CS369N: Beyond Worst-Case Analysis
Lecture #9: From Average-Case Analysis to Instance Optimality*

Tim Roughgarden†

December 4, 2009

1 Optimal Algorithms and Instance Optimality

Recall from Lecture #1 that an algorithm A is *α-instance optimal* if for *every* algorithm A *and* input z,

$$\text{cost}(A, z) \leq \alpha \cdot \text{cost}(B, z). \tag{1}$$

This is the strongest-possible optimality notion for an algorithm (assuming that α is small) and, as such, for many problems instance optimal algorithms do not exist (recall HW #1).

There are several more attainable versions of instance optimality that remain meaningful (recall also Lecture #1). Today we investigate restricting the algorithm B on the right-hand side of (1) to lie in some restricted class $\mathcal{C}$ of algorithms. (The algorithm A will not be so restricted.)

To some extent, we already took this approach in Lecture #1. When we proved the instance optimality of the Threshold Algorithm, we compared it only to other algorithms that use random access only on objects that it already discovered via sequential search (i.e., an algorithm can not "guess" the existence of an unseen object). In the 2-D Maxima problem, we used a definition that was intuitively trying to restrict attention to algorithms whose running time is independent of the order in which the point set was listed (again, to avoid algorithms that "memorize an answer").[1]

The obvious question is: *what is an appropriate choice of $\mathcal{C}$?* If we take $\mathcal{C}$ to be too big (e.g., all algorithms) then we're probably going to be stuck with negative results saying that no algorithm is instance optimal. If we take $\mathcal{C}$ too small, then an instance optimality

†Department of Computer Science, Stanford University, 462 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

[1]Recall that idea was implemented not by restricting the class of allowable algorithms, but by evaluating the performance of an algorithm on a given point set via its worst-case performance on an ordering of the points.

guarantee might not be very impressive. Ideally, we want to choose $\mathcal{C}$ so that at least one but very few algorithms are instance optimal.

The plan for today is to follow a principled, two-step approach for identifying classes $\mathcal{C}$ of algorithms.

1. Choose a set $\mathcal{D}$ of distributions over inputs, which ideally is as rich as possible.[2] '
2. Define
$$\mathcal{C}_{\mathcal{D}} := \{\text{algorithms } A \,:\, A \text{ optimal for some } D \in \mathcal{D},\}\,. \tag{2}$$
where an optimal algorithm for a distribution D the one that minimizes expected cost $\mathbf{E}_{z\sim D}[\text{cost}(A,d)]$.

An easy fact is that if A is approximately instance optimal with respect to the set $\mathcal{C}_{\mathcal{D}}$, then A is simultaneously has near-optimal expected cost with respect to every distribution $D \in \mathcal{D}$.

Proposition 1.1 *If A is α-instance optimal with respect to $\mathcal{C}_D$, then*
$$\mathbf{E}_{z\sim D}[cost(A,z)] \leq \alpha \cdot \mathbf{E}_{z\sim D}[cost(OPT_D,z)] \tag{3}$$
for every $D \in \mathcal{D}$, where OPT_D denotes the optimal algorithm for the distribution D.

The guarantee in Proposition 1.1 is interesting because the algorithm OPT_D on the right-hand side of (3) is carefully tailored to the distribution D, while the algorithm A on the left-hand side is defined independently of D.[3]

Proof of Proposition 1.1: Fix $D \in \mathcal{D}$. By the definition of $\mathcal{C}_{\mathcal{D}}$, $OPT_D \in \mathcal{C}_{\mathcal{D}}$. By the definition of instance optimality, $\text{cost}(A,z) \leq \alpha \cdot \text{cost}(OPT_D,z)$ for every input z. Taking expectations completes the proof. ■

We emphasize the being α-instance optimal with respect to $\mathcal{C}_{\mathcal{D}}$ is strictly stronger than the guarantee in Proposition 1.1, since it applies input-by-input rather than merely to expectations. Also, it is important to understand that distributions over inputs are used only to define the set $\mathcal{C}_{\mathcal{D}}$, and thereafter the goal is a distribution-free (input-by-input) guarantee.

2 Online Decision Making

2.1 The Setup

Consider the following rather unfair game, which takes place over T rounds. At each round $t = 12,\ldots,T$:

[2]Examples from previous lectures include Markov paging (where $\mathcal{D}$ is the distributions generated by Markov Chains), diffuse adversaries and block sources (where $\mathcal{D}$ is the "sufficiently random/uniform" distributions), and self-improving algorithms (where $\mathcal{D}$ is the distributions with independent components).

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

1. you probabilistically choose a strategy s_t from a fixed set S of strategies.

2. an adversary, who knows your probability distribution but not your actual strategy choice, chooses a cost vector $c^t : S \to [0, 1]$.

The cost $\text{cost}(A, c)$ of an online algorithm A for playing this game is defined in the expected value (of A's random coin flips) of $\sum_{t=1}^{T} c^t(s^t)$.

An easy fact is that no online algorithm is instance optimal against all online algorithms. The problem again is algorithms that "memorize the input". In more detail, fix an online algorithm A. If an adversary randomizes between the cost vectors $(1, 0)$ and $(0, 1)$ at every round, then the expected cost incurred by A is $T/2$. This implies that there exists a fixed input z such that $\text{cost}(A, z) \geq T/2$. On the other hand, there exists an online algorithm B with $\text{cost}(B, z) = 0$ (where B just correctly guesses the zero-cost strategy at every time step).

2.2 From Average-Case Analysis to Instance Optimality

The unnaturalness of the algorithm B above suggests looking for an instance optimality guarantee with respect to a smaller set of "reasonable algorithms". We use the two-step approach from Section 1 for this purpose.

The first step is a thought experiment: if the input (i.e., the cost vectors) where generated by a probability distribution, then what would we do? We consider the simple case where each vector c^t is an IID draw from a known distribution D. (There can be arbitrary dependence between the cost of different strategies, but there is no dependence between the the cost vectors at different rounds.) In the case, the optimal online algorithm is obvious: to minimize the expected cost, just set

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

Observe that since D is independent of t, the optimal choice of s^t is time-invariant. Thus, ranging over all such distributions D: *if $\mathcal{D}$ is the set of IID cost vector distributions, then $\mathcal{C}_{\mathcal{D}}$ is the set of constant (time-invariant) online algorithms.*

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 — α-instance optimality with respect to $\mathcal{C}_{\mathcal{D}}$. The main result of this section is a classic one.

Theorem 2.1 ([?]) *There is a randomized online algorithm for online decision-making that is α-instance optimal with respect to the above set $\mathcal{C}_{\mathcal{D}}$, where α is essentially 1.*

The meaning of "essentially 1" will be clear at the end of the proof. Theorem 2.1 has been rediscovered numerous times over the years in various communities. In the vocabulary of learning theory, it establishes the existence of a *no (external) regret algorithm*.

2.3 The Algorithm

The algorithm is simple and based on two principles. First, a strategy should be chosen randomly with the probabilities based on their past performances. (Note that since the input is arbitrary and not generated by a distribution, it's not obvious that this is the right thing to do.) Second, the probability assigned to poorly performing actions should be decreased aggressively (i.e., exponentially). The precise algorithm is as follows. Note that

---

1. Initialize $w_s = 1$ for every $s \in S$.
2. For each $t = 1, 2, \ldots, T$:
   (a) Choose strategy s with probability proportional to the weights: with probability w_s^t / W^t, where $W^t = \sum_s w_s^t$.
   (b) After seeing the cost vector c^t, update $w_s^{t+1} = w_s^t (1-\epsilon)^{c^t(s)}$, where $\epsilon > 0$ is a small (subconstant) parameter whose value we'll fix later.

Figure 1: How to make decisions on-line.

---

the weights of strategies only decrease. For example, if strategy s incurs zero cost at some time step, its weight stays the same; if it incurs unit cost, its weight is multiplied by a $(1-\epsilon)$ factor.

2.4 The Analysis

The high-level idea of the analysis is to use the total weight W^t of all strategies as an intermediary to relate the two quantities we care about, the cost of the best time-invariant strategy and the cost of our algorithm.

1. We can lower bound W^T as a function of the cost incurred by the best time-invariant strategy — if a good such strategy exists, it single-handedly provides a good lower bound on W^T.
2. The total weight W^T decreases exponentially fast with the cost incurred by our algorithm.

After these two ideas are formalized, the proof follows by a suitable rearrangement of terms.

Proof of Theorem 2.1: Consider an input $c^1, \ldots, c^T$. Let s^* denote the optimal time-invariant strategy (i.e., the smallest cost incurred by an algorithm in $\mathcal{C}_\mathcal{D}$. Define

$$OPT = \sum_{t=1}^{T} c^t(S^*).$$

Then,

$$W^T \geq w_{s^*}^T = w_{s^*}^1 \prod_{t=1}^{T} (1-\epsilon)^{c^t(s^*)} = (1-\epsilon)^{OPT}. \tag{4}$$

On the other hand, for every t, we can track the contraction of W^t in terms of the expected cost of our algorithm:

$$W^{t+1} = \sum_{s \in S} w_s^{t+1} = \sum_{s \in S} w_s^t (1-\epsilon)^{c^t(S)} \leq \sum_{s \in S} w_s^t (1 - \epsilon c^t(s)) = W^t \left(1 - \epsilon \gamma_t\right), \tag{5}$$

where the inequality holds because $\epsilon \in [0, \frac{1}{2}]$ and $c^t(s) \in [0,1]$ (as you are invited to check) and where γ_t is defined to make the last equality hold, as

$$\gamma_t = \sum_{s \in S} c^t(S) \cdot \frac{w_s^t}{W^t}.$$

Observe that γ_t is more famously known as the expected cost incurred by our algorithm at time t.

Iterative (5) for all t and combining the result with (4) yields

$$(1-\epsilon)^{OPT} \leq W^T \leq \underbrace{n}_{=W^1} \prod_{t=1}^{T} (1 - \epsilon \gamma_t).$$

This inequality is a good start because it relates the two quantities that we care about, OPT and our expected cost $\sum_t \gamma_t$. To relate them more directly, we take logarithms to get

$$OPT \ln(1-\epsilon) \leq \ln n + \sum_{t=1}^{T} \ln(1 - \epsilon \gamma_t).$$

To extract the γ_t's, we apply the Taylor series

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$$

to the right-hand side with $x = -\epsilon\gamma_t$ (and, while we're at it, to the left-hand side too). Since $\epsilon \in [0, \frac{1}{2}]$ and $\gamma_t \in [0,1]$, we can get away with throwing out all but the first term on the right-hand side (yielding an overestimate) and by doubling the second term on the left-hand side (yielding an underestimate of $\ln(1-\epsilon)$):

$$OPT(-\epsilon - \epsilon^2) \leq \ln n + \sum_{t=1}^{T} (-\epsilon \gamma_t)$$

and hence

$$\sum_{t=1}^{T} \gamma_t \leq OPT \cdot \left(1 + \epsilon + \frac{\ln n}{\epsilon OPT}\right). \tag{6}$$

The traditional interpretation of the guarantee (6) is that the per-round additive loss ($\epsilon OPT + \epsilon^{-1} \ln n)/T$ tends to 0 as $T \to \infty$, provided ϵ is chosen appropriately (a natural choice is $\epsilon = \sqrt{(\ln n)/T}$, to equalize the two additive loss terms). If we want to think in terms of a relative approximation of cost, as in α-instance optimality, then the simplest interpretation is: as OPT grows large (asymptotically larger than $\ln n$ we can take ϵ to be subconstant and hence the relative approximation factor in (6) is $1 + o(1)$. ■

3 Auctions for Digital Goods

3.1 The Setup

Our second example is an instance optimal guarantee for revenue-maximizing auctions. The simple setup is: there is a single type of good with can be produced in unlimited supply at zero cost (e.g., a mp3 that you want to sell). There are n potential buyers, each with a *valuation* v_i that represents i's maximum willingness to pay for the good.

Our "computational model" will be called the *truthful auctions*. If you haven't seem them before, the model might seem a bit unnatural and limited. Nevertheless, truthful auctions are a good abstraction of arbitrary selling procedures (as can be justified formally, though we won't do so here).

Formally a truthful auction has the following form: separately for each buyer $i = 1, 2, \ldots, n$:

1. Using only v_{-i} (the valuations other than v_i), formulate a "take-it-or-leave-it" offer price $p_i(v_{-i})$ for i (or, a probability distribution over such prices).

2. The revenue obtained from i is defined as $p_i(v_{-i})$ if $p_i(v_{-i}) \leq v_i$ (since i accepts the offer) and 0 otherwise (since i declines the offer).

A digression: the motivation for forcing the offer price to i to be independent of i's valuation is that typically the v_i is a priori known only to the potential buyer and not to the seller. The seller then has no choice but to (implicitly or explicitly) solicit a bid — which is then used as a proxy for v_i — from the potential buyer (e.g., you do this when you type a bid into eBay). If p_i was allowed to depend on v_i, this would encourage bidders to manipulate the system — this is generally undesirable and, in the present context, doesn't increase the maximum-obtainable revenue anyways.

For example, the *Vickrey auction* is defined as follows: it sells only one item, the buyer with the highest valuation, and charges the value of the second-highest valuation. As you should verify, this corresponds to the pricing functions $p_i(v_{-i}) = \max_{j \neq i} v_j$ for each i. If you think about eBay, it is precisely an implementation of the Vickrey auction — in particular, if you win, the price you pay is governed by the next-highest bid and not by the value of your own bid.

Of course, with an unlimited supply of goods, one wouldn't use the Vickrey auction. So which truthful auction is the best for revenue-maximization? As usual, the strongest-possible argument would exhibit an auction A that is α-instance optimal (with α small) with

respect to all truthful auctions. As in the online decision-making problem, this guarantee is simply not possible. The first observation is that, for every input v, there is a truthful auction A that obtains the full revenue $\sum_{i=1}^{n} v_i$ (A "memorized the input" and sets $p_i(\cdot)$ to be the constant function v_i). The second observation is that no (possibility randomized) truthful auction always earns revenue at least a constant fraction of $\sum_{i=1}^{n} v_i$. At the risk of oversimplifying things, the basic intuition is that $p_i(v_{-i})$ is essentially a guess about what v_i is — and v_i could be anything, from 1, to a million, to a billion, and so on. Thus any (even randomized) guess will perform poorly for some values of v_i.

3.2 From Average-Case Analysis to Instance Optimality

Again, the use of input-memorizing algorithms in the above discussion motivates proving an instance optimality guarantee with respect to a smaller set algorithms that is generated using the paradigm in Section 1.

Recall that we start with a thought experiment: if the input (i.e., the valuations) is drawn from a probability distribution, what would we do? We consider the simple case where each valuation v_i is an IID draw from a known distribution D. Under this assumption, the expected revenue-maximizing truthful auction is easy to characterize: for each buyer i, choose

$$p_i(v_{-i}) = \operatorname*{argmax}_p \underbrace{p}_{\text{revenue from sale}} \cdot \underbrace{(1 - F(p))}_{\text{probability of sale}},$$

where F is the distribution function of D.

Observe that this offer price is independent of v_{-i} (because of the independence of the v_i's) and of i (because buyers' valuations are identically distributed) Thus, ranging over all such distributions D: *if $\mathcal{D}$ is the set of IID valuation distributions, then $\mathcal{C}_{\mathcal{D}}$ is the set of auctions that always offer all buyers some fixed and common offer price p^*.*

Now that the relevant set $\mathcal{C}_{\mathcal{D}}$ has been defined, we forget about distributions and look for an instance optimality guarantee. A simple observation: an auction A is α-instance optimal with respect to $\mathcal{C}_{\mathcal{D}}$ if and only if for all sorted inputs v (with $v_1 \geq v_2 \geq \cdots \geq v_n$), the expected revenue of A is at least

$$\frac{1}{\alpha} \max_{1 \leq i \leq n} i \cdot v_i. \tag{7}$$

Restricting attention to sorted vectors is without loss (by renaming the bidders) since we only use auctions that are symmetric, meaning that the expected revenue is invariant under permutations of an input. For a sorted vector, we claim the maximum in (7) is the maximum revenue obtainable by a common offer price (i.e., by an auction of $\mathcal{C}_{\mathcal{D}}$ on the input v) — this is because, for each i, among all common offer prices p that sell to exactly i bidders, v_i generates the most revenue (namely $i \cdot v_i$).

We prove a result that is a little weaker than instance optimality.

Theorem 3.1 ([?]) *There is a randomized auction A such that, for every sorted input v, the expected revenue of A is at least*

$$\frac{1}{4} \max_{2 \le i \le n} i \cdot v_i. \tag{8}$$

Comparing (7) and (8), we see that Theorem 3.1 is 4-instance optimal only on the subset of inputs for which $\text{argmax}_{1 \le i \le n} i v_i \ge 2$. Note that the inputs which fail to satisfy this property are those with a single bidder with an extremely high valuation compared to the others. This deficiency is necessary, more or the less for the same reason it's impossible to always obtain revenue at least a constant fraction of $\sum_{i=1}^{n} v_i$ (even if you know that there is a single very wealthy buyer and you who it is, you still don't know what price to offer them). Fortunately, inputs with $\text{argmax}_{1 \le i \le n} i v_i = 1$ don't seem very relevant in theory or in practice. For example, if the valuations are IID draws from a distribution that satisfies some mild conditions, then almost all of the expected revenue will come from inputs that satisfy $\text{argmax}_{1 \le i \le n} i v_i \ge 2$ anyway — in this case, the inputs for which Theorem 3.1 doesn't apply are of little importance.

3.3 The Algorithm

The key subroutine in the auction of Theorem 3.1 is called a *Profit Extractor*. It takes as input valuations v and a revenue target R. The subroutine tries to find a set S of buyers are willing to pay at least $R/|S|$, starting from the set of all potential buyers and removing cheapskates as necessary.

Input: valuations v and a revenue target R.

1. Initialize S to be the set of all potential buyers.
2. While there is a buyer $i \in S$ for which $v_i < R/|S|$, delete an arbitrary such bidder.
3. Return the final set S of winners, and (if $S \neq$) charge each of them $R/|S|$.

Figure 2: The Profit Extract subroutine.

This subroutine can be implemented as a truthful auction by setting $p_i(v_{-i})$ as follows: imagine, as a thought experiment, resetting v_i to $+\infty$ and running ProfitExtract on v. It halts with at least one winner (i, if no one else) and charges all winners S some price $R/|S|$. Setting $p_i(v_{-i})$ to this value of $R/|S|$ for each i is the same as running the above 3-step auction (as you should verify). Note that $p_i(v_{-i})$ is independent of v_i in this definition, as required.

The point of the Profit Extractor is to reduce revenue maximization to a decision version, namely whether or not raising a given amount of revenue R is possible. The subroutine solves this decision problem in the following sense.

Lemma 3.2 *If there is a common offer price that extracts revenue R from the sorted input v (i.e., $\max_{i \le i \le n} i \cdot v_i \ge R$), then ProfitExtract will successfully compute such a price (and halt with $S \neq \emptyset$).*

Proof: Assume that $i \cdot v_i \ge R$ with T being the top i bidders, so $v_j \ge R/i$ for every $j \in T$. Initially, $T \subseteq S$. By induction on the number of iterations, $T \subseteq S$ at the end of the algorithm (since $|S| \ge |T| = i$, every buyer $j \in T$ is only asked to pay prices at most $R/|S| \le R/i \le v_j$). ■

So, if we can identify a suitable revenue target R, then we are done because of the Profit Extractor. You might wonder why we don't just set R to be the maximum revenue obtainable from all bidders and then run the Profit Extractor. The reason is that this is not a truthful auction — the value R is now a function of every valuation v_i, and the prices offered by the Profit Extractor then inherit this dependence. Instead, we need to make sure every buyer participates in a Profit Extractor while wielding no influence over the corresponding revenue target. Perhaps the simplest implementation of this idea is the *Random Sampling Profit Extractor (RSPE)* auction.

Input: valuations v.

1. Randomly partition the valuations v into two groups, x and y. (Flip an independent and fair coin for each of the potential buyers.)
2. Set R_1 to be the maximum revenue obtainable from the valuations x via common offer price.
3. Set R_2 to be the maximum revenue obtainable from the valuations y via common offer price.
4. ProfitExtract(x, R_2).
5. ProfitExtract(y, R_1).

Figure 3: The RSPE auction.

This is a truthful auction because, for each i, the valuation v_i does not affect the revenue target for the group to which i belongs, and because for a fixed revenue target the price offered to i by ProfitExtract is independent of v_i.

The next example shows that the RSPE auction is no better than 4-instance optimal, even when restricting to sorted inputs v for which $\text{argmax}_{1 \le i \le n} iv_i \ge 2$.

Example 3.3 Suppose $n = 2$, $v_1 = 1$, and $v_2 = \frac{1}{2}$. Note that $\max_{2 \le i \le n} iv_i = 1$. With 50% probability, the RSPE auction will put two bidders in one group and zero in the other

— in this case it will try to ProfitExtract 0 from the two bidders (successfully, for what its worth) and 1 from the empty group (unsuccessfully), for a revenue of 0. With 50% probability, the RSPE auction will split the two bidders into different groups — then, it will successfully extract revenue $\frac{1}{2}$ from the first bidder and will unsuccessfully try to extract 1 from the second bidder. In this case its revenue is $\frac{1}{2}$. Overall, its expected revenue is $\frac{1}{4} = \frac{1}{4} \cdot \max_{2 \le i \le n} iv_i$.

3.4 The Analysis

We now prove Theorem 3.1. We start with an easy lemma.

Lemma 3.4 *With probability 1, the revenue of the RSPE auction is at least* $\min\{R_1, R_2\}$.

Proof: Lemma 3.2 and the definition of the RSPE auction imply that Whichever of R_1, R_2 is smaller will be successfully extracted from the opposite group of valuations. ■

Thus the key lemma, which completes the proof of Theorem 3.1, is the following.

Lemma 3.5 *For every sorted input* v,

$$\mathbf{E}_{\textit{splits } (x,y)}[\min\{R_1, R_2\}] \ge \frac{1}{4} \cdot \left[\min_{2 \le i \le n} i \cdot v_i\right].$$

Proof:, Fix a sorted input v, and let $i = \text{argmax}_{2 \le j \le n}\, j \cdot v_j$. Let i_1 and i_2 denote the number of the top i bidders that are in x and y, respectively (so $i_1 + i_2 = i$ with probability 1). Now, v_i is one candidate common offer price for both x and y, and it will generate revenue $i_1 \cdot v_i$ and $i_2 \cdot v_i$, respectively. Since R_1, R_2 are defined using the *best* common offer price for each group, we have

$$\min\{R_1, R_2\} \ge v_i \cdot \min\{i_1, i_2\}.$$

Thus, we only need to prove that

$$\mathbf{E}_{\text{splits}}[\min\{i_1, i_2\}] \ge \frac{i}{4} \tag{9}$$

for every $i \ge 2$. (Inequality (9) would certainly be false when $i = 1$, since the left-hand side would be 0.) Note that, combinatorially, inequality (9) is asking about throwing i balls into two urns, and claim that the split is balanced 25%-75% or better on average.

We warm up with a little case analysis. When $i = 2$, $\min\{i_1, i_2\}$ is either 0 or 1, with 50/50 probability. So the left-hand side of (9) is $\frac{1}{4}$, which is the same as the right-hand side in this case. (Cf., Example 3.3.) When $i = 3$, $\min\{i_1, i_2\}$ is again either 0 or 1, now with 25/75 probability. The left-hand side of (9) is now $\frac{3}{4}$, as is the right-hand side in this case.

For a general number $i \ge 2$ of balls, we first throw in either 2 or 3 balls, depending on whether or not i is even or odd, respectively. We know (9) holds at this point. Now we throw in balls a pair at a time. Each time, i goes up by 2 and the right-hand side of (9) goes up by $\frac{1}{2}$. Each time, there is a 50% chance that the two balls go into different urns, in

which case the minimum population in an urn goes up by 1. Thus, the left-hand side goes up by at least $\frac{1}{2}$ each time. The proof (of Lemma 3.5 and hence Theorem 3.1) is complete. ■