Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Online Stochastic Matching with Unequal Probabilities

Aranyak Mehta*, Bo Waggoner**, and Morteza Zadimoghaddam*

*Google Inc. aranyak@google.com, zadim@google.com
**Harvard. bwaggoner@fas.harvard.edu

Abstract

The online stochastic matching problem is a variant of online bipartite matching in which edges are labeled with probabilities. A match will "succeed" with the probability along that edge; this models, for instance, the click of a user in search advertisement. The goal is to maximize the expected number of successful matches. This problem was introduced by Mehta and Panigrahi (FOCS 2012), who focused on the case where all probabilities in the graph are equal. They gave a 0.567-competitive algorithm for vanishing probabilities, relative to a natural benchmark, leaving the general case as an open question.

This paper examines the general case where the probabilities may be unequal. We take a new algorithmic approach rather than generalizing that of Mehta and Panigrahi: Our algorithm maintains, at each time, the probability that each offline vertex has succeeded thus far, and chooses assignments so as to maximize marginal contributions to these probabilities. When the algorithm does not observe the realizations of the edges, this approach gives a 0.5-competitive algorithm, which achieves the known upper bound for such "non-adaptive" algorithms. We then modify this approach to be "semi-adaptive:" if the chosen target has already succeeded, choose the arrival's "second choice" instead (while still updating the probabilities non-adaptively). With one additional tweak to control the analysis, we show that this algorithm achieves a competitive ratio of 0.534 for the unequal, vanishing probabilities setting. A "fully-adaptive" version of this algorithm turns out to be identical to an algorithm proposed, but not analyzed, in Mehta and Panigrahi (2012); we do not manage to analyze it either since it introduces too many dependencies between the stochastic processes. Our semi-adaptive algorithm thus can be seen as allowing analysis of competitive ratio while still capturing the power of adaptivity.

1 Introduction

1.1 Motivation. The Online Bipartite Matching problem has received considerable attention over the last decade, because it captures two sided markets with known demand and online supply (or vice-versa). The main application domain has been in Internet Advertising, although there has been recent work with applications in Crowdsourcing as well [11, 14]. A large number of variants have been studied, including budgeted, weighted and capacitated versions, as well as different stochastic models of online arrival (see Section 1.3 for more detailed references to this literature).

The basic online bipartite matching problem was introduced in [17]: There is a bipartite graph $G = (I, J, E)$, where the vertices in I (corresponding to advertisers in Internet advertising, tasks in crowdsourcing) are given offline, and a new vertex $j \in J$ (ad slot, workers) is revealed in each online step, together with its incident edges. The algorithm can either match j to one of its available (i.e. currently unmatched) neighbors in I or not match j at all, with the overall goal of maximizing the number of matched pairs at the end of the algorithm.

A recent paper [24] introduced a new variant of this classic problem, which they called the OnlineStochasticMatching problem. The motivation behind this variant was the observation that, in both the applications cited above, the goal is not simply to maximize the size of the matching, but to maximize the number of matched edges that become *successful*. Whether a matched edge becomes successful or not depends on a stochastic process which runs after the match has been made. The algorithm only has knowledge of the probability of success of each edge before making the match, and learns the outcome after making the match.

For example, in Internet advertising, the predominant payment method is the pay-per-click method: An advertiser pays only if the user actually clicks on the ad. So, matching the advertiser's ad to the arriving ad

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

slot is not sufficient; the ad should also be successful in terms of the user clicking on it. Only then can we say that the involved parties (user, advertiser and auctioneer) gained value from the match. The auctioneer has a prediction of the *clickthrough rate*, which is the probability that the ad will be clicked on if shown in this ad slot. Similarly, in crowdsourcing, the matching platform has an estimate of the probability that a given worker will successfully complete the task, and the value is derived only upon completion of the task. We formally define the problem next:

1.2 The OnlineStochasticMatching **Problem.** In this problem (also known as online matching with stochastic rewards), every edge (i, j) also has an associated probability of success p_{ij}. When a new vertex $j \in J$ arrives online, the algorithm either chooses not to match it at all, or matches it to one of its available neighbors. If j is matched to $i \in I$ using the edge (i, j), an independent random coin is tossed and the edge (i, j) becomes a successful match with probability p_{ij}. In this case, we say that i was successful and remove it from the set of available vertices. On the other hand, if the match (i, j) is not successful, i remains available and a neighbor of i that arrives later in the online order can be matched to i (but j cannot be matched again). The overall objective is to maximize the expected number of successful vertices $i \in I$. Note that one may assume the graph to be a complete bipartite graph, with $p_{ij} = 0$ for "non-edges". Also note that the case of all $p_{ij} \in \{0, 1\}$ is the classic non-stochastic case.

Competitive Ratio. In [24], the authors provide a definition for the competitive ratio of this problem, which captures the performance of the algorithm with respect to an optimum benchmark. Because the goal is to study both the online and the stochastic aspects of the problem, they define OPT as the optimal solution to the following offline, deterministic problem: The graph is known in advance and the reward of an edge (i, j) is deterministically p_{ij}. Each vertex $i \in I$ can be assigned multiple vertices $j \in J$, up to a budget of 1. That is, defining the load L_i on a vertex i to the sum of weights of the assigned edges, the goal is to maximize $\sum_i \min\{L_i, 1\}$. This definition reduces to the usual competitive ratio in the classic non-stochastic case. The paper also discusses other possible ways of defining competitive ratio, and the limitations of those alternate definitions.

1.3 Previous Results. Our results are directly related to the results in [24], which we describe in detail below.

In [24], the authors first show that a `Greedy` algorithm (which matches the arriving vertex j to that available neighbor i which has the highest value of p_{ij}) achieves a ratio of $1/2$. Thus, $1/2$ is a baseline for this problem upon which to improve (this is the case in the classic version as well, although the argument for the stochastic setting is a strict generalization).

Equal probabilities case: [24] focuses on the special case in which all the probabilities are equal, i.e., the p_{ij} are either 0 or p, for some value $p \in [0, 1]$. They provide an algorithm, called `StochasticBalance` to solve this special case, giving a competitive ratio of $0.5(1 + e^{-2}) \simeq 0.567$ as $p \to 0$, and which reduces to $1/2$ as p increases to 1. This algorithm works as follows:

Algorithm 1.1: StochasticBalance
(For the equal probabilities case, from [24].)
For each arriving vertex $j \in J$:
- Match j to that available (i.e., not yet successfully matched) neighbor that has been (unsuccessfully) matched the least number of times.

The algorithm makes a non-obvious observation: Suppose the arriving vertex j has two available neighbors. Why should we care which neighbor has more failed matches so far? After all, both neighbors have the same state right now, namely they are both available. But a `Greedy` algorithm, which ignores these failed matches, achieves only $1/2$. The difference comes from the online nature of the problem, where there is uncertainty over which neighbor will have more opportunities in the future.

The authors also show that the ratio of the `Ranking` algorithm (designed in [17] for the classic, non-stochastic case) stays above $1/2$ even in the stochastic case with equal probabilities, starting at $1 - 1/e$ with $p = 1$ and reducing to 0.534 as $p \to 0$.

Upper bound: The authors also provide an upper bound of $0.62 < 1 - 1/e$ for any (randomized) algorithm for this problem, showing that the combination of online and stochastic aspects of the problem make it strictly harder than the problem with only one of the two aspects: `Ranking` achieves $1 - 1/e$ in the non-stochastic online version, while the stochastic offline version has an simple algorithm achieving $1 - 1/e$: simply use the benchmark described above, *i.e.*, find a maximum allocation for the non-stochastic, budgeted, edge-weighted version of the instance, in which every vertex in I has a budget of 1, and an edge weight is equal to its probability of success.

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

The power of adaptivity: An adaptive algorithm is one which does not match to an already successful neighbor; a non-adaptive algorithm is one which ignores this information, and may make a wasteful match to an already succeeded vertex. In [24], the authors prove that no non-adaptive algorithm can achieve a ratio better than 1/2. Together with the algorithmic result, this gives a lower bound on the "adaptivity-gap" ([7]).

The general case: The authors leave the the following question open: Is there an algorithm which achieves a ratio strictly greater than 1/2 for non-uniform probabilities (even as all $p_{ij} \to 0$)? They suggest an algorithm, which they call `GeneralizedStochasticBalance`, as a strong candidate for solving the general case with small probabilities. This is a generalization of `StochasticBalance`, and makes intuitive sense. However, they state that the analysis becomes difficult, due to a certain technical argument (chaining of losses as we move items around during the charging argument) failing to go through. This failure even suggests a possible counter-example for the algorithm, although that seems hard to come up with. We will describe this algorithm in detail in Section 2.

1.4 Our Contributions. In this paper we attack the general problem with small probabilities, providing an algorithm, called `SemiAdaptive`, achieving a ratio of 0.534, thus resolving the open problem (but leaving open the optimal ratio).

THEOREM 1.1. *`SemiAdaptive` achieves a competitive ratio of 0.534 for the* ONLINESTOCHASTICMATCHING *problem with general probabilities p_{ij}, as $p_{ij} \to 0$.*

The general case is important, not only as a hard open question from the literature, but also because the special case is too artificial in the context of the motivating applications. In these applications, the question often becomes one of trading off two choices: one choice with a large probability of success and broad targeting (neighborhood), and another with low probability but narrow targeting. That is precisely what our algorithm does (as well as what `GeneralizedStochasticBalance` from [24] attempted to do).

Our algorithm is quite different from `GeneralizedStochasticBalance`, and our design and analysis approach is also different from that in [24]. The difference in the structure of the algorithm comes from the level of adaptivity it uses. Recall that [24] proved that a non-adaptive algorithm can do no better than a ratio of 1/2. Our algorithm is certainly adaptive, but in a more limited sense than `GeneralizedStochasticBalance`. Our algorithm uses scores which are updated non-adaptively, and which guide the adaptive decisions. The non-adaptive score updates and the adaptive choices work together in lock-step. Limiting the level of adaptivity makes the algorithm smoother in some sense, and more amenable to analysis, while retaining the power of adaptivity.

Our analysis techniques are also very different from those in [24]. The latter first finds a correspondence between this problem and that of budgeted allocation with stochastic, unknown budgets. This is a very interesting reduction, which helps in analyzing their algorithm using techniques from the non-stochastic problem. We do not use this correspondence at all, but provide a different, first-principles, argument.

Along the way, we also define a simple non-adaptive algorithm, which is a building block for our semi-adaptive algorithm, and prove that it achieves a ratio of 1/2. This shows that the upper bound of 1/2 for non-adaptive algorithms (proved in [24]) can be achieved. This ratio does not require vanishing probabilities.

THEOREM 1.2. *`NonAdaptive` achieves a competitive ratio of* 1/2 *for the* ONLINESTOCHASTICMATCHING *problem with general probabilities.*

Our building block also leads to the definition of a fully-adaptive algorithm (which we can not analyze), but which turns out to be precisely `GeneralizedStochasticBalance` – thus providing an interesting alternate derivation of the same.

In the next section, we briefly describe the landscape of other related work in the literature, before coming back to our problem and describing the algorithmic development and analysis in Section 2.

1.5 Other Related Work. There has been considerable work done recently on online matching, for which a detailed overview can be found in the survey article [23]. The basic non-stochastic version was introduced in [18], and the budgeted setting (the AdWords problem) was studied in [25, 5, 15]. Other variants in the non-stochastic version include vertex-weighted matching [1] and edge-weighted matching ([9], among others). Another series of results, also under the name of Stochastic Matching, deals with stochastic arrivals, either random order, or iid ([8, 10, 2, 16, 21, 22, 20, 19]) for the different versions of matching or budgeted allocation. Here, the rewards are non-stochastic, and the problems admit better ratios than the adversarial arrival models (as opposed to our problem which

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

is provably strictly harder than the non-stochastic version). Another related line of work is on matching with stochastic rewards on offline or stochastic arrival graphs [6, 3] with problem details making it uncomparable to our problem. While most of the work has been motivated by Online Advertising, there has been a recent spurt of activity from the motivation of Crowdsourcing as well [11, 14].

The question of the power of adaptivity in algorithms was introduced in [7], for the stochastic knapsack problem, and there has been some subsequent work ([12, 4, 13]) on other packing problems. In that framework, our result provides a bound on the adaptivity gap for online stochastic matching.

2 Our Algorithm

In this section we show the development of the algorithmic techniques. We will first describe two diametrically opposite algorithms: a fully-non-adaptive algorithm and a fully-adaptive algorithm (which turns out to be identical to `GeneralizedStochasticBalance`, and which we do not know how to analyze). Then we will define our semi-adaptive algorithm, which sits in a sweet-spot between adaptiveness and non-adaptiveness.

2.1 Starting Point: `NonAdaptive`. As a starting point, we begin by constructing a *non-adaptive* algorithm that achieves the upper bound of 0.5 for non-adaptive algorithms. We note that, in our setting of general edge probabilities, this result seems non-trivial. A naive greedy algorithm that assigns along the highest-probability edge has a competitive ratio of 0: consider a complete graph where all edges have probability p, except that all edges to the offline vertex i have probability $p + \epsilon$. The greedy algorithm will match every single arrival to i.

A better attempt could be to maximize the total online "weighted" matching (treating weights as probabilities), using an algorithm such as that in [25]. One difficulty is that we do not know how to set the "budgets" of that algorithm in this case, because we do not know the optimal matching in advance. Another question is in the analysis. Assigning a vertex a total weight of x only guarantees a probability of success of $1 - e^{-x}$. So naively, even if we could implement an algorithm guaranteeing a $1 - \frac{1}{e}$ ratio of the maximum weighted matching, this would only immediately guarantee a competitive ratio in our problem of $1 - e^{-(1-1/e)} = 0.468...$. It seems possible that such an approach could achieve 0.5, but a much finer analysis would be needed.

We briefly introduce some notation. Throughout this paper, we will take t_j to the the time at which the vertex $j \in J$ arrives, with $0 < t_j < 1$. $w_i(t)$ will be equal to the probability that vertex i has "succeeded" before time t in the algorithm (where the probability is over the realizations of the edges, and any randomness in the algorithm itself). We write w_i for $w_i(1)$, the probability that i succeeds.

ALGORITHM 2.1: `NonAdaptive`

Initialize $w_i(0) = 0,\ \forall i$

At time t_j, when vertex j arrives:

- Match j to the neighbor with the highest value of $(1 - w_i(t_j))p_{ij}$ (call that neighbor 1), irrespective of whether 1 has already succeeded or not.
- Update $w_1(t)$: For all $t > t_j$, update $w_1(t) = w_1(t_j) + (1 - w_1(t_j))p_{1j}$.

In Section 3, we show that `NonAdaptive` has a competitive ratio of 0.5. It exhibits two key characteristics that inspire our "semi-adaptive" algorithm. First, its update rule is "greedy at the margin": At each step, arrival j is matched to the vertex i that maximizes

$$\begin{aligned}&\Pr[i \text{ succeeds exactly on arrival } j]\\ &= \Pr\big[i \text{ has not succeeded by time } t_j\\ &\qquad \text{and edge } (i,j) \text{ succeeds}\big]\\ &= (1 - w_i(t_j))p_{ij}\end{aligned}$$

by independence of the edges. Thus, the algorithm is greedy in maximizing marginal contribution to expected number of successes.

The second characteristic is that the algorithm's variables, $w_i(t)$, depend only on the input instance and not the outcomes of the edges. If a certain edge succeeds with probability p_{ij}, then after we try that edge, we update $w_i(t)$ using the number p_{ij} regardless of whether the match actually succeeded or not. (Naturally, since the algorithm is non-adaptive.) Interestingly, this implies that, after running `NonAdaptive` on some given instance, our computed $w_i(t)$s can be used to calculate the competitive ratio on that input graph, even though this particular run of the algorithm may be better or worse than that ratio due to the randomness of the edges. While this property is natural for a non-adaptive algorithm, it will actually also hold for our semi-adaptive algorithm as well.

2.2 `FullyAdaptive` algorithm. Note how `NonAdaptive` performs non-adaptive choices as well as non-adaptive updates to the ws. At the other end of the spectrum is the following fully-adaptive algorithm, which performs adaptive choices as well as

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

updates.[1]

ALGORITHM 2.2: FULLYADAPTIVE
Initialize $w_i(0) = 0, \forall i$
At time t_j, when vertex j arrives:
- Order all vertices by decreasing value of $(1 - w_i(t_j))p_{ij}$.
- **Adaptive choice:** match the vertex to the first available (not yet successfully matched) neighbor in this order, call it i^*.
- **Adaptive updates:** if the match (i^*, j) fails to succeed, then update, $\forall\ t > t_j$: $w_{i^*}(t) = w_{i^*}(t_j) + (1 - w_{i^*}(t_j)) \cdot p_{i^*j}$

Interestingly, `FullyAdaptive` is essentially identical to `GeneralizedStochasticBalance` proposed in [24]. That algorithm picks the available neighbor with the highest value of $p_{ij}e^{-l(i)}$, where $l(i)$ is the "failed load" on i: the sum of $p_{ij'}$, where the sum is over all j' which were (unsuccessfully) matched to i in the past. Meanwhile, for the w_i defined in `FullyAdaptive`, $(1 - w_i(t))$ is equal to the product of $(1 - p_{ij'})$ over all j' which were (unsuccessfully) matched to i in the past. For vanishing probabilities, $(1-w_i(t)) \to e^{-l(i)}$. This equivalence shows that using the inverse exponential scaling function was the correct guess in `GeneralizedStochasticBalance`. As mentioned earlier, we do not know how to analyze this fully-adaptive algorithm.

2.3 Semi-adaptive: Attempt 1. In order to obtain both the power of adaptivity as well as the analyzability of `NonAdaptive`, we define a semi-adaptive algorithm. Here we first describe the first attempt at designing such an algorithm, and in the next section, we will define our final algorithm which is slightly different (mainly for ease of analysis). The overall strategy of the semi-adaptive algorithms can be described as doing **non-adaptive updates coupled with adaptive decisions**.

ALGORITHM 2.3: SEMI-ADAPTIVE ATTEMPT 1
Initialize $w_i(0) = 0, \forall i$
At time t_j, when vertex j arrives:
- Order all vertices by decreasing value of $(1 - w_i(t_j))p_{ij}$.
- **Adaptive choice:** Match the vertex to the first available neighbor in this order.
- **Non-adaptive updates:** $\forall\ i, \forall\ t > t_j$, update

$$w_i(t) = w_i(t_j) + \Pr[i \text{ is the first available neighbor}] \cdot p_{ij}$$

Note how the updates are completely non-adaptive, i.e., independent of the choice of the algorithm, or which edges actually succeed. This is critical in moving towards being able to analyze the algorithm. However, it seems like there is still not enough independence or controlled correlations between vertex successes in this algorithm to be able to analyze it. In particular, a difficulty in the analysis comes from "cascades". Consider an arrival j with first choice 1 and second choice 2. In the case where 1 has already succeeded, we would like to be able to lower-bound j's contribution to 2. But perhaps 1's success has caused some subsequent arrival to match to some i', which succeeded, and so on, causing 2 to succeed before j arrives. It is difficult to control the likelihood of this event analytically, so it is difficult to count j's contribution to 2.

This leads us to our final algorithm, which is also semi-adaptive (in the sense of decoupling the choices and the updates), but introduces certain limitations in order to keep the analysis manageable. First, we only utilize an arrival's first two choices in its ranking. It is perhaps surprising that this is sufficient to achieve our result, but on the other hand, the "power of two choices" is a common theme in randomized algorithms. Second, we introduce some independence that eliminates "cascades" of the type described above, at the cost of somewhat lowering the number of times a vertex utilizes its second choice.

2.4 Semi-adaptive: Final algorithm. As mentioned above, we will modify the previous semi-adaptive algorithm in two ways. First, for simplicity in the analysis, we will only consider the first two choices of an arriving vertex (rather than going down the entire list until finding an available neighbor).

Second, we will introduce some additional independence to help in the analysis. To visualize our approach, imagine the online arrival process as follows. First an input instance is selected; then, all edges are independently realized with the appropriate probabilities. Then, the arrival process begins. When a vertex j arrives, the algorithm observes the labels p_{ij} of its edges, but does not observe the realization. The algorithm can choose to "probe" a single edge; if the probed edge was actually realized, this match succeeds.

In the first attempt above at a semi-adaptive algorithm, we may describe the adaptive decision as follows: When j arrives, look at its first choice (call it 1). Look at all previous edges to 1 that have been

[1] For `FullyAdaptive`, unlike the other algorithms presented, $w_i(t)$ is not exactly equal to the probability that i succeeds by time t over all random edges realizations. Instead, it seems to be tracking probability of success only over edges matched to i.

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

probed by the algorithm; if any are successes, then j should move on to looking at its second choice instead. (If all are failures, then j should be assigned to 1.)

Our tweak will be this: Do not look at *all* previous edges to 1 that were probed. Instead, only look at edges coming from arrivals whose first choice was 1. If any are successes, then j may move on to looking at its second choice instead. Otherwise, attempt to assign j to 1 (but note this may fail because 1 has already suceeded due to some "second choice").

This is implementable for the following reason. As long as 1 is not yet successful, we *always* probe an edge from an arrival whose first choice is 1. If 1 becomes successful from a first choice arrival, then we do not need to probe any more edges; subsequent first choice arrivals should move on to their second choice. If 1 becomes successful from a second choice arrival, then for all subsequent first choice arrivals j', we can *simulate* probing the edge from j' to 1 by flipping a coin with weight $p_{1j'}$. We continue these simulated probes until one of them succeeds; after this time, first choice arrivals j should move on to their second choice.

The variable FirstSucc$(i) \in \{\texttt{False}, \texttt{True}\}$ will track whether or not a first-choice edge to i has been realized.

ALGORITHM 2.4: SEMIADAPTIVE

Initialize $w_i(0) = 0$, FirstSucc$(i) = \texttt{False}$, $\forall i$

At time t_j, when vertex j arrives:

- Order all vertices by decreasing value of $(1 - w_i(t_j))p_{ij}$.
- (Let $1 \in I$ be the highest, j's "first choice", and $2 \in I$ be the second-highest.)
- **Adaptive choice:**
  - If 1 has not yet succeeded, match j to 1 and if it succeeds, set FirstSucc(1) to `True`.
  - If 1 has succeeded but FirstSucc(1) = `False`, simulate matching j to 1: with probability p_{ij}, set FirstSucc(1) to `True`.
  - If 1 has succeeded, FirstSucc(1) = `True`, and 2 has not yet succeeded, match j to 2.
- **Non-adaptive updates:** For all $t > t_j$, update
  $w_1(t) = w_1(t_j) + \Pr[1 \text{ is available}]p_{1j}$
  $w_2(t) = w_2(t_j) +$
  $\Pr[\text{FirstSucc}(1) = \texttt{True},\ 2 \text{ is available}]p_{2j}$

Note that the updates in the last step are still non-adaptive, since they only rely on the $w_i(t)$s and the probabilities of FirstSucc(i) at time t. These are fixed for a given input instance regardless of edge realizations because they depend only on the identities of previous arrivals' first and second choices; these were non-adaptive because they relied on previously computed probabilities, and so on inductively.

To execute the algorithm, we need to be able to compute the updates to $w_i(t)$ at each time t, given all arrivals up until t. In Section 6, we show that $w_i(t)$ can be computed via constant-time updates for each arrival.

3 Warmup: Analyzing `NonAdaptive`

In this section, we show that `NonAdaptive` has a competitive ratio of 0.5, which is optimal for non-adaptive algorithms. We will be able to analyze `SemiAdaptive` by showing where it improves over `NonAdaptive` in this analysis.

Throughout this paper, given an input instance, in an optimal offline weighted assignment let $i^*(j)$ be the partner of an arrival j, let $j^*(i)$ be the set of partners of a vertex i, and let

$$o_i = \sum_{j \in j^*(i)} p_{ij}$$

be the total weight matched to i. For the algorithm under consideration (in this section, `NonAdaptive`), let 1_j and 2_j be the first and second choices of an arrival j. Recall that $w_i(t)$ is the probability that i has succeeded before time t and $w_i = w_i(1)$, the probability of success when all vertices have arrived.

Let

$$O_1 = \sum_i o_i(1 - w_i)$$
$$O_2 = \sum_i o_i w_i.$$

LEMMA 3.1. *For* `NonAdaptive`*, both of the following hold:*

$$\mathbb{E}[\textit{number of successes}] \geq O_1 \text{ , and}$$
$$\mathbb{E}[\textit{number of successes}] \geq O_2.$$

Proof. Since w_i is the probability that i has succeeded at the end of the arrivals,

$$\mathbb{E}[\text{number of successes}] = \sum_i w_i.$$

Since $o_i \leq 1$ for all i, we immediately get the second inequality. For the first, note that, for each arrival j that is assigned to 1_j, we have by definition of the algorithm that

$$(1 - w_{1_j}(t_j))p_{1_j j} \geq (1 - w_{i^*(j)}(t_j))p_{i^*(j)j}.$$

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Therefore,

$$\begin{aligned}\mathbb{E}[\text{number of successes}] &= \sum_j (1 - w_{1_j}(t_j))p_{1_j j} \\ &\geq \sum_j (1 - w_{i^*(j)}(t_j))p_{i^*(j)j} \\ &\geq \sum_j (1 - w_{i^*(j)})p_{i^*(j)j} \\ &\geq \sum_i (1 - w_i) \sum_{j:i^*(j)=i} p_{ij} \\ &= \sum_i (1 - w_i)o_i\end{aligned}$$

proving the first inequality.

THEOREM 3.1. *`NonAdaptive` has a competitive ratio of* 0.5.

Proof. Using Lemma 3.1,

$$\begin{aligned}2\,\mathbb{E}[\text{number of successes}] &\geq O_1 + O_2 \\ &= \sum_i o_i.\end{aligned}$$

4 Key Lemmas for `SemiAdaptive`

To analyze the performance of `SemiAdaptive`, we prove in this section certain structural lemmas. These give inequalities bounding the performance of the algorithm on any instance; in Section 5 we use these inequalities to obtain a numerical bound. Thus, the key part of our analysis lies in these structural lemmas (the remainder being mainly computation and adding less intuition). The overall strategy is to consider "gains", or contributions to the expected number of successes, broken down into categories depending on the arrival and the vertex it matches. These categories are formally defined and described in the next subsection; then we prove the lemmas.

4.1 Types of Gains. When an online vertex j arrives with first choice 1_j and second choice 2_j, it increases the probabilities that 1_j and 2_j have succeeded. We call each increase in probability of success a "gain", and we classify them into types, associated with colors to help with exposition and memory.

To recall notation introduced earlier in the paper: i denotes an offline vertex, j an arrival appearing at time $t_j \in (0,1)$, o_i the total weight assigned to i in the optimal offline matching, $i^*(j)$ and $j^*(i)$ the partners of j and i in the optimal offline matching, called their "OPT partners", $w_i(t)$ the probability that i has succeeded before time t, and $w_i = w_i(1)$.

The types of gains used are as follows:

- "Blue": gains of first choices; for each i, $B_i = \sum_{j:1_j=i}(1 - w_i(t_j))p_{ij}$.
- "Green": gains of second choices; for each i, $G_i = \sum_{j:2_j=i} \Pr[j \text{ matches to } i \text{ and succeeds}]$.
- "Purple": gains of first choices who are also OPT partners; for each i, $P_i = \sum_{j\in j^*(i):1_j=i}(1 - w_i(t_j))p_{ij}$.
- "Red": gains of first choices who are not OPT partners; for each i, $R_i = \sum_{j:1_j=i\neq i^*(j)}(1 - w_i(t_j))p_{ij}$.
- "Salmon": potential gains of second choices, from the perspective of the corresponding first choice. *I.e.*, if j's first choice is i, then j's contribution to S_i is how much gain j would have given its second choice had i not been ahead in the ranking. $S_i = \sum_{j:1_j=i}(1 - w_{2_j}(t_j))p_{2_j j}$.

For a final piece of notation, let

$$c_i = \sum_{j\in j^*(i):1_j=i} p_{ij}.$$

This corresponds to the total weight of OPT partners of i whose first choice in the matching is also i, or the purple gains.

To keep the big picture in mind, note that we have broken the gains down as follows by definition:

$$\begin{aligned}\text{Total gain} &= \text{Blue} + \text{Green} \\ \text{Blue} &= \text{Purple} + \text{Red}\end{aligned}$$

Blue corresponds to first choices and green to second choices. The salmon gains will be used to lower-bound the green gains, but there is no immediate connection.

4.2 Intuitive Overview. The previous analysis of `NonAdaptive` in Section 3 may be described as follows. First, we only counted blue, first-choice gains (since there are no second choices in that algorithm). Second, (as we will see,) red gains are "worse" in a sense than purple, so we lower-bounded the gains by assuming they were of the red type: matchings where the first choice is not the OPT partner. Third, since the algorithm chooses the first choice to maximize marginal gains, we could lower-bound the marginal gain of each arrival by the gain it would have given to its OPT partner, had that partner been its first choice.

To see how `SemiAdaptive` improves, consider an arrival j. Either its first choice is also its OPT partner, or not. If so, j gives us a "purple" gain, which is

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

quite good. The improvement here over "red" gains, intuitively, is that we can show that purple gains compound in a sense. Notice that, if all arrivals' first choices were their OPT partners, each offline vertex would match a total weight of at least o_i and the competitive ratio would be $1 - \frac{1}{e}$. So if j's first choice is its OPT partner, we already have an improvement over `NonAdaptive`.

If j's first choice is not its OPT partner, then by definition of the algorithm (just as with the analysis of `NonAdaptive`), j gives a "red" gain that is at least as large as it would have given to its OPT partner. Additionally (unlike with `NonAdaptive`), j may also be assigned to its second choice and give it a "green" gain. Since j's first choice was not its OPT partner, its second choice must be either its OPT partner or an option just as good. Thus, in this scenario, we get the same "red" gains as `NonAdaptive`, but we also get "green" second-choice gains.

Thus, both cases for an arrival j give improvements over `NonAdaptive`. The tradeoff between the cases is captured by the parameters c_i, which count how much arriving weight falls into the first case (js whose first choice is their OPT partner).

Finally, in order to bound the "green", second-choice gains, we will need to look from the perspective of a vertex i who has succeeded and whose first-choice arrivals begin assigning to their second choices. The contributions of such vertices can be bounded in terms of the hypothetical "salmon" gains S_i.

4.3 Lemmas. All lemmas refer to `SemiAdaptive` on any given input instance. We begin with simple bounds on the types of "blue", first-choice gains.

LEMMA 4.1. $\sum_i R_i \geq \sum_i (1 - w_i)(o_i - c_i)$.

Proof. As in the proof of Lemma 3.1, but only considering those first-choice partners of i whose OPT partner is not i.

$$\begin{aligned}
\sum_i R_i &= \sum_{j:1_j \neq i^*(j)} (1 - w_{1_j}(t_j)) p_{1_j j} \\
&\geq \sum_{j:1_j \neq i^*(j)} (1 - w_{i^*(j)}(t_j)) p_{i^*(j) j} \\
&\geq \sum_{j:1_j \neq i^*(j)} (1 - w_{i^*(j)}) p_{i^*(j) j} \\
&\geq \sum_i (1 - w_i) \sum_{j \in j^*(i):1_j \neq i} p_{ij} \\
&= \sum_i (1 - w_i)(o_i - c_i).
\end{aligned}$$

LEMMA 4.2. $P_i \geq (1 - w_i)(e^{c_i} - 1)$.

Proof. By definition,

$$P_i = \sum_{j \in j^*(i):1_j = i} (1 - w_i(t_j))\, p_{ij}.$$

Since $w_i(t)$ is monotonically increasing in t, this sum is minimized if all j in the sum arrive after all other vertices, *i.e.*, for some time t^*, $t_j > t^* \implies j \in j^*(i), 1_j = 1$. Thus, assume this is the case. Then we have

$$1 - w_i = (1 - w_i(t^*)) \prod_{j \in j^*(i):1_j = i} (1 - p_{ij}).$$

Verbally, the probability of failure at the end of the algorithm is the probability that we have not succeeded by time t^* *and* that each of our j fail (using independence to obtain the product). We can now lower-bound the total gain:

$$\begin{aligned}
w_i - w_i(t^*) &= (1 - w_i(t^*)) - (1 - w_i) \\
&= (1 - w_i) \left[\left(\prod_{j \in j^*(i):1_j = i} \frac{1}{1 - p_{ij}} \right) - 1 \right] \\
&\geq (1 - w_i) \left[\left(\prod_{j \in j^*(i):1_j = i} e^{p_{ij}} \right) - 1 \right] \\
&= (1 - w_i)(e^{c_i} - 1).
\end{aligned}$$

We simply used that $1 - x \leq e^{-x}$ for all x, that $(e^a)(e^b) = e^{a+b}$, and the definition of c_i.

Notice that, by combining Lemmas 4.1 and 4.2, we get

$$\begin{aligned}
B_i &= R_i + P_i \\
&\geq (1 - w_i)(e^{c_i} - 1 + o_i - c_i).
\end{aligned}$$

When $c_i > 0$, this is a strict improvement on the bound on `NonAdaptive` used in Lemma 3.1, which was $(1 - w_i)o_i$. And in fact, this inequality holds for `NonAdaptive` as well; however, for `NonAdaptive` one can set $c_i = 0$ and remove the benefit of this inequality.

Having bounded the blue (first-choice) types of gains, we now turn to bounding the green (second-choice) gains. Lemma 4.4, the most technically involved, will show that the green gains can be bounded in terms of "salmon" gains. Recall that S_i is the total gain that arrivals "would give their second choice if assigned". First, in Lemma 4.3, we lower-bound these salmon gains; the proof and intuition is almost identical to that of Lemma 4.1.

LEMMA 4.3. $\sum_i S_i \geq \sum_i (1 - w_i)(o_i - c_i)$.

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Proof.

$$\begin{aligned}\sum_i S_i &= \sum_j (1 - w_{2_j}(t_j)) p_{2_j j} \\ &\geq \sum_{j:1_j \neq i^*(j)} (1 - w_{2_j}(t_j)) p_{2_j j} \\ &\geq \sum_{j:1_j \neq i^*(j)} (1 - w_{i^*(j)}(t_j)) p_{i^*(j) j} \\ &\geq \sum_{j:1_j \neq i^*(j)} (1 - w_{i^*(j)}) p_{i^*(j) j} \\ &\geq \sum_i (1 - w_i) \sum_{j \in j^*(i):1_j \neq i} p_{ij} \\ &= \sum_i (1 - w_i)(o_i - c_i).\end{aligned}$$

The final lemma, Lemma 4.4, is crucial; without lower-bounding second-choice gains, we cannot guarantee a competitive ratio of larger than 0.5 (since then we are essentially non-adaptive). It is also the only lemma that requires vanishing probabilities; in particular, even `NonAdaptive` guarantees a competitive ratio of 0.5 for the large probabilities case. Finally, in Lemma 4.4 we will utilize our tweak to `SemiAdaptive` where an arrival only matches to its second choice if FirstSucc(i) = `True` for its first choice.

Explanation and sketch for Lemma 4.4. The intuition behind Lemma 4.4 is the following. Recall that the green gain variable G_i tracks the probability that i succeeds due to an arrival j where j's first choice was already taken, so it chose to match to i instead. Briefly, we say that G_i tracks that amount of "second-choice gains" sent to i. Meanwhile, we consider the salmon gain variable S_i, which does not have an immediately apparent contribution to the gain of the algorithm. Intuitively, S_i tracks the amount of hypothetical second-choice gains that could be sent out *from* i to other vertices. The total salmon gain $\sum_i S_i$ is relatively easy to lower-bound, by Lemma 4.3. Thus, in order to lower-bound the total green gain $\sum_i G_i$, we will compare it to a function of the salmon gains. This motivates Lemma 4.4.

The intuition behind the form and proof of Lemma 4.4 comes from the diminishing returns of arriving vertices. We will sketch an informal argument capturing the main intuition. Let $S_i(t)$ be the value of S_i summing only over vertices arriving before time t; *i.e.*, think of $S_i(t)$ as the value of S_i at the point in time t. Thus, $S_i = S_i(1)$.

Now, an arrival j whose first choice is i increases $S_i(t_j)$ by some amount, call it $dS_i(t_j)$, indicating that j could give a $dS_i(t_j)$ gain (increase in probability of success) to its second choice, call it i'. Sometimes, j actually does give this gain to i'. That is, intuitively, sometimes this $dS_i(t_j)$ gain contributes to $G_{i'}$. When does this occur? Exactly when j's first choice i is already taken and i' is available. We assert that the probability that i is already taken can be lower-bounded by the total $S_i(t_j)$ so far. So for each arrival, the green contribution is

$$\begin{aligned}&\Pr[i \text{ is taken}] \cdot (\text{amount of contribution}) \\ &\geq S_i(t_j) dS_i(t_j).\end{aligned}$$

So, the total green contribution over all arrivals, as probabilities vanish, has a lower bound that looks like

$$\int_{t=0}^{1} S_i(t) dS_i(t) = S_i^2/2.$$

There are two wrinkles in the above paragraph. First, the event that i is already taken is not independent of the event that the second choice i' is available, so we must be careful in awarding ourselves this gain. Second, because of our definition of `SemiAdaptive`, j can match to its second choice only if FirstSucc(i) = `True`, rather than any time i has already succeeded.

These wrinkles are related: The conditioning on FirstSucc(i) was added to `SemiAdaptive` in order to address the first wrinkle. By conditioning on FirstSucc(i) = `True` rather than on i having been already taken, we introduce independence between the event that j attempts to assign to its second choice i' and the event that i' has already succeeded. This allows us to lower-bound the green contributions almost as described above. After accounting for these subtleties, we lose an additive $S_i^3/6$ compared to the naive but faulty argument.

LEMMA 4.4. *In the limit as* $p_{ij} \to 0$, *we have* $\sum_i G_i \geq \sum_i \frac{S_i^2}{2} - \frac{S_i^3}{6}$.

Proof. **Notation:**

- Let $Take(i, t_j)$ be the event that offline vertex i has succeeded before time t_j. This stands for "i is **Take**n by some vertex before t_j." (Thus $w_i(t_j) = \Pr[Take(i, t_j)]$.)
- Let $Blue(i)$ be the set of arrivals whose first choice is offline vertex i.
- Let $Send_{ii'}$ be the subset of these whose first choice is i and whose second choice is i'. This stands for "the set of arrivals that i might **Send** to i'." All other vertices are denoted $nonSend_{ii'}$.
- Let $Take(i, t_j, Send_{ii'})$ be the event that i is successful before time t_j due to an arrival in the set $Send_{ii'}$ (standing for "i is **Take**n by a vertex in $Send_{ii'}$ before t_j"), and analogously for $Take(i, t_j, nonSend_{ii'})$.

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

- For an arriving vertex j with first choice 1_j, let BS_j be the event that a first-choice edge to j's first choice is realized before j arrives. This stands for "**B**lue **S**uccess at j's first choice." Thus, BS_j is the event that FirstSucc(1_j) is set to `True` before j arrives.

- Let $y_j = (1 - \Pr[BS_j])p_{1_j j}$. Thus, y_j is the probability that FirstSucc(1_j) is set to `True` exactly due to j's arrival.

- Let $x_j = (1 - \Pr[Take(2_j, t_j)])p_{2_j j}$. If j has no second choice, set $x_j = 0$. We have $S_i(t) = \sum_{j \in Blue(i): t_j < t} x_j$.

We note that $\Pr[BS_j] \leq \Pr[Take(1_j, t_j)]$, so $y_j \geq (1 - \Pr[Take(1_j, t_j)])p_{1_j j}$. Because of the algorithm's ranking of 1_j ahead of 2_j, this implies that $y_j \geq x_j$.

We will use the symbols for logical AND ($\wedge$) and NOT ($\neg$).

Proof: We wish to bound the total "green" gain, which is

$$\sum_i G_i = \sum_i \sum_{j \in Blue(i)} \Pr[BS_j \wedge \neg Take(2_j, t_j)] p_{2_j j}$$

$$= \sum_i \sum_{j \in Blue(i)} (\mathbf{P_1} \cdot \mathbf{P_2}) \tag{4.1}$$

where

$$\mathbf{P_1} = \Pr\Big[\neg Take(2_j, t_j, nonSend_{i2_j}) \mid BS_j \wedge \neg Take(2_j, t_j, Send_{i2_j})\Big]\, p_{2_j j},$$

$$\mathbf{P_2} = \Pr[BS_j \wedge \neg Take(2_j, t_j, Send_{i2_j})].$$

To bound this sum, we use the following claim (proof given at the end of the proof of the lemma):

Claim 1. *First,* $\mathbf{P_1} \geq x_j$.
Second,

$$\mathbf{P_2} \geq \sum_{\substack{j_1 \in Blue(1_j): \\ t_{j_1} < t_j}} x_{j_1} \left(1 - \sum_{\substack{j_2 \in Send_{1_j 2_j}: \\ t_{j_1} < t_{j_2} < t_j}} x_{j_2}\right).$$

Plugging Claim 1 into Equation 4.1, we get

$$\sum_i G_i = \sum_i \mathbf{Z_i},$$

where, letting $J_1 = \{j_1 \in Blue(i) : t_{j_1} < t_j\}$,

$$\mathbf{Z_i} \geq \sum_{j \in Blue(i)} x_j \sum_{j_1 \in J_1} x_{j_1} \left(1 - \sum_{\substack{j_2 \in Send_{i2_j}: \\ t_{j_1} < t_{j_2} < t_j}} x_{j_2}\right)$$

$$\geq \sum_{j \in Blue(i), j_1 \in J_1} x_j x_{j_1} - \sum_{\substack{j \in Blue(i), j_1 \in J_1, \\ j_2 \in Blue(i): \\ t_{j_1} < t_{j_2} < t_j}} x_j x_{j_1} x_{j_2}.$$

The first line was simply plugging in the claim, and for the second inequality, we used that $Send_{1_j 2_j}$ is a subset of $Blue(1_j)$ and each $x_i \geq 0$, so subtracting a sum over more terms can only decrease the total.

Now, recalling that $S_i = \sum_{j \in Blue(i)} x_j$, in the limit as all $p_{ij} \to 0$ (and therefore all $x_{ij} \to 0$), we get

$$\sum_i G_i \geq \frac{S_i^2}{2} - \frac{S_i^3}{6}.$$

It remains only to prove Claim 1.

Proof. [Proof of Claim 1, first part] By expanding the definition of x_j, we see that we must prove that

$$\Pr\left[\neg Take(2_j, t_j, nonSend_{1_j 2_j}) \mid BS_j \wedge \neg Take(2_j, t_j, Send_{1_j 2_j})\right] \geq \Pr[\neg Take(2_j, t_j)].$$

Intuitively, the reason this holds is that the choice of an arrival in $nonSend_{1_j 2_j}$ to attempt to assign[2] to 2_j is independent of the events BS_j and $\neg Take(2_j, t_j, Send_{1_j 2_j})$. So on both the left and the right side of the inequality, we have the same attempts from all $nonSend_{1_j 2_j}$ arrivals. On the other hand, the right side possibly includes attempts from $Send_{1_j 2_j}$ arrivals, which lowers the probability that 2_j is available, but on the left side we know that every attempt from $Send_{1_j 2_j}$ arrivals is unsuccessful.

More formally, we have that

$$\Pr[\neg Take(2_j, t_j)] = \prod_{j' : t_{j'} < t_j} \Pr[j' \text{ does not take } 2_j \mid \neg Take(2_j, t_{j'})]. \tag{4.2}$$

That is, the product, over all arrivals up to time t_j, of the probability that this arrival fails to assign to 2_j and succeed, given that 2_j is available (*i.e.* given that all prior arrivals have failed on 2_j).

[2] We say j *attempts to assign* to i if i is j's first choice, or if i is j's second choice and j's first choice has succeeded. In other words, j will assign to i if i is available.

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Meanwhile, letting
$J = \{j' \in nonSend_{1_j 2_j} : t_{j'} < t_j\}$,

$$\begin{aligned}
&\Pr\left[\neg Take(2_j, t_j, nonSend_{1_j 2_j}) \right. \\
&\qquad \left. \mid BS_j \wedge \neg Take(2_j, t_j, Send_{1_j 2_j})\right] \\
&= \prod_{j' \in J} \Pr\Big[j' \text{ does not take } 2_j \mid \neg Take(2_j, t_{j'}), \\
&\qquad\qquad BS_j, \neg Take(2_j, t_j, Send_{1_j 2_j})\Big] \\
&= \prod_{j' \in J} \Pr[j' \text{ does not take } 2_j \mid \neg Take(2_j, t_{j'})]. \qquad (4.3)
\end{aligned}$$

The final equality holds because, given that 2_j is available when j' arrives, j''s choice to assign to 2_j depends only on whether one of the following occur: its first choice $1_{j'} = 2_j$; or its second choice $2_{j'} = 2_j$ and $BS_{j'}$, that is, j''s first choice "would have succeeded from a blue" by time $t_{j'}$. Thus, the probability that j' assigns to 2_j, given that 2_j is available, is independent of BS_j and $\neg Take(2_j, t_j, Send_{1_j 2_j})$.

Now the first part of the claim reduces to proving that Expression 4.2 is at most Expression 4.3. Since each is a product over probabilities, and the former contains all the terms of the latter and some additional terms, we are done.

Proof. [Proof of Claim 1, second part]

$$\begin{aligned}
\mathbf{P_1} &= \Pr[BS_j \wedge \neg Take(2_j, t_j, Send_{1_j 2_j})] \\
&= \Pr[BS_j] - \Pr[BS_j \wedge Take(2_j, t_j, Send_{1_j 2_j})].
\end{aligned}$$

We will show that

$$\begin{aligned}
&\Pr[BS_j \wedge Take(2_j, t_j, Send_{1_j 2_j})] \\
&\leq \sum_{j_2 \in Send_{1_j 2_j} : t_{j_2} < t_j} x_{j_2} \left(\sum_{j_1 \in Blue(1_j) : t_{j_1} < t_{j_2}} y_{j_1} \right). \qquad (4.4)
\end{aligned}$$

Before proving Inequality 4.4, we show how it finishes the proof of the second part of the claim. Inequality 4.4, along with the fact that by definition, $BS_j = \sum_{j_1 \in Blue(1_j) : t_{j_1} < t_j} y_{j_1}$, gives that

$$\begin{aligned}
\mathbf{P_1} &= \Pr[BS_j] - \Pr[BS_j \wedge Take(2_j, t_j, Send_{1_j 2_j})] \\
&\geq \sum_{j_1 \in Blue(1_j) : t_{j_1} < t_j} y_{j_1} \\
&\qquad - \sum_{\substack{j_2 \in Send_{1_j 2_j} : \\ t_{j_2} < t_j}} x_{j_2} \left(\sum_{\substack{j_1 \in Blue(1_j) : \\ t_{j_1} < t_{j_2}}} y_{j_1} \right) \\
&= \sum_{\substack{j_1 \in Blue(1_j) : \\ t_{j_1} < t_j}} y_{j_1} \left(1 - \sum_{\substack{j_2 \in Send_{1_j 2_j} : \\ t_{j_1} < t_{j_2} < t_j}} x_{j_2} \right)
\end{aligned}$$

after rearranging. Noting that the coefficient of each y_{j_1} is nonnegative (as it is a probability), we use the inequality $y_{j_1} \geq x_{j_1}$ to complete the proof of the claim.

It only remains to prove Inequality 4.4.

$$\begin{aligned}
&\Pr[BS_j \wedge Take(2_j, t_j, Send_{1_j 2_j})] \\
&= \sum_{\substack{j_2 \in Send_{1_j 2_j} : \\ t_{j_2} < t_j}} \Pr[BS_{j_2}] \Pr[\neg Take(2_j, t_{j_2}) \mid BS_{j_2}] p_{2_{j_2} j_2}
\end{aligned}$$

We will use the fact that $\Pr[\neg Take(2_j, t_{j_2}) \mid BS_{j_2}] \leq \Pr[\neg Take(2_j, t_{j_2})]$. To show this fact, expand:

$$\begin{aligned}
&\Pr[\neg Take(2_j, t_{j_2})] \\
&= \prod_{j' : t_{j'} < t_{j_2}} \Pr[j' \text{ does not take } 2_j \mid \neg Take(2_j, t_{j'})], \qquad (4.5)
\end{aligned}$$

and

$$\begin{aligned}
&\Pr[\neg Take(2_j, t_{j_2}) \mid BS_{j_2}] \\
&= \prod_{j' : t_{j'} < t_{j_2}} \Pr[j' \text{ does not take } 2_j \mid \neg Take(2_j, t_{j'}), BS_{j_2}]. \qquad (4.6)
\end{aligned}$$

The terms in the two expressions correspond one-to-one. If we do not have $1_{j'} = 1_j$ and $2_{j'} = 2_j$, then there is no difference in the j' term between the two expressions. But if we do, then the j' term in Equation 4.6 is smaller than the term in Equation 4.5, because given the conditions of Equation 4.6, j' is certain to attempt to match to $2_{j'}$, whereas this is not certain in Equation 4.5. This proves the fact.

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Using this fact, we prove Inequality 4.4:

$$\begin{aligned}
&\Pr[BS_j \wedge Take(2_j, t_j, Send_{1_j 2_j})] \\
&\leq \sum_{j_2 \in Send_{1_j 2_j} : t_{j_2} < t_j} \Pr[BS_{j_2}] \Pr[\neg Take(2_j, t_{j_2})] p_{2_{j_2} j_2} \\
&= \sum_{j_2 \in Send_{1_j 2_j} : t_{j_2} < t_j} \Pr[BS_{j_2}] x_{j_2} \\
&= \sum_{j_2 \in Send_{1_j 2_j} : t_{j_2} < t_j} x_{j_2} \left(\sum_{j_1 \in Blue(1_j) : t_{j_1} < t_{j_2}} y_{j_1} \right).
\end{aligned}$$

Having proved the claim, we are done proving the lemma.

5 Deriving a Bound for `SemiAdaptive`

In this section, we use the constraints implied by our lemmas to prove the numerical bound on the competitive ratio of `SemiAdaptive`.

For any input instance, the performance of `SemiAdaptive` satisfies Lemmas 4.1, 4.2, 4.3, and 4.4. Therefore, the solution to the following optimization problem is a lower bound on the competitive ratio of `SemiAdaptive`:

$$\begin{aligned}
&\text{minimize} \quad \frac{\sum_i w_i}{\sum_i o_i} \\
&\text{s.t.} \\
&w_i = B_i + G_i && \forall i \\
&B_i = P_i + R_i && \forall i \\
&P_i \geq (1 - w_i)(e^{c_i} - 1) && \forall i \\
&\sum_i G_i \geq \sum_i \left(\frac{S_i^2}{2} - \frac{S_i^3}{6} \right) \\
&\sum_i R_i \geq \sum_i (1 - w_i)(o_i - c_i) \\
&\sum_i S_i \geq \sum_i (1 - w_i)(o_i - c_i).
\end{aligned}$$

The variables are $B_i, G_i, P_i, R_i, S_i, w_i, c_i, o_i$, all in $[0, 1]$ (and the number of offline vertices n).

THEOREM 1. (THEOREM 1.1) *The competitive ratio of `SemiAdaptive` is lower-bounded by 0.534.*

Proof. To solve the mathematical program, rewrite it more simply:

$$\begin{aligned}
&\min \quad \frac{\sum_i w_i}{\sum_i o_i} \\
&\text{s.t.} \\
&\sum_i w_i \geq \sum_i (1 - w_i)(e^{c_i} - 1) \\
&\qquad + \left(\sum_i R_i \right) + \left(\sum_i \frac{S_i^2}{2} \right) - \left(\sum_i \frac{S_i^3}{6} \right) \\
&\sum_i R_i \geq \sum_i (1 - w_i)(o_i - c_i) \\
&\sum_i S_i \geq \sum_i (1 - w_i)(o_i - c_i).
\end{aligned}$$

We have that each $S_i \in [0, 1]$ because $S_i \leq B_i$ by definition, and $B_i \leq w_i \leq 1$. So by convexity of $\frac{x^2}{2} - \frac{x^3}{6}$ on $[0, 1]$, letting $\Delta = \frac{1}{n} \sum_i S_i$,

$$\left(\sum_i \frac{S_i^2}{2} \right) - \left(\sum_i \frac{S_i^3}{6} \right) \geq n \left(\frac{\Delta^2}{2} - \frac{\Delta^3}{6} \right).$$

So rewrite:

$$\begin{aligned}
&\min \quad \frac{\sum_i w_i}{\sum_i o_i} \\
&\text{s.t.} \\
&\sum_i w_i \geq \sum_i (1 - w_i)(e^{c_i} - 1) \\
&\qquad + \left(\sum_i R_i \right) + n \left(\frac{\Delta^2}{2} - \frac{\Delta^3}{6} \right) \\
&\sum_i R_i \geq \sum_i (1 - w_i)(o_i - c_i) \\
&\Delta \geq \frac{1}{n} \sum_i (1 - w_i)(o_i - c_i).
\end{aligned}$$

At the optimum, the constraints on the sum of R_is and Δ are tight (otherwise, we could improve the solution). Therefore, we can assume without loss of value in the solution that

$$\Delta = \frac{1}{n} \sum_i R_i = \frac{1}{n} \sum_i (1 - w_i)(o_i - c_i).$$

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

This gives the simpler problem

$$\begin{aligned} \min \quad & \frac{\sum_i w_i}{\sum_i o_i} \\ \text{s.t.} \quad & \\ \sum_i w_i \geq & \sum_i (1-w_i)(e^{c_i}-1) \\ & + n\left(\Delta + \frac{\Delta^2}{2} - \frac{\Delta^3}{6}\right) \\ \Delta = & \frac{1}{n}\sum_i (1-w_i)(o_i - c_i). \end{aligned}$$

We have $o_i, w_i, \Delta \in [0,1]$ and $c_i \in [0, o_i]$. Now, we can also add the following constraint:

$$w_i \geq 1 - e^{-c_i},$$

because a total mass of at least c_i is assigned to i. (Formally, this constraint follows directly from $w_i \geq P_i$ and Lemma 4.2.)

Solution strategy. We divide the offline vertices into two categories. In Category 1, all i have $w_i = 1$. It is immediate from the optimization problem that if $w_i = 1$, then it is optimal to set $o_i = 1$ (since this does not affect any constraint) and the value of c_i is arbitrary within its bounds (for the same reason).

Category 2 contains all other offline vertices. We will show that it is without loss of optimality to suppose that, in Category 2, all i have $c_i = c^*$ for some fixed c^*, $w_i = w^*$ for the fixed value $w^* = 1 - e^{-c^*}$, and $o_i = 1$. With this, letting $\alpha \in [0,1]$ be the fraction of offline vertices in Category 1, the minimization problem becomes

$$\begin{aligned} \min \quad & \alpha + (1-\alpha)w^* \\ \text{s.t.} \quad & \\ \alpha + (1-\alpha)w^* \geq & (1-\alpha)(1-w^*)(e^{c^*}-1) \\ & + \Delta + \frac{\Delta^2}{2} - \frac{\Delta^3}{6} \\ \Delta = & (1-\alpha)(1-w^*)(1-c^*) \\ w^* = & 1 - e^{-c^*}. \end{aligned}$$

Simplifying [note that $(1-w^*)(e^{c^*}-1) = e^{-c^*}(e^{c^*}-1) = w^*$], we get

$$\begin{aligned} \min \quad & \alpha + (1-\alpha)(1-e^{-c^*}) \\ \text{s.t.} \quad & \\ \alpha \geq & \Delta + \frac{\Delta^2}{2} - \frac{\Delta^3}{6} \\ \Delta = & (1-\alpha)e^{-c^*}(1-c^*). \end{aligned}$$

This problem in two variables, α and c^*, is solved to yield an objective value of 0.53480... with $\alpha \approx 0.401$, $c^* \approx 0.252$, and $w^* \approx 0.223$. This solution can be found by a solver or, for instance, checking all feasible $(\alpha, c^*) \in [0,1]^2$, since the partial derivatives of the objective are bounded by 1 in absolute value, meaning that at a discretization level of ϵ the output is at most 2ϵ above the true minimum.

Remainder of solution. It remains to show that Category 2, all offline vertices where $w_i \neq 1$, contains (without loss of optimality) only vertices with: $c_i = c^*$ for some fixed c^*, $w_i = w^* = 1 - e^{-c^*}$, and $o_i = 1$.

The steps we take will be:

1. Show that, in Category 2, all $c_i = c^*$ for some fixed c^*.
2. Show that, without loss, all w_i are at an extreme (as small or as large as possible), with at most one exception. Since those w_i at the upper extreme have $w_i = 1$ and are in Category 1 by definition, this leaves all vertices in Category 2 at their minimum, $w^* = 1 - e^{-c^*}$, with possibly one exception.
3. Show that this exception does not exist, and show that without loss all $o_i = o^*$ in Category 2.
4. Show that $o^* = 1$.

This gives the above optimization problem and solution.

Step 1. First, we show that in Category 2, all $c_i = c^*$ for some c^*.

At an optimum solution the partial derivative of the Lagrangian with respect to c_i must be zero. The Lagrangian is

$$\begin{aligned} L = \frac{\sum_i w_i}{\sum_i o_i} \; + \; \lambda \Bigg(& -\sum_i w_i \; + \; \sum_i (1-w_i)(e^{c_i}-1) \\ & + n\left(\Delta + \frac{\Delta^2}{2} - \frac{\Delta^3}{6}\right)\Bigg) \end{aligned}$$

where

$$\Delta = \frac{1}{n}\sum_i (1-w_i)(o_i - c_i).$$

Thus,

$$\begin{aligned} \frac{\partial L}{\partial c_i} &= \lambda(1-w_i)e^{c_i} + \lambda\left(1 + \Delta - \frac{\Delta^2}{2}\right)(1-w_i) \\ &= \lambda(1-w_i)\left(e^{c_i} - \left(1 + \Delta - \frac{\Delta^2}{2}\right)\right). \end{aligned}$$

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Therefore, for $w_i < 1$ (it is easy to check that $\lambda > 0$ since $\frac{\partial L}{\partial w_i} = 0$), at an optimal solution

$$c_i = \ln\left(1 + \Delta - \frac{\Delta^2}{2}\right) = c^*.$$

Step 2. Now we show that, without loss, all w_i are at an extremum: $w_i = 1$ or $w_i = 1 - e^{-c_i}$. To see this, let the right side of the constraint be

$$f = \sum_i (1 - w_i)(e^{c_i} - 1) \; + n\left(\Delta + \frac{\Delta^2}{2} - \frac{\Delta^3}{6}\right).$$

Then

$$\frac{df}{dw_i} = -(e^{c_i} - 1) - \left(1 + \Delta - \frac{\Delta^2}{2}\right)(o_i - c_i).$$

The exact form is not important to us here. The only point is that we can take any i and i' with $\frac{df}{dw_i} \le \frac{df}{dw_{i'}}$. If we increase w_i by ϵ and decrease $w_{i'}$ by ϵ, then the objective stays constant and the constraint remains feasible (perhaps some slack was introduced), so we have not lost optimality. This operation was not possible only if w_i was already at its lower bound, $1 - e^{-c_i}$, or $w_{i'}$ was already at its upper bound, 1. So it is without loss to suppose that, in any optimal solution, all w_i are at an extremum — except possibly for one i. However, this possibility will be eliminated shortly.

This shows that without loss all offline vertices (except for one exception) in Category 2 have $w_i = 1 - e^{-c_i} = 1 - e^{-c^*} = w^*$.

Step 3. Now we eliminate the exception; then we show that all $o_i = o^*$ for some fixed o^*.

$$\frac{df}{do_i} = \left(1 + \Delta - \frac{\Delta^2}{2}\right)(1 - w_i).$$

Suppose the exceptional i exists, satisfying $w_i > w^*$. Then $\frac{df}{do_i} < \frac{df}{do_{i'}}$ for all $i' \ne i$. So we can decrease o_i by ϵ and increase some other $o_{i'}$ by an appropriate amount in order to maintain feasibility and decrease the value of the objective. This argument fails only if $o_i = c^*$ (its minimum) or all other $o_{i'} = 1$. If the $o_i = c^*$, then since $w_i > 1 - e^{-c^*}$, we have $\frac{w_i}{o_i} > (1 - e^{-c^*})/c^* \ge 1 - \frac{1}{e}$, which is a contradiction to the assumption that we are at an optimum: The optimum will be less than $1 - \frac{1}{e}$, so removing i entirely improves the solution. Thus, if the exception exists, we must have all other $o_{i'} = 1$ so that o_i cannot be lowered in this way. But note that $o_i \le o_{i'}$, while $c_i = c_{i'} = c^*$. If $o_i < o_{i'}$, then we have $\frac{df}{dw_i} > \frac{df}{dw_{i'}}$, so we could decrease w_i and increase $w_{i'}$ without changing the objective and introduce slack in the constraint; but then the solution could be improved and we are not at optimum. This argument fails only if $o_i = 1$ or w_i is at its minimum, $w_i = 1 - e^{-c^*}$; but in the latter case it is not an exception after all.

Now, we have in Category 2 all vertices having $c_i = c^*$, $w_i = w^* = 1 - e^{-c^*}$, and $o_i = o^*$ for some choices of c^* and o^*. We now show that $o^* = 1$.

Note that the derivatives $\frac{df}{do_i}$ are all equal and do not depend on the value of o_i (in fact, by plugging in $w_i = w^*$, they are equal to 1). Therefore, we can without loss suppose from now on that all o_i are equal to some o^*. (This is because, if they are not all equal, we can increase the smallest and decrease the largest without changing the objective or the constraint; repeat until they are equal.)

Step 4. We now have: An α-fraction of vertices in Category 1 with $w_i = o_i = 1$, and a $(1-\alpha)$-fraction in Category 2 with $c_i = c^*, w_i = w^* = 1 - e^{-c^*}$, and $o_i = o^*$. Thus, the problem is

$$\begin{aligned} \min \quad & \frac{\alpha + (1-\alpha)w^*}{\alpha + (1-\alpha)o^*} \\ \text{s.t.} \quad & \\ & \alpha \ge \Delta + \frac{\Delta^2}{2} - \frac{\Delta^3}{6} \\ & \Delta = (1-\alpha)(1-w^*)(o^* - c^*). \end{aligned}$$

We now show that $o^* = 1$ at optimum. Let $f = \Delta + \frac{\Delta^2}{2} - \frac{\Delta^3}{6}$; then

$$\begin{aligned} \frac{df}{do^*} &= \left(1 + \Delta - \frac{\Delta^2}{2}\right)(1-\alpha)(1-w^*) \\ &= 1 - \alpha \end{aligned}$$

since $c^* = \ln\left(1 + \Delta - \frac{\Delta^2}{2}\right)$ and $w^* = 1 - e^{-c^*}$. Meanwhile,

$$\begin{aligned} \frac{df}{d\alpha} &= -\left(1 + \Delta - \frac{\Delta^2}{2}\right)(1-w^*)(o^* - c^*) \\ &= -(o^* - c^*). \end{aligned}$$

Thus, changing o^* by ϵ and α by $\epsilon\frac{1-\alpha}{1+o^*-c^*}$ maintains a feasible solution (that is, $\alpha \ge f$). The change in the numerator of the objective, $\alpha + (1-\alpha)w^*$, is $\epsilon\frac{1-\alpha}{1+o^*-c^*}e^{-c^*}$. The change in the denominator, $\alpha + (1-\alpha)o^*$, is

$$\begin{aligned} & \epsilon\frac{1-\alpha}{1+o^*-c^*} + (1-\alpha)\epsilon - \epsilon\frac{1-\alpha}{1+o^*-c^*}o^* \\ &= \epsilon(1-\alpha)\left(1 + \frac{1-o^*}{1+o^*-c^*}\right). \end{aligned}$$

The ratio of change in numerator to change in denominator is $\frac{e^{-c^*}}{2-c^*} < 0.5$ for all $c^* \in (0, 1]$ (and

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

we know $c^* > 0$ at an optimum, else $\Delta > 0$ and we get a contradiction from the definition of c^*).

Now, since we were at optimum, we had some competitive ratio $\frac{A}{B} \geq 0.5$ (we know the optimum is at least 0.5 because by removing some constraints we recover the analysis of `NonAdaptive`, which obtained 0.5). And it may be checked that $\frac{A+dA}{B+dB} < \frac{A}{B}$ if $\frac{dA}{dB} < \frac{A}{B}$. This is the case here, with $\frac{dA}{dB} = \frac{e^{-c^*}}{2-c^*}$. Thus, our changes to α and o^* resulted in changes to the numerator and denominator that lowered the overall ratio. This contradicts the assumption that we were at optimum, so either α or o^* cannot be raised in the optimum. If it were the case for α, then $\alpha = 1$ and the optimum would be 1, so it must be the case for o^*, that is, $o^* = 1$.

This completes the four steps promised, giving the minimization problem described above, which is solved to an optimum ratio of 0.53480....

6 Computing Updates for the Algorithm

To execute `SemiAdaptive`, we need to be able to compute $w_i(t)$ at each time t, given all arrivals up until t. Here, we show that $w_i(t)$ can be computed via constant-time updates for each arrival. To do so, we will need to maintain the following variables at each time t (note that we only need the values at the current time, and do not need to store past values):

- $w_i(t)$ = the probability that i has succeeded by time t, for each offline vertex i.
- $wB_i(t)$ = Pr[FirstSucc(i) = `True` at time t]; that is, the probability that i "would have succeeded from a first choice" by time t, for each offline vertex i.
- $wG_{i'i}(t)$, which intuitively is the probability that i "would have succeeded from a second choice whose first choice was i'", if i did not succeed in any other way, for each pair of offline vertices i' and i. Specifically, $wG_{i'i}(t)$ is the probability that, at some time $t_1 < t$, FirstSucc(i') was set to `True`; and, when flipping a p_{ij}-weighted coin for all arrivals j at time $t_j \in (t_1, t)$ with first choice i' and second choice i, at least one of them comes up heads. (A formal definition for $wG_{i'i}(t)$ appears later in this section.)

Before describing the constant-time update rules, a brief note on memory. In the theoretical worst case, the total memory required is $O(n^2)$ where n is the number of offline vertices, since $wG_{ii'}(t)$ is maintained for all pairs i, i'. But in applications such as AdWords, we expect the memory requirement to be smaller: we need to maintain only those $wG_{ii'}$ where i is some arrival's first choice and i' is some arrival's second choice. This indicates that i and i' are both highly relevant advertisers for the same query; the set of such pairs may be quite sparse. Also, in practice we might expect relatively few advertisers n as compared to the number of online arrivals.

The updates are governed by the following relation: The probability that i has not yet succeeded at time t is

$$\begin{aligned}
&1 - w_i(t) \\
&= \prod_{j:t_j<t} \Pr\left[j \text{ fails to take } i \mid i \text{ is available at time } t_j\right] \\
&= \prod_{j:t_j<t, 1_j=i} (1 - p_{ij}) \\
&\quad \cdot \prod_{j:t_j<t, 1_j=i' \neq i} \Pr[j \text{ fails to take } i \mid i \text{ available at } t_j] \\
&= (1 - wB_i(t)) \prod_{i' \neq i} (1 - wG_{i'i}(t)) \,.
\end{aligned} \tag{6.7}$$

Now, suppose some j arrives at time t_j with first choice 1 and second choice 2. Then immediately, for $t > t_j$, we have that the probability 1 has succeeded is the probability it has already succeeded plus the probability that it has not, but it succeeds on the edge from j; similarly for the probability of FirstSucc(1) being set to `True`:

$$\begin{aligned}
w_1(t) &= w_1(t_j) + (1 - w_1(t_j))p_{1j} \\
wB_1(t) &= wB_1(t_j) + (1 - wB_1(t_j))p_{1j}.
\end{aligned}$$

(It can be verified that this update maintains Equation 6.7.) Meanwhile, for all $i \neq 1$, $wB_i(t)$ is unchanged; for all pairs $i'i \neq 12$, $wG_{i'i}(t)$ is unchanged; and for all $i \neq 1, 2$, $w_i(t)$ is unchanged. So we only need to determine the update to $wG_{12}(t)$ and to $w_2(t)$.

In fact, the update rule for $wG_{12}(t)$ is

$$wG_{12}(t) = p_{2j} wB_1(t_j) + (1 - p_{2j}) wG_{12}(t_j).$$

This can be shown in two ways: By a "counting" argument, and by arithmetic manipulation.

For the first method, for convenience let the event tracked by $wG_{12}(t)$ be called SecSucc(12), so that $wG_{12}(t)$ = Pr[SecSucc(12) = `True` by time t]. Consider the edge between 2 and j. Regardless of whether j is actually assigned to 2 during the execution of the algorithm, we can flip a p_{2j}-weighted coin to determine if this edge "would have" succeeded. So, suppose we flip the coin. If it comes up heads (with probability p_{2j}), *and* we have that FirstSucc(1) = `True` by time t_j, then it must be that SecSucc(12) = `True` occurs at or before t_j. (It may occur before t_j

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

due to some other arrival, but even if it does not, the edge between 2 and j is realized, so it occurs at time t_j.) This gives the contribution $p_{2j} w B_1(t_j)$ from the event that the coin comes up heads.

If the coin comes up tails (with probability $1-p_{2j}$), then the probability of SecSucc(12) = `True` occurring by time $t > t_j$ is exactly the chance that it occurred by time t_j, since if it occurs then it occurs by some arrival prior to j. This gives the contribution $(1-p_{2j}) w G_{12}(t_j)$ from the event that the coin comes up tails, which proves the update rule.

For those who prefer an arithmetic proof: Denote the arrivals before time t whose first choice is 1 and second choice is 2 by, in order, $j_1, \ldots, j_k$. For convenience, where t_{j_0} would appear in the following, let $t_{j_0} = 0$. Then we have

$$
\begin{aligned}
&1 - wG_{12}(t) \\
&= 1 - wB_1(t_{j_k}) + \sum_{l=1}^{k} \Pr[\text{1 succeeded at } t \in [t_{j_{l-1}}, t_{j_l})] \\
&\qquad \cdot \Pr[\text{each of } j_l, \ldots, j_k \text{ failed at 2}] \\
&= 1 - wB_1(t_{j_k}) \\
&\qquad + \sum_{l=1}^{k} \left[wB_1(t_{j_l}) - wB_1(t_{j_{l-1}}) \right] \prod_{r=l}^{k} (1 - p_{2j_r}) \,.
\end{aligned}
$$

This equality can be justified as follows. The probability that SecSucc(12) = `True` has not occurred is broken down according to the time at which FirstSucc(1) is set to `True`. With probability $1 - wB_1(t_{j_k})$, it does not occur (or only occurs after the last useful arrival has arrived), and SecSucc(12) = `True` certainly does not occur. Otherwise, FirstSucc(1) = `True` can occur in some interval between two of our arrivals, call them j_{l-1} and j_l (where, if $l = 1$, then j_{l-1} is not really an arrival but merely represents the start of the algorithm). The probability that FirstSucc(1) is set to `True` in this interval is $wB_1(t_{j_l}) - wB_1(t_{j_{l-1}})$. Given that it occurs in this interval, the probability that SecSucc(12) = `True` still fails to occur is the probability that the remainder of our arrivals, from j_l to j_k, all fail on their edges to 2.

With this formula in hand, consider the update for time $t > t_{j_k}$ when j_k arrives. After j_k arrives, we have

$$
\begin{aligned}
&wG_{12}(t) \\
&= wB_1(t_{j_k}) - \sum_{l=1}^{k} \left[wB_1(t_{j_l}) - wB_1(t_{j_{l-1}}) \right] \prod_{r=l}^{k} (1 - p_{2j_r}) \\
&= wB_1(t_{j_k}) - \left[wB_1(t_{j_k}) - wB_1(t_{j_{k-1}}) \right] (1 - p_{2j_k}) \\
&\qquad - \sum_{l=1}^{k-1} \left[wB_1(t_{j_l}) - wB_1(t_{j_{l-1}}) \right] \prod_{r=l}^{k} (1 - p_{2j_r}) \\
&= p_{2j_k} wB_1(t_{j_k}) \; + (1 - p_{2j_k}) wB_1(t_{j_{k-1}}) \\
&\qquad - \sum_{l=1}^{k-1} \left[wB_1(t_{j_l}) - wB_1(t_{j_{l-1}}) \right] \prod_{r=l}^{k} (1 - p_{2j_r}) \\
&= p_{2j_k} wB_1(t_{j_k}) \; + (1 - p_{2j_k}) \Bigg(wB_1(t_{j_{k-1}}) \\
&\qquad - \sum_{l=1}^{k-1} \left[wB_1(t_{j_l}) - wB_1(t_{j_{l-1}}) \right] \prod_{r=l}^{k-1} (1 - p_{2j_r}) \Bigg) \\
&= p_{2j_k} wB_1(t_{j_k}) + (1 - p_{2j_k}) wG_{12}(t_{j_{k-1}}).
\end{aligned}
$$

Now that we have the update for $wG_{12}(t)$, only the update for $w_2(t)$ remains. Since in Equation 6.7 only $wG_{12}(t)$ has changed, this can be accomplished in $O(1)$ steps by taking $1 - w_2(t_j)$, dividing by $1 - wG_{12}(t_j)$, and multiplying by the new value $1 - wG_{12}(t)$ to obtain the new value $1 - w_2(t)$.

7 Discussion and Future Work

In this paper, a competitive ratio of 0.534 is achieved for the OnlineStochasticMatching problem for the general case of unequal (but vanishing) probabilities. This result required a new algorithmic approach that we hope may be useful in other online or stochastic settings. The key feature is to maintain *non-adaptive state* about the probabilities of success of each vertex, and to make *adaptive decisions* based on that state. This approach allowed us to analyze `SemiAdaptive`, which achieves the above ratio. The other intuition behind `SemiAdaptive` is to make choices so as to maximize the marginal probability of success. We saw that this principle enabled `NonAdaptive` to achieve a ratio of 0.5, which is optimal for non-adaptive algorithms.

The OnlineStochasticMatching problem is young and many open questions remain. Even for the case of equal probabilities, the optimal achievable competitive ratio is unknown, lying somewhere between 0.567 and 0.621 for vanishing probabilities. For improving on the competitive ratio shown in this paper, `FullyAdaptive` seems to be a strong candidate algorithm, but it has eluded analysis thus far.

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 10/17/22 to 185.238.250.128 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

References

[1] Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *SODA*, pages 1253–1264, 2011.

[2] Bahman Bahmani and Michael Kapralov. Improved bounds for online stochastic matching. In *ESA (1)*, pages 170–181, 2010.

[3] Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra. When lp is the cure for your matching woes: Improved bounds for stochastic matchings - (extended abstract). In *ESA (2)*, pages 218–229, 2010.

[4] Anand Bhalgat, Ashish Goel, and Sanjeev Khanna. Improved approximation results for stochastic knapsack problems. In *SODA*, pages 1647–1665, 2011.

[5] Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *ESA*, pages 253–264, 2007.

[6] Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra. Approximating matches made in heaven. In *ICALP (1)*, pages 266–278, 2009.

[7] Brian C. Dean, Michel X. Goemans, and Jan Vondrák. Adaptivity and approximation for stochastic packing problems. In *SODA*, pages 395–404, 2005.

[8] Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *ACM Conference on Electronic Commerce*, pages 71–78, 2009.

[9] Jon Feldman, Nitish Korula, Vahab S. Mirrokni, S. Muthukrishnan, and Martin Pál. Online ad assignment with free disposal. In *WINE*, pages 374–385, 2009.

[10] Jon Feldman, Aranyak Mehta, Vahab S. Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e. In *FOCS*, pages 117–126, 2009.

[11] Gagan Goel, Afshin Nikzad, and Adish Singla. Allocating tasks to workers with matching constraints: Truthful mechanisms for crowdsourcing markets. In *Proc. International World Wide Web Conference (WWW)*, 2014.

[12] Michel X. Goemans and Jan Vondrák. Stochastic covering and adaptivity. In *LATIN*, pages 532–543, 2006.

[13] Anupam Gupta, Ravishankar Krishnaswamy, Marco Molinaro, and R. Ravi. Approximation algorithms for correlated knapsacks and non-martingale bandits. In *FOCS*, 2011.

[14] Chien-Ju Ho and Jennifer Wortman Vaughan. Online task assignment in crowdsourcing markets. In *AAAI*, 2012.

[15] Bala Kalyanasundaram and Kirk Pruhs. An optimal deterministic algorithm for online b-matching. *Theor. Comput. Sci.*, 233(1-2):319–325, 2000.

[16] Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *STOC*, pages 587–596, 2011.

[17] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on theory of computing*, pages 352–358. ACM, 1990.

[18] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC*, pages 352–358, 1990.

[19] Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *Algorithms–ESA 2013*, pages 589–600. Springer, 2013.

[20] Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *ICALP (2)*, pages 508–520, 2009.

[21] Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *STOC*, pages 597–606, 2011.

[22] Vahideh H. Manshadi, Shayan Oveis Gharan, and Amin Saberi. Online stochastic matching: Online actions based on offline statistics. In *SODA*, pages 1285–1294, 2011.

[23] Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.

[24] Aranyak Mehta and Debmalya Panigrahi. Online matching with stochastic rewards. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science*, pages 728–737. IEEE, 2012.

[25] Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5), 2007.

Copyright © 2015.
by the Society for Industrial and Applied Mathematics.