d)$ samples $z_1,\ldots,z_s$ from F, where d = pseudodimension of C (range of cost = [0,H])
- let A^* = algorithm of C with minimum average cost on the samples

Guarantee: with high probability, expected cost of A^* (w.r.t. F) within ε of optimal algorithm in C.

Pseudodimension: Examples

$64K question: do interesting classes of algorithms have small pseudodimension?

Examples:

• finite set C	O(log |C|)
• single-parameter greedy algorithms	O(log n)
• local search with neighborhood size n^k	O(k log n)
• "bucket-based" sorting algorithms	O(n log n)
• per-instance algorithm selection	O(|F| • pd(C))

Pseudodimension: Definition

[Pollard 84] Let F = set of real-valued functions on X.

(for us, X = instances, F = algorithms, range = cost(A,z))

F *shatters* a finite subset S={$\mathbf{v}_1,\ldots,\mathbf{v}_s$} of X if:

- there exist real-valued thresholds $t_1,\ldots,t_s$ such that:
- for every subset T of S
- there exists a function f in F such that:

$$f(\mathbf{v}_i) \geq t_i \;\Leftrightarrow\; \mathbf{v}_i \text{ in } T$$

Pseudodimension: maximum size of a shattered set.

Pseudodimension: Example

Let C = WIS greedy algorithms with scoring rule of the form $w_v/(\deg(v)+1)^p$ (e.g. for $p \geq 0$)

Claim: C can only shatter a subset $S=\{z_1,\ldots,z_s\}$ if $s = O(\log n)$. (hence pseudodimension $O(\log n)$)

Proof idea: Fix S. Call p,q *equivalent* if they induce identical executions on all inputs of S.

- Lemma: number of equivalence classes can only grow polynomially with n,s (uses "single-parameter" property)
- Since need 2^s labelings to shatter S, $s = O(\log n)$.

Proof of Lemma

Lemma: number of equivalence classes can only grow polynomially with n,s (uses "single-parameter" property).

Proof idea: Fix sample S of size s.

- greedy alg depends only on results of comparisons
- *single-crossing property:* for each possible comparison (between two vertices), flips at most one as p goes from 0 to infinity $[w_v/(deg(v)+1)^p$ vs. $w_x/(deg(x)+1)^p]$
- # possible comparisons = poly(n,s)
- only poly(n,s) distinct algorithms (w.r.t. S)

Pseudodimension: Upshot

Examples:

- finite set C — $O(\log |C|)$
- single-parameter greedy algorithms — $O(\log n)$
- local search with neighborhood size n^k — $O(k \log n)$
- "bucket-based" sorting algorithms — $O(n \log n)$
- per-instance algorithm selection — $O(|F| \cdot pd(C))$

Recall: Can learn the best algorithm with sample complexity polynomial in the pseudodimension.

- also: running time at most exponential in dimension

Gradient Descent

Recall: for strongly convex functions, have convergence guarantee for all sufficiently small step sizes.

In practice: use much more aggressive step sizes in hopes of converging more quickly.

Result: can learn the best step size (to minimize expected # of iterations) from few samples.

Open: more generally, hyperparameter optimization?

Selecting an Algorithm Online

Online learning setup: (fix a problem π)

- set of actions known up front (for us, algorithms of C)
- each time step t=1,2,…,T:
  - we commit to a distribution p^t over actions/algorithms
  - adversary picks a cost vector (here, induced by an instance z of P)
  - algorithm A selected according to p^t
  - incur cost(A,z)

Details: *see Rishi Gupta's talk at BWCA workshop (Nov 16)*

Regret-Minimization

Benchmark: best fixed algorithm A of C (in hindsight) for the adversarially chosen inputs $z_1,\ldots,z_T$

Goal: online algorithm that, in expectation, always incurs cost at most benchmark, plus o(T) error term.

Question #1: Weighted Majority/Multiplicative Weights?

- issue: what if A an infinite set?

Question #2: extension to Lipschitz cost vectors?

- issue: not at all Lipschitz! (e.g., for greedy WIS)

A Negative Result

Theorem: for a sufficiently large constant n and arbitrary nonnegative vertex weights, there is no online algorithm with a non-trivial regret guarantee for the greedy WIS algorithm selection problem.

- idea: each day t, learning algorithm knows an interval of length 2^{-t} that contains the optimal value of p, but if it guesses the wrong half it incurs high cost
  - (crucially exploits non-Lipschitzness)

A Smoothed Guarantee

Theorem: for "smoothed WIS instances" (a la [Spielman/Teng 01]), can achieve expected regret 1/poly(n) as T -> infinity

Idea:

- run a no-regret algorithm using a "net" of the space of algorithms
- smoothed instances => the optimal algorithm is typically equivalent to one of the net algorithms

Open Questions

- non-trivial learning algorithms? (or a proof that, under complexity assumptions, none exist)
- extend gradient descent result to more general hyperparameter optimization problems
- trade-offs between representation and learning error
- connections to more traditional measures of “algorithm/problem complexity”?

Summary

- application-specific algorithm selection naturally modeled as a learning problem (offline or online)
- for the offline/distributional model, use pseudodimension to bound the sample complexity of learning the best algorithm
  - pseudodimension is low for many natural algorithm classes
  - analytically tractable
- for the online/distribution-free model, no-regret impossible in worst case, possible for smooth instances