Fully dynamic maximal matching in $O(\log n)$ update time : with revised and corrected analysis

Surender Baswana
Department of CSE,
I.I.T. Kanpur, India
sbaswana@cse.iitk.ac.in

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

Sandeep Sen
Department of CSE,
I.I.T. Delhi, India
ssen@cse.iitd.ernet.in

October 11, 2016

Abstract

Baswana, Gupta and Sen [1] had presented a fully dynamic algorithm for maximal matching which achieved $O(\log n)$ expected amortized time per edge insertion or deletion. Moreover, for any sequence of t edge updates, the total time taken by the algorithm is $O(t \log n + n \log^2 n)$ with high probability. Unfortunately, the analysis given in [1] has a crucial flaw, viz., the statement of Lemma 4.10 [1] is not true in general. The readers familiar with [1] may refer to the box on page 5 to see the flaw. Although the bound on the update time was critically dependent on Lemma 4.10, we present an alternate proof of the claimed update time based on an interesting property of the original algorithm that was not reported earlier. The algorithms remain exactly the same in all their details as described in [1] except that the threshold for transferring a vertex to level i in our main algorithm is 4^i instead of 2^i.

We first present a result about an asymmetric random walk that we shall use in the analysis. We present the analysis for the 2-Level algorithm in Section 1. Thereafter, we present the analysis of our main algorithm in Section 2.

Asymmetric random walk on a line

Consider a particle performing a discrete random walk on a line starting from the origin. In each step, it moves one unit to the left with probability p or one step to the right with probability $q = 1 - p$. Each move is independent of the moves made in the past. The following lemma states a well known result about such a random walk.

Lemma 0.1 ([2]) *In a random walk starting at the origin with $p > q$, the probability of ever reaching the position $z > 0$ equals $(q/p)^z$.*

We shall use the following corollary of Lemma 0.1.

Corollary 0.1 *In a random walk starting at the origin with $p > q$, the probability of reaching the position $z > 0$ during any finite number of steps is less than $(q/p)^z$.*

1 Analysis of 2-Level Algorithm

Our analysis will crucially exploit the following lemma. The lemma holds since a vertex picks its matched edge uniformly from its owned edges while creating an epoch at level 1.

Lemma 1.1 *Suppose vertex v creates an epoch at level 1 during an update in the graph. and let O_v^{init} be the set of edges that v owned at the time of the creation of this epoch. Then, for any arbitrary sequence D of edge deletions of O_v^{init} and for any $(v, w) \in O_v^{init}$,*

$$\mathbf{Pr}[\text{MATE}(v) = w \mid D] \;=\; \frac{1}{|O_v^{init}|}$$

We first carry out the analysis for the high probability bound on the total update time taken by our algorithm. Thereafter we carry out the analysis for the expected value of the total update time.

Note that an edge e may be deleted and inserted multiple times during the algorithm. However, each of them is considered *distinct* from the perspective of the algorithm as well as its analysis.

1.1 High probability bound on the total update time

The key idea of randomization underlying our algorithm is that once a vertex v creates an epoch at level 1, there should be *many* edge deletions from $\mathcal{O}_v^{init}$ before the matched edge of v is deleted. In order to quantify this key idea, we introduce the following definition that categorizes an epoch as good or bad.

Definition 1.1 *An epoch is said to be bad if it gets terminated naturally with in the deletion of the first* $1/3$ *edges that it owned at the time of its creation. An epoch is said to be good if it is not bad.*

It follows from Definition 1.1 that a good epoch undergoes *many* edge deletions before getting terminated. So only the bad epochs are problematic. Now using Lemma 1.1, we establish an upper bound on the probability of an epoch to be bad.

Lemma 1.2 *Suppose vertex* v *creates an epoch at level* 1 *while processing some update in the graph. Then this epoch is going to be bad with probability* $1/3$ *for any sequence of future updates in the graph and the random bits picked during their processing.*

Proof: Let $\mathcal{O}_v^{init}$ be the set of edges owned by v at the time of the creation of the epoch. Recall that the mate picked by v, say w, is brought to level 1 by v when the epoch is created. As a result, none of the neighbors of w can pick it as a mate to terminate the epoch. Hence, the epoch will terminate only when the matched edge (v, w) is deleted. Consider any sequence of updates in the graph following the creation of this epoch. This sequence uniquely defines the sequence D of edge deletions of $\mathcal{O}_v^{init}$. It can be observed that the time when the epoch will be terminated is fully determined by this sequence D and the mate that v picks. Obviously the random bits picked during the processing of future updates has no influence on the duration of this epoch. The epoch will be bad if the mate of v is among the endpoints of the first $1/3$ edges in the sequence D. Now it follows from Lemma 1.1 that the mate of v is distributed uniformly among the endpoints of D. So the probability of the epoch to be bad is $1/3$ [1]. □

For the time complexity analysis, we will now show that the number of bad epochs may exceed the number of good epochs by at most $O(\log n)$ with very high probability.

Consider the sequence of updates in the graph in the chronological order. Each update may create zero or more epochs at level 1. The number of epochs created is thus a random variable. However, as shown by Lemma 1.2, each newly created epoch at level 1 will be bad with probability $\leq 1/3$ irrespective of the future sequence of updates in the graph and the random bits picked during their processing. The sequence of epochs terminated at level 1 can thus be seen as an instance of the asymmetric random walk as follows. Each step of the walk is one unit to the left of the current location with probability $2/3$ or one unit to the right with probability $1/3$ independent of the past moves. Therefore, the event that the number of bad epochs exceeds the number of good epochs by at least $2\log_2 n$ during the algorithm is stochastically identical to the event that the random walk starting from the origin reaches position $2\log_2 n$ within any finite number of steps. It follows from Corollary 0.1 that the probability of the latter event is less than $1/n^2$. So the following lemma holds.

Lemma 1.3 *During any sequence of* t *updates, the number of bad epoch at level 1 can exceed the number of good epochs by* $2\log_2 n$ *with probability at most* $1/n^2$.

Recall that each epoch at level 1 has a computation cost of $O(n)$ associated with it. Let t be the total number of updates in the graph. For each epoch at level 1, the number of owned edges at the time of its creation is at least $\sqrt{n}$. As a result the number of good epochs during t updates is bounded by $3t/\sqrt{n}$ deterministically. So the computation cost of good epochs at level 1 is bounded by $O(t\sqrt{n})$. Lemma 1.3 implies that the computation cost of all bad epochs at level 1 can exceed the computation cost of all good epochs at most by $cn\log n$ amount for some constant c with probability $\geq 1 - 1/n^2$. So overall the computation cost of all epochs at level 1 is bounded by $O((t\sqrt{n} + n\log n)$ with high probability. The computation cost of all epochs at level 0 is bounded deterministically by $O(t\sqrt{n})$. Hence the total computation time taken by our algorithm for any sequence of t updates is $O(t\sqrt{n} + n\log n)$ with high probability.

[1] The analysis assumed that each edge from $\mathcal{O}_v^{init}$ is going to be deleted sometime in future. If not, place all such edges of $\mathcal{O}_v^{init}$, that are not deleted, arbitrarily at the end of the sequence D. In this case, as the reader may also see, there are chances for the epoch to remain alive instead of getting terminated at the end of the algorithm. So the probability of the epoch to be bad in this case will be even less than $1/3$.

1.2 Expected value of the total update time

Let $X_{v,i,k}$ be a random variable which is 1 if v creates an epoch at level i during the processing of kth update, otherwise it is 0. We denote this epoch as EPOCH(v, i, k). Let $Z_{v,i,k}$ denote the number of edges from $\mathcal{O}_v^{init}$ that are deleted during the epoch. (If EPOCH(v, i, k) is not created, $Z_{v,i,k}$ is defined as 0). Since each edge deletion at level 1 is uniquely associated to the epoch that owned it, therefore, $\sum_{v,k} Z_{v,1,k} \leq t$. Hence,

$$\sum_{v,k} \mathbf{E}[Z_{v,1,k}] \leq t \tag{1}$$

We shall now derive a bound on the expected value of $Z_{v,1,k}$ in an alternate way.

Lemma 1.4 $\mathbf{E}[Z_{v,1,k}] \geq \sqrt{n}/2 \cdot \mathbf{Pr}[X_{v,1,k} = 1]$.

Proof: We shall first find the expectation of $Z_{v,1,k}$ conditioned on the event that v creates an epoch at level 1 during kth update. That is, we shall find $\mathbf{E}[Z_{v,1,k}|X_{v,1,k} = 1]$. Let $\mathcal{O}_v^{init}$ be the set of edges owned by v at the moment of creation of EPOCH$(v, 1, k)$, and let D be the deletion sequence associated with $\mathcal{O}_v^{init}$. It follows from Lemma 1.1 that the matched edge of v is distributed uniformly over D. So $\mathbf{E}[Z_{v,1,k}|X_{v,1,k} = 1] = |\mathcal{O}_v^{init}|/2 \geq \sqrt{n}/2$ since $|\mathcal{O}_v^{init}|$ for an epoch at level 1 is at least $\sqrt{n}$. Using conditional expectation, we get

$$\mathbf{E}[Z_{v,1,k}] = \mathbf{E}[Z_{v,1,k}|X_{v,1,k} = 1] \cdot \mathbf{Pr}[X_{v,1,k} = 1] \geq \sqrt{n}/2 \cdot \mathbf{Pr}[X_{v,1,k} = 1]$$

□

Notice that the computation cost of an epoch at level 1 is at most cn for some constant c. So the expected value of the computation cost associated with all natural epochs that get terminated at level 1 during t updates is

$$\begin{aligned}
\sum_{v,k} cn \cdot \mathbf{Pr}[X_{v,1,k} = 1] &= 2c\sqrt{n} \sum_{v,k} \sqrt{n}/2 \cdot \mathbf{Pr}[X_{v,1,k} = 1] \\
&\leq 2c\sqrt{n} \sum_{v,k} \mathbf{E}[Z_{v,1,k}] \qquad \text{using Lemma 1.4} \\
&\leq 2c\sqrt{nt} \qquad \text{using Equation 1}
\end{aligned}$$

We can thus conclude with the following theorem.

Theorem 1.1 *Starting with a graph on n vertices and no edges, we can maintain maximal matching for any sequence of t updates in $O(t\sqrt{n})$ time in expectation and $O(t\sqrt{n} + n \log n)$ with high probability.*

1.3 On improving the update time beyond $O(\sqrt{n})$

In order to extend our 2-LEVEL algorithm for getting a better update time, it is worth exploring the reason underlying $O(\sqrt{n})$ update time guaranteed by our 2-LEVEL algorithm. For this purpose, let us examine the second invariant more carefully. Let $\alpha(n)$ be the threshold for the maximum number of edges that a vertex at level 0 can own. Consider an epoch at level 1 associated with some edge, say (u, v). The computation associated with this epoch is of the order of the number of edges u and v own which can be $\Theta(n)$ in the worst case. However, the expected duration of the epoch is of the order of the minimum number of edges u can own at the time of its creation, i.e., $\Theta(\alpha(n))$. Therefore, the expected amortized computation per edge deletion for an epoch at level 1 is $O(n/\alpha(n))$. Balancing this with the $\alpha(n)$ update time at level 0, yields $\alpha(n) = \sqrt{n}$.

In order to improve the running time of our algorithm, we need to decrease the ratio between the maximum and the minimum number of edges a vertex can own during an epoch at any level. This insight motivates us for having a finer partition of vertices – the number of levels should be increased to $O(\log n)$ instead of just 2. When a vertex creates an epoch at level i, it will own at least 4^i edges, and during the epoch it will be allowed to own at most $4^{i+1} - 1$ edges. As soon as it owns 4^{i+1} edges, it should migrate to the level $i + 1$. Notice that the ratio of maximum to minimum edges owned by a vertex during an epoch gets reduced from $\sqrt{n}$ to a constant.

We pursue the approach sketched above combined with some additional techniques in the following section. This leads to a fully dynamic algorithm for maximal matching which achieves expected amortized $O(\log n)$ update time per edge insertion or deletion.

2 Analysis of the main algorithm

In the main algorithm of [1], we showed that the total computation cost associated with an epoch at any level i is $O(2^i)$ (see Lemma 4.7). The only change in the main algorithm now is that the threshold for transferring a vertex to level i in our main algorithm is 4^i instead of 2^i. So the following lemma holds directly for the revised algorithm.

Lemma 2.1 *For any $i \geq 0$, the computation cost associated with an epoch at level i is $O(4^i)$.*

An epoch corresponding to edge (u, v) at level i could be terminated if the matched edge (u, v) gets deleted. Such an epoch is called a natural epoch. However, this epoch could be terminated due to one of the following reasons also.

- u (or v) get selected as a random mate by one of their neighbors present at LEVEL $> i$.
- u (or v) starts owning 4^{i+1} or more edges.

Each of the above factors render the epoch to be an induced epoch. For any level $i > 0$, the creation of an epoch causes termination of at most two epochs at levels $< i$. It can be explained as follows: Consider an epoch at level i associated with an edge, say (u, v). Suppose it was created by vertex u. If u was already matched at some level $j < i$, let $w \neq v$ be its mate. Similarly, if v was also matched already at some level $k < i$, let $x \neq u$ be its mate. So matching u to v terminates the epoch of (u, w) and (v, x) at level j and k respectively. We can thus state the following lemma.

Lemma 2.2 *Creation of an epoch at a level i may cause termination of at most 2 epochs at level $< i$.*

2.1 Analysing an epoch

Consider an epoch created by a vertex v at level i. At the time of the creation of the epoch, let $\mathcal{O}_v^{init}$ be the set of edges owned by v, and let $w = \text{MATE}(v)$. The mate w is brought to level i by v. Therefore, none of the neighbors of w from level $\leq i$ can pick w as a mate so long as the neighbor stays at level $\leq i$. The epoch may terminate when the matched edge (v, w) is deleted. However, it may terminate even before as well as follows. The epoch may terminate when v or w moves to some level $> i$ before the deletion of (v, w). This happens when either of them gets picked as a mate by some vertex at level $> i$. The epoch may also terminate if either of them start owning $\geq 4^{i+1}$ edges. Therefore, in order to analyse the termination of an epoch, we associate an update sequence with it as follows. For each edge $(v, x) \in \mathcal{O}_v^{init}$, we consider the first time in future that x moves to some higher level [2]. The *update label* associated with edge (v, x) is defined as

if x moves to a level $> i$ before its deletion then it is classified as *upward* else it is *deletion*.

Likewise, we also consider the first time in future that v moves to a level $> i$. If v never moves to any level $> i$ in future, we just append v at the end of all the updates associated with $\mathcal{O}_v^{init}$. The update sequence U for the epoch is the sequence of these updates on the edges of $\mathcal{O}_v^{init}$ and vertex v arranged in the chronological order. Consider the following example. Suppose $\mathcal{O}_v^{init}$ has 10 edges and let the corresponding neighbors of v be $\{w_1, \ldots, w_{10}\}$. Let the updates in the chronological order be : the deletion of (v, w_4), upward movement of w_1, upward movement of w_9, the deletion of (v, w_5), and so on. The corresponding update sequence will be

$$U : \langle \overset{\bullet}{w_4},\ \overset{\uparrow}{w_1},\ \overset{\uparrow}{w_9},\ \overset{\bullet}{w_5},\ \overset{\bullet}{w_8},\ \overset{\bullet}{w_3},\ \overset{\uparrow}{w_2},\ \overset{\uparrow}{v},\ \overset{\bullet}{w_7},\ \overset{\uparrow}{w_{10}},\ \overset{\bullet}{w_8} \rangle$$

Observation 2.1 *If the update associated with (the owner) v appears at ℓth location in U, then the epoch will terminate on or before the ℓth update in U. Therefore, the updates at location $> \ell$ in U will have no influence on the termination of the epoch.*

[2] vertex x may move to level $> i$ and down multiple times while the algorithm processes a sequence of updates. However, it is only the first time (after the creation of the epoch) when x moves to a level $> i$ that is relevant as far as the possibility of the termination of the epoch by the upward movement of x is concerned.

The error in the analysis of main algorithm in [1]

In order to establish the high probability bound on the update time of our algorithm, we took the following approach. We introduced the following terminology. An edge e is said to be deleted at level i if at the time of the deletion, one of the endpoint of e is present at level i. So an edge deletion is associated with at most 2 levels. We made an attempt to bound the number of natural epochs terminated at a level in terms of the number of edge deletions occuring at that level. We made the following claim (stated as Lemma 4.10 in [1]). If t_i is the number of edge deletions occuring at level i, then the number of natural epochs terminated at a level i is $O(t_i/2^i + \log n)$ with high probability. However, the following example shows that the above property does not hold for some situations.
Consider an epoch created by a vertex u. Let the set of edges $\mathcal{O}_u^{init}$ owned by u at the time of creation of the epoch be incident from vertices $v_1, \ldots, v_q$. Without loss of generality, suppose the sequence of edge deletions of the owned edge be $\langle (u, v_1), \ldots, (u, v_q) \rangle$. Duration of the epoch is defined as the number of deletions of the edges from $\mathcal{O}_u^{init}$ before the epoch gets terminated. Since u picks its mate randomly, if all the updates associated with the epoch are deletions only, the duration of the epoch will be a random variable distributed uniformly in the range $[1, q]$. This property was used rightly to establish that the number of epochs terminated at level 1 in our 2-level algorithm is bounded by $O(t/\sqrt{n} + \log n)$ with high probability. However, this property fails in the case of our main algorithm where the updates associated with the edges of $\mathcal{O}_u^{init}$ can be upward movement as well. In order to realize it, suppose the update sequence associated with the epoch be

$$U \ : \ \langle \ \overset{\uparrow}{v_2}, \ \overset{\uparrow}{v_3}, \ \cdots, \overset{\uparrow}{v_{q-1}}, \ \overset{\uparrow}{v_q}, \ \overset{\bullet}{v_1}, \ \rangle$$

Basically before the first edge, that is (u, v_1), is deleted, all other edges from the set $\mathcal{O}_u^{init}$ move upward. If the mate picked by u is v_1, then the epoch will terminate naturally, and only one edge deletion, that is (u, v_1), will be associated with the termination of this epoch. If the mate of u is any other vertex, the epoch will be induced epoch only and no edge deletion will be associated with the epoch. Now suppose a series of epochs appear at level i such that the update sequence for each of them looks identical to the update sequence shown above. It follow that if t_i edge deletions take place at level i, the number of natural epochs terminated will be t_i and not $O(t_i/2^i)$. Hence Lemma 4.10 in [1] is incorrect. However, we present now an alternate way to establish that the total update time for maintaining a maximal matching by our algorithm for any sequence of t updates is $O(t + n \log n)$ with high probability. The new analysis is based on an insightful property of the algorithm and makes use of a more careful amortization arguments spanning multiple levels.

Unlike the 2-level algorithm, the update sequence associated with an epoch is not uniquely defined by the sequence of updates in the graph after the creation of the epoch. Rather, it also depends upon the current matching as well as the random bits chosen by the algorithm while processing the future updates. So there is a probability distribution defined over all possible update sequences that depends upon these two factors. Consequently, the analysis of an epoch in our final algorithm is more complex compared to the 2-level algorithm. In particular, it is not obvious whether there is any dependence between the random mate picked by a vertex while creating an epoch and the sequence of updates associated with the epoch. However, using an interesting and non-trivial property of our algorithm, we will establish that there is no dependence between the two.

Lemma 2.3 *Suppose vertex v creates an epoch at some level i while the algorithm processes kth update in the graph. Let $\mathcal{O}_v^{init}$ be the set of its owned edges at the time of the creation of this epoch. Then, for any update sequence U and for each $(v, w) \in \mathcal{O}_v^{init}$,*

$$\mathbf{Pr}[\text{MATE}(v) = w \mid U] \ = \ \mathbf{Pr}[\text{MATE}(v) = w] \ = \ \frac{1}{|\mathcal{O}_v^{init}|}$$

Remark 2.1 *The sample space considered in Lemma 2.3 is defined by the random bits picked for choosing the mate of v and all the random bits picked thereafter by the algorithm for processing the current as well as all the future updates.*

Lemma 2.3 can be seen as a generalization of Lemma 1.1 that we stated for our 2-level algorithm. Its proof is given in Section 2.4 where we actually establish that the probability distribution of all matchings at levels $> i$ during the future updates is the same irrespective of which mate v picks while creating an epoch at level i. This insight has

the following implication. Let U be any possible update sequence for an epoch of v. Upon picking a mate, say w, if U occurs with probability p, then upon picking any other mate, say w', as well, the update sequence U is going to occur with the same probability p. In other words, the update sequence associated with an epoch of v is independent of the mate picked by v while creating the epoch. It is easy to notice that Lemma 2.3 is nothing but a reformulation of this fact.

The analysis of our algorithm will be critically dependent on Lemma 2.3. Using this lemma, we shall first establish a high probability bound on the total update time of the algorithm to process a sequence of updates in the graph.

2.2 High probability bound on the total update time

Recall Definition 1.1 of a bad epoch. It can be observed from this definition that an induced epoch is always a good epoch. Using Lemma 2.3, the lemma for the probability of a bad epoch extends seamlessly from 2-level algorithm to our final algorithm as follows.

Lemma 2.4 *Suppose vertex v creates an epoch at level i while the algorithm processes kth update in the graph. This epoch will be bad with probability at most $1/3$ for any sequence of future updates in the graph and the random bits picked during their processing.*

Proof: Let $\mathcal{O}_v^{init}$ be the set of edges owned by v at the time of the creation of the epoch during the kth update. Let $(\mathcal{U}, P)$ be the probability space of all update sequences associated with the epoch for any sequence of future updates in the graph. It can be observed that the time when the epoch will be terminated is completely determined by the mate that v picks and the update sequence associated with the epoch. This update sequence, in turn, is determined by the random bits picked subsequent to the creation of the epoch and the future updates. However, as stated in Lemma 2.3, conditioned on each update sequence $U \in \mathcal{U}$, the mate of v will be uniformly distributed among the endpoints of $\mathcal{O}_v^{init}$. We shall use this fact to bound the probability of the epoch to be bad. First recall from Observation 2.1 that for the epoch to be terminated naturally, the mate of v must be among the endpoints of the deleted edges that precede v in U. So we distinguish between the following two cases.

Case1. There are less than $|\mathcal{O}_v^{init}|/3$ edge deletions preceding v in U.
The epoch will be bad only if the matched edge of v is one of these edge deletions preceding v in U. Since the number of these edge deletions is less than $|\mathcal{O}_v^{init}|/3$, so using Lemma 2.3 the probability of the epoch to be bad in this case is less than $1/3$.

Case2. There are at least $|\mathcal{O}_v^{init}|/3$ edge deletions preceding v in U.
The epoch will be bad only if the matched edge of v is one of the first $|\mathcal{O}_v^{init}|/3$ edge deletions in U. So it follows from Lemma 2.3 that the probability of the epoch to be bad in this case is exactly $1/3$.

It follows that in both the cases, the epoch is going to be bad with probability $\leq 1/3$. Hence the epoch is going to be bad with probability $\leq 1/3$ for any update sequence U associated with the epoch, that is, for any sequence of future updates in the graph and the random bits picked during their processing. □

We will now show that the number of bad epochs at a level i could exceed the number of good epochs at level i by at most $O(\log n)$ with very high probability.

Consider the sequence of updates in the graph in the chronological order. Each update may create zero or more epochs at level i. The number of epochs created is thus a random variable. However, as shown by Lemma 1.2, each epoch created at level i will be bad with probability $\leq 1/3$ irrespective of the future updates and the random bits picked by the algorithm to process them. Hence, the sequence of good and bad epochs at level i can be seen as an instance of the asymmetric random walk as established in the analysis of the 2-level algorithm. So the bad epochs at any level i may exceed the good epochs by $2\log_2 n$ with probability at most $1/n^2$. There are $O(\log n)$ levels in the hierarchy. Hence we get the following lemma using *union bound*.

Lemma 2.5 *For every level $i \leq \mathrm{L}_0$, the number of bad epochs will not exceed the number of good epochs by more than $2\log_2 n$ with probability at least $1 - (\log n)/n^2 > 1 - 1/n$.*

Let us temporarily exclude the maximum surplus of $O(\log n)$ bad epochs at each level from our analysis. Consequently, it follows from Lemma 2.5 that each bad epoch at a level can be mapped to a good epoch at the same level in a unique manner - see Figure 1(i). Also the creation of each epoch at a level $i+1$ can terminate at most two (induced) epochs at lower levels as stated in Lemma 2.2. Using this fact and the mapping between the good and bad epochs at a

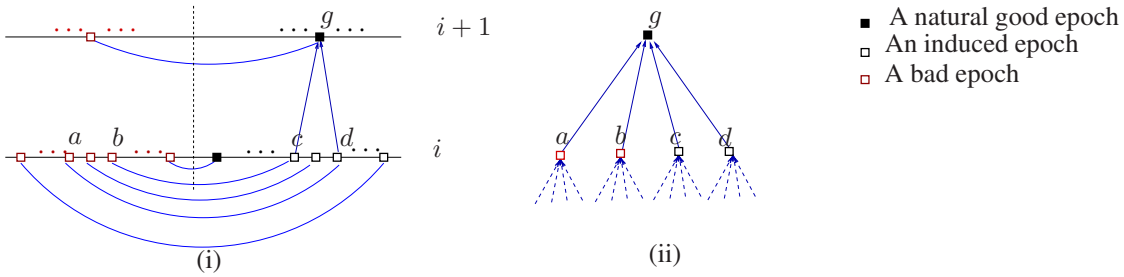

Figure 1: (i) Mapping between bad and good epochs at level i (ii) Assigning at most 4 epochs from lower levels to an epoch.

level, we can construct a forest whose nodes will be the epochs terminated across all levels during the algorithm. The intuition for defining this forest is that eventually the computation cost of a bad epoch or an induced epoch will be charged to a good natural epoch. Since a good natural epoch has sufficiently *large* number of edge deletions associated with it, these edge deletions can be charged to pay for all the computation carried out by our algorithm.
With this intuition, we now provide the construction of the forest by defining parent of each epoch using the following rules.

1. Parent of each induced epoch is the epoch at the higher level whose creation led to its termination.
2. Parent of a good epoch is itself (hence it is the root of its tree).
3. If a bad epoch is mapped to an induced epoch, then its parent is the same as the parent of the induced epoch. Otherwise, it is the parent of itself (hence it is the root of its tree).

It follows from rule 1 and 3 (the if part) that with an epoch at a level, at most 4 epochs from lower levels can be associated. Hence each node in the forest will have at most four children. See Figure 1(ii). Moreover, the root of each tree in the forest of epochs is either a bad epoch or a good natural epoch. Using Lemma 2.1, the computation cost $C(i)$ associated with a tree of epochs whose root is at level i obeys the following recurrence for some constant a.

$$C(i) \leq a4^i + 4C(i-1)$$

The solution of this recurrence is $C(i) = O(i4^i)$. It follows from Lemma 2.5 and rule 3(Otherwise part) that the number of trees rooted at good natural epochs at a level i is at least the number of trees rooted at bad epochs at level i. Hence, it suffices to analyze the computation cost associated with all the trees rooted at good natural epochs. Now for each good natural epoch at a level i, there are at least $4^i/3$ edge deletions associated uniquely to it. This natural epoch will be charged for the computation cost $C(i) = O(i4^i)$ associated with the tree rooted at it. So if t is the total number of updates in the graph, then the computation cost associated with all epochs in the forest is $O(t \log n)$. The computation cost associated with the surplus bad epochs at all levels is $O(\sum_i i4^i \log n) = O(n \log^2 n)$. Hence with high probability the computation cost for processing t edge updates by the algorithm is $O(t \log n + n \log^2 n)$. This also implies that the total expected update time is $O(t \log n)$ for $t = \Omega(n \log n)$. In the following subsection, we will establish $O(t \log n)$ bound on the expected update time for all values of t.

2.3 Expected value of the total update time

During a sequence of t updates in the graph, various epochs get created by various vertices at various levels. Let $X_{v,i,k}$ be a random variable which is 1 if v creates an epoch at level i during kth update, otherwise it is 0. We denote this epoch as EPOCH(v, i, k). Let $\mathcal{O}_v^{init}$ denote the set of edges that v owns at the time of the creation of the epoch. We now introduce a random variable $Z_{v,i,k}$. The creation of EPOCH(v, i, k) will be followed by the updates (deletion or upward movements) on the edges of $\mathcal{O}_v^{init}$. $Z_{v,i,k}$ is the number of updates of type 'edge deletions' in the corresponding update sequence associated with this epoch that occur before the epoch gets terminated. If EPOCH(v, i, k) is not created, $Z_{v,i,k}$ is defined as 0. The key role in bounding the expected running time is played by a random variable $B_{v,i,k}$ defined as follows:

$$B_{v,i,k} = \begin{cases} (8Z_{v,i,k} - 2 \cdot 4^i)X_{v,i,k} & \text{if EPOCH}(v,i,k) \text{ is natural} \\ (4^{i+1} - 2 \cdot 4^i)X_{v,i,k} & \text{if EPOCH}(v,i,k) \text{ is induced} \end{cases} \quad (2)$$

First observe that $B_{v,i,k} = 0$ if $X_{v,i,k} = 0$, else the random variable $B_{v,i,k}$ can be seen as the *credits* associated with EPOCH(v, i, k) to be used for paying its computation cost. For a natural epoch, the credits is defined in terms of the updates of type 'edge deletion' in the corresponding update sequence that occur before the epoch gets terminated. So we define $B_{v,i,k}$ to be $8Z_{v,i,k}$. However, we need to discount for the two epochs at lower levels that may be destroyed due to EPOCH(v, i, k). To this end, from the term, we deduct $2 \cdot 4^i$. Similarly, if EPOCH(v, i, k) is an induced epoch, then it gets 4^{i+1} credits from the epoch that destroyed it. But here again we need to discount for the two epochs at lower levels that might be destroyed by it. To this end, we again deduct $2 \cdot 4^i$ from 4^{i+1}. The following lemma gives a bound on $\sum_{v,i,k} B_{v,i,k}$.

Lemma 2.6 $\sum_{v,i,k} B_{v,i,k} \leq 8 \sum_{v,i,k} Z_{v,i,k} \leq 8t$*, where* t *is the total number of updates in the graph.*

Proof: We need to analyze the sum of $B_{v,i,k}$'s for all those (i, v, k) values for which the EPOCH(i, v, k) got created. If this epoch is an induced epoch, it can be associated with an epoch, say EPOCH(v', i', k'), at a higher level $i' > i$ whose creation destroyed it. Notice that the negative $4^{i'}$ term in $B_{v',i',k'}$ cancels out the positive 4^{i+1} term in $B_{v,i,t}$. Hence, the contribution of induced epochs in $\sum_{v,i,k} B_{v,i,k}$ is nullified and all that remains is the sum of terms $8Z_{v,i,k}$ for each natural epoch. Hence $\sum_{v,i,k} B_{v,i,k} \leq 8 \sum_{v,i,k} Z_{v,i,k}$. Notice that the deletion of an edge can appear in the update sequence of at most one of its endpoints, so an edge deletion will contribute to at most one random variable $Z_{v,i,k}$. Therefore, $\sum_{v,i,k} Z_{v,i,k}$ is upper bounded by the total number of edge updates in the graph. □

Corollary 2.1 $\sum_{v,i,k} \mathbf{E}[B_{v,i,k}] \leq 8t$

Lemma 2.7 *For each possible* i, v, k, $\mathbf{E}[B_{v,i,k}] \geq \mathbf{Pr}[X_{v,i,k} = 1] \cdot 4^i$.

Proof: Since $X_{v,i,k}$ is an indicator random variable, $\mathbf{E}[B_{v,i,k}] = \mathbf{Pr}[X_{v,i,k} = 1] \cdot \mathbf{E}[B_{v,i,k} | X_{v,i,k} = 1]$. So we will estimate $\mathbf{E}[B_{v,i,k} | X_{v,i,k} = 1]$, that is, the expected value of $B_{v,i,k}$ given that EPOCH(v, i, k) got created.

Let $(\mathcal{U}, P)$ be the probability space of all the update sequences associated with this epoch and let $U \in \mathcal{U}$ be any update sequence. Suppose among the updates in U that precede the update associated with v, only d are edge deletions. It follows from Lemma 2.3 that the matched edge of v is distributed uniformly over $\mathcal{O}_v^{init}$. So EPOCH(v, i, k) will be an induced epoch with probability $(|\mathcal{O}_v^{init}| - d)/|\mathcal{O}_v^{init}|$, and in that case $B(v, i, k)$ will be $4^{i+1} - 2 \cdot 4^i$. If the epoch is natural, it could be due to any one of the d edge deletions present in U. In that case the expected value of $B_{v,i,k}$ will be $1/d \sum_{j=1}^{d} (8j - 2 \cdot 4^i) \geq 4d - 2 \cdot 4^i$. Considering the cases of induced and natural epoch together,

$$\begin{aligned} \mathbf{E}[B_{v,i,k} \mid U] = \frac{|\mathcal{O}_v^{init}| - d}{|\mathcal{O}_v^{init}|}(4^{i+1} - 2 \cdot 4^i) \; + \; \frac{d}{|\mathcal{O}_v^{init}|}(4d - 2 \cdot 4^i) \quad &= \quad 2 \cdot 4^i - \frac{4^{i+1}d - 4d^2}{|\mathcal{O}_v^{init}|} \\ &\geq \quad 2 \cdot 4^i - \frac{4^i \cdot 4^i}{|\mathcal{O}_v^{init}|} \text{ (for all values of } d) \end{aligned}$$

Therefore,

$$\mathbf{E}[B_{v,i,k} \mid X_{v,i,k} = 1] \; = \; \sum_{U \in \mathcal{U}} \mathbf{E}[B_{v,i,k} \mid U] \; \cdot \; \mathbf{Pr}[U] \; \geq \; \left(2 \cdot 4^i - \frac{4^i \cdot 4^i}{|\mathcal{O}_v^{init}|}\right) \cdot \sum_{U \in \mathcal{U}} \mathbf{Pr}[U] \; = \; 2 \cdot 4^i - \frac{4^i \cdot 4^i}{|\mathcal{O}_v^{init}|}$$

Since $|\mathcal{O}_v^{init}| \geq 4^i$ for level i, the result follows. □

Let $W_{v,i,k}$ be a random variable with value equal to the computation cost of EPOCH(v, i, k) if the epoch is created, and 0 otherwise. Notice that the computation cost of an epoch at level i is $c4^{i+1}$ for some constant c. So, $\mathbf{E}[W_{v,i,k}] = \mathbf{Pr}[X_{v,i,k} = 1]c4^{i+1}$. Therefore, using Lemma 2.7,

$$\mathbf{E}[W_{v,i,k}] \leq 4c\mathbf{E}[B_{v,i,k}] \tag{3}$$

Using the above equation and Corollary 2.1, the total expected computation cost associated with all epochs that get destroyed during the algorithm can be bounded by $O(t)$ as follows.

$$\sum_{v,i,k} \mathbf{E}[W_{v,i,k}] \leq \sum_{v,i,k} 4c\mathbf{E}[B_{v,i,k}] \leq 32ct = O(t)$$

Since for each update in the graph, we incur $O(\log n)$ time to update ϕ at various levels, there is an $O(t \log n)$ overhead for t updates. We can thus conclude with the following theorem.

Theorem 2.1 *Starting with a graph on n vertices and no edges, we can maintain a maximal matching for any sequence of t updates in $O(t \log n)$ time in expectation and $O(t \log n + n \log^2 n)$ with high probability.*

2.4 Proof of Lemma 2.3

Our algorithm uses randomization to maintain a maximal matching. After any given sequence of updates, there is a set of possible maximal matchings that the algorithm may be maintaining and there is a probability distribution associated with these maximal matchings. So it is useful to think about the probability space of these matchings as the algorithm proceeds while processing a sequence of updates.

We introduce some notations first. Our algorithm ensures that both the endpoints of each matched edge in the matching $\mathcal{M}$ are present at the same level. So the matching maintained by our algorithm can be described as a set of tuples as follows.

$$\mathcal{M} = \{(u, v, \ell) \mid u \text{ is matched with } v \text{ at level } \ell\}$$

For any matching $\mathcal{M}$ maintained at any stage by our algorithm, let $\mathcal{M}_i$ denote the matching at level i. Let $\mathcal{M}_{>i} = \cup_{j>i}\mathcal{M}_j$ denote the matchings at all levels $> i$. Let V_i denote the set of all the vertices belonging to levels in the range $\in [-1, i]$. We now extend the notations to incorporate the updates in the graph. For any $k \geq 1$, let $G(k)$ denote the graph after a given sequence of k updates and let $\mathcal{M}(k)$ denote the maximal matching of $G(k)$ as maintained by our algorithm. Let $\mathcal{M}_{>i}(k)$ denote the matchings at all levels $> i$ after a given sequence of k updates.

After processing certain number of updates by the algorithm, suppose M and M' are any two possible matchings such that $M_{>i} = M'_{>i}$. Consider any single update in the graph at this stage. In order to process it, suppose we carry out two executions I and I' of our algorithm with the initial matching being M and M' respectively. That is, $\mathcal{M}(0) = M$ in the execution I and $\mathcal{M}(0) = M'$ in the execution I'. Our claim is that the probability distribution of matchings at levels $> i$ will be identical at the end of both the executions. More precisely, for any maximal matching $\mu(1)$ on a subset of vertices in graph $G(1)$,

$$\mathbf{Pr}[\mathcal{M}_{>i}(1) = \mu(1) | \mathcal{M}(0) = M] \;=\; \mathbf{Pr}[\mathcal{M}_{>i}(1) = \mu(1) | \mathcal{M}(0) = M'] \tag{4}$$

In order to establish our claim, we shall crucially exploit the following lemma.

Lemma 2.8 *For both the matchings M and M', $\phi_v(j)$ is the same for each $v \in V$ and each $j > i$.*

Proof: It is given that $M_{>i} = M'_{>i}$. This implies that for each level $j > i$ the sets of vertices present are identical in M and M'. Hence the set V_i of all the vertices present at levels $\in [-1, i]$ is identical in M and M'. Hence for any vertex v, and any level $j > i$, the set of all the neighbours of v at levels $< j$ is identical; notice that $\phi_v(j)$ is just the cardinality of this set. So it follows that $\phi_v(j)$ is the same for each vertex v and each $j > i$. □

We shall now establish the validity of Equation 4 for the deletion of an edge $e = (u, v)$. Establishing its validity for the insertion of an edge is similar and is sketched in Appendix. Notice that our algorithm does not alter the matching if e is not a matched edge. If e is a matched edge, a wave of free vertices originates from LEVEL(e) and propagates downward. The following facts hold for our main algorithm.

$F1$. The algorithm won't alter the matching at level $>$ LEVEL(e) while processing the deletion of e.

$F2$. The matching is updated in the decreasing order of levels, and once the updating of the matching at a level is complete, the matching at that level will remain unchanged during the updates of the matching at lower levels.

It follows from the description of M and M' that either LEVEL(e) is less than or equal to i in both the matchings or LEVEL(e) is the same in M and M'. Let us first consider the (easier) case when LEVEL(e) $\leq i$ in M as well as M'. It follows from Fact $F1$ stated above that the only changes in matchings M and M' will be at levels $\leq i$. Hence the matching $\mathcal{M}_{>i}(1)$ will be identical at the end of both the executions I and I'. Let us now consider the more interesting case of LEVEL(e) $> i$. Let LEVEL(e) $= \ell$. Both the executions I and I' invoke the procedure PROCESS-FREE-VERTICES($\langle(u, \ell), (v, \ell)\rangle$) in this case. For a better understanding, the reader is recommended to revisit this procedure from Section 4.2.1 of [1] before proceeding further.

In order to establish our claim about I and I', we shall establish the following assertion. While the matching at levels $> i$ is being updated, for each step executed in I, the same step will be executed in I'. Moreover, if the step in I is executed with some probability, the step will be executed with the same probability in I' as well. In order to show this, let us analyse the first iteration of the procedure PROCESS-FREE-VERTICES. Both I and I' will process u

first. After disowning its edges from its present level, u owns the same set of edges in both the executions. Thereafter, u will either stay at level ℓ or fall by one level. If u stays at level ℓ, it chooses a random edge to get matched. The probability that any specific random edge is picked by u is the same in both the executions. Let us consider the case that u falls by one level. It follows from Lemma 2.8 that $\phi_x(j)$ is the same in I as well as I' for each vertex x and each level $j > i$ just before the fall of u. When u falls by one level, for each neighbor z of u at any level $< \ell$, $\phi_z(\ell)$ increases exactly by one and remains unchanged for all other levels (refer to Lemma 4.2 from [1]). Hence the set of neighbors of u rising to level ℓ are the same in both the executions I and I'. In addition, the set of edges that each such vertex owns on rising to level ℓ is also the same, hence, the probability that any specific random mate is picked is the same in both the executions. So each update in M and M' is equally likely during the processing of u. The reader may note that after each such identical update in M and M', the matchings are identical at each level $> i$. Hence, Lemma 2.8 holds again for the updated matchings. Unlike the first iteration, a generic iteration of the procedure PROCESS-FREE-VERTICES may have free vertices at levels $\leq$ LEVEL(e) that are kept in respective queues at these levels. Suppose in the beginning of any such iteration of the procedure PROCESS-FREE-VERTICES there are two possible configurations such that the matching as well as the queue storing the free vertices are identical at each level $> i$ but differ at levels $\leq i$. Notice that Lemma 2.8 will hold for these configurations as well. Therefore, along exactly the same lines as the first iteration analysed above, it can be shown that every update in the matching at level $> i$ will be carried out with the same probability during any generic iteration for any two configurations that match at all levels $> i$.

Therefore, each sequence of updates in the matching is equally likely in both the executions I and I' till the last free vertex at level $i + 1$ is processed. Henceforth, the two executions may differ. But as follows from Fact $F2$, it will affect only the matching at levels $\leq i$ and there won't be any change in the matching at higher levels. This establishes our claim, i.e. Equation 4, for any single update in the graph. This claim can be invoked appropriately for a sequence of updates giving us the following theorem.

Theorem 2.2 *Let M and M' be any two possible matchings by our algorithm at any time such that $M_{>i} = M'_{>i}$. For any sequence of t update in the graph, suppose we carry out two executions I and I' of our algorithm with the initial matching being M and M' respectively. The probability distribution of matching at every level $> i$ will be identical after each update in both the executions. That is,*

$$\mathbf{Pr}[\mathcal{M}_{>i}(t) = \mu(t), \ldots, \mathcal{M}_{>i}(1) = \mu(1) | \mathcal{M}(0) = M] \;=\; \mathbf{Pr}[\mathcal{M}_{>i}(t) = \mu(t), \ldots, \mathcal{M}_{>i}(1) = \mu(1) | \mathcal{M}(0) = M']$$

where $\mu(j)$, for $1 \leq j \leq t$, is any maximal matching on a subset of vertices in the graph $G(j)$.

For the proof of Theorem 2.2, we shall apply the argument for single update inductively and use the following lemma from elementary probability theory.

Lemma 2.9 *Suppose A, B, C are three events defined over a probability space (Ω, P). Then,*

$$\mathbf{Pr}[A \cap B \mid C] = \mathbf{Pr}[A \mid B \cap C] \cdot \mathbf{Pr}[B \mid C]$$

Let us define events C as $\mathcal{M}(0) = M$ and C' as $\mathcal{M}(0) = M'$. Let us define event B as $\mathcal{M}_{>i}(1) = \mu(1)$. We have shown that $\mathbf{Pr}[\mathcal{M}_{>i}(1) = \mu(1) \mid C] = \mathbf{Pr}[\mathcal{M}_{>i}(1) = \mu(1) \mid C']$. That is, $\mathbf{Pr}[B \mid C] = \mathbf{Pr}[B \mid C']$. By another application of the arguments that we used for a single update, we get

$$\mathbf{Pr}[\mathcal{M}_{>i}(2) = \mu(2) \mid B, C] = \mathbf{Pr}[\mathcal{M}_{>i}(2) = \mu(2) \mid B, C']$$

Applying Lemma 2.9, we get

$$\mathbf{Pr}[\mathcal{M}_{>i}(2) = \mu(2), B \mid C] = \mathbf{Pr}[\mathcal{M}_{>i}(2) = \mu(2) \mid B, C] \cdot \mathbf{Pr}[B \mid C]$$

Since $\mathbf{Pr}[B \mid C] = \mathbf{Pr}[B \mid C']$, it follows that

$$\mathbf{Pr}[\mathcal{M}_{>i}(2) = \mu(2), B \mid C] = \mathbf{Pr}[\mathcal{M}_{>i}(2) = \mu(2), B \mid C']$$

The above argument can be inductively applied for every subsequent update. This completes the proof of Theorem 2.2

2.4.1 Connection to the analysis

We first state two lemmas from elementary probability theory that deal with the independence of events. For the sake of completeness, the proof of these lemmas is given in Appendix.

The first lemma deals with conditional probability.

Lemma 2.10 *Let A be an event and $B_1, \ldots, B_k$ be k mutually exclusive events defined over a probability space (Ω, P). If $\mathbf{Pr}[A \mid B_j] = \rho$ for each $1 \leq j \leq k$, then $\mathbf{Pr}[A \mid C] = \rho$ where event $C = \cup_j B_j$.*

The second lemma deals with independence of events. Let A and B be two events defined over a probability space (Ω, P). A is said to be independent of B if $\mathbf{Pr}[A \mid B] = \mathbf{Pr}[A \mid \bar{B}] = \mathbf{Pr}[A]$. Alternatively, $\mathbf{Pr}[A \cap B] = \mathbf{Pr}[A] \cdot \mathbf{Pr}[B]$. The notion of independence gets carried over from events to random variables in a natural manner as follows.

Definition 2.1 *An event A is said to be independent of a random variable X if for each $x \in X$, $\mathbf{Pr}[A \mid X = x] = \mathbf{Pr}[A]$.*

Lemma 2.11 *Suppose A is an event and X is a random variable defined over probability space (Ω, P). If A is independent of X, then for each $x \in X$,*

$$\mathbf{Pr}[X = x \mid A] = \mathbf{Pr}[X = x]$$

Now we shall establish the connection of Theorem 2.2 to the analysis of our algorithm. In particular, we shall use this theorem to prove Lemma 2.3. Suppose a vertex v creates an epoch at level i while the algorithm processes kth update in the graph for any $k < t$. We shall analyse the probability space of the future matchings starting from the time just before the creation of this epoch.

While creating its epoch, v chooses its mate randomly uniformly out of $\mathcal{O}_v^{init}$. Clearly, the change in the matching at levels $\leq i$ will depend on the mate that v picks. Let $\mathbf{M}$ be the set of all possible matchings once the algorithm completes the processing of the kth update. Now notice that all matchings from the set $\mathbf{M}$ are identical at each level $> i$. So it follows from Theorem 2.2 that for any two matchings $M, M' \in \mathbf{M}$,

$$\begin{aligned}\mathbf{Pr}[\mathcal{M}_{>i}(t) = \mu(t), \ldots, &\mathcal{M}_{>i}(k+1) = \mu(k+1) \mid \mathcal{M}(k) = M] \\ &= \mathbf{Pr}[\mathcal{M}_{>i}(t) = \mu(t), \ldots, \mathcal{M}_{>i}(k+1) = \mu(k+1) \mid \mathcal{M}(k) = M']\end{aligned}$$

Let this conditional probability be ρ. For each $(v, w) \in \mathcal{O}_v^{init}$, there may be several matchings in $\mathbf{M}$ in which v is matched to w. By applying Lemma 2.10, the following equation holds for every $(v, w) \in \mathcal{O}_v^{init}$,

$$\mathbf{Pr}[\mathcal{M}_{>i}(t) = \mu(t), \ldots, \mathcal{M}_{>i}(k+1) = \mu(k+1) \mid \textsc{mate}(v) = w] \;=\; \rho$$

Since this probability is the same for each $(v, w) \in \mathcal{O}_v^{init}$, so using Definition 2.1, it follows that the matchings at levels $> i$ during any sequence of updates is independent of the mate that v picks during the creation of its epoch. Now applying Lemma 2.11 we get the following lemma.

Lemma 2.12 *Suppose a vertex v creates an epoch at level i while the algorithm processes kth update in the graph. Consider any sequence of subsequent updates in the graph. The mate picked by v while creating the epoch is independent of the sequence of matchings at levels $> i$ computed by the algorithm while processing these updates. That is, for any $t > k$, and any $(v, w) \in \mathcal{O}_v^{init}$,*

$$\mathbf{Pr}[\textsc{mate}(v) = w | \mathcal{M}_{>i}(t) = \mu(t), \ldots, \mathcal{M}_{>i}(k+1) = \mu(k+1)] \;=\; \mathbf{Pr}[\textsc{mate}(v) = w] \;=\; \frac{1}{|\mathcal{O}_v^{init}|}$$

Consider any given sequence of t updates in the graph. Subsequent to the time v creates an epoch at level i during kth update, let $\mu = \langle \mu(k+1), \ldots, \mu(t) \rangle$ be the sequence of matching at levels $> i$ as computed by the algorithm. Notice that the upward movement of v and each $(v, w) \in \mathcal{O}_v^{init}$ after the creation of epoch is captured precisely by the corresponding update in the matching at level $> i$. Therefore, using μ we can define the update sequence associated with the epoch as follows. Consider an edge $(v, w) \in \mathcal{O}_v^{init}$ and let ℓth update in the graph be the deletion of (v, w).

Let $j < \ell$ be the smallest integer such that $w \in \mu(j)$, that is, w appears in the matching at level $> i$ while processing of jth update in the graph, then the update associated with (v, w) is the upward movement. If no such j exists, the update associated with (v, w) is its deletion. Likewise, we define the update associated with v. The update sequence U for the epoch is the sequence of these updates on v and the edges of $\mathcal{O}_v^{init}$ arranged in the chronological order.

For an update sequence U associated with an epoch, there may exist many sequences $\{\mu_1, \ldots, \mu_q\}$ of matchings at level $> i$ such that for each of them, the update sequence associated with the epoch is U. So, it follows from Lemma 2.12 that the mate picked by v during its epoch is independent of each such sequence $\mu_r, 1 \leq r \leq q$. Therefore, using Lemma 2.10, the mate picked by v during its epoch is independent of U as well. Thus we have established the validity of Lemma 2.3.

3 Acknowledgment

We are very thankful to Sayan Bhattacharya and Divyarthi Mohan for pointing out the error in Lemma 4.10 of [1]. The second author would also like to thank them for the discussions on the proof of the expectation bound and the definition of $B(v, i, k)$. In [1], we used 2^i as the threshold for raising a vertex to level i. The possibility of increasing this threshold from 2^i to b^i for any constant b without any impact on the time complexity of the algorithm was observed by Shay Solomon [3]. We are very thankful to him for this useful observation.

References

[1] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in O(log n) update time. *SIAM J. Comput. (*preliminary version appeared in FOCS 2011*)*, 44(1):88–113, 2015.

[2] William Feller. Random walk and ruin problems. In *An Introduction to Probability Theory and Its Applications*, volume 1, chapter 14, pages 344–348. Wiley, John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop # 02-01 Singapore 129809, 3 edition, 2000.

[3] Shay Solomon. Fully dynamic maximal matching in constant update time. *CoRR*, abs/1604.08491, 2016.

4 Appendix

Proof of Lemma 2.10

Proof:

$$\begin{aligned}
\mathbf{Pr}[A \cap C] &= \mathbf{Pr}[A \cap (\cup_i B_i)] \\
&= \sum_i \mathbf{Pr}[A \cap B_i] \qquad \text{since } B_i\text{'s are mutually exclusive} \\
&= \sum_i \mathbf{Pr}[A \mid B_i] \cdot \mathbf{Pr}[B_i] \qquad \text{using the definition of conditional probability} \\
&= \rho \cdot \sum_i \mathbf{Pr}[B_i] \\
&= \rho \cdot \mathbf{Pr}[\cup_i B_i] = \rho \cdot \mathbf{Pr}[C] \qquad \text{since } B_i\text{'s are mutually exclusive}
\end{aligned}$$

Hence $\mathbf{Pr}[A \mid C] = \mathbf{Pr}[A \cap C]/\mathbf{Pr}[C] = \rho$. □

Proof of Lemma 2.11

Proof: Since A is independent of X, so for each $x \in X$,

$$\mathbf{Pr}[A \cap X = x] = \mathbf{Pr}[A] \cdot \mathbf{Pr}[X = x] \tag{5}$$

Hence

$$\begin{aligned}\mathbf{Pr}[X = x|A] &= \frac{\mathbf{Pr}[A \cap X = x]}{\mathbf{Pr}[A]} \\ &= \frac{\mathbf{Pr}[A] \cdot \mathbf{Pr}[X = x]}{\mathbf{Pr}[A]} \quad \text{using Equation 5} \\ &= \mathbf{Pr}[X = x]\end{aligned}$$

□

Proof for Equation 4 for insertion of an edge

Consider insertion of an edge (u, v). It follows from the initial matchings M and M' that just after this insertion, the edges that u (likewise v) will own upon rising to any level $j > i$ will be the same though it may differ for $j \leq i$. Lemma 2.8 indeed holds. So if u rises to a level $j > i$ in I, u will also rise to level j in I as well. Otherwise, that is, if u rises to a level $\leq i$ in I, then u will also rise, if at all, to a level $\leq i$. In the latter case, the changes in M and M' will be only in levels $\leq i$ due to Fact $F2$ and we are done. In the former case, where u rises to the same level $j > i$ in I and I', henceforth, the proof goes identical to the case of deletion of an edge.