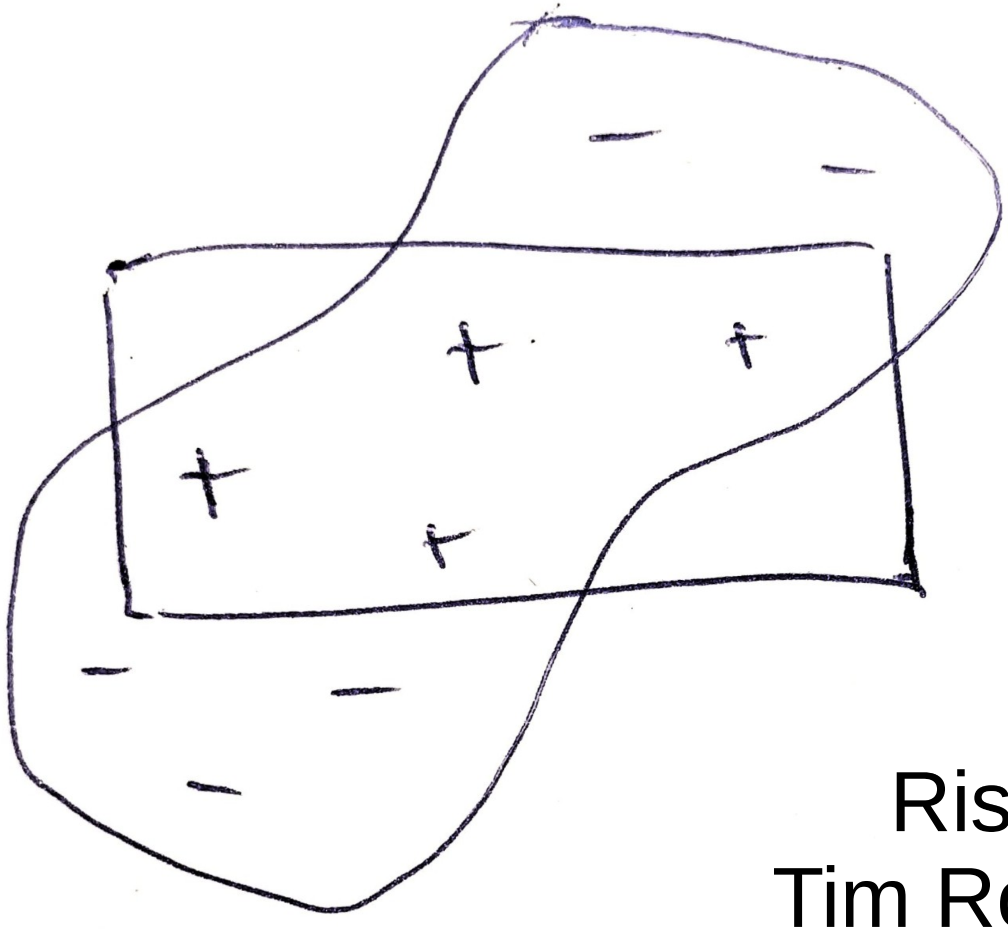
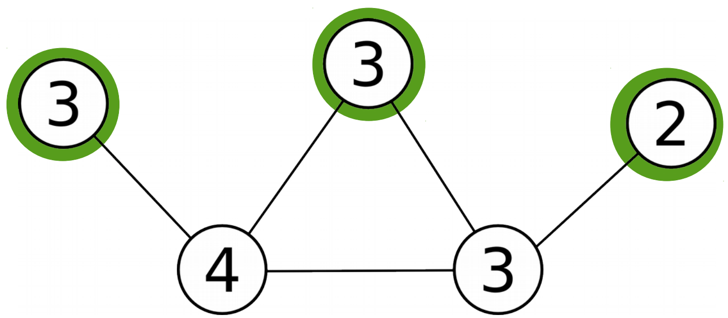
A PAC Approach to Application-Specific Algorithm Selection



Rishi Gupta
Tim Roughgarden
Simons, Nov 16, 2016

Motivation and Set-up

Problem



Max weight indep. set

Algorithms

Alg 1: Greedy algorithm

Alg 2: Linear programming

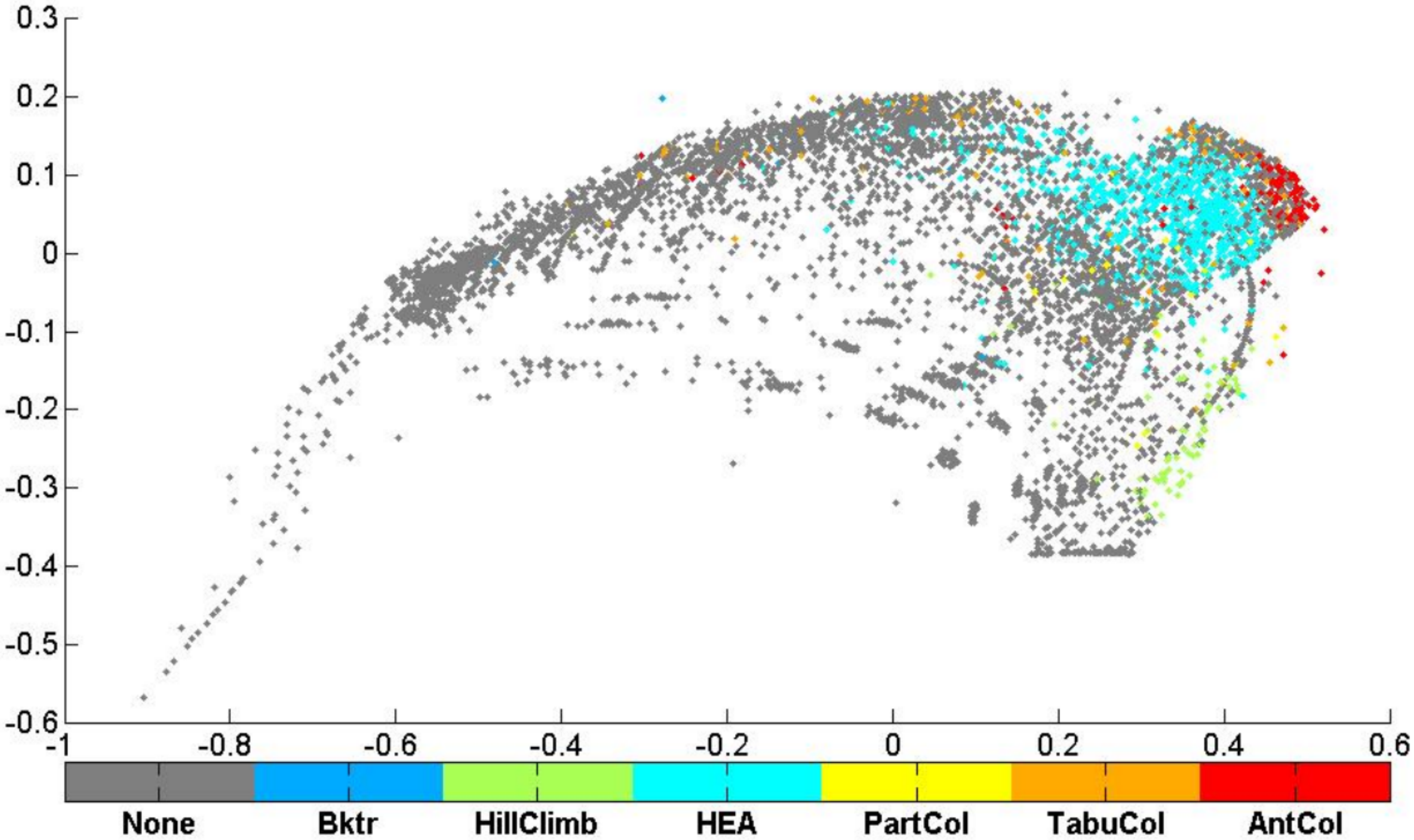
Alg 3: Local search

- Question: Which algorithm is the best?
- Advice from theory: Worst-case analysis, parameterized analysis, average-case analysis, etc.
  - Allows you to use what you know

Motivation

- Issues in practice
  - Instances have structure, but hard to articulate
  - Structure matters; the best algorithm is often not one suggested by worst-case analysis.
  - Often the algorithm is github/mwis.
  - Often there are many choices to make, e.g. algorithm parameters.
- How can theory help?

Graph coloring

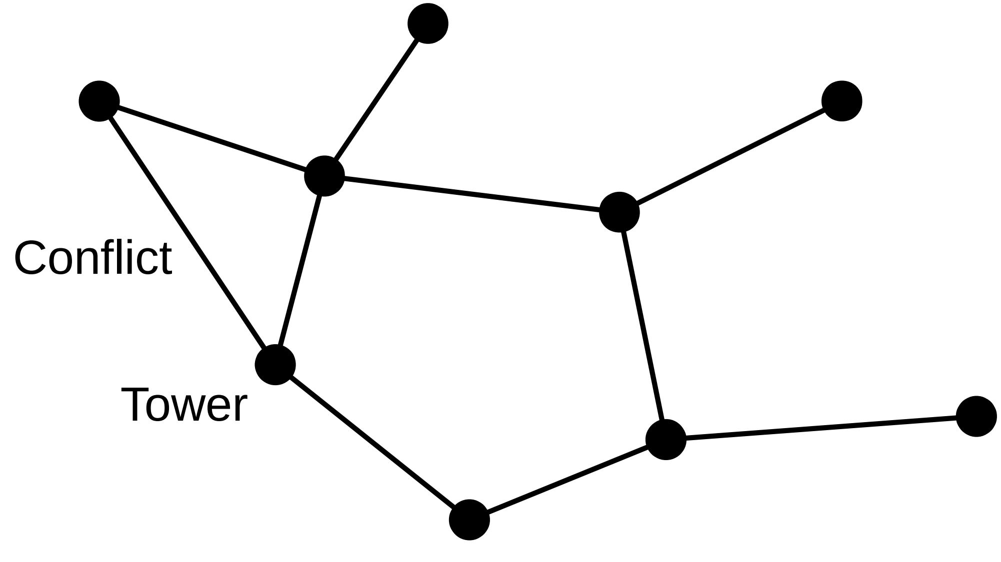


Towards objective measures of algorithm performance across instance space,
Smith-Miles/Baatar/Wreford/Lewis '14

- Instances have structure, but hard to articulate.
- Structure matters; the best algorithm is often not one suggested by worst-case analysis.
- Often the algorithm is github/mwis.
- Often there are many choices to make, e.g. algorithm parameters.

FCC Auctions

- *Broadcast Television Incentive Auction* (2016)
  - Buy TV licenses: Estimated 15B cost.
  - Sell 4G licenses: Estimated 40B revenue.
- Goal: buy out independent sets.



FCC Auctions

- Reverse auction (“descending clock auction”)
  - Broadcasters offered an initial price
  - People leave when the price is too low
- Goal: set initial prices such that you buy:
  - Broadcasters that don’t value their license much
  - Broadcasters that don’t interfere much
- Solution: initial price = V * C^.5

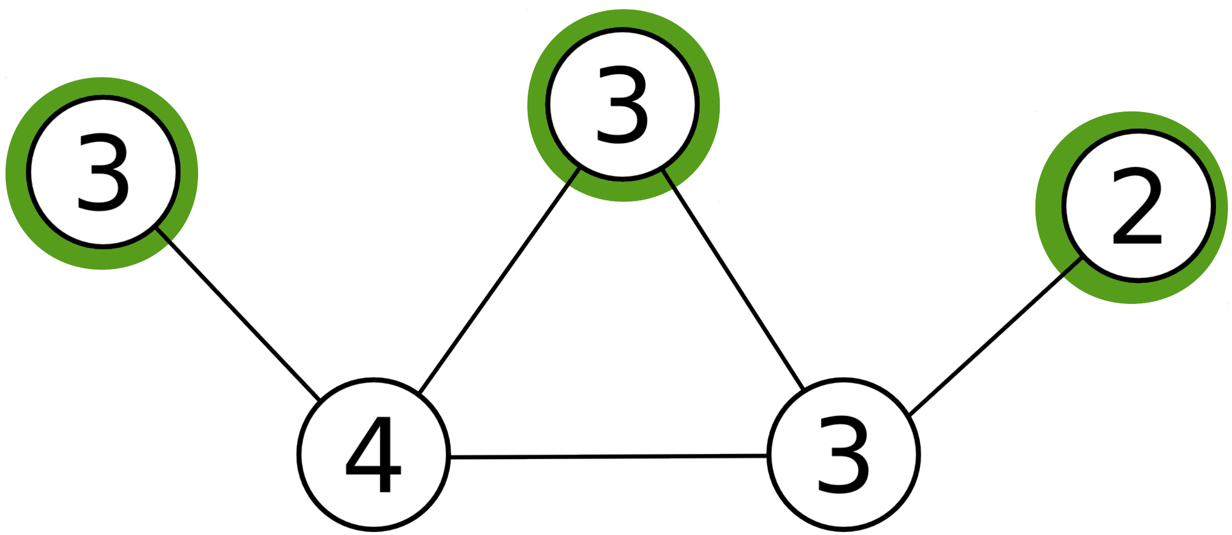
- Instances have structure, but hard to articulate.
- Structure matters; the best algorithm is often not one suggested by worst-case analysis.
- Often the algorithm is github/mwis.
- Often there are many choices to make, e.g. algorithm parameters.

General set-up

- Given source of problem instances:
  - Application generating graph coloring instances
  - FCC auction simulations
- Given a family of algorithms.
  - Seven algorithms you downloaded
  - Different settings of the exponent
- Task: pick which algorithm to run.

Running example (Problem)

- Maximum Weight Independent Set (MWIS)

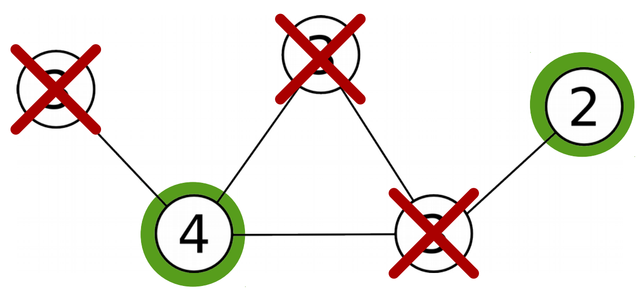


Find subset of nodes of maximum weight,
such that no two nodes share an edge

Running example (Algorithms)

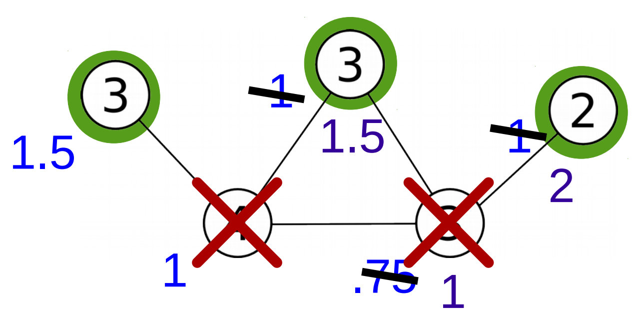
- Greedy ρ: Order by weight/(degree+1)^ρ, for ρ ∈ [0,1].

Greedy 0: (Repeatedly) take vertex of highest weight



Value: 6

Greedy 1: Take vertex of highest weight/(degree+1)



Value: 8

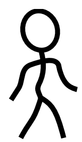
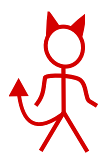
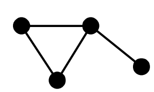
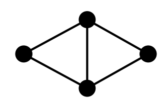
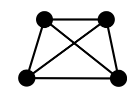
Notation: A_ρ for algorithm Greedy ρ

Two models

- Running example (Task)
  - Given a source of MWIS instances, pick an A_ρ
- Learning models
  - Inspired by online regret (learning from experts)
  - Inspired by PAC learning
  - Grounded in familiar theory

Model 1: Online regret

For T days, pick a ρ_t, and get an adversarial graph.
Benchmark: Best single ρ^* for those T graphs (in hindsight)

Learner 		$A_{.7}$	$A_{.3}$	$A_{.5}$	
Adversary 					
	Value	1	5	0	Sum = 6
Benchmark ($A_{.4}$)		7	2	3	Sum = 12

Average *regret* = (12 – 6) / 3 = **2**

Goal: Average regret –> 0 as T –> infinity (aka "no-regret")

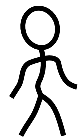
No computational restrictions.

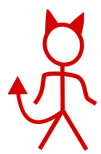
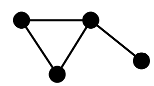
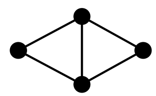
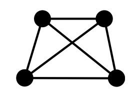
Number of vertices n is fixed. Max value is bounded.

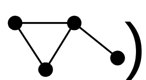
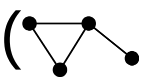
Model 1: Online regret

For T days, pick a ρ_t, and get an adversarial graph.

Benchmark: Best single ρ^* for those T graphs (in hindsight)

Learner  $A_{.7}$ $A_{.3}$ $A_{.5}$

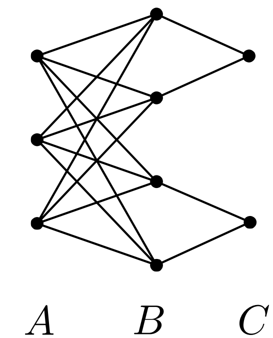
Adversary    

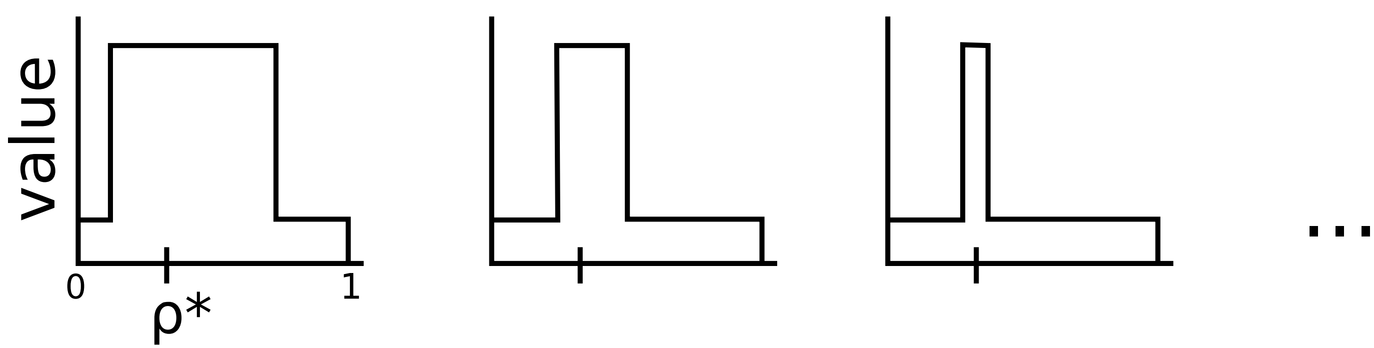
Our Model	Learning from experts
Algorithms (A_ρ)	Experts
Instances () induce A_ρ() for $\rho \in [0,1]$	Cost vectors

Model 1: Online regret

For T days, pick a ρ_t, and get an adversarial graph.
Benchmark: Best single ρ* for those T graphs (in hindsight)
Goal: Average regret –> 0 as T –> infinity

- First try: multiplicative weights
  - Infinite number of experts
  - Cost vector is not Lipschitz
- Theorem: Not possible for MWIS with A_ρ
  - Even with oblivious adversary





Model 1: Online regret

For T days, pick a ρ_t, and get an adversarial graph.
Benchmark: Best single ρ^* for those T graphs (in hindsight)

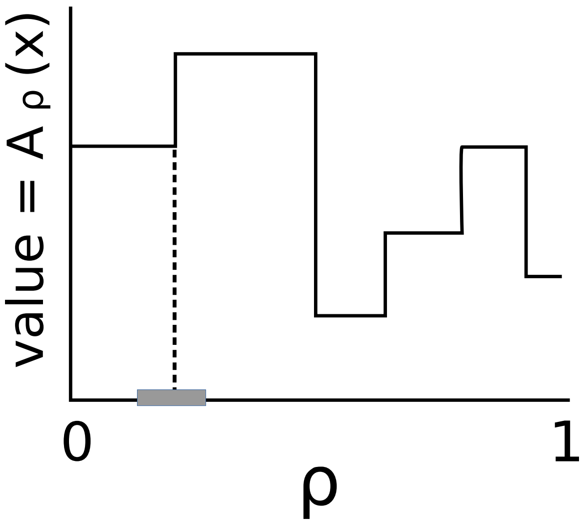
- New goal: Average regret –> ~~0~~ 1/poly(n) as T –> infinity ("~~no~~ low-regret")
- σ-Smoothing: Gaussian of width 1/σ added to each node weight.
- Theorem: poly(n,σ) time algorithm with expected regret 1/poly(n) after σ-smoothing.

Proof of Theorem

poly(n,σ) time algorithm with expected regret 1/poly(n) after σ-smoothing

Fix arbitrary MWIS instance x.

- Fact 1: cost vector = step function with poly(n) steps
- Fact 2: σ-smooth weights => Ω(σ)-smooth step boundaries



=> If T = poly(n), an ε-net of experts will include something equivalent to ρ* (on all of the T graphs) with high probability

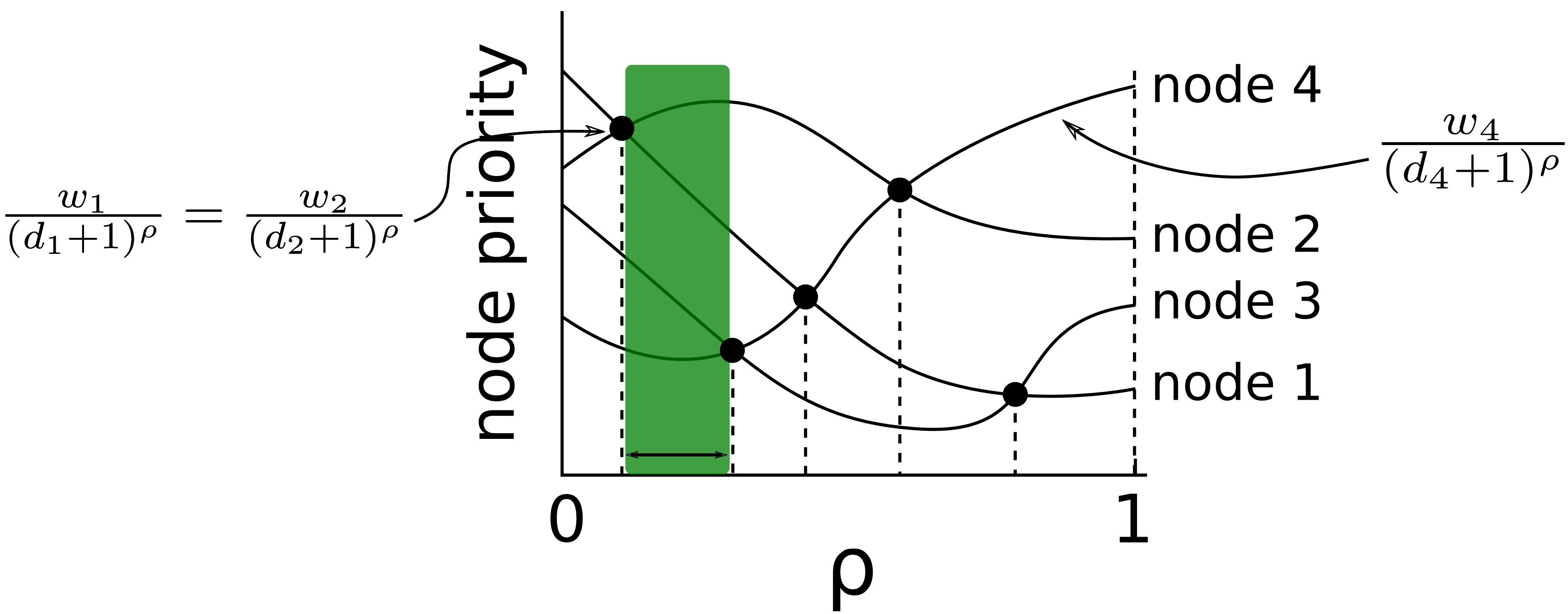
–> multiplicative weights on the ε-net of experts

Proof of Theorem (Fact 1)

cost vector = step function with poly(n) steps

Fixed MWIS instance x

- Cost vector only changes at intersections below
- Only n^2 possible intersections!



Model 2: PAC Learning

Adversarial distribution D over graphs. See m samples from D, pick A_ρ.
Benchmark: Best A_{ρ^*} for D.

(Expected) *error* = $A_{\rho^*}(D) - A_\rho(D)$

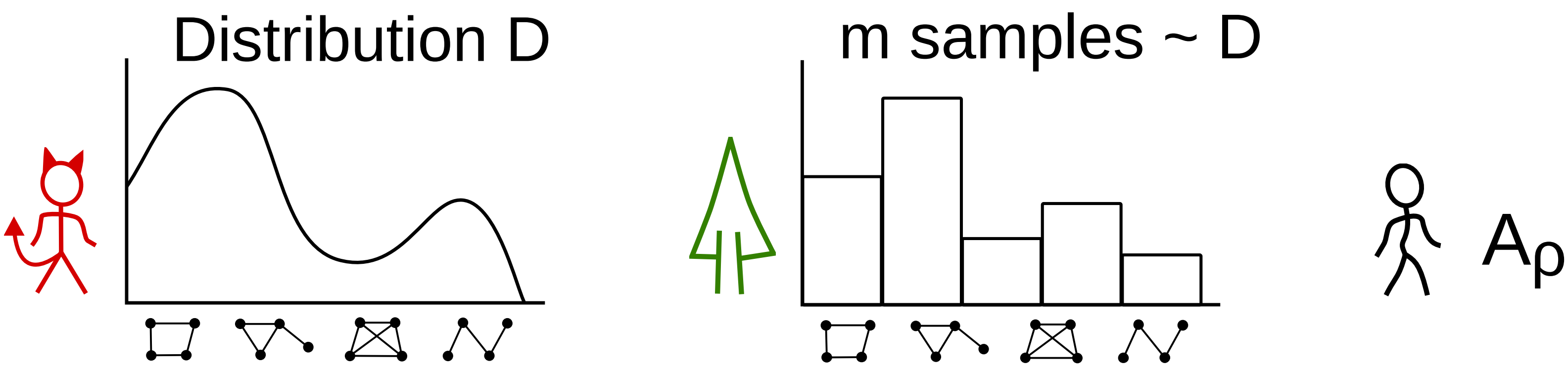
Goal: ε error, polynomial m

Results preview: $m = \tilde{O}(\log n)/\varepsilon^2$ MWIS & others,
$\tilde{O}(n^{1+c})/\varepsilon^2$ bucket sort, $\tilde{O}(1/c^3)$ gradient descent

As before: no computational restrictions, and max value is bounded

Model 2: PAC Learning

Adversarial distribution D over graphs. See m samples from D, pick A_ρ.
Benchmark: Best A_{ρ^*} for D.



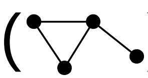
(Expected) error = $A_{\rho^*}(D) - A_\rho(D)$

Model 1	Model 2
Online decisions, adaptive adversary	Offline decisions, oblivious adversary
No distribution	Hidden distribution

Model 2: PAC Learning

Adversarial distribution D over graphs. See m samples from D, pick A_ρ.
Benchmark: Best A_{ρ^*} for D.

(Expected) error = $A_{\rho^*}(D) - A_\rho(D)$

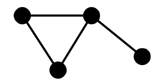
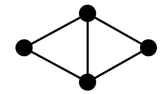
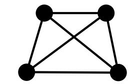
Model 2	PAC Learning
Algorithms (A_ρ) induce functions from instances () –> **R**	Concept
Set of algorithms	Concept class Hypothesis class

Model 2: PAC Learning

Adversarial distribution D over graphs. See m samples from D, pick A_ρ.
Benchmark: Best A_{ρ^*} for D.
Goal: ε error with polynomial m

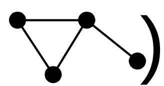
- Learner's task: choose A_ρ
  - Hope that samples are a good guide to D
  - Pick the A_ρ that works best on the m samples (ERM)
  - Computational issues: infinite number of A_ρ
- Reduction to *pseudo-dimension* of the set of algorithms A (similar to VC dimension, fat shattering dimension)
  - Theorem [Hau92]: If A has pseudo-dimension d, and $m \cong \tilde{\Omega}(d/\varepsilon^2)$, then the output of ERM has error < ε.

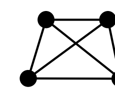
Pseudo-dimension

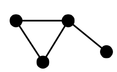
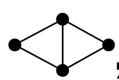
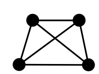
Fix d instances:   

Fix d thresholds: 3 7 2

M : $A_{.4}$ –> 1 1 0

$A_{.4}($  $) \geq 3$

$A_{.4}($  $) < 2$

A *pseudo-shatters* (, , ) at (3, 7, 2) if M is surjective.

The pseudo-dimension of A is the largest d for which there exist X and T such that A pseudo-shatters X at T.

Example: if |A| = k, then pseudo-dimension of $|A| \leq \log k$

Model 2: Example Results

If A has pseudo-dimension d, and m = $\Omega(d/\varepsilon^2)$, then
the output of ERM has error < ε.

- Natural families of greedy algorithms for MWIS, Knapsack, Machine Scheduling have d = O(log(n))
- Similar families of local search algorithms have d = O(log(n))
- Bucket sort with n buckets (n-1 parameters, 1 for each bucket boundary) has d = $O(n^{1+c})$ (reinterpretation of one part of self-improving algorithms in [ACCL06])
- Gradient descent (ρ = step size) in smooth, strongly convex functions has 1-fat shattering dimension 1/c, where c is the minimum progress made by each step.

For all examples above, best algorithm can be found in polynomial time.

Open directions

- Extend gradient descent to machine learning parameter tuning?
- A that is both near-optimal, and has low pseudo-dimension?
- A with poly sampling complexity, but where the learning algorithm (ERM) has super-poly runtime? (under P != NP, crypto assumptions)
  - Would approximately learning the best algorithm help?
- Other non-trivial relationships between pseudo-dimension and more traditional complexity measures?

Thank you!