Interval Selection in Sliding Windows

Cezar-Mihail Alexandru
School of Computer Science, University of Bristol, UK

Christian Konrad
School of Computer Science, University of Bristol, UK

Abstract

We initiate the study of the Interval Selection problem in the (streaming) sliding window model of computation. In this problem, an algorithm receives a potentially infinite stream of intervals on the line, and the objective is to maintain at every moment an approximation to a largest possible subset of disjoint intervals among the L most recent intervals, for some integer L.

We give the following results:

1. In the unit-length intervals case, we give a 2-approximation sliding window algorithm with space $\tilde{O}(|OPT|)$, and we show that any sliding window algorithm that computes a $(2-\varepsilon)$-approximation requires space $\Omega(L)$, for any $\varepsilon > 0$.
2. In the arbitrary-length case, we give a $(\frac{11}{3} + \varepsilon)$-approximation sliding window algorithm with space $\tilde{O}(|OPT|)$, for any constant $\varepsilon > 0$, which constitutes our main result.[1] We also show that space $\Omega(L)$ is needed for algorithms that compute a $(2.5 - \varepsilon)$-approximation, for any $\varepsilon > 0$.

Our main technical contribution is an improvement over the smooth histogram technique, which consists of running independent copies of a traditional streaming algorithm with different start times. By employing the one-pass 2-approximation streaming algorithm by Cabello and Pérez-Lantero [Theor. Comput. Sci. '17] for Interval Selection on arbitrary-length intervals as the underlying algorithm, the smooth histogram technique immediately yields a $(4 + \varepsilon)$-approximation in this setting. Our improvement is obtained by forwarding the structure of the intervals identified in a run to the subsequent run, which constrains the *shape* of an optimal solution and allows us to target optimal intervals differently.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Streaming models

Keywords and phrases Sliding window algorithms, Streaming algorithms, Interval selection

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.8

Related Version *Full Version*: `https://arxiv.org/abs/2405.09338` [2]

Funding *Cezar-Mihail Alexandru*: Supported by EPSRC Doctoral Training Studentship EP/T517872/1.
Christian Konrad: Supported by EPSRC New Investigator Award EP/V010611/1.

1 Introduction

Sliding Window Model. The *sliding window model* of computation introduced by Datar et al. [10] captures many of the challenges that arise when processing infinite data streams. In this model, an algorithm receives an infinite stream of data items and is required to maintain, at every moment, a solution to a given problem on the current *sliding window*, i.e., on the L most recent data items, for an integer L. The objective is to design algorithms that use much less space than the size of the sliding window L.

[1] We use the notation $\tilde{O}(.)$ to mean $O(.)$ where polylog factors and dependencies on ε are suppressed.

Many modern data sources are best modelled as infinite data streams rather than as data sets of large but finite sizes. For example, the sequence of Tweets on X (formerly Twitter), the sequence of IP packages passing through a network router, and continuous sensor measurements for monitoring the physical world are a priori unending. Such data sets typically constitute time-series data, where the resulting data stream is ordered with respect to the data items' creation times. When processing such streams, it is reasonable to focus on the most recent data items (as it is modelled in the sliding window model by the sliding window size L) since the near past usually affects the present more strongly than older data.

The sliding window model should be contrasted with the more traditional *one-pass data streaming model*. In the data streaming model, an algorithm processes a finite stream of n data items and is tasked with producing a single output once all items have been processed. Similar to the sliding window model, the objective is to design algorithms that use as little space as possible, in particular, sublinear in the length of the stream. Since sliding window algorithms with $L = n$ can immediately be used in the data streaming model, problems are generally harder to solve in the sliding window model.

Interval Selection Problem. In this work, we initiate the study of the Interval Selection problem in the sliding window model. Given a set $\mathcal{S}$ of n intervals on the real line, the objective is to find a subset $\mathcal{I} \subseteq \mathcal{S}$ of pairwise non-overlapping intervals of maximum cardinality. The problem can also be regarded as the Maximum Independent Set problem in the interval graph associated with the intervals $\mathcal{S}$. We consider both the unit-length case, where all intervals are of length 1, and the arbitrary-length case, where no restriction on the lengths of the intervals is imposed.

Interval Selection is fully understood in the one-pass streaming model. Emek et al. [11] gave a $\frac{3}{2}$-approximation streaming algorithm for unit-length intervals and a 2-approximation streaming algorithm for arbitrary-length intervals. Both algorithms use space $O(|OPT|)$, where OPT denotes an optimal solution, assuming that the space required for storing an interval is $O(1)$. Emek et al. also gave matching lower bounds, showing that, for both the unit-length and the arbitrary-length case, slightly better approximations require space $\Omega(n)$. Subsequently, Cabello and Pérez-Lantero [6] also gave algorithms for the unit-length and the arbitrary-length cases that match the guarantees of those by Emek et al. but are significantly simpler. We will reuse one of the algorithms by Cabello and Pérez-Lantero in this paper. Last, weighted intervals as well as the insertion-deletion setting, where previously inserted intervals can be deleted again, have also been considered [9, 3], where [3] addresses the challenge of outputting the size or weight of a largest/heaviest independent set rather than outputting the intervals themselves.

The Smooth Histogram Technique. Braverman and Ostrovsky [5] introduced the smooth histogram technique, which allows deriving sliding window algorithms from traditional streaming algorithms at the expense of slightly increased space requirements and approximation guarantees. The method works as follows. Given a streaming algorithm $\mathcal{A}$ for a specific problem P that fulfills certain smoothness properties (see [5] for details), multiple copies of $\mathcal{A}$ are run with different starting positions in the stream. The runs are such that consecutive runs differ only slightly in solution quality, and, thus, when a run expires due to the fact that its starting position fell out of the current sliding window, the subsequent run can be used to still yield an acceptable solution. The smooth histogram technique can be applied to the Interval Selection algorithms by Emek et al. [11] and by Cabello and Pérez-Lantero [6], and we immediately obtain sliding window algorithms for both the unit-length and the arbitrary-length cases using space $\tilde{O}(|OPT|)^2$. For unit-length intervals, the resulting approximation factor is $3+\varepsilon$, for any $\varepsilon > 0$, and for arbitrary-length intervals, the approximation factor is $4+\varepsilon$, for any $\varepsilon > 0$. We will provide the analysis of the $(4+\varepsilon)$-approximation for arbitrary-length intervals in this paper (**Theorem 6**) since it forms the basis of the analysis of one of our algorithms.

Our Results. In this work, we show that it is possible to improve upon the guarantees obtained from the smooth histogram technique. We give deterministic sliding window algorithms and lower bounds that also apply to randomized algorithms for Interval Selection for both the unit-length and arbitrary-length cases. Our algorithms use space $\tilde{O}(|OPT|)$ at any moment during the processing of the stream, where OPT denotes an optimal solution in the current sliding window. Observe that OPT may vary throughout the processing of the stream, and, thus, the space used by our algorithms may therefore also change accordingly.

Regarding unit-length intervals, we give a 2-approximation sliding window algorithm with space $O(|OPT|)$, and we prove that any sliding window algorithm with an approximation guarantee of $2-\varepsilon$, for any $\varepsilon > 0$, requires space $\Omega(L)$. Recall that, in the streaming model, a $\frac{3}{2}$-approximation can be achieved with space $O(|OPT|)$. Our lower bound thus establishes a separation between the sliding window and the streaming models for unit-length intervals.

In the arbitrary-length case, we give a $(\frac{11}{3}+\varepsilon)$-approximation sliding window algorithm with space $\tilde{O}(|OPT|)$, improving over the smooth histogram technique, which constitutes our main and most technical result. We also prove that any $(\frac{5}{2}-\varepsilon)$-approximation algorithm, for any $\varepsilon > 0$, requires space $\Omega(L)$. Since, in the streaming model, a 2-approximation can be achieved with space $O(|OPT|)$, our lower bound also establishes a separation between the sliding window and the streaming models in the arbitrary-length case.

We summarize and contrast our results with results from the streaming model in Figure 1.

	Streaming model [11, 6]		**Sliding window model** (this paper)	
	Algorithm	LB	Algorithm	LB
Unit-length Intervals	$\frac{3}{2}$	$\frac{3}{2}-\varepsilon$	2 (**Thm 3**)	$2-\varepsilon$ (**Thm 4**)
Arbitrary-length Intervals	2	$2-\varepsilon$	$\frac{11}{3}$ (**Thm 10**)	$\frac{5}{2}-\varepsilon$ (**Thm 11**)

Figure 1 Approximation factors achievable in the streaming and sliding window models. All algorithms use space $\tilde{O}(|OPT|)$, while all lower bound results are to be interpreted in that achieving the stated approximation guarantee requires space $\Omega(n)$ (streaming) or $\Omega(L)$ (sliding window model). The lower bound results hold for any $\varepsilon > 0$.

A Lack of Lower Bounds in the Sliding Window Model. Interestingly, to the best of our knowledge, for graph problems (recall that the interval selection problem is an independent set problem on interval graphs) no separation result between the one-pass streaming and the sliding window models are known. In particular, we are not aware of any space lower bounds for graph problems specifically designed for the sliding window setting, and the only lower bounds that apply are those that carry over from the one-pass streaming setting. Our work is thus the first to establish such a separation. While our results for arbitrary-length intervals are not tight, we stress that for most problems considered, including Maximum Matching and Minimum Vertex Cover, no tight bounds are known. It is unclear whether this is due to a lack of techniques for improved algorithms or for stronger lower bounds.

[2] We use the notation $\tilde{O}(.)$ to mean $O(.)$ where polylog factors and dependencies on ε are suppressed.

Techniques. We will first discuss the key ideas behind our results for unit-length intervals, and then discuss our results for arbitrary-length intervals.

Unit-length Intervals. Our algorithm for unit-length intervals is surprisingly simple yet optimal, as established by our lower bound result. For each integer r, maintain the latest interval within the current sliding window whose left endpoint lies in the interval $[r, r+1)$ if there is one. We argue that, if at any moment, the algorithm stores D intervals, then we can extract an independent set of size at least $D/2$ by considering either only the intervals $[r, r+1)$ where r is odd or where r is even, while OPT is bounded by $D/2 \leq OPT \leq D$, which establishes both the approximation factor of 2 and the space requirements. We note that the idea of considering either only the odd or even intervals for obtaining a 2-approximation was previously used by [3].

Our lower bound for unit-length intervals is obtained by a reduction to the Index_n problem in the one-way two-party communication setting. In this setting, there are two parties, denoted Alice and Bob. Each party holds a portion of the input data. Alice sends a single message to Bob, who then outputs the result of the computation. The objective is to solve a problem using a message of smallest possible size. In Index_n, Alice holds a bit-string $X \in \{0,1\}^n$, and Bob holds an index $J \in [n]$, where $[n] = \{1, 2, 3, ..., n\}$. The objective for Bob is to report the bit $X[J]$. It is well-known that a message of size $\Omega(n)$ is needed to solve the problem.

We argue that a sliding window algorithm $\mathcal{A}$ for Interval Selection on unit-length intervals with approximation guarantee slightly below 2 can be used to solve $\mathsf{Index}_{\Theta(L)}$. To this end, Alice translates the bit-string X into a *clique gadget*, i.e., a stack of overlapping $\Theta(L)$ interval slots that are slightly shifted from left-to-right, where interval i is present in the stack if and only if $X[i] = 1$. Clique gadgets have been used in all previous space lower bound constructions for intervals [11, 3, 9]. Alice then runs $\mathcal{A}$ on these intervals and sends the memory state of $\mathcal{A}$ to Bob. Bob subsequently feeds an interval located slightly to the right of the slot of interval J into the execution of $\mathcal{A}$ such that Bob's interval overlaps with all interval slots at positions $\geq J+1$ and does not overlap with all interval slots at positions $\leq J$. The key idea of this reduction is that, since $\mathcal{A}$ is a sliding window algorithm, it must be able to report a valid solution even if any prefix of intervals of the stack are deleted/have expired. Consider thus the situation when the intervals that are located in the first $J-1$ slots have expired. Then, the resulting instance has an independent set of size 2 if and only if $X[J] = 1$, otherwise a largest independent set is of size 1. Since the approximation factor of $\mathcal{A}$ is below 2, $\mathcal{A}$ can thus distinguish between the two cases and solve $\mathsf{Index}_{\Theta(L)}$. Since Alice only sent the memory state of $\mathcal{A}$ to Bob, we also obtain a space lower bound for $\mathcal{A}$. While this description covers the key idea of our lower bound, we note that our actual construction is slightly more involved due to an additional technical challenge. See proof of Theorem 4 for details.

Our lower bound construction shares similarities with the lower bounds by [3] and [11], as both of these lower bounds also work with clique gadgets and special intervals that render a specific interval in the clique gadget important. In [3], a reduction to the Augmented-Index problem is given in order to obtain a space lower bound for the *dynamic streaming setting*, where previously inserted intervals can be deleted again at any moment. In Augmented-Index, besides the index J, Bob also holds the prefix $X[1, \ldots, J-1]$. While in our setting, intervals are deleted due to the shifting sliding window, in their lower bound, intervals are explicitly deleted by Bob.

Arbitrary-length Intervals. Our algorithm and our lower bound for arbitrary-length intervals are substantially more involved, and our $(\frac{11}{3} + \varepsilon)$-approximation algorithm constitutes the main technical result of this paper.

Our algorithm constitutes an improvement over the smooth histogram method. Using the one-pass 2-approximation streaming algorithm for arbitrary-length intervals by Cabello and Pérez-Lantero [6], which we abbreviate by $\mathcal{CP}$, as the base algorithm of the smooth histogram method, we immediately obtain a $(4+\varepsilon)$-approximation sliding window algorithm using $\tilde{O}(|OPT|)$ space. The key idea of the method is to maintain various runs of $\mathcal{CP}$ with different starting times that are sufficiently spaced out so that only a logarithmic number of runs are needed, yet adjacent runs still have similar output sizes. Then, when a run expires, the subsequent run can still be used to report a good enough solution.

We observe that the executions of $\mathcal{CP}$ in the smooth histogram method are independent. Our key contribution that gives rise to our improvement is to forward the structure identified in a run of $\mathcal{CP}$ to the subsequent run. The $\mathcal{CP}$ algorithm, which we will discuss in detail in Section 4.1.1, maintains a partition of the real line that restrains the possible locations of optimal intervals that are yet to arrive in the stream. We target these locations individually in the subsequent run by initiating additional runs of $\mathcal{CP}$ on restricted domains where we expect to find many of these optimal intervals.

Our approach relies on a property of the $\mathcal{CP}$ algorithm that, at first glance, seems relatively insignificant. As proved by Cabello and Pérez-Lantero, the $\mathcal{CP}$ algorithm produces a solution of size at least $(|OPT|+1)/2$, and thus only has an approximation factor of 2 in an asymptotic sense. Consequently, if OPT is a small constant then the algorithm achieves an approximation factor strictly below 2. We exploit this property in that we execute the additional runs of $\mathcal{CP}$ on small domains where we expect to find only a small constant number of optimal intervals, see Section 4.1.3 for further details.

Our $(2.5-\varepsilon)$-approximation lower bound for arbitrary-length intervals is also achieved via a reduction to a hard problem in one-way communication complexity. However, instead of exploiting the hardness of the two-party problem Index as in the unit-lengths case, we use the three-party problem Chain_3 introduced by Cormode et al. [7] instead. In Chain_3, the first two parties and the last two parties hold separate Index instances $(X_1, J_1), (X_2, J_2) \in \{0,1\}^n \times [n]$ that are correlated in that they have the same answer bit, i.e., $X_1[J_1] = X_2[J_2] =: x$, and the objective for the third party is to determine the bit x. Chain_3 also requires a message of size $\Omega(n)$ to be solved. Similar to the unit-length case, the first two parties introduce clique gadgets based on the bit-strings X_1 and X_2, and the third party introduces additional crucial intervals. The strength of using Chain_3 is that, if the answer bit is zero, then the crucial intervals corresponding to $X_1[J_1]$ and $X_2[J_2]$ of all clique gadgets are missing, while if the answer bit is one then all of these intervals are present. The method thus allows us to work with multiple clique gadgets instead of only a single one, which we exploit to obtain a stronger lower bound. See Section 4.2 for details.

Further Related Work. Crouch et al. [8] initiated the study of graph problems in the sliding window model (recall that Interval Selection is an independent set problem on interval graphs). They showed that, similar to the streaming model, there exist sliding window algorithms that use space $\tilde{O}(n)$ for deciding Connectivity and Bipartiteness, where n is the number of vertices in the input graph. They also gave positive results for the computation of cut-sparsifiers, spanners and minimum spanning trees, and they initiated the study of the Maximum Matching problem in the sliding window model (see below).

The smooth histogram technique has been successfully applied for designing sliding window algorithms for graph problems, and the state-of-the-art sliding window algorithms for Maximum Matching and Minimum Vertex Cover rely on the smooth histogram technique.

For Maximum Matching, a 2-approximation with space $\tilde{O}(n)$ can easily be achieved in the streaming model by running the GREEDY matching algorithm, and the smooth histogram method immediately yields a $(4+\varepsilon)$-approximation sliding window algorithm when built on GREEDY. Crouch et al. [8] observed that the resulting algorithm can be analyzed more precisely and showed that it actually yields a $(3+\varepsilon)$-approximation sliding window algorithm. Regarding the weighted version of the Maximum Matching problem, the smooth histogram technique immediately yields a $(4+\varepsilon)$-approximation using the $(2+\varepsilon)$-approximation streaming algorithm by [16], and, again, as proved by Biabani et al. [4], the analysis can be tailored to the Maximum Matching problem to establish an approximation factor of $3.5+\varepsilon$ without changing the algorithm. Alexandru et al. [1] then improved the approximation factor to $3+\varepsilon$ by running the smooth histogram algorithm with a slightly different objective function.

Regarding the Minimum Vertex Cover problem, a smooth histogram-based algorithm is known to yield an approximation factor of $(3+\varepsilon)$ [17], improving over previous work [14].

Outline. In Section 2, we give notation, we provide some clarification on the sliding window model, and we introduce hard communication problems that we rely on for proving our lower bound results. Then, in Section 3, we give our algorithm and lower bound for the case of unit-length intervals and in Section 4, we give our algorithm and lower bound for arbitrary-length intervals. Finally, we conclude in Section 5 with open problems.

2 Preliminaries

For a set of intervals $\mathcal{I}$, we denote by $OPT(\mathcal{I})$ an independent subset of $\mathcal{I}$ of maximum size. We also apply $OPT(.)$ to substreams of intervals and to data structures that store intervals.

Sliding Window Algorithms. Throughout the document, we denote by L the size of the sliding window, and we assume that L is large enough, i.e., larger than a suitably large constant. For two streams of intervals A, B we denote the stream that is obtained by concatenating A and B simply by AB, i.e., we omit a concatenation symbol. Furthermore, for simplicity, we assume that the space required to store an interval is $O(1)$. However, if instead k bits are accounted for storing an interval then the space complexities of our algorithms need to be multiplied by k.

Communication Complexity. As it is standard in the data streaming literature, our space lower bounds are proved via reductions to problems in the *one-way communication setting*. In this setting, multiple parties $P_1, P_2, \dots, P_k$ each hold a portion of the input data and communicate in order to solve a problem. Communication is *one-way*, i.e., party P_1 sends a message to P_2, who in turn sends a message to P_3. This continues until party P_k has received a message from party P_{k-1} and then outputs the result of the computation. The parties can make use of public and private randomness and need to report a correct solution with probability $2/3$. We refer the reader to [15] for an introduction to communication complexity.

We will exploit the hardness of the two-party communication problem Index_n, where we denote the first party by Alice and the second party by Bob, and the k-party communication problem Chain_k, which was recently introduced by Cormode et al. [7].

Index_n:
- Input: Alice holds a bit-string $X \in \{0,1\}^n$, and Bob holds an index $J \in [n]$.
- Output: Bob outputs $X[J]$.

It is well-known that solving Index_n requires Alice to send a message of size $\Omega(n)$.

▶ **Theorem 1** (e.g. [13]). *Every randomized constant-error one-way communication protocol for Index_n requires a message of size $\Omega(n)$.*

The problem $\mathsf{Chain}_k(n)$ can be regarded as chaining together $k-1$ instances of Index_n, where the instances are correlated in that they are guaranteed to have the same output.

$\mathsf{Chain}_k(n)$:

- Input: For $1 \leq i \leq k-1$, player P_i receives a bitvector $X_i \in \{0,1\}^n$. Additionally, for any $2 \leq i \leq k$ player P_i receives an index $J_{i-1} \in [n]$. The inputs are correlated such that

$$X_1[J_1] = X_2[J_2] = \cdots = X_{k-1}[J_{k-1}] = x \in \{0,1\} \ .$$

- Output: Player P_k outputs x.

Sundaresan [18] recently settled the communication complexity of $\mathsf{Chain}_k(n)$, improving over the previous lower bounds by Cormode et al. [7] and Feldman et al. [12]:

▶ **Theorem 2** ([18]). *Every constant-error one-way communication protocol that solves $\mathsf{Chain}_k(n)$ requires at least one message of size $\Omega(n/k)$.*

3 Unit-length Intervals

In this section, we give our sliding window algorithm and our lower bound for unit-length intervals.

We now describe our algorithm for unit-length intervals (see Algorithm 1).

Our algorithm is simple: For each integer r, the algorithm maintains in $\texttt{latest}(r)$ the latest interval of the current sliding window with its left boundary in $[r, r+1)$. The key observation, which was also used in [3], is that the intervals $\{\texttt{latest}(r) \ : \ r \text{ even}\}$ and $\{\texttt{latest}(r) \ : \ r \text{ odd}\}$ form independent sets, and one of these sets constitutes a 2-approximation.

Algorithm 1 Sliding window algorithm for Interval Selection on unit-length intervals.

Input: Stream S of unit-length intervals, window length L

Initialization:

```
1: latest ← ∅ the indexed set of stored intervals
```

Streaming:

```
1: while an interval $I = [r, r+1]$ is revealed, for some real number $r$ do
2:     latest($\lfloor r \rfloor$) ← $I$
3:     if $\exists\ J' = [r', r'+1] \in$ latest that has expired then
4:         latest($\lfloor r' \rfloor$) ← ∅
```

Post-processing:

```
1: Return $OPT$(latest)
```

▶ **Theorem 3.** *Algorithm 1 is a deterministic 2-approximation sliding window algorithm for Interval Selection on unit-length intervals that, at any moment, uses $O(|OPT|)$ space, where OPT is a maximum independent set of intervals in the current sliding window.*

Proof. We will first prove that Algorithm 1 indeed computes a 2-approximation, and then argue that the algorithm satisfies the memory requirements.

We call a unit-length interval I active if it is included in the current sliding window (one of the L most recent intervals of the stream). Otherwise, we say that I is expired.

Let OPT be a maximum independent set in the current sliding window and let ALG be the independent set reported by Algorithm 1. Define the indexed set `latest` as in the algorithm.

Approximation. We will show that

$$|OPT| \leq |\texttt{latest}| \leq 2 \cdot |ALG| \tag{1}$$

holds, which then establishes the approximation factor of the sliding window algorithm.

First, we will prove $|OPT| \leq |\texttt{latest}|$ holds. To this end, we will show that the function $f : OPT \rightarrow \texttt{latest}$ defined as $f([x, x+1]) = \mathit{latest}(\lfloor x \rfloor)$ is injective.

We will first argue that f is well-defined in that $\texttt{latest}(\lfloor x \rfloor)$ exists, for every $[x, x+1] \in OPT$. Indeed, by inspecting the algorithm, when $I := [x, x+1] \in OPT$ arrives in the stream, $\texttt{latest}(\lfloor x \rfloor)$ is set to I, and, in particular, while I is active, $\texttt{latest}(\lfloor x \rfloor)$ is never set to $\emptyset$. It may, however, happen that it is replaced with an interval which appeared after I. In both cases, f is well-defined.

To see that f is injective, observe that for any two intervals in OPT, since these intervals are independent and of unit length, the integer parts of their left endpoints are distinct. Hence, $f(I_1) \neq f(I_2)$, for any two distinct intervals $I_1, I_2 \in OPT$.

Since f is well-defined and injective, we obtain that $|OPT| \leq |\texttt{latest}|$, which thus proves the first inequality of Inequality 1. It remains to prove the second, i.e., that $|\texttt{latest}| \leq 2 \cdot |ALG|$ also holds.

To see this, observe that, for two integers $x \neq y$ of the same parity, $\texttt{latest}(x)$ and $\texttt{latest}(y)$ (if they exist) are independent. This is because $|y - x| \geq 2$ and the intervals have unit length. By the pigeonhole principle, there are at least $\frac{|\texttt{latest}|}{2}$ intervals where their indices inside `latest` have the same parity, which implies that $|ALG| \geq \frac{|\texttt{latest}|}{2}$.

Space. The algorithm stores $|\texttt{latest}|$ intervals in the current sliding window. Then, as proved above, we have $|\texttt{latest}| \leq 2|ALG| \leq 2|OPT|$, which implies that the space used by the algorithm is $\mathrm{O}(|OPT|)$. ◀

We also prove the following lower bound:

▶ **Theorem 4.** *Let $\varepsilon > 0$ be any small constant. Then, any algorithm in the sliding window model that computes a $(2-\varepsilon)$-approximate solution to* Interval Selection *on unit-length intervals with probability at least $2/3$ requires a memory of size $\Omega(L)$.*

Proof. Let $\mathcal{A}$ be a sliding window algorithm for Interval Selection on unit-length intervals with approximation factor $2 - \varepsilon$, for some $\varepsilon > 0$.

We will show how $\mathcal{A}$ can be used in order to obtain a communication protocol for Index_{L-2}.

To this end, let $(X, J) \in \{0,1\}^{L-2} \times [L-2]$ be Alice and Bob's input to Index_{L-2}. The two players proceed as follows:

- **Alice:** Alice feeds the intervals $I_1, I_2, \ldots, I_{L-2}$ into $\mathcal{A}$ (in the given order), where

$$I_i = \begin{cases} [\frac{i}{2L-1}, 1 + \frac{i}{2L-1}], & \text{if } X[i] = 1 \ , \\ [1 - \frac{i}{L^2}, 2 - \frac{i}{L^2}], & \text{if } X[i] = 0 \ . \end{cases}$$

Alice then sends the memory state of $\mathcal{A}$ to Bob.

- **Bob:** Using Alice's message, Bob continues the execution of $\mathcal{A}$ and feeds the interval

$$I_{L-1} = \left[1 + \frac{J}{2L-1} + \frac{1}{(2L-1)^2}, 2 + \frac{J}{2L-1} + \frac{1}{(2L-1)^2}\right]$$

into $\mathcal{A}$. Bob also adds the intervals $I_i = [\frac{i}{2L-1}, 1 + \frac{i}{2L-1}]$ to $\mathcal{A}$, for $L \le i \le L+J-1$ in order to make the sliding window of the algorithm $\mathcal{A}$ advance. Bob computes $\mathcal{A}$'s output in the latest sliding window consisting of the intervals defined as $\mathcal{S} = \{I_i | J \le i \le L+J-1\}$.

This construction is illustrated in Figure 2.

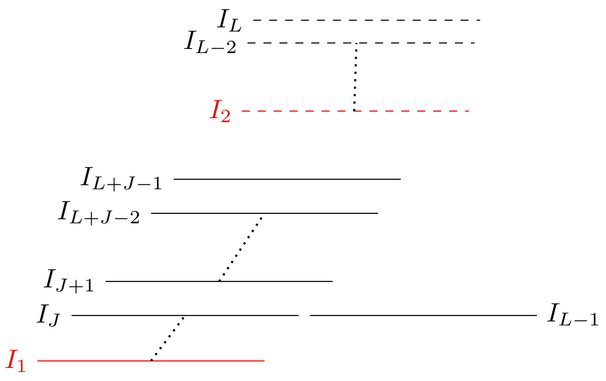

Figure 2 This figure illustrates the instances created by Alice and Bob in the proof of Theorem 4 for an instance of Index$_{L-2}$ with $X[J] = 1$. The dashed intervals on the upper part correspond to the zero elements of the bitvector X. The red intervals I_1, I_2 correspond to expired intervals. I_J is the only non-expired interval disjoint with the special interval I_{L-1}. Since $X[J] = 1$, the optimal solution is of size 2. If $X[J]$ was equal to 0 then the interval I_J would not be disjoint with I_{L-1}, and, thus, an optimal solution would be of size 1.

We observe that if $X[J] = 1$ then $|OPT(\mathcal{S})| = |\{I_J, I_{L-1}\}| = 2$, while if $X[J] = 0$ then $|OPT(\mathcal{S})| = 1$. Since $\mathcal{A}$ has an approximation factor of $2 - \varepsilon$, $\mathcal{A}$ needs to report the unique solution of size 2 if $X[J] = 1$, and a solution of size 1 when $X[J] = 0$. Bob can thus distinguish between the two cases and solve Index$_{L-2}$.

Since the protocol solves Index$_{L-2}$, by Theorem 1, the protocol must use a message of size $\Omega(L)$. The protocol's message is $\mathcal{A}$'s memory state, and, hence, $\mathcal{A}$ must use space $\Omega(L)$. ◀

4 Arbitrary-length Intervals

In this section, we give our $(\frac{11}{3} + \varepsilon)$-approximation sliding window algorithm and our $(\frac{5}{2} - \varepsilon)$-approximation lower bound for Interval Selection on arbitrary-length intervals.

4.1 $(\frac{11}{3} + \varepsilon)$-approximation Sliding Window Algorithm

Our algorithm is obtained by running multiple instances of the Cabello and Pérez-Lantero streaming algorithm for Interval Selection on arbitrary-length intervals [6]. In the following, we abbreviate the algorithm by $\mathcal{CP}$. Since we employ various properties of the $\mathcal{CP}$ algorithm, we discuss the $\mathcal{CP}$ algorithm in Subsection 4.1.1. We use the $\mathcal{CP}$ algorithm in the context of the smooth histogram technique, which we discuss in Subsection 4.1.2. Finally, we give our sliding window algorithm and its analysis in Subsection 4.1.3.

4.1.1 Cabello and Pérez-Lantero Algorithm

The Cabello and Pérez-Lantero algorithm $\mathcal{CP}$ is depicted in the full version of the paper.

The key idea behind the algorithm is to maintain a partition $\mathcal{R}$ of the real line $\mathbb{R}$ that we refer to as a region partition. Initially, the algorithm starts with the single region $\mathcal{R} = \{\mathbb{R}\}$, and as the algorithm proceeds, the real line is partitioned into half-open intervals. This is achieved as follows. Arriving intervals that cross a region boundary are ignored. Consider thus an arriving interval I that lies entirely within a region. In each region, the algorithm stores the left-most (the interval with the left-most right delimiter) and right-most (the interval with the right-most left delimiter) intervals within the region that it has observed thus far. If the interval I together with either the left-most or the right-most interval of the region forms an independent set of size two then the region is split into two regions and the left-most and right-most intervals are updated accordingly. Otherwise, if I intersects with both the left-most and right-most intervals of the region then I is only used to potentially replace the left-most and/or right-most intervals of the region.

Some key properties of the algorithm that we will reuse in this work are summarized in Figure 3 (see [6] for proofs).

C1 For each region $R \in \mathcal{R}$, the $\mathcal{CP}$ algorithm stores at least one (and at most two) intervals and the input instance is such that there are no two disjoint intervals that lie within region R.
C2 The $\mathcal{CP}$ algorithm outputs a solution of size $|\mathcal{R}|$, i.e., one interval per region. Furthermore, we have that $|\mathcal{R}| \geq \frac{|OPT|+1}{2}$, i.e., the algorithm has an approximation factor slightly better than 2.
C3 The algorithm uses space $O(|OPT|)$.

Figure 3 Key Properties of the $\mathcal{CP}$ Algorithm.

Besides these properties, we require another property that allows us to employ the algorithm in the context of the smooth histogram technique:

▶ **Lemma 5.** *The $\mathcal{CP}$ algorithm is monotonic, i.e., for any two streams of intervals A, B we have that*

$$|\mathcal{CP}(A)| \leq |\mathcal{CP}(AB)| .$$

Proof. The output produced by the $\mathcal{CP}$ algorithm consists of one interval per region. The lemma then follows since, by construction, the number of regions cannot decrease. ◀

4.1.2 The Smooth Histogram Technique

The $\mathcal{CP}$ algorithm can be employed in the context of the smooth histogram method to yield a $(4 + \varepsilon)$-approximation sliding window algorithm for Interval Selection for arbitrary-length intervals that uses space $\tilde{O}(|OPT|)$. This is achieved as follows (see Algorithm 2):

Upon the arrival of a new interval, Algorithm 2 first creates a new run of the $\mathcal{CP}$ algorithm and feeds the new interval into all currently running copies of $\mathcal{CP}$. The method relies on a clever way of deleting unnecessary runs: A run is deleted if it is expired (i.e it contains an interval which appeared before the start of the current window) and if the closest run with earlier start time and the closest run with later start time are such that their solutions differ in size by less than a $1 + \varepsilon$ factor. Consider the moment after a clean-up took place, and let us denote the stored runs by $\mathcal{CP}_1, \ldots, \mathcal{CP}_\ell$. The clean-up rule implies that the stored runs have the properties depicted in Figure 4.

Algorithm 2 Smooth Histogram Technique applied to the $\mathcal{CP}$ algorithm.

```
1: while an Interval $I$ is revealed do
2:     Create new instance of the $\mathcal{CP}$ algorithm
3:     Feed $I$ into all $\mathcal{CP}$ instances that are currently running
4:     Clean-up:
5:       Remove oldest run of $\mathcal{CP}$ if it has expired
6:       Denote by $\mathcal{CP}_1, \mathcal{CP}_2, \ldots$ the runs sorted in increasing order with respect to their
7:          starting positions. Then, repeatedly remove a run $\mathcal{CP}_i$ if maintained solutions of
8:          the adjacent runs $\mathcal{CP}_{i-1}$ and $\mathcal{CP}_{i+1}$ are within a factor of $1+\varepsilon$ in size
9:     output solution of oldest run
```

S1 For every $i \leq \ell - 2$: $|\mathcal{CP}_i| \geq (1+\varepsilon)|\mathcal{CP}_{i+2}|$, and
S2 Either $|\mathcal{CP}_i| \leq (1+\varepsilon)|\mathcal{CP}_{i+1}|$ holds, or, if $|\mathcal{CP}_i| \geq (1+\varepsilon)|\mathcal{CP}_{i+1}|$ then the starting positions of run i and $i+1$ differ by only a single interval.

Figure 4 Key Properties of the Smooth Histogram Technique.

Property **S1** implies that there are at most $O(\log_{1+\varepsilon}(L))$ active runs of $\mathcal{CP}$ and thus the space of Algorithm 2 is at most a factor $O(\log_{1+\varepsilon}(L))$ larger than the space used by $\mathcal{CP}$.

Property **S2** implies that either consecutive runs differ by at most a $1+\varepsilon$ factor in solution size or are such that their starting times differ by only a single interval.

We now provide a proof that allows us to see that Algorithm 2 is a $(4+2\cdot\varepsilon)$-approximation algorithm for Interval Selection for arbitrary-length intervals. This proof will establish insight into how the analysis of our more involved $(\frac{11}{3}+\varepsilon)$-approximation algorithm is conducted and thus serves as a warm-up. The proof is deferred to the full version of the paper.

▶ **Theorem 6.** *Algorithm 2 is a $(4+2\cdot\varepsilon)$-approximation sliding window algorithm for Interval Selection on arbitrary-length intervals that uses space $\tilde{O}(|OPT|)$, where OPT denotes an optimal solution in the current sliding window.*

4.1.3 Sliding Window Algorithm

We will expand upon the smooth histogram method as described in Algorithm 3. The key idea is to exploit the structure of the regions created by the runs of $\mathcal{CP}$ in the smooth histogram algorithm. Based on these regions, we instantiate additional runs that target areas in which we expect to find many optimal intervals.

We will now proceed and analyse Algorithm 3. To this end, we consider any fixed current sliding window.

First, similar to the analysis of Algorithm 2, we note that if the starting position of the oldest run of $\mathcal{CP}$, denoted $\mathcal{CP}_1$, coincides with the left delimiter of the sliding window then we immediately obtain a 2-approximation (by Property **C2**). Suppose thus that this is not the case. Again, we consider the run $\mathcal{CP}_0$, which is the latest run that has expired and was previously adjacent to $\mathcal{CP}_1$. We also consider the suffix of intervals $S = ABC$, where A are the intervals starting at the starting position of $\mathcal{CP}_0$ and ending before the starting position of $\mathcal{CP}_1$, C are the intervals that occurred after $\mathcal{CP}_0$ and $\mathcal{CP}_1$ became adjacent, and B are the remaining intervals. Let $OPT = OPT(ABC)$, let $OPT_A = OPT \cap A$, and define OPT_B and OPT_C similarly. Since the current sliding window is a subset of ABC, we have that an optimal solution in the current sliding window is of size at most OPT.

Algorithm 3 $(\frac{11}{3}+\varepsilon)$-approximation Algorithm for Interval Selection on arbitrary-length intervals.

Whenever two runs $\mathcal{CP}_i$ and $\mathcal{CP}_{i+1}$ in Algorithm 2 become adjacent (either because of the clean-up operation or because a new run was created), proceed as follows:

Denote by $R_1, \dots, R_\ell$ the regions created by $\mathcal{CP}_i$ thus far.

1. For each region R_i, we initiate a new run of $\mathcal{CP}$, i.e., this run only considers subsequent arriving intervals that lie within R_i.
2. For each pair of consecutive regions R_i, R_{i+1}, we initiate a new run of $\mathcal{CP}$ that considers all subsequent intervals that lie within the merged region $R_i R_{i+1}$ (the region consisting of the left boundary of R_i and the right boundary of R_{i+1}).
3. **Clean-up:** The additional runs established in Steps 1 and 2 are associated with the run $\mathcal{CP}_{i+1}$, and whenever $\mathcal{CP}_{i+1}$ is deleted then all associated runs are also deleted.
4. **Output:** The output is generated from the oldest active run of $\mathcal{CP}$ together with its associated runs as in steps 1 and 2 (see the proofs of Lemmas 8 and 9).

In Algorithm 3, we run the smooth histogram algorithm, Algorithm 2, with respect to a parameter $\beta > 0$ (i.e., replace parameter ε in the listing with β). Then, as proved in Theorem 6, we always have at least a $(4+2\beta)$-approximation at our disposal, i.e.,

$$|\mathcal{CP}_1| = |\mathcal{CP}(BC)| \leq \frac{|OPT|}{4+2\beta} .$$

We define $\varepsilon' \geq 0$ such that

$$|\mathcal{CP}(BC)| = \frac{|OPT|}{4+2\beta-\varepsilon'} . \tag{2}$$

In the following, we will argue that if ε' is close to 0 we can find a better solution using the runs associated with $\mathcal{CP}_1$.

Let $\mathcal{R}$ be the regions created by $\mathcal{CP}_0$ at the moment when $\mathcal{CP}_0$ and $\mathcal{CP}_1$ became adjacent, i.e., the regions created by the run $\mathcal{CP}(AB)$. By Property **C2**, we have $\ell := |\mathcal{R}| = |\mathcal{CP}(AB)|$. For each region $R_i \in \mathcal{R}$, let $x_i = |OPT_C \cap R_i|$, i.e., the number of optimal intervals in C that lie within the region R_i. Furthermore, we define $X := \sum_{i=1}^{\ell} x_i$.

In the next lemma, we prove that, provided ε is small, the quantity X is necessarily large, i.e., there are many optimal intervals in C that lie within the regions R_i. We will later argue that the associated runs with $\mathcal{CP}_1$ can then be used to find many of these.

▶ **Lemma 7.**

$$X \geq \frac{2-\varepsilon'}{1+\beta} \cdot \ell .$$

The proof of Lemma 7 is in the full version of the paper.

Consider now the run $\mathcal{CP}(B)$, which coincides with the run $\mathcal{CP}_1$ until $\mathcal{CP}_0$ and $\mathcal{CP}_1$ became adjacent. Let B_1 be the intervals computed by this run that do not intersect the boundaries of $\mathcal{R}$ and let B_2 be the intervals computed by this run that intersect the boundaries of $\mathcal{R}$. Then, since $|B_1| + |B_2| = |\mathcal{CP}(B)|$, we have that either $|B_1| \geq \frac{1}{3}|\mathcal{CP}(B)|$ or $|B_2| \geq \frac{2}{3}|\mathcal{CP}(B)|$. We treat both cases separately in Lemmas 8 and 9:

▶ **Lemma 8.** *Suppose that $|B_1| \geq \frac{1}{3}|\mathcal{CP}(B)|$. Then, using the associated runs of $\mathcal{CP}_1$, we can output a solution of size at least $\frac{7-3\varepsilon'}{6(1+\beta)}\ell$.*

Proof. We call a region R_i *good* if it contains an interval from B_1. We output the solution obtained from the runs of $\mathcal{CP}$ on R_i, for all i, and if such a run on a good region leads to no intervals (i.e. $x_i = 0$), then we output the interval from B_1 instead. Recall that $\mathcal{CP}$ outputs a solution of size $\frac{x_i+1}{2}$ if $x_i \neq 0$ (Property **C2**), and we stress here that the additive $+1$ is key for our analysis. We thus obtain a solution of size at least:

$$S = \sum_{R_i \text{ good}} \max\left\{\frac{x_i+1}{2}, 1\right\} + \sum_{R_i \text{ bad}, x_i \neq 0} \frac{x_i+1}{2} .$$

Recall that $\sum_{i=1}^{\ell} x_i = X \geq \frac{2-\varepsilon'}{1+\beta} \cdot \ell$.
Hence,

$$\begin{aligned} S &\geq \sum_{R_i \text{ good}} \frac{x_i+1}{2} + \sum_{R_i \text{ bad}, x_i \neq 0} \frac{x_i+1}{2} \\ &\geq \frac{X + |B_1|}{2} && |B_1| \text{ is the number of good regions} \\ &\geq \frac{2-\varepsilon'}{2(1+\beta)}\ell + \frac{1}{6}|\mathcal{CP}(B)| && \text{By Lemma 7} \\ &\geq \frac{2-\varepsilon'}{2(1+\beta)}\ell + \frac{\ell}{6(1+\beta)} && |\mathcal{CP}(B)| \geq \frac{|\mathcal{CP}(AB)|}{1+\beta} = \frac{\ell}{1+\beta} \text{ by Prop. } \mathbf{S2} \\ &\geq \frac{7-3\varepsilon'}{6(1+\beta)}\ell . \end{aligned}$$ ◀

Due to space restrictions, the proof of the following lemma is postponed to the full version of the paper. The proof follows the same idea as the proof of Lemma 8.

▶ **Lemma 9.** *Suppose that $|B_2| \geq \frac{2}{3}|\mathcal{CP}(B)|$. Then, using the associated runs of $\mathcal{CP}_1$, we can output a solution of size at least $\frac{7-3\varepsilon'}{6(1+\beta)}\ell$.*

▶ **Theorem 10.** *For any constant $\delta > 0$, Algorithm 3 is a $(11/3 + \delta)$-approximation sliding window algorithm for Interval Selection on arbitrary-length intervals that uses space $\tilde{\mathrm{O}}(|OPT|)$.*

Proof. The naive smooth histogram method gives us a solution of size

$$|\mathcal{CP}(BC)| \geq |\mathcal{CP}(B)| \geq \frac{|\mathcal{CP}(AB)|}{1+\beta} \geq \frac{\ell}{1+\beta} ,$$

where we used the monotonicity of $\mathcal{CP}$ (Lemma 5) and Property **S2**. Using the associated runs, by Lemmas 8 and 9, we get a solution of size at least

$$\frac{7-3\varepsilon'}{6(1+\beta)}\ell .$$

Since we can output the larger of the two solutions, in the worst case both solutions have the same value, i.e., when:

$$\frac{\ell}{1+\beta} = \frac{7-3\varepsilon'}{6(1+\beta)}\ell ,$$

which implies $\varepsilon' = \frac{1}{3}$. The approximation factor thus is for any $\delta > 0$:

$$4 + 2 \cdot \beta - \varepsilon' + \delta = 11/3 + 2 \cdot \beta + \delta .$$

Choosing $\beta = \frac{1}{2}\delta$, and rescaling δ to $\frac{1}{2}\delta$ gives the result.

As a consequence of Property **S1**, as previously established, the smooth histogram algorithm uses $\tilde{O}(|OPT|)$ space. It remains to argue that the runs created in Steps 1 and 2 of Algorithm 3 only increase the space requirements by a constant times $|OPT|$.

Indeed, for a fixed instance $\mathcal{CP}_i$, all the runs created by Step 1 are pairwise disjoint (they do not store common intervals) so their cumulative space is $O(|OPT|)$ as we assumed the memory required to store an interval is $O(1)$. Similarly, for the runs created by Step 2, an interval appears in at most two such runs. So, the cumulative space is again $O(|OPT|)$. Therefore, the total number of intervals stored in the associated runs is at most $O(|OPT|)$, completing the proof. ◀

The proof of the approximation factor of the algorithm is shown to be tight in the full version of the paper, meaning that the algorithm does not beat the approximation factor of $\frac{11}{3}$.

4.2 Space Lower Bound

We now give our space lower bound for sliding window algorithms for Interval Selection on arbitrary-length intervals. Our result is established by a reduction to the three-party communication problem Chain_3.

▶ **Theorem 11.** *Let $\varepsilon > 0$ be any small constant. Then, any algorithm in the sliding window model that computes a $(2.5 - \varepsilon)$-approximate solution to Interval Selection on arbitrary-length intervals requires a memory of size $\Omega(L)$.*

Proof. Let $\mathcal{A}$ be a sliding window algorithm with approximation factor $2.5 - \varepsilon$, for some $\varepsilon > 0$, as in the statement of the theorem, and let $n = \frac{L-2}{3}$, where L is the window length. We will argue how $\mathsf{Chain}_3(n)$ can be solved with the help of $\mathcal{A}$.

To this end, denote the three parties in the communication problem $\mathsf{Chain}_3(n)$ by Alice, Bob, and Charlie. Let $X_1 \in \{0,1\}^n$ be Alice's input, let $X_2 \in \{0,1\}^n$ and $J_1 \in [n]$ be Bob's input, and let $J_2 \in [n]$ be Charlie's input. The players proceed as follows:

- **Alice:** For every $i \in [n]$, Alice feeds the following intervals into $\mathcal{A}$ in the order $I_1(1), I_2(1), I_1(2), I_2(2), \dots, I_1(n), I_2(n)$:

$$I_1(i) = \begin{cases} \left[\frac{i}{3n}, 1 + \frac{i}{3n}\right], & \text{if } X_1[i] = 1 , \\ [-10 - i, 10 + i], & \text{if } X_1[i] = 0 . \end{cases} \qquad I_2(i) = \begin{cases} \left[2 - \frac{i}{3n}, 3 - \frac{i}{3n}\right], & \text{if } X_1[i] = 1 , \\ [-11 - i, 11 + i], & \text{if } X_1[i] = 0 . \end{cases}$$

  We observe that, for every $i \in [n]$, when $X_1[i] = 1$, the intervals $I_1(i)$ and $I_2(i)$ are disjoint. Alice sends the memory state of $\mathcal{A}$ to Bob.

- **Bob:** For every $i \in [n + 2(J_1 - 1)]$, Bob feeds the following interval into $\mathcal{A}$:

$$I_3(i) = \begin{cases} \left[1 + \frac{J_1}{3n} + \frac{1}{6n} + \frac{i-1}{6n^2}, 2 - \frac{J_1}{3n} - \frac{1}{6n} + \frac{i-1}{6n^2}\right], & \text{if } i \leq n \text{ and } X_2[i] = 1 , \\ [-10 - i, 11 + i], & \text{otherwise} . \end{cases}$$

  Let $k \in [n]$. Notice that, for every $j \in [n + 2(J_1 - 1)]$, when $X_1[k] = X_2[j] = 1$, we have that $I_3(j)$ is disjoint with both $I_1(k)$ and $I_2(k)$ if and only if $j \leq J_1$. Otherwise, $I_3(j)$ intersects with both $I_1(k)$ and $I_2(k)$.

  Bob sends the memory state of $\mathcal{A}$ and J_2 to Charlie.

- **Charlie:** We denote the interval boundaries of $I_3(i)$ by $a_{I_3(i)}$ and $b_{I_3(i)}$, i.e., $I_3(i) = [a_{I_3(i)}, b_{I_3(i)}]$. Charlie feeds the following two intervals into $\mathcal{A}$:

$$I_{J_2} = \left[\frac{2a_{I_3(J_2-1)} + a_{I_3(J_2)}}{3}, \frac{a_{I_3(J_2-1)} + 2a_{I_3(J_2)}}{3}\right] \text{ , and}$$
$$I'_{J_2} = \left[\frac{2b_{I_3(J_2-1)} + b_{I_3(J_2)}}{3}, \frac{b_{I_3(J_2-1)} + 2b_{I_3(J_2)}}{3}\right] .$$

Notice that I_{J_2} intersects all intervals of $I_3(i)$, for all $i < J_2$, while I'_{J_2} intersects all intervals of $I_3(i)$, for all $i > J_2$.
Using $\mathcal{A}$, Charlie computes the largest independent set OPT of

$$\mathcal{I} = \{I_1(k) | J_1 \le k \le n\} \cup \{I_2(k) | J_1 \le k \le n\} \cup \{I_3(k) | 1 \le k \le n+2(J_1-1)\} \cup \{I_{J_2}, I'_{J_2}\} ,$$

which is possible since $\mathcal{A}$ is a sliding window algorithm and thus able to solve the situation when the intervals $\cup_{1 \le k < J_1} (I_1(k) \cup I_2(k))$ have expired.

This construction is illustrated by Figure 5.

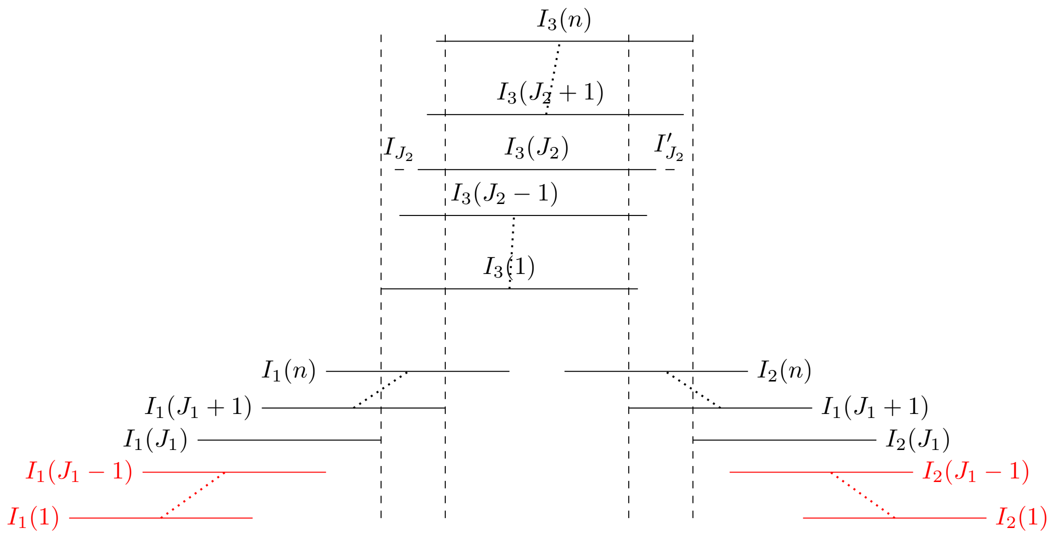

Figure 5 This figure illustrates the intervals created by Alice, Bob and Charlie in the proof of Theorem 11 for an instance of $\mathsf{Chain}_3(n)$ where $n = \frac{L-2}{3}$ with $X_1[J_1] = X_2[J_2] = 1$. The red intervals in the figure ($I_1(1), I_1(J_1-1)$,$I_2(1)$,$I_2(J_1-1)$) correspond to expired intervals. The optimal solution is $\{I_1(J_1), I_2(J_1), I_{J_2}, I'_{J_2}, I_3(J_2)\}$ of size 5 . Otherwise, if $X_1[J_1] = X_2[J_2] = 0$, then the optimal solution would have been of size 2. All intervals $I_3(i)$ are disjoint from $I_1(J_1)$ and $I_2(J_2)$. However, they intersect with $I_1(J_1+1)$ and $I_2(J_2+1)$ as emphasized by the vertical dashed lines. Intervals $I_3(i)$ for $n + 2(J_1 - 1) \ge i > n$ have been omitted as they do not impact the optimal solution and their only role is to advance the sliding window.

.

The total number of intervals added by the three players is $3n+2+2(J_1-1) = L+2(J_1-1)$. So, after Charlie's execution $\mathcal{A}$, the incumbent window indeed consists of $\mathcal{I}$.

We will argue now that if $X_1[J_1] = X_2[J_2] = 1$ then the optimal solution size is 5, while if $X_1[J_1] = X_2[J_2] = 0$ then the optimal solution size is 2.

Suppose thus that $X_1[J_1] = X_2[J_2] = 1$. Then it is not hard to see that the unique optimal solution is $\{I_1(J_1), I_2(J_1), I_3(J_2), I_{J_2}, I'_{J_2}\}$ of size 5.

Next, suppose that $X_1[J_1] = X_2[J_2] = 0$. Notice first that, in this case, $I_1(J_1), I_2(J_1), I_3(J_2)$ intersect with every other interval in the input, so they can only belong to independent sets of size at most 1.

Also, we have that any interval $I_1(i)$ with $i > J_1$ would block all the intervals $I_3(j)$ for $1 \leq j \leq n + 2(J_1 - 1)$ and I_{J_2}. So, an interval from $I_1(i)$ with $i > J_1$ can be included in an optimal set of size at most 2 (either $\{I_1(i), I'_{J_2}\}$ or $\{I_1(i), I_2(j)\}$ for some $j > J_1$). Similarly, $I_2(i)$ with $i > J_2$ can be included in an optimal set of size at most 2. Furthermore, we can construct from Bob and Charlie's input a solution of size at most 2 (similar to the $\frac{3}{2} - \varepsilon$ lower bound construction of [11]). The size of an optimal solution is thus in this case 2.

Recall that $\mathcal{A}$ has an approximation factor of $2.5 - \varepsilon$. Hence, if $X_1[J_1] = X_2[J_2] = 1$ then $\mathcal{A}$ reports a solution of size at least 3, thereby distinguishing it from the case when $X_1[J_1] = X_2[J_2] = 0$, which yields an optimal size of 2.

Since, by Theorem 2, $\mathsf{Chain}_3(n)$ requires a message of size $\Omega(n)$, and since the protocol solely consists of forwarding the memory state of $\mathcal{A}$, we conclude that $\mathcal{A}$ requires a memory of size $\Omega(n) = \Omega(L)$, which completes the proof. ◀

5 Conclusion

In this paper, we initiated the study of the Interval Selection problem in the sliding window model of computation. We gave algorithms and lower bounds for both unit-length and arbitrary-length intervals. In the unit-length case, we gave a 2-approximation algorithm that uses space $O(|OPT|)$, and we showed that this is best possible in that any $(2 - \varepsilon)$-approximation algorithm requires space $\Omega(L)$. In the arbitrary-length case, we gave a $(\frac{11}{3} + \varepsilon)$-approximation algorithm that uses space $\tilde{O}(|OPT|)$, and we showed that any $(\frac{5}{2} - \varepsilon)$-approximation algorithm requires space $\Omega(L)$. Contrasted with results known from the one-pass streaming setting, our result implies that Interval Selection in both the unit-length and the arbitrary-length cases is harder to solve in the sliding window setting than in the one-pass streaming setting.

We conclude with two open questions.

First, the approximation guarantees of our algorithm for arbitrary-length intervals and our respective lower bound do not match. Can we close this gap?

Second, the sliding window model has received significantly less attention for the study of graph problems than the traditional one-pass streaming setting. While from a theoretical perspective, the sliding window model is less clean than the one-pass streaming model, as discussed in the introduction, it is, however, the more suitable model for many applications. We are particularly interested in understanding the differences between the two models. For example, which graph problems can be solved equally well in the sliding window model as in the one-pass streaming setting, and which problems are significantly harder to solve?

References

1 Cezar-Mihail Alexandru, Pavel Dvorák, Christian Konrad, and Kheeran K. Naidu. Improved weighted matching in the sliding window model. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, March 7-9, 2023, Hamburg, Germany*, volume 254 of *LIPIcs*, pages 6:1–6:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.STACS.2023.6.

2 Cezar-Mihail Alexandru and Christian Konrad. Interval selection in sliding windows. *ArXiv*, abs/2405.09338, 2024. URL: https://api.semanticscholar.org/CorpusID:269772749.

3 Ainesh Bakshi, Nadiia Chepurko, and David P. Woodruff. Weighted maximum independent set of geometric objects in turnstile streams. In *International Workshop and International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2019. URL: `https://api.semanticscholar.org/CorpusID:67856291`.

4 Leyla Biabani, Mark de Berg, and Morteza Monemizadeh. Maximum-weight matching in sliding windows and beyond. In *International Symposium on Algorithms and Computation*, 2021. URL: `https://api.semanticscholar.org/CorpusID:245276580`.

5 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 283–293. IEEE Computer Society, 2007. `doi:10.1109/FOCS.2007.55`.

6 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`.

7 Graham Cormode, Jacques Dark, and Christian Konrad. Independent sets in vertex-arrival streams. *ArXiv*, abs/1807.08331, 2018. URL: `https://api.semanticscholar.org/CorpusID:49907556`.

8 Michael S. Crouch, Andrew McGregor, and Daniel M. Stubbs. Dynamic graphs in the sliding-window model. In Hans L. Bodlaender and Giuseppe F. Italiano, editors, *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013. `doi:10.1007/978-3-642-40450-4_29`.

9 Jacques Dark, Adithya Diddapur, and Christian Konrad. Interval selection in data streams: Weighted intervals and the insertion-deletion setting. In *Foundations of Software Technology and Theoretical Computer Science*, 2023. URL: `https://api.semanticscholar.org/CorpusID:266192962`.

10 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows: (extended abstract). In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 635–644, USA, 2002. Society for Industrial and Applied Mathematics.

11 Yuval Emek, Magnús M. Halldórsson, and Adi Rosén. Space-constrained interval selection. *ACM Trans. Algorithms*, 12(4):51:1–51:32, 2016. `doi:10.1145/2886102`.

12 Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 1363–1374, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3357713.3384286`.

13 T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory Comput.*, 4:129–135, 2008. URL: `https://api.semanticscholar.org/CorpusID:15825208`.

14 Robert Krauthgamer and David Reitblat. Almost-smooth histograms and sliding-window graph algorithms. *Algorithmica*, 84(10):2926–2953, 2022. `doi:10.1007/s00453-022-00988-y`.

15 Eyal Kushilevitz and Noam Nisan. *Communication complexity.* Cambridge University Press, 1997.

16 Ami Paz and Gregory Schwartzman. A $(2+\epsilon)$-approximation for maximum weight matching in the semi-streaming model. *ACM Trans. Algorithms*, 15(2):18:1–18:15, 2019. `doi:10.1145/3274668`.

17 Sai Krishna Chaitanya Nalam Venkata Subrahmanya. Vertex cover in the sliding window model. Master's thesis, Rutgers, The State University of New Jersey, 2021.

18 Janani Sundaresan. Optimal communication complexity of chained index. *ArXiv*, abs/2404.07026, 2024. URL: `https://api.semanticscholar.org/CorpusID:269033021`.