CS 671: Graph Streaming Algorithms and Lower Bounds **Rutgers: Fall 2020**

Lecture 4

September 29, 2020

Instructor: Sepehr Assadi *Scribe: Aditi Dudeja*

In this lecture, we will primarily focus on the following paper:

- "Ami Paz, Gregory Schwartzman, A $(2+\epsilon)$-Approximation for Maximum Weight Matching in the Semi-Streaming Model. In SODA 2017."

with additional backgrounds from:

- "Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, Jian Zhang, On Graph Problems in a Semi-streaming Model. In ICALP 2004 and Theor. Comput. Sci. 2005."
- "Michael Crouch, Daniel S. Stubbs, Improved Streaming Algorithms for Weighted Matching, via Unweighted Matching. In APPROX-RANDOM 2014."

1 The Maximum Weight Matching Problem

Let $G = (V, E, w)$ be a simple graph with non-negative edge weights $w(e) \geq 0$ on each edge e. As usual, we denote $n := |V|$ and $m := |E|$. Recall that a *matching* M in a graph G is a set of edges so that no two edge share a vertex. In this lecture, we consider the *maximum weight matching* problem, namely, the problem of finding a matching M in G which maximizes $\sum_{e \in M} w_e$. Throughout this lecture, we shall assume that weight of each edge is bounded by some poly(n) and thus can be represented with $O(\log n)$ bits[1].

Recall that by what we already proved in Lecture 2 even for unweighted matching, there is no hope to find an exact semi-streaming algorithm for maximum weight matching in a single pass. As such, we need to allow for approximation. We will study three different algorithms that compute an approximation to the maximum weight matching in a graph. These algorithms have successively improving approximation ratios, but as we shall see that the underlying techniques for analyzing these algorithms is similar: *charging arguments*.

Warm-Up: Unweighted matching via a charging argument. Recall the following 2-approximation semi-streaming algorithm for unweighted matching from Lecture 2:

Algorithm 1. A semi-streaming algorithm for 2-approximation of maximum cardinality matching.

1. Let $M \leftarrow \emptyset$.
2. For any edge (u, v) in the stream, add (u, v) to M if both u and v are unmatched in M.
3. Return M.

[1] This assumption is not necessary for any of the algorithms in this lecture and can be lifted using a standard idea: ignore any edge with weight $< W/n^2$ where W is the maximum-weight edge and then scale back any remaining edge by W/n^2 to make them all poly(n)-bounded—this can only change the optimal solution by a negligible amount.

We had discussed in Lecture 2, that this algorithm gives a 2-approximation to the maximum cardinality matching considering it outputs a maximal matching. Let us now see a different proof for this as a warm-up for charging arguments done in this lecture.

Lemma 1. *Algorithm 1 outputs a 2-approximate maximum (cardinality) matching.*

Proof. Let $M^\star$ be a maximum cardinality matching of G and M be the output matching of $M^\star$. Define the following mapping (function) $\sigma : M^\star \to M$:

- For any edge $e \in M^\star \cap M$, $\sigma(e) = e \in M$.
- For any edge $e \in M^* \setminus M$, there should be an edge f incident on one of the vertices of e as otherwise the algorithm would have added e to M as well; in this case, let $\sigma(e) = f$ (breaking the ties arbitrary).

In both cases above, we say that the edge $e \in M^\star$ is *charged* to the edge $\sigma(e) \in M$ (or alternatively, $\sigma(e) \in M$ is *blamed* for the edge $e \in M^\star$). Now notice that any edge in M can be charged at most twice as it is incident on at most two edges of M^* and we only charge an edge to an incident edge. In other words, for any edge $f \in M$, $|\{e \in M^\star \mid \sigma(e) = f\}| \leq 2$. This immediately means that $|M^\star| \leq 2 \cdot |M|$ as every edge of $M^\star$ should be charged to some edge of M while no edge in M is charged more than twice. This concludes the proof. □

Remark. The charging argument in Lemma 1 is a very simple argument which was done in an "overly formal" way for completeness. In the remainder of the lecture, we will simple define the charging scheme and the underlying mapping (function) σ will be implicit in the proof.

For the remainder of this lecture, we go back to the *weighted* matching problem. In this case, Algorithm 1 no longer works: an adversary can add heavier edges so that while $e \in M$ is charged by at most two edges of optimmum, the weight it is charged may be much greater than $2 \cdot w(e)$. An example is given in Figure 1.

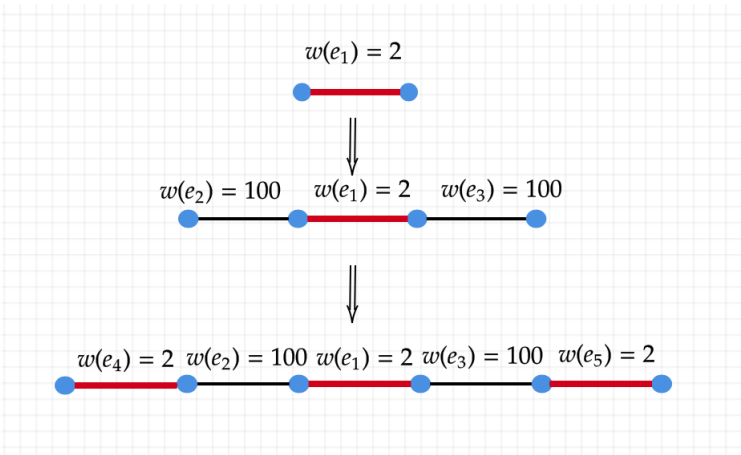

Figure 1: Algorithm 1, first includes e_1 into the matching M, since both endpoints of e_1 are unmatched. When e_2 and e_3 arrive, they are ignored by the algorithm, even though they are significantly heavier than e_1. When e_4 and e_5 arrive, they are included in the matching because their endpoints are unmatched. The final matching $M = \{e_1, e_4, e_5\}$ is much lighter than the optimal matching $M^\star = \{e_2, e_3\}$

.

Therefore, while the problem of 2-approximating the maximum cardinality matching is almost a trivial problem in the semi-streaming setting, approximating the maximum weight matching has turned out to be significantly more challenging. In this lecture, we see several interesting algorithms for this problem and conclude with the state-of-the-art $(2 + \varepsilon)$-approximation algorithm of Paz and Schwartzman [7].

2 A Simple 6-Approximation Semi-Streaming Algorithm

Having seen Algorithm 1, perhaps the simplest fix to try would be to modify the matching M when a heavy edge arrives: that is, suppose $e \in M$, and an edge e' arrives in the stream such that e' is sufficiently heavier than e, then the right thing to do would be to remove e and include e' in the matching. This is exactly what the 6-approximation algorithm of Feigenbaum et.al. [4] does:

Algorithm 2. A single-pass semi-streaming algorithm for 6-approximation of maximum weight matching.

(i) Let $M \leftarrow \emptyset$.

(ii) When a new edge e arrives in the stream, compare e with $C(e)$ where $C(e) = \{e' \in M \mid e \cap e' \neq \emptyset\}$[a].

(a) If $w(e) > 2 \cdot w(C(e))$, then $M \leftarrow M \cup \{e\} \setminus C(e)$.

(b) if $w(e) \leq 2 \cdot w(C(e))$, then ignore e.

[a] These are the at most two edges incident on e already in M.

At first glance, Algorithm 2 may seem "overly conservative" in updating the matching M; after all why not changing e with $C(e)$ even when $w(e) > w(C(e))$ (instead of $w(e) > 2w(C(e))$ currently)? However, it is easy to see that such an algorithm is doomed to fail: simply consider the case when the input G is a path with increasing weights in order of arrival, say, edges with weight $1, 2, 3, \ldots, W$—the new algorithm only picks the very last edge in M while the graph has a matching of weight $\Omega(W^2)$! In the following, we prove that this simple overly conservative update rule however leads to a constant factor approximation.

Lemma 2. *Let $M^\star$ be an optimal matching of the input graph G. Then, $w(M) \geq \frac{1}{6} \cdot w(M^\star)$.*

Proof. We need to introduce some notation:

- For an edge $e \in M$, define $C_0(e) = \{e\}$.
- For $i > 1$, consider any $e' \in C_{i-1}(e)$. Then, e' may be responsible for kicking out up two edges from M when it is added to M. Let $C_i(e)$ be the collection of edges that are kicked out by edges from $C_{i-1}(e)$.
- Let $T(e) := \bigcup_{i \geq 1} C_i(e)$, i.e., all the edges that were directly or indirectly got kicked out by e.

We start by proving that the total weight of the edges got kicked out by an edge e is at most equal to $w(e)$ using the conservative updating rule of the algorithm.

Claim 3. *For any $e \in M$, $w(T(e)) \leq w(e)$.*

Proof. Consider any $i > 0$, and let $e' \in C_{i-1}(e)$. Then, consider the edges e_2 and e_3 kicked out by e'. Observe that $w(e') \geq 2(w(e_2) + w(e_3))$ by the updating rule. Further, each edge is kicked out at most once. This implies that $2 \cdot w(C_i(e)) \leq w(C_{i-1}(e))$ and thus inductively, $w(C_i(e)) \leq 2^{-i} \cdot w(C_0(e)) = 2^{-i} \cdot w(e)$. Consequently, we have that:

$$w(T(e)) = \sum_{i \geq 1} w(C_i(e)) \leq \sum_{i \geq 1} 2^{-i} \cdot w(e) \leq w(e),$$

as desired. □ Claim 3

We now give a charging scheme to prove Lemma 2. Let $M^\star$ be an optimal matching of the input graph G. We will charge $w(M^\star)$ to the edges of M, so that each edge $e \in M$ gets weight at most $6 \cdot w(e)$. Similar to what we argued in Lemma 1, this necessarily implies that $w(M^\star) \leq 6 \cdot w(M)$, thus proving the lemma.

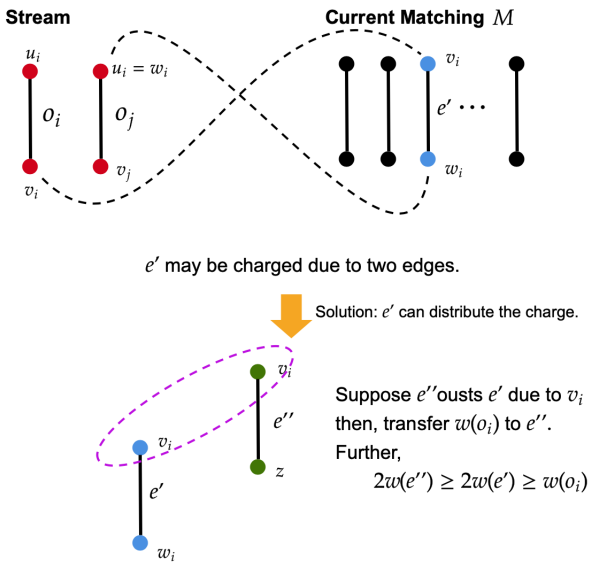

Figure 2: Suppose edge $e' = (v_i, w_i)$ is already in the matching. At some point in the stream, edge $o_i = (u_i, v_i) \in M^\star$ arrives, however is ignored by the algorithm since $2 \cdot w(e') \geq w(o_i)$. Similarly, an edge $o_j = (w_i, v_j)$ arrives and is ignored by the algorithm due to (w_i, v_i). In this case, e' could potentially be held responsible for two edges, and therefore it tries to distribute its charge.

As before, we charge any edge $e' = (u, v) \in M^\star \cap M$ to itself in M, which clearly satisfies the charging bound. Now consider $e' = (u, v) \in M^\star \setminus M$. There are two possible scenarios for the fate of (u, v):

1. **Edge (u, v) was never included in M**:
   In this case, there are at most two edges e_2 and e_3 that are incident on u and v respectively, that prevent (u, v) from being included into M (due to $2 \cdot (w(e_2) + w(e_3)) \leq w(e')$). In this case, we charge e_2 and e_3 with weights $\frac{w(e_2) \cdot w(e')}{w(e_2)+w(e_3)}$ and $\frac{w(e_3) \cdot w(e')}{w(e_2)+w(e_3)}$, respectively. This way, we charge all of $w(e')$ to e_2 and e_3 such that each gets charged a weight of at most $2 \cdot w(e_2)$ and $2 \cdot w(e_3)$, respectively.
2. **Edge (u, v) was included in M but was then removed**:
   Note that in this case, $(u, v) \in T(e)$ for some $e \in M$.

Essentially, in both the above cases, we are charging $w(e')$ to some edge in $T(e)$ for some $e \in M$ and not to M still. Moreover, in the above scheme, each edge $o = (x, y) \in T(e)$ may get charged twice (for each of its endpoints, see Figure 2 for an illustration). In this case, o tries to distribute its charge: suppose $o = (x, y)$ is charged at most $2 \cdot w(o)$ due to $o_1 = (z, y) \in M^\star$ and then o is ousted from M by $o_2 = (w, y)$, then o transfers the charge due to o_1 onto o_2. In this scheme o_2 is charged weight at most $2 \cdot w(o) \leq 2 \cdot w(o_2)$. The only way o would be charged twice is if it is never kicked out from M, that is if $o \in M$. Consequently, we can deduce that each $e' \in T(e)$ for $e \in M$ is charged at most $2 \cdot w(e')$ and each $e \in M$ is charged at most $4 \cdot w(e)$. This implies that:

$$\begin{aligned} w(M^\star) &\leq 2 \sum_{e \in M} w(T(e)) + 4 \sum_{e \in M} w(e) \\ &\leq 6 \sum_{e \in M} w(e) = 6 \cdot w(M). \qquad \text{(by Claim 3)} \end{aligned}$$

This proves the lemma. □

Finally, it is immediate that Algorithm 2 uses only $O(n)$ space as it only maintains a single matching at all times. This allows us to conclude the following theorem from [4].

Theorem 4 ([4]). *There is a semi-streaming 6-approximation algorithm for maximum weight matching.*

3 A $(4+\epsilon)$-Approximation Semi-Streaming Algorithm

In this section, we will discuss an algorithm due to Crouch and Stubbs [3] that achieves a $(4+\epsilon)$-approximation to the maximum weight matching in the input graph G. This algorithm reduces the problem of maximum weight matching to the problem of maximum cardinality matching, and makes $O\left(\frac{\log n}{\varepsilon}\right)$ calls to any α-approximation maximum cardinality matching algorithm; finally, it combines these matchings greedily to obtain a $(2+\varepsilon)\alpha$-approximation algorithm.

Algorithm 3. An algorithm that computes $(2+\epsilon)\,\alpha$-approximation to weighted matching *given oracle access* to any α-approximation algorithm for unweighted matchings.

(*i*) Let $E_i = \left\{e \in E \mid w(e) \geq (1+\epsilon)^i\right\}$ and define $G_i := (V, E_i)$[a].

(*ii*) *In parallel*, compute an α-approximate maximum cardinality matching C_i of G_i (using the oracle).

(*iii*) Let $M \leftarrow \emptyset$ and $W = \max\{w(e) \mid e \in E\}$.

(*iv*) For $i = \log_{1+\epsilon} W$ to 1, consider $e \in C_i$ and let $M \leftarrow M \cup \{e\}$ if both the endpoints of e are unmatched in M; return M at the end.

[a] Note that $E_1, E_2, \cdots$ are not a partition of the edges but rather $E_1 \subseteq E_2 \subseteq \cdots$.

We now prove that Algorithm 3 outputs a $(2+\varepsilon)\alpha$-approximation to the maximum weight matching of G.

Lemma 5. *Let $M^\star$ be an optimal matching of the input graph G. For any choice of the α-approximation algorithm for unweighted matching in Algorithm 3, $w(M^\star) \leq 2 \cdot (1+\varepsilon)\alpha \cdot w(M)$.*

Proof. For any $1 \leq i \leq \log_{1+\epsilon} W$, we define:

- Define $M_i = M \cap E_i$ and $M_i^\star = M^\star \cap E_i$.

We first claim that *size* (and not weight) of M_i itself is at most a 2α factor smaller than size of $M_i^\star$. Formally,

Claim 6. *For each $1 \leq i \leq \log_{1+\epsilon} W$, we have $|M_i| \geq \frac{1}{2\alpha}|M_i^\star|$.*

Proof. As C_i is an α-approximate (unweighted) matching of G_i, and $M_i^\star \subseteq G_i$, we have $|C_i| \geq \frac{1}{\alpha} \cdot |M_i^\star|$. Moreover, the algorithm builds the matching M from C_i's greedily. Thus, the same argument as in Lemma 1, we have $|M_i| \geq 1/2 \cdot |C_i|$. Combining these, we have that $|M_i| \geq \frac{1}{2\alpha}|M_i^\star|$. □ Claim 6

We are now ready to perform our charging argument.

Claim 7. *There is a mapping $\sigma : M^\star \to M$ such that:*

- *for any $e \in M$, $|\sigma^{-1}(e)| \leq 2\alpha$,*
- *for any $e' \in M^\star$, $w(e') \leq (1+\epsilon) \cdot w(\sigma(e'))$.*

Proof. Starting at the maximum value of i, we assign edges in $M_i^\star$ to edges in M_i. For any $e \in M_i^\star$, we let $\sigma(e)$ be an edge in M_i that has fewer than 2α edges already assigned to it. Claim 6 ensures that such an edge is always available. This guarantees the first property. Moreover, since we process $M^\star$ in the decreasing order of i, edges of $M_1^\star$ are mapped to M_1, edges of $M_2^\star \setminus M_1^\star$ to M_2, and generally $M_i^\star \setminus M_{i-1}^\star$ to M_i. This implies that $w(e') \leq (1+\epsilon)\, w(\sigma(e'))$. □ Claim 7

Finally, by Claim 7 (for first and second inequalities),

$$w(M^\star) = \sum_{e \in M^\star} w(e) \leq \sum_{e \in M^\star} w(\sigma(e))\,(1+\epsilon) \leq 2\alpha\,(1+\epsilon) \sum_{e \in M} w(e) = 2\alpha\,(1+\epsilon)\, w(M)\,,$$

concluding the proof. □

We can now conclude the following theorem from [3].

Theorem 8 ([3])**.** *There is a semi-streaming $(4+\varepsilon)$-approximation algorithm for maximum weight matching.*

Proof. We run Algorithm 3 (by re-parameterizing ε witth $\varepsilon/4$) with the simple greedy 2-approximation algorithm for unweighted matching (Algorithm 1) as the oracle. This requires running $O(\varepsilon^{-1} \cdot \log n)$ instances of Algorithm 1, each with $O(n)$ space, leading to $O(\varepsilon^{-1} \cdot n \log n)$ space overall. The correctness now follows from Lemma 5. □

Remark. The Crouch-Stubbs technique [3] gives a very general and "low overhead" way of obtaining an approximate algorithm for weighted matching from approximation algorithms for unweighted matching, which is applicable in many settings beyond streaming algorithms as well. Similar-in-spirit reductions from weighted to unweighted matchings, with "higher overhead", are also given in [5] that allow for transforming a 2-approximation of unweighted matching to a $(1+\varepsilon)$-approximation of weighted matching by running the unweighted matching oracle *adaptively* for $(1/\varepsilon)^{O(1/\varepsilon^2)}$ times—such an approach can be used to obtain *multi-pass* algorithms for weighted matching (with 1 pass per each adaptive call to the oracle, thus $(1/\varepsilon)^{O(1/\varepsilon^2)}$ passes overall).

4 A $(2+\epsilon)$-Approximation Semi-Streaming Algorithm

Finally, we get to the breakthrough result of Paz and Schwartzman who obtained a $(2+\varepsilon)$-approximation semi-streaming algorithm for weighted matching [7]. The authors of [7] adapt a non-streaming algorithm based on the local-ratio technique of Bar-Yehuda and Even [2] to the streaming setting. In Section 4.1 we state the local ratio theorem for weighted matching that gives us a 2-approximation algorithm in the sequential setting. In the subsequent section, we then show how [7] adapt this algorithm to the streaming setting. We then prove the space complexity and the approximation ratio guaranteed by this algorithm via a charging argument given by Ghaffari and Wajc [6].

4.1 A Simple Local-Ratio Algorithm for Weighted Matching

We first state and prove the local-ratio theorem for weighted matching that will be used to build our final streaming algorithm:

Proposition 9 (cf. [1])**.** *Let $G = (V, E, w)$ be a weighted graph. Let w_1 and $\bar{w}_1$ be two arbitrary weight functions (possibly even negative) on the edges of G such that $w = w_1 + \bar{w}_1$. Let M be an α-approximation to the maximum weight matching of G with respect to w_1 and $\bar{w}_1$* simultaneously*. Then, M is also an α-approximation to the maximum weight matching of G with respect to w.*

Proof. Let $M^\star$, M_1, and $\bar{M}_1$ be the optimal weight matchings of G with respect to w, w_1, and $\bar{w}_1$, respectively.

$$\begin{aligned} w(M^\star) &= w_1(M^\star) + \bar{w}_1(M^\star) \leq w_1(M_1) + \bar{w}_1(\bar{M}_1) \\ &\qquad \text{(Since } M_1 \text{ and } \bar{M}_1 \text{ are optimal with respect to } w_1 \text{ and } \bar{w}_1\text{, respectively.)} \\ &\leq \alpha \cdot w_1(M) + \alpha \cdot \bar{w}_1(M) = \alpha \cdot w(M), \\ &\qquad \text{(Since } M \text{ is an } \alpha\text{-approximate solution for both } w_1 \text{ and } \bar{w}_1\text{.)} \end{aligned}$$

proving the result. □

A 2-approximation local-ratio algorithm. Proposition 9 naturally suggests a recursive algorithm for weighted matching. Let e be any arbitrary edge in G with $w(e) > 0$ and define $N^+(e)$ to be the set of edges that share an endpoint with e plus e itself. Define the weight function $\bar{w}_1$ as follows: for any edge $f \in E$:

$$\bar{w}_1(f) = \begin{cases} w(e) & \text{if } f \in N^+(e) \\ 0 & \text{otherwise} \end{cases}.$$

Also, define $w_1 = w - \bar{w}_1$. Now suppose we *recursively* find a 2-approximate matching M_1 for G under the weight function w_1 (and letting $M_1 = \emptyset$ if $w_1(e) \leq 0$ for all $e \in G$). To apply Proposition 9, we need M_1 to also be an α-approximate matching of G under $\bar{w}_1$. Now, notice that $\bar{w}_1$ gives the same weight of $w(e)$ to all edges in $N^+(e)$ and is otherwise 0. Moreover, all these non-zero weight edges are incident on e and thus the maximum weight matching of G under $\bar{w}_1$ has weight at most $2w(e)$. This means that if $M_1 \cap N^+(e)$ is non-empty, M_1 is already a 2-approximate matching for $\bar{w}_1$ also as $\bar{w}_1(M_1) \geq w(e)$ in that case. Otherwise, both endpoints of e are unmatched in M_1 and thus we can consider directly adding e to M_1 to have the matching $M = M_1 \cup \{e\}$: As $w_1(e) = 0$, we have $w_1(M_1 \cup \{e\}) = w_1(M_1)$ (hence M is still a 2-approximate matching for w_1) and $\bar{w}_1(M_1 \cup \{e\}) = \bar{w}_1(e) = w(e)$ (hence M is now a 2-approximate matching for $\bar{w}_1$ also). By Proposition 9, in both case, we obtain a 2-approximate maximum weight matching of G under w as well.

Let us now define the same algorithm *sequentially* instead of recursively as follows:

Algorithm 4. A simple 2-approximation local-ratio algorithm for maximum weight matching.

(*i*) Let $w_1 = w$, $i = 1$, and define the *stack* S initialized to be empty.

(*ii*) Iterate over the edges e one by one in an arbitrary order:

- (a) If $w_i(e) \leq 0$ go to the next edge directly.
- (b) Otherwise, let $e_i = e$ and *push* e_i to the stack S;
- (b) For each $f \in N^+(e_i)$, let $w_{i+1}(f) \leftarrow w_i(f) - w_i(e_i)$.
- (c) Set $i \leftarrow i + 1$ and go to the next edge.

(*iii*) Unwind S, and create the matching M greedily in the stack order.

It is easy to see that Algorithm 4 is just a different implementation of the recursive algorithm:

- For any i in Algorithm 4, we can define the weight function $\bar{w}_{i+1}$ to be

$$\bar{w}_{i+1}(f) = \begin{cases} w_i(e) & \text{if } f \in N^+(e_i) \\ 0 & \text{otherwise} \end{cases},$$

  which results in $w_i = w_{i+1} + \bar{w}_{i+1}$. This is exactly the same weight function in the recursive algorithm.
- Unwinding the stack and creating the matching M at the end has the same exact effect as checking if any endpoint of edge e_i is already matched by the matching M_i of w_{i+1} (in which case the recursive algorithm does not change M_i), or not (in which case the recursive algorithm let $M \leftarrow M_i \cup \{e_i\}$).

As a result, we can prove the 2-approximation guarantee of Algorithm 4 verbatim as before.

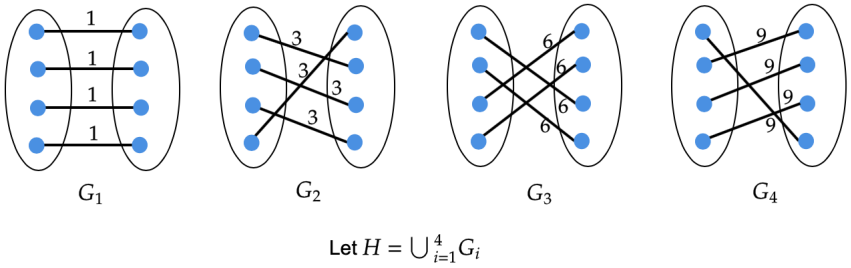

Figure 3: Consider the graph H, whose edges arrive as a stream: $E(G_1) \circ E(G_2) \circ E(G_3) \circ E(G_4)$. The algorithm initially adds all the edges of $E(G_1)$ to the stack. After processing $E(G_1)$, the weight of edge $e \in E(G_i)$ for $i \geq 2$ is reduced by 2. This implies that $w_5(e) = 1$ for all $e \in E(G_2)$. It follows that edges of $E(G_2)$ are also added to the stack. After processing $E(G_2)$, $w_9(e) = 2$ for $e \in E(G_3)$. This implies that when $E(G_3)$ arrive, all of these are added to the stack and so on until the stack size becomes $\Omega(n^2)$.

4.2 Adapting the Local-Ratio Algorithm to the Semi-Streaming Model

Let us discuss what are the potential obstacles to adapting Algorithm 4 to the semi-streaming setting, and how [7] overcome these:

1. The main obstacle is the fact that size of the stack can grow to $\Omega(n^2)$. An example of this is shown in Figure 3. To overcome this, authors of [7] make sure to only add an edge (u, v) to the stack if it causes the total weight of the edges around u and v that were added to the stack to grow by a factor of $(1 + \epsilon)$. Since we assume that all edges have weights that are polynomial in n, it follows that the total number of edges added per vertex is $O\left(\frac{\log n}{\epsilon}\right)$. This will allows us to bound the space by $O\left(\varepsilon^{-1} \cdot n \log n\right)$.
2. The iterative weight reduction done by Algorithm 4 cannot be implemented directly in the semi-streaming setting, since some edges $f \in N^+(e_i)$ may arrive only after the edge e_i has been processed. In order to take care of this, the algorithm maintains *potentials* at each vertex, where $\phi_i(v) = \sum_{v \cap e_j \neq \emptyset, j \leq i} w_j(e_j)$. As a result now, for any edge in the stream, the weight reduction process can be done retroactively.

With these solutions in mind, we now give the algorithm that we want to implement:

Algorithm 5. A semi-streaming $(2 + \varepsilon)$-approximation algorithm for maximum weight matching.

1. Define a *stack* $S = \emptyset$. Also, for each vertex $v \in V$, let $\phi_v \leftarrow 0$ initially.
2. For any arriving edge $e = (u, v) \in E$ in the stream:
   (a) If $w(e) \leq (1 + \epsilon)(\phi_u + \phi_v)$, then move onto the next edge in the stream.
   (b) Otherwise, push e to the stack S and set:
   $$w'(e) = w(e) - (\phi_u + \phi_v), \qquad \phi_u \leftarrow \phi_u + w'(e), \qquad \phi_v \leftarrow \phi_v + w'(e).$$
3. Unwind the stack, and create the matching M greedily in the stack order.

Before we prove the approximation guarantee of Algorithm 5, we will discuss what exactly this algorithm is doing by comparing it with Algorithm 4. Consider Algorithm 4, and suppose the stack S, right before

unwinding contains edges $\{e_1, \cdots, e_k\}$, where e_i was added before e_j to the stack if $i < j$. As discussed before, each of these e_i's is defining a weight function on the edges of the graph. These weight functions $w_1, \cdots, w_{k+1}$ are defined inductively, starting from $w_1 = w$, and

$$w_{i+1}(e) = \begin{cases} w_i(e) & \text{if } e \notin N^+(e_i) \\ w_i(e) - w_i(e_i) & \text{if } e \in N^+(e_i) \end{cases}.$$

On the other hand, this is not quite how the weight functions are being defined by Algorithm 5. We first state what are the weight functions being imposed on G by the edges in the stack, and then give some intuition behind why these are the right weight functions. Let $S = \{e_1, e_2, \cdots, e_k\}$, where e_i is pushed in the stack before e_j if $i < j$. For $i \in [k+1]$, let w_{i+1} be the weight function imposed by e_i, starting with $w_1 = w$; then,

$$w_{i+1}(e) = \begin{cases} w_i(e) & \text{if } e \notin N^+(e_i) \quad (1a) \\ w_i(e) - w_i(e_i) \text{ or } w_i(e) - (1+\epsilon) \cdot w_i(e_i) & \text{if } e \in N^+(e_i) \quad (1b) \end{cases}$$

While the definition in Case (1b) may seem arbitrary at the first glance, there is a valid reason behind this. To see this, it is important to think of condition (2a) of Algorithm 5 as analogous to condition ((*ii*)a) of Algorithm 4. Suppose at some time after we have processed edge e_i, and before we are processing e_{i+1}, we consider an edge e. Let $S_e = \{e_1, \cdots, e_i\} \cap N(e)$; then, condition (2a) states the following:

- If $w(e) - \sum_{e_j \in S_e} (1+\epsilon) \cdot w_j(e_j) \leq 0$, then ignore e. So, for such e's we may think of $w_{j+1}(e)$ as being defined inductively by subtracting $(1+\epsilon)w_j(e_j)$ from $w_j(e)$ iff $e \in N^+(e_j)$. Then, analogous to condition (*ii*)a of Algorithm 4, we may then say that we are ignoring e because $w_{i+1}(e) \leq 0$.
- On the other hand, if $w(e) - \sum_{e_j \in S_e} (1+\epsilon) \cdot w_j(e_j) > 0$ we define $w_{i+1}(e_i)$ by subtracting $\sum_{e_j \in S_e} w_j(e_j)$ from $w(e)$. To see this, consider step (2b) of Algorithm 5: $\phi(u) + \phi(v) = \sum_{e_j \in S_e} w_j(e_j)$, $w'(e) = w(e) - \phi(u) - \phi(v) = w(e) - \sum_{e_j \in S_e} w_j(e_j)$ is just $w_i(e_i)$ (so, in this case, $w_i(e_i)$ is defined as in Algorithm 4). Then, $w'(e) = w_i(e_i)$ is added to ϕ_u and ϕ_v, thus defining $w_{i+1}(e_i)$ implicitly.

With this discussion in mind, we now move on to proving the approximation guarantees.

Lemma 10. *Let $M^\star$ be a maximum weight matching in G and M be the output of Algorithm 5. Then, $w(M^\star) \leq 2(1+\epsilon)w(M)$.*

Proof. Our proof technique will be similar to the charging argument that we have used in past. In particular, we show that we can charge the weights of the edges of $M^\star$ to the edges of M, so that each $e \in M$ is charged with weight at most $2(1+\epsilon)w(e)$.

We prove this inductively. Before this, some notation is in order. Let $S = \{e_1, \cdots, e_k\}$ be the state of the stack right before the last step of the algorithm. Let w_{i+1} be the weight function defined by edge e_i on the graph for $i \in [k]$, and let $w_1 = w$. Let M_r be the matching maintained by the algorithm after e_r is unwound from the stack. Let $M_1 = M$, and $M_{k+1} = \emptyset$. We have the following inductive statement, whose $r = 1$ case proves our charging argument.

Claim 11. *For $r \in [k+1]$, we can charge $w_r(M^*)$ to M_r so that $e \in M_r$ gets weight at most $2(1+\epsilon)w_r(e)$.*

Proof of Claim 11. The statement holds for $r = k+1$ since $w_{k+1}(M^*) = 0$. Assume that the statement holds for M_{r+1}. That is, we can charge $w_{r+1}(M^*)$ to M_{r+1} so that each $e \in M_{r+1}$ gets weight at most $2(1+\epsilon)w_{r+1}(e)$. We want to show that this is true for M_r as well.

Observe that $w_r(e) \neq w_{r+1}(e)$ iff $e \in N^+(e_r)$. This is implied by the definition of w_{r+1} (see (1a) and (1b)). So, for all $f \in M^* \setminus N^+(e_r)$, we have in fact charged $w_r(f) = w_{r+1}(f)$ to the edges of $M_{r+1} \subseteq M_r$ by the induction hypothesis. In M^* there are at most two edges, $f', f'' \in N^+(e_r)$, whose weights we are now

required to charge to M_r. For f' and f'', we have already, by induction hypothesis, charged $w_{r+1}(f')$ and $w_{r+1}(f'')$ to edges of M_r. So, the remaining charge is at most $2(1+\epsilon)w_r(e_r)$ since

$$w_r(f') - w_{r+1}(f') + w_r(f'') - w_{r+1}(f'') \leq 2(1+\epsilon)w_r(e_r),$$

by (1b). To distribute this remaining charge, we consider the following two cases:

1. **Edge e_r is included in the matching M_r.** Considering e_r is in M_r but not M_{r+1} as it is only processed from the stack in step r, there is no charge on e_r currently. So we can charge the weight on f' and f'' which is at most $2(1+\epsilon)w_r(e_r)$ to e_r, satisfying the induction hypothesis for r.
2. **Edge e_r is not included in the matching M_r.** In this case, there is an edge $e' \in N^+(e_r)$ that is already in M_{r+1}. By induction hypothesis, the total charge on e' at the moment is $\leq 2(1+\epsilon)w_{r+1}(e')$. So, it can take an additional charge of $2(1+\epsilon)w_r(e_r)$, and the net charge on it will be at most $2(1+\epsilon)w_r(e')$ because $w_{r+1}(e') + w_r(e_r) \leq w_r(e')$ by the definition of w_{r+1} in (1b). So, the remaining weight due to f' and f'' may be charged to e' in this scenario. □ Claim 11

Claim 11 for $r = 1$ shows that each edge $e \in M$ is charged a weight of at most $2(1+\epsilon)w(e)$, thus implying that $w(M^\star) \leq 2(1+\epsilon)w(M)$ and concluding the proof of Lemma 10. □

Lemma 12. *The space complexity of Algorithm 5 is* $O\left(\epsilon^{-1} \cdot n \log n\right)$.

Proof. Suppose that the maximum weight of any edge is W which is some poly(n) by our assumption at the beginning of the lecture. Then, the maximum possible value ϕ_u can attain for any u, is at most $n \cdot W$. Moreover, each time an edge incident on u is added to the stack, ϕ_u grows by a $(1+\epsilon)$ factor. Hennce, the number edges added per vertex can be at most $\log_{1+\epsilon}(n \cdot W)$ which implies the space of $O\left(\varepsilon^{-1} n \cdot \log n\right)$. □

We can now conclude the following theorem from [7].

Theorem 13 ([7])**.** *There is a semi-streaming* $(2+\epsilon)$*-approximation algorithm for maximum weight matching.*

References

[1] R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz. Local ratio: A unified framework for approxmation algrithms in memoriam: Shimon even 1935-2004. *ACM Comput. Surv.*, 36(4):422–463, 2004. 6

[2] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25(27-46):50, 1985. 6

[3] M. Crouch and D. M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014. 5, 6

[4] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005. 3, 5

[5] B. Gamlath, S. Kale, S. Mitrovic, and O. Svensson. Weighted matchings via unweighted augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 491–500, 2019. 6

[6] M. Ghaffari and D. Wajc. Simplified and space-optimal semi-streaming (2+epsilon)-approximate matching. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 13:1–13:8, 2019. 6

[7] A. Paz and G. Schwartzman. A $(2+\epsilon)$-approximation for maximum weight matching in the semi-streaming model. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2153–2161. SIAM, 2017. 2, 6, 8, 10