In practice when designing algorithms, it is often the case that the input is a stream of data arriving over a span of time, and the algorithm needs to make decisions in real-time, before the entire input stream has been observed. This is the subject of *online algorithms*. In these lecture notes we explore a specific and archetypical example in the design and analysis of online algorithms: the online bipartite matching problem.

In the online bipartite matching problem, there is a bipartite graph $G = (V, E)$ whose vertex set V is partitioned into two sides L, R, known as the *left* and *right* or *offline* and *online* sides; every edge of G has one endpoint in L and the other endpoint in R. The contents of the set L are known to the algorithm at initialization time ($t = 0$), whereas the remaining information about G is revealed at times $t = 1, 2, \ldots, n = |R|$, by exposing one vertex of R at each time step. When vertex $j \in R$ arrives, all of its incident edges are revealed. The algorithm is then allowed to take one of the following actions: select one of the edges (i, j) that was revealed in the current step; or do nothing. The set of selected edges is required to be a matching; thus, if vertex $i \in L$ belongs to a previously selected edge, then (i, j) may not be selected in the current time step. The algorithm's objective is to maximize the number of edges selected.

A variation of this problem is the *online bipartite fractional matching* problem, in which the input sequence is the same, but the algorithm's output at time j is a tuple of numbers $(x_{ij})_{i \in L}$ satisfying:

- $x_{ij} = 0$ when $(i, j) \notin E$.
- $\sum_{i \in L} x_{ij} \leq 1$.
- for all $i \in L$, $\sum_{j \in R} x_{ij} \leq 1$.

In other words, the matrix of values $(x_{ij})_{(i,j) \in L \times R}$ eventually computed by the algorithm must belong to the fractional matching polytope of G. Fractional matching is of interest as a problem in its own right, and also as a window into the design of randomized online bipartite matching algorithms. From any such randomized algorithm, one can define a corresponding deterministic online fractional bipartite matching algorithm, obtained by setting x_{ij} to be the unconditional probability that the online algorithm selects edge (i, j). (Note that this unconditional probability can be computed at the time when vertex j arrives — i.e., it does not depend on any information to be revealed in the future — which is the reason why the fractional matching algorithm is a valid online algorithm.) Note that there is no obvious way to invert this transformation; in other words, given a deterministic fractional online matching algorithm, there is no obvious way to obtain a randomized online matching algorithm whose expected behavior yields the designated fractional algorithm.

1 A lower bound

What should we hope to achieve in an online bipartite matching algorithm? If we are unreasonably optimistic, we might hope to design an algorithm that is guaranteed to output a maximum cardinality matching. The following example shows that this is hopeless. Suppose $L = \{i_1, i_2\}$ and $R = \{j_1, j_2\}$. Consider two possible input sequences. In both of them, vertex j_1 arrives at

time $t = 1$ and reveals that it is connected to both i_1 and i_2. At time $t = 2$, vertex j_2 arrives and reveals that it has only one neighbor: in Input 1 this neighbor is i_1; in Input 2 it is i_2.

Notice that the maximum matching has size 2 in both of these inputs: j_2 can be matched to its only neighbor, whereas j_1 can be matched to the remaining element of L. Also notice that in both cases, this is the *unique* matching of size 2. Therefore, an online algorithm that seeks to select the maximum matching faces an insurmountable predicament: at time $t = 1$ it must match j_1 to one of its neighbors, there is a unique choice that is consistent with picking the maximum matching, and there is no way to know which choice this is until time $t = 2$. Thus, for every deterministic online algorithm, we can find an input instance that causes the algorithm to select a matching of size at most 1, while the maximum matching has size 2.

One can place this impossibility result in the broader context of *competitive analysis of online algorithms*, which evaluates algorithms according to the following criterion.

Definition 1. An online algorithm for a maximization problem is c-competitive if there exists a constant b such that for all input sequences,

$$c \cdot \mathsf{ALG} + b \geq \mathsf{OPT},$$

where ALG and OPT denote the values of the algorithm's solution and the optimum one, respectively. It is *strictly c-competitive* if $b = 0$ in the above bound. A randomized algorithm is c-competitive (against an oblivious adversary) if the above holds with $\mathbb{E}[\mathsf{ALG}]$ in place of ALG.

Our analysis of the two four-vertex input sequences above implies that deterministic online matching algorithms cannot be strictly c-competitive for any $c < 2$. By considering inputs comprising an arbitrarily long sequence of disjoint copies of either Input 1 or Input 2, we can eliminate the word "strictly" and conclude that deterministic algorithms cannot be c-competitive for any $c < 2$.

Our above discussion of the relationship between randomized and fractional algorithms shows that a lower bound on the competitive ratio of deterministic fractional online algorithms implies the same lower bound on the competitive ratio of randomized online algorithms. In particular, the competitive ratio of fractional (and hence randomized) online matching algorithms can be bounded below by 4/3, by an easy analysis of the same set of input sequences that furnished the lower bound of 2 for deterministic algorithms.

2 The greedy algorithm

It turns out that the example presented in Section 1 is the worst possible for deterministic algorithms, from the standpoint of competitive analysis. There is a strictly 2-competitive deterministic online algorithm. In fact, a competitive ratio of 2 is achieved by the most naïve algorithm: the greedy algorithm that matches each new vertex j to an arbitrary unmatched neighbor, i, whenever an unmatched neighbor exists. This fact follows directly from two simple lemmas.

Lemma 1. *Let G be any graph, M^* a maximum matching in G, and M a* maximal *matching in G (i.e., one that is not a proper subset of any other matching). The cardinalities of M and M^* satisfy $2|M| \geq |M^*|$.*

Proof. Construct a function f from M^* to M as follows. For every edge e in M^*, define $f(e)$ to be any edge in M that has an endpoint in common with e. There must be at least one such edge

in M, because otherwise $M \cup \{e\}$ would be a matching, contradicting our hypothesis that M is maximal. For every edge $e' = (i, j) \in M$, the set $f^{-1}(e')$ has at most two elements. (At most one with endpoint i, and at most one with endpoint j.) The inequality $2|M| \geq |M^*|$ follows immediately. □

Lemma 2. *The greedy online bipartite matching algorithm always selects a maximal matching in G.*

Proof. Let M denote the matching selected by the greedy algorithm. For every edge $e = (i, j)$ that does not belong to M, consider the time step in which vertex j arrived. Either j was matched to a vertex other than i at that time, or j was not matched to any vertex because all of its neighbors (including i) were already matched in M. In both cases, M contains an edge having either i or j as an endpoint, and therefore $M \cup \{e\}$ is not a matching. □

3 Online fractional matching: the waterfilling algorithm

It turns out that online fractional matching algorithms can achieve competitive ratios significantly better than 2, as we will see in this section.

First, a useful bit of terminology: we will refer to the sum $\sum_{j \in R} x_{ij}$ as the *fractional degree* of vertex i in fractional matching x. For a vertex $j \in R$ the fractional degree is defined similarly.

Perhaps the most natural idea for online fractional matching is to have each vertex j balance load equally among its neighbors. In other words, if a new vertex j arrives and has d neighbors, then for each neighbor i we set the value of x_{ij} to be $1/d$, unless that would violate the degree constraint of vertex i (the constraint that $\sum_j x_{ij} \leq 1$) in which case we merely increase x_{ij} as much as possible given the degree constraint.

However, this "stateless balancing" algorithm fails to be better than 2-competitive. To construct a counterexample, we take the example form Section 1 and blow up each vertex into n vertices, carefully modifying the edge set to cause the algorithm to make catastrophic decisions. The set L now has $2n$ vertices, which we will label as $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n$, and the set R has $2n$ vertices labeled $c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n$. Each vertex c_j has $n + 1$ neighbors: it is connected to a_j and also to $b_1, b_2, \ldots, b_n$. Each vertex d_j has only one neighbor, namely b_j. The maximum matching in this graph has size $2n$: it matches (a_i, c_i) and (b_i, d_i) for $i = 1, \ldots, n$. If the vertices $c_1, \ldots, c_n, d_1, \ldots, d_n$ arrive in that order, the stateless balancing algorithm will first assign a value of $\frac{1}{n+1}$ to each edge incident to $c_1, \ldots, c_n$. Thus, when $d_1, \ldots, d_n$ start arriving, each of them has a unique neighbor and the fractional degree of that neighbor is already $\frac{n}{n+1}$, so d_j can contribute only $\frac{1}{n+1}$ additional units to the size of the fractional matching. Thus, when the algorithm is finished processing the entire graph, the total size of its fractional matching is $n + \frac{n}{n+1}$, only slightly more than half of the optimum.

What went wrong in this algorithm? The vertices $b_1, \ldots, b_n$ are more "highly demanded" than $a_1, \ldots, a_n$ and it was unwise for vertices $c_1, \ldots, c_n$ to use up almost all of the capacity of $b_1, \ldots, b_n$ while using almost none of $a_1, \ldots, a_n$. The first vertex, c_1, can be forgiven for making this mistake since all of its neighbors looked indistinguishable when it arrived. But later on, we should have known better: we had already seen that the capacities of $b_1, \ldots, b_n$ were being depleted and should have taken measures to conserve that capacity. In short, there was nothing evidently wrong with the load-balancing idea, but it was silly to do *stateless* load-balancing; instead, we should have kept track of the current state (the amount of load already placed on each vertex in L) and adjusted our load-balancing decisions to correct for imbalances in the current load vector.

This bring us to the waterfilling algorithm. It keeps track of a "water level" for each $i \in L$ representing the current fractional degree $d(i) = \sum_j x_{ij}$, summing over all vertices $j \in R$ that have arrived in the past. When a new vertex j arrives, it allocates its one unit of fractional degree among its neighbors by finding the neighbors with the lowest water level and continuously raising their water level until either one unit of water has been poured into the graph, or the water level of all neighbors reaches 1, whichever comes first. In less metaphorical terms, the algorithm finds the unique number $\hat{\ell}(j)$ such that

$$\sum_{i \in N(j)} \max\{\hat{\ell}(j), d(i)\} = 1 + \sum_{i \in N(j)} d(i),$$

where $N(j)$ represents the set of all neighbors of j. It then sets

$$\begin{aligned} \ell(j) &= \min\{\hat{\ell}(j), 1\} \\ x_{ij} &= \max\{\ell(j), d(i)\} - d(i) \quad \forall (i,j) \in E \end{aligned}$$

and it updates $d(i)$ to $d(i) + x_{ij}$ for all i.

We will analyze the waterfilling algorithm using the primal-dual method. This means that we'll use the fractional matching LP

$$\begin{array}{lll} \max & \sum_{i,j} x_{ij} & \\ \text{s.t.} & \sum_j x_{ij} \leq 1 & \forall i \\ & \sum_i x_{ij} \leq 1 & \forall j \\ & x_{ij} \geq 0 & \forall i,j \end{array}$$

and its dual

$$\begin{array}{lll} \min & \sum_i \alpha_i + \sum_j \beta_j & \\ \text{s.t.} & \alpha_i + \beta_j \geq 1 & \forall (i,j) \in E \\ \alpha_i, \beta_j \geq 0 & \forall i,j & \end{array}$$

In particular, we define a dual solution $(\alpha_i)_{i \in L}$, $(\beta_j)_{j \in R}$ by specifying that

$$\alpha_i = g(d(i)) \quad \forall i \tag{1}$$

$$\beta_j = 1 - g(\ell(j)) \quad \forall j, \tag{2}$$

where

$$g(y) = \frac{e^y - 1}{e - 1}.$$

The choice of this specific function g will make more sense later in the analysis. The vital properties of g that are needed in the analysis are:

1. g is an increasing function.
2. $g(0) = 0$
3. $g(1) = 1$
4. $1 - g(t) + g'(t) = \frac{e}{e-1}$ for all t.

First, let's observe that the dual solution defined by (1)-(2) is feasible. This is because at the time we finish processing vertex j, the inequality $d(i) \geq \ell(j)$ is satisfied by all neighboring vertices i. Since the value $d(i)$ will not subsequently decrease, we also have $d(i) \geq \ell(j)$ at termination. Furthermore, since g is an increasing function, we have

$$\alpha_i + \beta_j = g(d(i)) + 1 - g(\ell(j)) \geq g(\ell(j)) + 1 - g(\ell(j)) = 1,$$

which verifies dual feasibility.

We claim that the fractional matching and the dual solution computed by our algorithm satisfy

$$\frac{e}{e-1}\sum_{(i,j)\in E} x_{ij} \geq \sum_{i\in L}\alpha_i + \sum_{j\in R}\beta_j. \tag{3}$$

By the weak duality, the sum on the right side is an upper bound on the size of any fractional matching in G, and therefore (3) implies that the waterfilling algorithm is $\left(\frac{e}{e-1}\right)$-competitive.

To prove (3), we compare β_j with a parameter β'_j defined as follows. For $t \in [0,1]$ let $n_j(t)$ denote the number of edges $(i,j) \in E$ such that the inequality $d(i) \leq t$ held at the time when j arrived. Note that

$$\int_0^{\ell(j)} n_j(t)\,dt = 1$$

provided that $\ell(j) < 1$, because in that case vertex j contributed one unit of "water" and the integrand denotes the rate at which water was filling the system as we increased the water level ℓ from t to $t + dt$. Now, define

$$\beta'_j = \int_0^{\ell(j)} (1 - g(t)) \cdot n_j(t)\,dt.$$

The inequality $\beta'_j \geq \beta_j$ always holds: when $\ell(j) = 1$ this is because $\beta_j = 0$, and when $\ell(j) < 1$ it is because $1 - g(t)$ is a decreasing function of t and therefore

$$\int_0^{\ell(j)} (1 - g(t)) \cdot n_j(t)\,dt > (1 - g(\ell(j)) \cdot \int_0^{\ell(j)} n_j(t)\,dt = 1 - g(\ell(j)) = \beta_j.$$

Letting $d(i)$ denote the degree of a vertex $i \in L$ before the arrival of vertex j, the amount by which the dual objective increases when processing j is:

$$\begin{aligned}
\beta_j + \sum_{i\in N(j)} [g(\ell(j)) - g(d(i))] &= 1 - g(\ell(j)) + \sum_{i\in N(j)} \int_{d(i)}^{\ell(j)} g'(t)\,dt \\
&= 1 - g(\ell(j)) + \int_0^{\ell(j)} g'(t) \cdot n_j(t)\,dt \\
&\leq \int_0^{\ell(j)} [1 - g(t) + g'(t)] \cdot n_j(t)\,dt \\
&= \frac{e}{e-1}\int_0^{\ell(j)} n_j(t)\,dt \\
&= \frac{e}{e-1}\sum_{i\in N(j)} x_{ij},
\end{aligned}$$

hence the increase in the dual objective is at most $\frac{e}{e-1}$ times the increase in the primal objective. Since the primal and dual objectives both start out at zero, this means that the dual objective at termination is at most $\frac{e}{e-1}$ times the primal objective, certifying inequality (3) and completing the proof that the waterfilling algorithm is $\left(\frac{e}{e-1}\right)$-competitive.

4 Randomized online matching: The RANKING algorithm

(Most of this section is an excerpt from the paper "Randomized Primal-Dual Analysis of RANKING *for Online Bipartite Matching" by N. Devanur, K. Jain, and R. Kleinberg, 2012.)*

Given an online fractional matching algorithm, it is tempting to try constructing a randomized online matching algorithm whose probability of choosing edge (i, j) is equal to the value x_{ij} computed by the fractional matching algorithm. If such a transformation were possible, it would yield a randomized online matching algorithm whose competitive ratio is exactly the same as that of the given fractional matching algorithm. Unfortunately, such a transformation is not possible in general. (For example, there is no randomized matching algorithm whose probability of selecting each edge (i, j) is exactly equal to the value assigned to that edge by the waterfilling algorithm. It is quite instructive to try proving this.)

However, there *is* a randomized online matching algorithm, known as RANKING, that achieves exactly the same competitive ratio as the waterfilling algorithm, namely $\frac{e}{e-1}$. Problem (1c) on this week's homework asks you to prove that RANKING achieves the best possible competitive ratio for randomized online matching algorithms.

The RANKING algorithm is actually very easy to describe: at initialization time, it samples a uniformly random total ordering of the vertices in L. Subsequently, as each vertex $j \in R$ arrives, if j has an unmatched neighbor in L then we choose the unmatched neighbor i that comes earliest in the random ordering, and we add (i, j) to the matching.

To analyze the RANKING algorithm, we begin with a reinterpretation of the algorithm in a way that is conducive to our analysis. Instead of picking a random total ordering of the vertices in L, each vertex in L picks a random number in $[0, 1]$ and a vertex $j \in R$, upon its arrival, is assigned to the unmatched neighbor who picked the lowest number. The algorithm is presented as Algorithm 1 below.

Algorithm 1: The RANKING algorithm.

```
foreach i ∈ L do
    Pick Y_i ∈ [0,1] uniformly at random
foreach j ∈ R do
    When j arrives, let N(j) denote the set of unmatched neighbors of j;
    if N(j) = ∅ then
        j remains unmatched
    else
        Match j to arg min{Y_i : i ∈ N(j)}
```

To analyze the algorithm, we note the standard LP relaxation for matching and its dual.

$$\text{maximize} \sum_{(i,j)\in E} x_{ij} \text{ s.t.} \qquad \text{minimize} \sum_{i\in L} \alpha_i + \sum_{j\in R} \beta_j \text{ s.t.}$$

$$\forall\, i \in V, \sum_{j:(i,j)\in E} x_{ij} \le 1. \qquad \forall\, (i,j) \in E, \alpha_i + \beta_j \ge 1.$$

$$\forall\, (i,j) \in E, x_{ij} \ge 0. \qquad \forall\, i, j, \alpha_i, \beta_j \ge 0.$$

Our analysis constructs a dual solution which is also randomized. The dual variables we construct may not always be feasible; in other words, they may violate the constraint $\alpha_i + \beta_j \ge 1$

for some edges (i, j). However, the *expected values* of the dual variables will constitute a feasible dual solution. The competitive ratio of $\frac{e}{e-1}$ will follow from the fact that the value of the dual solution is always $\frac{e}{e-1}$ times the size of the matching found, and that the expectation of the dual variables constitutes a feasible dual solution (whose value, of course, is also $\frac{e}{e-1}$ times the expected size of the matching found).

Our construction of the duals depends on a monotone non-decreasing function h that is closely related to the function g that came up in the analysis of the waterfilling algorithm in Section 3. The formula for h is $h(y) = e^{y-1}$ and its relevant properties are:

1. h is an increasing function;
2. $h(1) = 1$;
3. $\forall \theta \in [0, 1] \quad \int_0^\theta h(y)\, dy \; + 1 - h(\theta) = \frac{e-1}{e}$.

Note the similarity between the integral equation in property 3 of h and the differential equation in property 4 of the function g in Section 3; note also, however, that if we differentiate both sides of the integral equation defining h we certainly don't get the differential equation defining g.

Whenever i is matched to j, let

$$\alpha_i = \tfrac{e}{e-1} \cdot h(Y_i), \qquad \beta_j = \tfrac{e}{e-1} \cdot (1 - h(Y_i)).$$

For all unmatched i and j, set $\alpha_i = \beta_j = 0$. It will be useful to interpret the algorithm as follows: on matching i to j, we generate a value of 1 for the primal, which translates to a value of $\frac{e}{e-1}$ for the dual. Each unmatched vertex $i \in L$ that is a neighbor of j offers $\frac{e}{e-1} \cdot (1 - h(Y_i))$ of this value to j (to be assigned to β_j), while keeping the rest to itself (to be assigned to α_i). Then j is matched to the vertex that makes the highest offer.

Before we show that the expectation of the duals is feasible, we need certain properties of the algorithm specified by the following two lemmas. Let $(i, j) \in E$ be any edge in the graph. Consider an instance of the algorithm on $G \setminus \{i\}$, with the same choice of $Y_{i'}$ for all other $i' \in L$. Let y^c be the value of $Y_{i'}$ for the i' that is matched to j. Define y^c to be 1 if j is not matched. Let β_j^c be the value of β_j in this run, i.e. $\beta_j^c = \frac{e}{e-1} \cdot (1 - h(y^c))$.

Lemma 3 (Dominance Lemma)**.** *Given $Y_{i'}$ for all other $i' \in L$, i gets matched if $Y_i < y^c$.*

Proof. Suppose i is not matched when j arrives. This means that the run of the algorithm until then is identical to the run without i. From the definition of y^c, in the run without i, j is matched to i' such that $Y_{i'} = y^c$. Since $Y_i < y^c$, j is matched to i. □

Lemma 4 (Monotonicity Lemma)**.** *Given $Y_{i'}$ for all other $i' \in L$, for all choices of Y_i, $\beta_j \geq \beta_j^c$.*

Proof. Consider executing the algorithm on graphs G and $G \setminus \{i\}$ in parallel. At the start of every step of the two parallel executions, the unmatched vertices in L for the G execution constitute a superset of the unmatched vertices in L for the $G \setminus \{i\}$ execution. This statement is easily proven by induction: given that it holds at the start of one step, the only way it could be violated at the start of the next step is if the G execution chooses a vertex $i' \in L$ that is also unmatched, but is not chosen, in the $G \setminus \{i\}$ execution. Instead the $G \setminus \{i\}$ execution must choose some other vertex i'' such that $Y_{i''} < Y_{i'}$. By our induction hypothesis i'' was also unmatched in the G execution, contradicting the fact that the algorithm chose i' instead.

When node j arrives, its unmatched neighbors in the G execution form a superset of its unmatched neighbors in the $G \setminus \{i\}$ execution, so in the both executions j has an unmatched neighbor whose Y-value is y^c. If the algorithm instead chooses another neighbor of j, its Y-value can be at most y^c and hence, by the monotonicity of h, we have $\beta_j \geq \beta_j^c$. □

We now show that the above properties of h imply a competitive ratio of $\frac{e}{e-1}$ for RANKING.

Lemma 5. RANKING *is* $\frac{e}{e-1}$*-competitive.*

Proof. Whenever i is matched to j, $\alpha_i + \beta_j = \frac{e}{e-1}$. Therefore the ratio of the dual solution to the primal is always $\frac{e}{e-1}$. We show that the dual is feasible in expectation. In particular, we show that for all $(i, j) \in E$,

$$\mathbb{E}_{Y_i}[\alpha_i + \beta_j] \geq 1$$

for all choices of $Y_{i'}$ for all $i' \neq i \in L$. By the Dominance Lemma (Lemma 3) i is matched whenever $Y_i \leq y^c$. Hence

$$\mathbb{E}_{Y_i}[\alpha_i] \geq \frac{e}{e-1} \int_0^{y^c} h(y)\, dy.$$

By the Monotonicity Lemma (Lemma 4), $\beta_j \geq \beta_j^c = \frac{e}{e-1} \cdot (1 - h(y^c))$ for all choices of Y_i. The lemma now follows from the integral equation listed above as property 3 of h. □