Maintaining Approximate Maximum Weighted Matching in Fully Dynamic Graphs

Abhash Anand[1], Surender Baswana[2], Manoj Gupta[3], and Sandeep Sen[4]

1 Indian Institute of Technology, Kanpur
abhash@cse.iitk.ac.in

2 Indian Institute of Technology, Kanpur
sbaswana@cse.iitk.ac.in*

3 Indian Institute of Technology, New Delhi
gmanoj@cse.iitd.ernet.in

4 Indian Institute of Technology, New Delhi
ssen@iitd.ernet.in

Abstract

We present a fully dynamic algorithm for maintaining approximate maximum weight matching in general weighted graphs. The algorithm maintains a matching $\mathcal{M}$ whose weight is at least $\frac{1}{8}M^*$ where M^* is the weight of the maximum weight matching. The algorithm achieves an expected amortized $O(\log n \log \mathcal{C})$ time per edge insertion or deletion, where $\mathcal{C}$ is the ratio of the weights of the highest weight edge to the smallest weight edge in the given graph.

1998 ACM Subject Classification E.1 [Data Structures]: Graphs and Networks; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems; G.2.2 [Graph Theory]: Graph Algorithms

Keywords and phrases Matching, Dynamic Algorithm, Graph Algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2012.257

1 Introduction

Let $G = (V, E)$ be an undirected graph on $n = |V|$ vertices and $m = |E|$ edges. Let there be a weight function $w : E \to \mathbb{R}^+$ such that $w(e)$, for any $e \in E$, represents the weight of e. The weight function for a set of edges $M \subseteq E$ is represented by $w(M)$ and is defined as $\sum_{e \in M} w(e)$.

A subset M of E is a "matching" if no vertex of the graph is incident on more than one edge in M. In an unweighted graph, a maximum matching is defined as the maximum cardinality matching (MCM). In a weighted graph, maximum matching is defined as the matching having maximum weight (MWM). For any $\alpha > 1$, a matching is called α-MWM(α-MCM) if it is at least $\frac{1}{\alpha}$ factor of MWM(MCM).

A dynamic graph algorithm maintains a data structure associated with some property (connectivity, transitive closure, matching) of a dynamic graph. The aim of a dynamic graph algorithm is to handle updates in the graph and answer queries associated with the corresponding property. The updates in the graph can be insertion or deletion of edges (V is assumed to be fixed). The dynamic algorithms which handle only insertions are called

* Research supported by the Indo-German Max Planck Center for Computer Science (IMPECS).

incremental algorithms and those that can handle only deletions are called decremental algorithms. An algorithm that can handle both insertions and deletions of edges is called a *fully dynamic* algorithm. In this paper, we present a fully dynamic algorithm for maintaining an approximate maximum weight matching.

Previous Results

The fastest known algorithm for finding MCM in general graphs is by Micali and Vazirani[7] that runs in $O(m\sqrt{n})$ time. Their algorithm can be used to compute a matching having size $(1-\epsilon)$ times the size of maximum matching in $O(m/\epsilon)$ time. Mucha and Sankowski[8] designed an algorithm that computes MCM in $O(n^{\omega})$, where $\omega < 2.376$ is the exponent of n in the fastest known matrix multiplication algorithm. Relatively, fewer algorithms are known for maintaining matching in a dynamic graph. The first algorithm was designed by Ivkovic and Lloyd[5] with amortized update time $O(n+m)^{0.7072}$. Onak and Rubinfeld[9] presented an algorithm that achieves expected amortized polylogarithmic update time and maintains an α-approximate MCM where α was claimed to be some large constant but not explicitly calculated. Baswana, Gupta and Sen[1] presented a fully dynamic randomized algorithm for maintaining maximal matching in expected amortized $O(\log n)$ update time. It is well known that a maximal matching is a 2-MCM as well.

For computing maximum weight matching Gabow[4] designed an $O(mn + n^2 \log n)$ time algorithm. Preis[10] designed a $O(m)$ time algorithm for computing a 2-MWM. Drake and Hougardy[2] designed a simpler algorithm for the same problem. Vinkemeier and Hougardy[11] presented an algorithm to compute a matching which is $(2/3-\epsilon)$ times the size of MWM in $O(m/\epsilon)$ time. Duan, Pettie and Su[3] presented an algorithm to compute a matching which is $(1-\epsilon)$ times the size of MWM in $O(m\epsilon^{-1}\log\epsilon^{-1})$ time. To the best of our knowledge, there have been no sublinear algorithm for maintaining MWM or approximate MWM in dynamic graphs.

Preliminaries

Let M be a matching in a graph $G = (V, E)$. A vertex in the graph is called *free* with respect to M if it is not incident on any edge in M. A vertex which is not free is called *matched*. Similarly, an edge is called *matched* if it is in M and is called *free* otherwise. If (u, v) is a matched edge, then u is called be the *mate* of v and vice versa. A matching M is said to be *maximal* if no edge can be added to the matching without violating the degree bound of one for a matched vertex. An alternating path is defined as a path in which edges are alternately matched and free, while an augmenting path is an alternating path which begins and ends with free vertices.

Our Results

We present a fully dynamic algorithm that achieves expected amortized $O(\log n \log \mathcal{C})$ update time for maintaining 8-MWM. Here $\mathcal{C}$ is the ratio of the weights of the highest weight edge to the smallest weight edge in the given graph. Our algorithm uses, as a subroutine, the algorithm of Baswana, Gupta and Sen [1] for maintaining a maximal matching. We can state the main result of [1] formally in the following theorem.

▶ **Theorem 1.** *Starting from an empty graph on n vertices, a maximal matching in the graph can be maintained over any arbitrary sequence of t insertion and deletion of edges in $O(t \log n)$ time in expectation and $O(t \log n + n \log^2 n)$ time with high probability.*

Note that for the above algorithm, the matching(random bits) at any time is not known to the adversary[1] for it to choose the updates adaptively.

The idea underlying our algorithm has been inspired by the algorithm of Lotker, Patt-Shamir, and Rosen[6] for maintaining approximate MWM in distributed environment. Their algorithm maintains a 27-MWM in a distributed graph and achieves $O(1)$ rounds to update the matching upon any edge insertion or deletion.

Overview of our approach

Given that there exists a very efficient algorithm [1] for maintaining maximal matching (hence 2-MCM), it is natural to explore if this algorithm can be employed for maintaining approximate MWM. Observe that MCM is a special case of MWM with all edges having the same weight. Since a maximal matching is 2-MCM, it can be observed that a maximal matching is 2-MWM in a graph if all its edges have the same weight. But this observation does not immediately extend to the graphs having non-uniform weights on edges. Let us consider the case when the edge weights are within a range, say, $[\alpha^i, \alpha^{i+1})$, where $\alpha > 1$ is a constant. In such a graph the maximal matching gives a 2α approximation of the maximum weight matching. So, a maximal matching can be used as an approximation for MWM in a graphs where the ratio of weights of maximum weight edge to the smallest weight edge is bounded by some small constant. To exploit this observation, we partition the edges of the graph into levels according to their weight. We select a constant $\alpha > 1$ whose value will be fixed later on. Edges at level i have weights in the range $[\alpha^i, \alpha^{i+1})$ and the set of edges at level i is represented by E_i, viz., $\forall e \in E_i, w(e) \in [\alpha^i, \alpha^{i+1})$.

Observe that in this scheme of partitioning, any edge is present only at one level. The subgraph at level i is defined as $G_i = (V, E_i)$. We maintain a maximal matching M_i for G_i using the algorithm of Baswana, Gupta and Sen[1]. The maximal matching at each level provides an approximation for the maximum weight matching at that level. However, $\cup_i M_i$ is not necessary a matching since a vertex may have multiple edges incident on it from $\cup_i M_i$. Let $\mathcal{H} = (V, \bigcup M_i)$ be the subgraph of G having only those edges which are part of the maximal matching at some level. Our algorithm maintains a matching in the subgraph $\mathcal{H}$ which is guaranteed to be 8-MWM for the original graph G. The algorithm builds on the algorithm in [1], though the analysis of algorithm for maintaining 8-MWM is not straightforward.

2 Fully Dynamic 8-MWM

Our algorithm maintains a partition of edges according to their levels. A maximal matching M_i is maintained at each level using the fully dynamic algorithm in [1]. While processing any insertion or deletion of an edge, this algorithm will lead to change in the status of edges from being matched to free and vice-versa. This leads to deletion or insertion of edges from/to $\mathcal{H}$. However, since the algorithm [1] achieves expected amortized $O(\log n)$ time per update, so the expected amortized number of deletions and insertions of edges in $\mathcal{H}$ will also be $O(\log n)$ only. Our algorithm will maintain a matching $\mathcal{M}$ in the subgraph $\mathcal{H}$ taking advantage of the hierarchical structure of $\mathcal{H}$. Since $\mathcal{H}$ is formed by the union of matchings at various levels, a vertex can have at most one neighbor at each level. The matching $\mathcal{M}$ is maintained such that for every edge of $\mathcal{H}$ which is not in $\mathcal{M}$ there must be an edge adjacent

[1] The oblivious adversarial model is also used in randomized data-structure like universal hashing

to it at a higher level which is in $\mathcal{M}$. For an edge e, let $Level(e)$ denote its level. In precise words, the algorithm maintains the following invariant after every update.

$\forall e \in E(\mathcal{H})$, either $e \in \mathcal{M}$ or e is adjacent to an edge $e' \in \mathcal{M}$ such that $Level(e') > Level(e)$.

Notations

The algorithm maintains the following information at each stage.

- M_l - A maximal matching at the level l.
- $Free(v)$ - A variable which is true if v is free in the matching $\mathcal{M}$, and false otherwise.
- $Mate(v)$ - The mate of v, if it is not free.
- $Level((u,v))$ or $Level(e)$ - The level at which the edge e or the edge (u,v) is present according to the condition that $\forall e \in G_i, w(e) \in [\alpha^i, \alpha^{i+1})$.
- $OccupiedLevels$ - The set of levels where there is at least one edge from $\mathcal{H}$.
- L^{max} - The highest occupied level.
- L^{min} - The lowest occupied level.
- $N(v,i)$ - The neighbor of v in M_i, if any, and *null* otherwise.
- $\mathcal{M}$ - The matching maintained by our algorithm.

For a better understanding of our fully dynamic algorithm, the following section describes its static version for computing $\mathcal{M}$ in the graph $\mathcal{H}$.

Static Algorithm to obtain $\mathcal{M}$ from $\mathcal{H}$

Procedure 2.1: StaticCombine()

```
1 $\mathcal{M} = \phi$;
2 for $i = L^{max}$ to $L^{min}$ do
3     $\mathcal{M} = \mathcal{M} \cup M_i$;
4     for $(u,v) \in M_i$ do
5         for $j = i-1$ to $L^{min}$ do
6             for $(x,y) \in M_j$ do
7                 if $u = x$ or $u = y$ or $v = x$ or $v = y$ then
8                     $M_j = M_j \setminus \{(x,y)\}$;
```

The static algorithm divides the edges of the graph G into levels and a maximal matching M_i is obtained for each of the levels. Using these maximal matchings we get the graph $\mathcal{H}$. Thereafter the level numbers L^{max} and L^{min} are computed and the procedure `StaticCombine` is executed.

The procedure `StaticCombine` starts by picking all the edges in $\mathcal{H}$ at the highest level and adds them to the matching $\mathcal{M}$. For every edge (u,v) added to the matching $\mathcal{M}$, all the edges in the graph $\mathcal{H}$ incident on u and v have to be removed from the graph. The same process is repeated for the next lower level. Note that every edge in $\mathcal{H}$ is either in the matching $\mathcal{M}$ or its neighboring edge at some higher level is in $\mathcal{M}$ and thus the invariant is maintained. Observe that the matching $\mathcal{M}$ is a maximal matching in $\mathcal{H}$ because of the way it is being computed.

Dynamic Algorithm to maintain $\mathcal{M}$

After each insertion or deletion of any edge, our algorithm maintains a matching $\mathcal{M}$ satisfying the invariant described above. Our algorithm processes insertions and deletions of edges in $\mathcal{H}$ to update $\mathcal{M}$. An addition and deletion of the edges in $\mathcal{H}$ is caused due to addition/deletion of an edge in the original graph G. We describe some basic procedures first. Then the procedures for handling addition and deletion of edges in $\mathcal{H}$ are described and finally the procedures for handling addition and deletion of edges in G are described.

Procedure 2.2: AddToMatching(u, v)

```
Free(u) = False; Free(v) = False;
Mate(u) = v; Mate(v) = u;
$\mathcal{M} = \mathcal{M} \bigcup \{(u,v)\}$;
```

Procedure 2.3: DelFromMatching(u, v)

```
Free(u) = True; Free(v) = True;
$\mathcal{M} = \mathcal{M} \setminus \{(u,v)\}$;
```

The procedure `AddToMatching` adds an edge to the matching $\mathcal{M}$ updating the free and mate fields accordingly. The procedure `DelFromMatching` deletes an edge from the matching $\mathcal{M}$ updating the mate and the free fields correctly. Both of them execute in $O(1)$ time.

Procedure 2.4: HandleFree(u, lev)

```
for l from lev to $L^{min}$ do
    $v = N(u, l)$;
    if v is not null then
        if v is free then
            AddToMatching (u, v);
            return;
        else if Level((v, Mate(v))) < l then
            $v' = Mate(v)$;
            DelFromMatching (v, v');
            AddToMatching (u, v);
            HandleFree (v', Level((v, v')));
            return;
```

The procedure `HandleFree` takes as an input a vertex u which has become free in $\mathcal{M}$ and a level number lev from where it has to start looking for a mate. Note that it follows from the invariant that u does not have any free neighbor at any level above lev. The procedure `HandleFree` proceeds as follows. It searches for a neighbor of u in the decreasing order of levels starting from lev. In this process, on reaching a level $l \leq lev$ if it finds a free neighbor of u, the corresponding edge is added to the matching $\mathcal{M}$ and the procedure stops. Otherwise if some neighbor v is found which already has a mate at some lower level than l, then notice that we are violating the invariant as (u, v) does not belong to $\mathcal{M}$ and is neighboring to an edge in $\mathcal{M}$ at a lower level. So, the edge $(v, Mate(v))$ is removed from the matching $\mathcal{M}$, and the edge (u, v) is added to the matching $\mathcal{M}$. This change results in a free vertex which is at a lower level and so we proceed recursively to process it. Note that the recursive calls to `HandleFree` are all with lower level numbers. So, the procedure takes $O(L^{max} - L^{min})$ time.

Procedure 2.5: AddEdge(u, v)

```
1  $l = Level((u, v))$;
2  $N(u, l) = v$;
3  $N(v, l) = u$;
4  if $u$ is free and $v$ is free then
5      AddToMatching $(u, v)$;
6  else if $u$ is free and $v$ is not free then
7      if $Level((u, Mate(u))) < l$ then
8          $v' = Mate(v)$;
9          DelFromMatching $(v, v')$;
10         AddToMatching $(u, v)$;
11         HandleFree $(v', Level((v, v')))$;
12 else if $u$ is not free and $v$ is free then
13     if $Level((v, Mate(v))) < l$ then
14         $u' = Mate(u)$;
15         DelFromMatching $(u, u')$;
16         AddToMatching $(u, v)$;
17         HandleFree $(u', Level((u, u')))$;
18 else if $Level((v, Mate(v))) < l$ and $Level((u, Mate(u))) < l$ then
19     $u' = Mate(u)$; $v' = Mate(v)$;
20     DelFromMatching $(u, u')$; DelFromMatching $(v, v')$;
21     AddToMatching $(u, v)$;
22     HandleFree $(u', Level((u, u')))$;
23     HandleFree $(v', Level((v, v')))$;
```

The procedure `AddEdge` handles addition of edges to $\mathcal{H}$. Suppose the edge (u, v) is added to $\mathcal{H}$. If both u and v are free with respect to $\mathcal{M}$, then the edge (u, v) is added to the matching $\mathcal{M}$. Otherwise, there must be some edge(s) in $\mathcal{M}$ adjacent to (u, v). This follows due to the fact that $\mathcal{M}$ is a maximal matching in $\mathcal{H}$. If (u, v) is adjacent to a higher level edge in $\mathcal{M}$, then nothing is done. If (u, v) is adjacent to some lower level edge(s) in $\mathcal{M}$, then notice that the invariant maintained by the algorithm gets violated. Therefore, we remove these lower level edge(s) (adjacent to (u, v)) from the matching $\mathcal{M}$ and adds the edge (u, v) to the matching. At most 2 vertices can become free due to the addition of this edge to $\mathcal{M}$ and we handle them using the procedure `HandleFree`. If u' was the previous mate of u, then the edge (u, u') is removed from $\mathcal{M}$. Since $\mathcal{M}$ satisfied the invariant before addition of this edge, all the neighboring edges of u' at higher level than $Level(u, u')$ are matched to a vertex at higher levels. So u' has to start looking for mates from the level of (u, u'). The procedure makes a constant number of calls to `HandleFree` and thus runs in $O(L^{max} - L^{min})$ time.

The procedure `DeleteEdge` does nothing if an unmatched edge from $\mathcal{H}$ is deleted. If a matched edge (u, v) is deleted at level l, it calls `HandleFree` for both the end points to

Procedure 2.6: DeleteEdge(u, v)

```
1  $l = Level((u, v))$;
2  $N(u, l) = null$;
3  $N(v, l) = null$;
4  if $(u,\ v) \in \mathcal{M}$ then
5      DelFromMatching $(u, v)$;
6      HandleFree $(u, l)$;
7      HandleFree $(v, l)$;
```

restore the invariant in the matching. `HandleFree` is called with the level l because our invariant implies that all the neighbors of u and v are matched at higher levels. So they cannot find a mate at higher levels. This again takes $O(L^{max} - L^{min})$ time.

Procedure 2.7: EdgeUpdate$(u, v, type)$

```
1  $l = Level((u, v)) = \lfloor \log_\alpha w(u, v) \rfloor$;
2  if $type$ is addition and $M_l$ is $\phi$ then
3      $OccupiedLevels = OccupiedLevels \bigcup \{l\}$;
4      Update $L^{max}$ and $L^{min}$;
5  Update $M_l$ using the algorithm in [1];
6  if $type$ is deletion and $M_l$ is $\phi$ then
7      $OccupiedLevels = OccupiedLevels \setminus \{l\}$;
8      Update $L^{max}$ and $L^{min}$;
9  Let $\mathcal{D}$ be the set of edges deleted from $M_l$ in step 5;
10 Let $\mathcal{A}$ be the set of edges added to $M_l$ in step 5;
11 for $(x,\ y) \in \mathcal{D}$ do
12     DeleteEdge $(x, y)$;
13 for $(x,\ y) \in \mathcal{A}$ do
14     AddEdge $(x, y)$;
```

The function `EdgeUpdate` handles addition and deletion of an edge in G. It finds out the level of the edge and updates the maximal matching at that level using the algorithm of Baswana, Gupta and Sen [1]. It updates the *OccupiedLevels* set accordingly. This set is required because the values of L^{max} and L^{min} are to be maintained. The algorithm [1] can be easily augmented to return the set of edges being added or deleted from the maximal matching during each update in the graph G. As discussed before, expected amortized $O(\log n)$ edges change their status in $\mathcal{H}$ per update in G. Our algorithm processes these updates in $\mathcal{H}$ as described above. So, overall our algorithm has an expected amortized update time of $O(\log n \cdot (L^{max} - L^{min}))$. Let e^{max} and e^{min} represent the edges having the maximum and the minimum weight in the graph. Recall that $\mathcal{C} = w(e^{max})/w(e^{min})$.

$$L^{max} - L^{min} < \log_\alpha w(e^{max}) - \log_\alpha w(e^{min}) + 1 = O\left(\log \frac{w(e^{max})}{w(e^{min})}\right) = O(\log \mathcal{C})$$

So we can claim that

▶ Claim 2.1. The expected amortized update time of the algorithm per edge insertion or deletion is $O(\log n \log \mathcal{C})$.

In the next section we analyze the algorithm to prove that the matching $\mathcal{M}$ maintained by it at each stage is indeed 8-MWM.

2.1 Analysis

To get a good approximation ratio, we bound the weight of M^* with the weight of $\mathcal{M}$. We now state a few simple observations which help in understanding the analysis.

▶ Observation 2.1. Since M^* is a matching, no two edges of M^* can be incident on the same vertex.

▶ Observation 2.2. For any edge $e \notin M^*$, there can be at most two edges of M^* which are adjacent to e, one for each endpoint of e.

To bound the weight of M^* using the weight of $\mathcal{M}$, we define a many to one mapping $\phi : M^* \to \mathcal{M}$. This mapping maps every edge in M^* to an edge in $\mathcal{M}$. Using this mapping, we find out all the edges which are mapped to an edge $e \in \mathcal{M}$ and bound their weight using the weight of e. Let this set be denoted by $\phi^{-1}(e)$. For an edge $e^* \in M^*$, the mapping is defined as:

1. If $e^* \in E(\mathcal{H})$ and $e^* \in \mathcal{M}$ then $\phi(e^*) = e^*$.
2. If $e^* \in E(\mathcal{H})$ and $e^* \notin \mathcal{M}$ then our invariant ensures that e^* is adjacent to an edge $e \in \mathcal{M}$ such that $\text{Level}(e) > \text{Level}(e^*)$. In this case, we define $\phi(e^*) = e$. If e^* is adjacent to two matched edges in $\mathcal{M}$, map e^* to any one of them. As a rule, if two edges are available for mapping, then we will map e^* to any one of them.
3. If $e^* \notin E(\mathcal{H})$, then consider its level, say i. Since we maintain a maximal matching M_i at level i, at least one of the end point of e^* must be present in M_i. Let $e \in M_i$ be adjacent to e^*. If $e \in \mathcal{M}$, we define $\phi(e^*) = e$.
4. If $e^* \notin E(\mathcal{H})$ and the edge $e \in M_i$ adjacent to e^* is not present in $\mathcal{M}$ then e must be adjacent to an edge $e' \in \mathcal{M}$ such that $\text{Level}(e') > \text{Level}(e)$. In this case, we define $\phi(e^*) = e'$.

An edge $e^* \in M^*$ can either be present or absent from $E(\mathcal{H})$. If it is in $E(\mathcal{H})$, then it is mapped using type 1 and type 2 mapping else it is mapped using type 3 and type 4 mapping. This implies all the edges in M^* are mapped by ϕ. Now that we have defined a many to one mapping, we find out the edges of M^* which are mapped to an edge $e \in \mathcal{M}$. An edge which is mapped to an edge $e \in \mathcal{M}$ can either be e itself or be adjacent to e or not adjacent to e. If an edge of M^*, which is mapped to $e \in \mathcal{M}$, is e itself or is adjacent to e, then it is called a *Directly mapped edge*. An edge of M^* which is mapped to $e \in \mathcal{M}$ and is not adjacent to e is called an *Indirectly mapped edge*. Let $\phi_D^{-1}(e)$ and $\phi_I^{-1}(e)$ be the set of edges from M^* mapped directly and indirectly respectively to an edge $e \in \mathcal{M}$. Directly mapped edges are of type 1, 2 and 3 and indirectly mapped edges are of type 4.

If an edge $e \in \mathcal{M}$ has an edge of type 1 directly mapped to it, then e will not have any other edge directly mapped to it. This follows from the definition of a directly mapped edge and Observation 2.1. There can be at most two directly mapped edges of the type 2 (Observation 2.2). These edges mapped to e are always from a level $< \mathit{Level}(e)$. There can be at most two directly mapped edges of type 3 also if they are not in $\mathcal{H}$ but are adjacent to e. By Observation 2.2, there can only be two such edges.

▶ Claim 2.2. For an edge $e \in \mathcal{M}$, there can be at most two edges from M^* that are directly mapped to e.

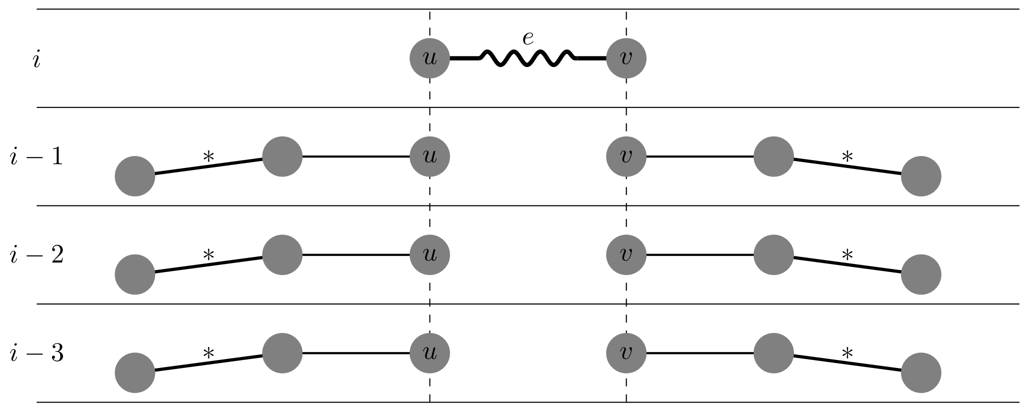

Figure 1 $e \in \mathcal{M}$. The edges marked $*$ are not in $\mathcal{H}$ and are in M^*. The edges which are not marked $*$ are all in $\mathcal{H}$. All the edges marked by $*$ are indirectly mapped to e.

The total weight of the edges directly mapped to e will be maximum when both of them are from the same level as e. Assume that e is at level i. Summing the weights of the edges which are directly mapped to e, we get

$$\sum_{e^* \in \phi_D^{-1}(e)} w(e^*) < 2 * \alpha^{i+1} < 2\alpha w(e) \tag{1}$$

An edge $e^* \in M^*$ mapped indirectly to an edge $e \in \mathcal{M}$ can only be of type 4. In this case e^* is not in $\mathcal{H}$, but is adjacent to an edge in $\mathcal{H}$, which in turn is adjacent to e. By definition, these edges are from a level lower than that of e. There can be at most two edges from each level lower than $Level(e)$ which are in $\mathcal{H}$ and are adjacent to e(see Figure 1).

▶ Claim 2.3. There can be at most two indirectly mapped edges to an edge $e \in \mathcal{M}$ at level $< Level(e)$.

Note that there can be a large number of edges which are indirectly mapped to e. Still we will be able to get a good bound on their total weight. This is because there can be at most two indirectly mapped edges from each level and the weight of edges in the levels decreases geometrically as we go to lower levels.

Assume that e is at level i. Summing the weight of edges which are indirectly mapped to e, we get

$$\sum_{e^* \in \phi_I^{-1}(e)} w(e^*) < 2 \sum_{j=i-1}^{L^{min}} \alpha^{j+1} < \frac{2\alpha^{i+1}}{\alpha - 1} < \frac{2\alpha w(e)}{\alpha - 1} \tag{2}$$

Thus, the total weight mapped to e is -

$$\sum_{e^* \in \phi^{-1}(e)} w(e^*) = \sum_{e^* \in \phi_D^{-1}(e)} w(e^*) + \sum_{e^* \in \phi_I^{-1}(e)} w(e^*) < w(e) \left(\frac{2\alpha}{\alpha - 1} + 2\alpha \right)$$

As reasoned before, an edge in M^* is mapped to some edge in $\mathcal{M}$. So summing this over all the edges in $\mathcal{M}$, we get

$$\sum_{e \in M} w(e) \left(\frac{2\alpha}{\alpha - 1} + 2\alpha \right) > \sum_{e \in M} \sum_{e^* \in \phi^{-1}(e)} w(e^*) = \sum_{e^* \in M^*} w(e^*)$$

The function $f(\alpha) = \left(\frac{2\alpha}{\alpha-1} + 2\alpha\right)$ attains its minimum value of 8 at $\alpha = 2$. So, if the value of α is picked to be 2, we get an 8 approximate maximum weight matching algorithm. We can state the following theorem.

▶ **Theorem 2.** *There exists a fully dynamic algorithm that maintains 8-MWM for any graph on n vertices in expected amortized $O(\log n \log \mathcal{C})$ time per update.*

3 Conclusion

We presented a fully dynamic algorithms for maintaining matching of large size or weight in graphs. The algorithm maintains a 8-MWM with expected $O(\log n \log \mathcal{C})$ amortized update time.

References

1 S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in O(log n) update time. In *52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 383–392. IEEE, 2011.

2 D.E. Drake and S. Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85(4):211–213, 2003.

3 R. Duan, S. Pettie, and H.H. Su. Scaling algorithms for approximate and exact maximum weight matching. *Arxiv preprint arXiv:1112.0790*, 2011.

4 H.N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 434–443. Society for Industrial and Applied Mathematics, 1990.

5 Z. Ivkovic and E. Lloyd. Fully dynamic maintenance of vertex cover. In *Graph-Theoretic Concepts in Computer Science*, pages 99–111. Springer, 1994.

6 Z. Lotker, B. Patt-Shamir, and A. Rosen. Distributed approximate matching. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 167–174. ACM, 2007.

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

8 M. Mucha and P. Sankowski. Maximum matchings via gaussian elimination. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 248–255. IEEE, 2004.

9 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*, pages 457–464. ACM, 2010.

10 R. Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *STACS 99*, pages 259–269. Springer, 1999.

11 D.E.D. Vinkemeier and S. Hougardy. A linear-time approximation algorithm for weighted matchings in graphs. *ACM Transactions on Algorithms (TALG)*, 1(1):107–122, 2005.