Metatheorems for Dynamic Weighted Matching

Daniel Stubbs[*1] **and Virginia Vassilevska Williams**[†2]

1 **Computer Science Department, Stanford University, USA**
dstubbs@stanford.edu
2 **Computer Science Department, Stanford University, USA**
virgi@cs.stanford.edu

Abstract

We consider the maximum weight matching (MWM) problem in dynamic graphs. We provide two reductions. The first reduces the dynamic MWM problem on m-edge, n-node graphs with weights bounded by N to the problem with weights bounded by $(n/\varepsilon)^2$, so that if the MWM problem can be α-approximated with update time $t(m, n, N)$, then it can also be $(1+\varepsilon)\alpha$-approximated with update time $O(t(m, n, (n/\varepsilon)^2)\log^2 n + \log n \log\log N))$. The second reduction reduces the dynamic MWM problem to the dynamic maximum cardinality matching (MCM) problem in which the graph is unweighted. This reduction shows that if there is an α-approximation algorithm for MCM with update time $t(m, n)$ in m-edge n-node graphs, then there is also a $(2+\varepsilon)\alpha$-approximation algorithm for MWM with update time $O(t(m, n)\varepsilon^{-2}\log^2 N)$. We also obtain better bounds in our reductions if the ratio between the largest and the smallest edge weight is small. Combined with recent work on MCM, these two reductions substantially improve upon the state-of-the-art of dynamic MWM algorithms.

1998 ACM Subject Classification F2.2 Nonnumerical Algorithms and Problems

Keywords and phrases dynamic algorithms, maximum matching, maximum weight matching

Digital Object Identifier 10.4230/LIPIcs.ITCS.2017.58

1 Introduction

The maximum matching problem is one of the most basic problems in algorithms. Starting from the 1950s, a series of influential papers (among others, [15, 13, 20, 23, 9, 14, 17, 19]) culminated in very efficient algorithms for the problem in n node m edge graphs. There are two versions of the problem: the unweighted version in which one needs to return a maximum cardinality matching (MCM), and the weighted version, in which the input graph has weights on its edges and one seeks a matching of maximum weight (MWM).

For MCM in general graphs, the best known algorithms are the Micali-Vazirani [18] $O(m\sqrt{n})$ algorithm and the $O(n^\omega)$ time algorithm of Mucha and Sankowski [19] (see also [12, 23, 20]), where $\omega < 2.373$ is the matrix multiplication exponent [26, 16]. For the special case of bipartite graphs, a recent breakthrough result of Madry [17] achieved a runtime of $\tilde{O}(m^{10/7})$. For the maximum weight matching problem, the best known running times are $O(m\sqrt{n}\log(nN))$ [9, 10] and $\tilde{O}(Nn^\omega)$ [25] where N is the largest edge weight.

Graphs in applications, however, are dynamic by nature. Edges and vertices fail and new ones are introduced. Because of this, more resilient, *dynamic* algorithms are desired. Such algorithms can update the solution efficiently when an edge insertion or deletion occurs.

* Supported by an NSF Graduate Fellowship.
† Supported by NSF Grants CCF-1417238, CCF-1528078 and CCF-1514339, and BSF Grant BSF:2012338.

Maintaining an exact solution to the maximum matching problem, however, turns out to be a difficult problem. Utilizing fast matrix multiplication, Sankowski [24] gave an algorithm with update time $O(N^{2.495}n^{1.495})$ that can maintain the weight of a maximum weight matching; his algorithm is also the best known in the unweighted case when $N = 1$. Abboud and Vassilevska Williams [1] showed in a formal sense that the use of fast matrix multiplication is inherent even for dynamic bipartite MCM, as fast enough updates would imply a faster Boolean matrix multiplication algorithm.

Because exact dynamic algorithms seem to be doomed to be inefficient, the majority of research on dynamic matching has been on maintaining approximate matchings. Following work by Onak and Rubinfeld [22], Baswana et al. [3] obtained a randomized fully dynamic algorithm that maintains a maximal (and hence 2-approximate) matching with expected amortized update time $O(\log n)$. Obtaining a fast deterministic algorithm has been a much more difficult task. Following Neiman and Solomon [21], Gupta and Peng [11] showed how to obtain a $(1+\varepsilon)$-approximation algorithm to the MCM with update time $O(\sqrt{m}/\varepsilon^2)$. Bernstein and Stein [5, 4] developed dynamic algorithms that achieve a $(3/2 + \epsilon)$-approximation in amortized update time $O(m^{1/4}\epsilon^{-2.5})$. Just recently Bhattacharya et al. [7] gave a very fast deterministic algorithm: for every $\epsilon > 0$, their algorithm achieves a $(2 + \epsilon)$-approximation algorithm with update time $\text{polylog}(n, 1/\epsilon)$.

For maintaining an approximate MWM, only two results are known. Anand et al. [2] obtained a randomized 4.911-approximation to the MWM in $O(\log n \log C + \log N)$ expected amortized update time[1], where N is the maximum edge weight and C is the ratio between N and the lowest edge weight. Gupta and Peng [11] obtained a $(1 + \varepsilon)$ approximation in deterministic worst case $O(\sqrt{m}\varepsilon^{-2-O(1/\varepsilon)} \log N)$ update time, where N is the largest edge weight in the graph.

There are two main questions that emerge:

1. *Can one get improved dynamic algorithms for MWM?* E.g., is there a faster than $O(\sqrt{m})$ deterministic worst case update algorithm for any constant approximation ratio? Is there a polylogarithmic update time dynamic algorithm with a better than 4.911-approximation guarantee? Can one use the recent techniques developed for dynamic MCM algorithms here?
2. All known dynamic MWM algorithms have update times that depend logarithmically on N. If N can be exponential in n, this causes a polynomial overhead in the running time. *Can the dependence on N be decreased?*
   If the algorithm had to be exact, then clearly the algorithm has to read the entire edge weights, so it is natural to have a dependence on $\log N$. However, here the answers can be approximate, so perhaps one can get away with reading only a few of the bits of the weight. If this is possible, then the algorithm would be fast even if the edge weights were exponential in the number of vertices. In particular, a truly polylogarithmic in n update time for approximate MWM would be possible.

1.1 Our results

In this paper we address both questions above.

Our first contribution is an efficient black-box reduction from dynamic approximate MWM to dynamic approximate MCM. Applying this reduction, we are able to improve the

[1] The update time stated in [2] is $O(\log n \log C)$, however, taking into account reading the edge weights on edge insertion actually adds a $\log N$ to the running time.

current best algorithms for dynamic MWM. The second contribution is a black-box reduction from dynamic approximate MWM on an exponential weight range to the same problem on a polynomial weight range, giving rise to the first algorithm for exponential range dynamic approximate MWM with truly polylogarithmic update time.

Let C be the ratio between the largest edge weight N and smallest edge weight L. Let n be the number of nodes and m the number of edges. The two theorems we prove are as follows:

▶ **Theorem 1.** *Let A be a dynamic algorithm that maintains an α-approximate MCM with update time $t(m,n)$. Then, for all $\varepsilon > 0$, there is a dynamic algorithm that maintains a $2\alpha \cdot (1+\varepsilon)$-approximate MWM with update time $O(t(m,n)\varepsilon^{-2}\log^2 C + \log N)$.*

If the original MCM algorithm was fully dynamic, so is the MWM algorithm, and if the original MCM algorithm is partially dynamic (incremental or decremental), so is the MWM one. If the MCM update time was worst case, then so is the MWM one, and if the original one was deterministic, so is the MWM one.

▶ **Theorem 2.** *Suppose that there is an algorithm that maintains an α-approximate MWM with update time $T(m,n,N)$. Then, for every constant $\varepsilon > 0$, one can convert it into an algorithm that maintains an $\alpha \cdot (1+\varepsilon)$-approximate MWM with asymptotic update time*

$$T(m,n,n^2\varepsilon^{-2})\log^2 n + \log n \log\log C + \log\log N.$$

The new algorithm enjoys the same properties as the old one (worst-case, deterministic etc).

Composing Theorems 1 and 2 we get that a $t(m,n)$-update time α-approximate MCM algorithm can be converted into a $(2+\varepsilon)\alpha$-approximate MWM algorithm with update time $O(t(m,n)\varepsilon^{-2}\log^4 n + \log n \log\log C + \log\log N)$.[2]

Applying Theorems 1 and 2 to the algorithms of Baswana et al. [3], Bhattacharya et al. [6, 7] and Bernstein and Stein [5, 4], we obtain the following immediate corollary.

▶ **Corollary 3.** *The following dynamic algorithms exist for MWM for all $\varepsilon > 0$:*

- *a $(4+\varepsilon)$-approximation with deterministic worst case update time $O(\ poly(\log n, 1/\varepsilon) + \log n \log\log C + \log\log N)$*
- *a $(4+\varepsilon)$-approximation with expected amortized update time $O(\varepsilon^{-2}\log^5 n + \log n \log\log C + \log\log N)$, and*
- *a $(3+\varepsilon)$-approximation in deterministic amortized update time $O(m^{1/4}\varepsilon^{-4.5}\log^4 n + \log n \log\log C + \log\log N)$.*

The above Corollary significantly improves upon the state of the art of dynamic MWM algorithms. In particular, it presents the first *truly polylogarithmic* update time for approximate dynamic MWM in the case when the edge weights can be exponential in n. Note that since our algorithm works on subsets of the edges of the original graph, we can derive dynamic weighted matching algorithms from dynamic maximum cardinality matching algorithms for *special classes of graphs* closed under edge deletions, such as the low-arboricity result of Neiman and Solomon [21].

A big advantage of our meta-algorithms is that they are simple, clean and combinatorial. Thus, if the original dynamic MCM algorithm is practical, then the MWM algorithms resulting from our theorems would likely also be practical.

[2] The composition gives this runtime for any $\varepsilon > 1/n^3$. Of course, if $\varepsilon \leq 1/n^3$, the trivial algorithm that recomputes the matching from scratch has update time $O(1/\varepsilon)$.

1.1.1 Overview of the reduction from dynamic MWM to MCM

Here we give an overview of our approach to proving Theorem 1. Our starting point is a result by Crouch and Stubbs [8] that reduced MWM to MCM in the streaming setting. This reduction shows how to reduce approximate MWM to a small number of instances of approximate MCM, but does not show how to maintain the output approximate MWM efficiently under arbitrary edge updates. Here we show how to make the reduction work in the dynamic setting.

Suppose that we are given a graph with integer edge weights between L and N and we want to compute an approximate MWM; let $C = N/L$. The basic idea of the Crouch and Stubbs reduction is to take maximal matchings from weight-threshold-based subgraphs of the underlying graph, and then merge the resulting maximal matchings together greedily. In particular, one computes an approximate MCM on all edges of weight at least $(1+\epsilon)^i$, for $i \in \{\lfloor(\log_{1+\epsilon} L\rfloor, \lfloor\log_{1+\epsilon} L\rfloor + 1, \ldots, \lfloor\log_{1+\epsilon} N\rfloor\}$. One produces the output matching from the approximate MCMs by including all edges from the matching in the weight class with the highest threshold, and then adding edges in descending order of the height of their weight class, as long as they are node-disjoint from the edges added so far.

Since we have $O(\epsilon^{-1} \log C)$ different weight classes, maintaining the approximate MCMs only incurs a small overhead, over any dynamic matching algorithm for approximate MCM. The only remaining concern, then, is how much time it takes to maintain the output matching by merging these together. This is our contribution on top of the result of Crouch and Stubbs: we show that the greedy merge of the MCMs can be *updated* efficiently.

We begin by assuming that we have approximate maximum cardinality matchings for each of the weight classes and an output matching M constructed *statically* by merging these matchings greedily in decreasing order of class height. When an update to the underlying graph occurs, it might cause some number of the edges in each of the matchings to change. This number in a weight class is at most the update time of the corresponding MCM data structure. Assuming that the dynamic algorithm for approximate MCM has update time $T(m, n)$, the number of changed edges in any one of the matchings is at most $T(m, n)$.

When edges change in an approximate MCM, we fold those changes into M one level at a time, in order of decreasing class height. If an edge in the output matching was deleted, we mark its endpoints as "newly free," so that we can look for new edges that cover those endpoints in lower classes – note that we needn't check the higher classes, since any such edge would have precluded the just-deleted edge from being in the output matching. If an edge was newly added, we add it to the output matching iff neither of its endpoints is covered by a higher class edge; if any edges are deleted in the process, we mark their endpoints as "newly free" and check for edges that cover these newly free nodes in lower classes, as before.

The process of checking for edges to cover a newly free node v is simple: in each lower class, as we're rolling in the changes to the current MCM, we also check whether v is matched in the updated MCM. If it is and its match u is not covered by an edge in the current output matching from a higher class, we add (u, v) to the output matching and v is no longer newly free. If u were already matched in the output matching to some node via a less exclusive edge e, we remove e from the output matching and make its other endpoint newly free.

The main part of the efficiency argument is as follows. The changes in the MCMs cause at most $O(T(m, n)\varepsilon^{-1} \log C)$ edges in our matching data structure to change. For each such changed edge, we carry out constant immediate processing, and also possibly create up to two "newly free" nodes. The crucial point is that if a newly free node is matched, then it can create at most one new newly free node, and so the total number of newly free nodes that have cascaded down the classes starting from a particular changed edge is at most 2.

For each weight class, we might need to consider up to $O(T(m,n)\varepsilon^{-1}\log C)$ newly free nodes from the classes above it, performing a constant time operation for each. Therefore the total time it takes to handle the entire merger is $O(\epsilon^{-2}\log^2(C)T(m,n))$. This process is sufficient to produce an output graph identical to one created by the static merge described above, since edges from higher classes are allowed to preempt ones from lower classes, just as if we'd greedily merged them in first. For the actual update time, we need to read in the weight of any inserted edge, and for this we pay an additional $O(\log N)$.

In Section 3 we provide the full details of the algorithm and the proofs of runtime, correctness and approximation guarantee.

1.1.2 Overview of improving the dependence on the weights

Here we provide an overview of our proof of Theorem 2 presented in Section 4.

A useful fact about weighted matching is that the maximum weight matching of the restriction of the graph to edges of weight at least a $2\epsilon/n$ fraction of the weight of the largest edge (so at least $2\epsilon N/n$, in our case) has weight at least $1/(1+\epsilon)$ times the true maximum weight matching, even if there are no edges other than the unique edge of weight N in this subgraph. The reason is simple: a matching has at most $n/2$ edges, and if none of these weigh more than $2\epsilon N/n$, the whole matching weighs only $(n/2)(2\epsilon N/n) = \epsilon N$, meaning the whole matching including the top edge weighs at most $(1+\epsilon)N$, as desired. Adding more edges above the threshold only makes the "best-only" matching a better approximation of the true matching.

It would be natural to want to apply this principle to fully dynamic weighted matching algorithms (like ours above) in order to reduce the dependence on the – potentially quite large – weight range to a dependence on $O(\epsilon^{-1}n)$. Unfortunately, N can potentially change in the fully dynamic setting: in particular, if we were relying on a single high-weight edge to carry our matching and that edge is deleted, we would have no approximation at all!

Our solution is to maintain enough copies of a dynamic maximum weight matching algorithm on a small range of weights, to guarantee that any two edges whose weights are within a factor of $2\epsilon/n$ of one another appear in the same data structure somewhere. In particular, the highest weight edge appears in the same data structure with any edge with weight within a $2\epsilon/n$ factor of the maximum weight. At the same time, we keep the weight ranges nearly disjoint so that only two of the data structures need to be altered by any update to the underlying graph.

To accomplish both goals, we take weight ranges of the form $((n\varepsilon^{-1})^i, (n\varepsilon^{-1})^{i+2}]$ for $\log_{n\varepsilon^{-1}} L \le i \le \log_{n\varepsilon^{-1}} N$ where L and N are the minimum and maximum edge weight in the data structure throughout its existence. The number of these ranges is now $O(\log_{n\varepsilon^{-1}} C)$ (where $C = N/L$), and we have guaranteed that the MWM in one of these classes is a $(1+\epsilon)\alpha$ approximation to the true MWM, and that the weight range in each class can be reduced to $n\varepsilon^{-1}$. However, we would like to have a persistent output matching that always contains a $(1+\epsilon)\alpha$ approximation of the maximum weight matching, that doesn't change dramatically in a single time step just because the highest weight edge was deleted from the graph.

To this end, we define and maintain a specially constructed output matching M that we call the *census matching*. The idea is as follows. M will contain a certain number of edges from the matchings in each weight class as follows. Consider some weight class $((n\varepsilon^{-1})^i, (n\varepsilon^{-1})^{i+2}]$ and suppose that the number of edges in its matching is n_i. Let $N_i = \sum_{j>i} n_j$ be the sum of cardinalities of matchings in classes above i. If $n_i > N_i$, we will have $n_i - N_i$ matching edges R_i from class i that the census matching M will consider. M is constructed by greedily merging R_i similar to our algorithm from the MWM to MCM reduction.

In particular, the census matching always contains all the edges of weight within $n\varepsilon^{-1}$ of the current maximum weight. Further, since the total number of edges in matchings that have any edges considered by the output matching doubles with each lower class that still has nonempty R_i, and since no matching has more than $n/2$ edges, the total number of classes with any edges considered is $O(\log n)$.

The output matching itself is relatively simple and behaves like the output matching of our first algorithm, except that it's drawing from these "representative sets" R_i, rather than from approximate maximum cardinality matchings. Fortunately, we can show that only two representative sets can change at a time, so it only takes $O(t(n, m, (n\varepsilon^{-1})^2, (n\varepsilon^{-1})^2) \log n)$ time to do the updates, where $t(n, m, N, C)$ is the update time of the underlying MWM algorithm, and thus the maximum number of edges that can be added or removed from a single MWM in a single time step.

We show that when an edge is to be updated, to figure out which pair of classes it belongs to, we only need to read $O(\log \log N)$ bits of its weight. To make everything work efficiently, we introduce a complete binary tree data structure (with the weight classes as leaves) that helps us maintain the representative sets R_i efficiently, though the tree is only conceptually complete, as we only add edges and nodes on paths from the root to leaves of non-empty weight classes to avoid wasting time when a new edge appears of much higher or lower weight than all previous edges. In particular, the $O(\log \log C)$ overhead in our running time is due to the tree having depth $O(\log \log C)$. The $\log \log N$ dependence is due to various pointers to positions in the bits of the weight. The details are in Section 4.

2 Merging matchings greedily

Here we prove a technical lemma used in both of our algorithms.

Let $S_1, \ldots, S_k$ be matchings in a graph G. For any edge (u, v), let $\ell(u, v) = \max\{i \mid (u, v) \in S_i\}$. Call a matching M a *Greedy Census* matching if for any edge $(u, v) \in S_i \setminus M$, there exists either (u, u') or $(v, v') \in M \cap S_j$, for some $j \geq i$. This property is equivalent to saying that M could have been constructed by greedily adding edges from each level from k down to 1, ensuring that the added edges are maximal within the current level before moving down. Thus, we have $\forall j : M \cap (\cup_{i>j} S_i)$ is maximal in $\cup_{i>j} S_i$

For an edge (u, v) let $L(u, v)$ be a decreasing set of indices $\{i_1, i_2, \ldots\}$ such that $(u, v) \in S_{i_j}$ for each j and $i_j > i_{j+1}$. In particular, $i_1 = \ell(u, v)$.

▶ **Lemma 4.** *Suppose we are given any collection of edge sets $In_1, In_2, \ldots, In_k$ and $Del_1, \ldots, Del_k$, the sets $L(u, v)$ for all (u, v) and a Greedy Census matching M of $S_1, \ldots, S_k$. Then one can insert all edges of In_j into S_j and delete all edges of Del_j from S_j and update M and all $L(u, v)$ so that M is a Greedy Census of the modified sets S_j, all in time*

$$\sum_{j=1}^{k} j \cdot \big(|In_j| + |Del_j|\big).$$

In our reduction from MWM to MCM, the sets S_j will correspond to the approximate maximum cardinality matchings of different weight classes, and in our algorithm for decreasing the dependence on the edge weight, they will correspond to the sets of representative edges of different weight classes.

Before we prove Lemma 4, let us introduce some notation.

For each $j \in [k]$, let NewFree(j), initially empty, be the set of newly free nodes created for level j. A newly free node u is any node that was covered by some edge in M, and then

became uncovered after a change to that matching. Specifically, u becomes newly free by the deletion of an edge (u, v) from M. This deletion could happen for one of two reasons: either because (u, v) was in $Del(j)$, meaning it was removed by an update to the underlying graph or some activity at a lower level of the algorithm, and is no longer in consideration for inclusion in M, or because $\ell(u, v) \leq j$ and (u, v) was deleted so that v can be matched via an edge of level greater than j; that is, we found another better edge for M to use in place of (u, v) and u was left uncovered.

Now the procedure is as follows. We set $\textsc{NewFree}(j) = \emptyset$ for all j. We iterate through all levels j from k down to 1. Fix a level j. Then, for each edge $(x, y) \in Del_j$, remove it from S_j, and remove j from $L(x, y)$. If now $\ell(x, y) < j$, and if $(x, y) \in M$, remove (x, y) from M and add x and y to $\textsc{NewFree}(j)$. For each $(x, y) \in In_j$, insert it into S_j, add j to $L(x, y)$. For each $(x, y) \in In_j$, in a second loop, check whether x and y are matched in M. Suppose that either x is not matched or x is matched to x' with $\ell(x, x') < j$. Suppose further that either y is not matched or it is matched to y' with $\ell(y, y') < j$. Then, remove (x, x') and (y, y') from M, add x' and y' to $\textsc{NewFree}(j)$ and insert (x, y) into M. If x or y are in $\textsc{NewFree}(j)$, remove them from $\textsc{NewFree}(j)$.

Now, for each $u \in \textsc{NewFree}(j)$, let v be its match in S_j (recall S_j is a matching). If v is matched to a node v' such that $\ell(v, v') \geq j$, then just move u to $\textsc{NewFree}(j-1)$ and move to the next u. Else if v is unmatched in M or if v is matched to some v' with $\ell(v, v') < j$, then remove (v, v') from M, add v' to $\textsc{NewFree}(j-1)$ and add (u, v) to M. This completes stage j.

Now we prove the following claims:

▶ **Claim 1.** *The runtime of the above algorithm is* $O(\sum_{j=1}^{k} j(|In_j| + |Del_j|))$.

Proof. Each deletion in Del_j can create at most 2 newly free nodes. Each insertion in In_j can cause the creation of at most 2 newly free nodes as well. If a newly free node is matched in some stage j, then it can create at most one new newly free node, and this one is put in $\textsc{NewFree}(j-1)$. Thus, the total cost of processing newly free nodes is $O(\sum_{j=1}^{k} j(|In_j| + |Del_j|))$. ◀

▶ **Claim 2.** *Let* $S'_i = S_i \cup In_i \setminus Del_i$. *$M$ is a greedy census matching of the updated matchings* $S'_1, \ldots, S'_k$.

Proof. Consider for contradiction an edge $(u, v) \in S_i \setminus M$ such that neither u nor v are matched with edges of level at least i. Before updating, one of the following was true, since M was a census matching: 1) $(u, v) \notin S_i$, 2) $(u, v) \in S_i \cap M$ or 3) (wlog) $(u, u') \in S_j \cap M$, for some $j > 1$. In case 1, (u, v) must have been in In_i, and, since neither u nor v are matched at a level above i, (u, v) would have been added to M, and then it's not true that $(u, v) \in S_i \setminus M$. In case 2, (u, v) left M with the update, but is still in S_i, so it wasn't in Del_i. This can only have happened because a higher level edge, wlog (u, u') was in $In_{\ell(u,u')}$, causing (u, v) to be removed from M. Since the In_i are processed in descending order, and the edges in $Del_{\ell(u,u')}$ get deleted before the edges from $In_{\ell(u,u')}$ get inserted, (u, u') will not be deleted by anything in this batch of updates, and so (u, u') is in M, and u is matched, violating the second part of the assertion. In case 3, the higher level edge, (u, u') must have been deleted in the update, either directly by being in Del_i or indirectly by having a higher level edge claim u'. In either case, u would be added to $\textsc{NewFree}(\ell(u', u))$, and it would either get matched to v, putting $(u, v) \in M$, or it would get matched to some (u, u'') of higher level than (u, v), both of which invalidate the assertion. ◀

3 A meta-algorithm for approximating MWM

Let $\varepsilon > 0$ be fixed. For every integer i, let E_i contain all edges of G that have weight $\geq (1+\varepsilon)^i$. Let D_i be a data structure that maintains an α-approximate MCM of E_i with update time $t(m,n)$.

Let ℓ be the smallest i such that $E_i \neq E_{i+1}$. During each stage of the dynamic algorithm we have a pointer to D_ℓ, for the current value of ℓ. Let M be the approximate MWM that we are maintaining. Let $\tilde{M}_i$ be the approximate MCM maintained by D_i, and let $M_i := M \cap \tilde{M}_i$. We will actually maintain M so that $M_i = M \cap E_i$.

We define the level $\ell(u,v)$ of edge (u,v) to be i such that $w(u,v) \in [(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$.

We use Lemma 4 and its algorithm from Section 2 with $S_i = \tilde{M}_i$ to maintain the greedy census matching M. To do this, when a weighted edge (u,v) is inserted or deleted, we insert or delete it from all data structures D_j with $j \leq \ell(u,v)$. If $\ell(u,v) = \ell$ and $D_\ell \setminus D_{\ell+1}$ became empty, update ℓ. Now, after the D_j are updated, we figure out all the sets In_j and Del_j to feed into the data structure from Section 2.

We immediately obtain:

▶ **Lemma 5.** *M is a maximal matching in the graph $\cup_i \tilde{M}_i$.*

▶ **Lemma 6.** *The update time is $O(t(m,n)(\log_{1+\varepsilon} N/L)^2)$.*

Proof. Let $\mu = \log_{1+\varepsilon} N/L$. $|Del_j|$ and $|In_j|$ for each j are at most $O(t(m,n))$ since the (amortized/expected/worst case) number of deletions or insertions performed by each D_j is $\leq t(m,n)$. By the proofs in Section 2, the running time should be asymptotically $\sum_j j \cdot |In_j| + |Del_j| \leq t(m,n) \sum_{j=1}^{\mu} j \leq O(\mu^2 t(m,n))$. ◀

The proof of the Lemma below directly follows from Claim 2 from Section 2.

▶ **Lemma 7.** *For every j, $M \cap E_j$ is maximal matching in the graph $\cup_{i \geq j} \tilde{M}_i$*

Now we can prove the approximation guarantee part of our result.

▶ **Lemma 8.** *If each $\tilde{M}_j$ is an α-approximate MCM, then M is a $2\alpha(1+\varepsilon)$-approximate MWM.*

Proof. Consider a fixed optimal MWM M^*, and let m_i be the cardinality of the MCM on edges of level i, and note that $|M^* \cap E_i| \leq m_i$. Clearly $\alpha|\tilde{M}_i| \geq m_i \geq |M^* \cap E_i|$. Further, $2|M \cap E_i| \geq |\tilde{M}_i|$ by Lemma 7, since $M \cap E_i$ is a maximal matching on a superset of the edges in $\tilde{M}_i$, meaning $|M \cap E_i| \geq \frac{1}{2\alpha}|M^* \cap E_i|$. This means that for each i, every 2α edges of M^* can be assigned to a single "babysitter" edge of M of equal or higher level – this is perhaps easier to see by subtracting out all of the already assigned "pairs" from higher levels, giving $|M \cap E_i| - \frac{1}{2\alpha}|M^* \cap E_{i+1}| \geq \frac{1}{2\alpha}(|M^* \cap E_i| - |M^* \cap E_{i+1}|)$. Since the babysitter edge e is of at least as high a level as all of its charges, $(1+\varepsilon)w(e) \geq w(e^*)$, and, in general, $2(1+\varepsilon)\alpha w(M \cap E_i) \geq w(M^* \cap E_i)$, which, taking $i = L$, proves the lemma. ◀

4 A Meta-meta-algorithm for approximating MWM

4.1 The Census Matching

We maintain a matching using the process from Section 2, partitioning the weight range, running a matching algorithm on each partition, and merging the results, just like in Section 3. However, we keep weighted (rather than unweighted) matchings in each class, and we only consider some of the edges from a few of these classes, rather than merging all of the edges.

In particular, we "semi-partition" the weight range, such that every value within that range falls into exactly two of our semi-partition intervals. Each interval is of the form $((n/\varepsilon)^i, (n/\varepsilon)^{i+2}]$, for $i \in \mathbb{Z}$ such that $\log(w_o) - 1 \leq i \leq \log(w^*) - 1$, where w_o and w^* are the highest and lowest weights that have appeared on any edge in the matching, even if that edge was later deleted. For each of these semi-partitions, we maintain a maximum weight matching algorithm on the subgraph defined by the edges in the underlying graph whose weights fall within that interval. We denote by W_i the approximate MWM maintained on the interval $((n/\varepsilon)^i, (n/\varepsilon)^{i+2}]$.

To merge the W_is together, we maintain a single census matching C, which uses the algorithm from Section 2 to combine a subset of the edges from $O(\log n)$ specific W_is, as determined by a binary tree data structure that we will call the "responsibility tree", T. We will describe this below.

As described in the introduction, we strive to select from each W_i a number of representative edges. Let n_i be the cardinality of W_i, and let $N_i = \sum_{j>i} n_j$ be the total sum of the cardinalities of the matchings in classes above i. If $n_i > N_i$, we will have $n_i - N_i$ matching edges R_i from class i that the census matching M will consider. The tree data structure T will facilitate maintaining the cardinalities that the R_is need to have.

4.2 The Responsibility Tree

We build a near-complete binary tree with leaves corresponding to the W_is to efficiently determine which edges should be sent to the census matching to ensure the desired properties. The W_is are arranged in descending order from left to right, with the leftmost leaf corresponding to $W_{\log_{n/\varepsilon}(w^*)}$ and the rightmost to $W_{\log_{n/\varepsilon}(w_o)}$. The leaves track the number of edges in the matchings and the number of those edges which are represented, and the internal nodes have attributes which depend on the attributes of their children. Whether or not each W_i has the correct number of represented edges can be determined just by looking at the attributes of the root, and if those attributes are wrong, the incorrectly represented classes/leaves can each be tracked down in a single traversal from root to leaf.

Every node in the tree has three attributes: mass, high, and low. The mass attribute is a running tally of the total number of edges among matchings in the classes associated with the leaves of this subtree. The "high" and "low" attributes validate the number of representatives that classes in this subtree have: if the "high" indicator $h(v)$ on any node v is negative, some class in that node's subtree $t(v)$ has too *many* representatives, and if the "low" indicator $l(v)$ on any node v in the left-most branch (the "spine") is positive, then some class in that node's subtree $t(v)$ has too *few* representatives.

We refer to edges that are in the selection pool for the census matching as "representatives," and we record the number of representatives for each class as $R(v)$.

We refer to the high indicator for a vertex v as $h(v)$, its left child as $v.l$, and its right child as $v.r$.

$$h(v) := \begin{cases} \min(h(v.l), h(v.r) - m(v.l)) & \text{if } v \text{ is not a leaf} \\ \infty & \text{if } leaf(v) \wedge R(v) = 0 \\ m(v) - R(v) & \text{if } leaf(v) \wedge R(v) > 0 \end{cases}$$

Similarly for the low indicator, $\ell(v)$,

$$\ell(v) := \begin{cases} m(v) - R(v) & \text{if } v \text{ is a leaf} \\ \max(h(v.l), h(v.r) - m(v.l)) & \text{otherwise} \end{cases}$$

Finally, mass, $m(v)$, is simply

$$m(v) := \begin{cases} \#\text{ edges in matching} & \text{if } v \text{ is a leaf} \\ m(v.l) + m(v.r) & \text{otherwise} \end{cases}$$

4.2.1 Tree Properties

▶ **Lemma 9.** *If the root r has $h(r) = l(r) = 0$, then for every class ℓ_i, $m(\ell_i) = R(\ell_i) + \sum_{j>i} m(\ell_j)$ if $R(\ell_i)$ is positive, and $m(\ell_i) \leq \sum_{j>i} m(\ell_j)$ otherwise. That is, if the root's high and low indicators are both 0, then no class has too many or too few representatives.*

We will prove the lemma using two claims below.

▶ **Claim 3** (No Class has Too Many Representatives.)**.** *For any node v,*

$$h(v) = \min_{\{a_i \in t(v) | R(a_i) \neq 0\}} \left(m(a_i) - R(a_i) - \sum_{j>i \in t(v)} m(a_j) \right).$$

Proof. By induction. For a leaf a, the subtree $t(a)$ is trivial, and this is the definition of $h(a)$. For an internal node v, $h(v) = \min(h(v.l), h(v.r) - m(v.l))$. $v.l$ has no higher classes to account for in $t(v)$, and subtracting $m(v.l)$ from $h(v.r)$ subtracts the masses of all previously unaccounted for classes, meaning both children fit the conditions, and $h(v)$ is simply the minimum of these, as desired. ◀

Consider a class a_i such that its number of represented edges is greater than it should be, i.e., the total number of edges in classes above it $\sum_{j>i}(m(a_j))$, plus the number of its representatives $R(a_i)$, is greater than its own mass $m(a_i)$. Then by Claim 3, $h(r)$ will be negative.

Thus, if $h(r) = 0$, there is no class with too many representatives.

▶ **Claim 4** (No Class has Too Few Representatives.)**.** *For any node v,*

$$\ell(v) = \max_{a_i \in t(v)} \left(m(a_i) - R(a_i) - \sum_{j>i \in t(v)} m(a_j) \right).$$

Proof. By induction, symmetrical to proof of Claim 3. ◀

Consider a class a_i such that its number of represented edges is less than it should be, i.e., $m(a_i) - R(a_i) - \sum_{j>i}(m(a_j)) > 0$. Then by Claim 4, $\ell(r) > 0$.

Thus, if $\ell(r) = 0$, no node has too few representatives.

4.3 The Census Matching is Nearly as Good as The Underlying MWM

We define the "Census matching" to be the resultant matching from combining the representative edges from each class in a way that mimics a static greedy merge, that is, using the algorithm of Section 2.

4.3.1 The Best is Good Enough

▶ **Lemma 10.** *There exists a matching, M_B, such that 1) the ratio between the lowest and highest weight edges in M_B is at most $\epsilon^{-1}n/2$ and 2) $(1+\epsilon)w(M_B) \geq w(M^*)$*

Proof. Consider the lowest interval of width $\epsilon^{-1}n/2$ such that there are no edges above that interval; thus, the interval will contains the highest weight edge(s) in the underlying graph at the top, as well as all edges that weigh at least $2\epsilon/n$ times as much as those edges. We call this interval $B = [2\epsilon/nw^*, w^*]$, where w^* is the highest weight of any edge in the underlying graph.

Consider the optimal matching on edges with weights that fall within B, $\textsc{Opt}_b$. Clearly, its weight is at least w^*, since the trivial matching formed by taking just the single highest weight edge has that weight, and is a matching on edges in B. Now consider the amount by which the true optimum matching, $\textsc{Opt}$, exceeds the weight of $\textsc{Opt}_b$. Any gains the true optimum makes must be from the inclusion of edges outside of (hence, below) B, each of which weighs at most $2\epsilon w^*/n$. Further, the optimum matching can only include $n/2$ such edges (because it is a matching), meaning

$$\begin{aligned} w(\textsc{Opt}) - w(\textsc{Opt}_b) &\leq \frac{n}{2}\frac{2\epsilon w^*}{n} && (1)\\ &= \epsilon w^* && (2)\\ &\leq \epsilon w(\textsc{Opt}_b) && (3) \end{aligned}$$

which tells us that

$$w(\textsc{Opt}) \leq (1+\epsilon)w(\textsc{Opt}_b) \quad (4)$$

◀

So if we have a maximum weight matching algorithm that gives an α-approximation, and run it on just edges with weights in B, we get a $(1+\epsilon)\alpha$-approximation to the overall maximum weight matching. Moreover, if we run it on edges from some interval containing B, we get the same guarantees.

▶ **Corollary 11.** *For any graph G, there exists an interval $I = [(\epsilon^{-1}n/2)^i, (\epsilon^{-1}n/2)^{i+2})]$ such that for any MWM algorithm A, $(1+\epsilon)w(A(G_I)) \geq w(A(G))$, where G_I is the restriction of G to edges with weights in I.*

4.3.2 The Census Displays the Best

▶ **Lemma 12.** $w(M) \geq w(M_B)$

Proof. Since we have a copy of our MWM algorithm running for each non-empty interval of the form $((\epsilon^{-1}n)^i, (\epsilon^{-1}n)^{i+2}]$, we have a MWM covering a superset of every interval of "width" $\epsilon^{-1}n/2$, and, in particular, B. By Lemma 1, the top non-empty class (which contains B) has all of its edges represented, and by the fact that the census matching prioritizes higher weight edges, all of these representatives will, in fact, be included in the final census matching. This means that the census matching is a superset of MWM_B, meaning its weight is at least $(1+\epsilon)\alpha w(\textsc{Opt})$. ◀

4.4 Maintaining the Census is Fast

▶ **Lemma 13.** *Given an underlying MWM running in time $T(n, m, C, N)$, the census matching can be maintained in $O(\log^2(n)T(n, m, (n/\varepsilon)^2, (n/\varepsilon)^2))$ time.*

Proof. By Lemma 9, any class with 1 or more representatives has more edges in its underlying matching than every class above it. Clearly, this means that each class with any representatives has twice as many edges in its matching than the last. Combined with the fact that no matching can have more than $n/2$ matchings, this means there are at most $O(\log n)$ represented classes, and consequently only $O(\log n)$ levels in the census matching. Then the proof follows from the Lemma in Section 2.

Within each of the classes, an instance of the underlying MWM is being run with weights between 1 and $(n/\varepsilon)^2$, and so the maximum number of edges that can change in any given matching (and thus the cardinality of Del_i and In_i) is bounded above by the running time of the MWM on that interval, i.e. $T(n, m, (n/\varepsilon)^2, (n/\varepsilon)^2)$ ◀

4.5 Maintaining the Tree is Fast

▶ **Lemma 14.** *Updating the tree after an insertion or deletion takes $O(\log n \log\log C)$ time*

Proof. Since the Responsibility Tree is a near-complete binary tree with $O(\log C)$ leaves, the length of the path from a leaf to the root is $O(\log\log C)$, meaning only $O(\log\log C)$ nodes need to be changed for each leaf whose underlying matching had a change in cardinality or represenatives. Since any edge is within the purview of only two W_is, only two classes can have a change in the cardinality of their underlying matchings.

By the proof of Lemma 13, $O(\log n)$ classes have a non-zero number of representatives before and after the update, meaning that for all but $O(\log n)$ classes, the number of representatives was 0 before and after the change. So the total number of classes which had a change in number of representatives or cardinality is at most $O(\log n)$, which, combined with the fact that updating each one takes $O(\log\log C)$ time, gives us the desired time complexity. ◀

4.6 Numerical Considerations (reading an update is fast)

We assume that updates arrive with weights w that are organized like standard floating point numbers: a significand s, a base b, and an exponent x such that $1 \leq s < b$ and $sb^x = w$, along with pointers to where in the number these segments begin and end. Note that s will be $O(\log w)$ bits long, b will be a constant number of bits, and x will be $O(\log\log w)$ bits.

We want two pieces of information from this update: the pair of W_is that the edge should be processed by, and enough information about the edge's weight to maintain those W_is without too much error. The first piece of information is easy to approximate from reading just the exponent and the base, since the significand only changes the weight of w by a factor of at most b. Then we can send the edges to class i and $i+1$ for $i = x/\log_b(n/\varepsilon)$. The only concern then is if $x/\log_b(n/\varepsilon) < i+1$ but $sx/\log_b(n/\varepsilon) = \log_{n/\varepsilon} w \geq i+1$, in which case the edge "belonged" in classes $i+1$ and $i+2$ but was sent to i and $i+1$ instead. Fortunately, if w is on the border in this way, and $i+2$ is the highest non-empty class, w would be on one of the lowest weight edges in that class, meaning that its weight is at most $\varepsilon b/n$ times the weight of the largest edge in the matching, and the error incurred is only $O(\varepsilon)$ times that weight, even if a full $n/2$ edges of this type are missed, and so the error is subsumed into the rounding error.

The second piece of information does require us to read the significand, but fortunately only very few bits of it. Since all of the edges within class i have weights between $(n/\varepsilon)^i$ and $(n/\varepsilon)^{i+2}$, and because misreporting the lower order bits of the weights only causes multiplicative $(1-\varepsilon/n)$ error in each edge weight, for a total error of ε from all edges in the matching, we can divide the weights of the edges by the lower bound of their classes, resulting in needing to send weights that can be described in only $O(\log(n/\varepsilon))$ bits. Further, since we can just truncate off the trailing bits of the significand rather than performing a true division, we only need to read the first $O(\log(n/\varepsilon))$ bits of the significand to determine which values to send.

References

1. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.
2. A. Anand, S. Baswana, M. Gupta, and S. Sen. Maintaining approximate maximum weighted matching in fully dynamic graphs. In *FSTTCS*, pages 257–266, 2012.
3. S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(logn)$ update time. In *FOCS*, pages 383–392, 2011.
4. Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 167–179, 2015.
5. Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *In Proc. SODA*, page to appear, 2016.
6. Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 785–804, 2015.
7. Sayan Bhattacharya, Monika Henzinger, and Danupon Nanangkai. New deterministic approximation algorithms for fully dynamic matching. In *Proc. STOC*, page to appear, 2016.
8. Michael Crouch and Daniel Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 96–104, 2014.
9. H. Gabow. A scaling algorithm for weighted matching on general graphs. In *Prof. FOCS*, pages 90–100, 1985.
10. H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for general graph-matching problems. *J. ACM*, 38(4):815–853, 1991.
11. M. Gupta and R. Peng. Fully dynamic $(1+\varepsilon)$-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 548–557, 2013.
12. N. J. A. Harvey. Algebraic structures and algorithms for matching and matroid problems. In *Proc. FOCS*, volume 47, pages 531–542, 2006.
13. J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.
14. Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on*

Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014, pages 217–226, 2014.

15 H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

16 François Le Gall. Powers of tensors and fast matrix multiplication. In *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303, 2014.

17 A. Madry. Navigating central path with electrical flows: from flows to matchings, and back. In *Proc. FOCS*, 2013.

18 Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27, 1980.

19 M. Mucha and P. Sankowski. Maximum matchings via gaussian elimination. In *Proc. FOCS*, volume 45, pages 248–255, 2004.

20 K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. In *Proc. STOC*, volume 19, pages 345–354, 1987.

21 O. Neiman and S. Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 745–754, 2013.

22 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 457–464, 2010.

23 M. O. Rabin and V. V. Vazirani. Maximum matchings in general graphs through randomization. *J. Algorithms*, 10(4):557–567, 1989.

24 P. Sankowski. Faster dynamic matchings and vertex connectivity. In *Proc. SODA*, pages 118–126, 2007.

25 P. Sankowski. Maximum weight bipartite matching in matrix multiplication time. *Theor. Comput. Sci.*, 410(44):4480–4488, 2009.

26 V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. STOC*, pages 887–898, 2012.