CS264: Beyond Worst-Case Analysis Lecture #16: The Random-Order Model for Online Algorithms[*]

Tim Roughgarden[†]

March 2, 2017

1 Preamble

The remaining five lectures will all be about "hybrid" analysis frameworks, meaning frameworks that blend aspects of worst-case and average-case analysis. We've already seen one example, in Lectures #11 and 12: semi-random models for planted bisection and planted clique. (Recall that "nature" goes first and picks a random instance, and then a "monotone adversary" gets to tinker with the graph before the algorithm sees it.) We'll see several more. Today, we'll look at a semi-random-type model that is particularly well suited for going beyond the worst-case analysis of online algorithms. Recall that an online algorithm is presented with its input piece-by-piece, and has to make an irrevocable decision each time it gets a new pieces of the input (like in the online paging problem from Lectures #3 and #4).

2 Secretary Problems

2.1 The Random Permutation Model

In the *random permutation model*, an adversary chooses an arbitrary (worst-case) input. Then, nature presents the input one piece at a time, in an ordering that is chosen uniformly at random.[1] The goal is to design algorithms that work well no matter what input the adversary chooses (provided it is subsequently randomly ordered). Like other semi-random-type models, this model is appealingly agnostic about the details of the input distribution.

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

[1]In many applications, it's possible to replace this uniformity assumption with the assumption that the ordering is "sufficiently random" in some sense.

Note that the order of the adversary and of nature is reversed compared to our semi-random models for planted bisection and clique.

2.2 The Secretary Problem

The canonical problem in the random permutation model is the *secretary problem*.[2] Probably you've heard of it: you are presented with n numbers one-by-one (e.g., the quality of a candidate you just interviewed), and at each time step you can either accept the current number (hire the current interviewee) or continue to the next one. The goal is to accept the highest number in the sequence, even though you have no idea what the overall set of numbers is. Perhaps this seems unfair—there's only one correct choice out of n options, and intuition may suggest that your success probability inevitably goes to 0 as $n \to \infty$. But this is not the case! This trick is to use a "look than leap" strategy, where you passively monitor the first several numbers to get calibrated, and then go for it the next time you have an opportunity.

Specifically, when the numbers are adversarially chosen but then randomly ordered, it is easy to stop on the largest number with 25% probability: watch the first $n/2$ numbers go by, and stop on the first subsequent number that exceeds them all (if any). As long as the highest number is in the second half of the sequence and the second-highest number is in the first half of the sequence (with $\geq 25\%$ probability), you are guaranteed to stop on the highest number. An optimized version of this algorithm (which looks at the first $1/e$ fraction of the elements) and analysis yields a success probability of $\approx \frac{1}{e} \approx 36.7\%$ (see e.g. Wikipedia), thus yielding the "37% rule" [**?**].

2.3 The Matroid Secretary Problem[3]

It is fun to muse about generalizations of the secretary problem, for example where the algorithm can accept more than a single number of the input sequence. An abstraction that captures many interesting generalizations is the *matroid secretary problem* [1]. There is a universe U of secretaries, each with a value, and a set $\mathcal{F} \subseteq 2^U$ of *feasible sets*. Both U and $\mathcal{F}$ are known to the algorithm in advance. The values $\{v_i\}_{i \in U}$ of the secretaries are unknown a priori, and are presented to the algorithm in random order. Each time an algorithm sees a new secretary, it must decide irrevocably whether or not to accept it. The algorithm must terminate with a set S of secretaries that is a feasible set — i.e., with $S \in \mathcal{F}$. The goal is to design an algorithm that, for every set of values $\{v_i\}_{i \in U}$, outputs a set that is feasible (with probability 1 over the random order of U), and has expected total value as close as possible to — ideally, within a constant factor of — the maximum total value of a feasible set.

[2]It's amusing to see how motivating anecdotes for this problem changed over the years (it's a very old problem). In one era it was a woman trying to choose among male suitors; in another is was a driver searching for a parking space. The of-the-time name "secretary problem" first appeared in print in the early 1960s, and the name stuck.

[3]We did not cover this section in lecture.

For example, the standard secretary problem can be captured by this general framework by taking $\mathcal{F}$ to be the singleton sets plus the empty set. In the natural generalization where the algorithm can hire up to k secretaries, $\mathcal{F}$ consists of all subsets of U that have cardinality at most k. It is not too difficult to extend the guarantee of $1/e$ for the secretary problem to the k-secretary problem for all k; see Homework #10 for details.

The matroid secretary problem imposes two restrictions on the feasible sets $\mathcal{F}$. First, $\mathcal{F}$ should be *downward-closed*, meaning that $S \in \mathcal{F}$ and $T \subseteq S$ implies that $T \in \mathcal{F}$ as well. This assumption obviously holds in most natural generalizations of the secretary problem, including the k-secretary problem. The second restriction is called the *exchange property*, and it insists that whenever $S, T \in \mathcal{F}$ with $|T| < |S|$, there exists $i \in S \setminus T$ such that $T \cup \{i\} \in \mathcal{F}$. This assumption also clearly holds for the k-secretary problem: if S includes more secretaries than T and both contain at most k secretaries, then adding any secretary of $S \setminus T$ to T increases the size of T by one without violating feasibility. Thus, the k-secretary problem is a special case of the matroid secretary problem.

There are several other natural special cases of the matroid secretary problem. In the *graphical secretary problem*, U is the edge set of an undirected graph $G = (V, U)$, and a subset $F \subseteq U$ is feasible (i.e., in $\mathcal{F}$) if and only if (V, F) is an acyclic graph. It is clear that such an $\mathcal{F}$ is downward closed; a simple argument shows that the exchange property also holds. There is an algorithm for the graphical secretary problem that guarantees expected value at least a constant fraction of the maximum possible (a good homework problem). Another special case is the *transversal secretary problem*, where the secretaries are the left-hand-side vertices of a bipartite graph $G = (U, V, E)$, and $F \subseteq U$ belongs to $\mathcal{F}$ if and only if there exists a matching in G such that every vertex in F is matched. (Think, for instance, of V denoting locations, and edges denoting the locations that a secretary can work, with only one worker permitted per location.) Like the previous example, it is clear that $\mathcal{F}$ is downward-closed, and the exchange property follows from a combinatorial argument. There is also an online algorithm for this problem that achieves a constant-factor approximation guarantee [2, 5]. For a final example, the *representable matroid problem*, let U denote a set of vectors over a field $\mathbb{F}$, with $F \subseteq U$ in $\mathcal{F}$ if and only if the vectors F are linearly independent over $\mathbb{F}$. It is again obvious that $\mathcal{F}$ is downward-closed, and elementary linear algebra verifies the exchange property. While online constant-factor approximation algorithms are known for special cases of this problem [3], it is an open question whether or not such an algorithm exists for all representable matroids. The *matroid secretary conjecture* [1] states that every matroid secretary problem admits an online algorithm that achieves a constant approximation. Currently, the best approximation factor known for the general matroid secretary problem is $O(\log \log k)$, there k is the rank of the matroid [6, 4].[4]

[4]The exchange property implies that every maximal independent set of a matroid has the same cardinality. This common size is called the *rank* of the matroid. For example, in a graphical matroid with graph $G = (V, U)$, the rank is precisely $|V| - c(G)$, where $c(G)$ denotes the number of connected components of G.

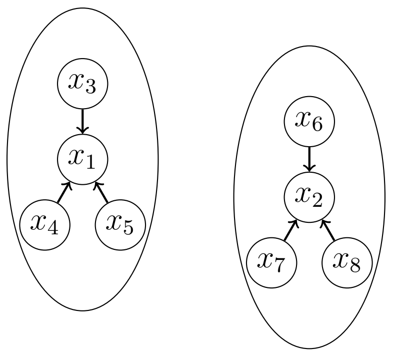

Figure 1: Points assigned to the same center can be viewed as a cluster.

3 Online Facility Location

The primary take-away from today's lecture is that the random-order model makes sense not only for the secretary problem, but more generally for many online problems. By "makes sense" we mean that you get a well-defined mathematical problem, and also that the assumption of random arrivals is reasonable in the motivating applications. We'll illustrate this point with a concrete case study on the *online facility location* problem [**?**]. This is basically the same as the k-median problem (Lecture #6), except with a cost-per-cluster (rather than a fixed number of clusters) and with points arrives online.

3.1 The Model

Here's the precise setup:

- Points $x_1 \ldots, x_n$ arrive online.[5]
- When point x_i arrives, its distance $d(x_i, x_j)$ to each previous point x_j ($j < i$) is unveiled. The distances are constrained to form a metric on the points that have arrived so far. (As usual, the crucial property will be the triangle inequality: $d(x,y) \leq d(x,z)+d(z,y)$ for all points x, y, z.)
- On x_i's arrival, the algorithm must either:
    1. make x_i a "center," at a cost of 1; or
    2. assign x_i to an existing center c (at a cost of $d(x_i, c)$).

The goal of the algorithm is to incur as little cost (center costs and assignment costs combined) as possible. As with the k-median problem, a solution naturally induces a clustering (i.e., partition) of the points, with one cluster per center, with all points assigned to the same center viewed as a cluster (Figure 1).

[5]Unlike in the secretary problem, here it won't matter for us whether or not n is known in advance.

Note that if x_i is at least distance 1 from every existing center, it always makes sense to make x_i a center. So the interesting case is where typical distances are small, like $\frac{1}{10}$ or $\frac{1}{100}$.[6] When a point is at distance less than 1 from an existing center, the myopically greedy course of action is to assign it to that center. But an algorithm might be better off by investing for the future and create a center for use by points that have yet to arrive.

The name "facility location" sounds old-school, and indeed the problem originates from the kinds of applications that the name suggests, of constructing physical buildings at multiple locations to serve various needs. A more modern interpretation is as an online clustering problem. The points could represent documents, images, etc. (mapped into some feature space), with distances representing (dis)similarities. If you don't know how many objects you ultimately want to cluster, it makes sense to not fix the number of clusters a priori. Since we also don't want each point in its own cluster, it then makes sense to incur some kind of penalty for each cluster used. For a machine learning analogy, the center cost can be thought of as a regularizer that penalizes more complex solutions (meaning solutions with more clusters). Finally, one should always remember that there are two distinct reasons to care about online algorithms: first, the problem itself may be fundamentally online; second, even if the entire input is known in advance, if the input is large and time is short, the one-pass flavor of online algorithms can make them a practical solution.[7]

3.2 Results

Recall from earlier lectures that the standard way of measuring the performance of an online algorithm is through its *competitive ratio*, which is defined as

$$\max_{\text{inputs}} \frac{\text{cost of online algorithm}}{\text{cost of offline optimum}}.$$

This is a very strong benchmark—the competition is both omniscient about the input and has unbounded computational power. Thus, it is not surprising that for many online problems, there are no online algorithms with good competitive ratios (recall the paging discussion from Lecture #3). The situation for online facility location is not as dire as for online paging, but there is a still a lower bound for the standard worst-case arrival order model.

Theorem 3.1 ([?]) *For the online facility location problem with an arbitrary arrival order, there is no $O(1)$-competitive online algorithm.*

The proof of Theorem 3.1 is an explicit example, and we leave the details to Homework #8.

Theorem 3.2 ([?]) *For the online facility location problem with an arbitrary metric space and a random arrival order, there is an algorithm that guarantees expected cost at most 8 times that of the offline optimal solution.*

[6]The assumption that the common center cost is 1 is without loss of generality, by scaling. It is also possible to extend the ideas from this lecture to accommodate different center costs at different points; see [**?**] for details.

[7]With the latter interpretation, it is often feasible to explicitly enforce the random order assumptions (e.g., randomly permuting the input before running the algorithm).

The constant factor may not seem especially impressive, but it seems likely that a better worst-case upper bound is possible and also that the (elegant) algorithm we present would perform better than its worst case in many cases. The algorithm will be randomized, so the expectation in Theorem 3.2 is over both the randomness in the arrival order and the algorithm's internal coin flips.

We'll spend the rest of the lecture proving Theorem 3.2.

3.3 Proof of Theorem 3.2

3.3.1 The Algorithm

The algorithm we'll analyze is a showcase for the elegance of randomized algorithms.

Online Facility Location Algorithm

```
1. Initialize $Y = \emptyset$. (These are the centers.)
2. When point $x_i$ arrives:
    (a) Let $c \in Y$ be the center closest to $x_i$.
    (b) Let $r_i = \min\{1, d(x_i, c)\}$. (Or for $x_1$, set $r_1 = 1$.) [Note $r_i \in [0,1]$]
    (c) With probability $r_i$:
        i. Make $x_i$ a center and add it to $Y$. (And assign $x_i$ to itself.)
    (d) Otherwise:
        i. Assign $x_i$ to the center $c$.
```

And that's the entire algorithm! Clearly, the algorithm can be implemented to run quite quickly.[8] We now proceed to the analysis.

3.3.2 The Analysis

We begin with a preliminary observation. Suppose a point x_i shows up with closest-center-distance r_i. Conditioned on everything that's happened in the past, the expected cost incurred by the algorithm in this step is

$$\underbrace{r_i \cdot 1}_{\text{make } x_i \text{ a center}} + \underbrace{(1 - r_i) \cdot r_i}_{\text{or not}} \leq 2r_i, \tag{1}$$

with the factor of 2 increasingly accurate as r_i approaches 0. The key point is that Theorem 3.2 reduces to proving an upper bound on $\mathbf{E}[\sum_{i=1}^n r_i]$ relative to the offline optimum, since the expected cost of the algorithm is at most twice this.

[8]The precise running time achievable depends on how the distances between a new point x_i and the already chosen centers are presented in the input (why?).

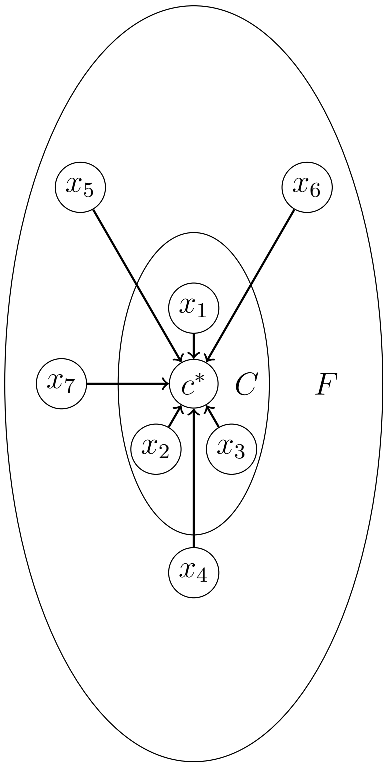

Figure 2: We can partition an optimal cluster into close points C and far points F.

We'll upper bound the r_i's cluster by cluster. That is, for the sake of analysis, we'll focus on a single center c^* chosen by the optimal solution (there must be at least one), and on the points A assigned to c^* in the optimal solution. We'll show that $\mathbf{E}\left[\sum_{x_i \in A} r_i\right]$ is at most 4 times the cost that the optimal solution incurs for this cluster. Summing over the clusters and applying the inequality (1) will then complete the proof of Theorem 3.2.

Also for the analysis, we partition A into two equal-sized groups, C and F. (Assume that $|A|$ is even, for simplicity.) In C are the points of A that are closest to c^* (the "close" points), and in F are the rest (the "far" points). That is, we imagine sorting the points of A in increasing order of distance from the center c^*, and put the first and second halves in C and F, respectively (Figure 2). Note that this classification of points into close and far depends only on the structure of the optimal solution, and is independent of the order in which the points are presented to our online algorithm. We'll bound $\mathbf{E}[\sum_i r_i]$ first for only the close points, and then argue that the cost incurred by the algorithm on the far points can be "charged" against the cost already incurred on the close points.

3.4 Close Points

Let r^* denote the average distance between a point of A and the cluster center c^*:

$$r^* = \frac{1}{|A|} \sum_{x_i \in A} d(x_i, c^*). \tag{2}$$

Note that for every close point $x_i \in C$, $d(x_i, c^*) \leq 2r^*$. (If 50% of the points of A were more than $2r^*$ from c^*, then the average distance would be more than r^*.)

We bound the cost incurred by our algorithm on close points in two phases. By definition, the second phase begins immediately after some close point $x_i \in C$ is chosen as a center. Everything prior to this event is the first phase.

We tackle the second phase first. Suppose $x_j \in C$ is the first close point chosen as a center. Recall from above that $d(x_j, c^*) \leq 2r^*$. So by the triangle inequality, every subsequent close point x_i is at distance at most $d(x_i, c^*) + 2r^*$ from the nearest open center (x_j, if nothing else). That is, $r_i \leq d(x_i, c^*) + 2r^*$. This is good news, since both terms on the right-hand side can be related to the optimal cost. Summing over all the close points that arrive in Phase 2, we have

$$\begin{aligned}
\sum_{\substack{x_i \in C \\ x_i \text{ in Phase 2}}} r_i &\leq \sum_{\substack{x_i \in C \\ x_i \text{ in Phase 2}}} (d(x_i, c^*) + 2r^*) \\
&\leq \sum_{x_i \in C} (d(x_i, c^*) + 2r^*) \\
&= \sum_{x_i \in C} d(x_i, c^*) + \sum_{x_i \in A} d(x_i, c^*), \qquad (3)
\end{aligned}$$

where in the final line we're using (2) and the fact that $|C| = |A|/2$.

We should be pleased with the progress so far: the expression in (3) is at most twice the cost incurred by the optimal solution in the cluster A (actually at most $\frac{3}{2}$ as large, do you see why?). Moreover, this upper bound on the sum of r_i's of the close points in Phase 2 holds with probability 1, independent of the arrival order and of the algorithm's random decisions.

So what's the sum of the r_i's of the close points that arrive in Phase 1? The answer depends on how long Phase 1 lasts. The first close point x_{i_1} is definitely part of Phase 1, so it contributes r_{i_1} to the sum. With probability r_{i_1}, x_{i_1} is made a center and Phase 1 ends immediately. With probability $1 - r_{i_1}$, Phase 1 continues, and the next close point x_{i_2} contributes r_{i_2} to the sum. And so on. Thus if the close points show up in the order $x_{i_1}, \ldots, x_{i_k}$, then the expected sum of the r_{i_j}'s in Phase 1 is exactly

$$r_{i_1} + (1 - r_{i_1})\left(r_{i_2} + (1 - r_{i_2})\left(r_{i_3} + (1 - r_{i_3})\left(\cdots\left(r_{i_k}\right)\cdots\right)\right)\right).$$

We claim that this expression is at most 1 (no matter what the r_{i_j}'s are). One way to see this is by induction on $|C|$. If $|C| = 1$ then this expression is just $r_{i_1} \leq 1$. For larger $|C|$, this expression is a weighted average (with weights r_{i_1} and $1 - r_{i_1}$) of the number 1 and a number that, by induction, is at most 1. This is again a number less than 1. Since this holds for any values of the r_{i_1}'s, the analysis again holds for an arbitrary ordering of the input points (in expectation over the internal coin flips of the algorithm).

Here's the summary of our progress so far:

Lemma 3.3 *For every ordering of the input points,*

$$\mathbf{E}\left[\sum_{x_i \in C} r_i\right] \le 1 + \frac{3}{2}\sum_{x_i \in A} d(x_i, c^*),$$

where the expectation is over the coin flips of the algorithm.

In Lemma 3.3, we're using the fact that $\sum_{x_i \in C} d(x_i, c^*) \le \frac{1}{2}\sum_{x_i \in A} d(x_i, c^*)$ (by the definition of close points).

3.5 Far Points

The next idea is to piggyback on Lemma 3.3 and "charge" the r_i's of far points to those of close points. In light of Theorem 3.1 and the worst-case bound in Lemma 3.3, we presumably need to use the random order assumption in this part of the proof.

Suppose that a far point x_i arrives after a previous close point x_j. Since centers never close, the center to which x_j was closest to upon arrival is one option for x_i upon its arrival. That is,

$$r_i \le r_j + d(x_i, x_j) \le r_j + d(x_i, c^*) + d(x_j, c^*), \tag{4}$$

where both inequalities follow from the triangle inequality.

Consider running our algorithm but withholding a far point x_i from the input. (A, C, F, etc. are defined as before, with respect to the original input.) Condition on the order of the input (excluding x_i). Lemma 3.3 implies that

$$\mathbf{E}\left[\sum_{x_i \in C} r_i\right] \le 1 + \frac{3}{2}\sum_{x_i \in A} d(x_i, c^*),$$

where the expectation is over the internal randomness of the algorithm. Now imagine re-inserting the withheld point x_i, at a random position. By this random order assumption, each of the close points $x_j \in C$ is equally likely to be the one that arrived most recently before x_i (plus there's the same probability $1/(|C|+1)$ that x_i arrives before all of the close points, in which case we use the bound $r_i \le 1$).[9] Hence, conditioned on the order arrival of

[9] A subtle point: inserting a new point x_i has no effect on the distribution of what happens before its arrival, such as the distribution of r_j's of close points that precede x_i.

all other points, inequality (4) implies that

$$\begin{aligned}\mathbf{E}[r_i] &\le \underbrace{\frac{1}{|C|+1}}_{\le \frac{1}{|C|}} \left[1 + \sum_{x_j \in C} (\mathbf{E}[r_j] + d(x_i, c^*) + d(x_j, c^*))\right] \\ &\le d(x_i, c^*) + \frac{1}{|C|}\left[1 + \mathbf{E}\left[\sum_{x_j \in C} r_j\right] + \sum_{x_j \in C} d(x_j, c^*)\right].\end{aligned}$$

Now we can use linearity of expectation to sum over all of the far points. Using the fact that $|F| = |C|$, we have

$$\mathbf{E}\left[\sum_{x_i \in F} r_i\right] \le \sum_{x_i \in F} d(x_i, c^*) + 1 + \mathbf{E}\left[\sum_{x_j \in C} r_j\right] + \sum_{x_j \in C} d(x_j, c^*).$$

Adding $\mathbf{E}\left[\sum_{x_j \in C} r_j\right]$ to both sides and using Lemma 3.3, we have

$$\begin{aligned}\mathbf{E}\left[\sum_{x_i \in A} r_i\right] &\le 1 + \sum_{x_i \in F} d(x_i, c^*) + 2\mathbf{E}\left[\sum_{x_j \in C} r_j\right] + \sum_{x_j \in C} d(x_j, c^*) \\ &\le 1 + \sum_{x_i \in F} d(x_i, c^*) + 2\left(1 + \frac{3}{2}\sum_{x_i \in A} d(x_i, c^*)\right) + \sum_{x_j \in C} d(x_j, c^*) \\ &= 3 + 4\sum_{x_i \in A} d(x_i, c^*).\end{aligned}$$

Since the optimal solution incurs cost $1 + \sum_{x_i \in A} d(x_i, c^*)$ in this cluster, we conclude that the expected sum of the r_i's of points in this cluster is at most 4 times as large. Summing over the clusters and recalling from (1) that the expected cost incurred by the algorithm is at most double the expected sum of the r_i's, we have established the competitive ratio of 8 claimed in Theorem 3.2.[10]

References

[1] M. Babaioff, N. Immorlica, and R. D. Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 434–443, 2007.

[10]Note that the entire proof works under a weaker assumption than a uniformly random ordering: an adversary can control the relative ordering of the close points, as long as the positions of the far points are (approximately) uniform.

[2] N. B. Dimitrov and C. G. Plaxton. Competitive weighted matching in transversal matroids. *Algorithmica*, 62(1-2):333–348, 2012.

[3] M. Dinitz and G. Kortsarz. Matroid secretary for regular and decomposable matroids. *SIAM Journal on Computing*, 43(5):1807–1830, 2014.

[4] M. Feldman, O. Svensson, and R. Zenklusen. A simple $O(loglogRank)$-competitive algorithm for the matroid secretary problem. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015. To appear.

[5] N. Korula and M. Pál. Algorithms for secretary problems on graphs and hypergraphs. In *Proceedings of the 36th Annual International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 508–520, 2009.

[6] O. Lachish. $O(loglogRank)$ competitive ratio for the matroid secretary problem. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 326–335, 2014.