CSC2421: Online and other myopic algorithms Spring 2021

Allan Borodin

April 8, 2021

Week 12

Agenda for today

I will be rapdily going through some recent work in online algorithms after first finishing up the prophet secretary problem. This will be the last meeting of the term.

- The prophet secretary matching problem
- Online algorithms with ML advice
- Probing problems and the stochastic rewards problem.

The prophet secretary matching problem

We have already seen that going from adversarial order to random order enabled an improvement in the competitive ratio from $\frac{1}{2}$ to $1-\frac{1}{e}$ and then to .669 for the single item prophet secretary problem. Can we extend the ideas used in the single item single threshold $1-\frac{1}{e}$ algorithm to obtain a $1-\frac{1}{e}$ competitive ratio for the prophet secertary matching problem?

It should be intutive that we will need more than a single threshold given that the values for each online buyer are being drawn independently from (possibly) different distributions. We will again assume continuous distrtibutions to avoid tie-breaking.

Conceptually, we can think of each offline node j having some intrinsic value which suggests some base price b_j for each offline node. It also seems reasonable that buyers arriving late might be afforded a lower threshold than if they arrived earlier as there are now less chances to sell item j. However, this "reasoning" was also true for the single item case where a single threshold sufficed and it turns out we only need threshold values for each offline node to obtain a $1-\frac{1}{e}$ ratio.

The prophet secretary matching algorithm

In fact, Ehsani et al show how to reduce the matching extension to the single item single threshold case. Whenever the i^{th} buyer arrives, they will be offered a specific offline item k given a price τ_k (i.e. a threshold for a centralized algorithm) and will accept that item iff the item is available and $\tau_k \leq v_{(i,k)}$; that is, its value for that match is at least the threshold.

Theorem

There is a non-adaptive threshold strategy that achieves competitive ratio $1 - \frac{1}{e}$ *for the prophet secretary matching problem.*

The analysis for the non-adaptive prophet secretary matching algorithm

Very roughly, here is the idea.
The analysis for the single item case can be strengthened for a very special case. Namely, if we assume $\sum_{i=1}^{n}[\Pr[v_i] < 0] \leq 1$ then $\mathbb{E}[Revenue] \geq \sum_{i=1}^{n}(1-\frac{1}{e})\mathbb{E}[v_i \cdot \mathbf{1}_{v_i<\tau}]$.

This then shows
$\mathbb{E}[ALG_\tau] \geq (1-\frac{1}{e})\sum_{i=1}^{n}\mathbb{E}[v_i]$

Now we have to indicate how we can "simulate" the assumption so as to set the appropriate thresholds.

The analysis of the non-adaptive threshold algorithm continued

Let $\mathbf{v}_{-i}$ denote the set of valuations of all buyers except the i^{th} buyer and let $\mathbf{v}_i$ denote the valuations of the i^{th} buyer. Given the valuations for each buyer, we calculate the probability $p_{k,i} = \Pr_{\mathbf{v}_{-i}}[(i, k)$ is in the optimal matching$]$.

For each buyer i, we choose a random number $r_i \in [0, 1]$ and choose item k such that $\sum_{\ell=1}^{k-1} p_{\ell,i} < r_i \leq \sum_{\ell=1}^{k} p_{\ell,i}$.

Now to compute the threshold for item k, we construct a new distribution $\mathbf{v}'$ such that for each buyer i, $\mathbf{v}_i$ is the distribution on the value of the edge adjacent to i in the expected maximum matching.

This then will satisfy the condition we need (i.e. $\sum_{i=1}^{n}[\Pr[v_i] < 0] \leq 1$) so that we can isolate out the contribution from each offline node.

Advice models

We have just been considering online algorithms in the context that the input sequence is stochastic to some extent. This is the most common approach to overcoming worst case results.

Another way to get around worst case results is to assume that the online algorithm has some precise "side information" about the input. For example, the algorithm designer may know the length of the input (an assumption we made for the secretary and prophet problems), or absolute lower and upper bounds of the cost/value of any item, or the maximum degree of vertices in a graph, etc.)

In online algorithms, we have tradionally viewed this side information as "advice" given by an oracle. That is, the oracle and algorithm work together and we assume the oracle always gives correct advice (about the input).

We consider two advice models, the per request model (where the algorithm requests information about the next item), and the tape model (where the algorithm has initial access to an advice string).

Advice models continued

Lets first consider the tape model and without worrying about the details of the tape model, assume that for any input sequence $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n$ that the computation uses only the first $\ell(n)$ bits of the advice string. Here we want $\ell(n)$ to be a short string containing information about the input set or sequence with n items.

For some problems, even $\ell(n) = 1$ for all n is sufficient to dramatically improve upon the optimal competitive ratio without advice. For example, consider the ski rental problem with rental cost $= 1$, and buying cost $= b$. Here each input determines whether or not the ski season ends. We know that the optimal deterministic online algorithm (without any advice) has competitve ratio $2 - \frac{1}{b}$.

Consider an online algorithm with one bit a of advice; namely, $a = 1$ iff there are at least B ski days. If the advice bit a is correct then we should buy on day 1 if $a = 1$ or else rent. This would yield an optimal cost whether $a = 1$ or $a = 0$.

The proportional knapsack problem

The *Proportional Knapsack Problem*
Input: $(w_1, w_2, \ldots, w_n)$ where $w_i \in \mathbb{R}^{\geq 0}$ and knapsack size W
Output: $\mathbf{z} = (z_1, z_2, \ldots, z_n)$ where $z_i \in \{0, 1\}$
Objective: To maximize $\sum_{i=1}^{n} z_i w_i$ subject to $\sum_{i=1}^{n} z_i w_i \leq W$

This is sometimes called the simple knapsack problem or the *subset-sum problem*. However, subset-sum also refers to the decision problem "does there exist $\mathbf{z}$ such that $\sum_{i=1}^{n} z_i w_i = W$?"
Aside: Subset-sum and 3-Subset-sum are basic problems the area of *fine-grained complexity*.

In our fast overview of the chapters, we mentioned (Week 3) the following results (appearing in Chapter 3): We saw a substantial improvemnent in the competitive ratio for the proportional knapsack even when using just one bit of randomization.

Without any randomization, no constant competitive ratio is possible.

But with one bit of randomization we obtain a $\frac{1}{4}$-competitiive ratio.

The randomized proportional knapsaack interpreted as an advice algorithm

Algorithm 10 Simple randomized algorithm for Proportional Knapsack

```
procedure SimpleRandom
    Let $B \in \{0, 1\}$ be a uniformly random bit                ▷ $W$ is the knapsack weight capacity
    if $B = 0$ then
        Pack items $w_1, \ldots, w_n$ greedily, that is if $w_i$ still fits in the remaining weight knapsack
capacity, pack it; otherwise, ignore it.
    else
        Pack the first item of weight $\geq W/2$ if there is such an item. Ignore the rest of the items. ▷
We can also pack the remaining items greedily (i.e. pack an iterm if it fits) but this is not needed
to establish the competitive bound.
```

We can re-interpret the randomized algorithm as a 1-bit advice algorithm. The one bit B indicates whether or not there is an input item $w_i \geq B/2$. With just this one bit of advice, the algorithm achieves a $\frac{1}{2}$ competitive ratio.

ML predicted advice

The ski rental problem is somewhat of an exception. Here 1 bit of correct advice results in competitive ratio $= 1$. In general, one expects that there is some tradeoff between the number of advice bits vs the competitive ratio one can obtain with advice.

More recently, a new line of research has emerged. Namely, instead of an oracle providing correct advice, we think of advice coming from an untrsted source. For example, consider predictions from a machine learning (ML) algorithm perhaps based on previously trained data (i.e. a weather forcaster that determines the number of days before warm weather will end the season).

In this situation we know the advice can be sometimes erroneous even if it is often highly reliable. However, for both the ski rental problem and the proportional knapsack problem, the competitive ratio can be arbitrarily bad if the prediction bit is incorrect.

The tradeoff between “robustness” and “consistency”

While every problem will have its own “natural” type or types of advice and ways to measure error in a prediction, there is a general basic question that we can ask once we have formulated a specific online problem with ML predictions or untrusted advice.

Informally the question is: Can we use ML or any untrusted advice to improve worst case performance (e.g. the competitive ratio) when the advice is correct or “almost correct” while not suffering dramatically when the advice is incorect and possibly far from being correct?

For predicted advice, we often want a parameterized algorithm that gives the algorithm designer a choice in determining the tradeoff between *robustness* and *consistency* where informally, robustness means that the algorithm will guard against terrible performance when the predictions are arbitrarily bad, while improving upon worst case results when predictions are correct or close to being correct (i.e. small error).

Consistent-robust algorithms: ski rental example

For definiteness, lets consider a cost problem where we are trying to minimnize the competitive ratio.

Let us say that an algorithm $\mathcal{A}$ is :

- α-consistent if its competitive ratio limits to α as the error limits to 0 and
- β-robust if the competitive ratio is bounded by β no matter the size of the error.

In the ski rental problem, the prediction y is a prediction on the number of ski days and lets say x is the actual number of ski days. Then the error $\eta = |y - x|$. (Note: Alternatively, we can think of the algorithm as having 1-bit of untrusted advice; namely, the bit predicts whether or not the ski rental season is at least b days long where b is the cost of buying the skis.) Consider (next slide) the following class of deterministic ski rental algorithms parameterized by λ due to Kumar et al [2018].

The ski rental parameterized algorithm

Algorithm 2: A deterministic robust and consistent algorithm.

```
if $y \geq b$ then
    Buy on the start of day $\lceil \lambda b \rceil$
else
    Buy on the start of day $\lceil b/\lambda \rceil$
end
```

[illegible]

Theorem

Algorithm 2 (above) is a deterministic algorithm that is $\min\{\frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda) \cdot OPT}\}$ *competitive with 1-bit of untrusted advice; hence the algorithm is* $\alpha = 1 + \lambda$ *consistent and* $\beta = 1 + \frac{1}{\lambda}$ *robust.*

For example, setting $\lambda = 1/2$, we would have a $\frac{3}{2}$ competitive algorithm when the advice is correct while being 3-competitive if the advice is incorrect (no matter how incorrect).

The MIN of online algorithms

A general approach to using ML advice is to try to combine two (or more) online algorithms in such a way that one obtains “approximately” the minimum of the individual competitive ratios.

This idea was first used by Fiat et al [1991] in the context of paging algorithms, and extended by Fiat, Rabani and Ravid [1994] to the k server problem enabling them to achieve competitve ratio $2^{O(k \log k)}$. This was the first online algorithm to achieve a ratio depending only on k (and not the size of the metric space or the number of server requests).

The idea was further extended by Blum and Burch [1997] in the context of the metrical task systems and expert learning.

We will use the MIN idea for caching ML advice but first a quick review of the classical paging problem.

Review of caching without advice

We recall the class of marking algorithms which work as follows: The algorithm executes in phases. At the start of a phase, all the pages in the cache are unmarked. When a page in the cache is requested, the page is marked. When a requested page is not in the cache (i.e. a cache miss), an unmarked page is evicted. If all pages are marked, a new phase begins with all pages now unmarked again. In either case, the newly requested page is brought into the cache and marked.

All deterministic marking algorithms have competitive ratio k (for a cache of size k) and this is optimal for deterministic algorithms. LRU and FIFO are examples of deterministic marking algorithms.

The randomized Fiat et al algorithm (called MARK) evicts a random unmarked page and has competetive ratio $2H_k \approx 2 \ln k$. The algorithms Partition (Mcgeouch and Sleator [1991] and Equitable (Achlioptas et al [1996] achieve the optimal randomized competitive ratio H_k.

Caching with ML advice

It is natural to revist the MIN idea in the context of ML advice. Here we now switch to the *advice per request* advice model. In a seminal paper promoting ML predicted advice, Lykouris and Vasilvitskii [2018] derive a consistent and robust randomized algorithm for caching.

The Lykouris and Vasilvitskii paper and the Kumar et al [2018] paper (on ski rental and non-clairvoyant job scheduling) are the two papers that popularized the idea of ML prediction advice applied to online algorithms.

The underlying idea is to combine an optimal on line algorithm (namely, a determninistic or randomized Marking algorithm) with the optimal offline algorithm (i.e. Belady's farthest in the future) algorithm. But of course, this has to be done carefully and one as to first define an appropriate error measure.

Lylouris and Vasilvitski also present evidence (on some real world data sets) that their consistent and robust algorithm "performs well on real data sets using off the shelf predictions".

Caching with ML advice continued

Let $\sigma_1, \sigma_2, \ldots, \sigma_n$ be a sequence of page requests. When a page request σ_i enters the cache, we record a prediction $y(i)$ as to the first time $x(i)$ when σ_i will be again be requested; that is the first time when $\sigma_{x(i)} = \sigma_i$. If σ_i is not requested again, $x(i) = n + 1$.

Lykouris and Vasilvitskii define *Blindoracle* as the non-robust eviction algorithm that uses a prediction y for the correct time x when a cache page will be requested again. The error bound is $\eta = \sum_{i=1}^{n} |y_i - x_i|$ which is normalized by OPT so as to be able to achieve meaningful results.

Rohatgi [2020] and Wei [2020] adapt the Lykouris and Vasilvitski algorithm to obtain improved results. For the deterministic algorithm, we present the result as in Wei.

Wei first improves the analysis for Blindoracle showing that the Blindoracle competitive ratio is bounded by $\min\{1 + 2\frac{\eta}{OPT}, 4 + \frac{4}{k-1}\frac{\eta}{OPT}\}$

Caching with ML advice: the deterministic algorithm

For a deterministic algorithm, Wei combines LRU with Blindoracle using the Fiat et al combiner which is essetially to "follow the leader". That is, upon receiving a request σ_t that causes a cache miss, the combined algorithm either uses LRU or Blindoracle to evict depending on which of these algorithms would have performed best so far (i.e. in processing $\sigma_1, \sigma_2, \ldots, \sigma_t$).

The result is the following determistic competitive ratio:

$$2 \min\{\min(1 + 2\frac{\eta}{OPT}, 4 + \frac{4}{k-1}\frac{\eta}{OPT}), k\}$$

.

Note that this means that the algorithm is consistent (with ratio approaching 2 when $\eta = o(OPT)$) and robust never exceding competitive ratio $2k$.

Caching with ML advice: the randomized algorithm

For a randomized algorithm, we will state the result as in Lykouris and Vasilvitskii. They combine the randomized MARK algorithm with Blindoracle. The combining here (for eviction) is more delicate than in the deterministic case.

Lykouris and Vasilvitskii achieve the following randomized competitive ratio (i.e., in expectation):

$$2 + O(\min\{\sqrt{\frac{\eta}{OPT}}, \log k\})$$

Note this means the algorithm is consistent with ratio approaching 2 when $\eta = o(OPT)$ and robust with ratio never exceding $O(\log k)$

Probing Problems

Consider an online application such as matching online arriving advertisers to specific (offline) advertisement slots where each advertisement slot has an associated reward that is realized if and only if the advertisement is in some sense “used” (i.e., the user clicks on the advertisement or purchases the item being advertised).

When an online node arrives, the matching algorithm knows both the value of each advertisement slot (if available and used) and only the probability that a given advertisement slot will be used. The algorithm must decide on which (if any) available slot (equivalently which edge from the online node to the slots) to *probe* (to see if the edge exists) and match to the online advertiser if that edge exists. If a match is not used, there is no reward. Determining if an edge exists can take time or be expensive and there is usually some limitation (or budget) on probes for each online node.

Probing in a general graph

While we are concentrating on online problems, the concept of probing (e.g., edges in a graph) was already explored in the context of offline algorithms for probing edges in a general graph within a *patience* (alsp called *timeout*) constraint that at most ℓ_v edges adjacent to a vertex v can be probed.

An important application is probing potential matchers for kidney translants where nodes are (donor,recipient) pairs and edges denote which pairs are biologically compatible. Clearly in this application probes can be expensive (and invasive) and there is often some limitation on which and how many probes can be attempted.

Another application is dating where probing can correspond to whether or not a match will turn into a romance. And again, matches can be time consuming and expensive. Indeed the initial paper in this area by Chen et al [2009] is called "Approximating matches made in heaven".

The stochastic rewards problem

We consider the following online *stochastic rewards (with patience) problem*. An adversary chooses a *stochastic graph* G which is a bipartite graph $G = (U, V, E, p, \ell, w)$ where V is the set of online nodes, U is the set of offline nodes, $p : E \to (0, 1]$ is a probabiliity on the edges, $\ell : V \to |U|$ is a patience parameter[1] and $w : U \to \mathbb{R}^{\geq 0}$ (or $w : E \to \mathbb{R}^{\geq 0}$) is a weight or value function. The interpretation is that p_e is the probability that the edge e exists, ℓ_v specifies the maxiumum number of times an edge adjacent to v can be probed (i.e., the maximum number of attempted matches) and w represents the value of an offline vertex (or edge). Furthermore, if an edge $e = (u, v)$ is probed and exists (with probability p_e), then (u, v) must be added to the current matching (if u is not already matched). This is called the *commitment* constraint.

[1]The patience constraint as initially defined in Chen et al (and most of ther following papers) was simply a limitatiion on the total number of edges that can be probed. More generally, there can be a set of constraints determining which edges can be probed.

Stochastic rewards continued

The objective is (as in the more "classical" online bipartite matching) to maximize the sum of the weights of vertices (or edges) that are included in the matching that is created. If the patience function ℓ is not specified then this means unit patience (i.e., only one attempted match for each online node) If w is not specified then this is the unweighted version where the objective is the size of the computed matching.

As indicated, the stochastic rewards problem belongs to a broader class of *stochastic probing problems*. Unlike more standard forms of stochastic optimization, it is not just that there is some stochastic uncertainty in the set of inputs, stochastic probing problems involve inputs that cannot be determined without probing (at some cost and/or within some constraint) so as to reveal the inputs. Applications of stochastic probing occur naturally in many settings. In a *non-adaptive* algorithm, there is a fixed order of probing the edges adjacent to each vertex so one might then probe (u, v) even though v is already matched.

Many variants of the stochastic rewards problem

Does the algorithm know the entire stochastic graph G in advance or does it only learn know the edges adjacent to an online node v when v arrives. The problem is meaningful in the case of a known stochastic G since the algorithm does not know the instantiation of any edge until that edge is probed.

Is the stochastic graph adversarial or generated according to some distibution (e.g. online vertices are generated by an i.d. or i.i.d. process).

Is the sequence of online nodes determined adversarially or is it random order (ROM)?

Is the stochastic graph vertex or edge weighted or unweighted.

What are the patience constraints (or other types of constraints) for each online node.

What is the “correct” benchmark?

What is the right benchmark for determining the competitive ratio of a stochastic rewards algorithm? The issue here is somewhat subtle. The following example shows that we cannot hope to obtain a reasonable competitive ratio if we are comparing an online algorithm to the expectation of the optimal value knowing the instantiations of the edge probabilities.

Example

Consider a single online vertex v having patience $\ell_v = 1$ and $n = |V|$ unweighted offline vertices where the probability of each edge is $\frac{1}{n}$. The expected value of any online algorithm is $\frac{1}{n}$ while the expectation of the optimal match is $1 - (1 - \frac{1}{n})^n \to 1 - \frac{1}{e}$. Hence the competitive ratio is $\Omega(n)$ and this also holds for the stochastic ROM and i.i.d. models as well as for adversarial inputs since there is only one online vertex.

The benchmark continued

For the stochastic rewards benchmark we shall hypothesize an optimal offline probing algorithm OPT. OPT knows the stochastic graph G and can probe edges in any order, adaptively choosing the next edge as a function of G and all previously revealed edges and decisions. In particular, OPT is not required to process edges in any online order. The benchmark may or may not have to satisfy commitment.

The (asymptotic) competitive ratio of an algorithm ALG is then defined as $\liminf_G \frac{\mathbb{E}[ALG(G)]}{\mathbb{E}[OPT(G)]}$. The strict competitive ratio is $\inf_G \frac{\mathbb{E}[ALG(G)]}{\mathbb{E}[OPT(G)]}$.

The expectation is with regard to the edge probabilities and the stochastic graph (if generated by some stochastic process and not adversarial).

While OPT is an ideal benchmark, it is not clear how to analyze the competitive ratio in terms of OPT. Instead, we can use an appropriate LP relaxation of OPT and analyze algorithms against the expected value of the optimum solution to the LP relaxation.

The choice of rhe LP relaation

The choice of the LP relaxation is important. If too restrictive it, may preclude legitimate solutions and hence not have any relevance to the desired competitive ratio against the ideal offline OPT. If it is “too relaxed” then the resulting competitive ratio (against the LP) will not be a good estimate of the competitive ratio wth respect to the ideal offline OPT.

We will aslo often use the chosen LP to guide our algorithms. And if we are interested in efficiency, then we have to be sure that the LP is efficiently solvable even though it may have exponentially many variables or constraints.

On the next two slides, I am stating a new LP relaxation that we (with Calum MacRury and Akash Rakheja) have recently introuduced. I am following the notation in previous work and our work where U is the offline vertices and V are the online vertices.

The configuration LP

Let $G = (U, V, E)$ be a stochastic graph with arbitrary edge weights, probabilities and constraints $(\mathcal{C}_v)_{v \in V}$. For each $k \geq 1$ and $\boldsymbol{e} = (e_1, \ldots, e_k) \in E^{(*)}$, define $g(\boldsymbol{e}) := \prod_{i=1}^{k}(1 - p_{e_i})$.

Notice that $g(\boldsymbol{e})$ corresponds to the probability that all the edges of $\boldsymbol{e}$ are inactive, where $g(\lambda) := 1$ for the empty string λ.

Observe then that $val(\boldsymbol{e}) := \sum_{i=1}^{|\boldsymbol{e}|} p_{e_i} w_{e_i} \cdot g(\boldsymbol{e}_{<i})$ corresponds to the expected weight of the first active edge of $\boldsymbol{e}$ if $\boldsymbol{e}$ is probed in order of its indices. Finally, for each $v \in V$ and $\boldsymbol{e} \in \mathcal{C}_v$, we introduce a decision variable $x_v(\boldsymbol{e})$. We can then express (on the next slide) the *configuration LP*.

The configuration LP

$$
\begin{aligned}
\text{maximize} \quad & \sum_{v \in V} \sum_{\boldsymbol{e} \in \mathcal{C}_v} \mathrm{val}(\boldsymbol{e}) \cdot x_v(\boldsymbol{e}) && && \text{(LP-config)} \\
\text{subject to} \quad & \sum_{v \in V} \sum_{\substack{\boldsymbol{e} \in \mathcal{C}_v : \\ (u,v) \in \boldsymbol{e}}} p_{u,v} \cdot g(\boldsymbol{e}_{<(u,v)}) \cdot x_v(\boldsymbol{e}) \le 1 && \forall u \in U && (2.1) \\
& \sum_{\boldsymbol{e} \in \mathcal{C}_v} x_v(\boldsymbol{e}) = 1 && \forall v \in V, && (2.2) \\
& x_v(\boldsymbol{e}) \ge 0 && \forall v \in V, \boldsymbol{e} \in \mathcal{C}_v && (2.3)
\end{aligned}
$$

The competitive ratio for the unknown stochastic graph with edge weights

We note that the setting of an unknown stochastic graph generalizes rhe standard online bipartite matching problem. Hence for edge weighted graphs, there cannot be a constant competitive ratio for adversarial order and $\frac{1}{e}$ is the best we can possibly obtain for online vertices arriving in random order.

We now show that the same $\frac{1}{e}$ ratio can be obtained by modifying the Kesselheim et al algorithm using the configuration LP. We non-adaptivley attempt a match for each online vertex using the fractional variuables of the configuration LP as probabilities.

The next slide is the randomized $\frac{1}{e}$ competitive algorithm for the edge weighted stochastic rewards problem in the random order model. This algorithm calls a subroutine (Vertex Probe) that attempts a match for an online vertex. Vertex probe is given in the following slide.

The edge weighted stochastic rewards problem for an unknown stochastic graph

Algorithm 2 Unknown Stochastic Graph ROM

Input: U and $n := |V|$.

Output: a matching $\mathcal{M}$ from the (unknown) stochastic graph $G = (U, V, E)$ of active edges.

```
 1: Set $\mathcal{M} \leftarrow \emptyset$.
 2: Set $G_0 = (U, \emptyset, \emptyset)$
 3: for $t = 1, \ldots, n$ do
 4:     Input $v_t$, with $(w_e)_{e \in \partial(v_t)}$, $(p_e)_{e \in \partial(v_t)}$ and $\mathcal{C}_{v_t}$.
 5:     Compute $G_t$, by updating $G_{t-1}$ to contain $v_t$ (and its relevant information).
 6:     if $t < \lfloor n/e \rfloor$ then
 7:         Pass on $v_t$.
 8:     else
 9:         Solve LP-config for $G_t$ and find an optimum solution $(x_v(\boldsymbol{e}))_{v \in V_t, \boldsymbol{e} \in \mathcal{C}_v}$.
10:         Set $e_t \leftarrow \text{VertexProbe}(v_t, \partial(v_t), (x_v(\boldsymbol{e}))_{\boldsymbol{e} \in \mathcal{C}_{v_t}})$.
11:         if $e_t = (u_t, v_t) \neq \emptyset$ and $u_t$ is unmatched then
12:             Add $e_t$ to $\mathcal{M}$.
13:         end if
14:     end if
15: end for
16: return $\mathcal{M}$.
```

The vertex probe subroutine

Algorithm 1 VertexProbe

Input: an online vertex s of a stochastic graph, $\partial(s)$, and probabilities $(z(\boldsymbol{e}))_{\boldsymbol{e} \in \mathcal{C}_s}$ which satisfy $\sum_{\boldsymbol{e} \in \mathcal{C}_s} z(\boldsymbol{e}) = 1$.

Output: an active edge $\mathcal{N}$ of $\partial(s)$.

```
1:  $\mathcal{N} \leftarrow \emptyset$.
2:  Draw $\boldsymbol{e}'$ from $\mathcal{C}_s$ with probability $z(\boldsymbol{e}')$.
3:  if $\boldsymbol{e}' = \lambda$ then                    ▷ the empty string is drawn.
4:      return $\mathcal{N}$.
5:  else
6:      Denote $\boldsymbol{e}' = (e'_1, \dots, e'_k)$ for $k := |\boldsymbol{e}'| \geq 1$.
7:      for $i = 1, \dots, k$ do
8:          Probe the edge $e'_i$.
9:          if $\mathrm{st}(e'_i) = 1$ then
10:             Add $e'_i$ to $\mathcal{N}$, and exit the "for loop".
11:         end if
12:     end for
13: end if
14: return $\mathcal{N}$.
```

The online stochastic rewards problem with offline vertex weights

We now consider stochastic graphs where the offline vertices (but not the edges) are weighted. We again are letting U be the offline nodes and V the online nodes.

We recall that in the classical (non-probing) setting, there is a randomized algorithm with competitive ratio $1 - \frac{1}{e}$ for offline vertex weighhted bipartite garphs and online nodes arriving in adversarial order. This result can also be reinterpreted as a deterministic $1 - \frac{1}{e}$ algorithm in the random order model.

We will extend this result to provide the same $1 - \frac{1}{e}$ ratio in the probing setting. Unfortuantely, we so far cannot obtain the result in the full generality that we want. In short, we can state a result when the probing constraint $\mathcal{C}_v$) for each on line vertex v is "rankable".

The competitive ratio for randable constraints on the online vertex probing sequence

For motivation, consider a patience constraiunt; that is, the number $|(u, v)|$ of probed edges is at most ℓ_v, the patience contraint.

Suppose we want to probe the edges (u, v) such that $u \in R \subseteq U$ where we R is the set of offline vertices not yet matched. This con be solved by using a dynmaic program. If the patience is $\ell_v = 1$ we would just probe the edge (u, v) with maximium value $p_{(u,v} \cdot w_u$. On the other hand if $\ell_v = |R|$, we would probe the edges (u, v) for $u \in R$ by non-increaing order of the weights w_u. For arbitrary ℓ, having sorted the vertices $u \in R$, we can use a dynmaic program to derive an optimal sequence of probes as given by Pruohit et al [2019] and Brubach et al [2019].

Rankable vertices

Informally, an online vertex v is rankable if there is a ordering π_v of the edges (u, v) such that the dynmaic program will yield the optimal expected value for any $R \subset U$. The permutation will depend only on the vertex weights an ad probabilitires and not R. .

This includes the following cases (extending known results):

- v has unit patience or unlimited patience; that is, $\ell_v \in \{1, |U|\}$.
- v has patience ℓ_v, and for each $u_1, u_2 \in U$, if $p_{u_1,v} \leq p_{u_2,v}$ then $w_{u_1} \leq w_{u_2}$.
- G is unweighted, and v has a budget[2] B_v with edge probing costs $(c_{u,v})_{u \in U}$, and for each $u_1, u_2 \in U$, if $p_{u_1,v} \leq p_{u_2,v}$ then $c_{u_1,v} \geq c_{u_2,v}$.

We will use an adaption of the primal dual analysis for this result due to Devanur et al [2013]. We note that this analysis was the basis for a proof of KVV (Eden et al [2021]) using an econmic interpretation.

[2] In the case of a budget B_v and edge probing costs $(c_v)_{v \in V}$, any subset of $\partial(v)$ may be probed, provided its cumulative cost does not exceed B_v.

The ROM algorithm for the vertex weighted case

While the primal dual analsis will employ an LP, the algorithm will be completely combinatorial.

Algorithm 3 Greedy-DP

Input: offline vertices U with vertex weights $(w_u)_{u \in U}$.

Output: a matching $\mathcal{M}$ of active edges of the unknown stochastic graph $G = (U, V, E)$.

```
1: $\mathcal{M} \leftarrow \emptyset$.
2: $R \leftarrow U$.
3: for $t = 1, \dots, n$ do
4:     Let $v_t$ be the current online arrival node, with constraint $\mathcal{C}_{v_t}$ and edges probabilities $(p_e)_{e \in \partial(v_t)}$.
5:     Set $\boldsymbol{e} \leftarrow$ DP-OPT$(v_t, R)$
6:     for $i = 1, \dots, |\boldsymbol{e}|$ do
7:         Probe $e_i$.
8:         if st$(e_i) = 1$ then
9:             Add $e_i$ to $\mathcal{M}$, and update $R \leftarrow R \setminus \{u_i\}$, where $e_i = (u_i, v_t)$.
10:        end if
11:    end for
12: end for
13: return $\mathcal{M}$.
```