Improved Weighted Matching in the Sliding Window Model

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

Pavel Dvořák
Tata Institute of Fundamental Research, Mumbai, India
Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic

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

Kheeran K. Naidu
Department of Computer Science, University of Bristol, UK

Abstract

We consider the Maximum-weight Matching (MWM) problem in the streaming sliding window model of computation. In this model, the input consists of a sequence of weighted edges on a given vertex set V of size n. The objective is to maintain an approximation of a maximum-weight matching in the graph spanned by the L most recent edges, for some integer L, using as little space as possible. Prior to our work, the state-of-the-art results were a $(3.5+\varepsilon)$-approximation algorithm for MWM by Biabani et al. [ISAAC'21] and a $(3+\varepsilon)$-approximation for (unweighted) Maximum Matching (MM) by Crouch et al. [ESA'13]. Both algorithms use space $\tilde{O}(n)$.

We give the following results:

1. We give a $(2+\varepsilon)$-approximation algorithm for MWM with space $\tilde{O}(\sqrt{nL})$. Under the reasonable assumption that the graphs spanned by the edges in each sliding window are simple, our algorithm uses space $\tilde{O}(n\sqrt{n})$.
2. In the $\tilde{O}(n)$ space regime, we give a $(3+\varepsilon)$-approximation algorithm for MWM, thereby closing the gap between the best-known approximation ratio for MWM and MM.

Similar to Biabani et al.'s MWM algorithm, both our algorithms execute multiple instances of the $(2+\varepsilon)$-approximation $\tilde{O}(n)$-space streaming algorithm for MWM by Paz and Schwartzman [SODA'17] on different portions of the stream. Our improvements are obtained by selecting these substreams differently. Furthermore, our $(2+\varepsilon)$-approximation algorithm runs the Paz-Schwartzman algorithm in *reverse direction* over some parts of the stream, and in *forward direction* over other parts, which allows for an improved approximation guarantee at the cost of increased space requirements.

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, Maximum-weight matching

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.6

Funding *Cezar-Mihail Alexandru*: Supported by EPSRC DTP studentship EP/T517872/1.
Pavel Dvořák: Supported by EPSRC New Investigator Award EP/V010611/1 and by Czech Science Foundation GAČR grant #22-14872O.
Christian Konrad: Supported by EPSRC New Investigator Award EP/V010611/1.
Kheeran K. Naidu: Supported by EPSRC DTP studentship EP/T517872/1.

1 Introduction

The *data streaming model* is a well-established computational model that provides a framework for studying massive data set algorithms. The defining features of the model are restricted access to the input data and sublinear space. A data streaming algorithm processes its input sequentially in a single pass while maintaining only a small summary of the input in memory.

In this paper, we study the Maximum-weight Matching (MWM) problem in the *(streaming) sliding window* model. In this variant of the streaming model, the input consists of a potentially infinite sequence $e_1, e_2, \dots$ of weighted edges on an underlying vertex set V of size n. The objective is to maintain a matching of large weight in the graph spanned by the L most recent edges, for some integer L, using as little space as possible. In more detail, after having processed the current edge e_i, for every i, the objective is to report an approximation of a maximum-weight matching in the graph spanned by the current sliding window $E_i := \{e_j : \max\{i - L + 1, 1\} \leq j \leq i\}$. Many of the known sliding window algorithms for graph problems operate within *semi-streaming space* [12], i.e., within space $O(n \operatorname{polylog} n) = \tilde{O}(n)$. In this paper, we will work both with the semi-streaming space regime and also consider algorithms that use more space.

While sliding window algorithms have been studied for two decades [10], sliding window algorithms for graph problems were first considered by Crouch et al. [7] in 2013. Amongst other results, they showed that there is a $(3 + \varepsilon)$-approximation semi-streaming sliding window algorithm for unweighted Maximum Matching (MM) and a 9.027-approximation semi-streaming sliding window algorithm for MWM. While no improved results are known for MM, Crouch and Stubbs [8] subsequently improved upon the result for MWM and gave a $(6 + \varepsilon)$-approximation semi-streaming algorithm, and, very recently, Biabani et al. [4] gave a $(3.5 + \varepsilon)$-approximation in the semi-streaming space regime. The state-of-the-art results for MM and MWM in the semi-streaming sliding window model therefore do not yet line up.

Our Results

In this paper, we give two sliding window algorithms for MWM that both improve upon the state-of-the-art approximation guarantee of $3.5 + \varepsilon$.

As our first result, we give a substantial improvement and obtain an approximation factor of $2 + \varepsilon$ at the expense of increased space requirements:

▶ **Theorem 1** (simplified version)**.** *There is a deterministic $(2 + \varepsilon)$-approximation sliding window algorithm for Maximum-weight Matching that uses space $\tilde{O}(\sqrt{nL})$ (with dependency on ε and logarithms suppressed), for any $\varepsilon > 0$.*

Some remarks are in order. First, we observe that going beyond the approximation factor of 2, even using space $O(n^{1.999})$, would answer a long-standing open problem in graph streaming research, namely, whether there is a one-pass $(2 - \Omega(1))$-approximation streaming algorithm for MM with space $O(n^{1.999})$. We thus cannot expect to obtain further improvements in the approximation guarantee with current techniques. Second, the space requirements of our algorithm depend on the sliding window length L. This is in contrast to the $(3.5 + \varepsilon)$-approximation algorithm by Biabani et al. [4] and our second algorithm described below. Under the natural assumption that the graphs described by all sliding windows are simple, we have $L = O(n^2)$, which yields a space bound of $\tilde{O}(n\sqrt{n})$.

As our second result, we close the gap between MM and MWM in the semi-streaming space regime. To this end, we give a semi-streaming sliding window algorithm for MWM that matches the approximation guarantee of the best-known sliding window algorithm for MM.

Table 1 Known sliding window algorithms for MM and MWM.

	Approximation Factor	Space	Reference
MM	$3+\varepsilon$	$\tilde{O}(n)$	Crouch et al. [7]
MWM	9.027	$\tilde{O}(n)$	Crouch et al. [7]
	$6+\varepsilon$	$\tilde{O}(n)$	Crouch and Stubbs [8]
	$3.5+\varepsilon$	$\tilde{O}(n)$	Biabani et al. [4]
	$3+\varepsilon$	$\tilde{O}(n)$	This paper (Theorem 2)
	$2+\varepsilon$	$\tilde{O}(\sqrt{nL})$	This paper (Theorem 1)

▶ **Theorem 2** (simplified version). *There is a deterministic semi-streaming sliding window algorithm for Maximum-weight Matching with approximation factor $3+\varepsilon$, for any $\varepsilon > 0$.*

Table 1 summarizes all results known for MM and MWM in the sliding window model.

Techniques

Both our algorithms make use of the one-pass $(2+\varepsilon)$-approximation streaming algorithm for MWM by Paz and Schwartzman [19]. Since we make use of the inner workings of the algorithm, we will discuss this algorithm first.

Paz and Schwartzman's MWM Algorithm. Paz and Schwartzman's original algorithm [19] uses space $O\big(\frac{1}{\varepsilon} \cdot n \log^2 n\big)$ and is based on the *local ratio technique* (see [3] for further details on this technique). Ghaffari and Wajc [13] gave a simplified version and improved the space complexity to the (optimal in n) bound $O\big(\frac{\log(1/\varepsilon)}{\varepsilon} \cdot n \log n\big)$.

The Paz and Schwartzman algorithm with Ghaffari and Wajc's improvement works as follows. For every vertex $v \in V$, it maintains a potential $\varphi(v)$ that is initialized with 0, and uses a stack data structure Stack. When an edge $e = \{u, v\}$ arrives in the stream, e is pushed onto Stack if its weight $w(e)$ exceeds the sum of the potentials of its incident vertices by a factor of at least $(1+\varepsilon)$, i.e., $w(e) \geq (1+\varepsilon)(\varphi(u)+\varphi(v))$. The discrepancy between $w(e)$ and $\varphi(u)+\varphi(v)$ is denoted the *reduced weight* of e and is abbreviated by $w'(e) := w(e) - (\varphi(u) + \varphi(v))$. Then, the potentials $\varphi(u)$ and $\varphi(v)$ are updated as $\varphi(u) = \varphi(u) + w'(e)$ and $\varphi(v) = \varphi(v) + w'(e)$. Last, if either u or v is adjacent to at least $\frac{3\log(1/\varepsilon)}{\varepsilon} + 1$ edges in Stack then the oldest (and thus lightest) edge incident to the vertex is removed from Stack, thereby limiting the number of edges on Stack. After having processed all the edges in the stream, the output matching $\hat{M}$ is computed in a post-processing step. The edges in Stack are popped one by one and greedily inserted into $\hat{M}$ if possible, i.e., as long as $\hat{M}$ remains a matching. We denote the Paz and Schwartzman algorithm by $\mathcal{ALG}^{\varepsilon}_{PS}$. See Section 2 for a formal description.

$(2+\varepsilon)$-approximation Algorithm with Space $\tilde{O}(\sqrt{nL})$. Our $(2+\varepsilon)$-approximation algorithm processes the input in blocks of size $s = \tilde{\Theta}(\sqrt{nL})$. Consider one such block B_j, i.e., a substream of s consecutive edges. The key idea of our algorithm is to run multiple instances of the Paz-Schwartzman algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$ on B_j, however, in *reverse direction*. We start with a single instance $\mathcal{I}_1$. At various moments during the processing of B_j, we fork the current instance $\mathcal{I}_i$ to obtain an additional instance $\mathcal{I}_{i+1}$, and then only continue to feed further edges into $\mathcal{I}_{i+1}$; thus, in any moment of processing the block B_j, we feed the edge to only one instance of the Paz-Schwartzman algorithm. The fork happens when the sum of reduced weights $W'(\mathcal{I}_i)$ of the edges on Stack in $\mathcal{I}_i$ exceeds the sum of reduced weights of the previous instance by a $1+\varepsilon$ factor, i.e., $W'(\mathcal{I}_i) > (1+\varepsilon) \cdot W'(\mathcal{I}_{i-1})$. As a result, we obtain instances of Paz-Schwartzman that processed suffixes of different lengths of block B_j (remember that we process B_j in the reverse direction), and adjacent instances have a similar sum of reduced weights (up to a $1+\varepsilon$ factor). As we will point out in Section 2, the sum of reduced weights in an instance of Paz-Schwartzman is strongly related to the weight of a maximum-weight matching among the edges observed thus far, and we heavily exploit this property in our proofs.

In each block B_j, besides preparing the instances of Paz-Schwartzman as described above, we also feed the edges of B_j (in the forward direction) into those instances of Paz-Schwartzman that were prepared during previous blocks $B_{j'}$, with $j' < j$, and that are still *alive*, i.e., have only been fed edges from the current sliding window. As such, each instance of Paz-Schwartzman is executed on a portion of the stream in the reverse direction, followed by all the subsequent edges from more recent blocks in the forward direction until the current edge. The output produced when processing the current edge is the output of the oldest alive instance of Paz-Schwartzman.

Consider two adjacent instances $\mathcal{I}_i$ and $\mathcal{I}_{i+1}$ of Paz-Schwartzman prepared in the same block, where $\mathcal{I}_i$ has processed only a subset of the edges of $\mathcal{I}_{i+1}$ and their sums of reduced weights W' are such that $W'(\mathcal{I}_{i+1}) \approx (1+\varepsilon)W'(\mathcal{I}_i)$. The key benefit of executing Paz-Schwartzman in the reverse direction as opposed to forward is that the edges processed by $\mathcal{I}_{i+1}$ but not by $\mathcal{I}_i$ contribute to the sum of reduced weights only with an ε-fraction of $W'(\mathcal{I}_i)$ (since $W'(\mathcal{I}_{i+1}) - W'(\mathcal{I}_i) \approx \varepsilon W'(\mathcal{I}_i)$). When $\mathcal{I}_i$ is the oldest alive instance and thus constitutes the output of our algorithm, we only *miss* an ε-fraction in terms of reduced weights of the edges in the sliding window that $\mathcal{I}_i$ has not considered. We remark that this property could not be established if we run Paz-Schwartzman in the forward direction. This property together with the fact that the sum of reduced weights is related to the weight of a maximum-weight matching allows us to establish the approximation factor of our algorithm.

Since only the L most recent edges are relevant, our algorithm considers at most $\frac{L}{s} = \tilde{\Theta}(\sqrt{L/n})$ blocks simultaneously. Each block consists of $\tilde{O}(1)$ instances of Paz-Schwartzman. Since each of these instances requires space $\tilde{O}(n)$, we obtain the final space bound of $\tilde{O}(n) \cdot \frac{L}{s} = \tilde{O}(\sqrt{nL})$.

$(3+\varepsilon)$-approximation Semi-streaming Algorithm. Our $(3+\varepsilon)$-approximation algorithm follows similar arguments as the $(3.5+\varepsilon)$-approximation algorithm by Biabani et al. [4]. We will therefore first explain the techniques behind Biabani et al.'s algorithm and then discuss our new ideas which yield the improved approximation guarantee.

Biabani et al.'s algorithm combines the *smooth histogram technique* for sliding window algorithms by Braverman and Ostrovsky [6] with the Paz and Schwartzman algorithm. Braverman and Ostrovsky showed that if a function f fulfills certain smoothness criteria[1] then a sliding window algorithm for approximating f can be obtained from a traditional (non-sliding window) streaming algorithm for f at the expense of only a logarithmic increase in the space requirements (as long as the approximation factor of the streaming algorithm is constant), and a slight increase in the approximation factor. In the context of MWM, the smoothness criteria are captured by Biabani et al. [4] via the notion of *lookahead* algorithm.

[1] Informally, a function $f : 2^X \to \mathbb{R}$ is considered to be *smooth* if it satisfies the following: If $f(A)$ is close to $f(B)$ for $A, B \subseteq X$, for a suitable notion of closeness, then the values $f(A \cup C)$ and $f(B \cup C)$ are close for all $C \subseteq X$.

▶ **Definition 3** ((f, α, β)-lookahead algorithm [4]). *Let $\beta \in (0, 1)$ and $\alpha > 0$ be real numbers. Let X be a ground set, S a stream of items of X, and let $f : 2^X \rightarrow \mathbb{R}^+$ be a non-decreasing function. We say that a streaming algorithm $\mathcal{ALG}$ is a (f, α, β)-*lookahead algorithm *if, for any partitioning of S into three substreams A, B, C with $\mathcal{ALG}(B) \geq (1 - \beta) \cdot \mathcal{ALG}(AB)$, the following holds: $f(ABC) \leq \alpha \cdot \mathcal{ALG}(BC)$.*

In this paper, the stream AB denotes the concatenation of streams A and B (as it is used in the previous definition). We observe that the previous definition holds for real-valued non-decreasing functions. In the context of MWM, the weight of a maximum-weight matching rather than the matching itself fulfills these conditions. We will therefore consider the problem of determining the weight of a maximum-weight matching instead, and, in order to be able to output an actual matching as required in MWM, we will rely on the fact that the underlying algorithm which we will consider also maintains the actual matching itself. Furthermore, we will write MWM(S) to denote the weight of a maximum-weight matching in stream S.

Biabani et al. [4] showed that if there is a (MWM, α, β)-lookahead algorithm that uses space s then there exists a sliding-window algorithm with approximation ratio α and space $O\big(\frac{1}{\beta} \cdot s \log \sigma\big)$, where $\sigma = \frac{n}{2} \cdot w_{\max}/w_{\min}$ and $w_{\max}$ and $w_{\min}$ are the maximum and minimum weights of an edge in the input stream, respectively. Observe that, under the usual assumption that $w_{\max}/w_{\min}$ is polynomial in n, we have $\log \sigma = O(\log n)$.

The main part of their analysis is to show that a monotonic version of the Paz and Schwartzman algorithm, denoted $\mathcal{ALG}_{mon}$, constitutes a (MWM, $(3.5 + \varepsilon), \beta$)-lookahead algorithm, for small values of ε and $\beta \leq \varepsilon/9$. Combined, this yields a $(3.5 + \varepsilon)$-approximation semi-streaming sliding window algorithm for MWM.

We first note (see Appendix A for details) that the analysis of Biabani et al. is best possible in that the Paz and Schwartzman algorithm and its monotonic version are no better than (MWM, $3.5, \beta$)-lookahead algorithms. The smooth histogram technique applied to lookahead algorithms as defined in Definition 3 thus cannot give an improved approximation guarantee when Paz and Schwartzman's algorithm is used as the underlying algorithm.

To illustrate our improvement, we first provide insight into the structure of Biabani et al.'s analysis. In order to prove that $\mathcal{ALG}_{mon}$ is a (MWM, $3.5 + \varepsilon, \beta$)-lookahead algorithm, Biabani et al. relate MWM(ABC) to the output of $\mathcal{ALG}_{mon}$ on various substreams of ABC:

$$\begin{aligned}\mathsf{MWM}(ABC) \quad &\leq \quad 2(1+\varepsilon) \cdot \big(\mathcal{ALG}_{mon}(AB) + \mathcal{ALG}_{mon}(BC)\big) \\ &\quad - \frac{1}{2(1+\varepsilon)} \cdot \mathcal{ALG}_{mon}(B) \ . \end{aligned} \tag{1}$$

They subsequently use the smoothness assumption from Definition 3 and a monotonicity property of $\mathcal{ALG}_{mon}$ to relate $\mathcal{ALG}_{mon}(AB)$ and $\mathcal{ALG}_{mon}(B)$ to $\mathcal{ALG}_{mon}(BC)$. This ultimately yields the desired bound MWM(ABC) $\leq (3.5 + \varepsilon) \cdot \mathcal{ALG}_{mon}(BC)$.

To obtain our improvement, we observe that a similar inequality to Inequality 1 can be obtained by considering *sums of reduced weights* of the respective runs of $\mathcal{ALG}_{mon}$ instead of the weights of the output matchings of $\mathcal{ALG}_{mon}$ on the different substreams. This idea is motivated by the fact that the sum of reduced weights is a lower bound on the weight of the matching produced by the algorithm, which can therefore give a more precise analysis. However, when departing from such an inequality involving sums of reduced weights, we unfortunately cannot immediately complete our analysis since, unlike when considering the outputs of $\mathcal{ALG}_{mon}$ directly, we do not have a sufficient smoothness property regarding sums of reduced weights at our disposal that would allow us to bound these quantities.

Our key idea is as follows. To establish the necessary smoothness properties, we employ the smooth histogram technique directly on sums of reduced weights rather than on the size of the output matching itself. To be consistent with the literature and to illustrate the increment over Biabani et al.'s work, we encapsulate this idea via an alternative definition of lookahead algorithms, denoted *refined lookahead algorithms* (see Definition 9 for details), which enables us to incorporate the required smoothness property of sums of reduced weights into the definition. We then prove that, similar to lookahead algorithms, refined lookahead algorithms can still be turned into sliding window algorithms with a similar small increase in the space complexity. Last, we finish our argument by proving that $\mathcal{ALG}^{\varepsilon}_{PS}$ is a refined lookahead algorithm with an approximation factor of $3 + \varepsilon$, which establishes our result.

Further Related Work

The sliding window model can be regarded as a streaming insertion-deletion model with highly structured deletions since, for each incoming edge, the oldest edge in the current window is deleted. Interestingly, the complexities of MM and MWM in the sliding window model are much closer to those in the insertion-only model, where no deletions are allowed, as opposed to the insertion-deletion model, where arbitrary deletions are allowed. In the insertion-only model, the currently best one-pass algorithm known for MM is the Greedy matching algorithm, which produces a 2-approximation and uses semi-streaming space $\tilde{O}(n)$. It is known that computing a $(1 + \ln 2)$-approximation requires strictly more space than $\tilde{O}(n)$ [15], see also the previous lower bounds [14, 16]. It remains a key open problem to close this gap. Regarding MWM, a series of works [12, 18, 21, 11, 8, 19, 13] culminated in the Paz and Schwartzman algorithm, which closes the gap between MWM and MM from an algorithmic perspective in the insertion-only model. In the insertion-deletion model, where arbitrary previously inserted edges can be deleted again, it is known that space $\Theta(n^2/\alpha^3)$ is necessary and sufficient for computing an α-approximation to MM, see [2] for the algorithm and [9] for a matching lower bound (see also the previous works [17, 1]). MWM reduces easily to MM in the insertion-deletion model, by, for example, grouping edges of similar weights into groups and running the MM algorithm a logarithmic number of times in parallel at the expense of only a marginal increase in the approximation factor.

The sliding window model is inspired by the problem of inferring statistics of data occurring within a certain time frame over a continuous stream of data (e.g., maintaining the number of distinct users who have accessed a social media page in the last 24 hours). The model was introduced by Datar et al. [10], and Crouch et al. [7] were the first to consider graph problems in the sliding window model. Among others, they showed that testing Connectivity and Bipartiteness, and constructing $(1 + \varepsilon)$-sparsifiers can be done in the sliding window model using semi-streaming space. Furthermore, as previously mentioned, they also gave the first sliding window algorithms for MM and MWM.

The smooth histogram technique used in our work was introduced by Braverman and Ostrovsky [6] and can be regarded as an improvement of the exponential histogram technique [10] for smooth functions. This technique has successfully been applied to a wide range of problems, including the computation of coresets [20] and for clustering problems [5].

Outline

We first give notation and a discussion of Paz and Schwartzman's algorithm including its properties in Section 2. The $(2 + \varepsilon)$-approximation is presented in Section 3. The semi-streaming $(3 + \varepsilon)$-approximation via the refined lookahead algorithms is then given in Section 4. Finally, we conclude with open questions in Section 5.

2 Preliminaries

In this section, we start with some important notation and a formal description of the improved version of Paz and Schwartzman's algorithm by Ghaffari and Wajc (see Algorithm 1). This is followed by some key insights about the algorithm.

Let S be an input stream representing an edge-weighted graph $G = (V, E, w)$ with a weight function $w : E \to \mathbb{R}^+$. We assume that each edge, including its weight, can be stored in a single word of memory; as such, all our space bounds are in terms of words of memory. For any subset of edges $F \subseteq E$, let $w(F) = \sum_{e \in F} w(e)$ be the sum of their weights. Then, for any maximum-weight matching in G, denoted by $M^*(S)$, we have that $\mathsf{MWM}(S) = w\big(M^*(S)\big)$.

Algorithm 1 $\mathcal{ALG}^{\varepsilon}_{PS}$ (Paz and Schwartzman's algorithm with Ghaffari and Wajc's improvements.)

```
Input: A stream S of weighted edges
Initialization:
 1: Stack ← an empty stack
 2: for every vertex $v \in V$ do $\varphi(v) \leftarrow 0$
Streaming:
 3: while a new edge $e = \{u, v\}$ of the stream $S$ is revealed do
 4:     if $w(e) < (1+\varepsilon) \cdot \big(\varphi(u) + \varphi(v)\big)$ then $w'(e) \leftarrow 0$
 5:     else
 6:         $w'(e) \leftarrow w(e) - \big(\varphi(u) + \varphi(v)\big)$          ▷ $w'(e)$ is the reduced weight of $e$
 7:         $\varphi(u) \leftarrow \varphi(u) + w'(e)$; $\varphi(v) \leftarrow \varphi(v) + w'(e)$          ▷ update potentials
 8:         Stack.Push($e$)
 9:     for $x \in \{u, v\}$ do          ▷ optimizing space
10:         if $x$ is adjacent to $> \frac{3\log(1/\varepsilon)}{\varepsilon} + 1$ edges in Stack then
11:             Remove the oldest edge adjacent to $x$ from Stack
Postprocessing:
12: Let $\hat{M}$ be an empty matching
13: while Stack is not empty do
14:     $e \leftarrow$ Stack.Pop()
15:     if $\hat{M} \cup \{e\}$ is a matching then $\hat{M} \leftarrow \hat{M} \cup \{e\}$
16: return $\hat{M}$          ▷ a GREEDY matching of the edges in Stack
```

$\mathcal{ALG}^{\varepsilon}_{PS}$ (Algorithm 1) uses the notions of reduced weights and vertex potentials. These are respectively represented by the functions $w'_S : E \to \mathbb{R}^+_0$ and $\varphi_S : V \to \mathbb{R}^+_0$ when the algorithm is executed on a stream S. The sum of all reduced weights is denoted by $W'_S = \sum_{e \in S} w'_S(e)$. For any edge in the stream, its reduced weight is non-negative and is unchanged by the processing of any subsequent edges. In particular, for a stream AB and any edge $e \in A$ (i.e., the edge e is present in the stream A), we have $w'_A(e) = w'_{AB}(e) \geq 0$. Hence, the sum of the reduced weights is a non-decreasing function, i.e., $W'_A \leq W'_{AB}$. The output matching of $\mathcal{ALG}^{\varepsilon}_{PS}$ on stream S is denoted by $\hat{M}(S)$.

Ghaffari and Wajc's analysis of the algorithm reveals the following key observations and results which we later use in our proofs.

▶ **Observation 4** (Ghaffari and Wajc [13]). *At any moment there are $O\left(\frac{\log(1/\varepsilon)}{\varepsilon} \cdot n\right)$ edges stored in* Stack *during the execution of $\mathcal{ALG}^{\varepsilon}_{PS}$.*

▶ **Proposition 5** (Ghaffari and Wajc [13]). *For any edge $e = \{u, v\}$ in a stream S, after the execution of $\mathcal{ALG}^{\varepsilon}_{PS}$, its weight is bounded as $w(e) \leq (1+\varepsilon) \cdot \big(\varphi_S(u) + \varphi_S(v)\big)$.*

▶ **Proposition 6** (Ghaffari and Wajc [13]). *Let $\varepsilon > 0$ and S be a stream of edges. Then, the following inequalities hold:*

$$w\big(M^*(S)\big) \geq W'_S,$$
$$w\big(\hat{M}(S)\big) \geq \frac{1}{1+4\varepsilon} \cdot W'_S = \frac{1}{2(1+4\varepsilon)} \sum_{v \in V} \varphi_S(v) \geq \frac{1}{2(1+4\varepsilon)(1+\varepsilon)} \cdot w\big(M^*(S)\big).$$

Note that Proposition 6 uses the important fact that $W'_S = \frac{1}{2}\sum_{v\in V} \varphi_S(v)$ as the potential of a vertex v is actually the sum of reduced weights of edges incident to v. Furthermore, its last inequality is due to Proposition 5 since each vertex in a matching is incident to at most one edge. Indeed, Proposition 6 shows that $\mathcal{ALG}^{\varepsilon}_{PS}$ is a $(2+\varepsilon)$-approximation streaming algorithm for MWM, and, by Observation 4, $\mathcal{ALG}^{\varepsilon}_{PS}$ uses space $O(\frac{\log(1/\varepsilon)}{\varepsilon} \cdot n)$ (in words).

3 $(2+\varepsilon)$-approximation Sliding Window Algorithm

In this section, we give a $(2+\varepsilon)$-approximation sliding window algorithm for MWM with space $\tilde{O}(\sqrt{nL})$, where L is the length of the sliding window.

Algorithm 2 MWM Sliding Window Algorithm.

```
Input: A stream $S$ with a sliding window of length $L$
$\mathcal{A}$: $\mathcal{ALG}^{\varepsilon}_{PS}$ with sum of reduced weights $W'$ and output matching $\hat{M}$.
Initialization:
 1: Stack ← an empty stack
 2: $k \leftarrow 0$                                                  ▷ Number of blocks
 3: Parameter $s \leftarrow \left\lfloor \frac{\sqrt{n \cdot L \cdot \log 1/\varepsilon \cdot \log \sigma}}{\varepsilon} \right\rfloor$ for $\sigma = \frac{n}{2} \cdot w_{\max}/w_{\min}$.
Streaming:
 4: while a new item $e$ of the stream $S$ is revealed do
 5:     Feed $e$ to all existing instances of $\mathcal{A}$
 6:     Delete all instances of $\mathcal{A}$ which have processed more than $L$ edges
 7:     Stack.Push($e$)
 8:     if |Stack| $\geq s$ then                            ▷ Create new instances of $\mathcal{A}$
 9:         $k \leftarrow k + 1$
10:         Let $\mathcal{I}^k_1$ be a new instances of $\mathcal{A}$
11:         Let $W'_{\text{prev}} \leftarrow 0, i \leftarrow 1$
12:         while Stack is not empty do       ▷ Process all edges in reverse order of arrival
13:             $e' \leftarrow$ Stack.Pop and feed $e'$ to $\mathcal{I}^k_i$
14:             if $W'(\mathcal{I}^k_i) > (1+\varepsilon) \cdot W'_{\text{prev}}$ then     ▷ $W'(\mathcal{I}^k_i)$ exceeds $(1+\varepsilon) \cdot W'(\mathcal{I}^k_{i-1})$
15:                 Create a new instance $\mathcal{I}^k_{i+1}$ as a copy of $\mathcal{I}^k_i$
16:                 $W'_{\text{prev}} \leftarrow W'(\mathcal{I}^k_i), i \leftarrow i + 1$
17:     if any instance of $\mathcal{A}$ exists then
18:         report output matching of the instance that has processed the most edges
19:     else report the maximum-weight matching of the edges in Stack
```

For brevity of notation, denote by $\mathcal{A}$ the Paz and Schwartzman algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$, which our algorithm (see Algorithm 2 for a listing) maintains several instances of. When the current edge e of the stream arrives, the algorithm feeds e to all existing instances of $\mathcal{A}$, then deletes any instance that has processed more than L edges, i.e., the ones that could return edges outside the sliding window. The edge e is subsequently pushed onto Stack.

When Stack has accumulated s edges, Algorithm 2 uses it to create several instances of $\mathcal{A}$: It first creates a new instance $\mathcal{I}_1$ of $\mathcal{A}$, then starts to pop the edges from Stack, processing the edges in reverse order of their arrival. When an edge e is popped it is fed into the last created instance $\mathcal{I}_i$ (initially $\mathcal{I}_1$). At any given moment, the algorithm stores the sum of reduced weights $W'(\mathcal{I}_{i-1})$ of the previous instance (initially set to 0). If the sum of reduced weights $W'(\mathcal{I}_i)$ of the latest instance $\mathcal{I}_i$ exceeds $(1+\varepsilon)\cdot W'(\mathcal{I}_{i-1})$, then a new instance $\mathcal{I}_{i+1}$ is created as a copy of $\mathcal{I}_i$. This procedure is repeated until Stack is empty again.

After processing edge e, the algorithm reports the matching computed by the instance of $\mathcal{A}$ which has processed the most edges of the current sliding window. If no instances have been created yet, then it reports an exact solution on the edges stored in Stack.

Overall, Algorithm 2 maintains multiple runs of $\mathcal{A}$, each fed with different suffixes of the sliding window. It uses Stack to implicitly partition the stream S into blocks $B_1, B_2, \dots$ of s edges each, thus processing it block by block. Each block B_j is then processed, crucially in reverse order of arrival, feeding each edge into an initially empty instance $\mathcal{I}^j_1$ of $\mathcal{A}$. Then, copies $\mathcal{I}^j_i$ are created whenever the sum of reduced weights exceed a $(1+\varepsilon)$ factor of the previous copy. Once the block B_j has been processed entirely, the subsequent edges of the stream are fed to the instances $\mathcal{I}^j_1, \mathcal{I}^j_2, \dots, \mathcal{I}^j_\ell$ in the natural arrival order. Note that the algorithm constructs the instances such that $\mathcal{I}^j_1$ only processes a single edge of the block B_j and $\mathcal{I}^j_\ell$ processes the entire block.

Intuitively, Algorithm 2 ensures that, as edges of the block start to fall outside of the sliding window, the oldest remaining instance is still a good approximation of the solution on the entire sliding window, i.e., consecutive runs of $\mathcal{A}$ are not too different in terms of output. Moreover, immediately after processing block B_j, it holds that $W'(\mathcal{I}^j_i) > (1+\varepsilon)\cdot W'(\mathcal{I}^j_{i-1})$ for all $1 < i \leq \ell$. Therefore, there are only logarithmically many runs of $\mathcal{A}$ per block.

In the following proofs, we use a notion $S(\mathcal{I})$ to denote a substream that is processed by the instance $\mathcal{I}$ of $\mathcal{A}$.

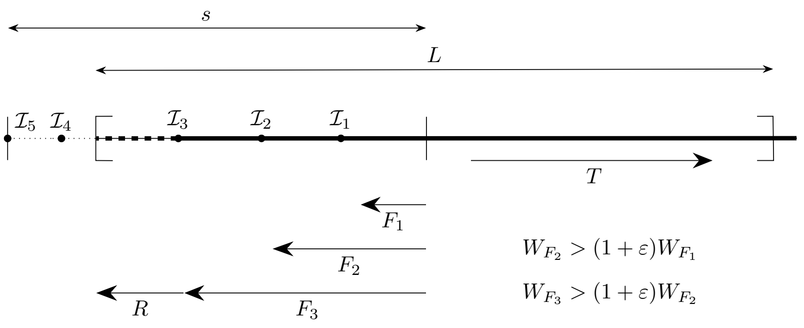

Figure 1 A schematic of a block of the algorithm. The notation here coincides with the notation used in the proof of Theorem 7. The window of length L is marked between the square brackets. There are five instances $\mathcal{I}_1, \dots, \mathcal{I}_5$ created for the block of length s. The instance $\mathcal{I}_i$ processed the stream $\overleftarrow{F_i}T$. Note that $\mathcal{I}_4$ and $\mathcal{I}_5$ are already expired, thus they were deleted. The algorithm outputs the result of the instance $\mathcal{I}_3$ meaning the stream $\overleftarrow{F_3}T$. The proof of Theorem 7 will show that the omission of the remainder $\overleftarrow{R}$ does not compromise the output matching too much.

▶ **Theorem 7.** *There is a deterministic streaming sliding window algorithm for* Maximum-weight Matching *with an approximation factor* $2+\varepsilon$ *that uses* $O\left(\frac{\sqrt{n \cdot L \cdot \log 1/\varepsilon \cdot \log \sigma}}{\varepsilon}\right)$ *words of memory for any* $\varepsilon > 0$ *and* $\sigma = \frac{n}{2} \cdot w_{max}/w_{min}$.

Proof. We will prove that Algorithm 2 satisfies the assertion of the theorem. Let B_j be the oldest block of the stream which is still partially contained in the current sliding window E, i.e., E contains at least one edge of B_j and no edge of B_{j-1}. Let $\mathcal{I}_1^j, \ldots, \mathcal{I}_\ell^j$ be the instances created during the processing of block B_j. Note that each instance $\mathcal{I}_i^j$ processes the edges of B_j in reverse order. Thus, we consider B_j as a stream of edges ordered in reverse to the order in which they arrived. For clarity, we denote this as $\overleftarrow{B_j}$ and similarly for all relevant substreams of $\overleftarrow{B_j}$. Let $\overleftarrow{F_i}$ be the substream of $\overleftarrow{B_j}$ fed into the instance $\mathcal{I}_i^j$, i.e., $\overleftarrow{F_i} = S(\mathcal{I}_i^j) \cap \overleftarrow{B_j}$. Note that $\overleftarrow{F_1} \subseteq \cdots \subseteq \overleftarrow{F_\ell} = \overleftarrow{B_j}$.

Approximation. Let T be the stream of edges that arrive after the stream $\overleftarrow{B_j}$, i.e., $E \subseteq \overleftarrow{B_j}T$. First suppose that $E = \overleftarrow{B_j}T$. Then, Algorithm 2 returns the matching computed by the oldest instance $\mathcal{I}_\ell^j$ which has processed the whole stream $\overleftarrow{F_\ell}T$, i.e., all edges of E (as $\overleftarrow{F_\ell} = \overleftarrow{B_j}$). Thus, it returns a $(2+\varepsilon)$-approximation of the optimal solution.

Now, suppose that $E \subset \overleftarrow{B_j}T$. Let $\overleftarrow{F_i}T \subseteq E \subset \overleftarrow{F}_{i+1}T$. Note that such an i exists as $E \cap \overleftarrow{B_j} \neq \emptyset$ and $|\overleftarrow{F_1}| = 1$. Algorithm 2 returns a matching computed by the instance $\mathcal{I}_i^j$ that processed the stream $S(\mathcal{I}_i^j) = \overleftarrow{F}T$ for $\overleftarrow{F} = \overleftarrow{F_i}$. Let $\overleftarrow{R}$ be the substream of $\overleftarrow{F}_{i+1} \setminus \overleftarrow{F}$ such that E contains exactly the edges of the stream $\overleftarrow{F}\overleftarrow{R}T$. See Figure 1, for an illustration of the substreams processed by various instances $\mathcal{I}_i$.

Since $E \subset \overleftarrow{F}_{i+1}T$, it holds by construction of Algorithm 2 that $W'_{\overleftarrow{F}\overleftarrow{R}} \leq (1+\varepsilon) \cdot W'_{\overleftarrow{F}}$, where $W'_{\overleftarrow{F}\overleftarrow{R}}$ and $W'_{\overleftarrow{F}}$ are the sums of reduced weights computed by $\mathcal{A}$ on streams $\overleftarrow{F}\overleftarrow{R}$ and $\overleftarrow{F}$, respectively. For any vertex v, let $\Delta(v) := \varphi_{\overleftarrow{F}\overleftarrow{R}}(v) - \varphi_{\overleftarrow{F}}(v)$. Recall that φ is an increasing function by the construction of the algorithm, thus $\Delta(v) \geq 0$. Then, by the proportionality between the sum of reduced weights and the sum of potentials ($\sum_e w'(e) = 2\sum_v \varphi(v)$, see Proposition 6), we have the following upper bound:

$$\sum_{v \in V} \Delta(v) = \sum_{v \in V} \varphi_{\overleftarrow{F}\overleftarrow{R}}(v) - \varphi_{\overleftarrow{F}}(v) \leq \sum_{v \in V} (1+\varepsilon) \cdot \varphi_{\overleftarrow{F}}(v) - \varphi_{\overleftarrow{F}}(v) = \varepsilon \cdot \sum_{v \in V} \varphi_{\overleftarrow{F}}(v) \ .$$

We now claim that if we assign, for every $v \in V$, a weight $c(v) := (1+\varepsilon) \cdot \big(\varphi_{\overleftarrow{F}T}(v) + \Delta(v)\big)$, then we have a valid (weighted) vertex cover in the graph consisting of all edges in $\overleftarrow{F}\overleftarrow{R}T$, i.e., for each edge $e = \{u, v\} \in \overleftarrow{F}\overleftarrow{R}T$, it holds that $c(e) := c(u) + c(v) \geq w(e)$. Consider two cases. If $e \in \overleftarrow{F}T$, then we have $c(e) \geq (1+\varepsilon) \cdot \big(\varphi_{\overleftarrow{F}T}(v) + \varphi_{\overleftarrow{F}T}(u)\big) \geq w(e)$. Otherwise, $e \in \overleftarrow{R}$ and we have

$$\begin{aligned}
c(e) &= (1+\varepsilon) \cdot \big(\varphi_{\overleftarrow{F}T}(v) + \varphi_{\overleftarrow{F}T}(u) + \varphi_{\overleftarrow{F}\overleftarrow{R}}(v) - \varphi_{\overleftarrow{F}}(v) + \varphi_{\overleftarrow{F}\overleftarrow{R}}(u) - \varphi_{\overleftarrow{F}}(u)\big) \\
&\geq (1+\varepsilon) \cdot \big(\varphi_{\overleftarrow{F}\overleftarrow{R}}(v) + \varphi_{\overleftarrow{F}\overleftarrow{R}}(u)\big) && (\overleftarrow{F} \text{ is a substream of } \overleftarrow{F}T) \\
&\geq w(e) \ . && \text{(by Proposition 5)}
\end{aligned}$$

Thus, we get a valid vertex cover as required. Now, we can use this to show that the returned matching $\hat{M}(\overleftarrow{F}T)$ computed by $\mathcal{A}$ on the stream $\overleftarrow{F}T$ is a $(2+\varepsilon)$-approximation of the maximum weighted matching M^* of the sliding window E:

$$\begin{aligned}
w(M^*) &= \sum_{e \in M^*} w(e) \le \sum_{v \in V} c(v) && \text{(each vertex is incident to at most one edge in } M^*\text{)} \\
&= (1+\varepsilon) \sum_{v \in V} \big(\varphi_{\overleftarrow{F}T}(v) + \Delta(v)\big) \\
&\le (1+\varepsilon) \sum_{v \in V} \big(\varphi_{\overleftarrow{F}T}(v) + \varepsilon \varphi_{\overleftarrow{F}}(v)\big) \\
&\le (1+\varepsilon)^2 \sum_{v \in V} \varphi_{\overleftarrow{F}T}(v) && \text{(since } \varphi \text{ is monotonic)} \\
&\le (1+3\varepsilon) \sum_{v \in V} \varphi_{\overleftarrow{F}T}(v) && \text{(since } \varepsilon^2 < \varepsilon \text{ for } 0 < \varepsilon < 1\text{)} \\
&\le 2(1+3\varepsilon)(1+4\varepsilon) \cdot w\big(\hat{M}(\overleftarrow{F}T)\big) && \text{(by Proposition 6)} \\
&\le (2+38\varepsilon) \cdot w\big(\hat{M}(\overleftarrow{F}T)\big) \; .
\end{aligned}$$

Space. The sliding window E can be covered by $O\left(\frac{L}{s}\right)$ many blocks, as s is the block size. First, we bound the number of instances ℓ created for a block $\overleftarrow{B_j}$. Recall that edges from the block B_j processed by the instance $\mathcal{I}_i^j$ are the edges in $\overleftarrow{F_i}$, and $\overleftarrow{F_1} \subseteq \cdots \subseteq \overleftarrow{F_\ell} = \overleftarrow{B_j}$. Furthermore, $\overleftarrow{F_1} = \{e'\}, W'_{\overleftarrow{F_1}} = w(e') \ge w_{\min}$, and $W'_{\overleftarrow{F}_{i+1}} > (1+\varepsilon) \cdot W'_{\overleftarrow{F_i}}$ for all $i < \ell$. Thus,

$$(1+\varepsilon)^{\ell-1} \cdot w_{\min} \le (1+\varepsilon)^{\ell-1} \cdot W'_{\overleftarrow{F_1}} < W'_{\overleftarrow{F_\ell}} \le w\big(\hat{M}(\overleftarrow{F_\ell})\big) \le \frac{n}{2} \cdot w_{\max}.$$

By rearranging, we get $\ell = O(\log_{1+\varepsilon} \sigma) = O\left(\frac{1}{\varepsilon} \cdot \log \sigma\right)$. By Observation 4, each instance of $\mathcal{A}$ stores $O\left(\frac{n \log(1/\varepsilon)}{\varepsilon}\right)$ edges. Thus, at any moment, all existing instances of $\mathcal{A}$ store $O\left(\frac{L}{s} \cdot \frac{\log \sigma}{\varepsilon} \cdot \frac{n \log(1/\varepsilon)}{\varepsilon}\right)$ many edges.

Note that we additionally need to store the edges of at most one block (stored in Stack), i.e., at most s edges. Overall, we need to store at most $O\left(\frac{L}{s} \cdot \frac{\log \sigma}{\varepsilon} \cdot \frac{n \log(1/\varepsilon)}{\varepsilon} + s\right)$ edges. Setting s to $\left\lfloor \frac{\sqrt{n \cdot L \cdot \log 1/\varepsilon \cdot \log \sigma}}{\varepsilon} \right\rfloor$ gives us the final space bound in words of memory. ◀

▶ Remark. Assuming that ε is constant and that σ is polynomial in n, we obtain an algorithm that uses $\tilde{O}(\sqrt{nL})$ space. This is $o(n^2)$ space as long as $L = \tilde{o}(n^3)$. If, additionally, the input graph of each window is simple, we have that $L = O(n^2)$ (a simple graph always has $O(n^2)$ edges) and a space bound of $O\left(n\sqrt{n} \cdot \frac{\sqrt{\log 1/\varepsilon \cdot \log \sigma}}{\varepsilon}\right)$, which simplifies to $\tilde{O}\left(n\sqrt{n}\right)$.

We can easily adapt the algorithm to the (unweighted) MM problem. More specifically, the Paz-Schwartzman algorithm becomes the Greedy matching algorithm, while the sum of reduced weights simply becomes the size of the Greedy matching obtained. While the approximation factor remains $2+\varepsilon$, the matchings of the instances now store $O(n)$ edges instead of $O\left(\frac{n \log(1/\varepsilon)}{\varepsilon}\right)$. Also, $\sigma = \frac{n}{2}$. Then, by setting s to $\left\lfloor \sqrt{\frac{n \cdot L \cdot \log n}{\varepsilon}} \right\rfloor$, we obtain a better memory bound for the algorithm. This adaptation yields the following result:

▶ **Theorem 8.** *There is a deterministic streaming sliding window algorithm for MM with an approximation factor of $(2+\varepsilon)$ that uses $O\left(\sqrt{\frac{n \cdot L \cdot \log n}{\varepsilon}}\right)$ words of memory for any $\varepsilon > 0$.*

4 $(3+\varepsilon)$-approximation Sliding Window Algorithm

In this section, we give a $(3+\varepsilon)$-approximation semi-streaming sliding window algorithm by applying the smooth histogram technique [6] in a similar manner as Biabani et al. [4]. We start with our definition of a *refined lookahead algorithm* which we use to describe a sliding window algorithm. Then, we show that $\mathcal{ALG}_{PS}^{\varepsilon}$ is a refined lookahead algorithm; thus, obtaining the sliding window algorithm for MWM.

▶ **Definition 9** ($(f,\alpha_1,\alpha_2,\beta)$-refined lookahead algorithm)**.** *Let $\beta \in (0,1)$, $\alpha_1,\alpha_2 \geq 1$ and, for a ground set X, let $f : 2^X \to \mathbb{R}^+$ be a non-decreasing function. We say a streaming algorithm $\mathcal{ALG}$ with two outputs O_1, O_2 is a $(f,\alpha_1,\alpha_2,\beta)$-*refined lookahead *algorithm if the following holds for any stream S of items of the set X:*

1. *$O_1(S) \leq f(S) \leq \alpha_1 \cdot O_1(S)$, i.e., the first output is an α_1-approximation of f.*
2. *For any partitioning of S into three disjoint sub-streams A, B, and C with $O_1(B) \geq (1-\beta)\cdot O_1(AB)$, we have $O_2(BC) \leq f(ABC) \leq \alpha_2 \cdot O_2(BC)$, i.e., if the first output on the substream AB is similar to the first output on the substream B then the second output on the substream BC is an α_2-approximation of f on the whole stream $S = ABC$.*

Observe that if $O_1 = O_2$ and $\alpha_1 = \alpha_2 = \alpha$ then we retrieve the standard definition of a (f,α,β)-lookahead algorithm as given by Biabani et al. (see Definition 3). Our refined lookahead algorithm is also similarly turned into a sliding window algorithm. In essence, the algorithm simulates runs of a traditional streaming algorithm on suffixes of the current sliding window. It maintains runs on suffixes such that the value of O_1 of any two consecutive runs are not too different, while the value of O_1 of any non-consecutive runs are sufficiently different so as to ensure that at most a logarithmic number of runs are required at any point of time. The second output O_2 is a solution which, given the smoothness assumptions of the runs, is always guaranteed to be an α_2-approximation of the next oldest run. Details of the algorithm and the proof of the following theorem are provided in Appendix B.

▶ **Theorem 10.** *Let $0 < \beta < 1$ and $\alpha_1,\alpha_2 \geq 1$, S be a stream of items from a set X, and $f : 2^X \to \mathbb{R}^+$ be a non-decreasing function. Suppose there exists a $(f,\alpha_1,\alpha_2,\beta)$-refined lookahead algorithm that uses at most s words of memory. Then, there is a sliding window algorithm that maintains an α_2-approximation of f using $O\big(\frac{1}{\beta}\cdot s\log(\alpha_1\sigma)\big)$ words of memory for $\sigma = f(S)/f_{\min}$ where $f_{\min} = \min\{f(e) : e \in S\}$.*

We will now apply Definition 9 to algorithm $\mathcal{ALG}_{PS}^{\varepsilon}$. To this end, we consider the first output O_1 as the sum of reduced weights W'_S, the second output O_2 as the weight of the returned matching $w\big(\hat{M}(S)\big)$, and function f as the weight of a maximum-weight matching $\mathsf{MWM}(S)$. In fact, we prove in Theorem 11 that this indeed yields a $\big(\mathsf{MWM}, (2+2\varepsilon), (3+20\varepsilon), \beta\big)$-refined lookahead algorithm. Hence, the algorithm given by Theorem 10 with $\mathcal{ALG}_{PS}^{\varepsilon}$ is a $(3+\varepsilon)$-approximation semi-streaming sliding window algorithms for MWM.

▶ **Theorem 11.** *Let $0 < \varepsilon \leq \frac{1}{10}$ and $0 < \beta \leq \frac{\varepsilon}{9}$. The algorithm $\mathcal{ALG}_{PS}^{\varepsilon}$ is a $\big(\mathsf{MWM}, (2+2\varepsilon), (3+20\varepsilon), \beta\big)$-refined lookahead algorithm.*

To prove Theorem 11, we follow the approach of Biabani et al. [4]. Let an input stream S be partitioned into three substreams ABC. They split the maximum matching of the stream $M^* = M^*(ABC)$ into two parts M^*_{AB} and M^*_C where $M^*_{AB} := M^* \cap AB$ is the restriction of M^* to the edges in AB, analogously for the substream C. Biabani et al. then bound the weights of these two parts separately. To this end, they use the notion of a *critical subgraph*.

▶ **Definition 12** (Critical Subgraph [4]). *Consider a graph G specified by a stream S of edges. Let A, B, C be disjoint substreams of S such that $S = ABC$. Then, the* critical subgraph *of G with respect to the maximum matching $M^*(ABC)$ and the substreams A, B, C is the subgraph $H = (V_H, E_H)$ such that*

- *$E_H := \{e \in B \mid e$ is adjacent to two edges in $M^*_C\}$.*
- *$V_H := V(E_H)$, i.e., V_H is the set of endpoints of the edges in E_H.*

Biabani et al. use the critical subgraph to bound the weights of M^*_{AB} and M^*_C in terms of the weight of the matching returned by the algorithm $w(\hat{M})$ (Lemmas 13 and 14 in their work [4]). In our analysis, in particular, in Lemmas 15 and 16, we use the same ideas to bound the weights of M^*_{AB} and M^*_C in terms of sums of reduced weights computed by the algorithm instead.

Before stating and proving Lemmas 15 and 16, we present the following auxiliary lemma already proved by Biabani et al. in the exact formulation as we need it. We highlight that their proof holds for any run of $\mathcal{ALG}^{\varepsilon}_{PS}$ on an arbitrary stream.

▶ **Lemma 13** (Biabani et al. [4], Lemma 15). *For any stream AB,*

$$(1+\varepsilon) \cdot \sum_{v \in V_H} \varphi_{AB}(v) \geq \sum_{e \in E_H} w'_B(e) \ .$$

For the statement of the next auxiliary lemma, we need the following notion. Let S be a stream of edges. For an edge $e \in S$, we define the set $P_S(e)$ as the set of edges incident to e (including e) arriving no later than e, i.e, $P_S(e) = \{e' \in S \mid e' \cap e \neq \emptyset, t_{e'} \leq t_e\}$, where, for any edge f, t_f is the arrival time of edge f. Biabani et al. [4] showed that the weight of any edge e can be bounded by the sum of the reduced weights of the edges in $P_S(e)$ (up to a $(1+\varepsilon)$ factor).

▶ **Lemma 14** (Biabani et al. [4], Lemma 5). *For each edge $e \in S$,*

$$w(e) \leq (1+\varepsilon) \sum_{e' \in P_S(e)} w'_S(e).$$

With that, we can finally prove our analogous lemmas of Biabani et al.'s Lemmas 13 and 14 [4] which bound $w(M^*_{AB})$ and $w(M^*_C)$, respectively.

▶ **Lemma 15** (Analogue of Lemma 13, [4]). *For any stream ABC,*

$$w(M^*_{AB}) \leq 2(1+\varepsilon) \cdot W'_{AB} - \sum_{e \in E_H} w'_B(e).$$

Proof. By definition, we have $w(M^*_{AB}) = \sum_{e \in M^*_{AB}} w(e)$. Let $e = \{u, v\} \in M^*_{AB}$. Note that the vertices u and v are not in V_H. Thus, we can bound the sum as follows.

$$\begin{aligned} w(M^*_{AB}) &\leq (1+\varepsilon) \sum_{v \in V \setminus V_H} \varphi_{AB}(v) && \text{by Proposition 5} \\ &= (1+\varepsilon) \left(\sum_{v \in V} \varphi_{AB}(v) - \sum_{v \in V_H} \varphi_{AB}(v) \right) \end{aligned}$$

By Proposition 6 and by Lemma 13, we have

$$\sum_{v \in V} \varphi_{AB}(v) = 2W'_{AB} \quad \text{and} \quad (1+\varepsilon) \sum_{v \in V_H} \varphi_{AB}(v) \geq \sum_{e \in E_H} w'_B(e).$$

Thus, we can conclude that

$$w(M^*_{AB}) \leq 2(1+\varepsilon) \cdot W'_{AB} - \sum_{e \in E_H} w'_B(e).$$ ◀

▶ **Lemma 16** (Analogue of Lemma 14, [4]). *For any stream ABC,*

$$w\big(M_C^*\big) \le 2(1+\varepsilon)\cdot W'_{BC} - (1+\varepsilon)\sum_{e\in B\setminus E_H} w'_B(e).$$

Proof. First, when considering a run of the algorithm on BC, by Lemma 14, we obtain

$$w\big(M_C^*\big) = \sum_{e\in M_C^*} w(e) \le (1+\varepsilon)\sum_{e\in M_C^*}\sum_{e'\in P(e)} w'_{BC}(e')\ .$$

Observe that any edge $e \in BC$ is incident to at most two edges of M_C^*, and the edges of $B \setminus E_H$ are incident to at most one edge of M_C^*. Hence, we can rewrite the previous double sum as follows:

$$\begin{aligned}\sum_{e\in M_C^*}\sum_{e'\in P(e)} w'_{BC}(e') &\le 2\cdot\sum_{e\in BC} w'_{BC}(e) - \sum_{e\in B\setminus E_H} w'_{BC}(e)\\ &= 2\cdot W'_{BC} - \sum_{e\in B\setminus E_H} w'_{BC}(e)\ ,\end{aligned}$$

which implies the result. ◀

Now, we are ready to prove Theorem 11, i.e., $\mathcal{ALG}_{PS}^{\varepsilon}$ is a $\big(\mathsf{MWM}, (2+2\varepsilon), (3+20\varepsilon), \beta\big)$-refined lookahead algorithm for suitable parameters ε and β.

Proof of Theorem 11. We recall that we consider a version of $\mathcal{ALG}_{PS}^{\varepsilon}$ such that the first output is the sum of reduced weight W' and the second output is the weight of the computed matching $w\big(\hat{M}\big)$. First, by Proposition 6, we get that for any stream S it holds that $W'_S \le w\big(M^*(S)\big) \le 2(1+\varepsilon)\cdot W'_S$. Thus, it remains to prove that for any stream ABC, given that $W'_B \ge (1-\beta)\cdot W'_{AB}$, the maximum matching $M^* = M^*(ABC)$ is such that $w\big(M^*\big) \le (3+20\varepsilon)\cdot w\big(\hat{M}(BC)\big)$.

$$\begin{aligned}
w\big(M^*\big) &\le 2(1+\varepsilon)\cdot W'_{AB} + 2(1+\varepsilon)\cdot W'_{BC} - W'_B && \text{by Lemmas 15 and 16}\\
&\le \frac{2(1+\varepsilon)}{1-\beta}\cdot W'_B + 2(1+\varepsilon)\cdot W'_{BC} - W'_B && \text{by } W'_B \ge (1-\beta)\cdot W'_{AB}\\
&\le (1+3\varepsilon)\cdot W'_B + 2(1+\varepsilon)\cdot W'_{BC} && \text{since } \beta \le \frac{\varepsilon}{9}\\
&\le (3+5\varepsilon)\cdot W'_{BC} && \text{by } W' \text{ being non-decreasing}\\
&\le (3+5\varepsilon)(1+4\varepsilon)\cdot w\big(\hat{M}(BC)\big) && \text{by Proposition 6}\\
&\le (3+20\varepsilon)\cdot w\big(\hat{M}(BC)\big) && \text{since } \varepsilon \le \frac{1}{10}
\end{aligned}$$ ◀

Theorems 10 and 11 together then imply our main result.

▶ **Theorem 2.** *There is a deterministic streaming sliding window algorithm for* Maximum-weight Matching *with an approximation factor* $3+\varepsilon$ *that uses* $O\left(\frac{\log(1/\varepsilon)}{\varepsilon^2}\cdot n\log\sigma\right)$ *words of memory, for any* $0<\varepsilon\le 0.1$ *and* $\sigma = \frac{n}{2}\cdot w_{max}/w_{min}$.

▶ Remark. Our $(3+\varepsilon)$-approximation algorithm for MWM yields the $(3+\varepsilon)$-approximation algorithm for MM by Crouch et al. [7] when $\mathcal{ALG}_{PS}^{\varepsilon}$ is replaced with the GREEDY matching algorithm (the sum of reduced weights becomes the size of the matching). The hard instance of their algorithm also holds for our algorithm.

5 Conclusion

In this paper, we gave two algorithms for MWM in the sliding window model. Our first algorithm has an approximation factor of $2+\varepsilon$ and uses space $\tilde{O}(\sqrt{nL})$, and our second algorithm has an approximation factor of $3+\varepsilon$ and uses semi-streaming space. The approximation factor of our semi-streaming algorithm matches the approximation factor of the best semi-streaming sliding window algorithm known for (unweighted) MM [7].

Regarding the semi-streaming space regime, since further improvements in the approximation factor would imply improvements for (unweighted) MM, the most natural direction for future research is to make further progress on the unweighted version of the problem first. Is there a 2.99-approximation semi-streaming space sliding window algorithm for MM?

While the known lower bounds for MM for one-pass streaming algorithms in the insertion-only model also apply to the sliding window model, no stronger lower bounds for the sliding window model are known. Can we prove a lower bound on the approximation factor of sliding window algorithms for MM that use semi-streaming space and are stronger than what is currently known for the insertion-only model, i.e., stronger than $1+\ln(2)$ [15]?

References

1 Sepehr Assadi, Sanjeev Khanna, Yang Li, and Grigory Yaroslavtsev. Maximum matchings in dynamic graph streams and the simultaneous communication model. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364. SIAM, 2016. `doi:10.1137/1.9781611974331.ch93`.

2 Sepehr Assadi and Vihan Shah. An asymptotically optimal algorithm for maximum matching in dynamic streams. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 – February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 9:1–9:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.ITCS.2022.9`.

3 Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approxmation algrithms in memoriam: Shimon even 1935-2004. *ACM Comput. Surv.*, 36(4):422–463, 2004. `doi:10.1145/1041680.1041683`.

4 Leyla Biabani, Mark de Berg, and Morteza Monemizadeh. Maximum-Weight Matching in Sliding Windows and Beyond. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, 2021.

5 Vladimir Braverman, Harry Lang, Keith Levin, and Morteza Monemizadeh. Clustering problems on sliding windows. In *SODA*, 2016.

6 Vladimir Braverman and Rafail Ostrovsky. Smooth histograms for sliding windows. In *FOCS 2007*, 2007.

7 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`.

8 Michael S. Crouch and Daniel Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *APPROX-RANDOM*, 2014.

9 Jacques Dark and Christian Konrad. Optimal lower bounds for matching and vertex cover in dynamic graph streams. In Shubhangi Saraf, editor, *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 169 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.CCC.2020.30`.

10 Mayur Datar, A. Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31:1794–1813, 2002.

11 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *ArXiv*, abs/0907.0305, 2011.

12 Joan Feigenbaum, Sampath Kannan, Andrew Mcgregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348:207–216, 2005.

13 Mohsen Ghaffari and David Wajc. Simplified and space-optimal semi-streaming $(2+\epsilon)$-approximate matching. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 13:1–13:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/OASIcs.SOSA.2019.13`.

14 Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485. SIAM, 2012. `doi:10.1137/1.9781611973099.41`.

15 Michael Kapralov. Space lower bounds for approximating maximum matching in the edge arrival model. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 1874–1893. SIAM, 2021. `doi:10.1137/1.9781611976465.112`.

16 Mikhail Kapralov. Better bounds for matchings in the streaming model. In *SODA*, 2013.

17 Christian Konrad. Maximum matching in turnstile streams. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms – ESA 2015 – 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 840–852. Springer, 2015. `doi:10.1007/978-3-662-48350-3_70`.

18 Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2005. `doi:10.1007/11538462_15`.

19 Ami Paz and Gregory Schwartzman. A $(2+\epsilon)$-approximation for maximum weight matching in the semi-streaming model. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2153–2161. SIAM, 2017. `doi:10.1137/1.9781611974782.140`.

20 Yanhao Wang, Yuchen Li, and Kian-Lee Tan. Coresets for minimum enclosing balls over sliding windows. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

21 Mariano Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20, 2012. `doi:10.1007/s00453-010-9438-5`.

A Hard Instance for Paz and Schwartzman's Algorithm

In this section, we show that the Paz and Schwartzman's algorithm and its monotonic version are no better than $(\mathsf{MWM}, 3.5, \beta)$-lookahead algorithms. The definition of a lookahead algorithm given by Biabani et al. (Definition 3) together with the Paz and Schwartzman's algorithm thus cannot be used to improve upon the approximation factor of 3.5.

Recall that a lookahead algorithm relies on the smoothness of the algorithm's output. More formally, an (f, α, β)-lookahead algorithm $\mathcal{ALG}$ satisfies the condition that for any stream ABC, if $\mathcal{ALG}(B) \geq (1-\beta)\cdot\mathcal{ALG}(AB)$ then $f(ABC) \leq \alpha\cdot\mathcal{ALG}(BC)$ (see Definition 3).

In other words, if the algorithm $\mathcal{ALG}$ outputs similar results on the streams B and AB then the algorithm's output on BC is required to be an α-approximation of the objective value $f(ABC)$ of the whole stream ABC.

We will present a graph G whose edges are divided into three substreams A, B and C such that $\mathcal{ALG}^{\varepsilon}_{PS}$ outputs matchings of the same weight on substreams AB and B, while the outputted matching on substream BC is roughly a 3.5-approximation of a maximum-weight matching of the entire stream ABC. The graph G is such that even if we modified $\mathcal{ALG}^{\varepsilon}_{PS}$ to return maximum-weight matchings among the edges stored in Stack then the same properties still hold. Thus, the hard instance is also hard for the monotonic version of the algorithm. The graph G is depicted in Figure 2.

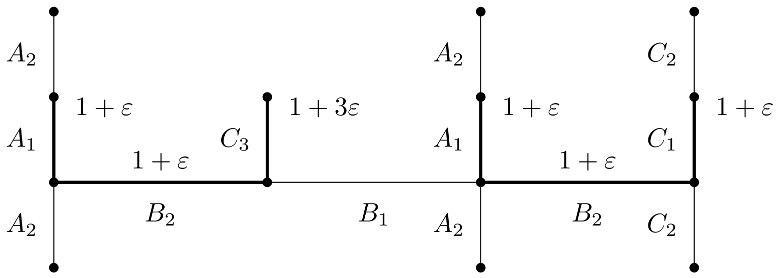

Figure 2 The edges of the graph G are divided into substreams A, B and C. The order of the edges within the substreams is indicated by subscripts (the order of the edges with the same subscript is not important). The thin edges have unit weight and the thick edges have the indicated larger weights.

Matchings computed on AB and B. First, we analyze $\mathcal{ALG}^{\varepsilon}_{PS}$ separately on the substreams A and B. See Figure 3 for the values of the reduced weights and potentials computed by the algorithm.

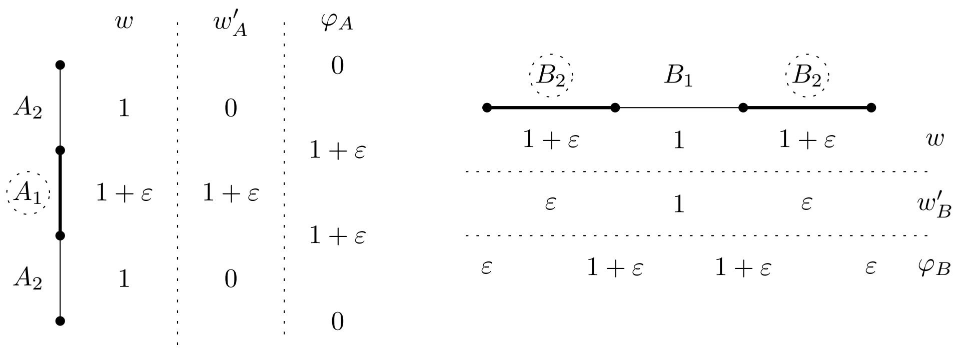

Figure 3 Reduced weights and potentials computed by $\mathcal{ALG}^{\varepsilon}_{PS}$ when run separately on substreams A and B. Recall that substream A consists of two paths, while only one of them is depicted here. The edges outputted by the runs of the algorithm are marked by dotted circles.

Observe that the substream A consists of two disjoint paths of length three. While only one of them is shown in Figure 3, the algorithm computes the same reduced weights and potentials for both paths.

We now analyze the execution of the algorithm on substream AB. To this end, consider the moment when the substream A has been fully processed and substream B begins. Observe that each edge of B is now incident to a single vertex with potential $1+\varepsilon$. Thus, by the construction of the algorithm, none of the edges of B are pushed onto Stack. These edges therefore have reduced weights zero and cannot be outputted by the algorithm. Furthermore, when run on AB, the algorithm outputs the two edges in A_1, i.e., $w\big(\hat{M}(AB)\big) = 2 + 2\varepsilon$, which are the only two edges pushed onto Stack.

As established in Figure 3, when the algorithm runs only on the substream B, it outputs the two edges in B_2, i.e., $w\big(\hat{M}(B)\big) = 2 + 2\varepsilon$. Hence, we have that $w\big(\hat{M}(B)\big) = w\big(\hat{M}(AB)\big)$. It follows that the stream ABC satisfies the condition $w\big(\hat{M}(B)\big) \geq (1-\beta) \cdot w\big(\hat{M}(AB)\big)$, for any value of $\beta \geq 0$, as required by the definition of a lookahead algorithm.

Matching computed on BC. Now, we analyze the execution of the algorithm on the substream BC. At the time when the substream C begins, the reduced weights of edges in B and the current potentials of the incident vertices are the same as when the algorithm is run only on the substream B – see Figure 3 for these values. See Figure 4, for the reduced weights of the edges in C when we run the algorithm on the substream BC.

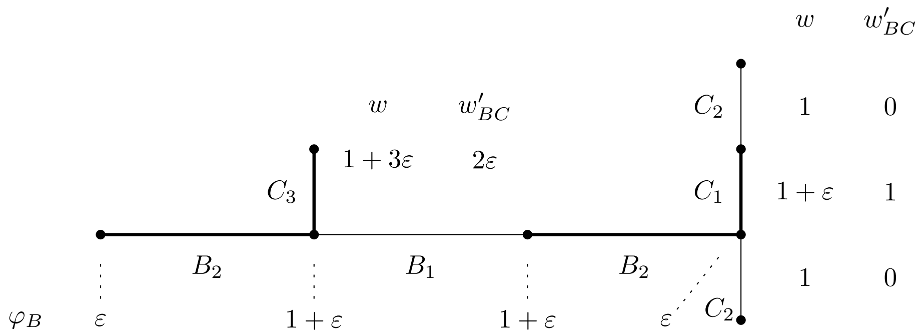

Figure 4 The reduced weights of the edges in C after the execution on the substream BC and the potentials of the vertices incident to the edges in B at the time when the substream B is processed.

By the end of the execution, only the two edges in C_1 and C_3 are pushed onto Stack since the edges in C_2 have reduced weights zero. The algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$ outputs a greedy matching of the edges pushed onto Stack (in the reverse order they arrived). In particular, it outputs the edges in C_1 and C_3 and they block all edges in B. Hence, $w\big(\hat{M}(BC)\big) = 2 + 4\varepsilon$. Observe further that these edges constitute a maximum-weight matching among the edges pushed onto Stack.

Maximum-weight Matching and Approximation Factor. First, observe that the unique maximum-weight matching in G consists of all the edges that have an endpoint of degree 1 (the edges in A_2, C_2, and C_3) and is thus of weight $7 + 3\varepsilon$. Since $w\big(\hat{M}(BC)\big) = 2 + 4\varepsilon$, we conclude that it is not possible for $\mathcal{ALG}^{\varepsilon}_{PS}$ to yield a (MWM, $3.5 - \varDelta, \beta$)-lookahead algorithm, for any constant $\varDelta > 0$ and suitable parameter β.

B More on Refined Lookahead Algorithms

In this section, we will prove Theorem 10. To this end, for convenience, we restate the definition of refined lookahead algorithms first.

▶ **Definition 9** ($(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm). *Let $\beta \in (0,1)$, $\alpha_1, \alpha_2 \geq 1$ and, for a ground set X, let $f : 2^X \to \mathbb{R}^+$ be a non-decreasing function. We say a streaming algorithm $\mathcal{ALG}$ with two outputs O_1, O_2 is a $(f, \alpha_1, \alpha_2, \beta)$-refined* lookahead *algorithm if the following holds for any stream S of items of the set X:*

1. *$O_1(S) \leq f(S) \leq \alpha_1 \cdot O_1(S)$, i.e., the first output is an α_1-approximation of f.*
2. *For any partitioning of S into three disjoint sub-streams A, B, and C with $O_1(B) \geq (1-\beta) \cdot O_1(AB)$, we have $O_2(BC) \leq f(ABC) \leq \alpha_2 \cdot O_2(BC)$, i.e., if the first output on the substream AB is similar to the first output on the substream B then the second output on the substream BC is an α_2-approximation of f on the whole stream $S = ABC$.*

Algorithm 3 LOOKAHEAD SLIDING WINDOW ALGORITHM.

```
Input: A stream $S$ with a sliding window of length $L$
$\mathcal{A}$: a $(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm with outputs $O_1$ and $O_2$
Initialization:
 1: Let $k \leftarrow 0$ be the number of instances
Streaming:
 2: while a new item $e$ of the stream $S$ is revealed do
 3:     Create an instance $\mathcal{I}_{k+1}$ of $\mathcal{A}$
 4:     Feed $e$ into all existing instances $\mathcal{I}_1, \ldots, \mathcal{I}_{k+1}$
 5:     $i \leftarrow 1$
 6:     while $i < k$ do                    ▷ Deleting instances with similar value of $O_1$
 7:         Let $j > i$ be the largest index for which $O_1(\mathcal{I}_j) \geq (1-\beta) \cdot O_1(\mathcal{I}_i)$
 8:         if no such $j$ exists then $j \leftarrow i+1$
 9:         Delete instances $\mathcal{I}_r$ for each $i < r < j$
10:         $i \leftarrow j$
11:     Let $\mathcal{I}_{>1}$ be the next existing instance after $\mathcal{I}_1$          ▷ $\mathcal{I}_1$ was not deleted
12:     if $\mathcal{I}_{>1}$ does not exist then continue to line 15
13:     if $|S(\mathcal{I}_{>1})| \geq L$ then          ▷ $|S(\mathcal{I}_{>1})|$ is the number of items fed into $\mathcal{I}_{>1}$
14:         Delete $\mathcal{I}_1$
15:     Renumber the instances and let $k$ be the number of remaining ones
16:     if $|S(\mathcal{I}_1)| = L$ then report $O_2(\mathcal{I}_1)$
17:     else report $O_2(\mathcal{I}_2)$
```

Let e be the current item of the stream being processed by Algorithm 3 and let E be the current sliding window consisting of the L most recently processed items (including e). While processing e, the algorithm first creates a new instance $\mathcal{I}_{k+1}$ (called a bucket in Biabani et al. [4]) of $\mathcal{A}$. Then, e is fed into all existing instances $\mathcal{I}_1, \ldots, \mathcal{I}_{k+1}$. Next, starting from the oldest instance $\mathcal{I}_1$, only its newest similar instance, determined by O_1 (Item 2 of Definition 9), is kept and every other instance in between is deleted. Whether a newest similar instance exists or not, the process is then repeated with the next oldest remaining instance until reaching the newest instance. Note that the oldest and newest instances, $\mathcal{I}_1$ and $\mathcal{I}_{k+1}$ respectively, are never deleted by this process. However, if the number of items fed into the second oldest remaining instance $\mathcal{I}_{>1}$ is at least L, i.e., the current sliding window E is fully contained in the stream $S(\mathcal{I}_{>1})$ of edges processed by $\mathcal{I}_{>1}$, then $\mathcal{I}_1$ is deleted. The instances are then renumbered to $\mathcal{I}_1, \ldots, \mathcal{I}_k$, from the oldest one to the newest, such that k is the number of remaining instances. At this stage, the sliding window E is sandwiched between streams $S(\mathcal{I}_1)$ and $S(\mathcal{I}_2)$. Finally, after processing the item, if the current sliding window contains exactly the edges processed by $\mathcal{I}_1$, then the algorithm reports the second output O_2 of the instance $\mathcal{I}_1$ as the solution, otherwise it reports $O_2(\mathcal{I}_2)$.

In essence, the instances of $\mathcal{A}$ created by Algorithm 3 simulate runs of a traditional streaming algorithm on suffixes of the current sliding window. Note that the oldest run always contains all items of the sliding window and potentially some additional ones. The idea is to maintain runs on suffixes such that the value of O_1 of any two consecutive runs are not too different, while the value of O_1 of any non-consecutive runs are sufficiently different so as to ensure that at most a logarithmic number of instances of $\mathcal{A}$ is used at any point of time.

This idea is exactly captured when $\mathcal{A}$, with two outputs O_1 and O_2, is a $(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm (which applies the smooth histogram technique by Braverman and Ostrovsky [6]). The first output O_1 is used to determine how often a run on a suffix should be maintained, which depends on the smoothness criteria given by Item 2 of Definition 9. The second output O_2 is a solution which, given the smoothness assumptions of the runs, is always guaranteed to be an α_2-approximation of the next oldest run. We highlight that the smoothness assumptions are only guaranteed to hold for consecutive runs whose suffixes differ by more than one item. Then, for a stream S of items from a set X and a non-decreasing function $f : 2^X \to \mathbb{R}^+$, the number of runs is at most logarithmic in n as long as $\sigma_f(S) = f(S)/f_{\min}$, where $f_{\min} = \min\{f(e) : e \in S\}$, is polynomial in n. We prove this formally in Theorem 10.

▶ **Theorem 10.** *Let $0 < \beta < 1$ and $\alpha_1, \alpha_2 \geq 1$, S be a stream of items from a set X, and $f : 2^X \to \mathbb{R}^+$ be a non-decreasing function. Suppose there exists a $(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm that uses at most s words of memory. Then, there is a sliding window algorithm that maintains an α_2-approximation of f using $O\big(\frac{1}{\beta} \cdot s \log(\alpha_1 \sigma)\big)$ words of memory for $\sigma = f(S)/f_{\min}$ where $f_{\min} = \min\{f(e) : e \in S\}$.*

Proof. We prove that Algorithm 3 satisfies the assertion of the theorem. Let $\mathcal{A}$ be the used $(f, \alpha_1, \alpha_2, \beta)$-refined lookahead algorithm with the outputs O_1 and O_2.

Approximation. Let E be the sliding window at any instance of the algorithm, i.e., the set of the L most recently processed items. The algorithm ensures that E is sandwiched between streams of items fed to $\mathcal{I}_1$ and $\mathcal{I}_2$, i.e., $S_2 \subseteq E \subseteq S_1$ for $S_i = S(\mathcal{I}_i), i \in \{1, 2\}$. We are now in one of two cases, either the items of S_1 and S_2 differ by exactly one item or more than one item.

In the former case, the algorithm asserts that $|S_2| < L$, otherwise S_1 would have been deleted, and therefore the items of S_1 are exactly those of the sliding window E, i.e., $|S_1| = L$. The reported solution is then always $O_2(S_1) = O_2(E)$ which by Item 2 of Definition 9 (consider the case when $E = ABC = BC$) is trivially an α_2-approximation of $f(E)$.

In the latter case, the algorithm would have, at some point, deleted instances which caused $\mathcal{I}_1$ and $\mathcal{I}_2$ to become consecutive instances (Line 9 of Algorithm 3). Consider the time t^* when they first became adjacent. Let S_1^* and S_2^* be the streams processed by $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively, in the time t^*. The algorithm asserts that $O_1(S_2^*) \geq (1 - \beta) \cdot O_1(S_1^*)$. Let C be the remaining items fed into the instances such that $S_1 = S_1^* C$ and $S_2 = S_2^* C$. Then, by Item 2 of Definition 9 and f being non-decreasing,

$$O_2(S_2) \leq f(S_2) \leq f(E) \leq f(S_1) \leq \alpha_2 \cdot O_2(S_2).$$

Hence, we have that, $O_2(S_2)$, is an α_2-approximation of $f(E)$. Now, if $|S_1| \neq L$ the solution reported is $O_2(S_2)$, otherwise $|S_1| = L$ and the solution reported is $O_2(S_1) = O_2(E)$. We conclude that in either case an α_2-approximation of $f(E)$ is reported.

Space. Let k be the maximum number of instances stored by the algorithm after processing an item. After the process of deleting and renumbering the instances, the algorithm ensures that $O_1(\mathcal{I}_{i+2}) < (1 - \beta) \cdot O_1(\mathcal{I}_i)$ holds for any instances $\mathcal{I}_i$ and $\mathcal{I}_{i+2}$. Thus for the largest odd number k' not exceeding k,

$$(1 + \beta)^{\frac{k'-1}{2}} O_1(\mathcal{I}_{k'}) < O_1(\mathcal{I}_1).$$

Recall that $\frac{f(S(\mathcal{I}_1))}{f(S(\mathcal{I}_{k'}))} \leq \sigma$. Then, by Item 1 of Definition 9, we have that $\frac{O_1(\mathcal{I}_1)}{O_1(\mathcal{I}_{k'})} \leq \alpha_1\sigma$. It follows that

$$\frac{k'-1}{2} < \log_{1+\beta}(\alpha_1\sigma) \qquad \text{and} \qquad k' = O\left(\frac{1}{\beta} \cdot \log(\alpha_1\sigma)\right).$$

This implies the result since there are only ever $k+1 \leq k'+2$ instances of $\mathcal{A}$, each of which uses at most s words of memory. ◀

A motivating example of the refined lookahead definition is exactly the Paz-Schwartzman algorithm $\mathcal{ALG}^{\varepsilon}_{PS}$ with the first output O_1 as the sum of reduced weights W'_S, the second output O_2 as the weight of the returned matching $w\big(\hat{M}(S)\big)$, and function f as the weight of a maximum-weight matching $\mathsf{MWM}(S)$. Now, consider the graph given in Appendix A (see Figure 2). We have that $w\big(\hat{M}(AB)\big) = w\big(\hat{M}(B)\big) = 2 + 2\varepsilon$, $W'_{AB} = 2 + 2\varepsilon$ and $W'_B = 1 + 2\varepsilon$. We showed in Appendix A that this is indeed a hard instance for (standard) lookahead algorithms when the weight of the matching computed is used as the smoothness constraint (recall that $(1-\beta) \cdot w\big(\hat{M}(AB)\big) \leq w\big(\hat{M}(B)\big)$ is then required in a hard instance, which is the case here). On the other hand, refined lookahead algorithms allow us to use the sum of reduced weights as the smoothness constraint. Since $(1-\beta) \cdot W'_{AB} \not\leq W'_B$, for small enough β, the instance therefore is not hard for refined lookahead algorithms.