Algorithmica
https://doi.org/10.1007/s00453-019-00630-4

Deterministic Dynamic Matching in $O(1)$ Update Time

Sayan Bhattacharya[1] · **Deeparnab Chakrabarty**[2] · **Monika Henzinger**[3]

Received: 2 March 2018 / Accepted: 9 September 2019

Abstract

We consider the problems of maintaining an approximate maximum matching and an approximate minimum vertex cover in a dynamic graph undergoing a sequence of edge insertions/deletions. Starting with the seminal work of Onak and Rubinfeld (in: Proceedings of the ACM symposium on theory of computing (STOC), 2010), this problem has received significant attention in recent years. Very recently, extending the framework of Baswana et al. (in: Proceedings of the IEEE symposium on foundations of computer science (FOCS), 2011) , Solomon (in: Proceedings of the IEEE symposium on foundations of computer science (FOCS), 2016) gave a randomized dynamic algorithm for this problem that has an approximation ratio of 2 and an amortized update time of $O(1)$ with high probability. This algorithm requires the assumption of an *oblivious adversary*, meaning that the future sequence of edge insertions/deletions in the graph cannot depend in any way on the algorithm's past output. A natural way to remove the assumption on oblivious adversary is to give a deterministic dynamic algorithm for the same problem in $O(1)$ update time. In this paper, we resolve this question. We present a new *deterministic* fully dynamic algorithm that maintains a $O(1)$-approximate minimum vertex cover and maximum fractional matching, with an amortized update time of $O(1)$. Previously, the best deterministic algorithm for this problem was due to Bhattacharya et al. (in: Proceedings of the ACM-SIAM symposium on discrete algorithms (SODA), 2015); it had an approximation ratio of $(2+\varepsilon)$ and an amortized update time of $O(\log n/\varepsilon^2)$. Our result can be generalized to give a fully dynamic $O(f^3)$-approximate algorithm with $O(f^2)$ amortized update time for the hypergraph vertex cover and fractional hypergraph matching problem, where every hyperedge has at most f vertices.

An extended abstract of this paper appeared as [5] in Proceedings of IPCO 2017.

D. Chakrabarty: Work done while the author was at Microsoft Research, India.
M. Henzinger: The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 340506.

✉ Sayan Bhattacharya
s.bhattacharya@warwick.ac.uk

Extended author information available on the last page of the article

Published online: 28 September 2019

Keywords Dynamic algorithms · Data structures · Graph algorithms · Matching · Vertex cover

1 Introduction

Computing a maximum cardinality matching is a fundamental problem in computer science with applications, for example, in operations research, computer science, and computational chemistry. In many of these applications the underlying graph can change. Thus, it is natural to ask how quickly a maximum matching can be maintained after a change in the graph. As nodes usually change less frequently than edges, dynamic matching algorithms usually study the problem where edges are inserted and deleted, which is called the *(fully) dynamic matching problem*.[1] The goal of a dynamic matching algorithm is to maintain either an actual matching (called the *matching version*) or the *value* of the matching (called the *value version*) as efficiently as possible.

Unfortunately, the problem of maintaining even just the value of the maximum cardinality matching is hard: There is a conditional lower bound that shows that no (deterministic or randomized) algorithm can achieve at the same time an amortized update time of $O(m^{1/2-\varepsilon})$ and a query (for the size of the matching) time of $O(m^{1-\varepsilon})$ for any small $\varepsilon > 0$ [15] (see [1] for conditional lower bounds using different assumptions). The best upper bound is Sankowski's randomized algorithm [21] that solves the value problem in time $O(n^{1.495})$ per update and $O(1)$ per query. Thus, it is natural to study the dynamic *approximate* maximum matching problem, and there has been a large body [3,7,8,14,17,18,22] of work on it and its dual, the approximate vertex cover problem, in the last few years.

Dynamic algorithms can be further classified into two types: Algorithms that require an *oblivious* (aka *non-adaptive*) adversary, i.e., an adversary that does *not* base future updates and queries on the answers to past queries, and algorithms that work even for an adaptive adversary. Obviously, the earlier kind of algorithms are less general than the later. Unfortunately, the known randomized dynamic approximate matching and vertex cover algorithms do not work with an adaptive adversary [3,18,22]. Solomon [22] gives the best such randomized algorithm: It achieves $O(1)$ amortized update time (with high probability) and $O(1)$ query time for maintaining a 2-approximate maximum matching and a 2-approximate minimum vertex cover. He also extends this result to the dynamic distributed setting (à la Parter, Peleg, and Solomon [19]) with the same approximation ratio and update cost.

Our Result In this paper we present the first *deterministic* algorithm that maintains an $O(1)$-approximation to the size of the maximum matching in $O(1)$ amortized update time and $O(1)$ query time. We also maintain an $O(1)$-approximate vertex cover in the same update time. Note that this is the first *deterministic* dynamic algorithm with constant update time for any non-trivial dynamic graph problem. This is significant as for other dynamic problems such as the dynamic connectivity problem or the dynamic planarity testing problem there are non-constant lower bounds in the cell probe model

[1] Node updates are usually handled through the insertion and deletion of isolated nodes, but there has been also some work on the node insertions-only or node deletions-only problem [11].

on the time per operation [16,20]. Thus, our result shows that no such lower bound can exist for the dynamic approximate matching problem.

Previous Work There has been prior work on deterministic algorithms for dynamic approximate matching, but they all have $\Omega(\text{poly}(\log n))$ update time. One line of work concentrated on reducing the approximation ratio as much as possible, or at least below 2: Neiman and Solomon [17] achieved an update time $O(\sqrt{m})$ for maintaining a 3/2-approximate maximum matching and 2-approximate minimum vertex cover. This result was improved by Gupta and Peng [14] who gave an algorithm with update time $O(\sqrt{m}/\varepsilon^2)$ for maintaining a $(1+\varepsilon)$-approximate maximum matching. Recently, Bernstein and Stein [4] gave an algorithm with $O(m^{1/4}/\varepsilon^2)$ amortized update time for maintaining a $(3/2+\varepsilon)$-approximate maximum matching. Another line of work, and this paper fits in this line, concentrated on getting a constant approximation while reducing the update time to polylogarithmic: Bhattacharya, Henzinger and Italiano [7] achieved an $O(\log n/\varepsilon^2)$ update time for maintaining a $(2+\varepsilon)$-approximate maximum *fractional* matching and a $(2+\varepsilon)$-approximate minimum vertex cover. Note that any fractional matching algorithm solves the value version of the dynamic matching problem while degrading the approximation ratio by a factor of 3/2 [8]. Thus, the algorithm in [7] maintains a $(3+\varepsilon)$-approximation of the value of the maximum matching. The fractional matching in this algorithm was later "deterministically rounded" by Bhattacharya, Henzinger and Nanongkai [8] to achieve a $O(\text{poly}(\log n, 1/\varepsilon))$ update time for maintaining a $(2+\varepsilon)$-approximate maximum matching. Very recently, a third line of work has focussed on getting small constant approximation in $O(\text{poly}\log n)$ *worst case* update time. Bhattacharya, Henzinger and Nanongkai [9] showed how to maintain a $(2+\varepsilon)$-approximate maximum fractional matching and a $(2+\varepsilon)$-approximate minimum vertex cover in $O(\log^3 n/\varepsilon^4)$ *worst case* update time. Subsequently, Charikar and Solomon [12] and Arar, Chechik, Cohen, Stein and Wajc [2] showed how to maintain a $(2+\varepsilon)$-approximate maximum (integral) matching in $O(\text{poly}\log(n, 1/\varepsilon))$ worst case update time.

Our method also generalizes to the hypergraph vertex (set) cover and hypergraph fractional matching problem which was considered by [6]. In this problem the hyperedges of a hypergraph are inserted and deleted over time. f indicates the maximum cardinality of any hyperedge. The objective is to maintain a hypergraph vertex cover, that is, a set of vertices that hit every hyperedge. Similarly a fractional matching in the hypergraph is a fractional assignment (weights) to the hyperedges so that the total weight faced by any vertex is at most 1. We give an $O(f^3)$-approximate algorithm with amortized $O(f^2)$ update time; compare this with the $O(f^2)$-approximate algorithm due to Bhattacharya et al. [6] whose amortized update time was $O(f \cdot \log(m+n))$; we trade-off $O(f)$ in the approximation factor with $O(\log n)$ in the update time.

Subsequent Work Very recently, after the publication of the conference version of our paper, Bhattacharya and Kulkarni [10] have shown how to maintain a $(2+\varepsilon)$-approximate minimum vertex cover deterministically in a dynamic graph in $O(1/\varepsilon^2)$ amortized update time. When $f = 2$, this improves upon the approximation guarantee of our result (which is a large constant). Arguably, however, our analysis in this paper is more intuitive and easy to follow, compared to the analysis in [10] which uses an extremely intricate potential function. Also, for technical reasons, the analysis in [10]

holds only for simple graphs (with $f = 2$). In other words, the result in [10] does *not* subsume our result on hypergraph vertex (set) cover and fractional matching.

1.1 Our Techniques

Our algorithm builds and simplifies the framework of hierarchical partitioning of vertices proposed by Onak and Rubinfeld [18], which was later enhanced by Bhattacharya, Henzinger and Italiano [7] to give a deterministic fully-dynamic $(2+\delta)$-approximate vertex cover and maximum matching in $O(\log n/\delta^2)$-amortized update time. The hierarchical partition divides the vertices into $L := O(\log n)$-many levels and maintains a *fractional matching* and vertex cover. To prove that the approximation factor is good, Bhattacharya et. al. [7] also maintain approximate complementary slackness conditions. An edge insertion or deletion can disrupt these conditions (and indeed at times the feasibility of the fractional matching), and a *fixing procedure* maintains various invariants. To argue that the update time is bounded, [7] gives a rather involved potential function argument which proves that the update time bounded by $O(L)$, the number of levels, and is thus $O(\log n)$. It seems unclear whether the update time can be argued to be a constant or not.

Our algorithm follows the same approach as Bhattacharya et. al. [7], maintaining invariants for the set of edges incident to a node in the hierarchical graph decomposition, calling nodes that violate them *dirty* and fixing all dirty nodes after each edge insertion or deletion. To fix dirty nodes the algorithm moves them up or down in the hierarchy. All the work in the algorithm consists of inserting and deleting edges and moving nodes up and down the hierarchy. Our algorithm and analysis which we sketch below is very different and simpler than the potential-function based analysis in [7].

To achieve an amortized constant running time, however, we need to develop a number of technical innovations:

- We introduce a new state, called SUPER- CLEAN , that fulfills much stricter invariants, not only on the total set of edges incident to a node, but also *on the subset of these edges whose other endpoint lies at a higher level in the hierarchy*. The fact that an upper bound on this new type of edges (a) can be achieved by the update algorithms and (b) can be exploited to show a constant bound on the amortized running time is the crucial new insight in this paper. Accordingly, we suitably modify the subroutines that move nodes up and down. We show that one of these subroutines, namely the one that moves nodes down the hierarchy, ensures the strict invariants of the SUPER- CLEAN state are fulfilled after it runs; however, the strict invariants need not hold when a node moves up. To analyze, we show that the total time of the "down-movements" is linear in the number T of the update operations; however, arguing about the "up-movements" is trickier. Intuitively, one down-movement might drop a vertex from some level i down to level 0, but it might take i up-movements for the node to achieve level i again. Thus, there might be L times as many up-movements as there are down-movements, and, hence a naive argument may count the total cost for all up-movements to be up to L times the total cost of all down-movements. But this would only result in an $O(LT)$ total time, which would only give a logarithmic amortized time. We show, however, by

using the stricter conditions of the SUPER- CLEAN state, that this is not the case and we can indeed bound the total cost of all up-movements up to level i by the cost of the single down-movement from i.

- To analyze the up-movements, we partition the sequence of operations we perform into epochs. Note that Solomon [22] also used epochs, but with a very different definition of epochs: In [22] an epoch of an edge is a maximal time interval *that an edge remains matched*. For us, an epoch of a vertex is a maximal time period *during which the vertex remains at the same level of the hierarchy*. Since a vertex only becomes SUPER- CLEAN when it moves down, we need to group together consecutive epochs where the vertex moves only up. We call such a consecutive sequence of epochs a *phase* and the crucial part in our analysis is to analyse the cost of the algorithm in each phase. At a high level it works as follows: The number of phases is linear in the number of down-movements plus all the edges that exist at the end. Additionally the way the invariants are set up, the cost of all the epochs in a given phase are geometrically increasing and, thus, the total cost of a phase is dominated by the last epoch in that phase. We then show that the cost of the last epochs of all phases can be "charged" to the cost of all down-movements.

As these technique easily generalize to the hypergraph setting and, thus, we state all our results in the hypergraph setting. It would be interesting to see other scenarios where approximation ratios can be slightly traded in for huge improvements in the update time.

Remark Independently of our work, Gupta et al. [13] has obtained a $O(f^3)$-approximation algorithm for maximum fractional matching and minimum vertex cover in a hypergraph in $O(f^2)$ amortized update time. Here, the symbol f denotes the maximum number of nodes that can be incident on a hyperedge. Their algorithm and analysis are also very similar to ours using an asymmetry between how one treats UP and DOWN moves, and using the notion of epochs like we do.

2 Notations and Preliminaries

Since the hypergraph result implies the graph result, henceforth we consider the former problem. The input hypergraph $\mathscr{G} = (V, E)$ has $|V| = n$ nodes. Initially, the set of hyperedges is empty, i.e., $E = \emptyset$. Subsequently, an adversary inserts or deletes hyperedges in the hypergraph $\mathscr{G} = (V, E)$. The node-set V remains unchanged with time. Each hyperedge contains at most f nodes. We say that f is the maximum *frequency* of a hyperedge. If a hyperedge e has a node v as one of its endpoints, then we write $v \in e$. For every node $v \in V$, we let $E_v = \{e \in E : v \in e\}$ denote the set of hyperedges that are incident on v.

A *matching* is a subset of hyperedges $M \subseteq E$ that do not share any common endpoint. The *size* of a matching M is given by $|M|$. In a *fractional matching*, each hyperedge $e \in E$ receives a (possibly fractional) weight $w(e) \geq 0$ subject to the following constraint: The total weight $W_v = \sum_{e \in E_v} w(e)$ received by every node $v \in V$ from all the hyperedges incident on v is at most 1. The *size* of a fractional matching is given by the sum of the weights of all the hyperedges in $\mathscr{G}$. Note that if

$w(e) \in \{0, 1\}$ for all $e \in E$, then the fractional matching becomes a simple matching $M \subseteq E$, where M is the subset of hyperedges e with $w(e) = 1$. Finally, a vertex cover is a subset of nodes $V' \subseteq V$ such that every hyperedge $e \in E$ has at least one endpoint in V'. The *size* of the vertex cover is given by $|V'|$. From LP-duality, it follows that in any hypergraph $\mathscr{G} = (V, E)$ the size of the maximum fractional matching is at most the size of the minimum vertex cover (see Appendix A).

We consider a fully dynamic setting, where the hypergraph $\mathscr{G} = (V, E)$ is getting updated via a sequence of edge insertions/deletions. In this setting, our goal is to maintain an approximate maximum fractional matching and an approximate minimum vertex cover in $\mathscr{G}$. Our main result is stated in Theorem 2.1.

Theorem 2.1 *We can maintain an $O(f^3)$ approximate maximum fractional matching and an $O(f^3)$ approximate minimum vertex cover in the input hypergraph $\mathscr{G} = (V, E)$ in $O(f^2)$ amortized update time.*

Throughout the rest of this paper, we fix two parameters α and β as follows.

$$\beta = 6, \text{ and } \alpha = 1 + 28 f^2 \beta^2. \tag{1}$$

We will maintain a hierarchical partition of the node-set V into $L + 1$ levels $\{0, \ldots, L\}$, where $L = \lceil f \cdot \log_\beta n \rceil + 1$. We let $\ell(v) \in \{0, \ldots, L\}$ denote the *level* of a node $v \in V$. We define the *level* of a hyperedge $e \in E$ to be the maximum level among its endpoints, i.e., $\ell(e) = \max_{v \in e} \ell(v)$. The levels of nodes (and therefore hyperedges) induce the following weights on hyperedges: $w(e) := \beta^{-\ell(e)}$ for every hyperedge $e \in E$. For all nodes $v \in V$, let $W_v := \sum_{e \in E_v} w(e)$ be the total weight received by v from its incident hyperedges. We will satisfy the following invariant after processing a hyperedge insertion or deletion.

Invariant 2.2 *Every node $v \in V$ at level $\ell(v) > 0$ has weight $1/(\alpha\beta^2) < W_v < 1$. Every node $v \in V$ at level $\ell(v) = 0$ has weight $0 \le W_v \le 1/\beta^2$.*

Corollary 2.3 *Under Invariant 2.2, the nodes in levels $\{1, \ldots, L\}$ form a vertex cover in $\mathscr{G}$.*

Proof Suppose that there is a hyperedge $e \in E$ with $\ell(v) = 0$ for all $v \in e$. Then we also have $\ell(e) = 0$ and $w(e) = 1/\beta^{\ell(e)} = 1/\beta^0 = 1$. So for every node $v \in e$, we get: $W_v \ge w(e) = 1$. This violates Invariant 2.2 as $\beta > 1$. □

Invariant 2.2 ensures that $w(e)$'s form a fractional matching satisfying approximate complementary slackness conditions with the vertex cover defined in Corollary 2.3. This gives the following theorem.

Theorem 2.4 *In our algorithm, the hyperedge weights $\{w(e)\}$ form a $f\alpha\beta^2$-approximate maximum fractional matching, and the nodes in levels $\{1, \ldots, L\}$ form a $f\alpha\beta^2$-approximate minimum vertex cover.*

Proof Consider the set of nodes $T = \{v \in V : \ell(v) > 0\}$. By Corollary 2.3, the set T forms a vertex cover in $\mathscr{G}$. We now compare the size of this vertex cover with the size of the fractional matching $\{w(e)\}$ as follows.

$$\begin{aligned} |T| &= |\{v \in V : \ell(v) > 0\}| \\ &\leq \alpha\beta^2 \cdot \sum_{v \in T} W_v \\ &\leq \alpha\beta^2 \cdot \sum_{v \in V} W_v \\ &\leq f\alpha\beta^2 \cdot \sum_{e \in E} w(e). \end{aligned}$$

In the above derivation, the first inequality holds since $W_v > 1/(\alpha\beta^2)$ for all $v \in T$ under Invariant 2.2. The third inequality holds since every node in $\mathscr{G}$ is incident upon at most f hyperedges. Thus, the set T forms a valid vertex cover in $\mathscr{G}$ and the weights $\{w(e)\}$ form a valid fractional matching in $\mathscr{G}$ such that the size of the vertex cover T is at most $f\alpha\beta^2$ times the size of the fractional matching $\{w(e)\}$. Since the size of the maximum fractional matching in $\mathscr{G}$ is upper bounded by the size of the minimum vertex cover in $\mathscr{G}$, we conclude that the fractional matching $\{w(e)\}$ is a $f\alpha\beta^2$-approximate maximum fractional matching in $\mathscr{G}$, and furthermore, the set of nodes T form a $f\alpha\beta^2$-approximate minimum vertex cover in $\mathscr{G}$. □

We introduce some more notations. For any vertex v, let $W_v^+ := \sum_{e \in E_v : \ell(e) > \ell(v)} w(e)$ be the total *up-weight* received by v, that is, weight from those incident hyperedges whose levels are strictly greater than $\ell(v)$. For all levels $i \in \{0, 1, \ldots, L\}$, we let $W_{v \to i}$ and $W^+_{v \to i}$ respectively denote the values of W_v and W_v^+ *if the node v were to go to level i and the levels of all the other nodes were to remain unchanged*. More precisely, for every hyperedge $e \in E$ and node $v \in e$, we define $\ell_v(e) = \max_{u \in e : u \neq v} \ell(u)$ to be the maximum level among the endpoints of e that are distinct from v. Then we have: $W_{v \to i} := \sum_{e \in E_v} \beta^{-\max(\ell_v(e), i)}$ and $W^+_{v \to i} := \sum_{e \in E_v : \ell_v(e) > i} \beta^{-\ell_v(e)}$. Our algorithm maintains a notion of time such that in each time step the algorithm performs *one* elementary operation. We let $W_v(t)$ denote the weight faced by v *right before* the operation at time t. Similarly define $W_{v \to i}(t)$, $W_v^+(t)$, and $W^+_{v \to i}(t)$.

Different States of a Node Before the insertion/deletion of a hyperedge in $\mathscr{G}$, all nodes satisfy Invariant 2.2. When a hyperedge is inserted (resp. deleted), it increases (resp. decreases) the weights faced by its endpoints. Accordingly, one or more endpoints can violate Invariant 2.2 after the insertion/deletion of a hyperedge. Our algorithm *fixes* these nodes by changing their levels, which may lead to new violations, and so on and so forth. To describe the algorithm, we need to define certain states of the nodes.

Definition 2.5 A node $v \in V$ is DOWN- DIRTY iff $\ell(v) > 0$ and $W_v \leq 1/(\alpha\beta^2)$. A node $v \in V$ is UP- DIRTY iff either $\{\ell(v) = 0, W_v > 1/\beta^2\}$ or $\{\ell(v) > 0, W_v \geq 1\}$. A node is DIRTY if it is either DOWN- DIRTY or UP- DIRTY.

A node that is not DIRTY is CLEAN. Invariant 2.2 is satisfied if and only if no node is DIRTY. Whenever a node becomes DIRTY, we use one of two subroutines to *fix* the

node, i.e., to make it CLEAN. However, for the running time analysis to go through, we need that the weight of node v after it was fixed has a "slack" (or gap) both with regard to the upper and the lower bound of the allowed weight. More specifically, right after a node v was fixed, it holds that $1/\beta^2 < W_v \leq 1/\beta$. But this fact alone still does not suffice to achieve constant running time. Instead we show that when a node was fixed by one of our subroutines, an even stronger condition is fulfilled, and we call a node fulfilling that condition to be a SUPER- CLEAN node.

Definition 2.6 A node $v \in V$ is SUPER- CLEAN iff one of the following conditions hold: (1) We have $\ell(v) = 0$ and $W_v \leq 1/\beta^2$, or (2) We have $\ell(v) > 0$, $1/\beta^2 < W_v \leq 1/\beta$, and $W_v^+ \leq 1/\beta^2$.

Note that a SUPER- CLEAN node v with $\ell(v) > 0$ fulfills a stronger upper bound on the weight W_v then a CLEAN node and additionally it fulfills an upper bound on the up-weight W_v^+.

Data Structures For all nodes $v \in V$ and levels $i \in \{0, 1, \ldots, L\}$, let $E_{v,i} := \{e \in E_v : \ell(e) = i\}$ denote the set of hyperedges incident on v that are at level i. Note that $E_{v,i} = \emptyset$ for all $i < \ell(v)$. We will maintain the following data structures.

1. For every level $i \in \{0, 1, \ldots, L\}$ and node $v \in V$, we store the set of hyperedges $E_{v,i}$ as a doubly linked list, and also maintain a counter that stores the number of hyperedges in $E_{v,i}$.
2. For every node $v \in V$, we store the weights W_v and W_v^+, its level $\ell(v)$ and an indicator variable for each of the states DOWN- DIRTY, UP- DIRTY, DIRTY and SUPER- CLEAN.
3. For each hyperedge $e \in E$, we store the values of its level $\ell(e)$ and therefore its weight $w(e)$. Finally, using appropriate pointers, we ensure that a hyperedge can be inserted into or deleted from any given linked list in constant time.

We now state two lemmata that will be useful in analysing the update time of our algorithm.

Lemma 2.7 *Suppose that a node v is currently at level $\ell(v) = i \in [0, L-1]$ and we want to move it to some level $j \in [i+1, L]$. Then it takes $O(f \cdot |\{e \in E_v : \ell_v(e) < j\}|)$ time to update the relevant data structures.*

Proof If a hyperedge e is not incident on the node v, then the data structures associated with e are not affected as v moves up from level i to level j. Further, among the hyperedges $e \in E_v$, only the ones with $\ell_v(e) < j$ get affected (i.e., the data structures associated with them need to be changed) as v moves up from level i to level j. Note that just before the node v moves up from level i to level j, the set of hyperedges $e \in E_v$ with $\ell_v(e) < j$ is given by $\bigcup_{k=i}^{j-1} E_{v,i}$. This holds since $i < j$. Finally, for every hyperedge that gets affected, we need to spend $O(f)$ time to update the data structures for its f endpoints. □

Lemma 2.8 *Suppose that a node v is currently at level $\ell(v) = i \in [1, L]$ and we want to move it down to some level $j \in [0, i-1]$. Then it takes $O(f \cdot |\{e \in E_v : \ell_v(e) \leq i\}|)$ time to update the relevant data structures.*

```
01.  WHILE the set of DIRTY nodes is nonempty
02.      IF there exists some UP-DIRTY node v:
03.          FIX-UP-DIRTY(v).
04.      ELSE IF there exists some DOWN-DIRTY node v:
05.          FIX-DOWN-DIRTY(v).
```

Fig. 1 FIX-DIRTY(·)

Proof If a hyperedge e is not adjacent to the node v, then the data structures associated with e are not affected as v moves down from level i to level j. Further, among the hyperedges $e \in E_v$, only the ones with $\ell_v(e) \leq i$ get affected (i.e., the data structures associated with them need to be changed) as v moves down from level i to level j. Finally, for every hyperedge that gets affected, we need to spend $O(f)$ time to update the data structures for its f endpoints. □

3 The algorithm: Handling the Insertion/Deletion of a Hyperedge

Initially, the graph $\mathscr{G}$ is empty, every node is at level 0, and Invariant 2.2 holds. By induction, we will ensure that the following property is satisfied just before the insertion/deletion of a hyperedge.

Property 3.1 *No node $v \in V$ is* DIRTY.

Insertion of a hyperedge e When a hyperedge e is inserted into the input graph, it is assigned a level $\ell(e) = \max_{v \in e} \ell(v)$ and a weight $w(e) = \beta^{-\ell(e)}$. The hyperedge gets inserted into the linked lists $E_{v,\ell(e)}$ for all nodes $v \in e$. Furthermore, for every node $v \in e$, the weights W_v increases by $w(e)$. For every endpoint $v \in e$, if $\ell(v) < \ell(e)$, then the weight W_v^+ increases by $w(e)$. As a result of these operations, one or more endpoints of e can now become UP- DIRTY and Property 3.1 might no longer be satisfied. Hence, in order to restore Property 3.1 we call the subroutine described in Fig. 1.

Deletion of a hyperedge e When a hyperedge e is deleted from the input graph, we erase all the data structures associated with it. We remove the hyperedge from the linked lists $E_{v,\ell(e)}$ for all $v \in e$, and erase the values $w(e)$ and $\ell(e)$. For every node $v \in e$, the weight W_v decreases by $w(e)$. Further, for every endpoint $v \in e$, if $\ell(v) < \ell(e)$, then we decrease the weight W_v^+ by $w(e)$. As a result of these operations, one or more endpoints of e can now become DOWN- DIRTY, and Property 3.1 might get violated. Hence, in order to restore Property 3.1 we call the subroutine described in Fig. 1.
The algorithm is simple – as long as some DIRTY node remains, it runs either FIX-UP-DIRTY or FIX-DOWN-DIRTY to take care of UP- DIRTY and DOWN- DIRTY nodes respectively. One crucial aspect is that we prioritize UP- DIRTY nodes over DOWN-DIRTY ones as it allows us to upper bound the weight of every node by 2, since any individual weight increase increases the weight of a node by at most 1 and the call FIX-UP-DIRTY does not increase the weight of any node, while FIX-DOWN-DIRTY might do so. Thus, as long as we are fixing UP- DIRTY nodes as soon as they become

UP- DIRTY, no node will ever have weight 2 or larger, a fact that we will exploit in the running time analysis.

FIX-DOWN-DIRTY(v) Suppose that $\ell(v) = i$ when the subroutine is called at time t. By definition, we have $i > 0$ and $W_v(t) \leq 1/(\alpha\beta^2)$. We need to increase the value of W_v if we want to ensure that v no longer remains DIRTY. This means that we should decrease the level of v, so that some of the hyperedges incident on v can increase their weights. Accordingly, we find the *largest* possible level $j \in \{1, \ldots, (i-1)\}$ such that $W_{v\to j}(t) > 1/\beta^2$, and move the node v down to this level j. If no such level exists, that is, if even $W_{v\to 1}(t) \leq 1/\beta^2$, then we move the node v down to level 0. Note that in this case there is no hyperedge $e \in E_v$ with $\ell_v(e) = 0$ for such a hyperedge would have $w(e) = \beta^{-1} > 1/\beta^2$ when v is moved to level 1. In particular, we get $W_{v\to 0}(t) = W_{v\to 1}(t)$.

Claim 3.2 FIX- DOWN- DIRTY*(v) makes the node v* SUPER- CLEAN .

Proof Suppose node v was at level i when FIX-DOWN-DIRTY(v) was called at time t and it ended up in level $j < i$. If $j = 0$, then $W_{v\to 0}(t) \leq 1/\beta^2$, and so v becomes SUPER- CLEAN after time t. Henceforth assume $j > 0$. Since $j \in \{1, \ldots, i-1\}$ is the maximum level where $W_{v\to j}(t) > 1/\beta^2$, we have $W_{v\to(j+1)}(t) \leq 1/\beta^2$. Now note that $W_{v\to j}(t) \leq \beta \cdot W_{v\to(j+1)}(t)$ since weights of hyperedges can increase by at most a factor β when one end point drops exactly one level. This implies $W_{v\to j}(t) \leq 1/\beta$. Together we get that after time t when v is fixed to level j, we have $1/\beta^2 < W_v \leq 1/\beta$.

Now we argue about the up-weights. Note that every hyperedge e that contributes to $W^+_{v\to j}(t)$ must have $\ell_v(e) \geq (j+1)$. The weight of such a hyperedge remains unchanged as v moves from level $(j+1)$ to j. We infer that $W^+_{v\to j}(t) \leq W_{v\to(j+1)}(t) \leq 1/\beta^2$. Therefore after time t when v is fixed at level j, we have $W^+_v \leq 1/\beta^2$. In sum, v becomes SUPER- CLEAN after time t. □

FIX-UP-DIRTY(v) Suppose that $\ell(v) = i$ when the subroutine is called at time t. At this stage, we have either $\{i = 0, W_v(t) > 1/\beta^2\}$ or $\{i > 1, W_v(t) \geq 1\}$. We need to increase the level of v so as to reduce the weight faced by it. Accordingly, we find the *smallest* possible level $j \in \{i+1, \ldots, L\}$ where $W_{v\to j}(t) \leq 1/\beta$ and move v up to level j. Such a level j always exists because $W_{v\to L}(t) \leq n^f \cdot \beta^{-L} \leq 1/\beta$.

Claim 3.3 *After a call to the subroutine* FIX- DOWN- DIRTY*(v) at time t, we have* $1/\beta^2 < W_v \leq 1/\beta$.

Proof Suppose that the node v moves up from level i to level $j > i$. We now consider four possible cases.

- *Case 1. We have* $i > 0$.
  Since $j \in \{i+1, \ldots, L\}$ is the minimum possible level where $W_{v\to j}(t) \leq 1/\beta$, and since $W_{v\to i}(t) \geq 1 > 1/\beta$, we infer that $W_{v\to(j-1)}(t) > 1/\beta$. As the node v moves up from level $(j-1)$ to level j, the weight it faces can drop by at most a factor of β. Hence, we get: $W_{v\to j}(t) \geq (1/\beta) \cdot W_{v\to(j-1)}(t) > 1/\beta^2$. Therefore after time t when the node v moves to level j, we have $1/\beta^2 < W_v \leq 1/\beta$.
- *Case 2. We have* $i = 0$, *and there is an edge* $e \in E_v$ *with* $\ell_v(e) = 0$ *at time* t.In this case we have $W_{v\to i}(t) \geq 1$ in the beginning of time-step t, since the edge $e \in E_v$ with $\ell_v(e) = 0$ has weight $w(e) = 1$. The rest of the proof is similar to Case 1.

- *Case 3. We have $i = 0$, there is no edge $e \in E_v$ with $\ell_v(e) = 0$ at time t, and $j = 1$.*
  The value of W_v does not change as v moves up from level $i = 0$ to level $j = 1$. Thus, we get: $W_{v\to j}(t) = W_{v\to 0}(t) > 1/\beta^2$, for the node v is UP- DIRTY at level $i = 0$ at time t. Since the node does not move further up than level j, we get: $W_{v\to j}(t) \le 1/\beta$.
- *Case 4. We have $i = 0$, there is no edge $e \in E_v$ with $\ell_v(e) = 0$ at time t, and $j > 1$.*
  Since the node v does not stop at level 1, we get: $W_{v\to 1}(t) > 1/\beta$. Hence, we infer that $j \in \{2, \ldots, L\}$ is the minimum possible level where $W_{v\to j}(t) \le 1/\beta$. Since the node v had to rise above level $j-1$, we infer that $W_{v\to(j-1)}(t) > 1/\beta$. Next, since the weight of the node v can drop by at most a factor of β when it rises from level $j-1$ to level j, we get: $W_{v\to j}(t) \ge (1/\beta) \cdot W_{v\to(j-1)}(t) > 1/\beta^2$. Therefore, we have $1/\beta^2 < W_{v\to j}(t) \le 1/\beta$. □

It is clear that if and when FIX-DIRTY() terminates, we are in a state which satisfies Invariant 2.2. In the next section we show that after T hyperedge insertions and deletions, the total update time is indeed $O(f^2 \cdot T)$ and so our algorithm has $O(f^2)$-amortized update time.

4 Analysis of the Algorithm

Starting from an empty graph $\mathcal{G} = (V, E)$, fix any sequence of T *updates*. The term "update" refers to the insertion or deletion of a hyperedge in $\mathcal{G}$. We show that the total time taken by our algorithm to handle this sequence of updates is $O(f^2 \cdot T)$. We also show that our algorithm has an approximation ratio of $O(f^3)$.

Relevant Counters We define three counters C^{up}, C^{down} and I^{down}. The first two counters account for the time taken to update the data structures while the third accounts for the time taken to find the index j in both FIX-DOWN-DIRTY(v) and FIX-UP-DIRTY(v). Initially, when the input graph is empty, all the three counters are set to zero. Subsequently, we increment these counters as follows.

1. Suppose node v moves from level i to level $j > i$ upon a call of FIX-UP-DIRTY(v). Then for every hyperedge $e \in E_v$ with $\ell_v(e) \le j-1$, we increment C^{up} by one.
2. Suppose node v moves from level i to level $j < i$ upon a call of FIX-DOWN-DIRTY(v). Then for every hyperedge $e \in E_v$ with $\ell_v(e) \le i$, we increment the value of C^{down} by one. Furthermore, we increment the value of I^{down} by β^{i-2}/α.

Remark We will see in Sect. 4.5 that the time complexity of finding the appropriate index j during FIX-UP-DIRTY is subsumed by the increment in I^{down} during FIX-DOWN-DIRTY (we will crucially exploit the fact that each sequence of up moves will be followed by a down move since the final graph is empty graph).

We now explain why we increment I^{down} during a call to FIX-DOWN-DIRTY which moves a node v down from level i to level $j < i$. Note that in order to compute

the new level of v, it suffices to process all the hyperedges whose other endpoints are at levels $\leq i$. Each of these hyperedges has weight β^{-i}. Furthermore, given that the node was down-dirty, its current weight is less than $1/(\beta^2\alpha)$. Hence, the number of hyperedges we need to scan to find the level j is upper bounded by β^{i-2}/α.

The next lemma upper bounds the total time taken by our algorithm in terms of the values of these counters. The proof of Lemma 4.1 appears in Sect. 4.5.

Lemma 4.1 *Our algorithm takes* $\Theta(f \cdot (C^{up} + C^{down} + T) + f^2 I^{down})$ *time to handle a sequence of* T *updates.*

We will show that $C^{up} = \Theta(f) \cdot T$ and $C^{down} + I^{down} = O(1) \cdot T$, which will imply an amortized update time of $O(f^2)$ for our algorithm. Towards this end, we now prove three lemmata that relate the values of these three counters. The proofs can be found in the next subsections.

The first lemma follows immediately from the fact that at most β^{i-2}/α edges fulfill the condition $\ell_v(e) \leq i$ when node v moves down, and thus, I^{down} is incremented by at most as much as C^{down}.

Lemma 4.2 *We have:* $C^{down} \leq I^{down}$.

The next lemma exploits the gap between the weight when a node v reaches a level i and when v moves down: When v is placed on level i we have $W_v > 1/\beta^2$, when it is DOWN- DIRTY it holds that $W_v \leq 1/(\alpha\beta^2)$. This "slack" suffices to upper bound I^{down} by a function linear in T and C^{up}. Thus, this proof compares the weight of node v at the beginning and at the end of a time interval during which v *remains at the same level*. Note that we picked α such that the constant is much smaller than 1 and even a function of $1/f$.

Lemma 4.3 *We have:* $I^{down} \leq \frac{f}{\alpha - 1} \cdot (T + C^{up})$.

Note that when in prior work such as [7] the "slack" was exploited by comparing the weight of v at two different points in time, node v had to be *at the same level* from time t_1 on until time t and *the whole weight of* v *on this level* was used in the comparison.

In order to prove the next lemma we compares the weight of v at two points in time *even though* v *is at different levels at these points in time*. More specifically, we use the following new three ideas:

(1) We break, for each node v, the sequence of operations into subsequences and then argue over a maximal subsequence of operations during which the level of v only increases, called *phase*. When we compare the weight of v, we use as t_1 the beginning of the phase and as t a later point in time during the phase. Thus, v will be t different levels at these points in time. (Note that both the definition of a phase and its subintervals, called *epochs*, are different from the epochs introduced in [22].)
(2) We use a *reference level* i^*, on which v might never be during the phase and compare the weight of v at time t_1 if it were at a level i^* with the weight of v if it were at level i^* at time t.

(3) In this comparison we use, however, not the complete weight of v, but only the *weight of v to upper levels*, i.e. we use the weight $W^{+}_{v \to i^*}$. When doing so, we use the strong properties that are guaranteed by the SUPER- CLEAN state.

These ideas enable us finally to upper bound C^{up} by a function linear in T and C^{down}.

Lemma 4.4 *We have:* $C^{up} \leq 7f\beta^2 \cdot (T + C^{down})$.

The proofs of lemmata 4.2, 4.3 and 4.4 appear in Sects. 4.2, 4.3 and 4.4 respectively, using the concepts of epochs, jumps and phases which are defined in Sect. 4.1. When combining their statements we can upper bound all three counters by a function that is linear in T, which was our goal, as it shows that the number of edges that are updated during all calls to FIX-DOWN-DIRTY and FIX-UP-DIRTY is linear in the number of update operations in the graph.

Lemma 4.5 *(*Corollary to Lemma 4.2,4.3, and 4.4.*) We have:* $C^{up} = \Theta(f) \cdot T$ *and* $C^{down} + I^{down} = \Theta(1) \cdot T$.

Proof Replacing C^{down} in the RHS of Lemma 4.4 by the upper bounds from lemmata 4.2 and 4.3, we get:

$$\begin{aligned} C^{up} &\leq (7f\beta^2) \cdot T + (7f\beta^2) \cdot C^{down} \\ &\leq (7f\beta^2) \cdot T + (7f\beta^2) \cdot I^{down} \\ &\leq (7f\beta^2) \cdot T + \frac{(7f\beta^2)f}{(\alpha - 1)} \cdot (T + C^{up}) \\ &\leq (7f\beta^2) \cdot T + (1/4) \cdot T + (1/4) \cdot C^{up} \qquad \text{(see equation (1))} \end{aligned}$$

Rearranging the terms in the above inequality, we get: $(3/4) \cdot C^{up} \leq (7f\beta^2 + 1/4) \cdot T = (28f\beta^2 + 1) \cdot (T/4)$. Multiplying both sides by $(4/3)$, we get: $C^{up} \leq (28f\beta^2/3 + 1/3) \cdot T \leq (10f\beta^2)T$. Since $\beta = 6$, we get:

$$C^{up} \leq \Theta(f) \cdot T \tag{2}$$

Since $\alpha = \Theta(f^2)$, lemmata 4.2 and 4.3 and Eq. (2) imply that:

$$C^{down} \leq I^{down} \leq \Theta(1) \cdot T \tag{3}$$

□

The main result of our paper (see Theorem 2.1) now follows from Theorem 2.4, Lemma 4.1 and Lemma 4.5.

4.1 Epochs, Jumps and Phases

Fix any node $v \in V$. An *epoch* of this node is a maximal time-interval during which the node stays at the same level. An epoch ends when either (a) the node v moves

up to a higher level due to a call to FIX-UP-DIRTY, or (b) the node v moves down to a lower level due to a call to the subroutine FIX-DOWN-DIRTY. These events are called *jumps*. Accordingly, there are UP- JUMPS and DOWN- JUMPS. Next, we define a *phase* of a node to be a maximal sequence of at least two consecutive epochs where the level of v keeps on increasing, i.e., where all but the last epoch end in an UP-JUMP. We use Φ_v to denote a phase of a node v and we will analyse each such phase. Suppose that a phase Φ_v consists of $k > 1$ consecutive epochs of v at levels $i_1, \ldots, i_k \in \{0, 1, \ldots, L\}$. Then we have: $i_1 < i_2 < \cdots < i_k$. By definition, the epoch immediately before Φ_v must have level larger than i_1 implying FIX-DOWN-DIRTY(v) landed v at level i_1. Similarly, the epoch subsequent to i_k is smaller than i_k implying FIX-DOWN-DIRTY(v) is called again.

4.2 Proof of Lemma 4.2

Suppose that a node v moves down from level i to level $j < i$ at time t due to a call to the subroutine FIX-DOWN-DIRTY(v). Let Δ^{down} and Δ_I^{down} respectively denote the increase in the counters C^{down} and I^{down} due to this event. We will show that $\Delta^{down} \leq \Delta_I^{down}$, which will conclude the proof of the lemma. By definition, we have:

$$\Delta_I^{down} = \beta^{i-2}/\alpha \tag{4}$$

Let $X = \{e \in E_v : \ell_v(e) \leq i\}$ denote the set of hyperedges incident on v that contribute to the increase in C^{down} due to the DOWN- JUMP of v at time t. Note that if a hyperedge e belongs to the set X, then it must be the case that $\ell_v(e) = i$. Specifically, we have: $|X| = \Delta^{down}$. Each edge $e \in X$ contributes a weight β^{-i} towards the node-weight $W_{v \to i}(t)$. Thus, we get: $|X| \cdot \beta^{-i} \leq W_{v \to i}(t) \leq 1/(\alpha\beta^2)$. The last inequality holds since v is DOWN- DIRTY in the beginning of time-step t. Rearranging the terms, we get: $\Delta^{down} = |X| \leq \beta^{i-2}/\alpha$. The lemma now follows from Eq. (4).

4.3 Proof of Lemma 4.3

Suppose we call FIX-DOWN-DIRTY(v) at some time t_2. Let $\ell(v) = i$ just before the call, and let $[t_1, t_2]$ be the epoch with level of v being i. Let $X := \{e \in E_v : \ell_v(e) \leq i\}$ at time t_2. By definition, I^{down} increases by β^{i-2}/α during the execution of FIX-DOWN-DIRTY(v); let us call this increase Δ_I^{down}. Thus, we have:

$$\Delta_I^{down} = \beta^{i-2}/\alpha \tag{5}$$

Consider the time between $[t_1, t_2]$ and let us address how W_v can decrease in this time while v's level is fixed at i. Either some hyperedge incident on v is deleted, or some hyperedge $e \in E_v$ incident on it decreases its weight. In the latter case, the level $\ell(e)$ of such an hyperedge e must increase above i. Let Δ^T denote the number of hyperedge deletions incident on v during the time-interval $[t_1, t_2]$. Let Δ^{up} denote the increase in the value of C^{up} during the time-interval $[t_1, t_2]$ *due to the hyperedges incident on* v. Specifically, at time t_1, we have $\Delta^T = \Delta^{up} = 0$. Subsequently, during

the time-interval $[t_1, t_2]$, we increase the value of Δ^{up} by one each time we observe that a hyperedge $e \in E_v$ increases its level $\ell(e)$ to something larger than i. Note that $\ell(v) = i$ throughout the time-interval $[t_1, t_2]$. Hence, each time we observe an unit increase in $\Delta^T + \Delta^{up}$, this decreases the value of W_v by at most β^{-i}. Just before time t_1, the node v made either an UP- JUMP, or a DOWN- JUMP. Hence, Claims 3.3 and 3.2 imply that $W_{v\to i}(t_1) > 1/\beta^2$. As $W_v(t_2) \leq 1/(\alpha\beta^2)$ at time t_2, we infer that W_v has dropped by at least $(1 - 1/\alpha) \cdot \beta^{-2}$ during the time-interval $[t_1, t_2]$. In order to account for this drop in W_v, the value of $\Delta^T + \Delta^{up}$ must have increased by at least $(1 - 1/\alpha) \cdot \beta^{-2}/\beta^{-i} = (1 - 1/\alpha) \cdot \beta^{i-2}$. Since $\Delta^T = \Delta^{up} = 0$ at time t_1, at time t_2 we get: $\Delta^T + \Delta^{up} \geq (1 - 1/\alpha) \cdot \beta^{i-2}$. Hence, (5) gives us:

$$\Delta_I^{down} \leq (\alpha - 1)^{-1} \cdot (\Delta^T + \Delta^{up}) \tag{6}$$

Each time the value of I^{down} increases due to FIX-DOWN-DIRTY on some node, inequality (6) applies. If we sum all these inequalities for all nodes v, then the left hand side (LHS) will be exactly equal to the final value of I^{down}, and the right hand side (RHS) will be at most $(\alpha - 1)^{-1} \cdot (f \cdot T + (f - 1) \cdot C^{up})$. The factor f appears in front of T because each hyperedge deletion can contribute f times to the sum $\sum \Delta^T$, once for each of its endpoints. Similarly, the factor $(f - 1)$ appears in front of C^{up} because whenever the level of an hyperedge e moves up due to the increase in the level $\ell(v)$ of some endpoint $v \in e$, this contributes at most $(f - 1)$ times to the sum $\sum \Delta^{up}$, once for every other endpoint $u \in e, u \neq v$. Since LHS $\leq$ RHS, we get: $I^{down} \leq (\alpha - 1)^{-1} \cdot (f \cdot T + (f - 1) \cdot C^{up}) \leq (f/(\alpha - 1)) \cdot (T + C^{up})$. This concludes the proof of the lemma.

4.4 Proof of Lemma 4.4

To upper bound C^{up} by a function linear in T and C^{down} we analyse the increase in C^{up} caused by each node v and each phase of v individually. Thus, we first need to define suitable variables. Fix a node v and consider a phase Φ_v where v goes through levels $i_1 < i_2 < \cdots < i_k$. Thus, the node v enters the level i_1 at time t_1 (say) due to a call to FIX-DOWN-DIRTY(v). For $r \in [2, k]$, the node v performs an UP- JUMP at time t_r (say) from the level i_{r-1} to the level i_r, due to a call to FIX-UP-DIRTY(v). This implies that $t_1 < t_2 < \cdots < t_k$. The phase ends, say, at time $t_{k+1} > t_k$ when the node v again performs a DOWN- JUMP from the level i_k due to a call to FIX-DOWN-DIRTY(v). As we will argue in this proof about the weight of v at various levels we use the notation $W_{v\to i_1}(t_1)$ to denote the weight of v on level i_1 at time t_1 as this notation clearly shows the three parameters that the weight depends on (v, i_1, and t_1), even if we could simply use $W_v(t_1)$ if the level of v at time t_1 is i_1.

Let Δ^{up} denote the total increase in the value of the counter C^{up} during the phase Φ_v, that happens *because of* the node v. For $r \in [2, k]$, let Δ_r^{up} denote the increase in the value of the counter C^{up} due to the UP- JUMP of v at time t_r. Thus, we have:

$$\Delta^{up} = \sum_{r=2}^{k} \Delta_r^{up} \tag{7}$$

We define two more counters: Δ^T, Δ^{down}. The former counter equals the number of hyperedge insertions/deletions incident on v during the time-interval $[t_1, t_k]$. The latter counter equals the increase in the value of C^{down} due to the hyperedges incident on v during the time-interval $[t_1, t_k]$. Thus, Δ^T and Δ^{down} are functions of v, t_1 and t_k, but we omit these parameters to simplify the notation in this proof. (Alternately, these two counters can be defined as follows. At time t_1, we set $\Delta^T \leftarrow 0$ and $\Delta^{down} \leftarrow 0$. Subsequently, whenever at any time $t \in [t_1, t_k]$, a hyperedge incident on v gets inserted into or deleted from the input graph, we increment the value of Δ^T by one. Further, whenever at any time $t \in [t_1, t_k]$, a hyperedge e incident on v gets its level decreased because of a DOWN- JUMP of some node $u \in e, u \neq v$, we increment the value of Δ^{down} by one.)

Before proceeding with the proof we first state a crucial property of $W_{v \to i_1}(t_1)$ that we will use repeatedly below. Since v enters the level i_1 at time t_1 due to a call to FIX-DOWN-DIRTY(v), Claim 3.2 implies that the node v is SUPER- CLEAN at level i_1 at time t_1. We now recall the definition of a SUPER- CLEAN node from Definition 2.6. There are two cases to consider, depending on the value of i_1.

Case 1. $i_1 > 0$. In this case, Definition 2.6 implies that $W_{v \to i_1}(t_1) \leq 1/\beta$ and $W^+_{v \to i_1}(t_1) \leq 1/\beta^2$.

Case 2. $i_1 = 0$. In this case, Definition 2.6 implies that $W_{v \to i_1}(t_1) \leq 1/\beta^2 \leq 1/\beta$. Furthermore, Definition 2.6 also implies that $W^+_{v \to i_1}(t_1) \leq W_{v \to i_1}(t_1) \leq 1/\beta^2$.

We therefore obtain the following strict bounds on W_v:

$$W_{v \to i_1}(t_1) \leq 1/\beta \qquad \text{and} \qquad W^+_{v \to i_1}(t_1) \leq 1/\beta^2 \tag{8}$$

Our main goal is to upper bound Δ^{up} in terms of the final values of the counters Δ^T and Δ^{down}. To achieve this we upper bound in the next claim Δ^{up}_r by $\beta^{i_r - 1}$ and in the following claim we lower bound $\Delta^T + \Delta^{down}$ by a function of $\beta^{i_r - 1}$. In the proof of the next claim we use the fact that we have an upper bound of $1/\beta$ on W_v right after an UP- JUMP and we know the exact weight of all edges that contribute to Δ^{up}_r for the UP- JUMP. As these edges also contribute to $W_{v \to i_r}(t_r + 1)$, the upper bound to $W_{v \to i_r}(t_r + 1)$ bounds the number of such edges and thus Δ^{up}_r.

Claim 4.6 *For* $2 \leq r \leq k$, $\Delta^{up}_r \leq \beta^{i_r - 1}$.

Proof By Claim 3.3 we have $W_{v \to i_r}(t_r + 1) \leq 1/\beta$, that is, the total weight incident on v after it has gone through FIX-UP-DIRTY at time t_r is at most $1/\beta$. Now, each hyperedge $e \in E_v$ which contributes to Δ^{up}_r has weight, right after time t_r, precisely $\beta^{-\ell(v)} = \beta^{-i_r}$. As $\Delta^{up}_r \cdot \beta^{-i_r} \leq W_{v \to i_r}(t_r + 1) \leq 1/\beta$, we get $\Delta^{up}_r \leq \beta^{i_r - 1}$. □

Using the slack provided by Eq. (8) and the fact that v performs an UP- JUMP at time t_{k-1} we can lower bound the increase in $W_{v \to i_{k-1}}$ during the time interval $[t_1, t_k]$ by $(1 - 1/\beta)$. As this weight increase can only happen through insertions of hyperedges incident to v or hyperedges incident to v increasing their weights, *and* each such event increases the weight of v by at most $\beta^{-i_{k-1}}$, we can lower bound the sum $\Delta^T + \Delta^{down}$ by $\beta^{i_{k-1}}/2$. Combining this lower bound with the preceeding claim shows the desired upper bound on the sum of all but the last Δ^{up}_k by a linear function of $\Delta^T + \Delta^{down}$.

Claim 4.7 *For $k > 2$ we have $\Delta^T + \Delta^{down} > \beta^{i_{k-1}}/2$ and for $k > 1$ we have $\sum_{r=2}^{k-1} \Delta_r^{up} < 2(\Delta^T + \Delta^{down})$.*

Proof If $k = 2$, then we have an empty sum $\sum_{r=2}^{k-1} \Delta_r^{up} = 0$, and hence the claim is trivially true.

For the rest of the proof, we suppose that $k > 2$, which implies that $i_{k-1} \geq i_2 > i_1 \geq 0$. Thus, we get:

$$k > 2 \text{ and } i_{k-1} > 0. \tag{9}$$

Summing over the inequalities from Claim 4.6, we get:

$$\sum_{r=2}^{k-1} \Delta_r^{up} \leq \beta^{i_{k-1}} \tag{10}$$

Since the node v performs an Up- Jump at time t_{k-1} from level $i_{k-1} > 0$ (see Eq. (9)), the node must be Up- Dirty at that time. It follows that $W_{v \to i_{k-1}}(t_k) > 1$. From Eq. (8), we have $W_{v \to i_{k-1}}(t_1) \leq W_{v \to i_1}(t_1) < 1/\beta$. Thus, during the time interval $[t_1, t_k]$ the value of $W_{v \to i_{k-1}}$ increases by at least $(1 - 1/\beta)$. This can be either due to (a) some hyperedge incident to v being inserted, or (b) some hyperedge $e \in E_v$ gaining its weight because of some endpoint $u \in e, u \neq v$, going down. The former increases Δ^T and the latter increases Δ^{down}. Furthermore, the increase in $W_{v \to i_{k-1}}$ due to every such hyperedge is at most $\beta^{-i_{k-1}}$. This gives us the following lower bounds:

$$\begin{aligned} &\Delta^T + \Delta^{down} \geq (1 - 1/\beta) \cdot \beta^{i_{k-1}} > \beta^{i_{k-1}}/2 \quad \text{and, with equation (10),} \\ &\sum_{r=2}^{k-1} \Delta_r^{up} < 2(\Delta^T + \Delta^{down}) \end{aligned} \tag{11}$$

The claim follows from Eq. (11). □

It now remains to upper bound Δ_k^{up}. This is done in Claim 4.8, whose proof appears in Sect. 4.4.1.

Claim 4.8 *We have: $\Delta_k^{up} < (6\beta^2) \cdot (\Delta^T + \Delta^{down})$.*

Putting it all together from Eqs. 1, 7 and Claims 4.7, 4.8, we get:

$$\Delta^{up} \leq (6\beta^2 + 2)(\Delta^T + \Delta^{down}) \leq (7\beta^2) \cdot (\Delta^T + \Delta^{down}) \tag{12}$$

Using Eq. (12), now we can prove Lemma 4.4. For every phase of a node v, as per Eq. (12) we can charge the increase in C^{up} to the increase in $(T + C^{down})$ corresponding to hyperedges incident of v. Summing up over all nodes and phases, the LHS gives C^{up} while the RHS gives $(9\beta^2) \cdot (f \cdot T + (f - 1) \cdot C^{down})$. The coefficient f before T comes from the fact that every hyperedge insertion can contribute f times to the RHS, once for each of its endpoints. The coefficient $(f - 1)$ before C^{down} comes from the fact that whenever the level of a hyperedge e decreases due to the Down- Jump of a

node $u \in e$, this event contributes at most $(f-1)$ times to the RHS: once for every other endpoint $v \in e$, $v \neq u$. Thus, we get:

$$C^{up} \leq (7\beta^2) \cdot (f \cdot T + (f-1) \cdot C^{down}) \leq 7f\beta^2 \cdot (T + C^{down})$$

4.4.1 Proof of Claim 4.8

To prove an upper bound of $6\beta^2) \cdot (\Delta^T + \Delta^{down})$ for Δ_k^{up} we consider two cases. One case arises if levels i_k and i_{k-1} are "close", the other one when they are far. The first case follows the proof pattern of Claim 4.7, which showed the bound for Δ_{k-1}^{up}. This is possible since the weight of edges contributing to Δ_k^{up} are within β^3 of the weight of edges contributing to Δ_{k-1}^{up} as i_k and i_{k-1} are "close". Thus, the same arguments as in Claim 4.7 can be used to show the desired bound for Δ_k^{up}. If, however, i_k and i_{k-1} are far, a new approach is needed, which we will explain in Case 2 below.

Case 1: $i_k \leq i_{k-1} + 3$.
Since $i_k \leq i_{k-1} + 3$, Claim 4.6 implies that:

$$\Delta_k^{up} \leq \beta^{i_k - 1} \leq \beta^{i_{k-1}} \cdot \beta^2 \tag{13}$$

In the proof of Claim 4.7, we derived Eq. (11), which gives us:

$$\Delta^T + \Delta^{down} \geq \beta^{i_{k-1}}/2 \tag{14}$$

Equations 13, 14 imply that $\Delta_k^{up} < 2\beta^2 \cdot (\Delta^T + \Delta^{down}) \leq 6\beta^2 \cdot (\Delta^T + \Delta^{down})$. This concludes the proof of Claim 4.8 for this case.

Case 2: $i_k > i_{k-1} + 3$.
The proof in this case consists of three steps:

(1) First we show that at any point in time the weight of any node is less than 2. This holds due to our subroutine FIX-DIRTY.
(2) Using this upper bound on W_v show that $W^+_{v \to (i_k - 3)}(t_k) > 1/(2\beta)$. It is interesting that we can show this bound, even though v is not at level $i_k - 3$ at time t_k. We show that claim by contradiction: If that weight were at most $1/(2\beta)$, then the weight of v on level $i_k - 1$ were less than $1/\beta$, since, as we show, the difference between $W_{v \to (i_k - 1)}(t_k)$ and $W^+_{v \to (i_k - 3)}(t_k)$ is at most $2/\beta^2$. But then the Up- jump would have placed v on level $i_k - 1$ and not on level i_k.
(3) Using the previous bound we show that $W^+_{v \to (i_k - 3)}$ increases from t_1 to t_k by at least $1/(2\beta) - 1/\beta^2$. Here we use the fact that v was Super- Clean at time t_1, which implies that $W^+_{v \to i_1}(t_1) \leq 1/\beta^2$ and the fact that $W^+_{v \to (i_k - 3)}(t_1) \leq W^+_{v \to i_1}(t_1)$. In previous situations there were two reasons why the weight of v increased over a certain time interval, one of them can be contributed to an increase in Δ^T and the other to an increase in Δ^{down}. Here, however, there are three such cases and the third one can neither be contributed to an increase in Δ^T nor to an increase in Δ^{down}. However, we can show that all events of the third type can lead to an increase of $W^+_{v \to (i_k - 3)}$ of at most $1/\beta^2$ during the time interval $[t_1, t_k]$. Thus, at

least $1/(2\beta) - 2/\beta^2$ of the increase of $W^+_{v\to(i_k-3)}$ can be contributed to either an increase in Δ^T or an increase in Δ^{down} and the same arguments at in 4.7 show the desired bound.

We start by noting that the weight of a node is always less than 2 at every time.

Claim 4.9 *We have:* $W_v(t) < 2$ *at every time* t.

Proof The crucial observation is that fixing an Up- Dirty node u never increases the weight of any node. Furthermore, a Down- Dirty node gets fixed only if no other node is Up- Dirty (see Fig. 1).

In the beginning of time-step $t = 0$, the input graph is empty, and we clearly have $W_v(t) = 0 < 1$. By induction, suppose that $W_v(t) < 1$ in the beginning of some time-step t. Now, during time-step t, the weight W_v can increase only if one of the following events occur:

- A hyperedge containing v gets inserted into the graph. This increases the value of W_v by at most one. Thus, we have $W_v(t+1) < 2$.
- We call the subroutine FIX-DOWN-DIRTY(u) for some node u. Note that fixing a Down- Dirty node u can increase the weight W_u by at most one, and hence this can increase the weight of a neighbour of u also by at most one. It again follows that $W_v(t+1) < 2$.

If $W_v(t+1) < 1$, then we are back in the same situation as in time-step t. Otherwise, if $1 \le W_v(t+1) < 2$, then v is Up- Dirty in the beginning of time-step $t+1$. In this case, no Down- Dirty node gets fixed (and no hyperedge gets inserted) until we ensure that W_v becomes smaller than one. Hence, the value of W_v always remains smaller than 2. □

Claim 4.10 *We have:* $W^+_{v\to(i_k-3)}(t_k) > 1/(2\beta)$.

Proof Suppose that the claim does not hold. Then we get:

$$\begin{aligned} W_{v\to(i_k-1)}(t_k) &\le W^+_{v\to(i_k-3)}(t_k) + \frac{W_{v\to(i_k-3)}(t_k) - W^+_{v\to(i_k-3)}(t_k)}{\beta^2} \\ &\le 1/(2\beta) + 2/\beta^2 \le 1/\beta \end{aligned} \tag{15}$$

The first inequality holds since the weights of the hyperedges $e \in E_v$ with $\ell_v(e) \le i_k - 3$ get scaled by at least a factor of $1/\beta^2$ when v moves from level $i_k - 3$ to $i_k - 1$, and the rest can only go down. The second inequality holds since $W_{v\to(i_k-3)}(t_k) \le W_{v\to i_{k-1}}(t_k) < 2$ by Claim 4.9 and the assumption $W^+_{v\to(i_k-3)}(t_k) \le 1/(2\beta)$. The last inequality holds since $\beta = 6$ by Eq. (1).

Note, however, that $W_{v\to(i_k-1)}(t_k) > 1/\beta$, since node v does not stop at level $i_k - 1$ while making the Up- Jump at time t_k. Thus, assuming that the claim does not hold leads to a contradiction. □

Claim 4.10 states that $W^+_{v\to(i_k-3)}(t_k) > 1/(2\beta)$. Since $i_{k-1} < i_k - 3$, Eq. (8) implies that $W^+_{v\to(i_k-3)}(t_1) \le W^+_{v\to i_{k-1}}(t_1) \le W^+_{v\to i_1}(t_1) \le 1/\beta^2$. Thus during the

time-interval $[t_1, t_k]$, the value of $W^+_{v\to(i_k-3)}$ increases by at least $1/(2\beta) - 1/\beta^2$. This increase can occur in three ways:

1. a hyperedge $e \in E_v$ is inserted with $\ell_v(e) > i_k - 3$ before the UP- JUMP at time t_k (which contributes to Δ^T),
2. some hyperedge $e \in E_v$ gains weight due to a DOWN- JUMP of some node (say) $u \in e, u \neq v$, and $\ell_v(e) > i_k - 3$ after the DOWN- JUMP (which contributes to Δ^{down}), and
3. some hyperedge $e \in E_v$ had $\ell_v(e) \leq i_k - 3$ at t_1 (i.e. does *not* contribute to $W^+_{v\to(i_k-3)}(t_1)$) but $\ell_v(e) > i_k - 3$ at t_k (i.e. contributes to $W^+_{v\to(i_k-3)}(t_k)$).

Note that the total weight of the hyperedges of type (3) at time t_1 incident on v at level $i_k - 3$ is at most $1/\beta$; this follows from (8). Therefore, when $\ell_v(e)$ for such an edge e raises to $> i_k - 3$, the weight decreases by at least a $1/\beta$ factor. Hence the total increase in $W^+_{v\to(i_k-3)}$ due to type (3) hyperedges is at most $1/\beta^2$, and the weight increase of at least $1/(2\beta) - 2/\beta^2$ must come from hyperedges of type (1) and type (2). However each such hyperedge e can contribute at most $\beta^{-(i_k-2)}$ to the weight (since $\ell_v(e) > i_k - 3$). And therefore, we get (recall that $\beta = 6$ by Eq. (1)):

$$\left(\Delta^T + \Delta^{down}\right) \cdot \beta^{-(i_k-2)} \geq \frac{1}{2\beta} - \frac{2}{\beta^2}$$
$$= \frac{1}{6\beta} \quad \text{implying} \quad \left(\Delta^T + \Delta^{down}\right) \geq \frac{\beta^{i_k-1}}{6\beta^2}$$

Claim 4.6 gives $\Delta^{up}_k \leq \beta^{i_k-1}$, and therefore we get:

$$\Delta^{up}_k \leq 6\beta^2 \left(\Delta^T + \Delta^{down}\right) \tag{16}$$

This concludes the proof of Claim 4.8 for this case.

4.5 Proof of Lemma 4.1

For technical reasons, we assume that we end with the empty graph as well. This is without loss of generality due to the following reason. Suppose we made T updates and the current graph is G. At this point, the graph has $T' \leq T$ edges. Suppose the time taken by our algorithm till now is T_1. Now delete all the T' edges, and let the time taken by our algorithm to take care of these T' updates be T_2. If $T_1 + T_2 = \Theta(f^2(T + T')) = \Theta(f^2 T)$, then $T_1 = \Theta(T)$ as well. Therefore, we assume that the process ends with an empty graph.

When a hyperedge e is inserted into or deleted from the graph, we take $O(f)$ time to update the relevant data structures for its f endpoints. The rest of the time is spent in implementing the WHILE loop in Fig. 1. We take care of the two subroutines separately.

Case 1. The subroutine FIX-DOWN-DIRTY(v) is called which moves the node v from level i to level $j < i$ (say). We need to account for the time to find the relevant index j and the time taken to update the relevant data structures. By Lemma 2.8, the

time taken for the latter is proportional $\Theta(f \cdot \Delta C^{down})$, where ΔC^{down} denotes the increase in the value of the counter C^{down} due to this event. Further, the value of C^{up} remains unchanged. For finding the index $j < i$, it suffices to focus on the edges $E_{v,i} = \{e \in E_v : \ell_v(e) \leq i\}$ since these are the only edges that can potentially change weight as v goes down. Therefore, this takes time $\Theta(|\{e \in E_v : \ell_v(e) \leq i\}|)$. Since each of these edges had $w(e) = \beta^{-i}$ and since $W_v \leq \frac{1}{\alpha\beta^2}$ before the FIX-DOWN-DIRTY(v) call, we have $|\{e \in E_v : \ell_v(e) \leq i\}| \leq \beta^{i-2}/\alpha$ which is precisely ΔI^{down}, where ΔI^{down} is the increase in the value of the counter I^{down} due to this event. Therefore, the time taken to find the index j is $\Theta(\Delta I^{down})$.

Case 2. The subroutine FIX-UP-DIRTY(v) is called which moves the node v from level i to level $j > i$, say. Once again, we need to account for the time to find the relevant index j and the time taken to update the relevant data structures, and once again by Lemma 2.7 the time taken for the latter is $\Theta(f \cdot \Delta C^{up})$. Further, the value of C^{down} remains unchanged. We now account for the time taken to find the index j.

Claim 4.11 *j can be found in time $\Theta(j - i)$.*

Proof To see this note that for $k \geq i$,

$$W_{v\to k}(t) = \sum_{\ell \geq k} \sum_{e \in E_{v,\ell}} w(e) + \sum_{\ell < k} \frac{1}{\beta^{k-\ell}} \sum_{e \in E_{v,\ell}} w(e)$$

since (a) edges not incident on v are immaterial, (b) the edges incident on v whose levels are already $\geq k$ do not change their weight, and (c) edges whose levels are $\ell < k$ have their weight go from $\beta^{-\ell}$ to β^{-k}. The above implies that for $k \geq i$,

$$\begin{aligned} W_{v\to(k+1)}(t) &= W_{v\to k}(t) - \left(1 - \frac{1}{\beta}\right) \sum_{e \in E_{v,k}} w(e) \\ &= W_{v\to k}(t) - \left(1 - \frac{1}{\beta}\right) |E_{v,k}| \cdot \beta^{-k} \end{aligned}$$

That is, $W_{v\to(k+1)}$ can be evaluated from $W_{v\to k}(t)$ in $\Theta(1)$ time since we store $|E_{v,k}|$ in our data structure. The claim follows. □

Note that in Claim 4.11 the time taken to find the index j can be as large as $\Theta(\log n)$. To account for this time, we again fix a vertex v and a phase Φ_v where the level of v changes from i_1 to say i_k. The total time for finding indices is $\Theta(i_k - i_1)$. After this, there must be a DOWN-JUMP due to a call to FIX-DOWN-DIRTY(v) since the final graph is empty. Thus, we can charge the time taken in finding indices in this phase Φ_v to ΔI^{down} in the FIX-DOWN-DIRTY(v) call right at the end of this phase. We can do so since $\Delta I^{down} = \beta^{i_k-2}/\alpha = \frac{1}{f^2}\Theta(i_k)$ since $\beta = \Theta(1)$ and $\alpha = \Theta(f^2)$ by (1). Therefore, the total time taken to find indices in the FIX-UP-DIRTY(v) calls in all is at most $f^2 I^{down}$.

To summarize, the time taken to initialize and update the data structures is at most $\Theta\left(f \cdot \left(C^{up} + C^{down} + T\right)\right)$, and the total time taken to find indices is at most $\Theta(f^2 \cdot I^{down})$. This proves Lemma 4.1.

A Duality Between Maximum Fractional Matching and Minimum Vertex Cover

Let $y_v \in \{0, 1\}$ be an indicator variable which denotes if the node $v \in V$ is picked in the vertex cover. The LP-relaxation for minimum vertex cover is as follows.

$$\begin{aligned} \text{Minimize} \quad & \sum_{v \in V} x_v \\ \text{such that:} \quad & \sum_{v \in e} x_v \geq 1 \text{ for all } e \in E. \\ & x_v \geq 0 \text{ for all } v \in V. \end{aligned}$$

In the above LP, the first constraint encodes the fact that for every hyperedge we have to pick at least one node incident on it. The dual of the above LP is the following.

$$\begin{aligned} \text{Maximize} \quad & \sum_{e \in E} y_e \\ \text{such that:} \quad & \sum_{e \in E : v \in e} y_e \leq 1 \text{ for all } v \in V. \\ & y_e \geq 0 \text{ for all } e \in E. \end{aligned}$$

Note that the dual LP precisely encodes the maximum fractional problem: We have to assign a weight $y_e \geq 0$ to every hyperedge $e \in E$ in such a way that the total weight received by any node v from all its incident hyperedges is at most 1. The goal is to maximize the total weight assigned to all the hyperedges. Hence, the size of the maximum fractional matching in a hypergraph is at most the size of its minimum vertex cover.

References

1. Abboud, A., Williams, V.V.: Popular conjectures imply strong lower bounds for dynamic problems. In: Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS) (2014)
2. Arar, M., Chechik, S., Cohen, S., Stein, C., Wajc, D.: Dynamic matching: reducing integral algorithms to approximately-maximal fractional algorithms. CoRR (2017). arXiv:1711.06625
3. Baswana, S., Gupta, M., Sen, S.: Fully dynamic maximal matching in $O(\log n)$ update time. In: Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS) (2011)
4. Bernstein, A., Stein, C.: Faster fully dynamic matchings with small approximation ratios. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) (2016)

5. Bhattacharya, S., Chakrabarty, D., Henzinger, M.: Deterministic dynamic matching in $o(1)$ update time. In: Proceedings of the MPS Symposium on Integer Programming and Combonatorial Optimization (IPCO) (2017)
6. Bhattacharya, S., Henzinger, M., Italiano, G.F.: Design of dynamic algorithms via primal-dual method. In: Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP) (2015)
7. Bhattacharya, S., Henzinger, M., Italiano, G.F.: Deterministic fully dynamic data structures for vertex cover and matching. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) (2015)
8. Bhattacharya, S., Henzinger, M., Nanongkai, D.: New deterministic approximation algorithms for fully dynamic matching. In: Proceedings of the ACM Symposium on Theory of Computing (STOC) (2016)
9. Bhattacharya, S., Henzinger, M., Nanongkai, D.: Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) (2017)
10. Bhattacharya, S., Kulkarni, J.: Deterministically maintaining a $(2+\varepsilon)$-approximate minimum vertex cover in $o(1/\varepsilon^2)$ amortized update time. In: SODA, pp. 1872–1885 (2019)
11. Bosek, B., Leniowski, D., Sankowski, P., Zych, A.: Online bipartite matching in offline time. In: Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS) (2014)
12. Charikar, M., Solomon, S.: Fully dynamic almost-maximal matching: Breaking the polynomial barrier for worst-case time bounds. CoRR (2017). arXiv:1711.06883
13. Gupta, A., Krishnaswamy, R., Kumar, A., Panigrahi, D.: Online and dynamic algorithms for set cover. In: Proceedings of the ACM Symposium on Theory of Computing (STOC) (2017)
14. Gupta, M., Peng, R.: Fully dynamic $(1+\varepsilon)$-approximate matchings. In: Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS) (2013)
15. Henzinger, M., Krinninger, S., Nanongkai, D., Saranurak, T.: Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In: Proceedings of the ACM Symposium on Theory of Computing (STOC) (2015)
16. Henzinger, M.R., Fredman, M.L.: Lower bounds for fully dynamic connectivity problems in graphs. Algorithmica **22**(3), 351–362 (1998)
17. Neiman, O., Solomon, S.: Simple deterministic algorithms for fully dynamic maximal matching. In: Proceedings of the ACM Symposium on Theory of Computing (STOC) (2013)
18. Onak, K., Rubinfeld, R.: Maintaining a large matching and a small vertex cover. In: Proceedings of the ACM Symposium on Theory of Computing (STOC) (2010)
19. Parter, M., Peleg, D., Solomon, S.: Local-on-average distributed tasks. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) (2016)
20. Patrascu, M.: Lower bounds for dynamic connectivity. In: Encyclopedia of Algorithms, pp. 1162–1167. Springer (2016)
21. Sankowski, P.: Faster dynamic matchings and vertex connectivity. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA) (2007)
22. Solomon, S.: Fully dynamic maximal matching in constant update time. In: Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS) (2016)

Affiliations

Sayan Bhattacharya[1] · Deeparnab Chakrabarty[2] · Monika Henzinger[3]

Deeparnab Chakrabarty
deeparnab.chakrabarty@dartmouth.edu

Monika Henzinger
monika.henzinger@univie.ac.at

[1] University of Warwick, Coventry, UK

[2] Department of Computer Science, Dartmouth College, 6211 Sudikoff Lab, Hanover, NH 03755, USA

[3] University of Vienna, Wien, Austria