# CS264: Beyond Worst-Case Analysis
# Lecture #20: From Unknown Input Distributions to Instance Optimality*

Tim Roughgarden†

December 3, 2014

## 1 Preamble

This lecture closes the loop on the course's circle of ideas, by connecting our most recent topic (algorithms for unknown input distributions) to our first topic (instance optimality guarantees). We use this connection to reinterpret classical results about "no-regret algorithms" for online decision-making.

## 2 Instance Optimality via the Distributional Thought Experiment

### 2.1 Review of Instance Optimality

Recall from Lecture #1 that an algorithm $A$ is *instance optimal* if there is a constant $\alpha$ such that, for every algorithm $B$ and input $z$,

$$\text{cost}(A, z) \leq \alpha \cdot \text{cost}(B, z).$$

For small $\alpha$, this is the strongest possible and least controversial notion that one could hope for. Indeed, the guarantee is so strong that, for many problems, no instance-optimal algorithms exist (recall Homework #1).

A relaxation of instance optimality that can enable meaningful positive results is to replace the condition "for every algorithm $B$" by "for every algorithm $B$ belonging to a class $\mathcal{C}$ of 'natural' algorithms." Obviously, the strength of such a guarantee increases with the set $\mathcal{C}$ of "competing algorithms."



†Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

Our main example of instance optimality, the Kirkpatrick-Seidel algorithm for computing the maxima of a point set in the plane, essentially made use of such a relaxation. Recall that our formalism in Lecture #2 effectively restricted attention to algorithms — or rather, running time bounds of algorithms — that are independent of the order in which the points are listed in the input. This maneuver excluded silly algorithms that memorize the answer for a single input.

Applying this relaxed version of instance optimality requires a choice of a set $\mathcal{C}$ of algorithms. What's a good choice? We're looking for a sweet spot — a class that is small enough to permit illuminating positive results, yet big enough that an instance-optimality guarantee is impressive. Ideally, we want to choose $\mathcal{C}$ so that at least one but very few algorithms are instance optimal.

## 2.2 From Distributions to Algorithms

We can use input distributions to provide a principled two-step approach to choosing the set $\mathcal{C}$ of competitor algorithms. While the last several lectures took a distributional assumption literally, and sought algorithms that perform well with respect to the assumed distribution, here we use distributions only in a "thought experiment" to define the set $\mathcal{C}$. With respect to this set, we'll seek input-by-input (i.e., instance-optimal) performance guarantees.

Precisely, for a computational problem, the first step is to choose a set $\mathcal{D}$ of input distributions. You're accustomed to such families from our many discussions of semi-random and hybrid analysis frameworks — such as perturbed versions of worst-case inputs (as in Lectures #12-15), "sufficiently random" distributions (as in Lecture #16), distributions over arrays with independent entries (Lecture #17), or "regular" distributions (Lecture #18). Any of these are examples of a set $\mathcal{D}$ of input distributions, and of course there are many other natural examples.

The second step is to use the set $\mathcal{D}$ of input distributions to define a corresponding set $\mathcal{C}_{\mathcal{D}}$ of algorithms:

$$\mathcal{C}_{\mathcal{D}} := \{\text{algorithms } A : A \text{ is optimal w.r.t. some } D \in \mathcal{D}\}. \tag{1}$$

By "optimal with respect to distribution $D$," we mean that the algorithm $A$ minimizes the expected cost $\mathbf{E}_{z\sim D}[\text{cost}(B, z)]$ over all algorithms $B$, where the expectation is over the random input (distributed according to $D$). Conceptually, we iterate over all distributions $D \in \mathcal{D}$ of interest, and put all of the corresponding optimal algorithms into the set $\mathcal{C}_{\mathcal{D}}$.

The key point is that an instance-optimal guarantee with respect to $\mathcal{C}_{\mathcal{D}}$ is at least as strong as a guarantee that holds for an arbitrary unknown input distribution from $\mathcal{D}$.

**Proposition 2.1** *Let $\mathcal{D}$ be a set of input distributions and define $\mathcal{C}_{\mathcal{D}}$ as in* (1)*. If algorithm $A$ is instance-optimal with respect to $\mathcal{C}_{\mathcal{D}}$ with approximation factor $\alpha$, then*

$$\mathbf{E}_{z\sim D}[\mathit{cost}(A, z)] \leq \alpha \cdot \mathbf{E}_{z\sim D}[\mathit{cost}(OPT_D, z)] \tag{2}$$

*for every $D \in \mathcal{D}$, where $OPT_D$ denotes the expected cost-minimizing algorithm with respect to the input distribution $D$.*

What's important to notice is that the algorithm $A$ on the left-hand side of (2) is independent of the distribution $D$, while the algorithm $OPT_D$ on the right-hand side is tailored to this distribution. In other words, algorithm $A$ is simultaneously near-optimal for all distributions $D \in \mathcal{D}$, and as such is a robustly excellent solution for an unknown input distribution in $\mathcal{D}$.[1] Proposition 2.1 is an even stronger statement — it offers an instance-optimality guarantee that has meaning even if we don't impose any distributional assumptions on inputs.

The proof of Proposition 2.1 is straightforward.

*Proof of Proposition 2.1:* Fix an input distribution $D \in \mathcal{D}$. By the definition (1) of $\mathcal{C}_{\mathcal{D}}$, the optimal algorithm $OPT_D$ for $D$ is a member of $\mathcal{C}_{\mathcal{D}}$. Since $A$ is instance optimal with respect to $\mathcal{C}_{\mathcal{D}}$ with approximation factor $\alpha$,

$$\text{cost}(A, z) \leq \alpha \cdot \text{cost}(OPT_D, z)$$

for every input $z$. Taking expectations (over $z \sim D$) establishes (2), as desired. ■

The next section gives an application of this paradigm to deriving instance-optimality guarantees with respect to a restricted set of algorithms, for the problem of online decision-making. The paradigm automatically regenerates the standard "no-regret" framework for analyzing such algorithms.[2]

# 3 Online Decision-Making

## 3.1 The Model

The following online decision-making problem has been studied exhaustively (e.g. [1]). There is a finite set $A$ of actions; even the case $|A| = 2$ (e.g., buy/sell stock) is interesting. There is a known time horizon $T$. Then:

- At each time step $t = 1, 2, \ldots, T$:
  - Your algorithm picks an action $a^t$, possibly according to a probability distribution.
  - A cost vector $c^t : A \to [0, 1]$ is unveiled.

Inputs $z$ thus correspond to cost vector sequences $c^1, c^2, \ldots, c^T$. The cost of an online decision making algorithm $B$ on such a sequence is

$$\text{cost}(B, z) = \sum_{t=1}^{T} c^t(a^t).$$

[1] This implies that no algorithm, whether in $\mathcal{C}_{\mathcal{D}}$ or not, can be $\alpha$-instance optimal with respect to $\mathcal{C}_{\mathcal{D}}$ with $\alpha < 1$.

[2] Other examples where this paradigm regenerates standard optimality notions occur in data structures, such as the "static optimality" property of binary search trees (see e.g. [4]), and in prior-free auction design (see [2]).

**Remark 3.1**

1. We assume in this lecture that the time horizon $T$ is known a priori. The results in this lecture extend to the case of an unknown time horizon with a modest amount of additional work.

2. An adversary is *adaptive* if it can choose the cost vectors $c^t$ as a function of the algorithm's coin flips on days 1 through $t-1$, and *oblivious* otherwise. For simplicity, this lecture focuses on oblivious adversaries, which effectively have to choose the cost vector sequence $c^1, c^2, \ldots, c^T$ with knowledge of the code of the employed algorithm $B$ but not its coin flips. There also exist online decision-making algorithms that have good performance with respect to adaptive adversaries.

Your first reaction to the model is probably that it seems pretty unfair. Indeed, because the adversary can choose its cost vectors as a function of the employed algorithm, it is impossible for any online algorithm to compete with the best action sequence in hindsight. For example, if $|A| = 2$ and $B$ is a deterministic algorithm, then the adversary can choose either the cost vector $(1, 0)$ or the cost vector $(0, 1)$, depending on whether or not $B$ is about to choose the first or second action, respectively. (Since the adversary has the code to $B$, it knows what it will do.) The algorithm $B$ incurs cost $T$ on this input while the best action sequence in hindsight has zero cost. Even if the algorithm $B$ is randomized, an adversary can force it to incur cost expected cost at least $\frac{1}{2}$ at each time step (when $|A| = 2$), despite the fact that zero cost could have been achieved with hindsight.

## 3.2 From Distributions to a "No-Regret" Benchmark

The bad example above can be interpreted as an impossibility result for instance-optimal algorithms, where the class of "competitor algorithms" is all possible action sequences. We next apply the paradigm of a "distributional thought experiment" to define a restricted class of competitor algorithms to enable instance optimality results.

Purely as a thought experiment, imagine that each cost vector $c^t$ is an i.i.d. draw from a known distribution $D$ over such vectors. (The "i.i.d." here refers to cost vectors at different times steps; the costs of different actions can be arbitrarily distributed and correlated with each other.) In this case, the optimal algorithm is extremely simple. At a time step $t$, the optimal algorithm sets

$$a^t = \operatorname*{argmin}_{a \in A} \mathbf{E}_{c^t \sim D}\left[c^t(a)\right].$$

Since $D$ doesn't depend on $t$, neither does $a^t$. Thus,

> *if $\mathcal{D}$ is the set of all distributions on cost vectors, then the induced class $\mathcal{C}_\mathcal{D}$ of algorithms consists of the $|A|$ algorithms that always play a fixed action.*

Having defined the set $\mathcal{C}_\mathcal{D}$ of algorithms that we seek to compete with, we now dispense with any distributions and seek an input-by-input guarantee — instance optimality with respect to $\mathcal{C}_\mathcal{D}$.

## 3.3 Main Result

The main result of this lecture is an online decision-making algorithm that is "approximately instance optimal" with respect to the algorithm class $\mathcal{C}_{\mathcal{D}}$ of fixed actions with constant $\alpha \approx 1$. The exact statement differs from previous instance optimality results in that the algorithm's error will be additive, rather than relative.[3] Precisely:

**Theorem 3.2** *There is a randomized online decision-making algorithm $B$ such that, for every time horizon $T$ and cost vector sequence $c^1, c^2, \ldots, c^T$,*

$$\mathbf{E}\left[\frac{1}{T}\sum_{t=1}^{T} c^t(a^t)\right] \leq \frac{1}{T}\min_{a \in A}\sum_{t=1}^{T} c^t(a) + \underbrace{O\left(\sqrt{\frac{\log n}{T}}\right)}_{\text{"regret" of } B \text{ on } c^1,\ldots,c^T}. \tag{3}$$

On the left-hand side of (3), the expectation is over the choice of the actions $a^1, a^2, \ldots, a^T$ made by the algorithm $B$. On the right-hand side, the first term is simply the best performance achieved on the input $c^1, c^2, \ldots, c^T$ by an algorithm in our competitor class $\mathcal{C}_{\mathcal{D}}$. Note that Theorem 3.2 applies input-by-input — no distribution over inputs is assumed.

The guarantee in Theorem 3.2 implies that the expected regret of the algorithm — the extent to which its average cost (over the time steps) exceeds that of the best fixed action — is bounded above by a term that goes to 0 as the time horizon $T$ goes to infinity. For example, the expected regret is at most $\epsilon$ provided $T = \Omega(\frac{\log n}{\epsilon^2})$. The loss in Theorem 3.2 is the smallest possible, as a function of both $n$ and $T$, up to a constant factor; see also Homework #10.

There are multiple online decision-making algorithms that achieve the optimal regret bound of Theorem 3.2. Probably the most well-known one is the "Multiplicative Weights" algorithm (a.k.a. "Hedge") [3].[4] This algorithm is based on two simple principles. First, choose an action randomly but based on past performance, with actions incurring less cumulative cost so far chosen with higher probabilities. Second, aggressively decrease the probability of choosing an action once it starts performing badly. See, for example, the instructor's CS364A lecture notes on algorithmic game theory for details. Here, for variety and to highlight connections to smoothed analysis, we instead analyze a different algorithm that achieves the optimal bound of Theorem 3.2.

## 3.4 Follow the Perturbed Leader (FTPL)

The *Follow the Leader (FTL) algorithm* is the simplest implementation of decision-making via past performances. Imagine that we maintain a "leaderboard" for all actions, where the current score of an action is the cumulative cost it has incurred over all previous time steps. (Thus lower scores are better.)

[3] This is clearly necessary: take $T = 1$ and, for any online algorithm, consider the worse of the two inputs $(0, 1)$ and $(1, 0)$.

[4] This algorithm also works well against adaptive adversaries; recall Remark 3.1.

- For each $t = 1, 2, \ldots, T$:
  - Follow the leader: Set
$$a^t = \operatorname*{argmin}_{a \in A} \sum_{s<t} c^s(a),$$
    breaking ties lexicographically (say).

There exists a cost vector sequence on which the FTL algorithm has regret $\Omega(1)$, so it is not a no-regret algorithm in the sense of Theorem 3.2. (Nor is any other deterministic algorithm; see Homework #10). The FTL algorithm does have vanishing expected regret on smoothed (i.e., perturbed) instances, however; see Homework #10.

We thus need randomization to achieve a no-regret guarantee. The *Follow the Perturbed Leader (FTPL)* algorithm injects randomness into the FTL algorithm in a very simple way:

- For each action $a \in A$, let $X_a$ denote an independent random variable, distributed geometrically with a parameter $\epsilon$ to be determined later.[5]
- For each $t = 1, 2, \ldots, T$:
  - Follow the (perturbed) leader: Set
$$a^t = \operatorname*{argmin}_{a \in A} \left( -X_a + \sum_{s<t} c^s(a) \right),$$
    breaking ties lexicographically (say).

In other words, the FTPL algorithm begins with a preprocessing step that awards each action a random "bonus," and then runs the FTL algorithm with respect to these fictitious bonuses and the actual cost vectors.

## 3.5 Analysis of FTPL

We now show that the FTPL algorithm meets the guarantee in Theorem 3.2. Fix an arbitrary sequence $c^1, c^2, \ldots, c^T$ of cost vectors. The key idea is to define, for the analysis only, a fictitious algorithm $F$ that on day $t$ chooses the action

$$f^t = \operatorname*{argmin}_{a \in A} \left( -X_a + \sum_{s \leq t} c^s(a) \right),$$

where the $X_a$'s are the same random bonuses used by the FTPL algorithm. The only difference between $F$ and the FTPL algorithm is that the former telepathically also uses the cost

[5] That is: flip a coin and count the number of flips until you get "heads," where $\epsilon$ is the probability of a "heads." Note $\mathbf{E}[X_a] = \frac{1}{\epsilon}$.

vector $c^t$ of the current time step in its selection of $a^t$, while FTPL (as an online algorithm) does not. While $F$ is not an online algorithm, it is nevertheless a useful intermediary in the analysis of FTPL.

We break the proof of Theorem 3.2 into three claims.

1. With probability 1 (over the $X_a$'s),

$$\underbrace{\frac{1}{T}\sum_{t=1}^{T} c^t(f^t) - \frac{1}{T}\min_{a\in A}\sum_{t=1}^{T} c^t(a)}_{\text{regret of } F} \leq \frac{1}{T}\max_{a\in A} X_a. \tag{4}$$

2. The expected regret of the FTPL algorithm is at most $\epsilon$ more than that of $F$. (The expectations are over the $X_a$'s.)

3. $\mathbf{E}[\max_{a\in A} X_a] = O(\frac{\log n}{\epsilon})$.

Observe that the three claims imply that the expected regret of the FTPL algorithm is at most $\epsilon + O(\frac{\log n}{\epsilon T})$. Judiciously choosing $\epsilon = \sqrt{(\log n)/T}$ yields the expected regret bound of $O(\sqrt{(\log n)/T})$ that is claimed in Theorem 3.2.

We now proceed to the three claims, and treat them in reverse order. The third claim is a simple probability calculation, and we leave it to Homework #10. The other two claims have clever proofs.

To prove the second claim, fix a time step $t \in \{1, 2, \ldots, T\}$. We show that, if the same random $X_a$'s are used in both the FTPL algorithm and in algorithm $F$, then the probability (over the choice of $X_a$'s) that the two algorithms take different actions at time $t$ is at most $\epsilon$. Integrating out over the $X_a$'s and using that $c^t(a) \in [0, 1]$ for every $t$ and $a \in A$, this implies that the difference between the expected cost of FTPL and that of $F$ at time $t$ is at most $\epsilon$. Averaging over the $T$ time steps and using linearity of expectation then yields the second claim.

So why are FTPL and $F$ likely to take a common action at time $t$? First, for fixed $X_a$'s, observe that the action taken by FTPL at time $t$ is exactly the action that the fictitious algorithm $F$ already took at time $t-1$ (namely, the action minimizing $-X_a + \sum_{s=1}^{t-1} c^s(A)$). Thus, an equivalent assertion is that, for every fixed $t > 1$, the probability (over the $X_a$'s) that algorithm $F$ plays different actions at times $t-1$ and $t$ is at most $\epsilon$. Second, an even stronger assertion is that the probability (over the $X_a$'s) that the smallest value of $-X_a + \sum_{s=1}^{t-1} c^s(a)$ is within 1 of the second-smallest such value is at most $\epsilon$ — since all costs are between 0 and 1, if the perturbed leader at time $t-1$ has outdistanced the nearest competitor by more than 1 so far, it cannot be dislodged from its perch at time $t$, no matter what happens at time $t-1$. To see why *this* statement holds, we use the Principle of Deferred Decisions, flipping coins lazily on a "need to know" basis.

In more detail, fix $t > 1$ and zoom in on time step $t-1$. Which action will algorithm $F$ choose at this time step? It depends, of course, on the random $X_a$'s. Recall that each random bonus $X_a$ is an independent random variable distributed according to the number

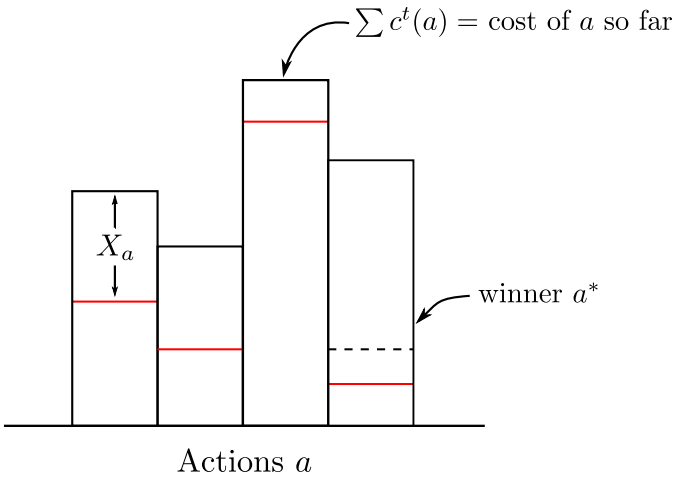


Figure 1: The gap between the score of $a^*$ and the second lowest score (dashed line) is at least 1 with probability $1 - \epsilon$.

of coin flips needed to get "heads," where the probability of "heads" is $\epsilon$. We begin with $X_a = 1$ for every $a \in A$ and compute $-X_a + \sum_{s=1}^{t-1} c^s(a)$ for each action. We then a flip a coin for the currently worst action $a$ (the one with the highest score).[6] If it comes up "heads" then the current value of $X_a$ is also its final value. Since the other actions will only have their scores decreased by future coin flips, there is no chance that $F$ will play $a$ and we can discard it for the rest of the random experiment. If the coin comes up "tails," then we increment $a$'s random bonus by 1, thereby decreasing its score by 1. Action $a$ might still be the action with the worst (i.e., highest) score, or this score decrement may have moved it up the leaderboard. In any case, at each iteration of this random experiment, we flip exactly one coin, for the remaining action with the highest score, with the action discarded if the coin comes up "heads."

Here is the key point. Consider the iteration of the random experiment that eliminates the second-to-last action, leaving only the action $a^*$. At this point, we know with certainty that algorithm $F$ will play the action $a^*$ at time $t-1$. But we haven't yet finished determining the precise value of the random bonus $X_{a^*}$! So we flip another coin for $a^*$. Since $a^*$ already has the lowest score, *if this coin comes up tails, then $a^*$'s score will be strictly more than one less than its nearest competitor*. See Figure 1. This happens with probability $1 - \epsilon$, and the proof of the second claim is complete.

To prove the first claim (4) and complete the proof of Theorem 3.2, fix values for the $X_a$'s. Let $a^1, a^2, \ldots, a^T$ be the actions chosen by algorithm $F$ and $a^*$ be the best fixed action. Multiply the desired inequality (4) through by $T$. As a warm up, we first assume that $X_a = 0$ for every $a \in A$. Inequality (4) asserts that, in this case, algorithm $F$ is at least as good as

[6] We leave the issue of handling ties to the interested reader.

the fixed action $a^*$. To see this, write

$$\begin{aligned}
\sum_{t=1}^{T} c^t(a^*) &\geq \sum_{t=1}^{T} c^t(a^T) && (5)\\
&= c^T(a^T) + \sum_{t=1}^{T-1} c^t(a^T)\\
&\geq c^T(a^T) + \sum_{t=1}^{T-1} c^t(a^{T-1}) && (6)\\
&= c^{T-1}(a^{T-1}) + c^T(a^T) + \sum_{t=1}^{T-2} c^t(a^{T-1})\\
&\geq c^{T-1}(a^{T-1}) + c^T(a^T) + \sum_{t=1}^{T-2} c^t(a^{T-2}) && (7)\\
&\quad \cdots\\
&\geq c^1(a^1) + \cdots + c^{T-1}(a^{T-1}) + c^T(a^T)
\end{aligned}$$

where (5) follows from the definition $a^T = \text{argmin}_{a\in A}(-X_a + \sum_{s\leq T} c^s(A))$,[7] inequality (6) follows from the definition $a^{T-1} = \text{argmin}_{a\in A}(-X_a + \sum_{s\leq T-1} c^s(a))$, inequality (7) follows from the definition $a^{T-2} = \text{argmin}_{a\in A}(-X_a + \sum_{s\leq T-2} c^s(a))$, and so on. This completes the proof in the special case where $X_a = 0$ for every $a \in A$. More generally, the upper bound above loses an additional term of $(X_{a^T} - X_{a^*})$ in (5), an additional term of $(X_{a^{T-1}} - X_{a^T})$ in (6), an additional term of $(X_{a^{T-2}} - X_{a^{T-1}})$ in (7), and so on. In total, these extra terms increase the upper bound by $X_{a^1} - X_{a^*} \leq X_{a^1} \leq \max_{a\in A} X_a$. This completes the proof of the first claim and of the theorem.

# References


[1] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[2] J. D. Hartline and T. Roughgarden. Optimal mechanism design and money burning. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 75–84, 2008. Full version last revised September, 2014.

[3] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[4] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.


[7] When $X_a = 0$ for all $a$, the definitions of $a^*$ and $a^T$ coincide and so equality holds.