



Unit Interval Selection in Random Order Streams

Cezar-Mihail Alexandru 

Independent Researcher, Bristol, UK

Adithya Diddapur  

School of Computer Science, University of Bristol, UK

Magnús M. Halldórsson   

ICE-TCS & Department of Computer Science, Reykjavik University, Iceland

Christian Konrad   

School of Computer Science, University of Bristol, UK

Kheeran K. Naidu 

Qworky Research, London, UK

Abstract

We consider the Unit Interval Selection problem in the one-pass random order streaming model. In this setting, an algorithm is presented with a sequence of n unit-length intervals on the line that arrive in uniform random order, one at a time, and the objective is to output (an approximation of) a largest set of disjoint intervals using space linear in the size of an optimal solution. Previous work only considered adversarially ordered streams and established that, within these space constraints, a $(2/3)$ -approximation can be achieved in such streams, and this is best possible, in that going beyond such an approximation factor requires space $\Omega(n)$ [Emek et al., TALG'16].

In this work, we show that an improved expected approximation factor can be achieved if the input stream is in uniform random order, where the expectation is taken over the stream order. More specifically, we give a one-pass streaming algorithm with expected approximation factor 0.7401 that uses space $O(|OPT|)$, where OPT denotes an optimal solution. We also show that random order algorithms with expected approximation factor above $8/9$ require space $\Omega(n)$, and algorithms that compute a better than $2/3$ -approximation with probability above $2/3$ also require $\Omega(n)$ space.

On a technical level, we design an algorithm for the restricted domain $[0, \Delta)$, for some constant Δ , and use standard techniques to obtain an algorithm for unrestricted domains. For the restricted domain $[0, \Delta)$, we run $O(\Delta)$ recursive instances of our algorithm, with each instance targeting the situation where a specific interval of an optimal solution arrives first. We establish the interesting property of our algorithm that it performs worst when the input stream consists solely of a set of independent intervals. It then remains to analyse the algorithm on these simple instances.

Our lower bound is proved via communication complexity arguments, similar in spirit to the robust communication lower bounds established by [Chakrabarti et al., Theory Comput. 2016].

2012 ACM Subject Classification Theory of computation \rightarrow Streaming, sublinear and near linear time algorithms; Theory of computation \rightarrow Random order and robust communication complexity

Keywords and phrases Random order streaming algorithms, unit interval selection

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.4

Funding *Magnús M. Halldórsson*: Partially supported by Icelandic Research Fund grant 2511609.

1 Introduction

In the one-pass streaming model of computation, an algorithm is presented with a stream of input items that, a priori, arrive in arbitrary order. The algorithm is tasked with computing a solution while maintaining a memory of size sublinear in the length of the input stream.



© Cezar-Mihail Alexandru, Adithya Diddapur, Magnús M. Halldórsson, Christian Konrad, and Kheeran K. Naidu; licensed under Creative Commons License CC-BY 4.0

43rd International Symposium on Theoretical Aspects of Computer Science (STACS 2026).

Editors: Meena Mahajan, Florin Manea, Annabelle McIver, and Nguyễn Kim Thăng

Article No. 4; pp. 4:1–4:20



Leibniz International Proceedings in Informatics
LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



We consider the Unit Interval Selection problem in the streaming model. In this problem, the input stream consists of n closed unit-length intervals \mathcal{S} on the line, presented to the algorithm in arbitrary order, and the objective is to compute a pairwise non-overlapping subset $\mathcal{I} \subseteq \mathcal{S}$ of largest cardinality.

Emek et al. [12] initiated the study of Unit Interval Selection in the one-pass streaming model. They gave a $2/3$ -approximation algorithm that uses $O(|OPT|)$ words¹ of space, where OPT denotes an optimal solution, and showed that $\Omega(n)$ bits of space are necessary for algorithms with even slightly better approximation guarantees, which renders their algorithm space optimal. Subsequently, Cabello and Pérez-Lantero [7] gave a simpler algorithm with the same guarantee. We emphasize that these results hold for streams that are in adversarial/arbitrary order.

Given that the adversarial order setting is fully understood, in this paper, we ask whether improved algorithms are possible if the input stream is in uniform random order. Indeed, random order streams have received significant attention, and, for many problems, improvements beyond known lower bounds that hold for adversarial orderings are known, such as Quantile Estimation [13], Maximum Matching [15, 5, 2], Bounded Degree Property Testing [17], and edge-arrival Set Cover [16]. Lower bounds for random order streams were pioneered by Chakrabarti et al. [9], with subsequent work [8] further developing these ideas. The ideas from [8] have since been employed in various random order streaming lower bound proofs, including [2] and [3]. The random order setting, which assumes worst-case instances and random order arrivals, is a less pessimistic setting than the arbitrary order setting, which assumes both worst-case instances and worst-case orderings, and is therefore more relevant from a practitioner's point of view.

1.1 Our Results

As our main result, we show that we can go beyond the $2/3$ -approximation barrier for adversarial order streams when considering a uniform random arrival order.

► **Theorem 1.** *There is a deterministic one-pass $O(|OPT|)$ words of space streaming algorithm for Unit Interval Selection on random order streams with expected approximation factor 0.7401 , where the expectation is taken over the random order of the stream.*

We then complement our algorithm with the following limitation result:

► **Theorem 2.** *Let $\epsilon > 0$ be any small constant. Then:*

1. *Any randomized one-pass streaming algorithm for random-order Unit Interval Selection with expected approximation factor $8/9 + \epsilon$ must use $\Omega(n)$ bits of space.*
2. *For any $\delta > 0$, any randomized one-pass streaming algorithm for random-order Unit Interval Selection with approximation factor $2/3 + \delta$ that succeeds with probability at least $2/3 + \epsilon$ on any input instance must use $\Omega(n)$ bits of space.*

The expectation in (1) and the probability in (2) are taken over both the stream order and the random choices of the algorithm.

We note that, in the previous theorem, (1) immediately follows from (2), however, the result stated in (1) is more suitable for comparison to our algorithmic result. Furthermore, the result in (2) explains why our algorithm can only go beyond the $2/3$ -approximation barrier that holds for adversarial order streams *in expectation*, and not with high constant probability.

¹ A word is the amount of space required to store a single interval.

Our algorithmic result combined with our lower bound result show that the optimal achievable approximation ratio lies in the interval $[0.7401, 0.8]$, and we leave it as a great open question to further narrow this gap.

1.2 Techniques

We will now discuss the main ideas behind both our algorithm and our lower bound.

Algorithm. We make use of an established technique that, given a one-pass streaming algorithm for unit-length intervals that are all contained in the restricted domain $[0, \Delta)$, for some constant integer Δ , and has approximation factor α , produces an algorithm for unrestricted domains with only a constant factor blow-up in space and a factor $(\Delta - 1)/\Delta$ blow-up in the approximation factor [11]. We thus see that it is enough to design an algorithm for the restricted domain $[0, \Delta)$, for some large but constant Δ , large enough so as to minimize the impact of the factor $(\Delta - 1)/\Delta$ on the final approximation factor.

To this end, as a key building block of our algorithm, we employ the following strategy: We maintain the left-most interval I_L observed thus far in the stream, and feed the substream of intervals that lie to the right of I_L into a recursive instance of our algorithm. This building block then returns the interval I_L together with the solution produced by the recursive call. To see the strength of this method, denote by OPT an arbitrary but fixed optimal solution. Furthermore, assume that the left-most interval $opt_1 \in OPT$ arrives before any of the other intervals in OPT . In this case, the algorithm would have either picked opt_1 as the interval I_L or an interval that is even further left as opt_1 , and the substream fed into the recursive call of the algorithm contains $OPT \setminus opt_1$, i.e., an independent set that is only by one smaller. In other words, the algorithm has acted optimally thus far since one interval I_L was selected and a subinstance that contains an independent set of size $|OPT| - 1$ was created whose solution can be combined with the selected interval I_L .

Unfortunately, the event that opt_1 appears first among the intervals of OPT has a probability of only $1/|OPT|$ (over the random order of the stream). Our aim, however, is to make this approach work, even if any other interval of OPT is the first to arrive among the intervals in OPT . First, by symmetry, we see that the approach above also works when the right-most interval of OPT arrives first. Hence, suppose now that the i -th (counted from left to right) optimal interval $opt_i \in OPT$ arrives before any of the other optimal intervals, for some $1 < i < |OPT|$. Furthermore, let ℓ be the largest integer strictly smaller than the left boundary of opt_i , and let r be the smallest integer strictly larger than the right boundary of opt_i . Then, we construct two solutions and pick the larger one, as follows. For solution 1, we combine the outputs of two independent runs of our algorithm, one on the domain $[0, \ell)$, and one on the domain $[\ell, \Delta)$. While these runs ignore intervals that intersect with the point ℓ and may thus potentially miss a crucial optimal interval that intersects this point, we can now exploit the fact that the interval opt_i is the left-most optimal interval in the domain $[\ell, \Delta)$. Hence, as discussed above, the run on $[\ell, \Delta)$ does not make any mistake when selecting the interval opt_i as the leftmost interval (or an interval within $[\ell, \Delta)$ that lies even further to the left as opt_i) into the final solution. For solution 2, we combine the outputs of the two runs of the algorithm on the domains $[0, r)$, and $[r, \Delta)$, respectively, and now exploit the fact that opt_i is the right-most optimal interval in the domain $[0, r)$.

Last, since our algorithm cannot know which interval of OPT will indeed arrive first, we run the above strategy for every integer $1 \leq i \leq |OPT|$. We observe that this still yields an algorithm with only constant space complexity since the domain $[0, \Delta)$ is of constant size, which further implies that the optimal solution within this domain is also only of constant size $|OPT| \leq \Delta - 1$.

In our analysis, we establish the monotonicity property that adding intervals to the input stream never decreases the size of the output produced by the algorithm. In other words, the worst-case performance of our algorithm is achieved when the algorithm is run on an independent set – a special case that could of course be solved optimally by just reporting these intervals. We can thus focus on analysing the performance of the algorithm on this special case. Due to the many recursive calls of the algorithm, we obtain a recursive formula for the solution size obtained. We observe that the approximation factor α of our algorithm decreases as Δ increases. Recall, however, that we still need to apply the technique mentioned above to go from the restricted domain $[0, \Delta)$ to an unrestricted domain, which incurs a further loss in the approximation factor by a factor of $(\Delta - 1)/\Delta$. Using a computer programme, we see that, for $\Delta = 5000$, the final algorithm has an approximation factor of at least 0.7401, which is provably very close to the best possible choice. This specific choice of Δ is discussed in more detail in the full version of this paper.

Lower Bound. Our lower bound is obtained via a reduction to the INDEX_t problem in the one-way two-party communication complexity setting, by combining the clique gadgets already used in [12] and [7] for adversarial order lower bounds with the random order lower bound ideas by Cormode et al. [8].

In the INDEX_t problem, there are two players, denoted Alice and Bob. Alice holds a vector $X \in \{0, 1\}^t$ consisting of t uniform independent random bits, and Bob is given a uniform random index $A \in [t]$. Alice sends a message to Bob and Bob outputs the bit $X[A]$. The objective is to obtain a communication protocol that communicates a message of smallest possible size. It is known that INDEX_t requires a message of size $\Omega(t)$. Since streaming algorithms can be used as one-way two-party communication protocols (Alice runs the streaming algorithm on her part of the input and sends the memory state to Bob, who then completes the execution of the algorithm on his part), lower bounds on the message size translate and yield lower bounds on the space used by streaming algorithms.

The INDEX_t problem can be reduced to Unit Interval Selection as follows. Using the bits in $X \in \{0, 1\}^t$, Alice constructs a *clique* of t mutually overlapping intervals, i.e., a stack of intervals such that, for every $i > 1$, the i -th interval is slightly further to the right than the $(i - 1)$ -th interval. The bits $X[i]$ are encoded in this construction in that, depending on the value of $X[i]$, the i -th interval is ever so slightly moved, which, however, does not affect the logic behind the construction. Crucially, an algorithm that outputs the j -th interval of the clique stack, for any j , must know the value $X[j]$. Then, based on the index A , Bob constructs two *wing intervals*, which surround, but are independent of, the interval corresponding to $X[A]$. The key property is that every clique interval, other than the one corresponding to $X[A]$, must intersect exactly one of these wing intervals, or, in other words, the only independent set of size 3 consists of the wing intervals together with the A -th interval of the stack. This in turn implies that any algorithm on this construction that computes a better than $2/3$ -approximation must report the interval corresponding to $X[A]$, and its location allows recovering the bit $X[A]$.

When bringing this construction into the random-order setting, for the resulting construction to be hard it is necessary that the two wing intervals appear after the A -th clique interval since otherwise the positions of the wing intervals reveal the index A and an algorithm could then simply collect the A -th clique interval when it arrives, obtain an optimal solution to the interval selection problem, and solve the underlying INDEX_t instance. We, however, observe that, under a uniform random arrival order, with probability $1/3$, the two wing intervals indeed arrive after the A -th clique interval, and we prove that this case indeed remains hard

to solve. Our expected approximation factor lower bound of $8/9$ is then explained by the fact that, with probability $1/3$, an algorithm can only obtain a solution of size 2, and with probability $2/3$, an algorithm can obtain a solution of optimal size 3, which results in an expected solution size $1/3 \cdot 2 + 2/3 \cdot 3 = 8/3$, or an approximation factor of $(8/3)/3 = 8/9$.

On a technical level, our approach is a reduction via the use of shared public randomness in the spirit of the argument from [8]. Alice and Bob sample a shared random permutation of the intervals that are used in the hard instance. Alice constructs each clique interval which arrives before a wing interval via the bits in X , and then Bob constructs the wing intervals via A , and also the remaining clique intervals via sampling public random variables. This results in a construction that reflects the bit $X[A]$ of the input to INDEX_t precisely when the interval corresponding to $X[A]$ arrives before either wing interval, and this occurs with probability $1/3$, as desired, which yields our lower bound results.

1.3 Further Related Work

Besides unit-length intervals, intervals of arbitrary lengths have also been considered. Emek et al. [12] showed that, in adversarial order streams, a $1/2$ -approximation for arbitrary-length intervals can be achieved with space $O(|OPT|)$, and they also proved that algorithms with improved approximation factor must use space $\Omega(n)$. Cabello and Pérez-Lantero [7] gave a simpler algorithm for this case. More recently, Dark et al. [11] explored algorithms for weighted interval streams and streams that allow for deletions. In a different line of work, the problem of estimating the size of a largest disjoint set of intervals has also been considered [7, 4]. Besides intervals, other geometric objects, such as squares and rectangles [10], and segments and 2-intervals [6], have also been explored.

Interval Selection has also been studied in the streaming sliding window model, where the objective is to maintain a solution on the L most recent intervals. Alexandru and Konrad [1] showed that **Interval Selection** is harder in the sliding window model than in the one-pass streaming setting in that, for both unit-length intervals and arbitrary-length intervals, algorithms cannot achieve the same approximation guarantees as in the streaming setting when using space $O(|OPT|)$.

1.4 Outline

We provide definitions and state the hardness of the INDEX_t communication problem that we use for our lower bound result in our preliminaries section, Section 2. Then, in Section 3, we present and analyse our algorithm, and in Section 4, we give our lower bound result. Finally, we conclude in Section 5.

2 Preliminaries

We use n to denote the length of the input stream and $[k]$ to denote the set $\{0, \dots, k-1\}$.

2.1 Interval Selection

In this work, we consider closed intervals $I = [a, b]$, where $a < b$. We mostly focus on intervals of *unit length*, i.e., when $b = a + 1$. We say that a is the *left endpoint* of I and b is the *right endpoint* of I . We say that an interval $I = [a_I, b_I]$ is *further left* than an interval $J = [a_J, b_J]$ if its left endpoint is strictly smaller, i.e., $a_I < a_J$ holds. Similarly, I is *further right* of J if its right endpoint is strictly larger, i.e., $b_I > b_J$ holds.

4:6 Unit Interval Selection in Random Order Streams

Two intervals I, J are *independent* if $I \cap J = \emptyset$, and a set of intervals \mathcal{I} is an *independent set* if the intervals contained in \mathcal{I} are pairwise independent. The Unit Interval Selection problem asks for an independent set of intervals of largest size, where we also assume that all intervals have length 1.

We use OPT to denote an arbitrary but fixed independent set of largest size, and we write $\alpha = \alpha(\cdot)$ to denote the size of a largest independent set, where the parameter passed on to $\alpha(\cdot)$ is either a stream or a set of intervals. For a parameter $0 < \beta \leq 1$, we say that an independent set \mathcal{I} is a β -*approximation* if $|\mathcal{I}| \geq \beta \cdot |OPT|$.

We also assume that any interval can be stored in a single word of space. Under the plausible assumption that the endpoint can be encoded in $O(\log n)$ bits of space, this yields $O(\log n)$ space overhead.

2.2 Communication Complexity

We use the standard one-way two-party communication complexity setting, first introduced by [19]. Our lower bounds will then be derived from a reduction to the well known $INDEX_t$ problem. We refer the reader to [18] for a comprehensive overview of communication complexity.

In this setting, we have two parties that we denote by Alice and Bob. They each hold a portion of the input, X_A and X_B , respectively, and their goal is to compute/approximate a function $f(X_A, X_B)$ on their inputs. To this end, Alice sends a single message to Bob, $M(X_A)$, and after receiving the message, Bob outputs $f(X_A, X_B)$ or a suitable approximation thereof. Then, the size of Alice's message is the relevant notion of complexity, and the goal is to send a message that is as small as possible. Alice and Bob can make use of infinite sequences of both private and public/shared random bits.

► **Definition 3.** For an integer t , $INDEX_t$ is the one-way two-party communication game where Alice holds a vector $X \in \{0, 1\}^t$ and Bob holds an index $A \in [t]$. Alice sends a single message to Bob who then outputs the value $X[A]$.

Let μ denote the uniform distribution over $INDEX_t$ instances. This is the distribution where X is sampled uniformly at random from $\{0, 1\}^t$ and A is sampled independently from X and uniformly at random from its support $[t]$.

It is well-known that solving $INDEX_t$ requires a message of size $\Omega(t)$, even in a distributional sense when the input is chosen from μ , and also regardless of any public randomness used.

► **Theorem 4** ($INDEX_t$ Hardness (e.g., [18])). Let $\delta > 0$ be any constant. Let P be a one-way communication protocol for $INDEX_t$. Let R denote any public randomness used by P . If

$$\mathbb{P}_{\mu, R}(P \text{ outputs the correct answer}) \geq 1/2 + \delta,$$

then P must send a message of size $\Omega(t)$ bits.

3 Random-Order Unit Interval Selection Algorithm

We now prove Theorem 1. To this end, we first present our algorithm \mathcal{A} (see Algorithm 1), and then analyse it.

Our algorithm proceeds as follows. It is parametrised by two integers a, b with $a < b$, and operates under the assumption that all intervals fed into the algorithm are fully contained in the domain $\mathcal{D} = [a, b)$. Our overall goal is to design an algorithm that operates on the domain $[0, \Delta)$, for some integer Δ . To accommodate for this, we will see later that, for technical reasons, we need to invoke our algorithm with parameters $a = -1$ and $b = \Delta + 1$.

Algorithm 1 Algorithm \mathcal{A} .

Input: A domain $\mathcal{D} = [a, b)$, where $a, b \in \mathbb{Z}$, and $a < b$

Initialisation:

```

1: // the recursive instances corresponding to each split point
2: for each integer  $i \in \{a + 1, \dots, b - 1\}$  do
3:   // Initialise the left and right most intervals closest to each integer position
4:    $L_i \leftarrow \emptyset$ 
5:    $R_i \leftarrow \emptyset$ 
6:   // Initialise the recursive instances to the left of each integer position
7:    $\mathcal{A}_i^L \leftarrow$  new instance  $\mathcal{A}([a, i))$ 
8:    $\mathcal{T}_i^L \leftarrow$  new instance  $\mathcal{A}([a, i))$ 
9:   // Initialise the recursive instances to the right of each integer position
10:   $\mathcal{A}_i^R \leftarrow$  new instance  $\mathcal{A}([i, b))$ 
11:   $\mathcal{T}_i^R \leftarrow$  new instance  $\mathcal{A}([i, b))$ 

```

During the Stream:

Input: A stream \mathcal{S} of unit-length intervals, each of which is fully contained in $[a, b)$.

```

1: for each interval  $I \in \mathcal{S}$  do
2:   for each integer  $i \in \{a + 1, \dots, b - 1\}$  do
3:     // Update  $R_i$  and feed  $I$  into the corresponding right recursive instances
4:     if  $I \subseteq [i, b)$  then
5:       Feed  $I$  into  $\mathcal{T}_i^R$ 
6:       if  $R_i = \emptyset$  or ( $R_i \neq \emptyset$  and  $I$  is further left than  $R_i$ ) then
7:          $R_i \leftarrow I$ 
8:       if  $R_i \neq \emptyset$ ,  $I$  is independent of  $R_i$ , and  $I$  is further right than  $R_i$  then
9:         Feed  $I$  into  $\mathcal{A}_i^R$ 
10:    // Update  $L_i$  and feed  $I$  into the corresponding left recursive instances
11:    if  $I \subseteq [a, i)$  then
12:      Feed  $I$  into  $\mathcal{T}_i^L$ 
13:      if  $L_i = \emptyset$  or ( $L_i \neq \emptyset$  and  $I$  is further right than  $L_i$ ) then
14:         $L_i \leftarrow I$ 
15:      if  $L_i \neq \emptyset$ ,  $I$  is independent of  $L_i$ , and  $I$  is further left than  $R_i$  then
16:        Feed  $I$  into  $\mathcal{A}_i^L$ 

```

Output:

1: The largest set among

$$OUT(\mathcal{T}_i^L) \cup R_i \cup OUT(\mathcal{A}_i^R) \quad \text{and} \quad OUT(\mathcal{A}_i^L) \cup L_i \cup OUT(\mathcal{T}_i^R)$$

for each integer $i \in \{a + 1, \dots, b - 1\}$

Next, our algorithm considers all integers in $\{a + 1, \dots, b - 1\}$ as *split points*. For each such split point $i \in \{a + 1, \dots, b - 1\}$, the algorithm maintains in variables L_i and R_i the closest interval to i that is located left of i and the closest interval to i that is located right of i , respectively. It also executes four recursive calls of the algorithm associated with i . The recursive calls \mathcal{T}_i^L and \mathcal{T}_i^R are instantiated on the domains $[a, i)$ and $[i, b)$, respectively, and all intervals that fall into these domains are fed into these algorithms. The recursive calls \mathcal{A}_i^L and \mathcal{A}_i^R are also instantiated on the domains $[a, i)$ and $[i, b)$, however, only intervals that are left of L_i (and within the domain $[a, i)$) are fed into \mathcal{A}_i^L , and intervals that right of R_i (and within the domain $[i, b)$) are fed into \mathcal{A}_i^R .

For each split point i , two potential outputs are considered: The output generated by \mathcal{T}_i^L , combined with the output of \mathcal{A}_i^R as well as the interval R_i , and the output generated by \mathcal{A}_i^L , combined with the output of \mathcal{T}_i^R as well as the interval L_i . We note that, by construction, both these potential outputs form independent sets. The output of the algorithm then is the largest output of this kind when considering all split points.

In the following, we write $OUT(\mathcal{A}(\mathcal{S}))$ to denote the output produced by algorithm \mathcal{A} when run on the input stream \mathcal{S} .

3.1 Monotonicity Property

We first prove that removing any interval from the input stream cannot increase the size of the output regardless of the order of the stream.

► **Lemma 5.** *Let \mathcal{A} be an instance of Algorithm 1 instantiated with integers $a < b$. Let \mathcal{S} be any input stream (not necessarily random-order) whose intervals are fully contained in $[a, b)$, and let \mathcal{S}' be any substream of \mathcal{S} . Then $|OUT(\mathcal{A}(\mathcal{S}'))| \leq |OUT(\mathcal{A}(\mathcal{S}))|$.*

Proof. Our proof is via induction over the quantity $b - a$. We first show that the recursion tree will always have finite size.

▷ **Claim 6.** For any value of $b - a > 0$, the recursion tree of \mathcal{A} has finite size.

Proof. Each recursive subproblem is instantiated before any intervals arrive. Thus, the size of the recursion tree depends only on $b - a$ and not on the input stream. Each recursive instance has a domain that is by one smaller than the domain of its parent, and an instance with domain size at most 1 creates no subproblems. ◁

We now proceed with the induction.

Base Case. Suppose \mathcal{A} is an instance of Algorithm 1 such that $b - a = 1$. Then $|OUT(\mathcal{A}(\mathcal{S}'))| \leq |OUT(\mathcal{A}(\mathcal{S}))|$.

Proof. If $b - a = 1$, then no unit length intervals fit within $[a, b)$. Therefore, \mathcal{S} and \mathcal{S}' must both be empty streams, and so $|OUT(\mathcal{A}(\mathcal{S}'))| = |OUT(\mathcal{A}(\mathcal{S}))| = 0$. ◀

Inductive Hypothesis. Suppose the lemma holds for all instances of \mathcal{A} such that $b - a \leq k - 1$.

We now consider when $b - a = k$.

Inductive Step. By construction, the output of \mathcal{A} is formed by combining the outputs of its recursive children, together with either L_i or R_i , for some i . We will first argue that if any L_i or R_i stores an interval under \mathcal{S}' , then it must also store an interval under \mathcal{S} . We will then argue that all of the recursive children must output solutions at least as large under \mathcal{S} compared to \mathcal{S}' . This is sufficient to prove the claim.

First, W.L.O.G., suppose some L_i is non-empty under \mathcal{S}' . Then, there must exist some interval in \mathcal{S}' which is contained in the domain $[a, i)$. Then, this same interval must also be contained in \mathcal{S} (\mathcal{S} is a superstream of \mathcal{S}'), and so L_i must also be non-empty under \mathcal{S} . This argument holds for all relevant integers i , and also for each R_i .

We now turn to the substreams fed into the recursive children of \mathcal{A} , and begin with the instances \mathcal{T}_i^L and \mathcal{T}_i^R . W.L.O.G. consider \mathcal{T}_i^L for some relevant integer i . Then, by construction, the size of the domain of \mathcal{T}_i^L is at most $k - 1$. Next, let \mathcal{S}'_i denote the substream of intervals fed into \mathcal{T}_i^L when \mathcal{S}' is fed into \mathcal{A} , and \mathcal{S}_i the substream fed in when \mathcal{S} is fed into \mathcal{A} . Then, \mathcal{S}'_i is precisely the substream of intervals in \mathcal{S}' that are fully contained in $[a, i)$. Then, every interval in \mathcal{S}'_i is also contained in \mathcal{S}' and so \mathcal{S} , and therefore every interval in \mathcal{S}'_i is also contained in \mathcal{S}_i . This gives us that \mathcal{S}'_i is a substream of \mathcal{S}_i . We can now apply the inductive hypothesis to obtain

$$|\text{OUT}(\mathcal{T}_i^L(\mathcal{S}'_i))| \leq |\text{OUT}(\mathcal{T}_i^L(\mathcal{S}_i))|.$$

The same argument applies for every relevant integer i , and also for each \mathcal{T}_i^R .

Finally, we now consider the instances \mathcal{A}_i^L and \mathcal{A}_i^R . W.L.O.G. consider \mathcal{A}_i^L for some relevant integer i . Then, by construction, the size of the domain of \mathcal{A}_i^L is at most $k - 1$. Next, let \mathcal{S}'_i denote the substream of intervals fed into \mathcal{A}_i^L when \mathcal{S}' is fed into \mathcal{A} , and \mathcal{S}_i the substream fed in when \mathcal{S} is fed into \mathcal{A} . Then, these are each the substreams of \mathcal{S}' and \mathcal{S} of the intervals fully contained in $[a, i)$, which are also further left and independent of L_i at the time each interval arrives. We now observe that whenever an interval in \mathcal{S}'_i arrives in \mathcal{S} , the interval stored in L_i must be at least as far right as when the same interval arrives in \mathcal{S}' - by the construction of the algorithm. Therefore, every interval in \mathcal{S}'_i is also contained in \mathcal{S}_i , giving us that \mathcal{S}'_i is a substream of \mathcal{S}_i . We can now apply the inductive hypothesis to obtain

$$|\text{OUT}(\mathcal{A}_i^L(\mathcal{S}'_i))| \leq |\text{OUT}(\mathcal{A}_i^L(\mathcal{S}_i))|.$$

The same argument applies for every relevant integer i , and also for each \mathcal{A}_i^R .

This concludes the proof of the Lemma. ◀

3.2 Bounding the Expected Approximation Factor

We now derive a bound on the expected approximation factor achieved by Algorithm 1. To this end, let Δ be an arbitrary but fixed integer, and we consider the performance of our algorithm when all intervals lie within the domain $[0, \Delta)$. For technical reasons, we run our algorithm \mathcal{A} on the domain $[-1, \Delta + 1)$ and analyse the performance of this instantiation of the algorithm.

For a set of intervals \mathcal{I} that constitutes the input instance, we denote by $\Pi(\mathcal{I})$ the set of permutations of \mathcal{I} . Observe that the input stream then constitutes a uniform random element of $\Pi(\mathcal{I})$.

We are interested in the expected solution size produced by the algorithm as a function of $\alpha(\mathcal{I})$, i.e., the optimal solution size. For this purpose, for any integer $x \geq 0$, we define the following quantity:

$$\text{out}(x) = \min_{\text{instance } \mathcal{I} \text{ with } \alpha(\mathcal{I}) = x} \mathbb{E}_{S \sim \Pi(\mathcal{I})} |\text{OUT}(\mathcal{A}(S))|, \quad (1)$$

i.e., the worst-case expected performance of the algorithm on instances with optimal solution size x . For $x < 0$, define $\text{opt}(x) = 0$.

We first observe that it suffices to analyse the performance of our algorithm on streams consisting solely of independent intervals.

4:10 Unit Interval Selection in Random Order Streams

► **Observation 7.** Let \mathcal{I} be any input instance, and let $OPT \subseteq \mathcal{I}$ denote the optimal intervals in \mathcal{I} . Then,

$$\mathbb{E}_{S_{OPT} \sim \Pi(OPT)} |OUT(\mathcal{A}(S_{OPT}))| \leq \mathbb{E}_{S_{\mathcal{I}} \sim \Pi(\mathcal{I})} |OUT(\mathcal{A}(S_{\mathcal{I}}))| .$$

Proof. For a permutation $S_{\mathcal{I}}$ of the intervals \mathcal{I} , we denote by $OPT(S_{\mathcal{I}})$ the substream of optimal intervals. Then, since $OPT(S_{\mathcal{I}})$ is a substream of $S_{\mathcal{I}}$, we can use Lemma 5 to obtain:

$$\mathbb{E}_{S_{\mathcal{I}} \sim \Pi(\mathcal{I})} |OUT(\mathcal{A}(OPT(S_{\mathcal{I}})))| \leq \mathbb{E}_{S_{\mathcal{I}} \sim \Pi(\mathcal{I})} |OUT(\mathcal{A}(S_{\mathcal{I}}))| .$$

The result then follows from the observation that S_{OPT} and $OPT(S_{\mathcal{I}})$ are identically distributed. ◀

We will thus assume from now on that the input instance \mathcal{I} is a collection of disjoint intervals. We then take the input stream to be $\mathcal{S} \sim \Pi(I)$ for the remainder of this subsection, and omit writing that \mathcal{S} is the input to \mathcal{A} .

In the following, we will establish lower bounds on $out(x)$, for every $x \geq 0$. To this purpose, we treat the cases $x \in \{0, 1, 2\}$ explicitly, and then provide a recurrence relation for larger values of x .

► **Lemma 8.** For each $x \in \{0, 1, 2\}$, $out(x) = x$.

Proof. First, $\alpha = 0$ implies the input stream has no intervals, which is trivial. Next, if $\alpha = 1$, then the input stream consists of a single interval. This interval will be picked up by R_1 since R_1 constitutes the left-most interval in the domain $[0, \Delta)$.

Now suppose that $\alpha = 2$, which implies that the input consists of two intervals opt_1 and opt_2 with opt_1 further left of opt_2 . If opt_1 arrives before opt_2 , then R_1 will store opt_1 , and opt_2 will be fed into \mathcal{A}_1^R and returned by this recursive call. The interval opt_2 will then be stored as the right most interval fed into \mathcal{A}_1^R , ensuring that $|OUT(\mathcal{A}_1^R)| \geq 1$. We thus obtain that $|R_0 \cup OUT(\mathcal{A}_0^R)| = 2$. Then, by the construction of the algorithm, we get

$$|OUT(\mathcal{A})| \geq |\{R_1\} \cup OUT(\mathcal{A}_1^R)| \geq 1 + 1 = 2.$$

Similarly, if I_2 arrives first then $L_{\Delta+1}$ and the output of $\mathcal{A}_{\Delta+1}^L$ constitutes a candidate solution of size 2. ◀

Next, we consider the case $\alpha \geq 3$. We write

$$OPT = \{opt_1, \dots, opt_\alpha\},$$

to denote the optimal intervals ordered from left to right, and consider the cases for which interval from OPT arrives first. We then fix some $i \in \{1, \dots, \alpha\}$, and suppose that opt_i arrives in the input stream before any other interval in OPT . Let $opt_i = [f, f + 1]$, for some real f , and let

$$r = \lfloor f + 2 \rfloor, \text{ and } l = \lfloor f \rfloor$$

denote the smallest integer larger than $f + 1$ that does not intersect with opt_i , and the largest integer smaller than or equal to f which may only intersect with opt_i 's left endpoint, respectively. Note that when f is an integer, we have $l = f$, but $r = f + 2$. This choice is due to the fact that every recursive instantiation of our algorithm is run on a domain with a closed left endpoint and an open right endpoint.

► **Lemma 9.** *Conditioned on $opt_i = [f, f + 1]$ arriving first, both of the following hold:*

1. $\mathbb{E} \left[|R_l \cup OUT(\mathcal{A}_l^R)| \mid opt_i \text{ arrives first} \right] = 1 + out(\alpha - i)$, and
2. $\mathbb{E} \left[|L_r \cup OUT(\mathcal{A}_r^L)| \mid opt_i \text{ arrives first} \right] = 1 + out(i - 1)$.

Proof. We only prove 1) – the proof of 2) is similar and omitted.

By definition, the interval opt_i is further right and independent of l , and so will be stored in R_l . Next, observe that each of the intervals $opt_{i+1}, \dots, opt_\alpha$ are fed into \mathcal{A}_l^R in uniform random-order. Then, by definition of $out(\cdot)$, we obtain $\mathbb{E} |OUT(\mathcal{A}_l^R)| \geq out(\alpha - i)$. Finally, combining R_l and \mathcal{A}_l^R via linearity of expectation yields the desired bound.

► **Remark.** We note that each interval being contained in the domain $[0, \Delta)$ implies that $0 \leq l$ and also $r \leq \Delta$. Recall that we run algorithm \mathcal{A} on the domain $[-1, \Delta + 1)$, which implies that both the instances \mathcal{A}_l^R and \mathcal{A}_r^L are guaranteed to exist. ◀

The next lemma follows by a similar proof.

► **Lemma 10.** *Conditioned on $opt_i = [f, f + 1]$ arriving first, both of the following hold:*

1. $\mathbb{E} \left[|OUT(\mathcal{T}_r^R)| \mid opt_i \text{ arrives first} \right] \geq out(\alpha - i - 1)$, and
2. $\mathbb{E} \left[|OUT(\mathcal{T}_l^L)| \mid opt_i \text{ arrives first} \right] \geq out(i - 2)$.

Proof. We only prove 2) – the proof of 1) is similar and omitted.

By construction, the substream \mathcal{S}' fed into \mathcal{T}_l^L consists of the intervals in $\{opt_1, \dots, opt_{i-1}\}$ that do not overlap with the point l . Then, since each interval is of unit length, and also $f - l \leq 1$ holds, \mathcal{S}' contains either the $i - 1$ or $i - 2$ left-most intervals of OPT , and these are fed into \mathcal{T}_l^L in a uniform random order. We thus obtain $\mathbb{E} |OUT(\mathcal{T}_l^L)| \geq \min\{out(i - 1), out(i - 2)\}$. Finally, by Lemma 5, $out(\cdot)$ is a non-decreasing function, which implies that $out(i - 1) \geq out(i - 2)$ holds, and completes the proof. ◀

We now combine these lemmas to form a recurrence for $out(\cdot)$. In the following, for convenience, we write the maximum function over multiple lines.

► **Lemma 11.** *Conditioned on opt_i arriving first, the following holds:*

$$\mathbb{E} [|OUT(\mathcal{A})| \mid opt_i \text{ arrives first}] \geq 1 + \max \left\{ \begin{array}{l} out(i - 1) + out(\alpha - i - 1) \\ out(\alpha - i) + out(i - 2) \end{array} \right\}.$$

Proof. The proof follows almost immediately from Lemmas 9 and 10:

$$\begin{aligned} \mathbb{E} |OUT(\mathcal{A})| \mid opt_i \text{ arrives first} &\geq \mathbb{E} \left[\max \left\{ \begin{array}{l} |OUT(\mathcal{A}_r^L) \cup L_r \cup OUT(\mathcal{T}_r^R)| \\ |OUT(\mathcal{T}_l^L) \cup R_l \cup OUT(\mathcal{A}_l^R)| \end{array} \right\} \right] \\ &\hspace{10em} \text{(Construction of the algorithm.)} \\ &\geq \max \left\{ \begin{array}{l} \mathbb{E} |OUT(\mathcal{A}_r^L) \cup L_r \cup OUT(\mathcal{T}_r^R)| \\ \mathbb{E} |OUT(\mathcal{T}_l^L) \cup R_l \cup OUT(\mathcal{A}_l^R)| \end{array} \right\} \\ &\hspace{10em} \text{(Jensen's inequality.)} \\ &\geq \max \left\{ \begin{array}{l} 1 + out(i - 1) + out(\alpha - i - 1) \\ 1 + out(\alpha - i) + out(i - 2) \end{array} \right\}. \\ &\hspace{10em} \text{(Lemmas 9 and 10.)} \\ &\geq 1 + \max \left\{ \begin{array}{l} out(i - 1) + out(\alpha - i - 1) \\ out(\alpha - i) + out(i - 2) \end{array} \right\} \quad \blacktriangleleft \end{aligned}$$

In the following lemma, we obtain a bound on the expected approximation factor of our algorithm.

4:12 Unit Interval Selection in Random Order Streams

► **Lemma 12.** *The expected approximation factor achieved by \mathcal{A} on intervals that lie in the domain $[0, \Delta)$ is bounded from below by*

$$\begin{aligned} & \mathbb{E}[\text{approximation factor}] \\ & \geq \min_{\alpha \in \{1, \dots, \Delta-1\}} \left\{ \frac{1}{\alpha} + \frac{1}{\alpha^2} \cdot \sum_{i=1}^{\alpha} \max \left\{ \begin{array}{l} \text{out}(i-1) + \text{out}(\alpha-i-1) \\ \text{out}(\alpha-i) + \text{out}(i-2) \end{array} \right\} \right\}, \end{aligned}$$

where $\text{out}(2) = 2$, $\text{out}(1) = 1$, and $\text{out}(y) = 0$ for all $y \leq 0$.

Proof. Let $\alpha \in \{1, \dots, \Delta-1\}$ and consider an input instance \mathcal{I} that establishes the minimum in the definition of $\text{out}(\alpha)$. We consider a run of \mathcal{A} on \mathcal{I} . Then:

$$\begin{aligned} \text{out}(\alpha) &= \mathbb{E}[|\text{OUT}(\mathcal{A})|] \\ &= \sum_{i=1}^{\alpha} \mathbb{P}(\text{opt}_i \text{ arrives first}) \cdot \mathbb{E}[|\text{OUT}(\mathcal{A})| \mid \text{opt}_i \text{ arrives first}] && \text{(Law of total expectation.)} \\ &= \sum_{i=1}^{\alpha} \frac{1}{\alpha} \cdot \mathbb{E}[|\text{OUT}(\mathcal{A})| \mid \text{opt}_i \text{ arrives first}] && \text{(Uniform random arrival order.)} \\ &\geq \frac{1}{\alpha} \cdot \sum_{i=1}^{\alpha} \left[1 + \max \left\{ \begin{array}{l} \text{out}(i-1) + \text{out}(\alpha-i-1) \\ \text{out}(\alpha-i) + \text{out}(i-2) \end{array} \right\} \right] && \text{(By Lemma 11.)} \\ &= 1 + \frac{1}{\alpha} \cdot \sum_{i=1}^{\alpha} \max \left\{ \begin{array}{l} \text{out}(i-1) + \text{out}(\alpha-i-1) \\ \text{out}(\alpha-i) + \text{out}(i-2) \end{array} \right\} \end{aligned}$$

In particular, this gives us a recurrence relation for $\text{out}(\cdot)$, which we will later use to compute values numerically. The expected approximation factor is then simply this quantity multiplied by $1/\alpha$.

Lastly, when our algorithm is run on intervals in the domain $[0, \Delta)$, then we only know that $\alpha \in \{1, \dots, \Delta-1\}$. Hence, to bound the approximation factor of our algorithm on such instances, we consider all possible values of $\alpha \in \{1, \dots, \Delta-1\}$ and pick the minimum approximation factor for these cases, which establishes the lemma. ◀

3.3 Bounding the Space Used

In this section, we establish a bound on the space used by our algorithm.

► **Lemma 13.** *Let \mathcal{A} be an instance of Algorithm 1 run on domain boundaries $a < b$ with $b - a = k$. Then, \mathcal{A} uses $O(4^k k^k)$ words of space.*

Proof. By construction, \mathcal{A} consists of the following:

- $4k - 4$ recursive instances of the algorithm, each on input domains of length at most $k - 1$,
- at most $2k - 2$ many stored intervals (in each L_i and R_i).

Then, letting $S(k)$ denote the space used by an instance of Algorithm 1 on a domain of length k , we obtain the recurrence

$$S(k) \leq (4k - 4) \cdot S(k - 1) + 2k - 2,$$

which implies $S(k) = O(4^k k^k)$. ◀

3.4 Going to Unrestricted Domains

We now use the shifting window idea, originally by Hochbaum and Mass [14], and later applied by [7] and [11], that allows going from restricted to unrestricted domains.

Our lemma below is a straightforward reformulation to random-order streams of a corresponding lemma given in [11] for adversarial order streams. We note that the proofs are quite similar.

► **Lemma 14.** *Let $\Delta \geq 2$ be an integer. Let \mathcal{A} be any (deterministic) algorithm for random-order unit-length interval selection on the restricted domain $[0, \Delta)$ that computes an α -approximation in expectation (over the random order of the stream) using space $O(s)$. Then there is a (deterministic) algorithm for random-order unit-length interval selection on unrestricted domains that computes an $\alpha \cdot (\Delta - 1)/\Delta$ -approximation in expectation (over the random order of the stream) using space $O(s \cdot \Delta \cdot |OPT|)$ that uses \mathcal{A} as a black box.*

Proof. Let \mathcal{A} be any deterministic algorithm for random-order unit-length interval selection on the restricted domain $[0, \Delta)$ that computes an α -approximation in expectation (over the random order of the stream) using space $O(s)$. We use \mathcal{A} as a black-box to present an algorithm for unrestricted domains.

To this end, for all $i \in \mathbb{Z}$, we first define the window $W_i^\Delta = [i, i + \Delta)$. Initially, we mark every window $\{W_i^\Delta : i \in \mathbb{Z}\}$ as inactive. For each interval I arriving in the stream, let $\mathcal{W}_I = \{W_i^\Delta : I \subseteq W_i^\Delta\}$ be the set of windows that fully contain I . For each $W \in \mathcal{W}_I$, if W is marked inactive, create a new instance of \mathcal{A} corresponding to W , and feed I into \mathcal{A} . Then mark W as active. Otherwise, if W is already marked active then there is already an instance of \mathcal{A} running that corresponds to W . In this case, we feed the interval I into the instance of \mathcal{A} . We note that, as a technicality, whenever an interval $I = [a, b]$ is to be fed into an instance \mathcal{A} corresponding to a window W_i^Δ then \mathcal{A} is given the transformed interval $I' = [a - i, b - i]$ so that \mathcal{A} 's input is contained within the restricted domain $[0, \Delta)$.

For each window W , let $OUT(W)$ denote the output from the instance of \mathcal{A} corresponding to W at the end of the stream. We implicitly transform each interval in $OUT(W)$ back so that they correspond to intervals on the unrestricted domain. At the end of the stream output a maximum independent set of intervals from

$$\bigcup_{\substack{i \in \mathbb{Z} \\ W_i^\Delta \text{ is active}}} OUT(W_i^\Delta)$$

We optimize the space used by the algorithm by only storing the set of windows that are marked as active, i.e. each window is implicitly marked as inactive until the time it is explicitly marked as active. Then, a window marked as inactive uses zero space.

The space used by the algorithm is precisely the space used by windows marked as active, and each window uses space $O(s + 1) = O(s)$ (to store \mathcal{A} as well as mark it active). For each window W_i^Δ , define $W_i^\Delta(L) = [i - 1, i - 1 + \Delta)$ and $W_i^\Delta(R) = [i + 1, i + 1 + \Delta)$ to be the windows shifted by one to the left and the right, respectively. Then define

$$\begin{aligned} W(OPT) &= \{W_i^\Delta : \exists I \in OPT \text{ s.t. } I \subseteq W_i^\Delta\}, \\ W_L(OPT) &= \{W(L) : W \in W(OPT)\}, \\ W_R(OPT) &= \{W(R) : W \in W(OPT)\}. \end{aligned}$$

Now define

$$\mathcal{W}' = W(OPT) \cup W_L(OPT) \cup W_R(OPT).$$

4:14 Unit Interval Selection in Random Order Streams

Next, for every interval I , there exists exactly $\Delta - 1$ many values of j such that $I \subseteq W_j^\Delta$. Therefore $|\mathcal{W}'| \leq 3 \cdot \Delta \cdot |OPT|$. Also, the set of windows marked as active must be a subset of \mathcal{W}' . If not, there is a window marked as active that intersects no interval in OPT , and so there is an interval in \mathcal{S} that intersects no interval in OPT . This contradicts the maximality of a maximum independent set. It follows that the total space used by the algorithm is bounded by

$$O(s) \cdot O(\Delta \cdot |OPT|) = O(s \cdot \Delta \cdot |OPT|).$$

For any window W , the set of intervals in \mathcal{S} contained in W is independent of the order of \mathcal{S} . Therefore, the substream of \mathcal{S} on W is also a random-order stream².

Next, for each $j \in [\Delta]$, define the set of disjoint windows $\mathbb{W}_j = \{W_{j+a\Delta}^\Delta : a \in \mathbb{Z}\}$ and define $OPT_j = \{I \in OPT : \exists W \in \mathbb{W}_j \text{ s.t. } I \subseteq W\}$. Then, using that every interval in OPT is contained in exactly $\Delta - 1$ windows and only a single window from any given \mathbb{W}_j ,

$$\sum_{j=1}^{\Delta} |OPT_j| = (\Delta - 1)|OPT|.$$

Therefore, there exists some $k \in [\Delta]$ such that

$$|OPT_k| \geq \frac{\Delta - 1}{\Delta} \cdot |OPT|. \quad (2)$$

We now consider the set of windows \mathbb{W}_k . Each window in \mathbb{W}_k is disjoint so $\bigcup_{W \in \mathbb{W}_k} OUT(W)$ is a valid independent set of the intervals of \mathcal{S} . Then, taking the expectation over the random order of the stream,

$$\begin{aligned} \mathbb{E}[|OUT|] &\geq \mathbb{E} \left[\sum_{W \in \mathbb{W}_k} |OUT(W)| \right] && (\bigcup_{W \in \mathbb{W}_k} OUT(W) \text{ is a valid independent set.}) \\ &= \sum_{W \in \mathbb{W}_k} \mathbb{E}[|OUT(W)|] \\ & \text{(Linearity of Expectation and the substream of } \mathcal{S} \text{ on } W \text{ is also random-order.)} \\ &\geq \sum_{W \in \mathbb{W}_k} \alpha \cdot |\{I \in OPT : I \subseteq W\}| \\ & \text{(} \mathcal{A} \text{ computes an } \alpha\text{-approximation in expectation.)} \\ &= \alpha \cdot \sum_{W \in \mathbb{W}_k} |\{I \in OPT : I \subseteq W\}| \\ &= \alpha \cdot \left| \bigcup_{W \in \mathbb{W}_k} \{I \in OPT : I \subseteq W\} \right| \\ & \text{(Each } W \in \mathbb{W}_k \text{ is disjoint, so each } I \text{ is contained in exactly one } W \in \mathbb{W}_k\text{.)} \\ &= \alpha \cdot |\{I \in OPT : \exists W \in \mathbb{W}_k \text{ s.t. } I \subseteq W\}| \\ &= \alpha \cdot |OPT_k| && \text{(Definition of } OPT_k\text{.)} \\ &\geq \alpha \cdot \frac{\Delta - 1}{\Delta} \cdot |OPT|. && \text{(Equation 2.)} \end{aligned}$$

This meta-algorithm for unrestricted domains is deterministic. Therefore, the final algorithm for unrestricted domains is deterministic if and only if \mathcal{A} is deterministic. ◀

² This is where our argument adds on to the similar argument by [11].

3.5 Completing the Analysis

We now combine Algorithm 1 with Lemma 14 to complete the proof of Theorem 1.

► **Theorem 1.** *There is a deterministic one-pass $O(|OPT|)$ words of space streaming algorithm for Unit Interval Selection on random order streams with expected approximation factor 0.7401, where the expectation is taken over the random order of the stream.*

Proof. The proof is by combining Algorithm 1 with Lemma 14 to obtain an algorithm for random-order unit-length interval streams without any restrictions on the input domain. We first consider the space used by this approach. We then lower bound the expected approximation factor achieved on unrestricted domains with a function that only depends on the integer window length Δ (from Lemma 14), and finally numerically optimise this function.

From Subsection 3.2, on a window of length Δ , we run an algorithm with domain size $\Delta + 2$. Then, from Lemmas 13 and 14, this yields an algorithm for unrestricted domains that uses

$$\begin{aligned} O(S(\Delta + 2) \cdot \Delta \cdot |OPT|) &= O(4^{\Delta+2} \cdot (\Delta + 2)^{\Delta+2} \cdot \Delta \cdot |OPT|) \\ &= O(4^\Delta \cdot (\Delta + 2)^{\Delta+3} \cdot |OPT|), \end{aligned}$$

words of space, where $S(k)$ denotes the space used by an instance of the algorithm instantiated on a domain of length k (from Lemma 13). Then, when $\Delta = O(1)$, this space collapses to become $O(|OPT|)$.

We now consider the approximation factor of the unrestricted algorithm. Observe that a largest independent set in the window $[0, \Delta]$ is of size at most $\Delta - 1$. Therefore, combining Lemmas 12 and 14, for any Δ , there is an algorithm for random-order unit-length interval selection on unrestricted domains that achieves an expected approximation factor of

$$\frac{\Delta - 1}{\Delta} \cdot \min_{\alpha \in \{1, \dots, \Delta - 1\}} \left\{ \frac{1}{\alpha} + \frac{1}{\alpha^2} \cdot \sum_{i=1}^{\alpha} \max \left\{ \begin{array}{l} out(i - 1) + out(\alpha - i - 1) \\ out(\alpha - i) + out(i - 2) \end{array} \right\} \right\}. \quad (3)$$

It remains to pick a suitable constant Δ . Computing (Equation (3)) numerically with $\Delta = 5,000$ (using the recurrence relation for $out(\cdot)$ of Lemma 12) yields a lower bound of (slightly) more than 0.7401 for the expected approximation factor, completing the proof. We discuss the choice of Δ further in the full version of this paper. ◀

4 The Lower Bound

We now establish our lower bound result as stated in Theorem 2.

Given a random order Unit Interval Selection algorithm \mathcal{A} , we define protocol Q that uses \mathcal{A} to solve the INDEX_t problem:

Protocol Q for INDEX_t :

Input

Let Alice and Bob be two players holding a INDEX_t instance (X, A) from the uniform distribution μ . Let \mathcal{A} be any algorithm for random-order Unit Interval Selection. For each $i \in [t]$, define the uniformly and independently distributed

binary random variable $Y_i \in \{0, 1\}$. Furthermore, define the uniformly distributed random variables $B_L = B_R \in [t]$. Let T denote the set containing these random variables, that is, $T = \{Y_1, \dots, Y_t, B_L, B_R\}$.

The Protocol

1. Alice and Bob use shared public randomness to sample each variable in T . They also sample a bijection $\sigma : T \rightarrow [|T|]$.
2. Let $j' = \min\{\sigma(B_L), \sigma(B_R)\}$.
3. For each $i \in [t]$ such that $\sigma(Y_i) < j'$, Alice constructs the interval

$$I[i] = \left[\frac{i}{t+1} + \frac{X[i]}{t^2}, 1 + \frac{i}{t+1} + \frac{X[i]}{t^2} \right].$$

4. Alice runs \mathcal{A} on these constructed intervals using the ordering σ . Note that σ defines an ordering on these intervals by associating each $I[i]$ with $Y[i]$. Alice then sends the state of \mathcal{A} to Bob as the message.
5. For each $i \in [t]$ such that $\sigma(Y[i]) > j'$, Bob constructs the interval

$$I[i] = \left[\frac{i}{t+1} + \frac{Y[i]}{t^2}, 1 + \frac{i}{t+1} + \frac{Y[i]}{t^2} \right].$$

Bob also constructs the intervals

$$J_L = \left[\frac{A}{t+1} - \frac{1}{t^3} - 1, \frac{A}{t+1} - \frac{1}{t^3} \right]$$

and

$$J_R = \left[1 + \frac{A}{t+1} + \frac{1}{t^2} + \frac{1}{t^3}, 2 + \frac{A}{t+1} + \frac{1}{t^2} + \frac{1}{t^3} \right].$$

6. Bob uses the message from Alice to continue running \mathcal{A} on these intervals using the ordering σ , associating J_L with B_L , and J_R with B_R , under σ .

Output

If either \mathcal{A} produces an output of size at most 2, or $\sigma(Y[A]) > j'$, then Bob outputs a uniform random bit. Otherwise, it must be that \mathcal{A} outputs a solution of size three, and $\sigma(Y[A]) < j'$. As observed below, this solution must contain the interval $I[A]$. Then, if $I[A]$ is located at the position $[\frac{A}{t+1}, 1 + \frac{A}{t+1}]$ then Bob outputs the bit $x = 0$, otherwise the interval must be located at position $[\frac{A}{t+1} + \frac{1}{t^2}, 1 + \frac{A}{t+1} + \frac{1}{t^2}]$ and Bob outputs the bit $x = 1$.

Protocol Q works as follows. Given a uniform random input (X, A) to INDEX_t , Alice and Bob first create a random interval instance that is fully described by uniform random bits Y_1, \dots, Y_t as well as the index A from the INDEX_t input: Each bit Y_i corresponds to an interval $I[i]$ so that the intervals $I[1], \dots, I[t]$ all mutually overlap as in Figure 1. The role of the bit Y_i is as follows: If $Y_i = 1$ then the interval $I[i]$ is shifted slightly to the right, while this does not happen if $Y_i = 0$. Then, the index A translates into two additional intervals J_L and J_R that surround the interval $I[A]$ so that $\{J_L, J_R, I[A]\}$ constitutes the unique independent set of size 3.

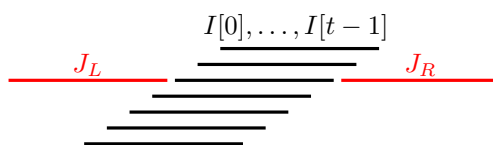
We observe that the instance created so far is a uniform random instance since all the variables Y_1, \dots, Y_t, A are independent uniform random variables.

Next, Alice and Bob choose a uniform random ordering σ of the intervals in this instance. We stress that, by construction, σ is independent of the intervals created, i.e., of the quantities Y_1, \dots, Y_t as well as A .

Alice now replaces some of the bits Y_i by the values X_i as follows: If in the uniform random ordering σ the interval $I[i]$ arrives before both J_L and J_R , then Y_i is updated to the value X_i . Distributionally speaking, we observe that this process replaces uniform random bits Y_i by different uniform random bits X_i (recall that the input to the INDEX_t instance is a uniform random instance). Hence, the distribution of the resulting interval instance is a uniform distribution, and, most importantly, the order of the intervals is random and independent of the instance. The reduction thus creates a random order stream.

We observe that Alice only updates intervals that arrive before the intervals J_L and J_R , or, in other words, the decision as to which intervals are updated depends on the stream order. One may falsely conclude that the resulting instance may therefore be correlated with the arrival order σ , which would mean that the resulting stream is not in uniform random order. We stress, however, that this is not the case since this process only replaces uniform random bits by a different set of uniform random bits. It is therefore not possible to establish a correlation.

Next, we see that whenever Alice encoded the correct bit $X[A]$ into interval $I[A]$, which, as we will see, happens with probability $1/3$, and if, at the same time, the algorithm \mathcal{A} succeeds, then Alice and Bob are able to solve the underlying INDEX_t instance. Otherwise, Bob outputs a uniform random bit. We will see that if \mathcal{A} succeeds with probability greater than $2/3$ then the protocol Q has a success probability strictly larger than $1/2$. The lower bound for INDEX_t as stated in Theorem 4 then implies that Q must communicate $\Omega(t)$ bits, and thus, algorithm \mathcal{A} must also use at least as much space.



■ **Figure 1** An example set of intervals constructed by Protocol Q . The black intervals correspond to each $I[i]$, and the red intervals show J_L and J_R . Whether or not each $I[i]$ interval was constructed using $X[j]$ or a public random bit depends on the sampled permutation σ .

► **Theorem 2.** *Let $\epsilon > 0$ be any small constant. Then:*

1. *Any randomized one-pass streaming algorithm for random-order Unit Interval Selection with expected approximation factor $8/9 + \epsilon$ must use $\Omega(n)$ bits of space.*
2. *For any $\delta > 0$, any randomized one-pass streaming algorithm for random-order Unit Interval Selection with approximation factor $2/3 + \delta$ that succeeds with probability at least $2/3 + \epsilon$ on any input instance must use $\Omega(n)$ bits of space.*

The expectation in (1) and the probability in (2) are taken over both the stream order and the random choices of the algorithm.

Proof. Let \mathcal{A} be a randomized algorithm for Unit Interval Selection with approximation factor $2/3 + \delta$, for any small constant δ , that succeeds with probability at least $2/3 + \epsilon$, for some $\epsilon > 0$, on every input instance, where the probability is taken over both the stream order and the random coin flips of the algorithm. We will first prove that \mathcal{A} must use $\Omega(n)$ space, i.e., we establish result 2. Then, we argue that result 2 immediately implies result 1.

To see that result 2 holds, we first observe that, whenever algorithm \mathcal{A} succeeds then it must output a solution of size 3 since its approximation factor is strictly larger than $2/3$ and every instance created contains an optimal solution of size 3. Second, as argued above, no matter the input instance created, it is presented to \mathcal{A} in uniform random order, which implies that the success probability of the algorithm is indeed $2/3 + \epsilon$. Third, we see that, with probability $1/3$, the interval $I[A]$ is created by Alice and thus correctly reflects the bit $X[A]$. We are thus interested in the probability that both $I[A]$ reflects $X[A]$ and the algorithm \mathcal{A} succeeds on the resulting instance.

Using the events $\mathcal{E}_1 = \text{“Alice creates } I[A]\text{”}$ and $\mathcal{E}_2 = \text{“}\mathcal{A} \text{ succeeds”}$, we obtain:

$$\Pr[\mathcal{E}_1 \text{ and } \mathcal{E}_2] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] = \frac{1}{3} \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1]. \quad (4)$$

Next, we use the fact that the success probability of \mathcal{A} is at least $2/3 + \epsilon$. Hence,

$$\begin{aligned} 2/3 + \epsilon &\leq \Pr[\mathcal{E}_2] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] + (1 - \Pr[\mathcal{E}_1]) \cdot \Pr[\mathcal{E}_2 \mid \overline{\mathcal{E}_1}] \\ &= \frac{1}{3} \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] + \frac{2}{3} \cdot \Pr[\mathcal{E}_2 \mid \overline{\mathcal{E}_1}] \\ &\leq \frac{1}{3} \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] + \frac{2}{3} \cdot 1, \end{aligned}$$

which implies $\epsilon \leq \Pr[\mathcal{E}_2 \mid \mathcal{E}_1]$. Using this bound in Equality 4 yields $\Pr[\mathcal{E}_1 \text{ and } \mathcal{E}_2] \geq \frac{1}{3} \cdot \epsilon$. In other words, with probability $\frac{1}{3} \cdot \epsilon$, Bob solves the underlying INDEX_t instance.

Lastly, in all other cases, i.e., with probability $1 - \epsilon/3$, Bob outputs a uniform random bit. Hence, the protocol Q solves the INDEX_t instance with probability $\epsilon/3 + (1 - \epsilon/3) \cdot \frac{1}{2} = \frac{1}{2} + \epsilon/6$. Then, by Theorem 4, the resulting protocol must use a message of size $\Omega(t)$ bits, and since the message size is identical to the space used by our algorithm, the same lower bound applies to the space usage of \mathcal{A} . Last, since $n = t + 2$ in our construction, result 2 follows.

To see that result 1 holds, consider an algorithm \mathcal{A} for Unit Interval Selection with expected approximation factor $\frac{8}{9} + \epsilon$, for some $\epsilon > 0$, where the expectation is taken over the random coin flips of the algorithm as well as the stream order. We will now argue that \mathcal{A} achieves an approximation factor of at least $2/3 + \epsilon$ with probability $2/3 + \epsilon$. Then, by result 2 we obtain that \mathcal{A} must use $\Omega(n)$ bits of space, thus establishing result 1.

We compute:

$$\begin{aligned} \frac{8}{9} + \epsilon &\leq \mathbb{E}[\text{approx factor}] \\ &\leq \Pr[\text{approx factor} \leq 2/3 + \epsilon] \cdot (2/3 + \epsilon) + \Pr[\text{approx factor} > 2/3 + \epsilon] \cdot 1 \\ &= (1 - \Pr[\text{approx factor} > 2/3 + \epsilon]) \cdot (2/3 + \epsilon) + \Pr[\text{approx factor} > 2/3 + \epsilon], \end{aligned}$$

which implies $\Pr[\text{approx factor} > 2/3 + \epsilon] \geq \frac{2}{3} + \epsilon$ as desired. \blacktriangleleft

5 Conclusion

In this paper, we gave a one-pass random order streaming algorithm for Unit Interval Selection with expected approximation factor 0.7401 that uses space $O(|OPT|)$, where the expectation is taken over the order of the stream. We also proved that, within this space constraint, no algorithm can achieve an expected approximation factor beyond $8/9$, and no algorithm can have an approximation factor better than $2/3$ with probability above $2/3$.

We conclude with two open questions.

1. Our results are not tight. Is there either an algorithm with improved expected approximation guarantee or is it possible to prove a stronger lower bound?
2. We have not addressed arbitrary-length intervals. Is there an algorithm with expected approximation factor above $1/2$ for Interval Selection on arbitrary-length intervals?

References

- 1 Cezar-Mihail Alexandru and Christian Konrad. Interval Selection in Sliding Windows. In Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms (ESA 2024)*, volume 308 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:17, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2024.8.
- 2 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.19.
- 3 Sepehr Assadi and Janani Sundaresan. (noisy) gap cycle counting strikes back: Random order streaming lower bounds for connected components and beyond. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 183–195, 2023. doi:10.1145/3564246.3585192.
- 4 Ainesh Bakshi, Nadiia Chepurko, and David P. Woodruff. Weighted Maximum Independent Set of Geometric Objects in Turnstile Streams. In Jarosław Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*, volume 176 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 64:1–64:22, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.64.
- 5 Aaron Bernstein. Improved bounds for matching in random-order streams. *Theor. Comp. Sys.*, 68(4):758–772, December 2023. doi:10.1007/s00224-023-10155-7.
- 6 Sujoy Bhore, Fabian Klute, and Jelle J. Oostveen. On streaming algorithms for geometric independent set and clique. In Parinya Chalermsook and Bundit Laekhanukit, editors, *Approximation and Online Algorithms - 20th International Workshop, WAOA 2022, Potsdam, Germany, September 8-9, 2022, Proceedings*, volume 13538 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2022. doi:10.1007/978-3-031-18367-6_11.
- 7 Sergio Cabello and Pablo Pérez-Lantero. Interval selection in the streaming model. *Theor. Comput. Sci.*, 702:77–96, 2017. doi:10.1016/J.TCS.2017.08.015.
- 8 Amit Chakrabarti, Graham Cormode, and Andrew McGregor. Robust lower bounds for communication and stream computation. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, STOC '08*, pages 641–650, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1374376.1374470.
- 9 Amit Chakrabarti, T. S. Jayram, and Mihai Pundefinedtraşcu. Tight lower bounds for selection in randomly ordered streams. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '08*, pages 720–729, USA, 2008. Society for Industrial and Applied Mathematics.
- 10 Graham Cormode, Jacques Dark, and Christian Konrad. Independent sets in vertex-arrival streams. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 45:1–45:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.45.
- 11 Jacques Dark, Adithya Diddapur, and Christian Konrad. Interval selection in data streams: Weighted intervals and the insertion-deletion setting. In Patricia Bouyer and Srikanth Srinivasan, editors, *43rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2023, December 18-20, 2023, IIT Hyderabad, Telangana, India*, volume 284 of *LIPIcs*, pages 24:1–24:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.FSTTCS.2023.24.
- 12 Yuval Emek, Magnús M. Halldórsson, and Adi Rosén. Space-constrained interval selection. *ACM Trans. Algorithms*, 12(4), September 2016. doi:10.1145/2886102.

- 13 Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009. doi:10.1137/07069328X.
- 14 Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, January 1985. doi:10.1145/2455.214106.
- 15 Michael Kapralov. Better bounds for matchings in the streaming model. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6–8, 2013*, pages 1679–1697. SIAM, 2013. doi:10.1137/1.9781611973105.121.
- 16 Sanjeev Khanna, Christian Konrad, and Cezar-Mihail Alexandru. Set cover in the one-pass edge-arrival streaming model. In Floris Geerts, Hung Q. Ngo, and Stavros Sintos, editors, *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2023, Seattle, WA, USA, June 18–23, 2023*, pages 127–139. ACM, 2023. doi:10.1145/3584372.3588678.
- 17 Morteza Monemizadeh, S. Muthukrishnan, Pan Peng, and Christian Sohler. Testable Bounded Degree Graph Properties Are Random Order Streamable. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 131:1–131:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2017.131.
- 18 Anup Rao and Amir Yehudayoff. *Communication Complexity: and Applications*. Cambridge University Press, 2020.
- 19 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 209–213, 1979. doi:10.1145/800135.804414.