Simplified and Space-Optimal Semi-Streaming $(2+\varepsilon)$-Approximate Matching

Mohsen Ghaffari
ETH Zurich, Zurich, Switzerland

David Wajc
Carnegie Mellon University, Pittsburgh, USA

Abstract

In a recent breakthrough, Paz and Schwartzman (SODA'17) presented a single-pass $(2+\varepsilon)$-approximation algorithm for the maximum weight matching problem in the semi-streaming model. Their algorithm uses $O(n\log^2 n)$ bits of space, for any constant $\varepsilon > 0$.

We present a simplified and more intuitive primal-dual analysis, for essentially the same algorithm, which also improves the space complexity to the optimal bound of $O(n\log n)$ bits – this is optimal as the output matching requires $\Omega(n\log n)$ bits.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases Streaming, Semi-Streaming, Space-Optimal, Matching

Digital Object Identifier 10.4230/OASIcs.SOSA.2019.13

Related Version A full version of the apper is available at [7], `https://arxiv.org/abs/1701.03730`.

Acknowledgements Mohsen Ghaffari is grateful to Gregory Schwartzman for sharing a write-up of [9], and to Ami Paz and Gregory Schwartzman for feedback on an earlier draft of this article. The work of David Wajc is supported in part by NSF grants CCF-1618280, CCF-1814603, CCF-1527110 and NSF CAREER award CCF-1750808.

1 Introduction and Related Work

The maximum weight matching (MWM) problem is a classical optimization problem, with diverse applications, which has been studied extensively since the 1965 work of Edmonds [4]. Naturally, this problem has received significant attention also in the *semi-streaming* model. This is a modern model of computation, introduced by Feigenbaum et al.[6], which is motivated by the need for processing massive graphs whose edge set cannot be stored in memory. In this model, roughly speaking, the edges of the graph arrive in a stream, and the algorithm should process this stream and eventually output its solution – a matching in our case – while using a small memory at all times. Ideally, this memory size should be close to what is needed for storing just the output. More formally, the setting of the MWM problem in the semi-streaming matching is as follows.

MWM in the Semi-Streaming Model. Let $G = (V, E, w)$ be a simple graph with non-negative edge weights $w : E \to \mathbb{R}_{>0}$ (for notational simplicity, we let $w_e = w(e)$). Let $n = |V|$, $m = |E|$. We assume that the edge weights are normalized so that the minimum edge weight is 1 and we use $W = \max_e w_e$ to denote the maximum edge weight. In the semi-streaming model, the input graph G is provided as a stream of edges. In each iteration, the algorithm receives an edge from the stream and processes it. The algorithm has a memory much smaller than m and thus it cannot store the whole graph. The amount of memory that the algorithm uses is called its *space complexity* and we wish to keep it as small as possible. The objective in the semi-streaming *maximum weight matching* (MWM) problem is that, at the end of the stream, the algorithm outputs a matching whose weight is close to the weight of the maximum weight matching, denoted by $M^* = \arg\max_{\text{matching } M} w(M)$, where $w(M) = \sum_{e \in M} w_e$ is the weight of matching M.

State of the Art. There has been a sequence of successively improved approximation algorithms for MWM in the semi-streaming model. Feigenbaum et al. gave a 6 approximation[6], McGregor gave a 5.828 approximation[8] (and a $2+\varepsilon$ approximation, but using $O(1/\varepsilon^3)$ passes on the input), Epstein et al. gave a $4.911+\varepsilon$ approximation[5] and Crouch and Stubbs gave a $4+\varepsilon$ approximation[3]. However, these approximations remained far from the more natural and familiar 2 approximation which the sequential greedy method provides.

In a recent breakthrough, Paz and Schwartzman[9] presented a truly simple algorithm that achieves an approximation of $2+\varepsilon$ for any constant $\varepsilon > 0$, using $O(n \log^2 n)$ bits of space. More concretely, the algorithm maintains $O(n \log n)$ edges, while working through the stream, and at the end, it computes a matching using these maintained edges.

Our Contribution. We present an alternative analysis of (a slight variant of) the algorithm of Paz and Schwartzman, which has two advantages: (1) it implies that keeping merely $O(n)$ edges suffices, and thus improves the space complexity to $O(n \log n)$ bits, which is optimal, (2) it provides a more intuitive and also more flexible accounting of the approximation.

Concretely, Paz and Schwartzman[9] used an extension of *the Local Ratio theorem*[1, 2] to analyze their algorithm. We instead present an alternative simpler primal-dual argument, which is more flexible and allows us to improve the space-complexity to obtain the optimal space bound. The main appeal of the primal-dual method is in the simple explanation it provides for the extension of the local ratio technique in [9] using little more than LP duality.

Roadmap. In Section 2, as a warm up, we review (a simple version of) the algorithm of Paz and Schwartzman. In Section 3, we present our primal-dual analysis for this algorithm. In Section 4, we present the improved algorithm that achieves a space complexity of $O(n \log n)$, the analysis of which relies crucially on the flexibility of the analysis presented in Section 3.

2 Reviewing the Algorithm of Paz and Schwartzman

2.1 The Basic Algorithm

The starting point in the approach of Paz and Schwartzman[9] is the following basic yet elegant algorithm, implicit in [1, Section 3]. For the sake of explanation, consider a sequential model of computation. We later discuss the adaptation to the streaming model.

Basic Algorithm. Repeatedly select an edge e with positive weight; reduce its weight from itself and its neighboring edges; push e into a stack and continue to the next edge, so long as edges with positive weight remain. At the end; unwind the stack and add the edges greedily to the matching.

In Section 3 we will see that this simple algorithm is 2-approximate.

Implementing the Basic Algorithm in the Semi-Streaming Model. To implement this while streaming, we just need to remember a parameter ϕ_v for each node v. This parameter is the total sum of the weight already reduced from the edges incident on vertex v, due to edges incident on v that were processed and put in the stack before. We assume for now that the space requirement of storing these n numbers is only $O(n \log n)$ bits (say, due to the edge weights having magnitude at most some $\text{poly}(n)$) and discuss the space requirement of these ϕ_v values at the end of the paper.

2.2 The Algorithm with Exponentially Increasing Weights

The space complexity of the basic algorithm can become $\Theta(n^2)$, as it may end up pushing $\Theta(n^2)$ edges into the stack. This brings us to the clever idea of Paz and Schwartzman [9], which reduces the space complexity to the equivalent of keeping $O(n \log W/\varepsilon)$ edges, where W is the normalized maximum edge weight, while still providing a $(2+\varepsilon)$-approximation.

The idea is to ensure that the edges incident on each node v that get pushed into the stack have exponentially increasing weights, by factors of $(1+\varepsilon)$. Thus, at most $O(\log_{1+\varepsilon} W) = O((\log W)/\varepsilon)$ edges per node are added to the stack, and the overall number of edges in the stack is at most $O((n \log W)/\varepsilon)$.

To attain this exponential growth, the method is as follows: When reducing the weight of an edge e from each neighboring edge e', we will decide between deducting either w_e or $(1+\varepsilon)w_e$ from the weight $w_{e'}$. In general, this can be any arbitrary decision. This arbitrary choice may seem mysterious, but as we shall see in Section 3, the particular choice is rather natural when considered in terms of LP duality. In the streaming model, this decision is done when we first see $e' = \{u, v\}$ in the stream, as follows.

- If $w_{e'} < (1+\varepsilon) \cdot (\phi_u + \phi_v)$ – i.e., if e' has less than $(1+\varepsilon)$ times of the total weight of the stacked up edges incident on u or v – then we deduct $(1+\varepsilon)w_e$ from $w_{e'}$ for each stacked up edge e incident on u or v. Hence, we effectively reduce the weight of $w_{e'}$ by $(1+\varepsilon) \cdot (\phi_u + \phi_v)$. This implies that we get left with an edge e' of negative weight, which can be ignored without putting it in the stack.
- Otherwise, if $w_{e'} \geq (1+\varepsilon) \cdot (\phi_u + \phi_v)$, we deduct only w_e from $w_{e'}$, for each edge e incident on u or v that is already in the stack. Thus, in total, we deduct only $(\phi_u + \phi_v)$ weight from $w_{e'}$ for the previously stacked edges. Thus, now we have an edge e' whose leftover weight is $w'_{e'} \geq (1+\varepsilon) \cdot (\phi_u + \phi_v) - (\phi_u + \phi_v) = \varepsilon \cdot (\phi_u + \phi_v)$. Then, we add e' to the stack, and thus ϕ_u and ϕ_v increase by $w'_{e'}$. Therefore, both ϕ_u and ϕ_v increase by at least a $(1+\varepsilon)$ multiplicative factor.

The concrete algorithm that formalizes the above scheme is presented in Algorithm 1.

▶ **Observation 2.1.** *When an edge $e = \{u, v\}$ is added to the stack, the value of ϕ_u increases by a $1+\varepsilon$ factor.*

The above observation also implies that the edges incident on each node that are added to the stack have an exponential growth in weight, which directly implies that the total number of edges pushed to the stack cannot exceed $O(n \log_{1+\varepsilon} W) = O((n \log W)/\varepsilon)$. In Section 3, we prove that this algorithm is $2(1+\varepsilon)$-approximate.

3 Simplified Analysis

In this section we present our primal-dual analysis of Algorithm 1, proving it yields a $2(1+\varepsilon)$ approximation of the MWM. Note that the basic algorithm above can be seen as Algorithm 1 run with $\varepsilon = 0$, and so we obtain that the basic algorithm is a 2-approximate algorithm.

Algorithm 1 The Paz-Schwartzman Algorithm [9] with Exponentially Increasing Weights.

```
1: $S \leftarrow emptystack$
2: $\forall v \in V : \phi_v = 0$
3: for $e = \{u, v\} \in E$ do
4:     if $w_e < (1+\varepsilon) \cdot (\phi_u + \phi_v)$ then
5:         continue                                  ▷ skip to the next edge
6:     $w'_e \leftarrow w_e - (\phi_u + \phi_v)$
7:     $\phi_u \leftarrow \phi_u + w'_e$
8:     $\phi_v \leftarrow \phi_v + w'_e$
9:     $S.push(e)$
10: $M \leftarrow \emptyset$
11: while $S \neq \emptyset$ do
12:     $e \leftarrow S.pop()$
13:     if $M \cap N(e) = \emptyset$ then $M \leftarrow M \cup \{e\}$
14: return $M$
```

Primal		Dual	
maximize	$\sum_{e\in E} w_e \cdot x_e$	minimize	$\sum_{v\in V} y_v$
subject to:		subject to:	
$\forall v \in V$:	$\sum_{e\ni v} x_e \leq 1$	$\forall \{u,v\} \in E$:	$y_u + y_v \geq w_{\{u,v\}}$
$\forall e \in E$:	$x_e \geq 0$	$\forall v \in V$:	$y_v \geq 0$

Figure 1 The LP relaxation of MWM and its dual.

▶ **Lemma 3.1.** *The matching M returned by Algorithm 1 is a $2(1+\varepsilon)$ approximation of the maximum weight matching.*

To prove Lemma 3.1 we will rely on a few observations. The first observation our duality-based proofs rely on is that $\vec{\phi_v}$ forms a (nearly) feasible solution to the dual of the LP relaxation of the MWM problem, in Figure 1. Indeed, this fact is not accidental, and it is the underlying reason for the choice of when to decrease weights of an edge neighboring a processed edge e by $(1+\varepsilon)w_e$ or by w_e.

▶ **Observation 3.2.** *The variables $\{\phi_v\}_{v\in V}$ scaled up by $1+\varepsilon$ form a feasible dual solution.*

Proof. Each edge $e = \{u,v\} \in E$ has its dual constraint satisfied by $(1+\varepsilon)\cdot\vec{\phi}$ after inspection; i.e., $w_e \leq (1+\varepsilon)\cdot(\phi_u+\phi_v)$. This constraint is either satisfied before e is inspected, or after performing lines 7 and 8, following which the new and old values of ϕ_u and ϕ_v satisfy $w_e \leq w_e + w'_e = (\phi_u^{old} + \phi_v^{old} + w'_e) + w'_e = \phi_u^{new} + \phi_v^{new} \leq (1+\varepsilon)\cdot(\phi_u^{new}+\phi_v^{new})$. As the ϕ_v variables never decrease, each dual constraint is satisfied by $(1+\varepsilon)\cdot\vec{\phi}$ in the end. ◀

By LP duality, the above implies the following upper bound on the optimal matching.

▶ **Corollary 3.3.** *The weight of any matching M^* satisfies*

$$w(M^*) \leq OPT(LP) \leq (1+\varepsilon)\cdot\sum_v \phi_v.$$

Let $\Delta\phi^e$ be the change to $\sum_{v\in V}\phi_v$ in Lines 7 and 8 of the algorithm during the inspection of edge e. Note that if the algorithm does not reach these lines when inspecting edge e (due to the test in Line 4), then we have $\Delta\phi^e = 0$. By definition, at the time that the algorithm terminates, $\sum_v \phi_v = \sum_e \Delta\phi^e$. The following lemma relates the weight of an edge e to $\Delta\phi^{e'}$ of edges e' incident on a common vertex with e (including e) inspected no later than e.

▶ **Lemma 3.4.** *For each edge $e = \{u, v\} \in E$ added to the stack S, if we denote its preceding neighboring edges by $\mathcal{P}(e) = \{e' \mid e' \cap e \neq \emptyset$, e' inspected no later than $e\}$, then*

$$w_e \geq \frac{1}{2} \cdot \sum_{e' \in \mathcal{P}(e)} \Delta\phi^{e'} = \sum_{e' \in \mathcal{P}(e)} w'_{e'}.$$

Proof. First, we note that $\Delta\phi^e = 2w'_e$, due to Lines 7 and 8, or put otherwise $w'_e = \frac{1}{2} \cdot \Delta\phi^e$. On the other hand, if we denote the values of ϕ_u and ϕ_v before the inspection of e by ϕ^e_u and ϕ^e_v, respectively, we have that $\phi^e_u + \phi^e_v \geq \frac{1}{2} \cdot \sum_{e' \in \mathcal{P}(e) \setminus \{e\}} \Delta\phi^{e'}$. Consequently, we have that indeed

$$w_e = w'_e + (\phi^e_u + \phi^e_v) \geq \frac{1}{2} \cdot \sum_{e' \in \mathcal{P}(e)} \Delta\phi^{e'} = \sum_{e' \in \mathcal{P}(e)} w'_{e'}. \qquad ◀$$

Proof of Lemma 3.1. By the algorithm's definition, every edge ever added to S and not taken into M has a previously-inspected neighboring edge taken into M. So, by Lemma 3.4 and Corollary 3.3 we have that $w(M) = \sum_{e \in M} w_e \geq \frac{1}{2} \sum_e \Delta\phi^e = \frac{1}{2} \sum_v \phi_v \geq \frac{1}{2(1+\varepsilon)} \cdot w(M^*)$. In other words, the matching M output by Algorithm 1 is a $2(1+\varepsilon)$-approximate MWM. ◀

4 Improved Algorithm

The $(2+\epsilon)$-approximate algorithm of the previous section stores $O(n \log W)$ edges for any constant $\epsilon > 0$. To improve the space complexity, we would like to keep only $O(n)$ edges, which is asymptotically the bare minimum necessary for keeping just a matching. For that purpose, we will limit the number of edges incident on each vertex v that are in the stack to a constant $\beta = \frac{3 \log 1/\varepsilon}{\varepsilon} + 1$. When there are more edges, we will take out the earliest one and remove it from the stack. This will be easy to implement using a queue $Q(v)$ for each of vertex v, where we keep the length of the $Q(v)$ capped at β, resulting in $O\big(n \cdot \frac{\log(1/\varepsilon)}{\varepsilon}\big)$ edges stored overall; i.e., $O(n)$ edges, for any constant $\epsilon > 0$. The pseudo-code is presented in Algorithm 2. We will prove in the following that this cannot hurt the approximation factor more than just increasing the parameter ε by a constant factor.

Remark. Paz and Schwartzman[9] used a similar algorithmic idea to keep only $O(n \log n)$ edges in total, instead of $O(n \log W)$ edges.[1] To be precise, they keep $\gamma = \Theta(\log n/\varepsilon)$ edges per node. Unfortunately, this also leads to quite a bit of complications in their analysis, as they need to adapt the local ratio theorem[1, 2] to handle the loss. In a rough sense,

[1] We note that keeping $O(n \log n)$ edges can be done in a much simpler way, by remembering the maximum edge weight W_{max} observed so far in the stream and discarding all edges of weight below $\delta = \varepsilon W_{\max}/(2(1+\varepsilon)n^2)$. This effectively narrows the range of weights that Algorithm 1 sees to $W' = O(n^2/\varepsilon)$, making its related space complexity $O(n \log n)$. On the other hand, ignoring all edges of weight below $\delta \leq \varepsilon W_{\max}/n^2$ can decrease the MWM, $w(M^*)$, by at most $n^2\delta \leq \varepsilon W_{\max} \leq \varepsilon w(M^*)$; that is, a $(1-\varepsilon)$ multiplicative term. Moreover, for each vertex v the edges of weight at most δ can increase ϕ_v by at most $n\delta = \varepsilon W_{\max}/(2(1+\varepsilon)n)$, thus decreasing the effective weight of edges of weight at least δ by no more than $(1+\varepsilon)(\phi_u + \phi_v) \leq \varepsilon W_{\max}/n \leq \varepsilon w(M^*)/n$. The weight of the maximum weight matching M^* under this new weight function is therefore at most $(1-\varepsilon)$ smaller than under w, so running Algorithm 1 on these weights yields a $2(1+O(\varepsilon))$-approximate solution to the MWM under w.

Algorithm 2 The Space-Optimal Algorithm.

```
1: $S \leftarrow$ empty stack
2: $\forall v \in V : Q(v) \leftarrow$ empty queue
3: $\forall v \in V : \phi_v = 0$
4: for $e = \{u, v\} \in E$ do
5:     if $w_e < (1+\varepsilon) \cdot (\phi_u + \phi_v)$ then
6:         continue                                      ▷ skip to the next edge
7:     $w'_e \leftarrow w_e - (\phi_u + \phi_v)$
8:     $\phi_u \leftarrow \phi_u + w'_e$
9:     $\phi_v \leftarrow \phi_v + w'_e$
10:    $S.push(e)$
11:    for $x \in \{u, v\}$ do
12:        $Q(x)$.enqueue$(e)$
13:        if $|Q(x)| > \beta = \frac{3\log(1/\varepsilon)}{\varepsilon} + 1$ then
14:            $e' \leftarrow Q(x).dequeue()$
15:            remove $e'$ from the stack $S$
16: $M \leftarrow \emptyset$
17: while $S \neq \emptyset$ do
18:     $e \leftarrow S.pop()$
19:     if $M \cap N(e) = \emptyset$ then $M \leftarrow M \cup \{e\}$
20: return $M$
```

their argument was that, per step, the process of limiting the queue size to γ creates a loss of $(1 - \exp(-\gamma))$ factor in the approximation in the accountings of the local ratio theorem. Thus, over all the $O(n^2)$ edges in the stream, the loss is $(1 - \exp(-\gamma))^{O(n^2)}$. The fact that m could be $\Omega(n^2)$ is why they had to set $\gamma = \Theta(\log n)$ to make this loss negligible.

Handling the loss caused by this queue-limitation is much simpler with the simple primal-dual argument that we used in Section 3. Furthermore, our analysis will allow us to curtail the per-node queue size to $\beta = O(1)$, while keeping the loss negligible. We now address the loss due to queue size limitation in Lines 11-15, starting with the following observation.

▶ **Observation 4.1.** *Suppose that an edge $e = \{u, v\}$ in the stack gets removed from the stack because another edge $e' = \{u, v'\}$ was pushed to the stack later and made the size of the queue $Q(v)$ exceed β. Then, we say e' evicted e. In that case, $w'_{e'} \geq w'_e/\varepsilon$ if $\varepsilon \leq 1$.*

Proof. Since e was evicted by e', there must have been $\beta - 1$ edges incident on u that arrived after e (following which $\phi_u \geq w'_e$) but before e' which were pushed into the stack. Hence, because of Observation 2.1, we have that before the arrival of e', $\phi_u \geq (1+\varepsilon)^{\beta-1} w'_e \geq w'_e/\varepsilon^2$ (the last inequality holds because $\beta - 1 = \frac{3\log(1/\varepsilon)}{\varepsilon} \geq \frac{2\log(1/\varepsilon)}{\log(1+\varepsilon)}$ for all $\varepsilon \in (0, 1]$). But since e' was added to the stack, we know that $w'_{e'} \geq \varepsilon(\phi_u + \phi'_v) \geq \varepsilon\phi_u \geq w'_e/\varepsilon$. ◀

The following recursive definition will prove useful when bounding the loss due to eviction of edges from the queue.

▶ **Definition 4.2.** *We say an edge e' which was inserted into S was discarded if it was later removed from S, and say the edge was kept otherwise. We say a discarded edge e' was discarded by a kept edge e if e' was evicted by e or if e' was evicted by an edge e'' which was itself later discarded by e. That is, there is a sequence of evictions which starts with e' and* *ends in edge e where in this sequence, each edge is evicted by the next edge in the chain. We denote the set of edges discarded by e by $\mathcal{D}(e)$.*

We now bound the leftover weights of edges discarded by a given edge e.

▶ **Lemma 4.3.** *For all $\varepsilon \leq 1/4$, for each edge $e \in E$, we have $w'_e \geq \frac{1}{4\varepsilon} \cdot \sum_{e' \in \mathcal{D}(e)} w'_{e'}$.*

Proof. The set $\mathcal{D}(e)$ contains at most two edges evicted by e – one for each endpoint of e. By Observation 4.1, we know that every such edge e' evicted by e satisfies $w'_e \geq w'_{e'}/\varepsilon$. Similarly, any edge e' evicted by e can evict at most two edges in $\mathcal{D}(e)$, where each such edge e'' satisfies $w'_{e''} \leq \varepsilon \cdot w'_{e'} \leq \varepsilon^2 \cdot w'_e$, and so on, by induction. Generally, the edges of $\mathcal{D}(e)$ can be partitioned into sets of at most 2^i edges each for all $i \in \mathbb{N}$, where edges e' in the i-th set have $w'_{e'} \leq \varepsilon^i \cdot w'_e$. Summing over all these sets, we find that indeed, as $\varepsilon \leq 1/4$,

$$\sum_{e' \in \mathcal{D}(e)} w'_{e'} \leq \sum_{i=1}^{\infty} (2\varepsilon)^i \cdot w'_e \leq 2\varepsilon \cdot \sum_{i=0}^{\infty} 2^{-i} \cdot w'_e \leq 4\varepsilon \cdot w'_e. \qquad \blacktriangleleft$$

Combining the simple arguments of Section 3 and Lemma 4.3 we obtain the following.

▶ **Theorem 4.4.** *For all $\varepsilon \leq \frac{1}{4}$, Algorithm 2 is $2(1+6\varepsilon)$-approximate.*

Proof. We note that Observation 3.2 (and consequently Corollary 3.3) as well as Lemma 3.4 hold for Algorithm 2, just as they did for Algorithm 1, as their proofs are unaffected by limiting of the queue sizes in Lines 11-15, which is the only difference between these algorithms.

Now, by Lemma 3.4, for each edge $e \in M$, we have $w_e \geq \sum_{e' \in \mathcal{P}(e)} w'_{e'}$. On the other hand, by Lemma 4.3, we have that $4\varepsilon \cdot w_e \geq 4\varepsilon \cdot w'_e \geq \sum_{e' \in \mathcal{D}(e)} w'_{e'}$. Therefore,

$$w_e \cdot (1+4\varepsilon) \geq \sum_{e' \in \mathcal{P}(e)} \left(w'_{e'} + \sum_{e'' \in \mathcal{D}(e')} w'_{e''} \right).$$

But each kept edge e' not added to M is due to a kept edge e which was added to M; that is, some e such that $e' \in \mathcal{P}(e)$. Likewise, each discarded edge is discarded due to some kept edge. Consequently, the right hand side of the above expression summed over all edges of the output matching M is precisely $\sum_{e' \in E} w'_{e'} = \frac{1}{2} \cdot \sum_{e' \in E} \Delta\phi^{e'} = \frac{1}{2} \cdot \sum_{v \in V} \phi_v$, which by Corollary 3.3 yields

$$\sum_{e \in M} w_e \geq \frac{1}{2(1+4\varepsilon)} \cdot \sum_{v \in V} \phi_v \geq \frac{1}{2(1+4\varepsilon)(1+\varepsilon)} \cdot w(M^*) \geq \frac{1}{2(1+6\varepsilon)} \cdot w(M^*). \qquad \blacktriangleleft$$

As the processing time per edge of Algorithm 2 is clearly $O(1)$, we obtain the following.

▶ **Theorem 4.5.** *For any $\varepsilon > 0$, there exists a semi-streaming algorithm computing a $(2+\varepsilon)$-approximation for MWM, using $O(n \log n \cdot \frac{\log(1/\varepsilon)}{\varepsilon})$ space and $O(1)$ processing time.*

Storing ϕ_v values. In the paper we implicitly assumed the maximum edge weight W is poly(n), implying the n dual variables ϕ_v can be stored using $O(n \log n)$ bits. For general W, this space requirement is $\Omega(n \log W)$. We briefly outline how to improve this space usage to $O(\log\log W + n \log n)$ bits by only keeping the *ratio* of these ϕ_v and the maximum weight observed so far, $W_{\max}$. First, by rounding all edge weights down to the nearest integer power of $(1+\varepsilon)$ (increasing the approximation ratio by at most a $(1+\varepsilon)$ multiplicative factor in the process), we can store $W_{\max}$ by simply storing $\log_{1+\varepsilon} W$, using

$O(\log\log_{1+\varepsilon} W) = O\big(\log\log W + \log(1/\varepsilon)\big)$ bits. Next, upper bounding the contribution of edge weights below $\varepsilon W_{\max}/n^2$ to ϕ_v by $\varepsilon W_{max}/n$, we can store ϕ_v using a bit array representing the $O(\log_{1+\varepsilon}(n^2/\varepsilon)) = O\big(\frac{\log n + \log(1/\varepsilon)}{\varepsilon}\big)$ possible values summed this way by ϕ_v. (Note that the values summed by ϕ_v are all distinct by Observation 2.1 and rounding of weights to powers of $(1+\varepsilon)$.) Arguments similar to those of Footnote 1 show that this approach keeps the $(2+O(\varepsilon))$ approximation guarantee, while by the above it only requires $O(\log\log W + n\log n)$ bits of memory for any constant $\varepsilon > 0$. That is, for any $W = 2^{2^{O(n\log n)}}$, this is still $O(n\log n)$ bits.

References

1 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM (JACM)*, 48(5):1069–1090, 2001.

2 Reuven Bar-Yehuda, Keren Bendel, Ari Freund, and Dror Rawitz. Local ratio: A unified framework for approximation algorithms. in memoriam: Shimon Even 1935-2004. *ACM Computing Surveys (CSUR)*, 36(4):422–463, 2004.

3 Michael Crouch and Daniel M Stubbs. Improved Streaming Algorithms for Weighted Matching, via Unweighted Matching. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 96–104, 2014.

4 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

5 Leah Epstein, Asaf Levin, Julián Mestre, and Danny Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011.

6 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2005.

7 Mohsen Ghaffari and David Wajc. Simplified and Space-Optimal Semi-Streaming for $(2+\varepsilon)$-Approximate Matching. *arXiv preprint*, 2018. `arXiv:1701.03730`.

8 Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181. Springer, 2005.

9 Ami Paz and Gregory Schwartzman. A $(2+\varepsilon)$-Approximation for Maximum Weight Matching in the Semi-Streaming Model. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2153–2161, 2017.