

Know When to Persist: Deriving Value from a Stream Buffer ^{*}

Konstantinos Georgiou¹, George Karakostas², Evangelos Kranakis^{3**}, and Danny Krizanc⁴

¹ Department of Mathematics, Ryerson University

² Dept. of Computing & Software, McMaster University

³ School of Computer Science, Carleton University

⁴ Department of Mathematics & Computer Science, Wesleyan University

Abstract. We consider PERSISTENCE, a new online problem concerning optimizing weighted observations in a stream of data when the observer has limited buffer capacity. A stream of weighted items arrive one at a time at the entrance of a buffer with two holding locations. A processor (or observer) can process (observe) an item at the buffer location it chooses, deriving this way the weight of the observed item as profit. The main constraint is that the processor can only move *synchronously* with the item stream; as a result, moving from the end of the buffer to the entrance, it crosses paths with the item already there, and will never have the chance to process or even identify it. PERSISTENCE is the online problem of scheduling the processor movements through the buffer so that its total derived value is maximized under this constraint. We study the performance of the straight-forward heuristic *Threshold*, i.e., forcing the processor to "follow" an item through the whole buffer only if its value is above a threshold. We analyze both the optimal offline and Threshold algorithms in case the input stream is a random permutation, or its items are iid valued. We show that in both cases the competitive ratio achieved by the Threshold algorithm is at least $2/3$ when the only statistical knowledge of the items is the median of all possible values. We generalize our results by showing that Threshold, equipped with some minimal statistical advice about the input, achieves competitive ratios in the whole spectrum between $2/3$ and 1 , following the variation of a newly defined density-like measure of the input. This result is a significant improvement over the case of arbitrary input streams, since in this case we show that no online algorithm can achieve a competitive ratio better than $1/2$.

1 Introduction

Suppose that the Automated Quality Control (AQC) of an assembly line has the ability to check all new parts as they enter the assembly line. Every such check increases our quality confidence by a certain percentage, which depends on the nature of the part itself. Now, suppose that the AQC is given the option of a second look at the same part in the next time slot, with a similar increase in our quality confidence. The downside of this option, is that when the AQC returns to the beginning of the assembly line, it will have completely missed the part immediately following the one that was double-checked. We are looking for an algorithm to decide whether to take the option or not with every new item. Obviously, a good strategy would strive to look twice at "low-quality" items, since that would imply the greatest increases to our confidence, while "missing" only pristine-looking ones.

This problem falls within the data stream setting: a sequence of input data is arriving at a very high rate, but the processing unit has limited memory to store and process the input. Data stream algorithms have been explored extensively in the computer science literature. Typical algorithms in this area work with only a few passes (often just one) over the data input and use memory space

^{*} An extended abstract of this paper appeared in the LNCS Springer Proceedings of the 11th International Conference on Algorithmic Aspects of Information and Management (AAIM 2016), July 18–20 2016, Bergamo, Italy.

^{**} Research supported in part by NSERC Discovery grant.

less than linear in the input size. Applications can be found in processing cell phone calls or Internet router data, executing Web searches, etc. (cf. [16, 17]).

In this work we study a new online problem in data stream processing with limited buffer capacity. An online stream of items (the parts in our AQC example) arrives (one item at a time) at a buffer with two locations L_0, L_1 (assembly points 1 and 2 respectively in the example above), staying at each location for one unit of time, in this order. A processor/observer (the AQC) can move between the two locations *synchronously*, i.e., its movements happen at the same time as the items move. This means that if the processor is processing (observing) the i -th item in time t at L_0 , moving to L_1 will result in processing again the i -th item at L_1 in time $t + 1$. On the contrary, if the processor is processing the i -th item in time t at L_1 , moving to L_0 will result in processing the $i + 2$ -th item at L_0 in time $t + 1$; the $i + 1$ -th item has already moved to L_1 and will leave the buffer without the processor ever encountering it! (just like the AQC totally missed a part). We emphasize that we restrict the processor to not even know what item it missed (i.e., cannot “see” into a location other than its current one). Processing the i -th item (either in L_0 or L_1) produces an added value or payoff. The processor has very limited (constant in our results) memory capacity, and cannot keep more than a few variables or pieces of data. The problem we address is whether such a primitive processor can have a strategy to persist and observe (if possible) mostly “good values”, especially when compared to an optimal algorithm that is aware of the input stream. We call this online problem PERSISTENCE, which to the best of our knowledge is also new.

Related Work: There is extensive literature on data stream algorithms. Here the emphasis is on input data arriving at a very high rate and limited memory to store and process the input (thus stressing a tradeoff between communication and computing infrastructure). A general introduction to data stream algorithms and models can be found in [16, 17]. Lower bound models for space complexity are elaborated in [3]. In the section on new directions for streaming models, [17] discusses several alternatives for data streams for permutation streaming of non-repeating items [1], windowed streaming whereby the most recent past is more important than the distant past [13], as well as reset model, distributed continuous computation, and synchronized streaming. Applications of data stream algorithms are explored extensively in the computer science literature and can be found in sampling (finding quantiles [12], frequent items [15], inverse distribution [7], and range-sums of items [2]).

Related to our study is the well-known *secretary problem* which appeared in the late 1950s and early 1960s (see [9] for a historical overview of its origins and [10] which discusses several extensions). It is concerned with the optimal strategy or stopping rule so as to maximize the probability of selecting the best job applicant assuming that the selection decision can be deferred to the end. Typically we are concerned with maximizing the probability of selecting the best job applicant; this can be solved by a maximum selection algorithm which tracks the running maximum. The problem has fostered the curiosity of numerous researchers and studied extensively in probability theory and decision theory. Several variants have appeared in the scientific literature, including on rank-based selection and cardinal payoffs [6], the infinite secretary problem in [11], secretary problem with uncertain employment in [18], the submodular secretary problem in [5], just to mention a few. The “secretary problem” paradigm has important applications in computer science of which it is worth mentioning the recent work of [4] which studies the relation of matroids, secretary problems, and online mechanisms, as well as [14] which is investigating applications of a multiple-choice secretary algorithm to online auctions. Obviously the secretary problem differs from PERSISTENCE in terms of the objective function: in our case the payoff is the sum of processing payoffs, as opposed to the

maximum for the secretary problem. The two problems also differ in the synchronicity and location of arrivals, i.e., what can be accessed and how it is accessed. Nevertheless, the two problems share the inherent difficulty of having to make decisions *on the spot* while missing parts of the input altogether.

1.1 High Level Summary of our Results & Outline of the Paper

Our primary focus is the study of the PERSISTENCE problem, which we formally define in Section 2.1. Our goal is to compare the performance of any primitive (online) algorithm, which is not aware of the input stream, against the optimal offline algorithm. In Section 2.2 we present all such possible primitive algorithms that we call *Threshold*. Subsequently, in Section 2.3 we analyze the performance of any Threshold online algorithm for deterministic input streams. Our findings indicate that simplistic primitive algorithms are actually optimal (among all online solutions), and are off no more than $1/2$ the performance of an optimal (offline) algorithm that is aware of the entire input. Similar to the setting of the secretary and other online decision problems, this motivates the study of PERSISTENCE problems when the input is random, which is also our main focus.

Our main contributions are discussed in detail in Section 2.3. At a high level, we show that when the online observer (processor) knows the median of the possible random values that can appear in the input stream, then it is possible to perform observations in a way such that the total payoff is asymptotically at least $2/3$ of the optimal offline solution (Theorem 1). Moreover, we prove that when the random input streams come from certain natural families of inputs in which the mass of possible values is concentrated in relatively few heavy items, the asymptotic performance of very primitive algorithms is nearly optimal. In fact, we parameterize the performance of online algorithms for such inputs using a proper density measure, and we show how the relative asymptotic performance changes from almost optimal (competitive ratio almost 1) to competitive ratio $2/3$ (Theorem 2).

The results discussed above are just the byproduct of our main technical contributions that pertain to an analytic exposition of the performance of optimal offline and any online algorithm for random inputs, parameterized by a proper statistical density-like measure on inputs. The two random models that we study are input streams that are either random permutations (Section 3) or input streams whose elements assume independent and identically distributed values (Section 4). In each case we provide closed formulas for the performance of the optimal offline algorithm and any online algorithm (Sections 3.1 and 4.1 respectively), which we think is interesting in its own right. Then we use the closed formulas to derive the promised asymptotic competitive analysis in Sections 3.2 and 4.2 respectively.

We emphasize that the analysis of a size-2 buffer we provide is technically involved, and we cannot see how it could be extended to larger buffers without considerable extra effort. But even for this restricted case, the problem is interesting. Indeed, given our model of algorithms allowed (streaming algorithms with a constant-size memory that can keep only a few variable values, i.e., memoryless), the fact that the simple threshold algorithm achieves non-trivial improvements is already a rather surprising result.

2 Preliminaries

2.1 Model & Problem Definition

Assume that n incoming data values $v_1, v_2, \dots, v_{n-1}, v_n$ arrive sequentially and synchronously from the left one at a time at a processing unit consisting of two registers L_0 and L_1 which are capable of storing these values instantaneously (see also Figure 1). The values pass first through location (register) L_0 and then through location L_1 , before exiting. A processing unit can process (i.e., derive some payoff from or contribute some additive value to) an item either in L_0 or L_1 . The value v_i derived by processing item i comes from a set of possible values $a_0 < a_1 < \dots < a_{k-1}$, and is independent of the location that processing happened. The main constraint is that all processing is *synchronous*, i.e., at every time unit exactly one new item enters L_0 and the processor (observer) is allowed to either do some processing (observe) at the location it's already in, or perform a single move (and then do processing in) to the other location. The other important constraint is the fact that the processor has only a constant-size memory (i.e., it has space to hold at most $O(1)$ variables) as well as it is only aware of the value of the register of its location. In particular, when processor is located at one register, it is *oblivious* to the value of the other register.

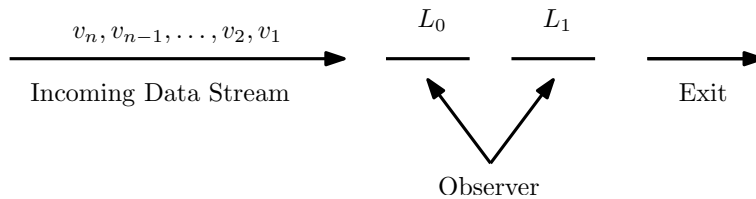


Fig. 1. Incoming values arriving sequentially and synchronously from the left one at a time and occupying first location L_0 and then location L_1 . An observer (processor) can look at only one of these two locations at a time. Data is exiting synchronously from the right.

More formally, our model is the following:

1. At time step $t = 1$, the processor (observer) occupies position L_0 , which holds value v_1 .
2. At time step $t \geq 2$, the following take place in that order:
 - The processor may change the location it is about to process (observe); at the same time, locations L_0, L_1 get (new) values v_t, v_{t-1} respectively.
 - Processing is done at the location of the processor; the added value achieved at t is the value of the item in that location.

In the *online model* the observer is *not* aware of the sequence $v_1, v_2, \dots, v_{n-1}, v_n$, rather she may only know some statistical information that requires constant memory. The limited memory implies a limited ability of keeping statistics or historical data, and, therefore, there is not much leeway for sophisticated processing policies. The (possible) movement of the observer can be determined exclusively by the current value she is observing and in particular not by the value of the location that the observer is not occupying. As a result, the only power an online algorithm has is to choose to observe a value twice in two consecutive time steps, if she thinks that this value provides high enough reward. In contrast, and in the *offline model*, the observer is aware of the entire sequence

$v_1, v_2, \dots, v_{n-1}, v_n$ in advance, and may choose to move between registers with no restrictions so as to maximize her total payoff.

Our main goal is to design PERSISTENCE strategies for the observer that maximize the total added value (or, equivalently, the *average* or *relative* added value or payoff). Our focus is to understand how the lack of information affects the performance of an oblivious online algorithm, compared to the optimal offline algorithm. The standard performance measurement that we use is the so-called *competitive ratio*, defined as the (worst case) ratio between the (expected - when the input stream is random) payoffs of an online and the optimal offline algorithm. It is immediate that for any input stream (even random), the competitive ratio of a fixed online algorithm is $ALG/OPT < 1$, where ALG, OPT are the (expected) payoff of the online and the optimal offline algorithm respectively.

2.2 On Persistence Strategies

Given an input stream $v_1, v_2, \dots, v_{n-1}, v_n$, the optimal solution for the offline model is straightforward; If the processor (observer) is in L_0 , processing (observing) an item i with value v_i , then it moves to L_1 only if the item that follows i has a value smaller than v_i ; If the processor is in L_1 , processing an item i with value v_i , then it moves to L_0 only if the item $i + 2$ that will enter L_0 in the next round has a value v_{i+2} greater than the value v_{i+1} of the item $i + 1$ currently in L_0 . As a result, an offline and optimal observer may choose to always occupy the location (and subsequently obtain its value as a reward) that holds the maximum value that currently appears in the two locations L_0, L_1 . Since at any step, an algorithm cannot have payoff more than the maximum value of the two registers, we conclude that

Observation 1 *For input stream $v_1, v_2, \dots, v_{n-1}, v_n$, and at each time step $t = 2, \dots, n$, the optimal solution of an offline algorithm incurs payoff equal to $\max\{v_t, v_{t-1}\}$.*

We will invoke Observation 1 later, when we will derive closed formulas for the performance of the optimal offline algorithm when the values of the input stream come from certain distributions.

Now we turn our attention to PERSISTENCE strategies in the online model. Recall that any online algorithm is oblivious, non-adaptive and with limited memory. In particular, when at register L_0 , an observer has the option to process the same value for one more time in the next step, or stay put at the register and watch in the next step the (currently unknown) value which will enter L_0 . If the observer is at register L_1 , then the possible payoff at the next step is unknown independently of the move of the observer. Hence it is natural to move the observer back to L_0 , giving her the option (in the future) to observe favorable values more than once. This primitive idea gives rise to the following *threshold algorithms*, which are determined by a choice of threshold that dictates when a register value will be observed twice in case the observer is at register L_0 .

Threshold Algorithm(T)

Input: a sequence of n items with values $v_1, v_2, \dots, v_{n-1}, v_n$

1. When the processor has finished processing an item of value τ_0 at L_0 then
 - 1a. **if** $\tau_0 \geq T$ **then** move to L_1
 - 1b. **if** $\tau_0 < T$ **then** stay at L_0
 2. When the processor has finished processing an item at L_1 **then** move to L_0
-

Our main contribution in subsequent sections is the (competitive) analysis of Threshold algorithms for various choices of thresholds. In what follows we call the simplistic algorithm that doesn't move the processor from L_0 (or, equivalently, has a threshold greater than a_{k-1}) *Naive*.

2.3 General Input Streams

In its most general version, the input stream to PERSISTENCE is chosen by an unrestricted adversary. Here we demonstrate that the threshold algorithm cannot achieve a competitive ratio better than $1/2$. There are the following cases:

1. $a_{k-1} < T$ In this case, the processor stays always at L_0 , and, therefore, acquires the payoff for *each* item exactly *once*, for a total payoff of exactly $\sum_{i=1}^n v_i$. On the other hand, the optimal offline algorithm has the chance of acquiring the payoff of the largest-value items at most *twice* (by processing them in both processors), for a total payoff of, at most, $2 \sum_{i=1}^n v_i$. Hence the competitive ratio is at least $1/2$.
2. $a_{k-2} \leq T < a_{k-1}$. In this case, the threshold algorithm always gets the payoff of an item with value a_{k-1} twice, and exactly once the values of the other items it processes in L_0 (obviously it misses the items that follow immediately after the items of value a_{k-1} processed in both L_0 and L_1). It is clear that the optimal offline algorithm does the same. Therefore, the adversary will minimize this overlap between the threshold and optimal offline algorithms, by creating a sequence without value- a_{k-1} items; this is the same as the previous case, and the competitive ratio is at least $1/2$.
3. $T < a_{k-2}$ In this case, the adversary creates the sequence of items with values $a_{k-2}, a_{k-1}, a_{k-2}, a_{k-1}, \dots$. Then the relative (average) payoff for the threshold algorithm is a_{k-2} (the algorithm will always process the a_{k-2} items twice, missing the more valuable a_{k-1} items). The offline optimal algorithm will behave exactly in the opposite way, achieving a relative payoff a_{k-1} . Hence the competitive ratio is at most a_{k-2}/a_{k-1} , which can be made to be arbitrarily close to 0.

Therefore, the best threshold is any number greater than a_{k-1} , and the competitive ratio is at least $1/2$. An upper bound of almost $1/2$ for the ratio is achieved by the input sequence with values $a_{k-2}, a_{k-1}, a_{k-2}, a_{k-1}, \dots$ and $a_{k-2} \ll a_{k-1}$.

2.4 Competitive Analysis for Randomized Input Streams - A Summary of our Results

The fact that, for arbitrary input streams, the Threshold algorithm cannot do better than the $1/2$ competitive ratio of the Naive algorithms, shows that in order for the threshold algorithm to perform better, we need to restrict the input instances by making assumptions about the input stream. There are two assumptions that are common in online problems such as the secretary problem [10], or resource allocation problems [8]: One is the *IID* assumption, i.e., the value of each new item is drawn independently and uniformly from the set $\{a_1, a_2, \dots, a_{k-1}\}$. Another is the *random order* assumption, i.e., the input is a (uniformly) random permutation of n items, each with its own distinct value. In what follows, we study the threshold algorithm under these assumptions.

More formally, we study the following two random models of input streams $v_1, v_2, \dots, v_{n-1}, v_n$:

- *Random Permutations*: Input sequence stream is a random permutation of values $a_0 \leq \dots, \leq a_{n-1}$.
- *Independent and Identically Distributed Values*: Each v_i assumes the value a_j with probability p_j independently at random, where $j = 0, \leq k - 1$ (note that we allow that $n = \omega(k)$).

For both input families we assume that an online algorithm is oblivious, non-adaptive and with minimal memory, still we assume it has access in advance to some limited statistical information in order to determine a proper threshold. Our main technical contribution pertains to a detailed analysis of the performance of both the optimal offline and any Threshold online algorithm for any such random input. As a result we demonstrate that if the online algorithm knows the median of the set from which the input stream elements assume values, then the competitive ratio improves significantly.

Theorem 1. *For any random permutation or uniform iid input stream, the online Threshold algorithm that uses as threshold the median of the values $\{a_i\}_i$ has (asymptotic) competitive ratio $2/3$.*

We emphasize that Theorem 1 is the byproduct of analytic and closed formulas that we derive for optimal offline and Threshold online algorithms, when the input stream is a random permutation (Section 3) or iid (Section 4), and not necessarily uniform.

Next we ask whether it is possible for certain families of random inputs to achieve a competitive ratio better than $2/3$, and given that the online algorithm has access to some statistical information. Again, we answer this in the positive by studying generic families of instances parametrized by the relative weight of their largest values.

Definition 1 (c -dense input streams). *Consider a random input stream (in either the random permutation or the iid model) whose values are chosen from $\mathcal{A} = \{a_1, \dots, a_t\}$, with $a_i \leq a_{i+1}$. The input stream is called c -dense if the total weight of the largest $\lfloor ct \rfloor$ many values of \mathcal{A} , relative to the total weight of \mathcal{A} , is equal to $1 - c$, i.e. when*

$$1 - c = \frac{\sum_{i=t-\lfloor ct \rfloor+1}^t a_i}{\sum_{i=1}^t a_i} \quad (1)$$

Note that, although c cannot be greater than $1/2$ by definition, when c is 0 or 1, then the left hand-side of (1) is 1 and 0 respectively, while the right hand-side is 0 and 1 respectively. At the same time, the two sides have different monotonicity as c increases, and as such the notion of c -dense input streams is well defined. Our main contribution pertaining to the families of random inputs which are asymptotically c -dense, for some $c \in (0, 1/2]$, is the following.

Theorem 2. *For any random permutation or uniform iid c -dense input stream, the online Threshold algorithm that uses as threshold the $\lfloor c \cdot n \rfloor$ largest value of a_i 's has (asymptotic) competitive ratio $\frac{1}{2} \frac{2-c}{(1-c)(1+c)^2}$.*

Clearly, when c tends to 0, the performance of our Threshold algorithms is nearly optimal for c -dense input streams. Most notably, the worst configuration for such an input is when $c = 1/2$, inducing a competitive ratio equal to $2/3$ (and as already predicted by Theorem 1). The proof of Theorem 2 for random permutations and uniform iid inputs can be found in Sections 3 and 4 respectively. Any omitted proofs can be found in the Appendix.

3 Random Permutation Input Streams

In this section we study the special case of inputs that are a random permutation of n items with distinct values $a_0 < a_1 < \dots < a_{n-1}$ (with $n \geq 2$). First we find closed formulas for the performance of the optimal offline algorithm and any Threshold online algorithm for the PERSISTENCE problem, and then we conclude with the competitive analysis.

3.1 Performance of Offline and Online Algorithms for Random Permutations

Using Observation 1 we show in Appendix A.1 that

Theorem 3. *The relative expected payoff (asymptotic payoff per time step) of the optimal offline algorithm when the input is a random permutation is: $\frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} i \cdot a_i$.*

The main technical contribution of this section is the performance analysis of any Threshold online algorithm.

Theorem 4. *Let $k = k(n)$ be such that $\lim_{n \rightarrow \infty} \frac{k}{n} = c \in \Theta(1)$. Let also A^-, A^+ denote the summation of the smallest $n - k$ and largest k values respectively. Then the relative expected payoff of the Threshold algorithm (payoff per time step) when the threshold is $T := a_{n-k}$ is: $\frac{A^-}{1+c} + \frac{2A^+}{1+c}$.*

The remaining of the section is devoted in proving Theorem 4. We will need the following random variables: A_i denotes the profit of our algorithm from value a_i , or in other words, the contribution of a_i to the performance of the algorithm. Clearly, if $a_i \geq T$ then $A_i \in \{0, 2a_i\}$, and if $a_i < T$ then $A_i \in \{0, a_i\}$. $V_t \in \{a_i\}_{i=0, \dots, n-1}$ is the value that appears in position t of the (random) permutation, where position 1 is the value that will be read first ($t = 1, \dots, n$). Finally, O_i is the indicator random variable that equals 1 iff value a_i is observed.

Since all values a_i will appear in every permutation, we have that

$$\text{Expected Payoff} = \mathbb{E} \left[\sum_{i=0}^{n-1} A_i \right] = \sum_{i=0}^{n-1} \mathbb{E} [A_i]. \quad (2)$$

The contribution of each a_i clearly depends on whether the value is observed. This motivates the following lemmata.

Lemma 1. *For every $a_{j_0} \geq a$ and for every a_i ($i \neq j_0$) we have*

$$\mathbb{E} [O_i | V_t = a_i \ \& \ V_{t-1} \geq T] = 1 - \mathbb{E} [O_j | V_t = a_i \ \& \ V_{t-1} = a_{j_0}]. \quad (3)$$

Proof. If $a_i \geq T$, and for any fixed a_{j_0} , $j_0 \neq i$, we have

$$\begin{aligned} & \mathbb{E} [O_i | V_t = a_i \ \& \ V_{t-1} \geq T] \\ &= \sum_{j: a_j \geq T, j \neq i} \mathbb{P} [V_{t-1} = a_j | V_t = a_i \ \& \ V_{t-1} \geq T] \mathbb{E} [O_i | V_t = a_i \ \& \ V_{t-1} = a_j] \\ &= \frac{1}{k-1} \sum_{j: a_j \geq T, j \neq i} \mathbb{E} [O_i | V_t = a_i \ \& \ V_{t-1} = a_j] = \mathbb{E} [O_i | V_t = a_i \ \& \ V_{t-1} = a_{j_0}] \end{aligned}$$

where the last equality is due to the fact that the penult expectations are all the same for all j in the range of the summation. From the description of the threshold algorithm, and given that $V_t = a_i$ and $V_{t-1} = a_j \geq T$, we have that $O_i = 1$ exactly when $O_j = 0$. Therefore, we see that $\mathbb{E} [O_i | V_t = a_i \ \& \ V_{t-1} \geq T] = 1 - \mathbb{E} [O_j | V_t = a_i \ \& \ V_{t-1} = a_{j_0}]$ as we promised in (3). The proof for $a_i < T$ is almost identical. ■

We can now compute the expected value of O_i given that a_i has a certain position in the permutation.

Lemma 2.

$$\mathbb{E}[O_i|V_t = a_i] = \begin{cases} 1 - \frac{k}{n-1} f_{n-1,k}^{t-1}, & \text{if } a_i < T \\ f_{n,k}^t, & \text{if } a_i \geq T \end{cases} \quad (4)$$

where

$$f_{n,k}^t = \frac{1}{\binom{n-1}{k-1}} \sum_{s=0}^{\min\{t,k\}-1} (-1)^s \binom{n-1-s}{k-1-s}.$$

Proof. From the behaviour of the threshold algorithm, it is immediate that $\mathbb{E}[O_i|V_t = a_i]$ depends only on whether $a_i \geq T$ or not. First we observe that

$$\mathbb{E}[O_i|V_t = a_i] = \left(\begin{array}{l} \mathbb{P}[V_{t-1} < T|V_t = a_i] \mathbb{E}[O_i|V_t = a_i \ \& \ V_{t-1} < T] \\ + \mathbb{P}[V_{t-1} \geq T|V_t = a_i] \mathbb{E}[O_i|V_t = a_i \ \& \ V_{t-1} \geq T] \end{array} \right) \quad (5)$$

where

$$\mathbb{P}[V_{t-1} \geq T|V_t = a_i] = \begin{cases} \frac{k}{n-1}, & \text{if } a_i < T \\ \frac{k-1}{n-1}, & \text{if } a_i \geq T \end{cases}.$$

The next observation is that $\mathbb{E}[O_i|V_t = a_i \ \& \ V_{t-1} < T] = 1$. Indeed, if $V_{t-1} = a_j < T$, then a_j is either observed or not. If it is observed, then this happens only in L_0 , so the observer will also observe the next coming value which is a_i . If on the other hand a_j is not observed, then necessarily the next coming value is observed. Hence, expression (5) simplifies to

$$\mathbb{E}[O_i|V_t = a_i] = \begin{cases} 1 - \frac{k}{n-1} + \frac{k}{n-1} \mathbb{E}[O_i|V_t = a_i \ \& \ V_{t-1} \geq T], & \text{if } a_i < T \\ 1 - \frac{k-1}{n-1} + \frac{k-1}{n-1} \mathbb{E}[O_i|V_t = a_i \ \& \ V_{t-1} \geq T], & \text{if } a_i \geq T \end{cases} \quad (6)$$

We are now ready to justify (4) examining the two cases.

Case $a_i \geq T$: For every $a_i \geq T$, we set $\mathbb{E}[O_i|V_t = a_i] := f_{n,k}^t$, since the value is independent of a_i , but it is depended on the number of available values n , the position t , as well as the number of values k not less than T . Then we observe that (3) of Lemma 1 can be written as $\mathbb{E}[O_i|V_t = a_i \ \& \ V_{t-1} \geq T] = 1 - f_{n-1,k-1}^{t-1}$. Continuing from (6), we see then that $f_{n,k}^t = 1 - \frac{k-1}{n-1} f_{n-1,k-1}^{t-1}$. Given that for all $t, k \geq 1$ we have that $f_{n,k}^t = 1$ whenever $t = 1$ or $k = 1$ the claim follows.

Case $a_i < T$: Similar to the previous case we write (3) as $\mathbb{E}[O_i|V_t = a_i \ \& \ V_{t-1} \geq T] = 1 - f_{n-1,k}^{t-1}$ (note that in this case, and since $a_i < T$ we still have k many values at least T to choose from). Hence, (6) becomes $\mathbb{E}[O_i|V_t = a_i] = 1 - \frac{k}{n-1} f_{n-1,k}^{t-1}$, again as promised. ■

It is clear from the previous lemmata that the formulas of the payoff of online Threshold algorithms involves numerous binomial expressions, which we simplify in Appendices A.2,A.3. Given this quite technical work, we are ready to prove Theorem 4.

Proof (of Theorem 4). Let a denote some threshold value, such that $n - k$ many a'_i 's are less than a . For every $i = 0, \dots, n - 1$ we have

$$\mathbb{E}[A_i] = \sum_{t=1}^n \mathbb{P}[V_t = a_i] \mathbb{E}[A_i|V_t = a_i] = \frac{1}{n} \sum_{t=1}^n \mathbb{E}[A_i|V_t = a_i]. \quad (7)$$

Using the random variables O_i that indicate whether a_i is observed, we have

$$\mathbb{E}[A_i|V_t = a_i] = \begin{cases} a_i \mathbb{E}[O_i|V_t = a_i] & , \text{ if } a_i < a \\ 2a_i \mathbb{E}[O_i|V_t = a_i] & , \text{ if } a_i \geq a \end{cases}$$

whose values are given by Lemma 2. Hence, by (2) and (7), we have that

$$\begin{aligned} \text{Expected Payoff} &= \sum_{i=0}^{n-k-1} \frac{1}{n} \sum_{t=1}^n \mathbb{E}[A_i|V_t = a_i] + \sum_{i=n-k}^{n-1} \frac{1}{n} \sum_{t=1}^n \mathbb{E}[A_i|V_t = a_i] \\ &= \left(1 - \sum_{s=0}^{k-1} (-1)^s \frac{\binom{n-1-s}{k-1-s}}{\binom{n}{k}}\right) A^- + 2 \left(\sum_{s=0}^{k-1} (-1)^s \frac{\binom{n-s}{k-1-s}}{\binom{n}{k-1}}\right) A^+. \end{aligned}$$

(From Lemma 3 in Appendix A.2)

The relative performance is obtained by dividing by n . According to technical Lemma 4 in Appendix A.3, and given that $\frac{k}{n} \rightarrow c$, the theorem follows. ■

3.2 Competitive Analysis for Random Permutations

We can now prove Theorems 1 and 2 pertaining to random permutations. Suppose that the Threshold algorithm chooses threshold value T equal to the \bar{k} largest element of the value a_i . Denote by A the sum of all values a_i , and by $L_{\bar{k}}$ the sum of the \bar{k} largest values of them. Abbreviate also \bar{k}/n by c .

Theorem 4 applies with $T := a_{n-\bar{k}}$, to give (asymptotically) that

$$ALG = \frac{1}{1+c} \frac{1}{n} A + \frac{c}{1+c} \frac{1}{\bar{k}} L_{\bar{k}}. \quad (8)$$

At the same time, Theorem 3 implies that for the optimal offline algorithm we have

$$OPT = \frac{1}{\binom{n}{2}} \sum_{i=0}^{n-1} i \cdot a_i \leq \frac{1}{\binom{n}{2}} \left((n-k) \sum_{i=0}^{n-\bar{k}-1} a_i + n \sum_{i=n-\bar{k}}^{n-1} a_i \right) = \frac{2}{n} ((1-c)A + cL_{\bar{k}}). \quad (9)$$

Proof (of Theorem 1 for Random Permutations). When the Threshold value is the median, we have that $c = 1/2$. Using the bounds (8) and (9) it is straightforward that ALG, OPT are indeed within $2/3$ of each other as promised. ■

Proof (of Theorem 2 for Random Permutations). When the input is a c -dense stream, the Threshold algorithm can choose \bar{k} satisfying $1 - \bar{k}/n = 1 - c = L_{\bar{k}}/A$. But then the competitive ratio becomes

$$\frac{ALG}{OPT} \stackrel{(8),(9)}{\geq} \frac{1}{2} \cdot \frac{1}{1+c} \cdot \frac{1 + \frac{L_{\bar{k}}}{A}}{(1-c) + c \frac{L_{\bar{k}}}{A}} = \frac{1}{2} \cdot \frac{2-c}{(1-c)(1+c)^2}.$$

■

4 Random iid-Valued Input Streams

In this section we study the special case of inputs streams whose elements are iid valued. As per the description of the model in Section 2.4, we assume that the value v_i of the i -th input item of the stream is an independent random variable assuming a value $a_0 < a_1 < \dots < a_{k-1}$ (with $k \geq 2$) with probability p_0, p_1, \dots, p_{k-1} respectively (i.e., $\Pr[v_i = a_j] = p_j$).

4.1 Performance of Offline and Online Algorithms for iid-Valued Streams

Using Observation 1, we compute in Appendix B.1 the asymptotic payoff of the optimal offline algorithm.

Theorem 5. *The relative expected payoff (asymptotic payoff per time step) of the optimal offline algorithm when the input is a random i.i.d. sequence is $\sum_{i=0}^{k-1} p_i a_i + \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} p_i p_j (a_j - a_i)$.*

The remaining of this section is devoted in determining the asymptotics of any Threshold algorithm.

Theorem 6. *The relative expected payoff of the Threshold algorithm (asymptotic payoff per time step) that uses threshold $T = a_r$ and when the input is a random i.i.d. is $\frac{\sum_{i=0}^{r-1} p_i a_i + 2 \sum_{i=r}^{k-1} p_i a_i}{\sum_{i=0}^{r-1} p_i + 2 \sum_{i=r}^{k-1} p_i}$.*

Proof. In what follows we introduce abbreviations $Avg := \sum_{i=0}^{k-1} p_i a_i$ and $P := \sum_{i=r}^{k-1} p_i$. Let also Y_i be the random variable such that $Y_i = b$ indicates that, at time i , the observer is at L_b , $b \in \{0, 1\}$. Let also $q_i := \mathbb{P}[Y_i = 0]$. By definition, $q_0 = 1$. Next we observe that

$$1 - q_{i+1} = \mathbb{P}[Y_{i+1} = 1] = \mathbb{P}[Y_i = 0 \ \& \ X_i \geq T] = \mathbb{P}[X_i \geq T \mid Y_i = 0] \mathbb{P}[Y_i = 0] = P q_i.$$

Technical Lemma 5 in Appendix B.2 implies that

$$q_i = \frac{1 - (-1)^i P^i}{1 + P}. \tag{10}$$

Next we observe that $\mathbb{E}[X_i \mid Y_i = 0] = Avg$. Also, if we set $Avg^+ := \sum_{s=r}^{k-1} a_s p_s$ we see that

$$\mathbb{E}[X_i \mid Y_i = 1] = \mathbb{E}[X_i \mid Y_{i-1} = 0 \ \& \ X_{i-1} \geq T] = \frac{Avg^+}{P}.$$

We now compute

$$\begin{aligned} \mathbb{E}\left[\sum_{i=1}^n X_i\right] &= \sum_{i=1}^n \mathbb{E}[X_i] = \sum_{i=1}^n \left(\mathbb{P}[Y_i = 0] \mathbb{E}[X_i \mid Y_i = 0] + \mathbb{P}[Y_i = 1] \mathbb{E}[X_i \mid Y_i = 1] \right) \\ &\stackrel{(10)}{=} \left(\frac{n}{1+P} + \frac{P + (-P)^{n+1}}{(1+P)^2} \right) \cdot Avg + \left(\frac{n}{1+P} + \frac{(-P)^n - 1}{(1+P)^2} \right) \cdot Avg^+. \end{aligned}$$

Dividing the last quantity by n , and taking the limit $n \rightarrow \infty$ gives the promised formula. \blacksquare

4.2 Competitive Analysis for Uniform iid-Valued Input Streams

Note that the formulas derived in Section 4.1 hold for all iid-valued input streams. In this section we provide competitive analysis for input streams that are uniformly valued, i.e. when $p_i = \frac{1}{k}$, for all $i = 0, \dots, k-1$. That would be Theorems 1 and 2 pertaining to uniform iid-valued random input streams.

As before, denote by A the sum of all values a_i , and by $L_{\bar{r}}$ the sum of the \bar{r} largest values of them. Abbreviate also \bar{r}/n by c . Suppose also that the Threshold algorithm uses as threshold the

\bar{r} -th largest value of the a_i 's. We use Theorem 5 to find an upper bound for the performance of the offline algorithm:

$$OPT = \frac{1}{k}A + \frac{1}{k^2} \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} (a_j - a_i) = \frac{1}{k}A + \frac{1}{k^2} \sum_{i=0}^{k-1} (2i - k + 1)a_i \leq 2 \left(\frac{k - \bar{r} + 1/2}{k^2}A + \frac{\bar{r}}{k^2}L_{\bar{r}} \right). \quad (11)$$

Next, using Theorem 6 (which is written for threshold value $a_r = a_{k-1-\bar{r}}$) we obtain that for the Threshold algorithm

$$ALG = \frac{1}{k} \cdot \frac{A + L_{\bar{r}}}{1 + \bar{r}/k}. \quad (12)$$

Proof (of Theorem 1 for Uniform iid-Valued Streams). When the Threshold value is the median, we have that $\bar{r}/k = 1/2$. Using bounds (11) and (12), it is straightforward then to see that ALG, OPT are indeed within $2/3$ of each other as promised. ■

Proof (of Theorem 2 for Uniform iid-Valued Streams Random Permutations). When the input is a c -dense stream, the Threshold algorithm can choose \bar{r} satisfying $1 - \bar{r}/n = 1 - c = L_{\bar{r}}/A$. But then the competitive ratio becomes

$$\frac{ALG}{OPT} \stackrel{(11),(12)}{\geq} \frac{1}{2} \cdot \frac{1}{1+c} \cdot \frac{1 + \frac{L_{\bar{r}}}{A}}{(1-c + o(c)) + c\frac{L_{\bar{r}}}{A}} \rightarrow \frac{1}{2} \cdot \frac{2-c}{(1-c)(1+c)^2}.$$

■

5 Open Problems

As described in the introduction, our model can be extended in many different ways. An obvious extension is to have a bigger buffer, i.e., $k > 2$ locations L_0, L_1, \dots, L_{k-1} . In this case, there are different possibilities of moving the processor within the buffer: a *single jump* model would require the processor to always jump to L_0 , while a *local jump* model would allow the processor to move close to its current location. Another obvious extension would be to consider general payoffs, i.e., allowing an item to have different values in different buffer locations. Also, we leave open the potential increase in the power of the processor if it is allowed to know the item it's going to miss in L_0 (if it moves to L_0 from L_1 in the next time slot).

The threshold algorithm is probably the simplest algorithm one can use to tackle PERSISTENCE. The obvious question is whether there are better algorithms for the non-oblivious setting. Also, are there *upper bounds* that can be shown? In the oblivious setting, it is obvious that the thresholds we calculated above do not apply since we do not know the payoffs ahead of time. In that setting, it is natural to consider adaptive algorithms, probably using a prefix of the input in order to 'learn' something about it before employing a threshold-like or some other strategy.

References

1. M. Ajtai, T. S. Jayram, R. Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 370–379. ACM, 2002.
2. N. Alon, N. Duffield, C. Lund, and M. Thorup. Estimating arbitrary subset sums with few probes. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 317–325. ACM, 2005.

3. N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.
4. M. Babaioff, N. Immorlica, and R. Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of SODA*, pages 434–443. SIAM, 2007.
5. M. Bateni, M. Hajiaghayi, and M. Zadimoghaddam. Submodular secretary problem and extensions. *ACM Transactions on Algorithms (TALG)*, 9(4):32, 2013.
6. J. N. Bearden. A new secretary problem with rank-based selection and cardinal payoffs. *Journal of Mathematical Psychology*, 50(1):58–59, 2006.
7. G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proceedings of the 31st international conference on Very large data bases*, pages 25–36. VLDB Endowment, 2005.
8. N. Devanour, K. Jain, B. Sivan, and C. Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. In *Proceedings of the 12th ACM conference on Electronic commerce*, pages 29–38. ACM, 2011.
9. T. S. Ferguson. Who solved the secretary problem? *Statistical science*, pages 282–289, 1989.
10. P. R. Freeman. The secretary problem and its extensions: A review. *International Statistical Review/Revue Internationale de Statistique*, pages 189–206, 1983.
11. J. Gianini and S. M. Samuels. The infinite secretary problem. *The Annals of Probability*, pages 418–432, 1976.
12. M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD*, volume 30(2), pages 58–66. ACM, 2001.
13. M. Hoffman, S. Muthukrishnan, and R. Raman. Location streams: Models and algorithms. Technical report, DIMACS TR, 2004.
14. R. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 630–631. Society for Industrial and Applied Mathematics, 2005.
15. G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment, 2002.
16. S. Muthukrishnan. *Data streams: Algorithms and applications*. Now Publishers Inc, 2005.
17. S. Muthukrishnan. *Data stream Algorithms (Notes from a series of lectures)*. The 2009 Barbados Workshop on Computational Complexity, March 1 - 8, 2009.
18. M. Smith. A secretary problem with uncertain employment. *Journal of applied probability*, pages 620–624, 1975.

A Parts Omitted from Section 3.1

A.1 Proof of Theorem 3

If X_i is the random variable whose value is the profit of the optimal algorithm at time i , we need to calculate $\mathbb{E} \left[\sum_{i=1}^{n+1} X_i \right] = \sum_{i=1}^{n+1} \mathbb{E} [X_i]$. When calculating the relative expected payoff, the extreme case observations X_1, X_{n+1} have (asymptotically) 0 contribution. So we may focus on a fixed and arbitrary time step i and evaluate $\mathbb{E} [X_i]$.

At time i , let T_0, T_1 denote the random variables that are equal to values in the two windows. For the optimal algorithm, we have that $X_i = \max\{T_0, T_1\}$. Since the input is a random permutation, and for all $i \neq j$, we observe that

$$\mathbb{P} [T_0 = a_i \ \& \ T_1 = a_j] = \frac{(n-2)!}{n!} = \frac{1}{n(n-1)}.$$

Hence, using also Observation 1, we have

$$\begin{aligned}
\mathbb{E}[X_i] &= \sum_{i=0}^{n-1} \sum_{j \in \{0,1,\dots,n-1\} \setminus \{i\}} \mathbb{P}[T_0 = a_i \text{ \& } T_1 = a_j] \max\{a_i, a_j\} \\
&= \frac{1}{n(n-1)} \left(\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} \max\{a_i, a_j\} + \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \max\{a_i, a_j\} \right) \\
&= \frac{1}{n(n-1)} \left(\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} a_i + \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} a_j \right) \\
&= \frac{1}{n(n-1)} \sum_{i=1}^{n-1} 2i \cdot a_i.
\end{aligned}$$

A.2 Combinatorial Identities - Part I

Lemma 3.

$$\begin{aligned}
(a) \quad \sum_{t=1}^n f_{n,k}^t &= n \sum_{s=0}^{k-1} (-1)^s \frac{\binom{n-s}{k-1-s}}{\binom{n}{k-1}} \\
(b) \quad \sum_{t=1}^n \left(1 - \frac{k}{n-1} f_{n-1,k}^{t-1} \right) &= n - n \sum_{s=0}^{k-1} (-1)^s \frac{\binom{n-1-s}{k-1-s}}{\binom{n}{k}}.
\end{aligned}$$

Proof.

$$\begin{aligned}
(a) \quad \binom{n}{k-1} \sum_{t=1}^n f_{n,k}^t &= \sum_{t=1}^n \sum_{s=0}^{\min\{t,k\}-1} (-1)^s \binom{n-1-s}{k-1-s} \\
&= \sum_{t=1}^k \sum_{s=0}^{t-1} (-1)^s \binom{n-1-s}{k-1-s} + \sum_{t=k+1}^n \sum_{s=0}^{k-1} (-1)^s \binom{n-1-s}{k-1-s} \\
&= \sum_{s=0}^{k-1} (-1)^s (k-s) \binom{n-1-s}{k-1-s} + (n-k) \sum_{s=0}^{k-1} (-1)^s \binom{n-1-s}{k-1-s} \\
&= \sum_{s=0}^{k-1} (-1)^s (n-s) \binom{n-1-s}{k-1-s} \\
&= n \sum_{s=0}^{k-1} (-1)^s \binom{n-s}{k-1-s}
\end{aligned}$$

$$\begin{aligned}
(b) \quad \sum_{t=1}^n \left(1 - \frac{k}{n-1} f_{n-1,k}^{t-1} \right) &= n - \frac{k}{n-1} \sum_{t=1}^n f_{n-1,k}^{t-1} \\
&\stackrel{(f_{n,k}^0=0)}{=} n - \frac{k}{n-1} \sum_{t=2}^n f_{n-1,k}^{t-1} \\
&= n - \frac{k}{n-1} \sum_{t=1}^{n-1} f_{n-1,k}^{t-1} \\
&\stackrel{(a)}{=} n - k \sum_{s=0}^{k-1} (-1)^s \frac{\binom{n-1-s}{k-1-s}}{\binom{n-1}{k-1}} \\
&= n - n \sum_{s=0}^{k-1} (-1)^s \frac{\binom{n-1-s}{k-1-s}}{\binom{n}{k}}.
\end{aligned}$$

■

A.3 Combinatorial Identities - Part 2

Lemma 4. For every $k = k(n)$, let $\lim_{n \rightarrow \infty} \frac{k}{n} = c \in \Theta(1)$. Then

$$\lim_{n \rightarrow \infty} \frac{1}{\binom{n}{k-1}} \sum_{s=0}^{k-1} (-1)^s \binom{n-s}{k-1-s} = \frac{1}{1+c}.$$

Proof. We show that for every $\epsilon > 0$ (that can be chosen to be arbitrarily small), such that $\log \epsilon / \log c$ is an even integer, we have

$$\frac{1}{1+c} - \frac{\epsilon}{1+c} \leq \lim_{n \rightarrow \infty} \frac{1}{\binom{n}{k-1}} \sum_{s=0}^{k-1} (-1)^s \binom{n-s}{k-1-s} \leq \frac{1}{1+c} + \frac{\sqrt{\epsilon}}{1-c}. \quad (13)$$

Indeed, consider the odd constant $r := \log \epsilon / \log c - 1$. Then we have

$$\begin{aligned}
&\lim_{n \rightarrow \infty} \frac{1}{\binom{n}{k-1}} \sum_{s=0}^{k-1} (-1)^s \binom{n-s}{k-1-s} \\
&= \lim_{n \rightarrow \infty} \underbrace{\frac{1}{\binom{n}{k-1}} \sum_{s=0}^r (-1)^s \binom{n-s}{k-1-s}}_{A(n)} + \lim_{n \rightarrow \infty} \underbrace{\frac{1}{\binom{n}{k-1}} \sum_{s=r+1}^{k-1} (-1)^s \binom{n-s}{k-1-s}}_{B(n)}.
\end{aligned}$$

Since $A(n)$ is a finite sum we have

$$\begin{aligned}
\lim_{n \rightarrow \infty} A(n) &= \sum_{s=0}^r (-1)^s \left(\lim_{n \rightarrow \infty} \frac{1}{\binom{n}{k-1}} \binom{n-s}{k-1-s} \right) \\
&= \sum_{s=0}^r (-1)^s c^s \\
&= \frac{1 + (-1)^r c^{r+1}}{1 + c} \\
&= \frac{1 - \epsilon}{1 + c}.
\end{aligned} \tag{14}$$

Now, without loss of generality assume that k is even (otherwise the last summand is positive and our bound below is still valid). We observe that

$$\begin{aligned}
B(n) &= \sum_{s=r+1}^{k-1} (-1)^s \frac{\binom{n-s}{k-1-s}}{\binom{n}{k-1}} \\
&= \sum_{s=(r+1)/2}^{(k-2)/2} \left(\frac{\binom{n-s}{k-1-s}}{\binom{n}{k-1}} - \frac{\binom{n-s-1}{k-2-s}}{\binom{n}{k-1}} \right) \\
&= \left(1 - \frac{k+1}{n} \right) \sum_{s=(r+1)/2}^{(k-2)/2} \prod_{j=0}^{s+1} \frac{k-1-j}{n-1-j}.
\end{aligned}$$

It is easy to see that $\frac{k-1-j}{n-1-j}$ are decreasing with j , and since each term is positive, we have

$$0 < B(n) < \left(1 - \frac{k+1}{n} \right) \sum_{s=(r+1)/2}^{(k-2)/2} \left(\frac{k-1}{n-1} \right)^{s+2}. \tag{15}$$

The non-negativity of $\lim_{n \rightarrow \infty} B(n)$, together with (14), imply the lower bound of (13). As for the upper bound, we introduce the shorthand $q := (k-1)/(n-1)$, and we see that (15), after we compute the sum in the right-hand side, implies that

$$B(n) < \frac{q^2}{1-q} \left(q^{(r+1)/2} - q^{k/2} \right) < \frac{q^{(r+5)/2}}{1-q}.$$

Therefore

$$\lim_{n \rightarrow \infty} B(n) < \frac{\left(\lim_{n \rightarrow \infty} \frac{k-1}{n-1} \right)^{(r+5)/2}}{1 - \lim_{n \rightarrow \infty} \frac{k-1}{n-1}} = \frac{c^{(r+5)/2}}{1-c} = \frac{c^2 \sqrt{\epsilon}}{1-c}.$$

Combining the above, and given that $c \leq 1$, we conclude that (13) holds. \blacksquare

B Parts Omitted from Section 4.1

B.1 Proof of Theorem 5

A random i.i.d. input realizes into the sequence $a_{i_1} a_{i_2} \dots a_{i_{n-1}}$ with probability $\prod_{j=1}^n p_{i_j}$, where $i_j \in \{0, k-1\}$, $j = 1, \dots, n$. By Observation 1, it follows that the expected profit between time 2

and n equals

$$\sum_{i_n=0}^{k-1} \sum_{i_{n-1}=0}^{k-1} \dots \sum_{i_1=0}^{k-1} \prod_{j=1}^n p_{i_j} \sum_{t=2}^n \max\{a_{i_t}, a_{i_{t-1}}\}. \quad (16)$$

We focus at the case $t = 2$, since it is immediate from the above formula that calculations will be identical for any $t \in \{2, \dots, n\}$. The summand of (16) corresponding to $t = 2$ equals

$$\begin{aligned} & \sum_{i_n=0}^{k-1} \sum_{i_{n-1}=0}^{k-1} \dots \sum_{i_1=0}^{k-1} \prod_{j=1}^n p_{i_j} \max\{a_{i_2}, a_{i_1}\} \\ = & \sum_{i_n=0}^{k-1} \sum_{i_{n-1}=0}^{k-1} \dots \sum_{i_3=0}^{k-1} \prod_{j=3}^n p_{i_j} \left(\sum_{i_2=0}^{k-1} \sum_{i_1=0}^{k-1} p_{i_1} p_{i_2} \max\{a_{i_2}, a_{i_1}\} \right) \\ = & \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} p_i p_j \max\{a_i, a_j\} \\ = & \sum_{i=0}^{k-1} \sum_{j=0}^i p_i p_j \max\{a_i, a_j\} + \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} p_i p_j \max\{a_i, a_j\} \\ = & \sum_{i=0}^{k-1} \sum_{j=0}^i p_i p_j a_i + \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} p_i p_j a_j \\ = & \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} p_i p_j a_i - \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} p_i p_j a_i + \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} p_i p_j a_j \\ = & \sum_{i=0}^{k-1} p_i a_i + \sum_{i=0}^{k-1} \sum_{j=i+1}^{k-1} p_i p_j (a_j - a_i). \end{aligned}$$

The last formula concludes the theorem.

B.2 Solution to a Recurrence

Lemma 5. *The solution to the recurrence $1 - q_{i+1} = Pq_i$ is given by the formula $q_i = \frac{1 - (-1)^i P^i}{1 + P}$.*

Proof. This recurrence can be solved using generating functions. Indeed, let us define the function $f(x) := \sum_{i \geq 0} q_i x^i$. If we multiply the recurrence by x^{i+1} we see that $q_{i+1} x^{i+1} + Aq_i x^{i+1} = x^{i+1}$. Summing all these recurrences for $i \geq 0$ we conclude that

$$\sum_{i \geq 0} q_{i+1} x^{i+1} + Ax \sum_{i \geq 0} q_i x^i = \sum_{i \geq 0} x^{i+1}.$$

This last equation is easily seen to be equivalent to $f(x) + Ax f(x) = \frac{x}{1-x}$. It follows that $f(x) = \frac{x}{(1-x)(1+Ax)}$ from which we easily derive that $q_i = \frac{1 - (-1)^i P^i}{1 + P}$, as wanted. \blacksquare