The Secretary Problem with Independent Sampling: E-companion

José Correa[1], Andrés Cristi[2], Laurent Feuilloley[3], Tim Oosterwijk[4], and Alexandros Tsigonias-Dimitriadis[5]

[1] *Department of Industrial Engineering, Universidad de Chile, Chile*
[2] *Center for Mathematical Modeling, Universidad de Chile, Chile*
[3] *CNRS, INSA Lyon, UCBL, LIRIS, UMR5205, F-69622 Villeurbanne, France*
[4] *School of Business and Economics, Vrije Universiteit Amsterdam, Netherlands*
[5] *European Central Bank, Germany*

Here we provide all the missing proofs and any other parts omitted from the main text.

1 Appendix for Section 3

1.1 Intuition behind threshold for AOSp

We briefly show the intuition behind the choice of k in the k-max algorithm for AOSp. For the analysis, assume that the samples are sorted in non-increasing order $s_1 \geq s_2 \geq \cdots \geq s_r$. An algorithm A, given input $(\mathcal{S}, p)$, draws $t \sim \mathcal{D}$ and sets a threshold $\tau = s_t$. Here, $t \in [r]$ and $\mathcal{D}$ is a probability distribution over the indices of the samples. Thus, A succeeds at least in the instances where *exactly one* of the t largest values of the adversarial input ends up in the online set and the $(t+1)^{\text{th}}$ largest ends up in the sample set. Since the coin flips are independent, we obtain

$$\Pr[A \text{ stops at } \max_i v_i] = lp^l(1-p), \qquad \text{where } l = \mathbb{E}_{t\sim\mathcal{D}}[t].$$

To find the maximizer of the function $f(p,l) = lp^l(1-p)$, consider its derivative:

$$\frac{\partial f(p,l)}{\partial l} = \frac{\partial(lp^l - lp^{l+1})}{\partial l} = 0 \Leftrightarrow l = -\frac{1}{\ln p}\,.$$

Substituting this value for l into $f(p,l)$ yields an upper bound $f_1(p) = \frac{p-1}{\log p}\cdot p^{-1/\log p}$. To turn it into a practical algorithm, we wish to have an integer value of l without a logarithmic term. The latter can be fixed by taking an approximation. Taking the Maclaurin series of $\ln(1+x)$, substituting $p = x+1$ and dropping the higher order terms yields

$$\ln p = \sum_{n=1}^{\infty}(-1)^{n-1}\frac{(p-1)^n}{n} \approx p-1 \Leftrightarrow -\frac{1}{\ln p} \approx \frac{1}{1-p}, \qquad p \in (0,1)\,.$$

Substituting this value for l into $f(p,l)$ yields a lower bound $f_2(p) = p^{1/(1-p)}$. To get to an integer value, we simply take the floor function of this expression and obtain the k-max algorithm as defined previously.

Let us be a bit more precise regarding the success guarantee after these modifications and show that it interpolates between the two functions mentioned above. Because $-\frac{1}{\ln p}$ is a maximizer of f, we see that $f\left(p, \left\lfloor \frac{1}{1-p} \right\rfloor\right) \leq f\left(p, -\frac{1}{\ln p}\right)$. Second, $f(p,l)$ is increasing for $l \leq -\frac{1}{\ln p}$ and decreasing in the rest of the domain. Moreover, one can easily check that $f\left(p, \frac{1}{1-p} - 1\right) = f\left(p, \frac{1}{1-p}\right)$ and $\frac{1}{1-p} - 1 < -\frac{1}{\ln p} \leq \frac{1}{1-p}$ for $p \in (0,1)$. Finally, since it holds that $\frac{1}{1-p} - 1 < \left\lfloor \frac{1}{1-p} \right\rfloor \leq \frac{1}{1-p}$, we get $f(p, \frac{1}{1-p}) \leq f(p, \left\lfloor \frac{1}{1-p} \right\rfloor)$. Combining all the above, we can conclude that $f\left(p, \left\lfloor \frac{1}{1-p} \right\rfloor\right) \approx f\left(p, -\frac{1}{\ln p}\right)$.

1.2 Omitted proofs from Section 3.2.3

Proof of Lemma 2. Let $I_1 \nsim I_2$ be two instances in conflict. Consider a deterministic algorithm that succeeds in I_1, meaning, this algorithm stops at the position r of the last 0. Note that at any position $j \leq r$, the knowledge of the algorithm up to that point consists of $||I_1||_1$ as well as $I_1[1, j]$. Now run the same algorithm on I_2. Since the algorithm is deterministic and has the same information available at every point in time, it must make the exact same decision at every $j \leq r$. In particular, it stops at position r. However, since $||I_2||_1 = ||I_1||_1$ but $|I_2| > |I_1|$, there must be a 0 after position r in I_2, and the algorithm fails in I_2. □

Proof of Lemma 3. Let $I_1 \nsim I_2$ be two instances in conflict with sizes n_1, n_2 such that $n_1 < n_2$. Since they are in conflict, $||I_2||_1 = ||I_1||_1$ and $I_1[1, r] = I_2[1, r]$, where r is the position of the last 0 of I_1. Let $s > r$ be the last 0 of I_2. Consider the instance I_3 such that $I_3[1, s-1] = I_2[1, s-1]$ and $I_3[s, n_2 - 1] = I_2[s+1, n_2]$, i.e., I_3 is the instance I_2 without its last 0.

If $I_3 = I_1$, the proof is done. Otherwise, observe that $||I_3||_1 = ||I_2||_1 = ||I_1||_1$ and that $I_1[1, r] = I_3[1, r]$, thus, $I_3 \nsim I_1$. Also, $I_2[1, t] = I_3[1, t]$ where $t \in [r, s)$ is the position of the last 0 of I_3, and therefore also $I_3 \nsim I_2$.

For an instance I containing at least one 0, define the operation $I \ominus 0$ that maps I to a new instance I' which is equal to I with its last 0 removed, as above. Starting with I_2, we can create a sequence of $d := n_2 - n_1$ instances with $I_{i+1} = I_i \ominus 0$ such that $I_i \nsim I_1$ and $I_i \nsim I_2$ for all $i \in [1, d+2]$. This yields the sequence $I_2, I_3, \ldots, I_{d+1}, I_{d+2} = I_1$, whose instances form a monotone path in the conflict graph, proving one direction. The other direction of the proof follows similarly. □

Proof of Lemma 5. Let I be an instance of size n. By Lemma 4, an instance I' of size $n+1$ is in conflict with I (and therefore a child of I) if it contains a 0 anywhere after the last 0 of I. Therefore, if I ends in 0, the node corresponding to this instance has degree one, since the new 0 can only be inserted in one place. Half of the instances of size n end in 0, so 2^{n-1} nodes have degree one. Similarly, we see that a quarter of the instances of size n end in 01, i.e., 2^{n-2} nodes have degree two. In general, for every i there are 2^{n-i} instances with suffix 01^{i-1}, and therefore, 2^{n-i} nodes have degree i.

A node v with degree k has a suffix 01^{k-1}. Any child w of v has one additional 0 after the last 0 of v. If this additional 0 occurs at the very end, the suffix of w is 0 so $\deg(w) = 1$. If this additional 0 occurs directly before the last 1, the suffix of w is 01 so $\deg(w) = 2$. In general, inserting the additional 0 directly before the i-th 1 from the end, a child is created with degree i. □

Proof of Lemma 6. The proof of Lemma 5 reveals that the instances of degree one are exactly these which have a suffix of 0. Summing over their weights will yield $w_1 = 1 - p$, which is the probability that the last bit of an instance is equal to 0. In general, a node of degree i ends in 01^{i-1}. Accordingly, the probability that an instance with this suffix results from the sampling process is $w_i = p^{i-1}(1-p)$.

Now consider an instance I_1 with $|I_1| = n$ and $\deg(I) = i$. Then I_1 has suffix 01^{i-1}. Now consider an instance $I_2 \nsim I_1$ with $|I_2| = n+1$ and $\deg(I_2) = j$ for some $j \leq i$. Since $I_2 \nsim I_1$, $I_2[1, n-i+1] = I_1[1, n-i+1]$, since the last 0 of I_1 is at position $n - i + 1$. In particular, compared to I_1 the suffix 1^{i-1} contains one additional 0 in I_2. Therefore, I_2 has suffix $01^{i-j}01^{j-1}$.

Now consider the set of all instances that have the form of instance I_1. Because of the suffix 01^{i-1}, the weight of these instances can be computed as $p^{i-1}(1-p)$. On the other hand, the weight of all instances that have the form of the instance I_2 can be computed as $(1-p)p^{i-j}(1-p)p^{j-1} = p^{i-1}(1-p)^2$. □

Proof of Lemma 7. By Lemma 3, any two instances that are in a monotone path are in conflict, and by Lemma 2 an algorithm can win in at most one instance of a pair of conflicting instances. □

1.3 Omitted proofs from Section 3.2.4

Proof of Lemma 8. We adapt the proof of Lemma 2 in a straightforward manner. Let $I_1 \nsim I_2$ be two instances in conflict. Consider a randomized algorithm that succeeds in I_1, meaning, this algorithm stops at the position r of the last 0. Note that at any position $j \leq r$, the knowledge of the algorithm up to that point consists of $||I_1||_1$ as well as $I_1[1, j]$. Now run the same algorithm on I_2 with the same random seed. Since the algorithm has the same random seed and has the same information available at every point in time, it must make the exact same decision at every $j \leq r$ with the exact same probability. In particular, it stops at position r with probability q. However, since $||I_2||_1 = ||I_1||_1$ but $|I_2| > |I_1|$, there must be a 0 after position r in I_2, and the algorithm can only reach this last 0 of I_2 with probability at most $1 - q$, and therefore, succeed in I_2 with probability no more than $1 - q$. □

Proof of Lemma 9. This follows immediately from Lemma 5. It implies that v and w have children with the same degree distribution and the same holds for all their descendants. □

Proof of Lemma 10. Consider the instances I_1 and I_2 with $|I_1| = |I_2|$ and an algorithm A with $I_1 \in A_S$ but $I_2 \notin A_S$. Since $I_1 \in A_S$, its descendants are removed. On the other hand, as I_2 is neither selected nor removed, it is possible for A to select some nodes in its subtree $T(I_2)$.

Let I be the instance of I_1 and I_2 with highest degree (breaking a tie arbitrarily) and I' be the other instance. Now consider the subtree T consisting of I, its $\deg(I')$ children, and their descendants, and let T' be the subtree of I'. Observe that $\deg_T(I) = \deg(I')$, and hence, as observed in Lemma 9, T and T' are isomorphic. Since $|I_1| = |I_2|$, they are not neighbours, and moreover, since any node has at most one parent, T and T' are also disjoint.

Let φ be the isomorphic mapping between T and T' and let $U = A_S \cap T'$. Now consider the algorithm B with $B_S = A_S \cup \{I_2\} \cup \varphi(U) \setminus (\{I_1\} \cup U)$. By construction, for every given size n, the nodes that both algorithms select carry the same weight, so the success guarantees are equal. □

Proof of Lemma 11. Consider an algorithm A that does not follow the small degrees first strategy. Then there exists an instance $I_1 \in A_S$ and an instance $I_2 \notin A_S$ with $|I_2| = |I_1|$ and $\deg(I_2) < \deg(I_1)$.

Consider the part $T'(I_1)$ of the subtree $T(I_1)$ that consists of I_1 and its $\deg(I_2)$ children of smallest degree and their subtrees. By the structure given by Lemma 5, $T'(I_1)$ is isomorphic to $T(I_2)$. Then the same swapping argument as in the proof of Lemma 10 between $T'(I_1)$ and $T(I_2)$ exhibits another algorithm with the same success guarantee that does follow the small degrees first strategy. □

Proof of Claim 1. We prove the claim by induction. For the base case $n = N_0$ we have $\rho_{N_0} = 1/4 + \varepsilon$, which corresponds to the formula of the claim. Now suppose that the lemma holds for some size $n + t - 1$, so $\rho_{n+t-1} = (\frac{1}{4} + \varepsilon) \cdot \sum_{i=1}^{t-1} 1/2^{i-1}$. We first determine the fraction of removed nodes in the next size $n + t$. Since each node of degree 1 removes one node of the next size, the number of nodes removed for size $n + t$ is the same. However, as there are twice as many instances in total in size $n + t$, the fraction is half this number, namely $(\frac{1}{4} + \varepsilon) \cdot \sum_{i=1}^{t-1} 1/2^i$. The fraction of selected nodes is $1/4 + \varepsilon$, thus in total the cover ratio becomes

$$\rho_{n+t} = \left(\frac{1}{4} + \varepsilon\right) \cdot \left(\sum_{i=1}^{t-1} \frac{1}{2^i} + 1\right) = \left(\frac{1}{4} + \varepsilon\right) \cdot \left(\sum_{i=1}^{t} \frac{1}{2^{i-1}}\right) .$$ □

Proof of Claim 2. Let us consider a size n where $\rho_n > 1/2$, say $\rho_n = 1/2 + \delta$ for some $\delta > 0$. Then for size $n+1$, first, the $1/2$-fraction of nodes of size n remove $1/4$ of the nodes of size $n+1$ (since all these nodes have degree 1). Then, by Lemma 5, the δ-fraction of degree 2 nodes remove one instance of degree 1 and one instance of degree 2 in the next size. That is, in size $n+1$, a $(1/4 + \delta/2)$-fraction of the degree 1 nodes and a $\delta/2$-fraction of the degree 2 nodes are removed in total.

The algorithm must now select a $(1/4 + \varepsilon)$-fraction of the nodes that have not been removed. Following the small degrees first strategy, the algorithm chooses the remaining $1/4 - \delta/2$-fraction of degree 1 nodes, and a $\delta/2 + \varepsilon$-fraction of the degree 2 nodes. In total, $\rho_{n+1} = 1/2 + \delta + \varepsilon$, and the claim follows. □

Putting everything together, we restate and prove Proposition 3.

Proposition 3. *For the last zero problem with $p = 1/2$, no deterministic size-oblivious algorithm can have a better success guarantee than the k-max algorithm, even if we consider only instances of size larger than N_0, for any N_0.*

Proof. Claim 2 gives $\rho_{i+1} \geq \rho_i + \varepsilon$ for every stap i in the second phase of the algorithm. Therefore, there exists some t such that $\rho_t > 3/4 - \varepsilon$. Therefore, the algorithm cannot select an $1/4 + \varepsilon$ fraction of the nodes of size t. Therefore, no algorithm can achieve a success guarantee of $1/4 + \varepsilon$ for any $\varepsilon > 0$. □

1.4 Generalization to any value of p

In this subsection, we generalize the previous results beyond the case of size-oblivious deterministic algorithms for $p = 1/2$. This case is a bit more complicated, because when $p \neq 1/2$ the instances of the same size do not have the same probability of occurring. For example, for $p = 3/4$, succeeding in the instance 1^k0 contributes more to its performance for that instance size than to succeed in the instance 0^{k+1}, as the first has probability $(3/4)^k(1/4)$ to occur and the second has probability $(1/4)^{k+1}$. Recall that we associated these probabilities as weights to the nodes in the conflict graph (cf. Definition 10).

Building on the intuition of the previous section, but using more sophisticated techniques, we show what is the best possible success guarantee that any algorithm can achieve. We then link our k-max algorithm for AOSp to the conflict graph of the last zero problem, such that we finally prove the main theorem of this section: The k-max algorithm, although very simple, is optimal for AOSp for all values of p. We first focus on deterministic algorithms and prove the optimality of the k-max algorithm there. Finally, we show how to adapt the proof to randomized algorithms.

1.4.1 Local operators and average performance.

The main reason the proof techniques for the case $p = 1/2$ need to be adapted is the fact that instances of a given size do not have the same weight anymore, and therefore, the swapping argument used in Lemma 10 and Lemma 11 no longer holds. Thus, we transform a strategy using moves that select and deselect nodes from instances of different sizes: *local operators*. For a given set of selected nodes (i.e., an algorithm), these local operators might increase the weight of selected nodes for a specific size but decrease it for another size, yet positively contribute to the success guarantee of the algorithm in the end. To keep track of this, we introduce the notion of the *average performance* of an algorithm. With a *window* $[n, n+t]$ we mean the set of instances of size i such that $n \leq i \leq n+t$ for some $t \geq 1$.

Definition 16. Consider an algorithm A for the last zero problem that selects the node set A_S. The *average performance* $\pi_{A,[n,n+t]}$ of A in the window $[n, n+t]$ is the average of the performance of A in this window, i.e.,

$$\pi_{A,[n,n+t]} = \frac{1}{t+1} \sum_{i \in [n,n+t]} \mathrm{perf}(A, i) = \frac{1}{t+1} \sum_{i \in [n,n+t]} \left(\sum_{v \in A_S^i} w(v) \right) .$$

If the window is clear from the context, we drop this subscript and simply write π_A.

We will show that there exists a set of local operators that can be used to increase the average performance of an algorithm. Informally, the proof is then completed as follows. It turns out that the k-max algorithm applied to the conflict graph is very consistent, in the sense that it selects the same total weight for every size. This means that its average performance is approximately equal to the infimum of its performance for every size (i.e., its success guarantee). Therefore, if an algorithm would have a larger success guarantee than the k-max algorithm, it would also exceed the average performance in every window. In this section, we show that the latter is a contradiction.

We define the following two local operators.

Definition 17. Consider an algorithm A for the last zero problem and its set A_S of selected nodes and A_R of removed nodes. Consider the window $[n, n+t]$. Let $c(v)$ and $d(v)$ denote the set of children and descendants of a given node v, respectively.

The *high-degree local operator* can be applied if there exists a node $v \in A_S$ with $\deg(v) > 1/(1-p)$ and $|v| \in [n, n+t-1]$. It transforms A into another algorithm A' that selects $A'_S = A_S \cup c(v) \setminus \{v\}$ and removes $A'_R = A_R \setminus c(v)$.

The *low-degree local operator* can be applied if there exists a $v \notin A_S \cup A_R$ with $\deg(v) \leq 1/(1-p)$ and $|v| \in [n, n+t-1]$. It transforms A into another algorithm A'' that selects $A''_S = A_S \cup \{v\} \setminus d(v)$ and removes $A''_R = A_R \cup d(v)$.

The next lemma proves that these local operators improve the average performance of an algorithm in the window. We say that an algorithm is *valid* if it selects at most one node along each monotone path in the conflict graph.

Lemma 20. *Consider a valid deterministic algorithm A. Applying the high-degree local operator or the low-degree local operator to A on a window $[n, n+t]$ yields a new valid algorithm A' with $\pi_{A'} \geq \pi_A$.*

Proof. It is clear that the new algorithm A', obtained after applying any of the two local operators to an algorithm A, is still valid. We prove that $\pi_{A'} \geq \pi_A$ after applying these local operators.

Consider the high-degree local operator and a node $v \in A_S$ with $\deg(v) > 1/(1-p)$ and $|v| \in [n, n+t-1]$. After applying the operator, $\mathrm{perf}(A', |v|) = \mathrm{perf}(A, |v|) - w(v)$. By Lemma 6, the total weight of the children of v equals $\deg(v)w(v)(1-p) > w(v)$, since $\deg(v) > 1/(1-p)$. Therefore, $\mathrm{perf}(A', |v|+1) > \mathrm{perf}(A, |v|+1) + w(v)$ and indeed $\pi_{A'} > \pi_A$.

Now consider the low-degree local operator and a window $[n, n+t]$. We will construct a reversed sequence $A_0, A_1, \ldots, A_\ell$ of valid algorithms that starts at $A' = A_0$ and ends in $A = A_\ell$ after some number of iterations that we call ℓ. We will prove that $\pi_{A_{i+1}} \leq \pi_{A_i}$ for all $i \in \{1, \ldots, t-1\}$. Since as a base $\pi_{A_0} \leq \pi_{A'}$ is clearly true, this shows in the end that indeed $\pi_A \leq \pi_{A'}$, which is what we need to prove.

Let v be the node the operator is applied on, so that $v \notin A_S \cup A_R$ with $\deg(v) \leq 1/(1-p)$ and $|v| \in [n, n+t-1]$. For the remainder of this argument, consider only nodes in its subtree $T(v)$. Consider some fixed iteration i in which algorithm A_i transforms into algorithm A_{i+1}. Let $U_i = (A_i)_S \setminus A_S$ be the set of nodes that algorithm A_i selects but the original algorithm A does not. Let $U'_i \subset U_i$ be the subset of nodes of U_i with minimum size s_i and let $u \in U'_i$.

We will distinguish two cases. For the first case we will show that (i) $|U_{i+1}| < |U_i|$, while for the second case we will show that (ii.a) $s_{i+1} > s_i$, and otherwise, (ii.b) $|U'_{i+1}| < |U'_i|$. In other words, every transformation of an algorithm A_i into an algorithm A_{i+1} (i) either decreases the set of nodes that A_i selects but A does not, or (ii.a) the minimum instance size in this set has increased, and otherwise at least (ii.b) the subset of instances of minimum size of this set decreases.

If (ii.b) happens, at some point this subset will have cardinality 1, and then (ii.a) happens. So – interjected with some iterations in which (ii.b) happens – either $|U_{i+1}| < |U_i|$, and otherwise the smallest instance size in this set has increased. Since we do not modify instances outside of the interval $[n, n+t]$, we can apply this shrinking process only a finite number of times. This implies that in the end the set U_ℓ will be empty after some number of iterations ℓ. Indeed, this is the end of the sequence of algorithms since then $A_\ell = A$ by construction.

Let us discuss the two cases. For any node v let $c(v)$ and $d(v)$ denote the set of its children and its descendants, respectively.

First, if $A_S \cap d(u) = \emptyset$, set $(A_{i+1})_S = (A_i)_S \setminus \{u\}$. Then clearly $\pi_{A_{i+1}} \leq \pi_{A_i}$. Observe moreover that $|U_{i+1}| < |U_i|$.

Otherwise, there exists a node $u' \in A_S \cap d(u)$. In this case set $(A_{i+1})_S = (A_i)_S \cup c(u) \setminus \{u\}$. Let us compare A_{i+1} to A_i. First observe that $\text{perf}(A', |u|) = \text{perf}(A, |u|) - w(u)$, but $\text{perf}(A', |u|+1) = \text{perf}(A, |u|+1) + \sum_{v \in c(u)} w(v)$. Recall that each of the $\deg(u)$ children of u has a weight of $w(u)(1-p)$ and that $\deg(u) \leq \deg(v) \leq 1/(1-p)$ (by Lemma 5). The total weight of the children of u is thus $\deg(u)w(u)(1-p) \leq w(u)$. So, indeed, $\pi_{A_{i+1}} \leq \pi_{A_i}$. Observe moreover that the set of nodes U'_i has decreased by one. If $|U'_i| = 1$ was true, the instances in U'_{i+1} are one size larger than the instances in U_i. Otherwise, $|U'_{i+1}| < |U'_i|$ for the same size.

In both cases $\pi_{A_{i+1}} \leq \pi_{A_i}$ and the inductive properties stated above hold. As argued above, this means the procedure is finite and terminates after some number of iterations ℓ with $A_\ell = A$, and therefore, $\pi_A \leq \pi_{A'}$, completing the proof.

□

1.4.2 Fill-in strategy.

We introduced these local operators to prove that an algorithm called the *fill-in stategy*, denoted by $\mathcal{F}$, has optimal average performance. The fill-in strategy for a window $[n, n+t]$ scans the instance sizes from n to $n+t$ in increasing order, selects all the non-removed instances of degree up to $\left\lfloor \frac{1}{1-p} \right\rfloor$ for each size $s \in [n, n+t-1]$, and all the non-removed instances for instance size $n+t$. The following lemma follows immediately from the local operators we defined, using a proof by contradiction.

Lemma 21. *The fill-in strategy $\mathcal{F}$ has optimal average performance for any window $[n, n+t]$, i.e., for all size-oblivious algorithms A for the last zero problem and every window $[n, n+t]$, it holds that $\pi_{\mathcal{F},[n,n+t]} \geq \pi_{A,[n,n+t]}$.*

Proof. Consider an optimal algorithm A that is not the fill-in strategy $\mathcal{F}$. There are three possibilities in which A can be different from $\mathcal{F}$.

In the first case, there exists a node $v \notin A_S$ of size $|v| \in [n, n+t-1]$ with $\deg(v) \leq \left\lfloor \frac{1}{1-p} \right\rfloor$. But in this case, applying the low-degree operator on v to A yields a new algorithm A' with $\pi_{A'} > \pi_A$, which contradicts the optimality of A.

In the second case, there exists a node $v \notin A_S$ of size $|v| \in [n, n+t-1]$ with $\deg(v) > \left\lfloor \frac{1}{1-p} \right\rfloor$. Now we can apply the high-degree operator to strictly improve the average performance of A, and we have a contradiction.

In the last case, there exists a node $v \notin A_S \cup A_R$ of size $|v| = n+t$. But the algorithm A' with $A'_S = A_s \cup \{v\}$ has $\pi_{A'} > \pi_A$, which is again a contradiction. □

To prove the negative results of Theorem 2 it remains to show that the success guarantee of the fill-in strategy $\mathcal{F}$ does not exceed the success guarantee of the k-max algorithm for AOSp. Therefore, we now proceed to describe the k-max algorithm in the conflict graph.

1.4.3 The k-max algorithm in the conflict graph.

Recall that we can transform the k-max algorithm for AOSp to the last zero problem using the proof of Proposition 1. This yields the following description of the k-max algorithm for the last zero problem: It stops at the first 0 that arrives after it has seen the k last 1s. This means that in the conflict graph, it selects all instances at which the last 0 is the only 0 after the k-th last 1. Denote the k-max algorithm for the last zero problem by $\mathcal{K}$ for short.

To illustrate this, let $p \in [1/2, 2/3)$ such that $k = \left\lfloor \frac{1}{1-p} \right\rfloor = 2$, so the k-max algorithm sets the second largest sampled value as a threshold. Thus, in any instance of the last zero problem, $\mathcal{K}$ stops at the first 0 after the second-to-last 1. This implies that for any size n, it stops at the last 0 in the instances which end in 110 or 101. Indeed, these are the instances $\mathcal{K}$ selects in the conflict graph, and these correspond to the AOSp instances in which the k-max algorithm stops at the online element with the maximum value. Similarly, for $p \in [2/3, 3/4)$, $\mathcal{K}$ stops at the last zero (i.e., succeeds) in all instances that end in 1110, 1101 or 1011, and these form the set $\mathcal{K}_S$ of selected instances in the conflict graph.

We analyze the dynamics of $\mathcal{K}$ in the conflict graph in the following lemma.

Lemma 22. *Consider instances of size $n > k$ and consider the k-max algorithm $\mathcal{K}$ for the last zero problem, starting at instance size $k+1$ and iteratively considering instances of increasing size. For every size $n > k$, $\mathcal{K}_S^n = \{v \notin \mathcal{K}_R : |v| = n, ||v||_1 \geq k, \deg(v) \leq k\}$.*

Proof. For this proof define the *m-cut suffix* of an instance I for some value m as the last m bits in case $|I| \geq m$ and as I otherwise.

Consider the conflict graph for $n \geq k+1$ suppose by contradiction that the lemma is false. There are three cases. In the first case, there exists a node $v \in \mathcal{K}_S$ with $||v||_1 < k$. In the second case, there exists a node $v \in \mathcal{K}_S$ with $\deg(v) > k$. In the third case, there exists a node $v \notin (\mathcal{K}_S \cup \mathcal{K}_R)$ with $||v||_1 \geq k$ and $\deg(v) \leq k$.

In the first case, since there are less than k samples in the instance corresponding to v, the k-max algorithm sets a threshold of zero and accepts the first online value. So it only succeeds in v if the first online value is the maximum online value, i.e., v contains only one 0. Observe that $||v||_1 < k$ means the instance corresponding to the node contains strictly less than k 1s. But then $|v| < k+1$, contradiction.

In the second case, note that a node v with $\deg(v) > k$ has a suffix consisting of one 0 followed by at least k 1s. In such an instance, however, the k-max algorithm fails, so $v \notin \mathcal{K}_S$. Contradiction.

In the third case, let $U = \{v : v \notin \mathcal{K}_S, v \notin \mathcal{K}_R, ||v||_1 \geq k, \deg(v) \leq k\}$ and let $v \in \arg\min_{|u|}\{u \in U\}$. Denote $d = \deg(v)$ such that its suffix is 01^{d-1}. Consider the k-cut suffix v' of v and note that v' contains at least one 0. Now, as long as v' contains more than one 0, remove its last 0. Consider the unique resulting instance v'' of this procedure. As its k-suffix contains exactly one 0, note that $|v''| \geq ||v''||_1 + 1 \geq k+1$. Since both $||v''||_1 \geq k$ and $\deg(v'') \leq k$, and v was the smallest (in terms of size) such node that $\mathcal{K}$ did not select or remove, we know $v'' \in \mathcal{K}_S$ or $v'' \in \mathcal{K}_R$. But then as v is a descendant of v'', we know in both cases that $v \in \mathcal{K}_R$, contradiction. □

With this lemma we can analyze the success guarantee of $\mathcal{K}$. Indeed, this yields the same success guarantee of $kp^k(1-p)$ as the k-max algorithm for AOSp, as proved in Lemma 1.

Lemma 23. *For every $n \geq N_0$ for some N_0, perf$(\mathcal{K}, n) = (1-p)kp^k$, where $k = \left\lfloor \frac{1}{1-p} \right\rfloor$.*

Proof. For a set of nodes U let $w(U) = \sum_{u \in U} w(u)$ be its total weight. We need to show $w\left(\mathcal{K}_S^n\right) = kp^k(1-p)$ for all $n \geq N_0$ where $k = \lfloor 1/(1-p) \rfloor$. We do so by first computing $w\left(\mathcal{K}_R^n \cup R^n \cup \mathcal{K}_S^n\right)$, where $R^n = \{v : |v| = n, ||v||_1 < k\}$. From this quantity we will subtract $w\left(\mathcal{K}_R^n \cup R^n\right)$, which will give us $w\left(\mathcal{K}_S^n\right)$ as all three sets are disjoint.

Lemma 22 shows that for every size n, $\mathcal{K}_R^n \cup \mathcal{K}_S^n = \{v : |v| = n, ||v||_1 \geq k, \deg(v) \leq k\}$. Together with the set of nodes R^n of norm less than k, we see that $\mathcal{K}_R^n \cup R^n \cup \mathcal{K}_S^n = \{v : |v| = n, \deg(v) \leq k\}$. Then

$$w\left(\mathcal{K}_R^n \cup R^n \cup \mathcal{K}_S^n\right) = \sum_{i=1}^{k} w_i = (1-p)\sum_{i=0}^{k-1} p^i \,.$$

Consider all children v of nodes of size n that have degrees 1 up to k, together with the nodes of size $n+1$ with norm less than k. In the latter case $v \in R^{n+1}$. In the former case the parent of v is in $\mathcal{K}_R^n \cup \mathcal{K}_S^n$ and therefore $v \in \mathcal{K}_R^{n+1}$. Therefore, $\mathcal{K}_R^{n+1} \cup R^{n+1}$ consists of all children of nodes of size n that have degrees 1 up to k. Recalling the degree structure from Lemma 5, we see that a selected instance of degree i has one descendant of weight w_{ij} for all $j = 1, \ldots, i$. Therefore,

$$\begin{aligned} w\left(\mathcal{K}_R^n \cup R^n\right) &= w_{11} + (w_{21} + w_{22}) + \ldots + (w_{k1} + \ldots + w_{kk}) \\ &= (1-p)w_1 + 2(1-p)w_2 + \ldots + k(1-p)w_k \\ &= (1-p)\sum_{i=1}^{k} i w_i = (1-p)^2 \sum_{i=0}^{k-1} (i+1)p^i \,, \end{aligned}$$

and the instances of size n that will be selected by $\mathcal{K}$ have total weight

$$w\left(\mathcal{K}_S^n\right) = w\left(\mathcal{K}_R^n \cup R^n \cup \mathcal{K}_S^n\right) - w\left(\mathcal{K}_R^n \cup R^n\right)$$

$$
\begin{aligned}
&= (1-p)\left[\sum_{i=0}^{k-1} p^i - (1-p)\sum_{i=0}^{k-1}(i+1)p^i\right] \\
&= (1-p)\sum_{i=0}^{k-1}\left[1-(i+1)(1-p)\right]p^i \\
&= (1-p)\sum_{i=0}^{k-1}\left[(i+1)p^{i+1} - ip^i\right] \\
&= (1-p)kp^k\,.
\end{aligned}
$$

This completes the proof. □

1.4.4 Connecting the fill-in strategy to the k-max algorithm.

Lemma 22 allows us to compare the performance of the fill-in strategy $\mathcal{F}$ to the performance of the k-max algorithm for the last zero problem, $\mathcal{K}$. In fact, they select almost the same nodes in the conflict graph.

Lemma 24. *Consider a window $[n, n+t]$.*

If $n > 1$, then for every size $s \notin \{n, n+t\}$, $\mathcal{F}_S^s = \mathcal{K}_S^s$. For sizes $s \in \{n, n+t\}$, $\mathcal{K}_S^s \subset \mathcal{F}_S^s$ (i.e., $\mathcal{K}$ selects a strict subset of the set of nodes selected by $\mathcal{F}$).

If $n = 1$, then $\mathcal{F}_S^1 = \mathcal{K}_S^1$ as well.

Proof. First we prove that $\mathcal{K}_S^n$ is a strict subset of $\mathcal{F}_S^n$. Suppose that both $\mathcal{F}$ and $\mathcal{K}$ start with size $n > 1$, so $\mathcal{F}_R^n = \mathcal{K}_R^n = \emptyset$. Writing $k = \lfloor 1/(1-p) \rfloor$, observe that $\mathcal{F}_S^n = \{v : |v| = n, \deg(v) \leq k\}$ and $\mathcal{K}_S^n = \{v : |v| = n, \deg(v) \leq k, ||v||_1 \geq k\}$, completing the proof.

We will now prove that for sizes $n < s < n+t$, $\mathcal{F}_S^s = \mathcal{K}_S^s$. It is clear that $\mathcal{K}_S^s \subseteq \mathcal{F}_S^s$. We prove $\mathcal{F}_S^s \subseteq \mathcal{K}_S^s$ by contradiction, so assume there exists a $v \in \mathcal{F}_S^s \setminus \mathcal{K}_S^s$, i.e., $\deg(v) \leq k$ and $||v||_1 < k$. Without loss of generality, assume that $v \in \arg\min_{|u|} \{u \in \mathcal{F}_S^s \setminus \mathcal{K}_S^s\}$. We consider two cases: v has a parent w of size $s-1$ or v has no parent.

In the first case, note that $w \notin \mathcal{F}_S$. Otherwise, $v \in \mathcal{F}_R$ and thus $v \notin \mathcal{F}_S$, contradiction. Also note that $\deg(w) \leq k$. Otherwise, w would have suffix 01^d for $d \geq k$, but then $||w||_1 \geq k$ and therefore $||v||_1 \geq ||w||_1 \geq k$, contradiction. But if $\deg(w) \leq k$, $\mathcal{F}$ would select w unless it selected a node of whom w is a descendant, and thus $w \in \mathcal{F}_R$. But then also $v \in \mathcal{F}_R$, contradiction.

In the second case, note that nodes without a parent are exactly the nodes that have at most one 0. In the single instance that contains no 0s, $\mathcal{F}$ and $\mathcal{K}$ make the same decision by definition, so we restrict ourselves to instances that contain exactly one 0. Since $||v|| \leq k$, the k-max strategy sets a threshold of 0 and succeeds, since the only 0 is the maximum online value. But then $v \in S_2$, contradiction.

We wrap up the first part of the proof by considering the size $s = n+t$. Here, $\mathcal{F}$ selects all non-removed nodes, while $\mathcal{K}$ selects all non-removed nodes v with $\deg(v) \leq k$ and $||v||_1 \geq k$. Observe that $\mathcal{K}_R^s = \mathcal{F}_R^s$ and the set of non-removed nodes contains nodes of degree more than k and norm less than k, so indeed $\mathcal{K}_S^s \subset \mathcal{F}_S^s$.

Finally, if $n = 1$, both $\mathcal{F}$ and $\mathcal{K}$ select instance 0 and cannot succeed in instance 1, so in this case they select exactly the same nodes also in the first size of the window. □

Combining everything, the negative result of Theorem 2 for deterministic size-oblivious algorithms follows, as $\pi_{\mathcal{K},[n,n+t]} \geq \pi_{\mathcal{F},[n,n+t]} - 2/(t+1)$ and the latter term vanishes.

Theorem 2. *Let $k = \lfloor 1/(1-p) \rfloor$. The k-max algorithm achieves a guarantee of $kp^k(1-p)$ for AOSp. Furthermore, no algorithm can achieve a better success guarantee.*

Proof of the negative result for size-oblivious deterministic algorithms. To prove the theorem, suppose by contradiction that there exists a size-oblivious deterministic algorithm A with $\text{perf}(A, n) \geq kp^k(1-p) + \varepsilon$ for some $\varepsilon > 0$ for every size n (larger than some size N_0) with $k = \lfloor 1/(1-p) \rfloor$. Let $t > 2/\varepsilon - 1$ and consider the window $I = [N_0, N_0 + t]$. Since the average performance is at least as good as the worst case performance, $\pi_{A,I} \geq kp^k(1-p) + \varepsilon$.

Consider $\mathcal{K}$ and $\mathcal{F}$ starting at size N_0 and consider the window I. First, note that Lemma 24 implies that $\text{perf}(\mathcal{F}, s) \leq \text{perf}(\mathcal{K}, s) + 1$ for $s \in \{N_0, N_0 + t\}$ and $\text{perf}(\mathcal{F}, s) = \text{perf}(\mathcal{K}, s)$ for $s \in (N_0, N_0 + t)$. So $\pi_{\mathcal{F},I} \leq \pi_{\mathcal{K},I} + 2/(t+1)$.

Lemma 23 argues that the performance of $\mathcal{K}$ is very consistent, in the sense that it selects the same total weight for every size. Therefore, $\pi_{\mathcal{K},I}$ is arbitrarily close to $kp^k(1-p)$, since that is its worst case performance. In particular, we can choose any $\delta > 0$ such that $\pi_{\mathcal{K},I} \leq kp^k(1-p) + \delta$ and $\pi_{\mathcal{F},I} \leq kp^k(1-p) + \delta + 2/(t+1)$.

Choosing $\delta < \varepsilon - 2/(t+1)$ yields $\pi_{\mathcal{F},I} < \pi_{A,I}$. However, $\pi_{\mathcal{F},I}$ is optimal by Lemma 21. Therefore, this is a contradiction since A cannot be better.

□

1.5 Generalization to randomized algorithms

In this subsection we adapt the above proof to randomized algorithms by generalizing Lemma 20 to the randomized setting. The rest of the proof follows immediately from the same arguments as for deterministic algorithms if in the definition of the average performance (cf. Definition 16) we multiply the weight of a node by its selection probability, so extending this lemma suffices to extend the negative results to randomized algorithms.

For a node v and a randomized algorithm A, let $A_S(v)$ and $A_R(v)$ be its selection probability and its removed fraction (cf. Lemma 8), respectively. Recall that a node selected with probability $A_S(v)$ removes a fraction $A_S(v)$ of its descendants.

For randomized algorithms, we define the following two local operators similar to Definition 17.

Definition 18. Consider a randomized algorithm A for the last zero problem and consider the window $[n, n+t]$. Let $c(v)$ and $d(v)$ denote the set of children and descendants of a given node v, respectively.

The *randomized high-degree local operator* can be applied if there exists a node v with $A_S(v) > 0$, $\deg(v) \leq 1/(1-p)$ and $|v| \in [n, n+t-1]$. It transforms A into another randomized algorithm A' with $A'_S(v) = 0$ and $A'_R(w) = A_R(w) + A_S(v)$ for all $w \in c(v)$.

The *randomized low-degree local operator* can be applied if there exists a node v with $A_S(v) + A_R(v) < 1$, $\deg(v) \leq 1/(1-p)$ and $|v| \in [n, n+t-1]$. Defining $\varepsilon = 1 - A_S(v) - A_R(v) > 0$, it transforms A into another randomized algorithm A'' with $A''_S(v) = A_S(v) + \varepsilon$, and $A_S(w) = 0$ and $A_R(w) = 1$ for all $w \in d(v)$.

We say that a randomized algorithm *valid* if the sum of $q_s(v)$ over all vertices v of a monotone path in the conflict graph is at most 1, for all monotone paths.

Lemma 25. *Consider a valid randomized algorithm A. Applying the randomized high-degree local operator or the randomized low-degree local operator to A on a window $[n, n+t]$ yields a new valid randomized algorithm A' with $\pi_{A'} \geq \pi_A$.*

Proof. For both local operators, the claim that applying them does not decrease the average performance follows from the arguments of Lemma 20, so in this proof we will show that both local operators result in a valid algorithm. Let v be the node under consideration and for any node w denote by $q'_s(w)$ and $q'_r(w)$ its selection probability and its removed fraction, respectively, after applying one of the local operators.

Consider the high-degree local operator and any monotone path $P = (v, v_1, v_2, \ldots)$. Note that every monotone path contains exactly one child of v. Then

$$\sum_{w \in P} q'_s(w) = q'_s(v) + q'_s(v_1) + \sum_{i \geq 2} q'_s(v_i) = 0 + (q_s(v_1) + q_s(v)) + \sum_{i \geq 2} q_s(v_i) = \sum_{w \in P} q_s(w)\,.$$

So if the original algorithm was valid, so is the algorithm after applying this operator.

For the low-degree operator, note that we change $q_s(v)$ to $q_s(v) + \varepsilon = q_s(v) + 1 - q_s(v) - q_r(v) = 1 - q_r(v)$. Therefore, after applying the operator, we have $q_s(v) + q_r(v) = 1$. Since in general for any child w of v we have $q_r(w) = q_s(v) + q_r(v)$, we see that $q_s(w) \leq 1 - q_r(w) = 1 - 1 = 0$, which is what we needed to prove. □

With this lemma, the proof of the negative results of Theorem 2 is complete, except for the final assumption we need to drop: The proof also holds for algorithms that are not size-oblivious.

1.6 Generalization to non-size-oblivious algorithms

We now prove that if an algorithm is not size-oblivious, it cannot perform better asymptotically. To do so, we first introduce a colored variant of the last zero problem. For a colored string s, its *red norm* $||s||_r$ and *blue norm* are the number of red 1s and blue 1s s contains, respectively.

Definition 19. The *colored last zero problem* is the following;

- Phase 1: An adversary picks two integers m and n, with $m \leq n$.
- Phase 2: A string of bits of length n is generated, where in each position independently the bit equals 1 with probability p and 0 otherwise.
- Phase 3: We color the entries 1 to m with red, while the entries $m+1$ to n are colored blue.
- Phase 4: The red norm, blue norm and length of the string are revealed to the player.
- Phase 5: The bits of the string are presented to the player one after the other. The player chooses at each step to continue or to stop the game. If the player continues, the next bit is presented to her.

The player wins if she stops on the last red 0 of the string.

Note that now the player has three numbers to start with: the number of red samples $r := ||s||_r$, the number of blue samples $b := ||s||_b$ and the size $|s| = n$.

Proposition 4. *The colored last zero problem is equivalent to a specific instance of AOSp with known size n. Therefore, any negative result for the colored last zero problem also holds for AOSp.*

Proof. An algorithm A again only succeeds if it stops at the element of the online set with the largest value, only that now it knows in advance how many online elements it is going to observe. Imagine now that A is facing an instance of the following form: The first m elements are assigned a series of positive strictly increasing values, and the remaining $n-m$ take arbitrary negative values. Thus, in this instance A is aiming for the last non-sampled element among the first m. This is basically the same game as the colored last zero problem, where the red values correspond to the positive values and the blue values correspond to the negative ones. The formal proof is almost the same as the proof of Proposition 1. □

With this analog of Proposition 1, we are going to prove the following theorem in the remainder of this section.

Theorem 6. *In the colored last zero problem, no algorithm can achieve performance $kp^k(1-p)+\varepsilon$ on every size $n \geq N_0$ (for some $N_0 > 0$) for any $\varepsilon > 0$.*

Intuitively, the colored last zero problem is not much different from the case without colors: there is still an unknown point in the string where the algorithm should stop, and there is still a string of bits before this point (the red bits). The only difference is that the algorithm is not size-oblivious and is also given the total number of 1s in the last $n-m$ bits (the blue bits). At first sight these blue 1s seem useless, because the algorithm wants to stop before reaching them. On the other hand, the fact that an algorithms knows how many they are, gives an indication about the size of $n-m$ and this could be already enough to improve the performance. We show that this is not the case. To do so, we define a slightly different conflict graph, and study its structure to show that up to negligible terms the dynamics are the same as for the standard conflict graph.

Modified conflict graph.

For the colored last zero problem, m basically plays the role that n was playing before. Therefore, the different layers of the conflict graph correspond to the various sizes of m in this case, and there is a separate conflict graph for each value of n. Note that for a given size n its conflict graph has a finite number of layers as m varies between 1 and n.

A node of the graph is a couple (s, b), where s is a string of bits of length m, that represents the string of red bits, and b is an integer that represents the number of blue 1s. The exact positions of 0s and 1s in the blue bits are irrelevant for our proof, only the total number of blue 1s matters.

Finally, just as before, the nodes have different weights, with the difference here that the weights also depend on b. In particular, the weight of a node (s, b) is

$$p^{r+b}(1-p)^{n-r-b}\binom{n-m}{b}.$$

Indeed, the probability that $r+b$ 1s occur in an instance of size n when sampling with probability p is $p^{r+b}(1-p)^{n-r-b}$, where r and b are the number of red 1s and blue 1s respectively. As in the conflict graph all the instances with b blue 1s are grouped together, the weight of the combined node is multiplied by the total number of such instances.

Conflict structure.

Now let us consider the conflicts. One can see that two nodes (s, b) and (s', b') are in conflict if and only if $b = b'$, and $s \nsim s'$ (in the sense of the standard conflict graph). Note that for an instance and its descendants the values b, r and n are the same. In other words, to move from size m to size $m+1$ we can add a 0 in the appropriate position, just as in Lemma 4, and monotone paths in the conflict graph determine conflicts again.

We now study the relation between the weights of an instance and its children. Let I_1 be a node with a string s of size m and let I_2 be one of its children (note that I_2 has size $m+1$ and is in conflict with I_1). Let p_1 and p_2 be the weights associated with these nodes. We derive from the formula above that:

$$\frac{p_2}{p_1} = \frac{\binom{n-m-1}{b}}{\binom{n-m}{b}} = \frac{n-m-b}{n-m}.$$

Having defined the modified conflict graph, we are now ready to show the main result of this section.

Proof of Theorem 6. Consider again the ratio p_2/p_1. The expected value of b is $(n-m)p$, but this will not be the case for all instances that we consider. For large values of $n-m$ though, we can apply standard concentration arguments (see e.g. Lemma 27) and obtain that with high probability we have

$$\frac{(n-m)-(n-m)p-\varepsilon}{n-m} \leq \frac{p_2}{p_1} \leq \frac{(n-m)-(n-m)p+\varepsilon}{n-m},$$

which is true if and only if

$$1-p-\varepsilon' \leq \frac{p_2}{p_1} \leq 1-p+\varepsilon',$$

where $\varepsilon' = \frac{\varepsilon}{n-m}$. From here it is easy to observe that when ε takes a value very close to 0, so does ε'. Furthermore, as $n-m$ grows, ε' vanishes. Thus the modified conflict graph has the same weight distribution as in Lemma 6 with high probability.

Therefore, with high probability, the modified conflict graph is (almost) the same as the weighted conflict graph from Section 3.2.3. Thus, we can follow again the arguments for size-oblivious algorithms, since they all hold in this case too. We end up with the same impossibility results, which hold here both for deterministic and for randomized algorithms as well. □

2 Omitted proofs from Section 4

Proof of Lemma 12. Fix a sequence t and a sampling probability p. We use a coupling argument between realizations of the arrival times in instances with n and $n+1$ values. We start with an instance $\alpha_1, \ldots, \alpha_{n+1}$, and assume the values are indexed in decreasing order. Consider a realization of the arrival times $\tau_1 = \tau'_1, \ldots, \tau_{n+1} = \tau'_{n+1}$ and couple it with the corresponding realization $\tau_1 = \tau'_1, \ldots, \tau_n = \tau'_n$ in the instance $\alpha_1, \ldots, \alpha_n$. Assume that in the instance with n values and for this particular realization of the arrival times, ALG_t fails. This means that $V \setminus \{\alpha_{n+1}\}$ is non-empty and either ALG_t never stops or it accepts a value that is not the maximum of $V \setminus \{\alpha_{n+1}\}$. Note that regardless of τ'_{n+1}, the rankings of the values in $V \setminus \{\alpha_{n+1}\}$ are the same in both instances because α_{n+1} is smaller than all other values. Thus, if $\tau'_{n+1} < p$, ALG_t does

not succeed either when applied in the instance of $n+1$ values. On the other hand, if $\tau'_{n+1} > p$, we have to distinguish between two cases. If ALG_t accepts α_{n+1}, it fails, because $V \setminus \{\alpha_{n+1}\}$ is non-empty and then α_{n+1} cannot be the largest in V. If ALG_t does not accept α_{n+1}, then the behavior of ALG_t in the rest of the variables is the same as in the instance with n values and then it fails.

Since the distribution of $\tau_1, \dots, \tau_n$ is the same in both instances, we conclude with this argument that the probability that ALG_t fails in the instance with $n+1$ values is at least as large as in the instance with n values. □

Proof of Lemma 13. We first calculate the success probability of ALG_t for fixed p and n and then take the limit when n tends to infinity.

We say a value α_i is *acceptable* for ALG_t (for a particular realization of the arrival times) if $p < \tau_i$, for some $j \in \mathbb{N}$ we have that $t_j \le \tau_i < t_{j+1}$, and α_i is larger than the j-th largest value in S. Now, note that if $\max V$ is not acceptable for ALG_t, then ALG_t does not stop. This is because we restricted the sequence t to be increasing, so values that arrive before $\max V$ are not acceptable, and values arriving after $\max V$ will not be the best seen so far from V. We use this to decompose the success probability as follows.

$$\mathbb{P}(ALG_t \text{ succeeds}) = \mathbb{P}(\max V \text{ is acceptable}) - \mathbb{P}(ALG_t \text{ stops before seeing } \max V)\,. \tag{1}$$

In this definition, if V is empty we also say $\max V$ is acceptable. We first calculate the probability that $\max V$ is acceptable. Assume that the values are indexed in decreasing order, i.e., that $\alpha_1 > \cdots > \alpha_n$.

$$\begin{aligned}
\mathbb{P}(\max V \text{ is acceptable}) &= \mathbb{P}(V = \emptyset) + \sum_{i=1}^{n} \mathbb{P}(\max V = \alpha_i) \cdot \mathbb{P}(t_i \le \tau_i \mid \max V = \alpha_i) \\
&= p^n + \sum_{i=1}^{n} p^{i-1}(1-p) \cdot \frac{1 - \max\{p, t_i\}}{1-p} \\
&= p^n + \sum_{i=1}^{n} p^{i-1} \left(1 - \max\{p, t_i\}\right)\,. \qquad (2)
\end{aligned}$$

By the same argument, ALG_t stops before seeing $\max V$ if and only if at least one value arrives after p and before the arrival time of $\max V$, and the maximum such value is acceptable.

$$\begin{aligned}
&\mathbb{P}(ALG_t \text{ stops before seeing } \max V) \\
&= \sum_{j=1}^{n} \mathbb{P}(\max V = \alpha_j) \cdot \mathbb{P}(\text{maximum before } \max V \text{ is acceptable} \mid \max V = \alpha_j) \\
&= \sum_{j=1}^{n} \mathbb{P}(\max V = \alpha_i) \sum_{i=j}^{n-1} \mathbb{P}\Big(\text{max. in } [p, \tau_j) \text{ has rank } i \text{ among elements that arrive in } [0, \tau_j), \\
&\qquad\qquad \text{and arrives in } [t_i, \tau_j) \;\Big|\; \max V = \alpha_j\Big) \\
&= \sum_{j=1}^{n} p^{j-1}(1-p) \sum_{i=j}^{n-1} \frac{1}{1-p} \int_p^1 \mathbb{P}\Big(\text{max. in } [p, t) \text{ has rank } i \text{ among elements that arrive in } [0, t), \\
&\qquad\qquad \text{and arrives in } [t_i, t) \;\Big|\; \max V = \alpha_j, \tau_j = t\Big)\, \mathrm{d}t \\
&= \sum_{j=1}^{n} p^{j-1}(1-p) \sum_{i=j}^{n-1} \frac{1}{1-p} \int_{\max\{p,t_i\}}^1 \left(\frac{p}{t}\right)^{i-j} \cdot \frac{(t - \max\{p, t_i\})}{t} \\
&\qquad\qquad \cdot \mathbb{P}(\text{at least } i \text{ values arrive before } t \mid \max V = \alpha_j, \tau_j = t)\, \mathrm{d}t \\
&= \sum_{j=1}^{n} p^{j-1} \sum_{i=j}^{n-1} \int_{\max\{p,t_i\}}^1 \left(\frac{p}{t}\right)^{i-j} \cdot \frac{(t - \max\{p, t_i\})}{t} \Big(1 - B_{t,n-j}(i-j+1)\Big)\, \mathrm{d}t
\end{aligned}$$

$$= \sum_{i=1}^{n-1} p^{i-1} \int_{\max\{p,t_i\}}^{1} \sum_{j=1}^{i} \frac{t - \max\{p,t_i\}}{t^j} \Big(1 - B_{t,n-j}(i-j+1)\Big)\, \mathrm{d}t\,, \tag{3}$$

where $B_{p,n}(x) = \sum_{i=0}^{x} \binom{n}{i} p^i (1-p)^{n-i}$ is the CDF of a Binomial distribution of parameters p and n. Note that for any fixed integers i and j, and time $t \in (0,1)$, $B_{t,n-j}(i-j+1)$ converges to 0 when n tends to infinty. Therefore, replacing Eq. (2) and Eq. (3) in Eq. (1), and taking the limit when n tends to infinity, we conclude the proof of the lemma. □

We note that as a by-product of the previous proof we can compute related statistics such as the probability that the algorithm stops. Indeed the event that the algorithm stops exactly coincides with the event that the maximum element of V is acceptable. This holds because first, if the maximum is acceptable, then clearly, the algorithm stops. And, second, if the algorithm stops, then there is an acceptable element in V which is either the maximum or an element that arrives earlier. In the latter case, the maximum element in V is also acceptable since its relative rank can only be better. We thus conclude that

$$\mathbb{P}(ALG_t \text{ stops}) = p^n + \sum_{i=1}^{n} p^{i-1} \left(1 - \max\{p,t_i\}\right)\,.$$

Proof of Lemma 14. First, we relax the monotonicity constraint on the sequence of t_i's. The resulting relaxed optimization problem is separable, i.e., optimizing over the entire sequence is equivalent to optimizing over each variable independently. For each t_i we get the following equivalent problem.

$$\max_{t_i \in [0,1]} p^{i-1} \left(1 - \max\{p,t_i\} - \int_{\max\{p,t_i\}}^{1} \sum_{j=1}^{i} \frac{t - \max\{p,t_i\}}{t^j}\, \mathrm{d}t\right)\,.$$

Equivalently, we can remove the factor p^{i-1} and restrict t_i to be in $[p,1]$, obtaining

$$\max_{t_i \in [p,1]} 1 - t_i - \int_{t_i}^{1} \sum_{j=1}^{i} \frac{t - t_i}{t^j}\, \mathrm{d}t\,.$$

Denoting by $G_i(t_i)$ this objective function, we get that

$$\frac{d}{dt_i} G_i(t_i) = -1 + \int_{t_i}^{1} \sum_{j=1}^{i} \frac{1}{t^j}\, \mathrm{d}t\,, \text{ and } \quad \frac{d^2}{dt_i^2} G_i(t_i) = -\sum_{j=1}^{i} \frac{1}{t_i^j}\,.$$

Therefore, $G_i(t_i)$ is a concave function and then the optimum is $\max\{p, t_i^*\}$, where t_i^* is the solution of $\frac{d}{dt_i} G_i(t_i) = 0$. In the original objective function t_i appears always as $\max\{p,t_i\}$ so there we can simply take t_i^* as the solution. Now we prove that t_i^* is actually increasing in i, so it is also the optimal solution before doing the relaxation. In fact, t_i^* satisfies

$$\int_{t_i^*}^{1} \sum_{j=1}^{i} \frac{1}{t^j}\, \mathrm{d}t = 1\,.$$

Note that the left-hand side of this equation is decreasing in t_i^*, and is increasing in i. Thus, necesarily $t_i^* \le t_{i+1}^*$, for all $i \ge 1$. We conclude that t_i^* satisfies Eq. (2) by simply integrating on the left-hand side of the last equation. □

Proof of Lemma 15. We study the optimal ordinal policy obtained with backward induction, and prove that it is in fact a sequential-ℓ-max algorithm for certain ℓ. Recall that we can assume the optimal policy is ordinal, so this algorithm will be optimal not only among ordinal algorithms.

Denote by $X_i = \alpha_{\pi(i)}$ the i-th value, in the order of increasing arrival times. Denote by $R(X_1, \ldots, X_j)$ the relative ranks of values $X_1, \ldots, X_j$. In what comes, we use the notation $R(X_1, \ldots, X_j) = x$ to condition

on a particular realization x of the relative ranks. Let x be a realization of the ranks such that X_j is the maximum in V so far, i.e., $X_j = \max V \cap \{X_1, \cdots, X_j\}$, and has rank r among $X_1, \cdots, X_j$. Then,

$$\begin{aligned}
&\mathbb{P}\Big(X_j = \max V \;\Big|\; R(X_1, \ldots, X_j) = x\Big) \\
&= \mathbb{P}\Big(X_{j+1}, \ldots, X_n \text{ have overall rank at most } r+1 \;\Big|\; R(X_1, \ldots, X_j) = x\Big) \\
&= \mathbb{P}\Big(X_{j+1}, \ldots, X_n \text{ have overall rank at most } r+1\Big) \\
&= \prod_{s=0}^{r-1} \frac{j-s}{n-s}\,.
\end{aligned}$$

The optimal policy is to accept X_j if this probability is larger or equal than the probability of picking $\max V$ after rejecting X_j if from $j+1$ onwards we use the optimal policy, conditional on $R(X_1, \ldots, X_j) = x$.

Let now x' be a realization of $R(X_1, \ldots, X_{j+1})$ such that the relative rank of the best of V up to step $j+1$ is r. Suppose that conditional on $R(X_1, \ldots, X_{j+1}) = x'$, the probability of winning if we use the optimal strategy from $j+2$ onwards depends solely of n, $j+1$ and the relative rank r, for all possible ranks r. Denote this conditional probability by $W(n, j+1, r)$. We want to inductively prove that this is in fact true for all n, j and r. It is of course true in the last step, when $j+1 = n$, so we do induction on j. Let x'' be a realization of $R(X_1, \ldots, X_j)$ such that the relative rank of the best of V up to step j is r. We have that

$$\begin{aligned}
&\mathbb{P}\Big(\text{win after } j \;\Big|\; R(X_1, \ldots, X_j) = x''\Big) \\
&= \mathbb{P}\Big(X_{j+1} \text{ has relative rank } \geq r+1 \;\Big|\; R(X_1, \ldots, X_j) = x''\Big) \cdot W(n, j+1, r) \\
&\quad + \sum_{r'=1}^{r} \mathbb{P}\Big(X_{j+1} \text{ has relative rank } r' \;\Big|\; R(X_1, \ldots, X_j) = x''\Big) \\
&\qquad \cdot \max\left\{ W(n, j+1, r'), \prod_{s=0}^{r'-1} \frac{j+1-s}{n-s} \right\}. \qquad (4)
\end{aligned}$$

But for all x,

$$\mathbb{P}\Big(X_{j+1} \text{ has relative rank } r' \;\Big|\; R(X_1, \ldots, X_j) = x\Big) = \frac{1}{j+1}\,.$$

This proves the inductive step. Therefore, $W(n, j, r)$ is well defined for all n, j and r, and the optimal policy accepts X_j that has relative rank r and is the maximum so far in V if and only if

$$\prod_{s=0}^{r-1} \frac{j-s}{n-s} \geq W(n, j, r)\,.$$

From Eq. (4) it is easy to check that $W(n, j, r)$ is decreasing in j for fixed n, r and increasing in r for fixed n, j.[1] Therefore the optimal policy is the sequential-ℓ-max algorithm, for ℓ defined as

$$\ell(j) = \max\left\{ r : \prod_{s=0}^{r-1} \frac{j-s}{n-s} \geq W(n, j, r) \right\}.$$

This concludes the proof of the lemma. □

[1] At an intuitive level it is also easy to be convinced of this: as time passes it is harder to win, and if only low values (with large rank) have appeared, it is easier to win in the future.

Proof of Lemma 16. We calculate first the probability of some events. For $i \in \{h+1, \dots, n\}$, denote by A_i the event that the i-th element is the largest of V and the algorithm never stops. Notice that A_i is equivalent to the event that the overall largest $\hat{\ell}(i)$ elements are in S, and the i-th element is the largest of V (for this equivalence it is necessary that ℓ is non-decreasing). Therefore, we have that

$$\mathbb{P}(A_i) = \frac{1}{n-h} \prod_{j=0}^{\ell(\hat{i})-1} \frac{h-j}{n-j}.$$

Note that this is 0 if $\hat{\ell}(i) = h+1$. Now, for $h+1 \le r \le i \le n$, define $B_{r,i}$ the event that the r-th element is the largest among positions $\{h+1, \dots, i\}$ and the algorithm does not stop before $i+1$. This is equivalent to the event that the r-th element is the largest among positions $\{h+1, \dots, i\}$ and the largest $\hat{\ell}(r)$ elements among positions $\{1, \dots, i\}$ are in S. Thus,

$$\mathbb{P}(B_{r,i}) = \frac{1}{i-h} \prod_{j=0}^{\hat{\ell}(r)-1} \frac{h-j}{i-j}.$$

Now, note that $B_{r,i} \setminus A_r$ is the event that the r-th element is the largest among positions $\{h+1, \dots, i\}$, but not of V, and the algorithm does not stop before $i+1$. Note also that $A_r \subseteq B_{r,i}$. Therefore, the probability that the algorithm does not stop before $i+1$ and the maximum of V is among positions $\{i+1, \dots, n\}$ is

$$\sum_{r=h+1}^{i} \mathbb{P}(B_{r,i}) - \mathbb{P}(A_r) = \sum_{r=h+1}^{i} \frac{1}{i-h} \prod_{j=0}^{\hat{\ell}(r)-1} \frac{h-j}{i-j} - \frac{1}{n-h} \prod_{j=0}^{\hat{\ell}(r)-1} \frac{h-j}{n-j}.$$

Conditional on this event, the probability that the number in the $i+1$-th position is the largest of V is $1/(n-i)$, because the relative order within positions $\{i+1, \dots, n\}$ is independent of this event. Thus, we obtained the probability that the $i+1$-th element is the largest of V and the algorithm does not stop before $i+1$. To obtain the probability of winning in step $i+1$, we have to subtract the probability that the $i+1$-th element is the largest of V, but the algorithm never stops, i.e., $\mathbb{P}(A_{i+1})$. Therefore, the probability of winning at step $i+1$ is

$$\frac{1}{n-i} \sum_{r=h+1}^{i} \left(\frac{1}{i-h} \prod_{j=0}^{\hat{\ell}(r)-1} \frac{h-j}{i-j} - \frac{1}{n-h} \prod_{j=0}^{\hat{\ell}(r)-1} \frac{h-j}{n-j} \right) - \frac{1}{n-h} \prod_{j=0}^{\hat{\ell}(i+1)-1} \frac{h-j}{n-j}.$$

The probability of winning at step $h+1$ is slightly different, because the algorithm never stops before it. In that case the probability of winning is

$$\frac{1}{n-h} \left(1 - \prod_{j=0}^{\hat{\ell}(h+1)-1} \frac{h-j}{n-j} \right).$$

Adding these expressions concludes the proof of the lemma. □

Proof of Lemma 17. First we show that the function ℓ that maximizes Eq. (3), in a certain sense converges to a function $\tilde{\ell} : (0,1) \to \mathbb{N}$. Then, we do a Riemann sum analysis to show that the success probability of the sequential-ℓ-max algorithm converges to an expression in terms of $\tilde{\ell}$, and then we show that this can be equivalently expressed as Eq. (1) for some sequence t.

Except for terms that vanish when n tends to infinity, Eq. (3) can be rewritten as

$$\sum_{r=h+1}^{n} \left(\sum_{i=r}^{n} \frac{1}{n-i} \left(\frac{1}{i-h} \prod_{j=0}^{\hat{\ell}(r)-1} \frac{h-j}{i-j} - \frac{1}{n-h} \prod_{j=0}^{\hat{\ell}(r)-1} \frac{h-j}{n-j} \right) - \frac{1}{n-h} \prod_{j=0}^{\hat{\ell}(r)-1} \frac{h-j}{n-j} \right). \tag{5}$$

To find the optimal $\ell(r)$ we simply maximize the following term as a function of s.

$$F_n(r,s) = \sum_{i=r}^{n} \frac{1}{n-i}\left(\frac{1}{i-h}\prod_{j=0}^{s-1}\frac{h-j}{i-j} - \frac{1}{n-h}\prod_{j=0}^{s-1}\frac{h-j}{n-j}\right) - \frac{1}{n-h}\prod_{j=0}^{s-1}\frac{h-j}{n-j}\,.$$

Between s and $s+1$ the change is

$$\begin{aligned}
&F_n(r,s+1) - F_n(r,s)\\
&= \sum_{i=r}^{n} \frac{1}{n-i}\left(\frac{\frac{h-s}{i-s}-1}{i-h}\prod_{j=0}^{s-1}\frac{h-j}{i-j} - \frac{\frac{h-s}{n-s}-1}{n-h}\prod_{j=0}^{s-1}\frac{h-j}{n-j}\right) - \frac{\frac{h-s}{n-s}-1}{n-h}\prod_{j=0}^{s-1}\frac{h-j}{n-j}\\
&= \sum_{i=r}^{n} \frac{1}{n-i}\left(-\frac{1}{i-s}\prod_{j=0}^{s-1}\frac{h-j}{i-j} + \frac{1}{n-s}\prod_{j=0}^{s-1}\frac{h-j}{n-j}\right) + \frac{1}{n-s}\prod_{j=0}^{s-1}\frac{h-j}{n-j}\\
&= \beta(n,s,h)\left(\sum_{i=r}^{n}\frac{1}{n-i}\left(1-\frac{n-s}{i-s}\prod_{j=0}^{s-1}\frac{n-j}{i-j}\right)+1\right),
\end{aligned}$$

where $\beta(n,s,h)$ is a positive term, so the sign of this difference is not affected by it. The other term is decreasing in s, so $F_n(r,s)$ is maximized when this differences changes sign. In other words, it is maximized in

$$\ell_n^*(i) = \min\left\{s \in [n] : \sum_{i=r}^{n}\frac{1}{n-i}\left(1-\prod_{j=0}^{s}\frac{n-j}{i-j}\right)+1 \le 0\right\}.$$

Now, doing a Riemann sum analysis, we have that $\tilde{\ell}(\tau) = \lim_{n\to\infty}\ell_n^*(\lfloor \tau n \rfloor)$ satisfies

$$\tilde{\ell}(\tau) = \min\left\{s \in \mathbb{N} : \int_\tau^1 \left(\frac{1}{1-t}\left(1-\frac{1}{t^{s+1}}\right)+1\right)\mathrm{d}t \le 0\right\}. \tag{6}$$

Thus, interpreting Eq. (5) as a Riemann sum, and noting that $|S|/n$ converges to p almost surely, we have that the success guarantee of the optimal policy converges to

$$\int_p^1\int_\tau^1 \frac{1}{1-t}\left(\frac{1}{t-p}\left(\frac{p}{t}\right)^{\tilde{\ell}(\tau)} - \frac{1}{1-p}p^{\tilde{\ell}(\tau)}\right)\mathrm{d}t - \frac{1}{1-p}p^{\tilde{\ell}(\tau)}\,\mathrm{d}\tau\,.$$

From Eq. (6) it is clear that $\tilde{\ell}$ is non-decreasing, so we can define the sequence $t_i = \inf\left\{\tau \in [p,1] : \tilde{\ell}(\tau) \ge i\right\}$ and rewrite the limiting success guarantee in terms of it. Thus, we obtain

$$\sum_{i=0}^{\infty}\left(\int_{t_i}^{t_{i+1}}\int_\tau^1 \frac{1}{1-t}\left(\frac{1}{t-p}\left(\frac{p}{t}\right)^i - \frac{1}{1-p}p^i\right)\mathrm{d}t\,\mathrm{d}\tau - \frac{t_{i+1}^i - t_i^i}{1-p}\right).$$

If we rearrange the terms, turning the integral from t_i to t_{i+1} into the difference between the integral from t_i to 1 and the integral from t_{i+1} to 1, we obtain

$$\begin{aligned}
&\int_p^1\int_\tau^1 \frac{1}{(t-p)(1-p)}\,\mathrm{d}t\,\mathrm{d}\tau - \frac{p}{1-p}\\
&+\sum_{i=1}^{\infty}\left(\int_{t_i}^1\int_\tau^1 \frac{1}{1-t}\left(\frac{\left(\frac{p}{t}\right)^i - \left(\frac{p}{t}\right)^{i-1}}{t-p} - \frac{p^i-p^{i-1}}{1-p}\right)\mathrm{d}t\,\mathrm{d}\tau + \frac{t_i\left(p^i-p^{i-1}\right)}{1-p}\right)\\
=&\frac{1}{1-p} - \sum_{i=1}^{\infty}p^{i-1}\left(\int_{t_i}^1\int_\tau^1 \frac{1}{1-t}\left(\frac{t-p}{t^i(t-p)} - \frac{1-p}{1-p}\right)\mathrm{d}t\,\mathrm{d}\tau + t_i\frac{1-p}{1-p}\right)
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{1-p} - \sum_{i=1}^{\infty} p^{i-1} \left(\int_{t_i}^{1} \int_{\tau}^{1} \frac{1}{t^i(1-t)} \left(1-t^i\right) \,\mathrm{d}t \,\mathrm{d}\tau + t_i \right) \\
&= \frac{1}{1-p} - \sum_{i=1}^{\infty} p^{i-1} \left(\int_{t_i}^{1} \int_{\tau}^{1} \sum_{j=0}^{i-1} \frac{t^j}{t^i} \,\mathrm{d}t \,\mathrm{d}\tau + t_i \right) \\
&= \sum_{i=1}^{\infty} p^{i-1} \left(1 - t_i - \int_{t_i}^{1} \int_{\tau}^{1} \sum_{j=1}^{i} \frac{1}{t^j} \,\mathrm{d}t \,\mathrm{d}\tau \right) \\
&= \sum_{i=1}^{\infty} p^{i-1} \left(1 - t_i - \int_{t_i}^{1} \sum_{j=1}^{i} \frac{t-t_i}{t^j} \,\mathrm{d}t \right) .
\end{aligned}
$$

This concludes the proof, since we defined the t_i's in a way that they satisfy $t_i = \max\{p, t_i\}$. □

Proof of Lemma 18. We analyze separately the sum when $p = \max\{p, t_i'\}$ and when $t_i' = \{p, t_i'\}$. We call the first part V_1, which includes the terms up to $i = \lfloor \frac{c}{1-p} \rfloor$, and V_2 the rest.

$$
\begin{aligned}
V_1 &= \lim_{p\to 1} \sum_{i=1}^{\lfloor \frac{c}{1-p} \rfloor} p^{i-1} \left(1 - p - \int_{p}^{1} \sum_{j=1}^{i} \frac{t-p}{t^j} \,\mathrm{d}t \right) \\
&= \lim_{p\to 1} \sum_{i=1}^{\lfloor \frac{c}{1-p} \rfloor} p^{i-1} \left(1 - p - \int_{p}^{1} \,\mathrm{d}t + \int_{p}^{1} \frac{\mathrm{d}t}{t^i} - \int_{p}^{1} \sum_{j=1}^{i} \frac{1-p}{t^j} \,\mathrm{d}t \right) \\
&= \lim_{p\to 1} \sum_{i=1}^{\lfloor \frac{c}{1-p} \rfloor} p^{i-1} \left(\frac{p^{-(i-1)}-1}{i-1} - (1-p)\ln(1/p) - (1-p) \sum_{j=2}^{i} \frac{p^{-(j-1)}-1}{j-1} \right) \\
&= \lim_{p\to 1} \sum_{i=1}^{\lfloor \frac{c}{1-p} \rfloor} \frac{1-p^{i-1}}{i-1} - \lim_{p\to 1} \sum_{i=1}^{\lfloor \frac{c}{1-p} \rfloor} (p^{i-1}-p^i) \sum_{j=2}^{i} \frac{e^{-(j-1)\ln p}-1}{j-1} \\
&= \lim_{p\to 1} \sum_{i=1}^{\lfloor \frac{c}{1-p} \rfloor} \frac{1-(p^{\frac{1}{1-p}})^{(i-1)(1-p)}}{(i-1)(1-p)}(1-p) - \lim_{p\to 1} \sum_{i=1}^{\lfloor \frac{c}{1-p} \rfloor} (p^{i-1}-p^i) \sum_{j=2}^{i} \frac{e^{-\frac{(j-1)}{i} i \ln p}-1}{(j-1)/i} \cdot \frac{1}{i}
\end{aligned}
$$

Interpreting these two sums as Riemann sums, we obtain

$$
\begin{aligned}
V_1 &= \int_0^c \frac{1-e^{-x}}{x} \,\mathrm{d}x - \int_{e^{-c}}^{1} \int_0^1 \frac{e^{-x\ln y}-1}{x} \,\mathrm{d}x \,\mathrm{d}y \\
&= \int_0^c \frac{1-e^{-x}}{x} \,\mathrm{d}x - \int_{e^{-c}}^{1} \int_0^1 \frac{e^{-x\ln y}-1}{-x\ln y}(-\ln y) \,\mathrm{d}x \,\mathrm{d}y \\
&= \int_0^c \frac{1-e^{-x}}{x} \,\mathrm{d}x - \int_{e^{-c}}^{1} \int_0^{-\ln y} \frac{e^{x}-1}{x} \,\mathrm{d}x \,\mathrm{d}y \\
&= \int_0^c \frac{1-e^{-x}}{x} \,\mathrm{d}x - \int_0^c \int_{e^{-c}}^{e^{-x}} \frac{e^{x}-1}{x} \,\mathrm{d}y \,\mathrm{d}x \\
&= \int_0^c \frac{1-e^{-x}-(e^{-x}-e^{-c})(e^{x}-1)}{x} \,\mathrm{d}x \\
&= e^{-c} \int_0^c \frac{e^{x}-1}{x} \,\mathrm{d}x
\end{aligned}
$$

$$= e^{-c} \int_0^1 \frac{e^{cx}-1}{x}\,\mathrm{d}x$$
$$= e^{-c}\,,$$

where the last step comes from the definition of c. On the other hand, we have that

$$\begin{aligned}
V_2 &= \lim_{p\to 1} \sum_{i=\lfloor \frac{c}{1-p}\rfloor+1}^{\infty} p^{i-1}\left(\frac{c}{i} - \int_{1-\frac{c}{i}}^1 \sum_{j=1}^{i} \frac{t-1+c/i}{t^j}\,\mathrm{d}t\right)\\
&= \lim_{p\to 1} \sum_{i=\lfloor \frac{c}{1-p}\rfloor+1}^{\infty} p^{i-1}\left(\frac{c}{i} - \int_{1-c/i}^1 \mathrm{d}t + \int_{1-c/i}^1 \frac{1}{t^i}\,\mathrm{d}t - \int_{1-c/i}^1 \sum_{j=1}^{i} \frac{c/i}{t^j}\,\mathrm{d}t\right)\\
&= \lim_{p\to 1} \sum_{i=\lfloor \frac{c}{1-p}\rfloor+1}^{\infty} p^{i-1}\left(\frac{(1-c/i)^{-(i-1)}-1}{i-1} + \frac{c}{i}\ln(1-c/i) - \sum_{j=2}^{i} c\frac{(1-c/i)^{-(j-1)}-1}{i(j-1)}\right)\\
&= \lim_{p\to 1} \sum_{i=\lfloor \frac{c}{1-p}\rfloor+1}^{\infty} (p^{i-1}-p^i)\frac{(1-c/i)^{-(i-1)}-1}{\frac{1-p}{-\ln p}(i-1)(-\ln p)}\\
&\quad - \lim_{p\to 1} \sum_{i=\lfloor \frac{c}{1-p}\rfloor+1}^{\infty} \frac{p^{i-1}-p^i}{\frac{1-p}{-\ln p} i(-\ln p)} \sum_{j=2}^{i} \frac{c\left((1-c/i)^{-i\frac{j-1}{i}}-1\right)}{j/i}\cdot\frac{1}{i}\,,
\end{aligned}$$

where in the last equality we omitted a term that vanishes when p tends to 1. We again interpret the sums as Riemann sums.

$$\begin{aligned}
V_2 &= \int_0^{e^{-c}} \frac{e^c-1}{\ln(1/x)}\,\mathrm{d}x - c\int_0^{e^{-c}} \frac{1}{\ln(1/x)} \int_0^1 \frac{e^{cy}-1}{y}\,\mathrm{d}y\,\mathrm{d}x\\
&= (e^c-1-c)\int_0^{e^{-c}} \frac{1}{\ln(1/x)}\,\mathrm{d}x\\
&= (e^{-c}-1-c)\int_1^{\infty} x^{-1}e^{-cx}\,\mathrm{d}x\,.
\end{aligned}$$

In the second equality we used the definition of c and in the third one we performed a change of variables. Summing V_1 and V_2 we get Eq. (4). □

Proof of Lemma 19. We compute thresholds $\tilde{t}$ as follows. For the first $O\left(\frac{1}{\varepsilon\cdot(1-p)}\right)$ thresholds, we find approximate solutions to Eq. (2) using binary search in the interval $[1/e, 1]$. The left-hand side is monotone and continuous, and is at least 1 at $1/e$, and 0 at 1. With $O\left(\log\left(\frac{1}{\varepsilon(1-p)}\right)\right)$ iterations per threshold, we can get solutions within an error of $O(\varepsilon^2(1-p)^2)$ of the optimal thresholds. By taking the right-hand endpoint of the last interval in the binary search, we additionally ensure that $t_i^* \le \tilde{t}_i$ for each threshold we compute. We set all the other thresholds to be equal to 1. Evaluating Eq. (2) for a given $i \in \mathbb{N}$ takes time $O(i)$, so the previous computation takes in total $O\left(\frac{1}{\varepsilon^2\cdot(1-p)^2}\log\left(\frac{1}{\varepsilon\cdot(1-p)}\right)\right)$ steps.

We show now that by using $\tilde{t}$ we get a success guarantee of $\gamma(p) - O(\varepsilon)$. Notice first that since $t_i^* \le \tilde{t}_i$ for all $i \in \mathbb{N}$, $ALG_{\tilde{t}}$ never stops eariler than ALG_{t^*}. Now, consider a realization in which ALG_{t^*} succeeds in selecting the best element in $[p, 1]$. In the same realization, if $ALG_{\tilde{t}}$ does not succeed, then it must be the case that for some $i \in \mathbb{N}$, the best element in $[p, 1]$ arrived in $[t_i^*, \tilde{t}_i)$, so it was acceptable for ALG_{t^*}, but not for $ALG_{\tilde{t}}$. Since the arrival of the best element is uniform in $[p, 1]$, we can bound the probability that ALG_{t^*} succeeds and $ALG_{\tilde{t}}$ does not by the measure of $\cup_{i\in\mathbb{N}}[t_i^*, \tilde{t}_i)$, relative to $[p, 1]$. We know that $t_i^* = 1 - O(1/i)$, so the contribution of the thresholds we approximate with 1 is only $O(\varepsilon(1-p))$. For the

thresholds for which we solve the equation, the total error is also $O(\varepsilon(1-p))$. Therefore, relative to $[p,1]$, the measure of the set is only $O(\varepsilon)$.

□

3 Omitted proofs from Section 5

Proof of Theorem 4. Consider an instance of AOSp for a fixed unknown value of p where the player is faced with h samples. The proof follows from the next expression for the success probability and concentration bound. Note that for $\hat{p} = h/n$, we can write $k = \left\lfloor \frac{1}{1-\hat{p}} \right\rfloor = \left\lfloor \frac{n}{n-h} \right\rfloor$.

Lemma 26. *For a given sample set S with h values and an online set V with $n-h$ values, the k-max algorithm with $k = \left\lfloor \frac{n}{n-h} \right\rfloor$ chooses the maximum value of the online set with probability*

$$\mathbb{P}[\mathit{Success}] = \sum_{h=0}^{n} \left\lfloor \frac{n}{n-h} \right\rfloor \left(\frac{h}{n}\right)^{\left\lfloor \frac{n}{n-h} \right\rfloor} \frac{n-h}{n} \binom{n}{h} p^h (1-p)^{n-h} \,,$$

where p is the probability of independently sampling a value from the initial set.

Proof. Assume that the values of the adversarial input $\mathcal{A}$ are sorted in decreasing order $\alpha_1 > \alpha_2 > \ldots > \alpha_n$. Let us call p_h the probability that the k-max algorithm succeeds in a particular instance with h samples and S_h the event where $|S| = h$. Then the total probability that the k-max algorithm succeeds equals

$$\begin{aligned} \mathbb{P}[\text{Success}] &= \sum_{h=0}^{n} \mathbb{P}[k\text{-max algorithm wins} \mid S_h] \cdot \mathbb{P}[S_h] \\ &= \sum_{h=0}^{n} p_h \binom{n}{h} p^h (1-p)^{n-h} \,, \end{aligned}$$

since each value of the initial set is sampled independently with probability p. It remains to determine p_h. Conditioned on the fact that we end up with h samples, all the different labelings (as a sample or online value) of the initial n values are equally likely to happen. There are $\binom{n}{h}$ different labelings, and each α_i is labeled as a sample in an h/n-fraction of the possible labelings and as an online value in the rest.

Observe, as in the proof of Lemma 1, that the algorithm succeeds only if *exactly one* of the $\lfloor n/(n-h) \rfloor$ largest values of the adversarial input ends up in the online set and the $(\lfloor n/(n-h) \rfloor + 1)$-th largest ends up in the sample set. To compute the number of such labelings, first consider those such that $\alpha_1, \alpha_2, \ldots, \alpha_{\lfloor n/(n-h) \rfloor + 1}$ are all labeled as samples except for exactly one. From those, we can exclude the labelings that mark $\alpha_{\lfloor n/(n-h) \rfloor + 1}$ as an online value, since in this case the $\lfloor n/(n-h) \rfloor$-th largest sample is larger than all the online values. Therefore, we obtain

$$\begin{aligned} p_h &= \left(\left\lfloor \frac{n}{n-h} \right\rfloor + 1 \right) \left(\frac{h}{n}\right)^{\left\lfloor \frac{n}{n-h} \right\rfloor} \left(\frac{n-h}{n}\right) - \left(\frac{h}{n}\right)^{\left\lfloor \frac{n}{n-h} \right\rfloor} \left(\frac{n-h}{n}\right) \\ &= \left\lfloor \frac{n}{n-h} \right\rfloor \left(\frac{h}{n}\right)^{\left\lfloor \frac{n}{n-h} \right\rfloor} \left(\frac{n-h}{n}\right) , \end{aligned}$$

and the lemma follows. □

Lemma 27 (Hoeffding (1963))**.** *Let $X_1, X_2, \ldots, X_n$ be i.i.d. Bernoulli random variables with parameter p and let $\bar{X} = \left(\sum_{i=1}^{n} X_i\right)/n$. Then for any $\varepsilon > 0$,*

$$\mathbb{P}\left[\left|\bar{X} - pn\right| \geq \varepsilon\right] \leq 2e^{-2n\varepsilon^2} \,.$$

Alternatively, by setting $\delta = 2e^{-2n\varepsilon^2}$ we get that

$$\left|\bar{X} - pn\right| \leq \sqrt{\frac{1}{2n} \ln \frac{2}{\delta}} \qquad \textit{with probability at least } 1-\delta \,.$$

To finish the proof of Theorem 4, let ε_1 and ε_2 be such that

$$\varepsilon_1 \leq 1 - \frac{\left(\frac{h}{n}\right)^{\frac{n}{n-h}}}{p^{\frac{1}{1-p}}} \quad \text{and} \quad \varepsilon_2 \leq 2e^{-2n}\,.$$

Note that the first value is chosen such that

$$\left\lfloor \frac{n}{n-h} \right\rfloor \left(\frac{h}{n}\right)^{\left\lfloor \frac{n}{n-h} \right\rfloor} \frac{n-h}{n} \geq \left(\left\lfloor \frac{1}{1-p} \right\rfloor \right) p^{\left\lfloor \frac{1}{1-p} \right\rfloor} (1-p) \cdot (1-\varepsilon_1)\,,$$

while the second is chosen such that Lemma 27 yields $\mathbb{P}\left[\left|\bar{X} - pn\right| < 1\right] \geq 1 - 2e^{-2n} \geq 1 - \varepsilon_2$. Therefore, with probability at least $1 - \varepsilon_2$, we have

$$\sum_{h=pn-\varepsilon}^{pn+\varepsilon} \binom{n}{h} p^h (1-p)^{n-h} = \binom{n}{h} p^h (1-p)^{n-h} \Big|_{h=pn} \geq 1 - \varepsilon_2\,.$$

Therefore, we can bound the success guarantee given by Lemma 26 as follows.

$$\begin{aligned}
\mathbb{P}[\text{Success}] &= \sum_{h=0}^{n} \left\lfloor \frac{n}{n-h} \right\rfloor \left(\frac{h}{n}\right)^{\left\lfloor \frac{n}{n-h} \right\rfloor} \frac{n-h}{n} \binom{n}{h} p^h (1-p)^{n-h} \\
&\geq \left\lfloor \frac{1}{1-p} \right\rfloor p^{\left\lfloor \frac{1}{1-p} \right\rfloor} (1-p) \cdot (1-\varepsilon_1) \cdot \sum_{h=pn-\varepsilon}^{pn+\varepsilon} \binom{n}{h} p^h (1-p)^{n-h} \\
&\geq \left\lfloor \frac{1}{1-p} \right\rfloor p^{\left\lfloor \frac{1}{1-p} \right\rfloor} (1-p) \cdot (1-\varepsilon_1) \cdot (1-\varepsilon_2)\,.
\end{aligned}$$

For any given $\varepsilon > 0$, one can take ε_1 and ε_2 that adhere to the bounds above and such that $(1-\varepsilon_1)(1-\varepsilon_2) \leq (1-\varepsilon)$. This yields a success guarantee that is at least $1-\varepsilon$ times the success guarantee of the k-max algorithm for known p. □

Proof of Theorem 5. We prove that for any $\varepsilon > 0$, it is not possible to achieve a success guarantee of ε.

Consider the following new game for any $\delta > 0$. The adversary selects a size n and generates an instance of this size with increasing values. Then, the adversary again selects p appropriately, so that the probability that there is at least one sample is at most δ and the probability that there are no samples is at least $1-\delta$. Then the sampling process happens and the player faces the sequence. If at least one value is sampled, the player automatically wins, otherwise, she wins if and only if she selects the last non-sampled value.

Consider the case where there are no sampled values. Since the player does not learn anything along the game, any deterministic algorithm waits $t-1$ values before it selects the t-th value. A randomized algorithm can be thought of as a distribution over the stopping times t. Since the domain of t are all positive integers, it is not possible that this distribution has weight at least λ for every size, for any constant $\lambda > 0$. Therefore, on instances with stopping probability less than λ, the player only wins with probability at most λ. Such an instance occurs trivially with probability at most 1.

Overall, in this new game, the player wins in at most $\delta + \lambda$ values. Taking e.g. δ and λ slightly smaller than $\varepsilon/2$, the success guarantee of this game is less than ε.

The proof for AOSp with unknown p and n follows easily now. The adversary chooses values of n and p as above. In case there are no sampled values, both games are the same, since in both cases the player has the same information and the same available strategies. In case there is at least one sampled value, the player wins in the new game with probability 1 and in AOSp with probability strictly less than 1. Therefore, the success guarantee of AOSp is at most the success guarantee of the new game, which is less than ε. □

References

Hoeffding, W. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.