2013 IEEE 54th Annual Symposium on Foundations of Computer Science

Fully Dynamic $(1+\epsilon)$-Approximate Matchings

Manoj Gupta
Department of CSE,
I.I.T. Delhi, India
gmanoj@cse.iitd.ernet.in

Richard Peng
Computer Science Department
CMU, Pittsburgh, USA
yangp@cs.cmu.edu

Abstract— We present the first data structures that maintain near optimal maximum cardinality and maximum weighted matchings on sparse graphs in sublinear time per update. Our main result is a data structure that maintains a $(1+\epsilon)$ approximation of maximum matching under edge insertions/deletions in worst case $O(\sqrt{m}\epsilon^{-2})$ time per update. This improves the $3/2$ approximation given by Neiman and Solomon [20] which runs in similar time. The result is based on two ideas. The first is to re-run a static algorithm after a chosen number of updates to ensure approximation guarantees. The second is to judiciously trim the graph to a smaller equivalent one whenever possible.

We also study extensions of our approach to the weighted setting, and combine it with known frameworks to obtain arbitrary approximation ratios. For a constant ϵ and for graphs with edge weights between 1 and N, we design an algorithm that maintains an $(1+\epsilon)$-approximate maximum weighted matching in $O(\sqrt{m}\log N)$ time per update. The only previous result for maintaining weighted matchings on dynamic graphs has an approximation ratio of 4.9108, and was shown by Anand et al.{[2], [3]}

1. Introduction

The problem of computing maximum or near-maximum matchings in a graph has played a central role in the study of combinatorial optimization[18], [22]. A matching is a set of vertex-disjoint edges in a graph, and two variants of the problem are finding the maximum cardinality matching in an unweighted graph, and finding the matching of maximum weight in a weighted graph. The problem is appealing for several reasons: it has a simple description; matchings sometimes need to be improved by highly non-local steps; and certifying the optimality of a matching yields a surprising amount of structural information about a graph. On static graphs, the current best algorithms for maximum cardinality matching run in $O(m\sqrt{n})$ time, on bipartite graph by Hopcroft and Karp [15], and on general graph by Micali and Vazirani [19]. In the weighted case, algorithms with similar running times were given by Gabow and Tarjan [10], and by Duan et al. [7].

A natural question from a data structure perspective is whether on a dynamically changing graph the solution to an optimization problem can be maintained faster than recomputing it from scratch after each update. For maximum cardinality matching, an $O(m)$ time algorithm follows by executing one phase of the static algorithm described by Tarjan[24]. For dense graphs, a faster running time of $O(n^{1.495})$ has been shown by Sankowski [23], and to date this is the only known result that gives sublinear time per update. For trees, Gupta and Sharma [11] gave an algorithm based on top trees that takes $O(\log n)$ time per update.

On static graphs, a nearly-optimal matching can be computed much faster than finding the optimum matching. So it stands to reason that the same should apply in the dynamic case. Ivković and Llyod[16] gave the first result in this direction: an algorithm that maintains a maximal matching with $O((n+m)^{0.7072})$ update time. Recently there has been a growing interest in designing efficient dynamic algorithms for approximate matching. Onak and Rubinfeld designed a randomized algorithm that maintains a c-approximation of maximum matching in $O(\log^2 n)$ update time [21], where c is a large unspecified constant. Baswana, Gupta and Sen[4] showed that maximal matching, which is a 2-approximation of maximum matching, can be maintained in a dynamic graph in amortized $O(\log n)$ update time with high probability. Subsequently, Anand et al. {[2], [3]} extended this work to the weighted case, and showed how to maintain a matching with weight that is *expected* to be at least $1/4.9108 \approx 0.2036$ of the optimum.

These results show that a large matching can be maintained very efficiently in dynamic graphs, but leave open the question of maintaining a matching closer to the optimum matching. Recently, Neiman and Solomon[20] showed that a matching of size at least $2/3$ of the size of optimum matching can be maintained in $O(\sqrt{m})$ time per update in general graphs , as well as $O(\log n/\log\log n)$ time per update on bounded arboricity graphs. A similar result of maintaining 3/2-approximate matchings was obtained independently by Anand [1]. This leads to the following question: Can we maintain a matching close to maximum matching (say $(1+\epsilon)$-approximate matching) in a dynamic weighted or unweighted graph? We answer this question in affirmative by designing the first data structure that maintains arbitrary quality approximate max-cardinality and max-weighted matching in sublinear time on sparse graph.

Our algorithm differs significantly from previous ones in that we do not maintain strict invariants. Baswana et al. [4] maintained a maximal matching, which ensures no edge has both endpoints unmatched; and the $3/2$-approximate algorithm designed by Neiman and Solomon[20] remove

0272-5428/13 $26.00 © 2013 IEEE
DOI 10.1109/FOCS.2013.65

IEEE computer society

all length three augmenting paths in the graph at each update step. Our approach makes crucial use of the fact that the optimization objectives involving matching is *stable*. That is, a single update can only change the value of the optimum matching by 1. So if we find a matching close to maximum matching at some update step, it remains close to maximum even after several updates to the graph. In case the current matching ceases to be a good approximation of the maximum matching, we then re-run the static algorithm to get a matching that is close to optimum. This approach of re-running a expensive routine occasionally is a common technique in dynamic graph data structures [13], [14], [5]. It is particularly powerful for approximating matchings since the stability property gives us freedom in choosing when to re-run the static algorithms. But re-running static algorithm occasionally works well when the maximum matching in the graph is large. To deal with graphs having small maximum matching, we introduce the concept of *core subgraph* which is the central concept of our paper. A *core subgraph* is a subgraph of a graph having the following two properties: Its size is considerably smaller than the entire graph. Secondly, the size of maximum matching in *core subgraph* is same as the size of maximum matching in the entire graph. We will crucially use these two properties in designing a dynamic algorithm for approximate matching. A detailed description of our algorithm, as well as other components of our data structure are presented in Section 3. The main result for approximating the maximum cardinality matching can be stated as follows:

Theorem 1.1: For any constant $\epsilon < 1/2$, there exists an algorithm which maintains a $(1+\epsilon)$-approximate matching in an unweighted dynamic graph in $O(\sqrt{m})$ worst case update time.

It can be argued that the *stability* property of matchings that we rely on is rare among optimization problems. For most other problems like shortest paths and minimum spanning tree, there exist updates that require immediate changes in the approximate solution maintained. For matchings, such updates exists in the weighted version, where the objective is the sum of weight over edges in the matching. Direct extensions of our approach have linear dependencies on N in update time, where N is the maximum weight of an edge. This dependency can in fact be viewed as a quantitative measurement of the decrease in stability as we allow larger weights.

As a result, we investigate rounding/bucketing based approaches which have logarithmic dependency on N in Section 4. This was first studied for maintaining dynamic matchings by Anand et al. [2], and they used dynamic maximal matchings as a subroutine in their algorithm. Directly substituting our result for maximum cardinality matching leads to immediate improvements in the approximation ratio which is the second result in this paper.

Theorem 1.2: For any constant $\epsilon < 1/2$, there exists an algorithm that maintains $(3+\epsilon)$-approximate maximum weighted matching in a graph where edges have weights between $[1, N]$ in $O(\sqrt{m} \log N)$ worst case update time.

Our $(3+\epsilon)$-approximation algorithm is derived from known schemes which bucket edges based on their weights. The rounding scheme we use in this algorithm is based on algorithm designed by Anand et al. [2]. It is not clear whether any extension of this bucketing scheme will lead to a $(1+\epsilon)$-approximate matching. To do this, we use another rounding scheme designed by Lingas and Di[17]. Using this new rounding scheme, we obtain arbitrarily good approximations of maximum matching, albeit at the cost of a much higher dependency on $1/\epsilon$ in the running time.

Theorem 1.3: For any constant $\epsilon < 1/2$, there exists an algorithm that maintains $(1+\epsilon)$-approximate maximum weighted matching in a graph where edges have weights between $[1, N]$ in $O(\sqrt{m} \log N)$ worst case update time.

As with the algorithm by Neiman and Solomon [20], our algorithms are deterministic and the update time guaranteed by them is worst case. However, for simplicity in our presentations we will often start by describing the simpler amortized variants.

2. Preliminaries

We start by stating the notations that we will use, and reviewing some well-known results on matchings. An undirected graph is represented by $G = (V, E)$, where V represents the set of vertices and E represents the set of edges in the graph. We will use n to denote the number of vertices $|V|$, and m to denote the number of edges $|E|$.

A *matching* in a graph is a set of independent edges in the graph. Specifically, a subset of edges, $M \subseteq E$ is a matching if no vertex of the graph is incident on more than one edge in M. A vertex is called *unmatched* if it is not incident on any edge in M, otherwise it is *matched*. Similarly, an edge is called *matched* if it is in M or *free* otherwise. A vertex cover is a set of vertices in a graph such that each edge has at least one of its endpoint in the vertex cover.

The maximum cardinality matching(MCM) in a graph is the matching of maximum size. Similarly, given a set of weights $w : E \rightarrow [1, N]$, we can denote the weight of a matching M as $w(M) = \sum_{e \in M} w(e)$. The maximum weight matching(MWM) in a graph is in turn the matching of maximum weight. We will use $\mathcal{M}$ to denote a optimum matching for either of these two objectives depending on context.

For measuring the quality of approximate matching, we will use the notation of α-approximation, which indicates that the objective (either cardinality or weight) given by the current solution is at least $1/\alpha$ of the optimum. Specifically, a matching M is called α-MCM if $|M| \geq \frac{1}{\alpha}|MCM|$, and α-MWM if $w(M) \geq \frac{1}{\alpha}|MWM|$.

Finding or approximating MCMs and MWMs in the static setting have been intensely studied. Nearly linear time algorithms have been developed for finding $(1+\epsilon)$ approximations, and we will make crucial use of these algorithms in our data structure. For maximum cardinality matching, such an algorithm for bipartite graph was introduced by Hopcroft and Karp[15], and extended to general graphs by Micali and Vazirani[19], [25].

Lemma 2.1: There exists an algorithm APPROXMCM that when given a graph G with m edges along with a parameter $\epsilon < 1$, return an $(1+\epsilon)$-MCM in $O(m\epsilon^{-1})$ time.

For approximate MWM, there has been some recent progress. Duan et al.[6], [7] designed an algorithm that find a $(1+\epsilon)$ approximate maximum weighted matching in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ time.

Lemma 2.2: [6], [7] There exists an algorithm APPROX-MWM that when given a graph G with m edges along with a parameter $\epsilon < 1$, return an $(1+\epsilon)$-MWM in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ time.

All logarithms in this paper are with base 2 unless mentioned otherwise.

3. $(1+\epsilon)$-MCMS USING LAZY UPDATES

3.1. Overview

To maintain approximate matching, we exploit the *stability* of the matching and use the static algorithm for matching APPROXMCM periodically. Our starting point is the observation that the size of maximum matching changes by at most 1 per update. This means that if we have a large matching that's close to the maximum, it will remain close to maximum matching over a large number of updates. So we use the following approach: Find a matching at certain update step and wait for certain number of updates till the matching is a good approximation of maximum matching. This approach works well if the maximum matching is itself large to begin with. But if the maximum matching itself is small, we still need to run the static algorithm many times.

To overcome this, we show that instead of finding a maximum matching on the entire graph, we can use a small *special* subgraph such that the size of maximum matching in this subgraph is same as the size of maximum matching in the entire graph. We call this subgraph a *core subgraph*, and it is the central idea of our $(1+\epsilon)$ approximate algorithm. As this subgraph is considerably smaller, the time needed to find a maximum matching on it is considerably less. We will show that this *core subgraph* can be formed using the vertex cover of the entire graph. Specifically, we take the vertex-induced subgraph formed by the cover, along with some *special* chosen edges out of vertices belonging to the cover.

But this leads to another question: How do we maintain a vertex cover in a dynamic graph? For this, we can use the algorithm of Neiman and Solomon [20]. One of the invariants in this algorithm is that there are no edges between unmatched vertices, which means the set of matched vertices form a 2-approximate minimum vertex cover. Therefore reporting these vertices suffices for a vertex cover at any update step. However, note that our dependence on the above algorithm is not critical. Specifically, we design another simple algorithm which does not depend on the algorithm of Neiman and Solomon[20] for finding the *core subgraph*. A description of this, as well as modifications for handling edges with weights in a small range, and obtaining worst case bounds are in Section 3.3

3.2. Algorithm

We start with some notations that we will use in this section. We number the updates from 1 to t and use the following notations:

- $G(i)$: The graph after the i$^{\text{th}}$ update.
- $M(i)$: A matching computed on $G(i)$
- $M(i \setminus j)$: Let $del_M(i,j)$ denote the set of all edges in $M(i)$ that are deleted from the graph between update steps i and j. We define $M(i \setminus j)$ to be $M(i) \setminus del_M(i,j)$, i.e., $M(i \setminus j)$ consists of all the edges in the matching $M(i)$ that are not deleted between update step i and j.

Also, we will use $\mathcal{M}(i)$ to denote the optimal matching at step i. The approximation guarantees of $M(i \setminus j)$ is as follows:

Lemma 3.1: If $\epsilon, \epsilon' \leq 1/2$ and $M(i)$ is an $(1+\epsilon)$-MCM in $G(i)$, then for $j \leq i + \epsilon'|M(i)|$, $M(i \setminus j)$ is an $(1+2\epsilon+2\epsilon')$-MCM in $G(j)$

Proof: Suppose there were k_{ins} insertions and k_{del} deletions in the $k = \epsilon'|M(i)|$ updates between updates i and j. The assumption about $M(i)$ implies that $|\mathcal{M}(i)| \leq (1+\epsilon)|M(i)|$. Since each insert can increase the size of the maximum matching by 1, we have $|\mathcal{M}(j)| \leq |\mathcal{M}(i)| + k_{ins}$. Also, each deletion can remove at most one edge from $M(i)$, so $|M(i \setminus j)| \geq |M(i)| - k_{del}$. The approximation ratio is then at most:

$$\begin{aligned}
\frac{|\mathcal{M}(j)|}{|M(i \setminus j)|} &\leq \frac{(1+\epsilon)|M(i)| + k_{ins}}{|M(i)| - k_{del}} \\
&= 1 + \frac{\epsilon|M(i)| + k}{|M(i)| - k_{del}} \\
&\leq 1 + \frac{\epsilon|M(i)| + \epsilon'|M(i)|}{1/2|M(i)|} \\
&\qquad \text{Since } k_{del} \leq \epsilon'|M(i)| \leq 1/2|M(i)| \\
&\leq 1 + 2\epsilon + 2\epsilon'
\end{aligned}$$

■

This fact has immediate algorithmic consequences for situations where the maximum matching is large. Suppose we computed an $(1+\epsilon/4)$-MCM for $G(i)$, $M(i)$, then $M(i \setminus j)$ is $(1+\epsilon)$ approximate maximum matching as long as $j \leq i + \epsilon|M(i)|/4$. The $O(m\epsilon^{-1})$ cost of the call to APPROXMCM (given by Lemma 2.1) can then be charged to the next $\epsilon|M(i)|/4$ updates, giving $O(\frac{m}{|M(i)|}\epsilon^{-2})$ time per update. When $|M(i)|$ is large, this cost is fairly small. On the

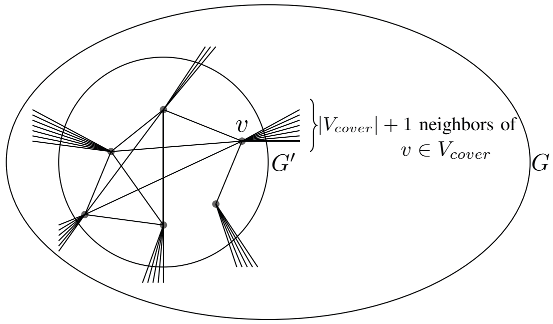

Figure 1. An example showing the *core subgraph* G' of G. All the vertices in the inner circle form a vertex cover V_{cover} of G. The *core subgraph* contains all the edges induced by the vertices in V_{cover} plus at most $|V_{cover}| + 1$ edges from each vertex $v \in V_{cover}$ whose other endpoint is not in the vertex cover

other hand, when $|M(i)|$ is of constant size, this approach will make a call to ApproxMCM almost every update.

For small size matching, we introduce the concept of *core subgraph*. As mentioned previously, *core subgraph* can be found by using a vertex cover G.

Definition 3.2: Given a graph G and a vertex cover V_{cover}, a core subgraph G' consists of:

- All edges between vertices in V_{cover}
- For each vertex $v \in V_{cover}$, the $|V_{cover}| + 1$ edges of maximum weight of v to vertices in $V \setminus V_{cover}$. In case of an unweighted graph, these edges can be chosen arbitrarily.

An illustration of a *core subgraph* is shown in Figure 1. It can be used algorithmically as follows.

Lemma 3.3: Let G' be a core subgraph of G formed using a vertex cover $V_{cover} \subseteq V$. If M' is a $(1+\epsilon)$-MCM in G', then it's also a $(1+\epsilon)$-MCM in G.

Proof: We first show that the size of the maximum matching in G is the same as the size of the maximum matching in G'. Among all maximum matchings in G, let $\mathcal{M}$ be one that uses the maximum number of edges in $E(G')$. For the sake of contradiction, suppose $\mathcal{M}$ contains an edge (u, v) in $E(G) \setminus E(G')$. Since V_{cover} is a vertex cover, one of u or v is in V_{cover}, without loss of generality assume it's u. By the construction rule, for (u, v) to not be included in G', there exist $|V_{cover}| + 1$ neighbors of u in $V \setminus V_{cover}$ that are in G', let them be $N_{V \setminus V_{cover}}(u)$. As the maximum matching in G has size at most $|V_{cover}|$ and there are no edges with both endpoints in $V \setminus V_{cover}$, at most $|V_{cover}|$ vertices in $N_{V \setminus V_{cover}}(u)$ can be matched. Therefore there exists an unmatched vertex x in $N_{V \setminus V_{cover}}(u)$. Substituting (u, v) with (u, x) gives a maximum matching that uses one more edge in G', giving a contradiction.

Combining this with the fact that $E(G') \subseteq E(G)$ implies that the size of the maximum matchings in G and G' are the same. Therefore any $(1+\epsilon)$-MCM in G' is also a $(1+\epsilon)$-MCM in G. ∎

As mentioned previously, we can find V_{cover} in the graph by using the algorithm of Neiman and Solomon [20]. Their algorithm maintains 3/2 approximate matching in $O(\sqrt{m})$ update time in the worst case which is less than the bound we are claiming. Whenever we need a vertex cover, we can report all the matched vertices in the 3/2 approximate matching. From now on we will assume an oracle access to the vertex cover at any update step.

Any vertex cover V_{cover} in the graph $G(i)$ formed out of a valid matching has the following property: $|V_{cover}| \leq 2|\mathcal{M}(i)|$. This is because the size of any valid matching is always less than the maximum matching size $|\mathcal{M}(i)|$. Therefore when $|\mathcal{M}(i)|$ is small, we only need to run the static algorithm given by Lemma 2.1 on a *core subgraph* $G'(i)$ of $G(i)$. We can construct this graph in $O(|V_{cover}|^2)(= O(|\mathcal{M}(i)|^2))$ time by examining up to $O(|V_{cover}|)$ neighbors of each vertex in V_{cover}. Using Lemma 2.1, we can find a $(1+\epsilon)$ approximate matching in this graph in $O(|\mathcal{M}(i)|^2\epsilon^{-1})$ time. Furthermore, Lemma 3.1 allows us to charge this $O(|\mathcal{M}(i)|^2\epsilon^{-1})$ time to the next $\epsilon|\mathcal{M}(i)|/4$ updates. Therefore, cost charged per update can be bounded by $O(|\mathcal{M}(i)|\epsilon^{-2})$, which is small for small values of $|\mathcal{M}(i)|$. Our data structure maintains the following global states:

1) A matching M.
2) A counter t indicating the number of updates until we make the next call to ApproxMCM
3) A vertex cover V_{cover} (Using the algorithm of Neiman and Solomon [20])

We assume that the graph is empty initially. So $M = \emptyset$ at the start of the algorithm. Since we handle insertions and deletions in almost symmetrical ways, we present them as a single routine Update, shown in Figure 2

```
Procedure Update(u, v)
if Update is a deletion and (u, v) ∈ M then
    Remove (u, v) from M;
t ← t − 1 ;
if t ≤ 0 then
    Construct a core subgraph G′ of the current
    graph;
    M ← ApproxMCM(G′, 1 + ε/4) ;
    t ← ε/4|M| ;
```

Figure 2. Lazy update algorithm for maintaining $(1+\epsilon)$-MCMs

The bounds of this routine is as follows:

Theorem 3.4: The matching M is an $(1+\epsilon)$-MCM over all updates. Furthermore, the amortized cost per update is $O(\sqrt{m}\epsilon^{-2})$.

Proof: Let the current update be at time j, and the matching M that we maintained was computed in iteration $i < j$. So at update step i, the matching M is same as $M(i)$ and at update step j, it is $M(i \setminus j)$. If $t > 0$, then since t

was initialized to $\epsilon/4|M(i)|$, we have $j - i \leq \epsilon/4|M(i)|$. The guarantees for $M(i \setminus j)$ follows from Lemma 3.1 with $\epsilon \leftarrow \epsilon/4$ and $\epsilon' \leftarrow \epsilon/4$.

We now turn our attention to running time. Consider a call to APPROXMCM made at update i. Assume that $\epsilon|M(i)| \geq 1$. We have seen that there exists a *core subgraph* $G'(i)$ such that the number of edges $|E(G'(i))|$ can be bounded by $O(\min\{m, |\mathcal{M}(i)|^2\})$. Since $M(i)$ is a $(1+\epsilon/4)$ approximate matching, $(1 + \epsilon/4)|M(i)| \geq |\mathcal{M}(i)|$. So, the size of $E(G'(i))$ is $O(\min\{m, |M(i)|^2\})$. Moreover, the cost of finding the matching(in APPROXMCM) in the graph is at most $O(\min\{m, |M(i)|^2\}\epsilon^{-1})$. This cost can be charged to the $\epsilon|M(i)|/4$ updates starting at update i, implying the following amortized cost per update:

$$\frac{O(\min\{m, |M(i)|^2\}\epsilon^{-1})}{\frac{\epsilon}{4}|M(i)|} = O\left(\min\left\{\frac{m}{|M(i)|}, |M(i)|\right\}\epsilon^{-2}\right)$$

If $|M(i)| \geq \sqrt{m}$, the first term inside min is at most $\sqrt{m}$, otherwise the second is at most $\sqrt{m}$. Combining these two cases gives our desired bound.

Now we take a look at some corner cases to complete the proof. We assumed that the cost of finding the matching at level i can be charged to next $\epsilon|M(i)|/4$ updates. This is true except for last call to APPROXMCM. The number of updates after this last call can be less than $\epsilon|M(i)|/4$. This cost can be amortized to all the updates. Since the number of updates is at least m, the total cost charged to each update step is $O(\epsilon^{-1})$.

The other case is when $\epsilon|M(i)| < 1$. This implies that $G'(i)$ has size at most $O(\epsilon^{-2})$ and finding a matching in such a graph takes time $O(\epsilon^{-3})$. For any constant ϵ, this bound is $O(\sqrt{m}\epsilon^{-2})$ and can be charged to the update step itself. So the amortized cost charged to any update step is at most $O(\sqrt{m}\epsilon^{-2})$. ∎

3.3. Improvements, Worst-Case Bound, and Weights

Several improvements can be made to the simpler version of our algorithm described above. Due to space constraints we only state the main statements here, and more details on these modifications can be found in the arXiv version of our paper [12].

First, note that we depend on the algorithm of Neiman and Solomon [20] to maintain approximate vertex cover. Instead of using their algorithm, we design another simple dynamic algorithm which maintains approximate vertex cover. This algorithm is similar in spirit as our approximate matching algorithm, i.e, we use the property that vertex covers are also *stable* and a single update to the graph can change the vertex cover by 1. Using the techniques similar to the one presented in the previous section, we design an algorithm in which take $O(\sqrt{m})$ update time in the worst case to maintain approximate vertex cover.

Note that APPROXMCM may take $O(m\epsilon^{-1})$ time in the worst case. So our algorithm in the previous section had an *amortized* running time of $O(\sqrt{m}\epsilon^{-2})$ per update. We show that we can maintain approximate matching in worst case $O(\sqrt{m}\epsilon^{-2})$ update time. Specifically, we show that computation cost of $O(m\epsilon^{-1})$ time in APPROXMCM can be distributed across a number of updates.

Furthermore, our ideas of maximum cardinality matching can also be adapted to maximum weighted matchings. This extension is natural because maximum cardinality matchings is a special case where all edges have weight 1. A closer examination of the proofs of Lemma 3.1 shows that when all edge weights are in the range $[1, N]$, the stability properties only degrade by a factor of N. In [12], we present the following result:

Theorem 3.5: There exists an algorithm that maintains $(1 + \epsilon)$-approximate maximum weighted matching in a graph where edges have weights between $[1, N]$ in $O(\sqrt{m}N\epsilon^{-2}\log(\epsilon^{-1}))$ update time.

4. APPROXIMATE WEIGHTED MATCHINGS WITH POLYLOG DEPENDENCY ON N

We now show algorithms that approximate the maximum weighted matching in time that depends on $\log N$ instead of **poly**(N). This reduced dependency on N is a subject of study in static algorithms since N is often **poly**(n) or larger.

Our overall scheme is based on the data structure for weighted matchings by Anand et al. [2], [3]. Their algorithm maintains $\log N$ levels and the edges are partitioned across various levels according to their weights. A matching M_l is maintained at each level l, and they gave a way to form a single matching $\hat{M}$ from these $\log N$ matchings. Algorithmically, it can be viewed as adding an edge $(u, v) \in M_l$ to $\hat{M}$ and removing all edges incident to u and v from all $M_{l'}$s where $l' < l$. At any update step, the matching maintained is equivalent to the one generated in Figure 3.

```
M̂ = ∅;
Let l_max and l_min be the maximum and minimum
level number respectively;
for l = l_max to l_min do
    M̂ = M̂ ∪ M_l;
    for (u, v) ∈ M_l do
        Remove all the edges adjacent to u and v
        from M_l' such that l' < l
```

Figure 3. Generating $\hat{M}$

Anand et al. [2], [3] showed that the combined matching $\hat{M}$ can be maintained on a dynamic graph if the matching at each level l can be maintained. We will use their result as a black-box via. the following Lemma.

Lemma 4.1: ([3]) If the matching on each level is maintained in $O(f(m, n))$ update time, then the overall matching can be maintained in $O(f(m, n)\log N)$ update time.

In their work, $f(m,n) = O(\log n)$ due to the use of the dynamic maximal matching data structure by Baswana et al.[4], which leads to a total bound of $O(\log n \log N)$. We will substitute our algorithms in place of this algorithm, and investigate different leveling schemes which lead to improved approximation ratios. This comes at a cost of a higher value of $f(m,n) = O(\sqrt{m}\mathbf{poly}(\epsilon^{-1}))$, which leads to a time of $O(\sqrt{m}\log N\mathbf{poly}(\epsilon^{-1}))$ per update.

In Section 4.1, we present a deterministic algorithm which maintains a $(3+\epsilon)$-MWM in $O(\sqrt{m}\log N\epsilon^{-3})$ time, and in Section 4.2, we given an alternate approach which maintains a $(1+\epsilon)$-approximate MWM in $O(\sqrt{m}\log N\epsilon^{-2-O(\epsilon^{-1})})$ time per update. Note that in both the above algorithm, we will maintain approximate MCM or MWM matching at each level. For this we can use the amortized and worst-case versions of our data structures described in Section 3 and Section 3.3 leading to corresponding types of final bounds for the above algorithm.

In many of our proofs, we will incur $(1+O(\epsilon))$ multiplicative error in several places. As a result, the final approximation factors in our calculations will often be $1+c\epsilon$ for some constant c. Such bounds can be converted to $1+\epsilon$ approximations by initiating the calls with smaller values of ϵ. As a result, we will omit these steps to simplify presentation.

4.1. $(3+\epsilon)$-*Approximation Using Approx MCMs*

We first show that our data structure for maintaining $(1+\epsilon)$-MCMs given in Theorem 1.1 can be used on each level. The transformation for turning a MWM problem into a set of $O(\log N)$ MCM instances is based on a rounding scheme by Eppstein et al. [8], [9]. For a fixed value of r, we assign an edge e with $w(e) \in [\alpha^{l+r}, \alpha^{l+r+1})$ to level l where α is a constant which we will calculate later. Note that the level of some edges can be -1, but our proof below can extend to any negative level as well. We define the rounded weight of an edge e assigned to level l using:

$$w_r(e) \stackrel{\text{def}}{:=} \alpha^{l+r}$$

Our analysis of the quality of $\hat{M}$ is based on mapping each edge in $\hat{M}$ to a set of edges in M_l's. For $e = (u,v) \in \hat{M}$ from level l, we define $\mathcal{R}(e)$ as:

$$\mathcal{R}(e) = \{e\} \cup \{(x,y) \mid (x,y) \in M_{l'} \text{ where } l' < l, \text{ and } \{x,y\} \cap \{u,v\} \neq \emptyset\}$$

In other words, $\mathcal{R}(e)$ contains edge e and all those edges adjacent to u and v from lower levels that were removed when (u,v) was added to $\hat{M}$. Note that e is the only edge in $\mathcal{R}(e)$ from level l. And for all $l' < l$, there can be at most 2 edges from level l' in $\mathcal{R}(e)$. To simplify our notations, we will use $w(S)$ to denote the total weight of a set of edges S (that could be either $\hat{M}$, $\mathcal{M}$ or M_l for some l')

For an edge $e \in \hat{M}$, let $\Phi(e)$ denote the total rounded weights of edges in $\mathcal{R}(e)$, i.e., $\Phi(e) = w_r(\mathcal{R}(e))$. We can show that $\Phi(e)$ is closely related to $w_r(e)$.

Lemma 4.2: For $e \in \hat{M}$,

$$\Phi(e) \leq \frac{\alpha+1}{\alpha-1} w_r(e)$$

Proof: Let $e \in \hat{M}$ be on level i. Since there is at most 1 edge on level i assigned to e (e itself) and 2 edges per level assigned to e for each level $j < i$, we have:

$$\begin{aligned}
\Phi(e) = &\sum_{e' \in \mathcal{R}(e)} w_r(e') \\
=& w_r(e) + \sum_{j<i} \sum_{e' \in M_j \,\&\, e' \in \mathcal{R}(e)} w_r(e') \\
\leq& \alpha^{i+r} + \sum_{j<i} 2\alpha^{j+r} \\
\leq& \alpha^{i+r}\left(1 + 2\sum_{j<i}\alpha^{j-i}\right) \\
=& w_r(e)\left(1 + 2\frac{1}{\alpha-1}\right) \\
=& \frac{\alpha+1}{\alpha-1} w_r(e)
\end{aligned}$$

■

This allows us to relate the weight of $\hat{M}$ to the weight of the optimum matching, $\mathcal{M}$.

Lemma 4.3:

$$(1+\epsilon)\frac{\alpha+1}{\alpha-1} w(\hat{M}) \geq w_r(\mathcal{M})$$

Proof:

Let $\mathcal{M}(i)$ denote the edges of $\mathcal{M}$ at level i. Since M_i is a $(1+\epsilon)$ approximate matching at level i, we have:

$$\begin{aligned}
|\mathcal{M}(i)| \leq& (1+\epsilon)|M_i| \\
w_r(\mathcal{M}(i)) \leq& (1+\epsilon) w_r(M_i)
\end{aligned}$$

Since edges on same level have the same values of $w_r(e)$

$$w_r(\mathcal{M}) \leq (1+\epsilon)\sum_i w_r(M_i)$$

Consider an edge $e = (u,v) \in M_i$. If $e \in \hat{M}$, then $e \in \mathcal{R}(e)$. If $e \notin \hat{M}$, then there exists an edge $e' \in \hat{M}$ at level $j > i$ such that one of the endpoints of e' is either u or v, which means e is in the set $\mathcal{R}(e')$. Therefore each edge e can be mapped to one or more $\mathcal{R}(e')$, and we have:

$$\Phi(\hat{M}) \geq \sum_i w_r(M_i)$$

Which implies $(1+\epsilon)\Phi(\hat{M}) \geq w_r(\mathcal{M})$. Applying Lemma 4.2 on all edges in $\hat{M}$ then gives:

$$(1+\epsilon)\frac{\alpha+1}{\alpha-1} w_r(\hat{M}) \geq w_r(\mathcal{M})$$

And the result follows from the fact that the rounded down edge weights satisfy $w_r(e) \le w(e)$. ■

Hence, it suffices to bound ratio between $w_r(\mathcal{M})$ and $w(\mathcal{M})$. The analysis in Anand et al.[3] bounded this ratio over a uniformly random choices of r. They showed that the expected rounded value of the optimum matching, $\mathbf{E}_r[w_r(\mathcal{M})]$ satisfies $\mathbf{E}_r[w_r(\mathcal{M})] \ge \dfrac{\alpha-1}{\alpha \ln \alpha} w(\mathcal{M})$, which when combined with Lemma 4.3 leads to an expected approximation ratio of about $3+\epsilon$ when $\alpha \approx 5.704$. Here we show instead that a deterministic and worst-case bound can be obtained by using $O(1/\epsilon)$ versions of our data structure, each with a pre-selected value of r.

We have $k = \ln \alpha / \ln(1+\epsilon)$ copies of our algorithm which work exactly identically but with different value of r. For the j^{th} copy, $r(j) = \frac{j-1}{k}$. Consider an edge e such that $w(e) = \alpha^{i+\delta}$ where $0 < \delta \le 1$. Let j^* is the value such that $\frac{j^*-1}{k} \le \delta < \frac{j^*}{k}$. Then we have:

$$w_{r(j)}(e) = \begin{cases} \alpha^{i+\frac{j-1}{k}} & \text{if } j \le j^* \\ \alpha^{i+\frac{j-1}{k}-1} & j > j^* \end{cases}$$

Informally, an edge e is at level i in j[th] copy, if $j \le j^*$ otherwise it is at level $i-1$. We want to relate the weight of maximum matching $\mathcal{M}$ in G to the new weight in these k copies. Specifically, we want to get a relation similar to the relation between $\mathbf{E}_r[w_r(\mathcal{M})]$ and $w(\mathcal{M})$ mentioned above. We show that there exists a j with the following relation.

Lemma 4.4: There exists a j such that:

$$w_{r(j)}(\mathcal{M}) \ge (1-\epsilon)\frac{\alpha-1}{\alpha \ln \alpha} w(\mathcal{M})$$

Proof: Summing over all j of $w_{r(j)}(e)$ gives: $\frac{\sum_{j=1}^{k} w_{r(j)}(e)}{w(e)}$

$$\begin{aligned}
&= \frac{\sum_{j=1}^{j^*} \alpha^{i+\frac{j-1}{k}} + \sum_{j=j^*+1}^{k} \alpha^{i+\frac{j-1}{k}-1}}{\alpha^{i+\delta}} \\
&= \frac{\sum_{j=1}^{j^*} \alpha^{\frac{j-1}{k}} + \sum_{j=j^*+1}^{k} \alpha^{\frac{j-1}{k}-1}}{\alpha^{\delta}} \\
&= \alpha^{-\delta+\frac{j^*-1}{k}} \left(\sum_{j=1}^{j^*} \alpha^{j/k-j^*/k} + \sum_{j=j^*+1}^{k} \alpha^{j/k-j^*/k-1} \right) \\
&= (1+\epsilon)^{-k\delta+j^*-1} \left(\sum_{j=1}^{j^*} (1+\epsilon)^{j-j^*} + \sum_{j=j^*+1}^{k} (1+\epsilon)^{j-j^*-k} \right)
\end{aligned}$$

Since j^* was chosen such that $\frac{j^*}{k} > \delta$, $-k\delta + j^* - 1 \ge -k(\frac{j^*}{k}) + j^* - 1 = -1$ and $(1+\epsilon)^{-k\delta+j^*-1} \ge (1+\epsilon)^{-1}$. Substituting this gives: $\frac{\sum_{j=1}^{k} w_{r(j)}(e)}{w(e)} \ge$

$$(1+\epsilon)^{-1} \left(\sum_{j=1}^{j^*} (1+\epsilon)^{j-j^*} + \sum_{j=j^*+1}^{k} (1+\epsilon)^{j-j^*-k} \right)$$

The two summations are a rearranged version of a geometric sum. It can be rearranged by substituting $l = j^* - j + 1$ and $l = j^* - j + k + 1$ in the first and second summation respectively to obtain:

$$\begin{aligned}
\frac{\sum_{j=1}^{k} w_{r(j)}(e)}{w(e)} &= (1+\epsilon)^{-1} \left(\sum_{l=1}^{j^*} (1+\epsilon)^{-l+1} + \sum_{l=j^*+1}^{k} (1+\epsilon)^{-l+1} \right) \\
&= \sum_{l=1}^{k} (1+\epsilon)^{-l} \\
&= \frac{1-(1+\epsilon)^{-k}}{\epsilon} \\
&= \frac{(1-1/\alpha)}{\epsilon} \\
&= \frac{\alpha-1}{\alpha\epsilon}
\end{aligned}$$

Summing this over all edges in $\mathcal{M}$ gives:

$$\begin{aligned}
\sum_{e \in \mathcal{M}} \sum_{j} w_{r(j)}(e) &\ge \sum_{e \in \mathcal{M}} \frac{\alpha-1}{\alpha\epsilon} w(e) \\
\sum_{j} w_{r(j)}(\mathcal{M}) &\ge \frac{\alpha-1}{\alpha\epsilon} w(\mathcal{M})
\end{aligned}$$

By an averaging argument we get:

$$\begin{aligned}
\max_j \{w_{r(j)}(\mathcal{M})\} &\ge \frac{1}{k} \sum_{j} w_{r(j)}(\mathcal{M}) \\
&\ge \frac{\alpha-1}{\alpha\epsilon k} w(\mathcal{M})
\end{aligned}$$

Note that $k = \ln \alpha / \ln(1+\epsilon)$. Here we make use of the following known fact

Fact 4.5: For $\epsilon < 1$, if $0 \le x \le \epsilon$, then $\ln(1+x) \ge (1-\epsilon)x$.

Applying it with $x = \epsilon$ gives:

$$\begin{aligned}
\max_j \{w_{r(j)}(\mathcal{M})\} &= \frac{(\alpha-1)\ln(1+\epsilon)}{\alpha\epsilon \ln \alpha} w(\mathcal{M}) \\
&\ge \frac{(\alpha-1)(1-\epsilon)\epsilon}{\alpha\epsilon \ln \alpha} w(\mathcal{M}) \qquad \text{By Fact 4.5} \\
&= (1-\epsilon)\frac{\alpha-1}{\alpha \ln \alpha} w(\mathcal{M})
\end{aligned}$$

■

Combining Lemmas 4.3 and 4.4 gives the following theorem.

Theorem 4.6: For any $\epsilon < 1/2$, there exists a fully dynamic algorithm that maintains a $(3+\epsilon)$-MWM for any graph on n in worst case $O(\sqrt{m} \log N \epsilon^{-3})$ time per update.

Proof: Consider maintaining k copies of our data structure and picking the maximum weighted matching among these copies as the current best matching.

Using Lemma 4.3, we get:

$$\forall j \qquad (1+\epsilon)\frac{\alpha+1}{\alpha-1} w(\hat{M}(j)) \ge w_{r(j)}(\mathcal{M})$$

Using Lemma 4.4, there exists a $j' = \arg\max_j \{w(\hat{M}(j))\}$ such that

$$w_{r(j')}(\mathcal{M}) \geq (1-\epsilon)\frac{\alpha-1}{\alpha \ln \alpha} w(\mathcal{M})$$

Combining the above two equations we get:

$$\begin{aligned}(1+\epsilon)\frac{\alpha+1}{\alpha-1} w(\hat{M}(j')) \geq& (1-\epsilon)\frac{\alpha-1}{\alpha\ln\alpha} w(\mathcal{M}) \\ \left(\frac{1+\epsilon}{1-\epsilon}\right)\frac{(\alpha+1)\alpha\ln\alpha}{(\alpha-1)^2} w(\hat{M}(j')) \geq& w(\mathcal{M})\end{aligned}$$

Where one can check that $\frac{1+\epsilon}{1-\epsilon} \leq (1+4\epsilon)$ when $\epsilon < 1/2$. By a suitable choice of ϵ, this factor of $1+4\epsilon$ can be turned into $1+\epsilon'$. This implies that the approximation ratio obtained by our algorithm is $\frac{(1+\epsilon)(\alpha+1)\alpha\ln\alpha}{(\alpha-1)^2}$. This term achieves its minimum value of $\approx 3+3\epsilon$ when $\alpha \approx 5.704$. Again this approximation ratio can be turned into $3+\epsilon'$ by a suitable choice of ϵ.

For the update time, note that since α is a constant, $k = O(1/\log(1+\epsilon)) = O(1/\epsilon)$ copies of the structure are needed. In each such copy, a matching can be maintained in $O(\sqrt{m}\log N\epsilon^{-2})$ update time. So matching in all the copies can be maintained in $O(\sqrt{m}\log N\epsilon^{-3})$ time per update. ■

4.2. $(1+\epsilon)$-*MWMs Using Approximate MWMs*

Overview: In this section, we present an algorithm that maintains a $(1+\epsilon)$-MWM using a more gradual bucketing scheme. This bucketing scheme was first used by Lingas and Di[17]. We start by observing the definition of $\mathcal{R}(e)$ for an edge $e(u,v)$ in $\hat{M}$ from the previous section. Informally, $\mathcal{R}(e)$ contains edge e and all those edges adjacent to u and v from lower levels that were removed when (u,v) was added to $\hat{M}$. A closer look at our algorithm reveals that the approximation ratio depends on the ratio of weight of e and the combined weight of edges in $\mathcal{R}(e)$. This ratio can be reduced if the edges at lower level have significantly less weight than the weight of edge e. To achieve this, we will artificially create levels such that the ratio of weight between two consecutive level is significant. For this, we will drop some edges from the graph to create a *gap* between two consecutive levels. In order to account for the weight of these dropped edges, we in turn need to keep several copies of our data structure with different edges left out in the other copies.

We then proceed with the same algorithm as mentioned in Section 4 with one main difference. Instead of maintaining $(1+\epsilon)$-MCM at each level, we maintain $(1+\epsilon)$-MWM at each level using the Theorem 3.5. Note that this theorem has a dependence of N in its running time. We will show that each level can be formed in such a way that N can be bounded by $O(\epsilon^{-O(\epsilon^{-1})})$. So the running time for maintaining $(1+\epsilon)$ approximate MWM at each level will have exponential dependence on $(1/\epsilon)$.

Thereafter, we combine the matching across the various level using the same procedure as mentioned in Section 4.

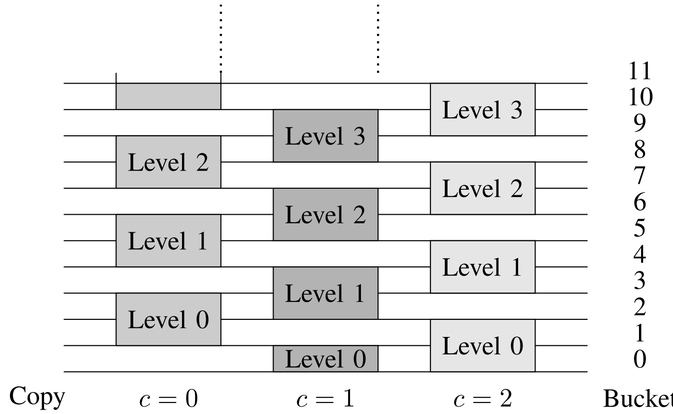

Figure 4. Bucketing and level scheme where $C = 3$

We will show that there exists a copy of our data structure such that the weight of the matching maintained by our algorithm in that copy is a good approximation of maximum weighted matching in the entire graph.

Algorithm: Once again we partition the edges by weights geometrically: an edge e is in bucket b if $w(e)$ is in the range $[\epsilon^{-b}, \epsilon^{-(b+1)})$. However, our levels no longer corresponds to individual buckets, but instead to a set of $C-1$ continuous buckets for a value of C to be specified later. We will also remove some of these buckets, and the choices of buckets to remove leads us to run several copies of our data structure simultaneously.

We will run $C = \lceil \epsilon^{-1} \rceil$ copies of our algorithm, where in the c^{th} copy, we remove all buckets i such that $i \mod C = c$. This leads to a set of graphs $G^0 \ldots G^{C-1}$. Removing the buckets creates natural partitions of the remaining edges, which gives our levels. For a copy c, we will place buckets with $b = [lC+c+1 \ldots (l+1)C+c-1]$ into level l. Note that the ratio of maximum to minimum edge weight in each level is bounded by $\epsilon^{-(C-2)} (= O(\epsilon^{-O(\epsilon^{-1})}))$. Therefore, the algorithm given in Theorem 3.5 allows us to maintain an $(1+\epsilon)$-MWM in $O(\epsilon^{-O(\epsilon^{-1})}\sqrt{m}\epsilon^{-2}\log(\epsilon^{-1})) = O(\epsilon^{-2-O(\epsilon^{-1})}\sqrt{m}\log(\epsilon^{-1}))$ time at each level. These matchings can in turn be combined together in the same way as in Section 4. An illustration of leveling scheme used by our algorithm is shown in Figure 4.

We start by analyzing the guarantees of our algorithm on the c$^{\text{th}}$ copy. Specifically, the approximation ratio of the combined matching $\hat{M}^c$ w.r.t. the maximum matching $\mathcal{M}^c$ in this copy . Let $\mathcal{M}^c_l$ be the edges of $\mathcal{M}^c$ at level l. Also let M^c_l denote the matching maintained at each level using Theorem 3.5. Once again, for an edge $e = (u,v)$ in $\hat{M}^c$ at level l, we define $\mathcal{R}^c(e)$ as:

$$\begin{aligned}\mathcal{R}^c(e) = \{e\} \cup & \{(x,y) | (x,y) \in M^c_{l'} \\ & \text{where } l' < l, \text{ and } \{x,y\} \cap \{u,v\} \neq \emptyset\}\end{aligned}$$

For an edge $e \in \hat{M}^c$, let $\Phi^c(e)$ denote the total rounded weights of edges in $\mathcal{R}^c(e)$, i.e., $\Phi^c(e) = w(\mathcal{R}^c(e))$. We can show that $\Phi^c(e)$ is related to $w(e)$ by the following inequality:

Lemma 4.7: For any edge e in the combined matching $\hat{M}^c$, we have:

$$\Phi^c(e) \leq (1 + 3\epsilon)\, w(e)$$

Proof: Assume that e is on level l. Since there are at most 1 edge on level l assigned to e (e itself) and 2 edges per level $l' < l$ assigned to e, we have:

$$\Phi^c(e) = \sum_{e' \in \mathcal{R}^c(e)} w(e') = w(e) + \sum_{l'<l} \sum_{e' \in M^c_{l'} \cap \mathcal{R}^c(e)} w(e')$$

Since an edge on level l' is in bucket $[l'C + c + 1 \ldots (l'+1)C + c - 1]$ and an edge in bucket b has weight at most $(1/\epsilon)^{b+1}$, the weight of an edge at level l' is $\leq \epsilon^{-c-(l'+1)C}$.

$$\Phi^c(e) \leq w(e) + \sum_{l'<l} 2\epsilon^{-c-(l'+1)C}$$

Which can in turn be bounded relative to $w(e)$. Since e is in level l and an edge in bucket b has weight at least $(1/\epsilon)^b$, $w(e) \geq \epsilon^{-lC-c-1}$

$$\begin{aligned}\Phi^c(e) \leq& w(e) + \sum_{l'<l} 2\epsilon^{(l-l'-1)C+1} w(e)\\ \leq& w(e) + \frac{2\epsilon}{1-\epsilon^C} w(e)\\ \leq& w(e)(1+3\epsilon) \qquad \text{assuming that } \epsilon < 1/2\end{aligned}$$

■

We now show the relation between the combined matching $\hat{M}^c$ and $\mathcal{M}^c$.

Lemma 4.8:

$$(1 + 7\epsilon) w(\hat{M}^c) \geq w(\mathcal{M}^c)$$

Proof:

Since M^c_l is a $(1 + \epsilon)$ approximate maximum weighted matching on level l, we have:

$$w(\mathcal{M}^c_l) \leq (1+\epsilon) w(M^c_l), \text{ so } w(\mathcal{M}^c) \leq \quad (1+\epsilon) \sum_l w(M^c_l)$$

Consider an edge $e = (u, v) \in M^c_l$. If $e \in \hat{M}^c$, then $e \in \mathcal{R}^c(e)$. If $e \notin \hat{M}^c$, then there exists an edge e' at level $l' > l$ such that one of the endpoints of e' is either u or v, which means e is in the set $\mathcal{R}^c(e')$. Therefore each edge e can be mapped to one more $\mathcal{R}^c(e')$, and we have:

$$\Phi^c(\hat{M}^c) \geq \sum_l w(M^c_l)$$

Which implies $(1 + \epsilon)\Phi^c(\hat{M}^c) \geq w(\mathcal{M}^c)$. Applying Lemma 4.7 over all edges in $\hat{M}^c$ then gives:

$$(1 + \epsilon)(1 + 3\epsilon) w(\hat{M}^c) \geq w(\mathcal{M}^c)$$

And the bound follows from $(1+\epsilon)(1+3\epsilon) = 1+4\epsilon+3\epsilon^2 \leq 1 + 7\epsilon$ when $\epsilon < 1$. ■

We now find a relation between $w(\mathcal{M}^c)$ and $w(\mathcal{M})$. We show there is at least one copy whose maximum matching has weight at least $(1 - 1/C)$ of $w(\mathcal{M})$.

Lemma 4.9: At any update step, if the maximum weight matching in the current graph is $w(\mathcal{M})$, there exist a copy c such that $w(\mathcal{M}^c) \geq (1 - 1/C)w(\mathcal{M})$.

Proof:

Let $\bar{\mathcal{M}}^c$ denote the set of edges in $\mathcal{M}$ that are not present in the c^{th} copy. Since each bucket is removed in only one copy, we have:

$$\cup_c \bar{\mathcal{M}}^c = \mathcal{M}, \text{ so } \sum_c w(\bar{\mathcal{M}}^c) = \qquad w(\mathcal{M})$$

Since $\mathcal{M} \setminus \bar{\mathcal{M}}^c$ is a matching in G^c, we have $w(\mathcal{M}^c) \geq w(\mathcal{M}) - w(\bar{\mathcal{M}}^c)$. Note that the inequality is due to $\mathcal{M}^c$ being the maximum weighted matching in G^c instead of the restriction of $\mathcal{M}$ on it. Summing over all c copies gives:

$$\begin{aligned}\sum_c w(\mathcal{M}^c) \geq& \sum_c \big(w(\mathcal{M}) - w(\bar{\mathcal{M}}^c)\big)\\ =& C \cdot w(\mathcal{M}) - \left(\sum_c w(\bar{\mathcal{M}}^c)\right)\\ =& (C-1) \cdot w(\mathcal{M})\end{aligned}$$

Dividing both sides by C gives that the average of $w(\mathcal{M}^c)$ is at least $(1 - 1/C)w(\mathcal{M})$. Therefore there exist some c where $w(\mathcal{M}^c) \geq (1 - 1/C)w(\mathcal{M})$.

■

Combining Lemmas 4.9 and 4.8, we deduce that there exists a copy c such that
$(1 + 7\epsilon)w(\hat{M}^c) \geq (1 - \epsilon)w(\mathcal{M})$. So,

$$\frac{w(\mathcal{M})}{w(\hat{M}^c)} \leq \frac{(1+7\epsilon)}{(1-\epsilon)}$$

This ratio is less that $(1 + 16\epsilon)$ for $\epsilon < 1/2$. By a suitable chose of ϵ', the factor of $(1+16\epsilon)$ can be turned into $(1+\epsilon')$.

This means that if we set $C = \lceil \epsilon^{-1} \rceil$ and maintain $(1+\epsilon)$-MWMs on each copy of our data structure, then one of the maximum weight matching among these C copies will always be a good approximation of the maximum weighted matching for the entire graph.

Note that in each copy of our data structure there are $O(\log_{\epsilon^{-1}} N)/C = O\big(\frac{\log N}{C \log(\epsilon^{-1})}\big)$ levels and in each level an approximate MWM is maintained in $O(\epsilon^{-2-O(\epsilon)^{-1}} \sqrt{m} \log(\epsilon^{-1}))$ time. This implies that the overall update time taken by our algorithm across the C copies is $O(C \cdot \frac{\log N}{C \log(\epsilon^{-1})} \sqrt{m} \epsilon^{-2-O(\epsilon)^{-1}} \log(\epsilon^{-1})) = O(\sqrt{m}\epsilon^{-2-O(\epsilon)^{-1}} \log N)$. So we can state the following theorem:

Theorem 4.10: For any $\epsilon < 1/2$, there exists a fully dynamic algorithm that maintains a $(1 + \epsilon)$-MWM in worst case $O(\sqrt{m}\epsilon^{-2-O(1/\epsilon)} \log N)$ time per update.

5. Conclusion

We showed a simpler method for maintaining approximate matchings that maintains $(1+\epsilon)$-approximations in about $\sqrt{m}$ time per update. A challenging question is to design an algorithm with better update time. Since a maximal matching can be maintained in $O(\log n)$ update time [4], it is natural to believe that we may be able to maintain a $(1+\epsilon)$-approximate matching in polylog time. A starting point in this direction might be dense graphs, where $m \approx n^2$. Even for a $(2-\epsilon)$-approximation, no algorithm with $o(n)$ update time is known.

Theoretically our arbitrary quality approximation algorithm from Section 4.2 outperforms the $(3+\epsilon)$ approximation given in Section 4.1. It falls short of a practical algorithm for maintaining $(1+\epsilon)$-MWMs due to an exponential dependency on ϵ^{-1}. We believe a more intricate rounding scheme such as the one given in Section 4.2, or possibly a data structure that incorporates details of the Duan et al. algorithm [6], [7] are promising approaches in this direction.

Acknowledgements

The authors wish to thank Surender Baswana, Gary Miller, Krzysztof Onak, Sandeep Sen, and Danny Sleator for their very helpful comments and discussions. Richard Peng is supported by a Microsoft Research PhD Fellowship. Manoj Gupta is supported by a Microsoft Research India PhD Fellowship.

References

[1] A. Anand, "Fully dynamic algorithms for maintaining approximate matching in graphs," Master's thesis, IIT Kanpur, Kanpur, 2012.

[2] A. Anand, S. Baswana, M. Gupta, and S. Sen, "Maintaining Approximate Maximum Weighted Matching in Fully Dynamic Graphs," in *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 18. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 257–266. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2012/3864

[3] ——, "Maintaining approximate maximum weighted matching in fully dynamic graphs," *CoRR*, vol. abs/1207.3976, 2012.

[4] S. Baswana, M. Gupta, and S. Sen, "Fully dynamic maximal matching in $O(\log n)$ update time," in *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. IEEE, 2011, pp. 383–392.

[5] S. Baswana, S. Khurana, and S. Sarkar, "Fully dynamic randomized algorithms for graph spanners," *ACM Transactions on Algorithms*, vol. 8, no. 4, p. 35, 2012.

[6] R. Duan and S. Pettie, "Approximating maximum weight matching in near-linear time," in *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, ser. FOCS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 673–682. [Online]. Available: http://dx.doi.org/10.1109/FOCS.2010.70

[7] R. Duan, S. Pettie, and H.-H. Su, "Scaling algorithms for approximate and exact maximum weight matching," *CoRR*, vol. abs/1112.0790, 2011.

[8] L. Epstein, A. Levin, J. Mestre, and D. Segev, "Improved approximation guarantees for weighted matching in the semi-streaming model," in *STACS*, 2010, pp. 347–358.

[9] L. Epstein, A. Levin, D. Segev, and O. Weimann, "Improved bounds for online preemptive matching," *CoRR*, vol. abs/1207.1788, 2012.

[10] H. Gabow and R. Tarjan, "Faster scaling algorithms for general graph matching problems," *Journal of the ACM (JACM)*, vol. 38, no. 4, pp. 815–853, 1991.

[11] M. Gupta and A. Sharma, "An $O(\log n)$ Fully Dynamic Algorithm for Maximum matching in a tree," *Arxiv preprint arXiv:0901.2900*, 2009.

[12] M. Gupta and R. Peng, "Fully dynamic $(1+\epsilon)$-approximate matchings," *CoRR*, vol. abs/1304.0378, 2013.

[13] M. R. Henzinger and V. King, "Randomized fully dynamic graph algorithms with polylogarithmic time per operation," *J. ACM*, vol. 46, no. 4, pp. 502–516, 1999.

[14] J. Holm, K. de Lichtenberg, and M. Thorup, "Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity," *J. ACM*, vol. 48, no. 4, pp. 723–760, 2001.

[15] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 225–231, 1973.

[16] Z. Ivkovic and E. Lloyd, "Fully dynamic maintenance of vertex cover," in *Graph-Theoretic Concepts in Computer Science*. Springer, 1994, pp. 99–111.

[17] A. Lingas and C. Di, "Near approximation of maximum weight matching through efficient weight reduction," in *TAMC*, 2011, pp. 48–57.

[18] L. Lovasz and M. Plummer, *Matching Theory*. North-Holland, AmsterdamNew York: AMS Chelsea Publishing, 1986.

[19] S. Micali and V. Vazirani, "An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs," in *Foundations of Computer Science, 1980., 21st Annual Symposium on*. IEEE, 1980, pp. 17–27.

[20] O. Neiman and S. Solomon, "Simple deterministic algorithms for fully dynamic maximal matching," in *STOC*, 2013, pp. 745–754.

[21] K. Onak and R. Rubinfeld, "Maintaining a large matching and a small vertex cover," in *Proceedings of the 42nd ACM symposium on Theory of computing*. ACM, 2010, pp. 457–464.

[22] C. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[23] P. Sankowski, "Faster dynamic matchings and vertex connectivity," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 118–126.

[24] R. E. Tarjan, *Data structures and network algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1983.

[25] V. V. Vazirani, "An improved definition of blossoms and a simpler proof of the mv matching algorithm," *CoRR*, vol. abs/1210.4594, 2012.