# Faster Fully Dynamic Matchings with Small Approximation Ratios

Aaron Bernstein* Cliff Stein†

**Abstract**

Maximum cardinality matching is a fundamental algorithmic problem with many algorithms and applications. The fully dynamic version, in which edges are inserted and deleted over time has also been the subject of much attention. Existing algorithms for dynamic matching (in general $n$-vertex $m$-edge graphs) fall into two groups: there are fast (mostly randomized) algorithms that achieve a 2-approximation or worse, and there are slow algorithms with $\Omega(\sqrt{m})$ update time that achieve a better-than-2 approximation. Thus the obvious question is whether we can design an algorithm that achieves a tradeoff between these two: a $o(\sqrt{m})$ update time and a better-than-2 approximation simultaneously. We answer this question in the affirmative. Previously, such bounds were only known for the special case of bipartite graphs.

Our main result is a fully dynamic *deterministic* algorithm that maintains a $(3/2+\epsilon)$-approximation in amortized update time $O(m^{1/4}\epsilon^{-2.5})$. In addition to achieving the trade-off described above, our algorithm manages to be polynomially faster than all existing deterministic algorithms (excluding an existing $\log n$-approximation of Onak and Rubinfeld), while still maintaining a better-than-2 approximation.

We also give stronger results for graphs whose arboricity is at most $\alpha$. We show how to maintain a $(1+\epsilon)$-approximate *fractional* matching or a $(3/2+\epsilon)$-approximate *integral* matching in worst-case time $O(\alpha(\alpha+\log n))$ for constant $\epsilon$. When the arboricity is constant, this bound is $O(\log n)$ and when the arboricity is polylogarithmic the update time is also polylogarithmic. Previous results for small arboricity non-bipartite graphs could only maintain a maximal matching (2-approximation).

We maintain the approximate matching without explicitly using augmenting paths. We define an intermediate graph, called an EDCS and show that the EDCS $H$ contains a large matching, and show how to maintain an EDCS in $G$. The EDCS was used in previous works on bipartite graphs, however the details and proofs are completely different in general graphs. The algorithm for bipartite graphs relies on ideas from flows and cuts to non-constructively prove the existence of a good matching in $H$, but these ideas do not seem to extend to non-bipartite graphs. In this paper we instead *explicitly construct* a large fractional matching in $H$. In some cases we can guarantee that this fractional matching is $\gamma$-restricted, which means that it only uses values either in the range $[0,\gamma]$ or 1. We then combine this matching with a new structural property of maximum matchings in non-bipartite graphs, which is analogous to the cut induced by maximum matchings in bipartite graphs.

*Department of Computer Science, Columbia University. Supported in part by an NSF Graduate Fellowship and a Simons Foundation Graduate Fellowship. bernstei@gmail.com

†Department of IEOR and Computer Science, Columbia University. Supported in part by NSF grants CCF-1349602 and CCF-1421161. cliff@ieor.columbia.edu

## 1 Introduction

Matching is a classical problem, with a variety of applications (e.g. [14, 7]), and many algorithms. In this paper, we study a variant of the maximum cardinality matching problem in which the goal is to maintain a near-maximum matching in a dynamic graph that is changing over time. More formally, we are working in the *fully dynamic* setting, in which we are given a graph $G$ and a sequence of edge insertions and deletions. Our goal is to design an algorithm that always maintains a near-maximum matching for the current graph. With each insertion or deletion, we could recompute a matching from scratch in $O(\sqrt{n}m)$ time [15, 21], but it is more efficient to take advantage of the fact that only one edge has changed. For example, it would suffice to compute at most one augmenting path to update the matching.

Some of our results are for *small-arboricity* graphs, which we define here. The *arboricity* of a graph, denoted by $\alpha(G)$ is $\max_J \frac{|E(J)|}{V(J)-1}$ where $J=(V(J),E(J))$ is any subgraph of $G$ induced by at least two vertices. Many classes of graphs in practice have constant arboricity, including planar graphs, graphs with bounded genus and graphs with bounded tree width. Every graph has arboricity $O(\sqrt{m})$.

**1.1 Previous Work** There is a great deal of work on matchings. We describe previous work on dynamic matching in detail below, but first we briefly mention the related problems of finding approximate and online matchings. Duan and Pettie showed how to find a $(1+\epsilon)$-approximate weighted matching in nearly linear time [6]; their paper also contains an excellent summary of the history of matching algorithms. There are many papers on "online matching" (e.g. [14, 7]), both exact and approximate. This model also involves edges being added to the graph, but is in other ways very different from the dynamic model: in online matching there are no deletions, the matching cannot be altered, and the quality of the algorithm is judged by its competitiveness to the optimal offline algorithm. A related measure is the number of changes needed to maintain a matching [5, 8, 4].

Fully dynamic matching algorithms can be classified by update time, approximation ratio, whether they are

randomized or deterministic and whether they have a worst-case or amortized update time. The distinction between deterministic and randomized is particularly important in the dynamic setting as all of the existing randomized algorithms for dynamic matching (as well as for many other problems on dynamic graphs) assume a weaker model. The key idea behind the randomized matching algorithms is that if the algorithm chooses a *random* matching in a dense graph then the probability of an update deleting an edge in the matching is very small. This approach, however, only functions under the assumption that the adversary is *non-adaptive*; that is, the update sequence must be fixed in advance, or put otherwise, it must be completely independent of matching maintained by the algorithm. This extra assumption on the model makes the existing randomized algorithms inadequate in certain settings, such as those in which the dynamic matching algorithm is a black-box subroutine inside a larger data structure (unless of course one can separately ensure that the subroutine is used non-adaptively).

For maintaining an *exact* maximum matching in the fully dynamic setting, the best known update time is $O(n^{1.495})$ (Sankowski [19]), which in dense graphs is much faster than reconstructing the matching from scratch. Most work on dynamic matching shows how to obtain faster update times than $O(n^{1.495})$ if we allow approximation. One relevant line of work focuses on approximation ratios 2 or worse. 2 is a significant "barrier" because a straightforward greedy algorithm maintains a *maximal* matching (which is a 2-approximation to the maximum matching) in $O(n)$ time per update, and potentially using only local operations. Ivkovic and Lloyd [11] showed how to improve the update time to $O((m+n)^{\sqrt{2}/2})$. Onak and Rubinfeld [18] were the first to achieve truly fast update times, with a randomized algorithm that maintains a $O(1)$-approximate matching in amortized update time $O(\log^2 n)$ time (with high probability). Baswana *et al.*[1] gave a faster randomized algorithm that maintains a maximal matching (2-approximation) in amortized update $O(\log n)$ time per update. These two algorithms are extremely fast, but are inherently randomized. In addition, their techniques focus on local changes, and are unlikely to break through the barrier of a 2-approximation. Recently, Bhattacharya, Henzinger, and Italiano [3] presented a *deterministic* algorithm with worst-case update time $O(m^{1/3}\epsilon^{-2})$ that maintains a $(4+\epsilon)$ approximation; this can be improved to $(3+\epsilon)$ at the cost of introducing amortization. In the same paper they present a deterministic algorithm with amortized update time only $O(\epsilon^{-2}\log n)$ that maintains a $(2+\epsilon)$ *fractional* matching. Finally, Neiman and Solomon [17] showed that, in graphs of constant arboricity, we can maintain a maximal (so 2-approximate) matching in amortized time $O(\log n/\log\log n$, or *worst-case* time $O(\log n)$ using. a recent dynamic orientation algorithm of Kopelowitz *et al.*[12].

Another line of work gives approximation ratios better than 2, but with running times that are $\Omega(m^{1/2})$. Neiman and Solomon [17] gave a *deterministic, worst-case* algorithm for maintaining a 3/2-approximate matching, with an update time of $O(\sqrt{m})$. Gupta and Peng [9] later improved upon the approximation, presenting a deterministic algorithm that maintains a $(1+\epsilon)$-approximate matching in worst-case update time $O(\sqrt{m}\epsilon^{-2})$ (the same paper achieves an analogous result for maintaining a near-maximum *weighted* matching in weighted graphs). The two deterministic algorithms are strongly tethered to the $\sqrt{m}$ bound and do not seem to contain any techniques for breaking past it.

For the special case of *bipartite graphs*, Bernstein and Stein [2] recently gave algorithms that maintain a $(3/2+\epsilon)$-approximation in *worst-case* update time $O(m^{1/4}\epsilon^{-2.5})$ and a $(1+\epsilon)$ approximation in worst-case time $O(\alpha(\alpha+\log n))$ for constant $\epsilon$ in graphs with arboricity $\alpha$.

Very recently there have been some conditional lower bounds for dynamic approximate matching. Kopelowitz *et al.*[13] show that assuming 3-sum hardness any algorithm that maintains a matching in which all augmenting paths have length at least 6 requires an update time of $\Omega(m^{1/3}-\zeta)$ for any fixed $\zeta>0$. Henzinger *et al.* [10] show that such an algorithm requires $\Omega(m^{1/2}-\zeta)$ time if one assumes the Online Matrix-Vector conjecture.

Despite all this work, the problem of designing an algorithm that simultaneously achieves an update time of $o(\sqrt{m})$ and an approximation better than 2 remained open for general graphs.

### 1.2 Results

THEOREM 1.1. *Let $G$ be a graph subject to a series of edge insertions and deletions, and let $\epsilon<1$. We can maintain a $(3/2+\epsilon)$-approximate matching in $G$ in deterministic amortized update time $O(m^{1/4}\epsilon^{-2.5})$.*

Even allowing randomization, this result is the first algorithm in general (non-bipartite) graphs to achieve $o(\sqrt{m})$ update time and a better-than-2 approximation. The algorithm is also faster than all previous deterministic $O(1)$-approximation algorithms in general graphs. Also, since $m^{1/4}=O(\sqrt{n})$, our algorithm is the first to achieve a better-than-2 approximation in time strictly sublinear in the number of nodes.

For small arboricity graphs we also show how to break through the maximal matching (2-approximation) barrier and achieve the following:

THEOREM 1.2. *Let $G$ be a graph subject to a series of edge insertions and deletions, and let $\epsilon < 1$. Say that at all times $G$ has arboricity at most $\alpha$. Then, we can maintain a $(3/2 + \epsilon)$-approximate matching in $G$ in deterministic amortized update time $O(\alpha(\alpha + \log n + \epsilon^{-2}) + \epsilon^{-6})$. For constant $\alpha$ and $\epsilon$ the update time is $O(\log n)$, and for $\alpha$ and $\epsilon$ polylogarithmic the update time is polylogarithmic.*

THEOREM 1.3. *Let $G$ be a graph subject to a series of edge insertions and deletions, and let $\epsilon < 1$. Say that at all times $G$ has arboricity at most $\alpha$. Then, we can maintain a $(1 + \epsilon)$-approximate* **fractional** *matching in $G$ in deterministic amortized update time $O(\alpha(\alpha + \log n + \epsilon^{-4}) + \epsilon^{-6})$.*

Our algorithms introduce the (as far as we can tell) new notion of a $\gamma$-restricted fractional matching, and prove an accompanying theorem which we believe might be of independent interest.

DEFINITION 1. *A $\gamma$-restricted fractional matching is a fractional matching assigning values to the edges such that for all edges $e$,* value$(e)$ *is either* 1 *or in the interval* $[0, \gamma]$.

It is well known that there always exists a maximum fractional matching with values 0, 1/2 or 1 [20] (i.e. a 1/2-restricted fractional matching). Moreover, the support of this matching contains an integral matching of at least 2/3 the value of the fractional matching. We prove a generalization of this second fact for smaller $\gamma$.

THEOREM 1.4. *Given a graph $G$, let $M_f$ be a $\gamma$-restricted fractional matching, and let $M$ be a maximum integral matching in the graph formed by edges in the support of $M_f$. Then $|M| \geq \text{value}(M_f)\frac{1}{\gamma+1}$.*

Observe that this bound is tight when $\gamma = 1/c$ for some even $c$. Consider a clique on $c + 1$ vertices. The fractional solution places value $1/c$ on each edge and has total weight $\binom{c+1}{2}/c = (c + 1)/2$. But the best integral matching has $c/2$ edges.

**Comparison to Results on Bipartite Graphs** The *statements* (but not the proofs) of many of the theorems in this paper directly correspond to theorems in Bernstein and Stein's paper on bipartite graphs [2]. However, the shift to non-bipartite graphs requires *completely* different proofs for all the theorems involved, and different algorithms. As an analogy, consider matching in a static graph. In both bipartite and non-bipartite graphs, an algorithm that repeatedly augments along a maximal set of shortest paths yields a running time of $O(\sqrt{n}m)$, but the details (e.g. blossoms, finding paths, data structures), analysis, and ease of understanding (teachable in class vs. years to provide a complete proof) are completely different. In dynamic graphs, we have a similar situation.

We provide a brief discussion of some of the differences here. In both bipartite and non-bipartite graphs, we use an intermediate graph $H$, called an edge degree constrained subgraph (EDCS), show that the EDCS $H$ contains a large matching, and show how to maintain an EDCS in $G$. However, the proofs of both these steps are completely different in bipartite and non-bipartite graphs. In bipartite graphs, we rely on ideas from flows and cuts to non-constructively prove the existence of a good matching in $H$, but these ideas do not seem to extend to non-bipartite graphs. In this paper we instead *explicitly construct* a large fractional matching in $H$. In some cases we can guarantee that this fractional matching is $\gamma$-restricted, which combined with our Theorem 1.4 guarantees a large integral matching. The proof of Theorem 1.4 relies on a new structural property of maximum matchings in non-bipartite graphs, which is analogous to the cut induced by maximum matchings in bipartite graphs.

Finally, the approach to efficiently maintaining the edge degree constrained subgraph $H$ is different. Bernstein and Stein [2] showed that in bipartite graphs the EDCS could be maintained after an update to $G$ by fixing all the edges on a specific (short) path. In general graphs, the existence of blossoms (among other things) makes the maintenance more difficult. We use an approach that does not explicitly use augmenting paths (which would have to deal with blossoms in some way), instead switching to a simpler, amortized approach. The resulting analysis is much more general, and shows that if we simply fix up violating edges in the EDCS in an *arbitrary* order, then on average we will terminate quickly.

## 2 Preliminaries

Let $G = (V, E)$ be an undirected, unweighted graph where $|V| = n$ and $|E| = m$. We will often deal with graphs other than $G$, so all of our notation will be explicit about the graph in question. We define $d_G(v)$ to be the degree of a vertex $v$ in $G$; if the graph in question is weighted, then $d_G(v)$ is the sum of the weights of all incident edges. We define *edge degree* as $\delta(u, v) = d(u) + d(v)$. If $H$ is a subgraph of $G$, we say that an edge in $G$ is *used* if it is also in $H$, and *unused* if it is not in $H$. Throughout this paper we will only be dealing with subgraphs $H$ that contain the full vertex

set of $G$, so we will use the notion of a subgraph and of a subset of edges of $G$ interchangeably. Recall the set difference operator $Z_1 \setminus Z_2 = \{x \in Z_1 | x \notin Z_2\}$.

A matching in a graph $G$ is a set of disjoint edges in $G$. We let $\mu(G)$ denote the size of the maximum matching in $G$. A vertex is called *matched* if it is incident to one of the edges in the matching, and *free* or *unmatched* otherwise. A fractional matching is an assignment of values to edges such that the total value of edges incident to any vertex is at most 1. We let $\text{VAL}(u, v)$ denote the value of edge $(u, v)$ in a fractional matching. Given a vertex $v$, $\text{VAL}(v)$ will denote the sum of the values of all edges incident to $v$. We say a fractional matching is *feasible* if $\text{VAL}(v) \leq 1 \quad \forall v \in V$. Given some fractional matching $M_f$, $\text{VAL}(M_f)$ will denote the sum of all edge values in $M_f$. We let $\mu_f(G)$ denote the size of the maximum-valued fractional matching in $G$. We will often compute the value of a fractional matching by summing over vertices. To avoid double counting edges, we let each endpoint of an edge account for a fraction of that edge. More formally, given a graph $G$, we define an *accounting* of the edges to be a function that assigns to each edge $(x, y)$ two values $a_x(x, y)$ and $a_y(x, y)$ such that $a_x(x, y) + a_y(x, y) \leq 1$. Given a fractional matching $M_f$ and some accounting of the edges, we define the *profit* of $x$, $\rho(x)$, to be $\rho(x) = \sum_{(x,y)\in E} a_x(x, y)\text{VAL}(x, y)$.

OBSERVATION 1. *Given some fractional matching $M_f$ in $G$, and some accounting of the edges of $G$, we always have $\text{VAL}(M_f) \leq \sum_{x\in V} \rho(x)$. This inequality holds with equality when, for each edge $(x, y)$, we have $a_x(x, y) + a_y(x, y) = 1$.*

We now state a lemma from [2] which is an easy corollary of a result of [9].

LEMMA 2.1. ([9]) *If a dynamic graph $G$ has maximum degree $B$ at all times, then we can maintain a $(1 + \epsilon)$-approx. matching under insertions and deletions in worst-case update time $O(B\epsilon^{-2})$ per update.*

**Orientations** An orientation $G'$ of an undirected graph $G$ is an assignment of a direction to each edge in $E$. Given an orientation of edge $(u, v)$ from $u$ to $v$, we will say that $u$ *owns* edge $(u, v)$ and will define the *load* of a vertex $u$ to be the number of edges owned by $u$. Orientations of small max load are closely linked to arboricity: every graph with arboricity $\alpha$ has an orientation with max load $O(\alpha)$ [16]. Our algorithms will maintain an orientation of the *dynamic* graph $G$ using the algorithms referred to in the theorems below. The first result is due to Bernstein and Stein [2], the second to Kopelowitz *et al.*[12].

THEOREM 2.1. *[2] In a graph $G$, we can maintain an orientation, under insertions and deletions, with the following properties: the max load at all times is at most $3\sqrt{m}$, the worst-case number of edge reorientations per insert/deletion in $G$ is $O(1)$, and the worst-case time spent per insertion/deletion in $G$ is $O(1)$.*

THEOREM 2.2. ([12]) *Let $G$ be a graph that always has arboricity at most $\alpha$. We maintain an orientation, under edge insertions and deletions, with the following properties: the maximum load at all times is $O(\alpha + \log n)$, the worst-case number of edge reorientations per insertion/deletion is also $O((\alpha + \log n))$, and the worst-case time to process an insertion/deletion in $G$ is $O(\alpha(\alpha + \log n))$.*

## 3 The Framework

DEFINITION 2. *An unweighted edge degree constrained subgraph, denoted $\text{EDCS}(G, \beta, \beta^-)$ is a subset of the edges $H \subseteq E$ (we will also refer to it as a subgraph) with the following properties:*

*(P1) if $(u, v)$ is used (i.e. $(u, v) \in H$) then $d_H(u) + d_H(v) \leq \beta$ ,*

*(P2) if $(u, v)$ is unused (i.e. $(u, v) \in G \setminus H$) then $d_H(u) + d_H(v) \geq \beta^-$.*

We also define a similar subgraph where edges in $H$ have positive integer weights, effectively allowing them to be used more than once. Note that $d_H(v)$ now refers to the sum of the weights of $v$'s incident edges.

DEFINITION 3. *A weighted edge degree constrained subgraph(*EDCS*) $(G, \beta, \beta^-)$ is a subset of the edges $H \subseteq E$ with positive integer weights that has the following properties:*

*(P1) if $(u, v)$ is used then $d_H(u) + d_H(v) \leq \beta$,*

*(P2) for* all *edges $(u, v)$, we have $d_H(u) + d_H(v) \geq \beta^-$.*

Below is an outline of how our algorithm processes an edge insertion/deletion in $G$:

1. Update the small-max-load edge orientation using either Theorem 2.1 or Theorem 2.2 .
2. Update the subgraph $H$ so it remains a valid edge degree constrained subgraph of the changed graph $G$. This step uses the orientation from step 1 for efficiency. (See Section 6.)
3. Update the $(1 + \epsilon)$-approx. matching in $H$ with respect to the changes in $H$ from step 2. (Lemma 2.1.)

The maintained $(1+\epsilon)$-approx matching of $H$ (step 3) is also our final matching in $G$; much of the paper is devoted to showing that because $H$ is an EDCS, $\mu(H)$ is not too far from $\mu(G)$.

Because much of our algorithm runs on a dynamic subgraph of $G$ we need the following definition: Let $H$ be a subgraph of a dynamic graph $G$, and let $A$ be an algorithm that modifies the edges of $H$ as $G$ changes; then, we say that $A$ has an *amortized update ratio* of $r$ if for any large enough sequence $S$ of edge changes (insertions or deletions) to $G$, the algorithm makes at most $r|S|$ edge changes to $H$.

We can now state the main theorems of the paper. We present general and small arboricity graphs separately, but the basic framework described above remains the same in both cases. In all the theorems below, the parameter $\epsilon > 0$ is chosen to obtain a desired approximation ratio (either $(1+\epsilon)$ or $(3/2+\epsilon)$).

**3.1 General Graphs** For the sake of intuition, in the two theorems below, think of $\beta$ as quite large (roughly $m^{1/4}$). Also recall that $\mu(H)$ denotes the size of the maximum matching in $H$.

THEOREM 3.1. *Let $G$ be a graph and let $\lambda = \epsilon/6$. Let $H$ be an unweighted* EDCS *of $G$ with $\beta^- = \beta(1-\lambda)$, where $\beta \geq 32\lambda^{-3}$ is a parameter we will choose later. Then $\mu(H) \geq (2/3 - 2\epsilon)\mu(G)$.*

THEOREM 3.2. *Let $G$ be a graph. Let $H$ be an unweighted* EDCS *of $G$ with $\beta^- = \beta(1-\lambda)$, with $\beta \geq 36\lambda^{-1}$. There is an algorithm that maintains $H$ over updates in $G$ (i.e. maintains $H$ as a valid* EDCS*) with the following properties:*

- *The algorithm has amortized update time $O\left(\frac{\sqrt{m}}{\lambda^2\beta}\right)$.*
- *The amortized update ratio of the algorithm is $O(1/\lambda)$.*

**Proof of Theorem 1.1** We use the algorithm outline presented at the beginning of Section 3. We set $H$ to be an unweighted EDCS$(G, \beta, \beta(1-\lambda))$ with $\lambda = \epsilon/6$ and $\beta = m^{1/4}\epsilon^{1/2}$. By Theorem 3.2 we can maintain $H$ in amortized update time $O(\frac{\sqrt{m}}{\lambda^2\beta}) = O(m^{1/4}\epsilon^{-2.5})$. The update-ratio is $O(\lambda^{-1}) = O(\epsilon^{-1})$. Since degrees in $H$ are clearly bounded by $\beta$, by Lemma 2.1 we can maintain a $(1+\epsilon)$-approx. matching in $H$ in time $O(\beta\epsilon^{-2})$; multiplying by the update ratio of maintaining $H$ in $G$, we need $O(\beta\epsilon^{-3}) = O(m^{1/4}\epsilon^{-2.5})$ time to maintain the matching per change in $G$. By Theorem 3.1, $\mu(H)$ is a $(3/2+2\epsilon)$-approximation to $\mu(G)$, so our matching is a $(3/2+2\epsilon)(1+\epsilon) = (3/2+O(\epsilon))$-approx. matching in $G$. □

**3.2 Small Arboricity Graphs** For the sake of intuition, in the two theorems below think of $\beta$ as quite small, roughly $O(\epsilon^{-2})$.

THEOREM 3.3. *Let $G$ be a graph, and let $\beta \geq 25\epsilon^{-2}$. Let $H$ be a weighted* EDCS *with $\beta^- = \beta(1-\lambda)$, where $\lambda = \epsilon^2/25$. Consider the fractional matching $M_f^H$ in $H$, with $\text{VAL}(u,v) = 1/\max\{d_H(u), d_H(v)\}$. Then $M_f^H$ is a feasible fractional matching, and $\text{VAL}(M_f^H) \geq \mu_f(G)(1-\epsilon)$.*

THEOREM 3.4. *Let $G$ be a dynamic graph that at all times has arboricity $\leq \alpha$. Let $H$ be a weighted* EDCS$(G, \beta, \beta(1-\lambda))$ *with $\beta \geq 4\lambda^{-1}$. There is an algorithm that maintains $H$ over updates in $G$ with the following properties:*

- *The algorithm has amortized update time $O(\alpha(\alpha + \log n + \beta\lambda^{-1}))$.*
- *The amortized update ratio of the algorithm is $O(\beta\lambda^{-1})$.*

*If $H$ is instead an* unweighted EDCS$(G, \beta, \beta(1-\lambda))$*, the algorithm has amortized update time $O(\alpha(\alpha+\log n+\lambda^{-1}))$ and amortized update ratio $O(\lambda^{-1})$.*

**Proof of Theorem 1.2** The proof is essentially identical to that Theorem 1.1; for the transition subgraph $H$, we use an (unweighted) EDCS$(G, \beta, \beta(1-\lambda))$ with $\beta = 100\epsilon^{-2}$ and $\lambda = \epsilon^2/25$ (so $\beta(1-\lambda) = \beta - 4$). □

**Proof of Theorem 1.3** Let $H$ be a weighted EDCS$(G, \beta, \beta(1-\lambda))$ with $\beta = 100\epsilon^{-2}$ and $\lambda = \epsilon^2/25$. By Theorem 3.4 we can maintain $H$ in amortized update time $O(\alpha(\alpha + \log n + \epsilon^{-4}))$ and amortized update ratio $O(\epsilon^{-4})$.

We now need to maintain a fractional matching in $H$. We cannot rely on Lemma 2.1 as in the previous proofs, because this lemma only applies to maintaining an *integer* matching. Instead, we explicitly maintain the fractional matching $M_f^H$ defined in Theorem 3.3. By definition of update ratio, every update in $G$ only changes an average of $O(\epsilon^{-4})$ edges in $H$, so only $O(\epsilon^{-4})$ vertices $v$ have their degree $d_H(v)$ changed. For each such vertex $v$, we spend $O(\beta) = O(\epsilon^{-2})$ time going through its incident edges in $H$ and updating their value to the new $1/\max\{d_H(u), d_H(v)\}$. This yields a total update time of $O(\epsilon^{-6})$ for maintaining $M_f^H$ in $H$. By Theorem 3.3, $M_f^H$ is the desired $(1+\epsilon)$ approximation to $\mu_f(G)$. □

**Overview of the paper** As mentioned in the introduction many of the theorems in this framework directly correspond to theorems in Bernstein and Stein's paper on bipartite graphs [2], but the proofs are completely different. Theorem 3.3 is proved in Section 5 and

is analogous to Theorem 7 in [2]; in bipartite graphs, no distinction between integer and fractional matchings was needed. Theorem 3.1 is discussed in the second half of Section 5 and is analogous to Theorem 5 in [2]. Finally, Theorems 3.2 and 3.4 are discussed in Section 6 and are analogous to Theorems 6 and 8 in [2].

## 4 A $\gamma$-restricted Fractional Matching Contains a Large Integral Matching

To prove Theorem 1.4, we first prove a structural theorem about matchings in non-bipartite graphs. In bipartite graphs, because of the relationship between matching and flows, a maximum matching induces a cut in the graph. The following is an attempt to exhibit an analogous property for non-bipartite graphs.

LEMMA 4.1. *Let $M$ be a maximum matching in a graph $G$. We can partition the vertices of $G$ into three sets, $C$ (connected) , $L$ (lonely) and $B$ (both) such that the following properties hold:*

1. *All free vertices are in $L$.*
2. *Each vertex in $C$ is matched to a vertex in $L$.*
3. *There are no edges in $L \times L$.*
4. *Each vertex in $B$ has at most one neighboring vertex in $L$.*

*Proof.* Given a graph $G$ with maximum matching $M$, we define a free-free path to be a (non-empty) simple alternating path that starts and ends with edges not in $M$. We define a free-matched path to be a (non-empty) simple alternating path that starts with an edge not in $M$ and ends with an edge in $M$. Note that a free-free path need not start or end with a free vertex. We now define two vertex sets FER (free-edge-reachable) and MER (matched-edge-reachable). We say that a vertex $v$ is in MER if there is a free-matched path from some free vertex $w$ to $v$. Similarly, we say that a vertex $v$ is in FER if there is a free-free path from some free vertex $w$ to $v$. Note that FER and MER are not necessarily disjoint, and there may be vertices that are in neither set. We also let $F$ denote the set of free vertices. $F$ is disjoint from MER by definition. Observe that $F$ is also disjoint from FER because $M$ is a maximum matching and so contains no augmenting paths; suppose, for contradiction, that $v \in F \cap \text{FER}$: then the free vertex $v$ is reachable by some free-free path from a free vertex $w$, which is precisely an augmenting path from $w$ to $v$.

It is not hard to check that in bipartite graphs FER and MER are disjoint, because if a vertex $v$ was in both sets then there would be an augmenting path in the matching. Thus, in a bipartite graph we would achieve the desired partition by setting $C = \text{FER}, L = \text{MER} \cup F, B = V - C - L$. It is not hard to check that this partition satisfies all four properties of Lemma 4.1, and that in fact it achieves a stronger version of property 4, where there are *no* edges from $B$ to $L$. In a non-bipartite graph however, there could be vertices which are in both FER and MER, which roughly correspond to vertices in blossoms. For this reason we need a more involved definition of the set $B$ which ends up indirectly capturing all the blossom vertices.

For nonbipartite graphs, we define the *connected* set $C = \text{FER} \setminus \text{MER}$ and *lonely* set $L = (\text{MER} \setminus \text{FER}) \cup F$. We then define $B = V \setminus (C \cup L)$. Informally, $B$ contains non-free vertices that either are in both MER and FER or are not reachable by any alternating path from a free vertex.

We now proceed to verify the conditions of the theorem. The first condition is immediate from the definition of $L$. The second condition says that each vertex in $C$ is matched to a vertex in $L$. To prove this, we note that each vertex $v$ in $C$ is also in FER, so it is reachable by a free-free path $P_v$ from a free vertex $w$. We also know that $v$ is matched since FER is disjoint from $F$, so let $x$ be the vertex to which $v$ is matched. We want to show that $x \in \text{MER} \setminus \text{FER}$. Note that the free-free path $P_v$ cannot contain $x$ as then it would end on the matched edge $(x, v)$ The path $P_v$ followed by edge $(v, x)$ is thus a simple free-matched alternating path from $w$ to $x$ and so $x \in$ MER. Now assume, fpoc, that also $x \in$ FER. Then there is a free-free path $P_x$ from a free vertex $w'$ (which could be the same as $w$). But then $P_x$ followed by $(x, v)$ is a free-matched path from $w'$ to $v$ and therefore $v \in$ MER, which contradicts the assumption that $v \in C$. (Note that $P_x$ cannot already contain $v$ earlier in the path, for if it did it would also contain $x$ as an interior vertex, which contradicts $P_x$ being simple).

To prove the third condition, assume fpoc that there is an edge $(x, y) \in L \times L$. There are three cases to consider, depending on whether $x$ and $y$ are free. If both $x$ and $y$ are free, then there is an augmenting path from $(x, y)$, contradicting $M$ being a maximum matching. Now consider the case where one vertex, say $x$, is free, and $y \in \text{MER} \setminus \text{FER}$. Then the edge $(x, y)$ is a free-free path, so $y \in$ FER, a contradiction. The last case to consider is where both $x$ and $y$ are in $\text{MER} \setminus \text{FER}$, yet the edge $(x, y)$ exists. By definition, there is some free-matched path $P_x$ from a free vertex $w$ to $x$. Now, if $y \notin P_x$ then the path $P_x$ followed by edge $(x, y)$ is a simple free-free path from $w$ to $y$, so $y \in$ FER – contradiction. If $y \in P_x$ then let $P_y$ be the subpath of $P_x$ from $w$ to $y$. Since $y \notin$ FER, $P_y$ ends on a matched edge; thus the path $P_y$ followed by edge $(y, x)$ is a free-

free path from $w$ to $x$, so $x \in \text{FER}$, a contradiction.

The fourth condition states that each vertex in $B$ is incident to at most one vertex in $L$. Consider a vertex $v \in B$. There are two cases to consider: $v \in \text{FER} \cap \text{MER}$ and $v \in V - \text{FER} - \text{MER}$. In the latter case, there clearly cannot be an edge $(v, x)$ to some $x \in L$, since if $x$ is free then the existence of edge $(x, v)$ would imply that $v \in \text{FER}$, contradicting our assumption; similarly, if $x \in \text{MER}$, then there is some free-matched path $P_x$ from a free vertex to $x$, and so considering the path $P_x$ followed by edge $(x, v)$, we would have that $v \in \text{FER}$. We now consider the case where $v \in \text{FER} \cap \text{MER}$. Since $v \in \text{MER}$, it is the end of some free-matched path $P_v$ starting at a free vertex $w$, and is possibly the end of many such paths. Fix one such path $P_v$, let $w$ be the free vertex at the start of the path, and define the *apex* $\text{APEX}(P)$ to be, among all vertices in $P_v \cap L$, the one closest to $v$ on $P$. We now show that the only possible edge from $v$ to a vertex $x \in L$ is the edge $(v, \text{APEX}(P))$. We will establish this fact by ruling out all other possible edges. To this end, there are three cases to consider: $x \notin P$, $x \in P$ and is between between $\text{APEX}(P)$ and $v$, and $x \in P$ and between $w$ and $\text{APEX}(P)$. In the first case, the path $P$ followed by edge $(v, x)$ is simple and hence free-free, and so $x \in \text{FER}$ and not in $L$. The second case cannot occur by the definition of APEX, for $x$ would be the $\text{APEX}(P)$ in this case. In the third case, we claim that if there is such an edge, then $\text{APEX}(P) \in \text{FER}$, contradicting the fact that $\text{APEX}(P) \in L$. We show this by observing that there is a free-free path consisting of the portion of $P$ from $w$ to $x$, followed by the edge $(x, v)$, and then followed by the portion of $P$ from $v$ to $\text{APEX}(P)$. So we have ruled out all possible edges from $v$ to $L$ except the edge $(v, \text{APEX}(P))$.

It might seem strange that we picked one specific free-matched path $P_v$ and proved that the only possible edge from $v$ to $L$ is to the apex of that path, since in fact, there can be many free-matched paths to $v$. But the proof actually shows that for *any* free-matched path $P$ from a free vertex to $v$, the *only* possible edge from $v$ to $L$ is $(v, \text{APEX}(P))$; thus, if there are two different free-matched paths to $v$ with different apexes then we have shown that there are in fact *no* edges from $v$ to $L$.

**Proof of Theorem 1.4** Let $M_f$ be our $\gamma$-restricted fractional matching in $G$, let $G^*$ be the support of $M_f$ and let $M^*$ be the maximum integer matching in $G^*$. We want to show that $\text{VAL}(M_f) \leq |M^*|(1+\gamma)$.

We can assume by induction on the size of $M_f$ that all edges in $M_f$ have value $\leq \gamma$; if some edge $(x, y)$ had value 1, then $G^*$ contains no other edges incident to $x$ or $y$, so by the induction hypothesis we could find an integral matching of size at least $(\text{VAL}(M_f) - 1)/(1+\gamma)$ in $G^* - \{x\} - \{y\}$. We could then add edge $(x, y)$ to this integral matching, yielding the desired matching of size $\geq (\text{VAL}(M_f) - 1)/(1+\gamma) + 1 > \text{VAL}(M_f)/(1+\gamma)$.

We can partition the vertices into sets $C$, $L$, and $B$, that satisfy Lemma 4.1 with respect to the graph $G^*$ and the matching $M^*$. To bound $\text{VAL}(M_f)$, we consider the following accounting of the edges in $G^*$. For an edge $(x, y) \in C \times C$, let $a_x(x, y) = a_y(x, y) = 1/2$. For an edge $(x, y) \in C \times V \setminus C$, let $a_x(x, y) = 1$ and $a_y(x, y) = 0$. For an edge $(x, y) \in B \times B$, let $a_x(x, y) = a_y(x, y) = 1/2$, For an edge $(x, y) \in B \times L$, let $a_x(x, y) = 1$ and $a_y(x, y) = 0$. Note that by property 3 of Lemma 4.1, $G^*$ contains no edges in $L \times L$.

Note that for all edges $(x, y)$ we have $a_x(x, y) + a_y(x, y) = 1$. Thus, combining Observation 1 with the fact that vertices in $L$ receive no profit under this accounting:

$$\text{VAL}(M_f) = \sum_{x \in V} \rho(x) = \sum_{x \in C} \rho(x) + \sum_{x \in B} \rho(x) \ .$$

To upper bound the total profit of $C$, we simply observe that since $M_f$ is a fractional matching, for any vertex $x$, $\text{VAL}(x) \leq 1$ and $\rho(x) \leq 1$, so $\sum_{x \in C} \rho(x) \leq |C|$. To upper bound the total profit from $B$, consider an $x \in B$, and let $\text{VAL}_L(x)$ be the total value of all edges from $x$ to $L$. Then the total value of all edges from $x$ to $V \setminus L$ is at most $1 - \text{VAL}_L(x)$, and since our accounting only lets $x$ account for at most half of each edge to $V \setminus L$, we have that $\rho(x) \leq \text{VAL}_L(x) + (1 - \text{VAL}_L(x))/2 = (\text{VAL}_L(x) + 1)/2$. But by property 4 of Lemma 4.1 there is at most one edge from $x$ to $L$, and since $M_f$ is by assumption a $\gamma$-restricted fractional matching, we have that $\text{VAL}_L(x) \leq \gamma$, so $\rho(x) \leq (1+\gamma)/2$. We thus have

$$\begin{aligned} \text{VAL}(M_f) &= \sum_{x \in C} \rho(x) + \sum_{x \in B} \rho(x) \\ &\leq |C| + |B|(1+\gamma)/2 \\ &\leq (1+\gamma)(|C| + |B|/2) \ . \end{aligned} \tag{4.1}$$

There are $2M^*$ matched vertices in total; by property 1 of Lemma 4.1 all vertices in $C$ and $B$ are matched, and by property 2 there are at least $|C|$ matched vertices in $L$, so $2|C| + |B| \leq 2|M^*|$, so $|C| + |B|/2 \leq |M^*|$. By Equation 4.1 above this implies that $\text{VAL}(M_f) \leq (1+\gamma)|M^*|$, as desired. □

## 5 An EDCS Contains a Large Matching

**A Weighted EDCS Contains a Large Fractional Matching** We now sketch the proof of Theorem 3.3. The proof is conceptually quite simple, but somewhat technical because of the $\lambda$ slackness in property P2 of a weighted EDCS. Thus, we start by giving a proof that assumes a tighter version of property P2, which we call $P2'$: that for any edge $(u, v)$ in $G$, $d_H(u) + d_H(v) \geq \beta$. We leave the full proof without this assumption for Section A of the appendix.

**Proof Sketch of Theorem 3.3** We will start by defining a simple accounting on the edges of $H$. If $d_H(v) > \beta/2$ we set $a_v(v,w) = 1$ for all edges $(v,w)$. If $d_H(v) < \beta/2$ we set $a_v(v,w) = 0$ for all edges $(v,w)$. If $d_H(v) = \beta/2$ we set $a_v(v,w) = 1/2$ for all edges $(v,w)$. Property P1 of an EDCS clearly ensures that this is a valid accounting in that $a_v(v,w) + a_w(v,w) \leq 1$ for any edge $(v,w)$ in $H$.

Recall that for every edge $(u,v) \in H$, $M_f^H$ assigns $\text{VAL}(u,v) = 1/\max\{d_H(u), d_H(v)\}$. Given the accounting above, there is a simple formula for the profit of vertex $v$. If $d_H(v) > \beta/2$ then, by property P1 of an EDCS, for every edge $(v,w)$ we have $d_H(w) < d_H(v)$, so $\text{VAL}(v,w) = 1/\max\{d_H(u), d_H(v)\} = 1/d_H(v)$. Since $v$ has $d_H(v)$ incident edges and full accounting of all of them, we have $\rho(v) = 1$. Similarly if $d_H(v) = \beta/2$ then $\rho(v) = 1/2$, since $v$ now only accounts for half of each incident edge. Finally, if $d_H(v) < \beta/2$, then $v$ does not account for its edges, so clearly $\rho(v) = 0$.

Now, it is well-known [20] that there must exist a maximum fractional matching in which all edge values are 0, 1/2 or 1. From this fact, we can easily show $G$ must contain some maximum fractional matching that consists of disjoint odd cycles and disjoint isolated edges; the isolated edges all have value 1, the edges on cycles have value 1/2, and all other edges have value 0. To verify this statement, observe first that all the 1 edges must be isolated in any matching. If we remove the 1 edges from the matching, the remaining edges with value 1/2 must form a set of disjoint cycles (that are also disjoint from the set of removed 1 edges). Any even cycle can be replaced with a cycle that alternates between edges of value 0 and edges of value 1, leaving only odd cycles for the 1/2-edges. Let $M_f^G$ be such a maximum fractional matching in $G$. $M_f^G$ gets a value of 1 from each isolated edge, and a value of $d/2$ for each odd cycle of length $d$; we will show that $M_f^H$ receives exactly the same amount from each edge / odd cycle.

For each isolated edge $(v,w)$ in $M_f^G$ we have by property $P2'$ that $d_H(v) + d_H(w) \geq \beta$, so either one of $d_H(v)$ or $d_H(w)$ is larger than $\beta/2$, or both are equal to $\beta/2$; either way, $\rho(v) + \rho(w) \geq 1$, so $M_f^H$ gets a value of 1 from the edge, as desired. For a cycle of odd length $d$, let $v$ be the vertex on the cycle with maximum $d_H(v)$. It is not hard to see that $d_H(v) \geq \beta/2$, since otherwise, by property $P2'$, the neighbors of $v$ on the cycle would have degree higher than $d_H(v)$. Thus, we have that $\rho(v) \geq 1/2$. The remainder of the cycle clearly contains $(d-1)/2$ disjoint edges in $G$, and by the argument above, for each of these edges $M_f^H$ gains a profit of at least 1. Thus, the total profit of $M_f^H$ from the cycle is at least $1/2 + (d-1)/2 = d/2$. □

LEMMA 5.1. *Let $H$ be a edge degree constrained subgraph$(G, \beta, \beta(1-\lambda))$ with $\lambda = \epsilon/6$ and $\beta \geq 32\lambda^{-3}$. Then $H$ contains an $\epsilon$-restricted fractional matching $M_f^H$ with total value at least $\mu(G)(2/3 - \epsilon)$.*

*Proof.* (sketch) As in the proof of Theorem 3.3 in Section 5,we explicitly construct a fractional matching; in this case, a $\epsilon$-restricted one. Unfortunately, this is by far the most complex proof in our paper, and it relies on the probabilistic method; if we can construct a $\epsilon$-restricted matching that has size $k$ in *in expectation*, then a $\epsilon$-restricted matching of size $k$ is guaranteed to exist. Note that our overall algorithm is still deterministic because we only use randomization in the analysis. We suspect that there exists a more natural construction of a large $\epsilon$-restricted matching, but have so far been unable to find it. For this reason, we leave the full proof of Lemma 5.1 for Section B of the appendix. Here we give a sketch of the proof.

Let $M^G$ be some maximum matching in $G$. Let $M_H^G$ be the edges of $M^G$ in $H$, and let $M_{G\setminus H}^G$ be the edges of $M^G$ in $G \setminus H$, so $|M_H^G| + |M_{G\setminus H}^G| = |M^G| = \mu(G)$. Let the vertex sets $S^G$, $S_H^G$, and $S_{G\setminus H}^G$ contain the endpoints of the edges of $M^G$, $M_H^G$, and $M_{G\setminus H}^G$ respectively.

Let us consider the properties of $S_H^G$ and $S_{G\setminus H}^G$. Firstly, $S_H^G$ contains a perfect matching using edges in $H$; namely, the edges of $M_H^G$. Secondly, $|S_H^G| + |S_{G\setminus H}^G| = 2(|M_H^G| + |M_{G\setminus H}^G|) = 2\mu(G)$. Thirdly, for every edge $(u,v) \in S_{G\setminus H}^G$ we have by property P2 of an EDCS that $d_H(u) + d_H(v) \geq \beta(1-\epsilon)$. Thus, the *average* degree $d_H(v)$ of vertices in $S_{G\setminus H}^G$ is at least $\beta(1-\epsilon)/2$. For this proof sketch, let us make a big simplifying assumption: *all* vertices in $S_{G\setminus H}^G$ have degree *exactly* $\beta/2$. Note that dropping the $(1-\epsilon)$ factor is a minor assumption we could easily do without, but setting all degrees to be the same makes the proof qualitatively simpler, and allows us to avoid recourse to the probabilistic method. Finally, as we show in the full proof, if we pick our maximum matching $M^G$ carefully we can preserve the properties above while also ensuring the fourth property below. (This is a simplified version of Lemma B.2 in the full proof of Lemma 5.1).

1. $S_H^G$ contains a perfect matching using edges in $H$
2. $|S_H^G| + |S_{G\setminus H}^G| = 2\mu(G)$
3. Every vertex $v \in S_{G\setminus H}^G$ has $d_H(v) = \beta/2$ (big simplifying assumption)
4. All edges incident to $S_{G\setminus H}^G$ go to $S_H^G$

Now consider the $\epsilon$-restricted fractional matching $M_f^H$ in $H$ in which all edges in $H$ from $S_{G\setminus H}^G$ to $S_H^G$ are

given value $2/\beta$; (this is $\epsilon$-restricted because $\beta$ is large, so $2/\beta << \epsilon$). Let us first verify that no vertex has total value more than 1. Every vertex $v \in S^G_{G\setminus H}$ has degree exactly $\beta/2$ in $H$ (property 3 above), so it has total value $(\beta/2)(2/\beta) = 1$. Now consider a vertex $v \in S^G_H$. If there exists no edge $(v, w) \in H$ with $w \in S^G_{G\setminus H}$ then $v$ has total value 0 and we done. Otherwise, by property 3 above we have $d_H(w) = \beta/2$, so by property P1 of an EDCS we have $d_H(v) \leq \beta/2$, so $v$ has total value at most $(\beta/2)(2/\beta) = 1$.

Let us now consider the total value of $M^H_f$. Since each vertex in $S^G_{G\setminus H}$ has degree $\beta/2$ (property 3) and all edges from $S^G_{G\setminus H}$ go $S^G_H$ (property 4), we have a total of $\frac{\beta}{2}|S^G_{G\setminus H}|$ edges in $S^G_{G\setminus H} \times S^G_H$, and each has value $2/\beta$, so $\text{VAL}(M^H_f) = |S^G_{G\setminus H}|$.

There are now two cases to consider. The first is that $|S^G_{G\setminus H}| \geq |S^G_H|/2$. By property 2 above we then have

$$\begin{aligned}\text{VAL}(M^H_f) = |S^G_{G\setminus H}| &= \frac{1}{3}|S^G_{G\setminus H}| + \frac{2}{3}|S^G_{G\setminus H}| \\ &\geq \frac{1}{3}(S^G_{G\setminus H} + S^G_H) = \frac{2}{3}\mu(G)\end{aligned}$$

so we have sucesfully constructed the large $\epsilon$-restricted matching needed by Lemma 5.1.

The second case is that $|S^G_{G\setminus H}| < |S^G_H|/2$. In this we choose a different $\epsilon$-restricted matching which is simply the integral matching that assigns weight 1 to all the edges in the perfect matching of $S^G_H$ (property 1 above). This matching has size $|S^G_H|/2$ and by property 2 above we have:

$$\frac{1}{2}|S^G_H| = \frac{1}{6}|S^G_H| + \frac{1}{3}|S^G_H| > \frac{1}{3}(S^G_{G\setminus H} + S^G_H) = \frac{2}{3}\mu(G) \ .$$

Note that in the full proof of Lemma 5.1 we end up mixing the two cases; that is, we end up taking some edges directly from the perfect matching in $S^G_H$ (with weight 1), and some from $S^G_{G\setminus H} \times S^G_H$ (with small weight $< \epsilon$).

**Proof of Theorem 3.1** By Lemma 5.1, $H$ contains an $\epsilon$-restricted fractional matching $M^H_f$ with total value at least $\mu(G)(2/3-\epsilon)$. By Theorem 1.4, $M^H_f$ must contain an (integral) matching $M$ of size at least $\mu(G)(2/3 - \epsilon)\left(\frac{1}{1+\epsilon}\right) > \mu(G)(2/3 - 2\epsilon)$, as desired. □

## 6 Maintaining an edge degree constrained subgraph

Due to space constraints, in this section we only flesh out the algorithm for maintaining $H$ in small arboricity graphs (Theorem 3.4), leaving the proof of Theorem 3.2 for Section D of the appendix. The first half of the latter proof is nearly identical to the proof in this section, but the parameters are less intuitive so for the sake of clarity we separate the two proofs. Let $H$ be an EDCS$(G, \beta, \beta(1 - \lambda))$, and say that at all times the graph $G$ has arboricity $\leq \alpha$. Recall that by the statement of Theorem 3.4 we have $\beta \geq 4\lambda^{-1}$.

As the graph $G$ changes, we need an algorithm that changes the graph $H$ in a way that preserves the EDCS properties. An adversary will make *external* insertions and deletions in $G$, but these differ in a crucial way. If the adversary deletes an edge $(u, v) \in G$, and $(u, v)$ was also in $H$, then we must necessarily delete $(u, v)$ from $H$. However, if the adversary inserts an edge $(u, v) \in G$, we do not immediately change $H$. In either case, after the external operation, conditions P1 or P2 may be violated for $(u, v)$ or for other edges (in particular those incident to vertex $u$ or $v$). The basic idea of our algorithm is very simple: whenever the algorithm detects an edge that violates one of the EDCS properties, it fixes it through an insertion/deletion, and we call these operations *internal*: if the edge violates property P1 it removes the edge from $H$, and if the edge violates property P2 it adds the edge to $H$. If multiple edges violate the EDCS properties they can be fixed in any order. However, fixing one violation may cause violations to other edges, and the process of performing internal operations cascades. We will analyze our algorithm with a potential function which shows that *on average*, each update to $G$ only leads to a small number of internal updates to $H$.

DEFINITION 4. *We say that algorithm A for maintaining $H$ is* locally repairing *if:*

1. *Algorithm A only internally deletes edge $(u, v)$ from $H$ if before the deletion $d_H(u) + d_H(v) > \beta$ (i.e., the edge violated property P1).*
2. *Algorithm A only inserts edge $(u, v)$ into $H$ if before the insertion $d_H(u) + d_H(v) < \beta(1 - \lambda)$ (i.e., the edge violated property P2).*

If the algorithm is locally repairing, then *after* an internal insertion/deletion of edge $(u, v)$, $(u, v)$ does not violate the EDCS properties. If edge $(u, v)$ originally violated property P1, then before the update $d_H(u) + d_H(v) > \beta$; removing $(u, v)$ only decreases the edge degree by 2, so after the update $d_H(u)+d_H(v) > \beta-2 > \beta(1 - \lambda)$ (because $\beta \geq 4\lambda^{-1}$), and so $(u, v)$ satisfies property P2. Similarly, if edge $(u, v)$ violated property P2 then after the update $d_H(u)+d_H(v) < \beta(1-\lambda)+2 < \beta$, so $(u, v)$ satisfies property P1.

LEMMA 6.1. *If an algorithm $A$ for maintaining $H$ is locally repairing, then $A$ performs an amortized $O(1/\lambda)$ internal updates to $H$ for each update to $G$.*

*Proof.* Consider the potential function

$$\Phi = \sum_{(u,v)\in H} (d_H(u) + d_H(v)) - \beta(1-\lambda/2)\sum_{v\in V} d_H(v) \ .$$

We will show that an internal insertion or deletion to $H$ *decreases* $\Phi$ by at least $\beta\lambda/2$, while an external deletion to $G$ *increases* $\Phi$ by at most $2\beta$, and an external insertion leaves $\Phi$ unchanged; together these facts clearly imply the lemma. Given any update to edge $(x,y)$, we let $d^{\mathrm{O}}(x), d^{\mathrm{O}}(y), d^{\mathrm{N}}(x), d^{\mathrm{N}}(y)$ ($O$ for old, $N$ for new) denote the degrees of $x$ and $y$ before and after the edge update. Let $\Delta\Phi$ denote the change to $\Phi$ due to the update.

Consider first an internal insertion of edge $(x,y)$. $d_H(x)$ and $d_H(y)$ each increase by 1, and thus $\sum_{v\in V} d_H(v)$ increases by 2. To bound the total change to $\sum_{(u,v)\in H}(d_H(u)+d_H(v))$, observe that the degree of each of the edges incident to $x$ and $y$ increases by 1 so the total increase in edge degree of all these edges is precisely $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)$. We also added a new edge of edge degree $d^{\mathrm{N}}(x) + d^{\mathrm{N}}(y) = d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y) + 2$; thus $\Delta\Phi = 2(d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)) + 2 - 2\beta(1-\lambda/2)$. But because our algorithm is locally repairing we know that for an internal insertion $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y) < \beta(1-\lambda)$, so $\Delta\Phi < 2\beta(1-\lambda) + 2 - 2\beta(1-\lambda/2) = -\beta\lambda + 2 \le -\beta\lambda/2$, as desired. (The last inequality follows from $\beta \ge 4\lambda^{-1}$).

Consider next an internal deletion of edge $(x,y)$. The total change to $\sum_{v\in V} d_H(v)$ is now $-2$. The deletion causes the degree of each edge incident to $x$ and $y$ to go down by one, so the total change to all these edge degrees is $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)$. We also deleted an edge of edge degree $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)$. Thus, $\Delta\Phi = -2(d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)) + 2\beta(1-\lambda/2)$. Since our algorithm is locally repairing we know that for an internal deletion $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y) > \beta$, so $\Delta\Phi \le -\beta\lambda$, as desired.

Consider next an external deletion of edge $(x,y)$. The same argument as for internal deletions yields $\Delta\Phi = -2(d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)) + 2\beta(1-\lambda/2)$. Now however, we have no lower bound on $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)$ except that it is clearly positive. This yields $\Delta\Phi \le 2\beta(1-\lambda/2) < 2\beta$, as desired.

Finally, external insertions do not change $H$ and hence do not alter $\Phi$.

We now present a locally-repairing algorithm for maintaining $H$. Note that we maintain a dynamic orientation in $G$ using Theorem 2.2, so in addition to being able to process updates to $G$, the algorithm also has to be able to process edge reorientations in $G$.

LEMMA 6.2. *There exists a locally repairing algorithm $A$ for maintaining $H$ such that the update time of $A$ per update to $G$ is $O(\alpha(1+$ number of internal updates performed by $A))$, and the update time of $A$ per edge reorientation in $G$ is $O(1)$.*

We say that an edge is *violating* if it violates one of constraints P1 or P2 of an EDCS. Before proving this lemma, we design and analyze a data structure for detecting a violating edge incident to a given vertex.

LEMMA 6.3. *Let $G$ be a graph on which we maintain a dynamic orientation in which each vertex owns $O(\alpha)$ edges. There exists a data structure* VO *(violation oracle) that supports the following operations:*

- VO.*find(v) returns, in $O(\alpha)$ time, some violating edge $(v,w)$ incident to $v$, or* NIL *if none exists.*
- VO.*change-status(v,w) updates the data structure in $O(\alpha)$ time when edge $(v,w)$ is added/removed from $H$, or inserted/deleted from $G$.*
- VO.*reorient(v,w) updates the data structure in $O(1)$ time when edge $(x,y)$ is reoriented.*

*Proof.* Here we give a proof that incurs an extra $O(\log n)$ factor on each operation. We show in Section C how to remove this factor. For each vertex $v$ we keep track of $d_H(v)$ and we maintain two balanced binary search trees. $N_v^H$ contains the degrees $d_H(w)$ of all vertices $w$ for which $(v,w) \in H$ and $(v,w)$ is *not* owned by $v$. $N_v^{G\setminus H}$ contains the degrees $d_H(w)$ of all vertices $w$ for which $(v,w) \in G \setminus H$ and is not owned by $v$.

To implement VO.find(v), we first spend $O(\alpha)$ time scanning the owned edges of $d_H(v)$ for a violating edge. If none is found, we need to look for a violating edge $(v,w)$ that is not owned by $v$. Note that if $(v,w)$ violates constraint P1 of an EDCS then $(v,w) \in H$ and $d_H(v) + d_H(w) > \beta$; thus, we can find such an edge in $O(\log n)$ time by checking if $N_v^H$ contains any vertices $w$ with $d_H(w) > \beta - d_H(v)$. Similarly, we can find an edge $(v,w) \in G \setminus H$ that violates property P2 of an EDCS by searching in $N_v^{G\setminus H}$ for elements $d_H(w) < \beta(1-\lambda) - d_H(v)$.

To implement VO.change-status$(v,w)$ when the edge $(v,w)$ changes we might have to insert or remove edge $(v,w)$ from the various data structures $N_v^H, N_v^{G\setminus H}, N_w^H, N_w^{G\setminus H}$; this can clearly be done in $O(\log n)$ time. Also, $d_H(v)$ and $d_H(w)$ might have changed by 1. If $d_H(v)$ changes, for every edge $(v,x)$ that is *owned* by $v$ we have to update $N_x^H$ or $N_x^{G\setminus H}$ depending on whether $(v,x)$ is in $G$ or $G \setminus H$. But $v$ only owns $O(\alpha)$ edges, so this requires $O(\alpha \log n)$ time. We

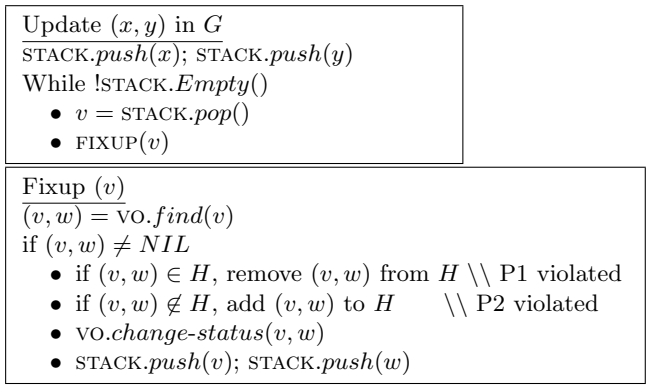


Figure 1: How the algorithm of Lemma 6.2 handles an update to $G$

can similarly handle the change to $d_H(w)$ in $O(\alpha \log n)$ time.

Finally, to implement VO.reorient$(v, w)$, if $(v, w)$ was previously owned by $v$ we have to move $d_H(v)$ out of $N_w^H$ or $N_w^{G \setminus H}$ and move $d_H(w)$ into $N_v^H$ or $N_v^{G \setminus H}$. This requires $O(\log n)$ time. We can save an $O(\log n)$ factor in this proof by replacing the binary search tree with a simple array-based structure (Section C).

**Proof of Lemma 6.2** Note that all we are looking for is a procedure for quickly detecting and fixing a violating edge; Lemma 6.1 already guarantees that any such procedure will terminate quickly.

The algorithm at all times maintains the violation oracle VO of Lemma 6.3. To handle a reorientation of edge $(x, y)$ the algorithm simply calls VO.reorient(x,y), which takes $O(1)$ time. The procedure for handling an update to $G$ is defined in Figure 1. We maintain a stack (STACK) which will contain vertices that might possibly have an incident violating edge. The key observation is that an edge $(x, y)$ can only become violating if one of $d_H(x)$ or $d_H(y)$ changes, so we will catch all violating edges if every time we add/remove some edge $(x, y)$ to/from $H$ we put $x$ and $y$ onto STACK. The while loops guarantees that when we terminate no violating edges are left.

We now need to show that the algorithm spends $O(\alpha)$ time per internal update. Each iteration of the while loop requires $O(\alpha)$ time to run VO.find and VO.change-status. Other than the two initial vertices $v$,$w$ put on STACK by the update of edge $(v, w)$ in $G$, a vertex $x$ is only put on STACK when the algorithm performs an internal update to edge $(x, y)$, so the time to process $x$ and $y$ can be charged to the internal update of $(x, y)$. The algorithm thus requires $O(\alpha)$ time for the initial update to $G$, and $O(\alpha)$ time for each subsequent internal update. □

**Proof of Theorem 3.4** Maintaining an (unweighted) EDCS: By Theorem 2.2 each update to $G$ leads to $O(\alpha + \log n)$ edge reorientations, and requires $O(\alpha(\alpha + \log n))$ time to compute the correct reorientations. by Lemma 6.1 each update to $G$ leads to amortized $O(\lambda^{-1})$ internal updates, yielding the $O(\lambda^{-1})$ bound for amortized update ratio. Thus, by Lemma 6.2, an update to $G$ can be processed in time $O(\alpha(\alpha + \log n + \lambda^{-1}))$.

To maintain a weighted EDCS, let $G^\beta$ be the graph $G$ where every edge has multiplicity $\beta$; maintaining a weighted EDCS on $G$ is equivalent to maintaining an unweighted one on $G^\beta$. Each update to $G$ can cause up to $\beta$ updates to $G^\beta$, multiplying the update time and amortized update ratio by $\beta$. However, it is not hard to show that the terms corresponding to maintaining an orientation are not multiplied by $\beta$, since an orientation of the original graph $G$ suffices. Thus the total update time is $O(\alpha(\alpha + \log n + \beta\lambda^{-1}))$. □

## 7 Conclusions

We have presented the first fully dynamic matching algorithm in general graphs to achieve a $o(m^{1/2})$ update time while maintaining a better-than-2-approximate matching. It is also the fastest known deterministic algorithm for achieving *any* constant approximation, and certainly any better-than-2 approximation. The two main open questions are whether we can achieve a $(1 + \epsilon)$ approximation in update time $O(m^{1/2-\zeta})$ for some fixed $\zeta > 0$, and whether we can achieve some better-than-2 approximation (even say 1.99) in polylog update time. These results would be interesting even if randomized, and/or limited to bipartite graphs.

## References

[1] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in O (log n) update time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 383–392, 2011.

[2] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *ICALP*, 2015.

[3] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *SODA 2015, 4-6 January, San Diego, CA, USA*, pages 785–804, 2015.

[4] Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *55th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2014, 16-21 October, 2014, Philadelphia, PA, USA*, pages 384–393, 2014.

[5] Kamalika Chaudhuri, Constantinos Daskalakis, Robert D. Kleinberg, and Henry Lin. Online bipartite

perfect matching with augmentations. In *INFOCOM*, pages 1044–1052, 2009.

[6] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1, 2014.

[7] J. Feldman, M. Henzinger, N. Korula, V. Mirrokni, and C. Stein. Online stochastic packing applied to display ad allocation. *Algorithms–ESA 2010*, pages 182–194, 2010.

[8] Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In *SODA*, pages 468–479, 2014.

[9] Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 548–557, 2013.

[10] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *47th ACM Symposium on Theory of Computing (STOC 2015)*, June 2015.

[11] Zoran Ivkovic and Errol L. Lloyd. Fully dynamic maintenance of vertex cover. In *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG '93, pages 99–111, London, UK, UK, 1994. Springer-Verlag.

[12] Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 532–543, 2014.

[13] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. 3sum hardness in (dynamic) data structures. *CoRR*, abs/1407.6756, 2014.

[14] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized on-line matching. In *focs05*, pages 264–273, 2005.

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

[16] C. St. J. A. Nash-Williams. Edge disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 36:445–450, 1961.

[17] Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013.

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

[19] Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 118–126, 2007.

[20] Edward R Scheinerman and Daniel H Ullman. Fractional graph theory: a rational approach to the theory of graphs, 2011.

[21] Vijay V. Vazirani. An improved definition of blossoms and a simpler proof of the MV matching algorithm. *CoRR*, abs/1210.4594, 2012.

## Appendices

## A Full proof of Theorem 3.3

We start by defining the following accounting for the edges of $H$. For any vertex $v$, if $d_H(v) \leq \frac{\beta}{2}(1-\sqrt{\lambda})$ we set $a_v(v,w) = 0$ for all edges $(v,w)$. If $d_H(v) \geq \frac{\beta}{2}(1+\sqrt{\lambda})$, we set $a_v(v,w) = 1$ for all edges $(v,w)$. Else, if $d_H(v) \in (\frac{\beta}{2}(1-\sqrt{\lambda}), \frac{\beta}{2}(1+\sqrt{\lambda}))$ for all edges $(v,w)$ we set $a_v(v,w) = 1/2 + \frac{d_H(v)-\beta/2}{\beta\sqrt{\lambda}}$. (It is easy to check that this is always between 0 and 1). To check that for any edge in $H$ we always have $a_v(v,w) + a_w(v,w) \leq 1$, let $(v,w)$ be some edge in $H$, WLOG let $d_H(v) \geq d_H(w)$, and recall that by property P1 of an EDCS we have $d_H(v) + d_H(w) \leq \beta$. If $d_H(w) \leq (\beta/2)(1-\sqrt{\lambda})$ then $a_w(v,w) = 0$, so since $a_v(v,w) \leq 1$, we are done. If $d_H(v) \geq (\beta/2)(1+\sqrt{\lambda})$ then $d_H(w) \leq (\beta/2)(1-\sqrt{\lambda})$, so again we are done. Finally, if both $d_H(v)$ and $d_H(w)$ are in the interval $[(\beta/2)(1-\sqrt{\lambda}), (\beta/2)(1+\sqrt{\lambda})]$, then

$$\begin{aligned}
&a_v(v,w) + a_w(v,w) \\
&= (1/2 + \frac{d_H(v)-\beta/2}{\beta\sqrt{\lambda}}) + (1/2 + \frac{d_H(w)-\beta/2}{\beta\sqrt{\lambda}}) \\
&= 1 + \frac{d_H(v)+d_H(w)-\beta}{\beta\sqrt{\lambda}} \\
&\leq 1
\end{aligned}$$

where the last inequality follows from $d_H(v) + d_H(w) \leq \beta$.

Recall that for every edge $(u,v) \in H$, $M_f^H$ assigns $\text{VAL}(u,v) = 1/\max\{d_H(u), d_H(v)\}$. Note that given a vertex $v$, for every edge $(v,w)$ we have $d_H(w) \leq \beta - d_H(v)$ (property P1 of an EDCS), so $\text{VAL}(v,w) \geq 1/\max\{d_H(v), \beta - d_H(v)\}$. Thus, we have

(A.1)

$$\begin{aligned}
\rho(v) &= \sum_{(v,w)} \text{VAL}(v,w) a_v(v,w) \\
&\geq \frac{d_H(v)}{\max\{d_H(v), \beta - d_H(v)\}} \cdot \min\{1, 1/2 + \frac{d_H(v)-\beta/2}{\beta\sqrt{\lambda}}\}
\end{aligned}$$

LEMMA A.1. *The function $\rho(v)$ satisfies the following properties:*

1. *If* $d_H(v) \geq (\beta/2)(1-\lambda)$ *then* $\rho(v) \geq (1/2)(1-3\sqrt{\lambda})$.
2. *If* $d_H(v) + d_H(w) \geq \beta(1-\lambda)$ *then* $\rho(v) + \rho(w) \geq 1 - 5\sqrt{\lambda}$.

*Proof.* To verify the first property, we simply plug in $d_H(v) = (\beta/2)(1-\lambda)$ into Equation A.1, obtaining

$$\begin{aligned}\rho(v) &\geq (1/2)((1-\lambda)/(1+\lambda))(1-\sqrt{\lambda}) \\ &> (1/2)(1-2\lambda)(1-\sqrt{\lambda}) \\ &> (1/2)(1-3\sqrt{\lambda}) \ .\end{aligned}$$

To verify the second property, let us say that $d_H(v) \geq d_H(w)$. If we had $d_H(v) \geq (\beta/2)(1+\sqrt{\lambda})$ then it is easy to see that $\rho(v) = 1$ because $d_H(v)/\max\{d_H(v), \beta - d_H(v)\} = d_H(v)/d_H(v) = 1$, and the min term in Equation A.1 also comes out to 1. Thus the only case left to consider is when $d_H(v) \leq (\beta/2)(1+\sqrt{\lambda})$, and so by property P2 of an EDCS, $d_H(w) \geq \beta(1-\lambda) - d_H(v) > (\beta/2)(1-2\sqrt{\lambda})$. Note that in this case

$$\begin{aligned}\frac{d_H(w)}{\max\{d_H(w), \beta - d_H(w)\}} &\geq \frac{(\beta/2)(1-2\sqrt{\lambda})}{(\beta/2)(1+2\sqrt{\lambda})} \\ &= \frac{1-2\sqrt{\lambda}}{1+2\sqrt{\lambda}} \\ &\geq 1 - 4\sqrt{\lambda} \ .\end{aligned}$$

Similarly, since $d_H(v) \geq d_H(w)$, we have

$$\frac{dh(v)}{\max\{d_H(v), \beta - d_H(v)\}} \geq 1 - 4\sqrt{\lambda} \ .$$

Thus, recalling that $d_H(v) + d_H(w) \geq \beta(1-\lambda)$, we have:

(A.2)

$$\begin{aligned}&pr(v) + pr(w) \\ &\geq \left(1 - 4\sqrt{\lambda}\right)\left(\frac{1}{2} + \frac{d_H(v) - \beta/2}{\beta\sqrt{\lambda}} + \frac{1}{2} + \frac{d_H(w) - \beta/2}{\beta\sqrt{\lambda}}\right) \\ &= \left(1 - 4\sqrt{\lambda}\right)\left(1 + \frac{d_H(v) + d_H(w) - \beta}{\beta\sqrt{\lambda}}\right) \\ &\geq \left(1 - 4\sqrt{\lambda}\right)\left(1 + \frac{\beta(1-\lambda) - \beta}{\beta\sqrt{\lambda}}\right) \\ &= \left(1 - 4\sqrt{\lambda}\right)\left(1 - \sqrt{\lambda}\right) \\ &\geq \left(1 - 5\sqrt{\lambda}\right) \ .\end{aligned}$$

The proof of Theorem 3.3 is almost complete. As argued in the proof sketch in Section 5, it is easy to see from [20] that any graph $G$ has a maximum fractional matching whose support consists of disjoint odd cycles and disjoint individual edges: the individual edges have value 1, the edges on the odd cycles all have value 1/2, and all other edges have value 0. Let $M_f^G$ be such a maximum matching in $G$. We will prove that $\textsc{val}(M_f^H) \geq (1-5\sqrt{\lambda})\textsc{val}(M_f^G)$. Note that $M_f^G$ gets a value of 1 from each individual edge not on a cycle, and a value of $d/2$ for each odd cycle of length $d$. We will now show that $M_f^H$ always gains at least a $(1-5\sqrt{\lambda})$ fraction of what $M_f^G$ gains from these edges/cycles. Since we set $\lambda = \epsilon^2/25$, this proves that $\textsc{val}(M_f^H) \geq \textsc{val}(M_f^G)(1-\epsilon)$.

For each isolated edge $(v,w)$ in $M_f^G$ we have by property P2 of an EDCS that $d_H(v) + d_H(w) \geq \beta(1-\lambda)$, so by property 2 of Lemma A.1 we get that $\rho(v) + \rho(w) \geq (1-5\sqrt{\lambda})$; in other words, $M_f^H$ gains at least a $(1-5\sqrt{\lambda})$ fraction of what $M_f^G$ gains on that edge. For a cycle of odd length $d$, let $v$ be the vertex on the cycle with maximum $d_H(v)$. It is not hard to see that $d_H(v) \geq \frac{\beta}{2}(1-\lambda)$, since otherwise by property P2 of an EDCS the neighbors of $v$ on the cycle would have degree higher than $d_H(v)$. Thus, by Property 1 of Lemma A.1 we have that $\rho(v) \geq (1/2)(1-3\sqrt{\lambda}) > (1/2)(1-5\sqrt{\lambda})$. The remainder of the cycle clearly contains $(d-1)/2$ disjoint edges in $G$, and by the argument above, for each of these edges $M_f^H$ gains a profit of at least $(1-5\sqrt{\lambda})$. Thus, the total profit gained by $M_f^H$ from the cycle is at least $(1/2)(1-5\sqrt{\lambda}) + ((d-1)/2)(1-5\sqrt{\lambda}) = (d/2)(1-5\sqrt{\lambda})$; again at least a $(1-5\sqrt{\lambda})$ fraction of what $M_f^G$ gains.

## B Proof of Lemma 5.1

Recall that we are dealing with an (unweighted) EDCS$(G, \beta, \beta(1-\lambda))$. We will explicitly construct a $\epsilon$-restricted fractional matching $M_f^H$ of large size. We start by defining a function $\phi(x)$ for $x \in [0, \beta]$; loosely speaking, $\phi(d_H(u))$ will end up corresponding to the profit gained by vertex $u$ in the fractional matching $M_f^H$.

$$\phi(x) = \min\left\{1, \frac{x}{2(\beta - x)}\right\} \ . \tag{B.3}$$

LEMMA B.1. *If* $a, b \in [0,1]$ *and* $a + b \geq \beta(1-\zeta)$ *for some* $\zeta \geq 0$, *then* $\phi(a) + \phi(a) \geq 1 - 5\zeta$.

*Proof.* We first show that if $a+b \geq \beta$ then $\phi(a)+\phi(b) \geq 1$. The claim is trivially true if $\phi(a) \geq 1$ or $\phi(b) \geq 1$, so we can assume that $\phi(a) < 1$ and $\phi(b) < 1$. In this case, we have

$$\begin{aligned}\phi(a) + \phi(b) &= \frac{a}{2(\beta - a)} + \frac{b}{2(\beta - b)} \\ &\geq \frac{a}{2b} + \frac{b}{2a} = \frac{a^2 + b^2}{2ab} = \frac{(a-b)^2}{2ab} + 1 \geq 1\end{aligned}$$

Now, let $\phi'(x) = \frac{\mathrm{d}}{\mathrm{dx}}\phi(x)$. To complete the lemma, it is sufficient to show that we always have $\phi'(x) \leq 5/\beta$.

To prove this inequality, first note that if $x \geq 2\beta/3$ then $\phi(x) = 1$. Thus, if $x > 2\beta/3$ then $\phi'(x) = 0$. Now, if $x \leq 2\beta/3$ then $\phi'(x) = \frac{\mathrm{d}}{\mathrm{dx}} \frac{x}{2(\beta - x)} = \frac{\beta}{2(\beta-x)^2}$. This function clearly increases with $x$, and is maximized precisely at $x = 2\beta/3$, in which case $\phi'(x) = \frac{9}{2\beta} < \frac{5}{\beta}$, as desired.

LEMMA B.2. *Given any* EDCS$(G, \beta, \beta(1-\lambda))$*, $H$, we can find two disjoint sets of vertices $X$ and $Y$ that satisfy the following properties. (Recall the function $\phi$ defined in Equation B.3).*

1. $|X| + |Y| = 2\mu(G)$.
2. *There is a perfect matching in $Y$ using edges in $H$.*
3. *Letting $\sigma = |Y|/2 + \sum_{x \in X} \phi(x)$, we have $\sigma \geq \mu(G)(1-5\lambda)$.*
4. *All edges in $H$ have at least one endpoint in $Y$.*

*Proof.* Let $M^G$ be some maximum integral matching in $G$. Some of the edges in $M^G$ are in $H$, while others are in $G \setminus H$. Let $X_0$ contain all vertices incident to edges in $M^G \cap (G \setminus H)$, and let $Y_0$ contain all vertices incident to edges in $M^G \cap H$. We now show that $X_0$ and $Y_0$ satisfy the first three properties of the lemma. Property 1 is satisfied because $X_0 \cup Y_0$ consists of all matched vertices in $M^G$. Property 2 is satisfied by definition of $Y_0$. To see that property 3 is satisfied, we show that the vertices in $X_0 \cup Y_0$ contribute an average of at least $(1-5\lambda)/2$ to $\sigma$. The vertices in $Y_0$ each contribute exactly $1/2$. Now, $X_0$ consists of $|X_0|/2$ disjoint edge in $G \setminus H$, and by property P2 of an EDCS, for each such edge $(x, x')$ we have $d_H(x) + d_H(x') \geq \beta(1-\lambda)$, so by Lemma B.1 we have $\phi(x) + \phi(x') \geq 1 - 5\lambda$, and between them $x$ and $x'$ contribute an average of at least $(1-5\lambda)/2$ to $\sigma$, as desired.

However, the sets $X_0$ and $Y_0$ might not satisfy property 4 of Lemma B.2. We first show how to transform $X_0, Y_0$ to sets $X_1, Y_1$ such that the first three properties are still satisfied, and there are no edges in $H$ between $X_1$ and $V \setminus (X_1 \cup Y_1)$; at this stage, however, there will possibly be edges in $H$ between vertices in $X_1$. To construct $X_1, Y_1$, we start with $X = X_0$ and $Y = Y_0$, and present a transformation that terminates with $X = X_1$ and $Y = Y_1$. Recall that $X_0$ has a perfect matching using edges $G \setminus H$. The set $X$ will maintain this property throughout the transformation, and each vertex $x \in X$ always has a unique *mate* $x'$. The construction does the following: as long as there exists an edge $(x, z)$ in $H$ where $x \in X$ and $z \in V \setminus (X \cup Y)$, let $x'$ be the mate of $x$; we then remove $x$ and $x'$ from $X$ and add $x$ and $z$ to $Y$. Property 1 is maintained because we removed two vertices from $|X|$ and added two to $|Y|$. Property 2 is maintained because the vertices we added to $Y$ were connected by an edge in $H$. Property 3 is maintained because $X$ clearly still has a perfect matching in $G \setminus H$, and for every pair $(x, x')$ the average contribution to $\sigma$ among $x$ and $x'$ is still at least $(1-5\lambda)/2$, as above. We continue this process while an edge $(x, y)$ in $H$ from $X$ to $V - X - Y$ exists; the process terminates because each time we are removing two vertices from $X$ and adding two to $Y$. We thus end with two sets $X_1, Y_1$ such that the first three properties of the lemma are satisfied, and there are no edges between $X_1$ and $V - X_1 - Y_1$.

We now set $X = X_1$ and $Y = Y_1$ and show how to transform $X$ and $Y$ into two sets that satisfy all four properties of the lemma. Recall that $X_1$ still contains a perfect matching using edges in $G \setminus H$: denote this matching by $M^G_X$. Our final set $X$, however, will not guarantee such a perfect matching. Let $M^H_X$ be a maximal matching in $X$ using edges in $H$. Consider the edge set $E^*_X = M^G_X \cup M^H_X$. Now, $E^*_X$ is a degree 2 graph so it can be decomposed into vertex-disjoint path and cycles. Since both $M^G_X$ and $M^H_X$ are matchings, the paths and cycles alternate between edges in $M^G_X$ and edges in $M^H_X$; in particular, they alternate between edges in $G \setminus H$ and edges in $H$. Each cycle contains an even number of vertices because otherwise it could not alternate. Because $M^G_X$ is a perfect matching, every vertex in $X_1$ is in a path or cycle, and each path starts and ends with edges in $M^G_X$. In particular, each path contains an even number of vertices and is of the form $x_1, x'_1, x_2, x'_2, ..., x_k, x'_k$, where for every $i \leq k$ there is an edge $(x_i, x'_i)$ in $M^G_X$, and for every $i < k$ there is an edge $(x'_i, x_{i+1})$ in $M^H_X$.

We now perform the following transformation. For each cycle $C$ in $E^*_X$, we simply remove all the vertices in $C$ from $X$ and add them to $Y$. Property 1 is preserved because we are moving vertices from one set to the other. Property 2 is preserved because $C$ contains a perfect matching in $H$ since every second edge in $C$ is in $M^H_X$. We will argue that property 3 is preserved momentarily. We now continue describing the transformation. For each path $P$ in $E^*_X$, if $P$ consists of a single edge in $M^G_X$, we do nothing. Else, if $P$ is longer, then $P$ is of the form $x_1, x'_1, x_2, x'_2, ..., x_k, x'_k$ indicated above. The transformation moves all vertices except $x_1$ and $x'_k$ (the ends of the path) from $X$ to $Y$. Property 1 is clearly preserved. Property 2 is preserved because the vertices moved had a perfect matching among them in $M^H_X$ and so in $H$ (the perfect matching that matches $x'_i$ to $x_{i+1}$ for $i < k$).

We must now check that property 3 is preserved by this transformation. As before, this involves showing that after the transformation, the average contribution

of a vertex in $X \cup Y$ to $\sigma$ is at least $(1-5\lambda)/2$. (Because every vertex in $X$ is incident to an edge in $E^*_X$, each vertex is accounted for in the transformation.) Now, all vertices that were in $Y_1$ remain in $Y$, so their average contribution remains at 1/2. We thus need to show that the average contribution to $\sigma$ among vertices in $X_1$ remains at least $(1-5\lambda)/2$ *after* the transformation. We will in particular show that given any cycle $C$ or path $P$ in $E^*_X$, the average contribution of vertices in that path/cycle is at least $(1-5\lambda)/2$ after the transformation. For any cycle $C$, all the vertices are moved to $Y$ and thus each contribute exactly 1/2 to $\sigma$. For any path $P = (x, x')$ that consists of a single edge in $M^G_X$, we still have that by property P2 of an EDCS, $d_H(x_i) + d_H(x'_i) \geq \beta(1-\lambda)$, so by Lemma B.1, $\phi(x) + \phi(x') \geq 1 - 5\lambda$. Finally, consider a path $P = (x_1, x'_1, ..., x_k, x'_k)$. We have that for all $i$, $(x_i, x'_i) \in G \setminus H$, so $d_H(x_i) + d_H(x'_i) \geq \beta(1-\lambda)$, so $\sum_{x\in P} d_H(x) \geq k\beta(1-\lambda)$. On the other hand, since for all $i < k$ there is an edge $(x'_i, x_{i+1})$ in $M^H_X$ and so in $H$, we have by property P1 of an EDCS that for all $i < k$, $d_H(x'_i) + d_H(x_{i+1}) \leq \beta$. Thus

(B.4)
$$\begin{aligned} d_H(x_1) + d_H(x'_k) &= \sum_{x\in P} d_H(x) - \sum_{i<k} (d_H(x'_i) + d_H(x_{i+1})) \\ &\geq k\beta(1-\lambda) - (k-1)\beta \\ &= \beta - k\beta\lambda \ . \end{aligned}$$

Thus, by Lemma B.1, we have $\phi(x_1) + \phi(x'_k) \geq 1 - 5k\lambda$. We are now ready to bound the average contribution to $\sigma$ among vertices in $P$ *after* the transformation. Since we have the endpoints contributing a total of at least $1-5k\lambda$ and $2k-2$ vertices that move to $Y$ each contributing 1/2, the average contribution of vertices on the path to $\sigma$ is at least $((1-5k\lambda)+(1/2)(2k-2))/(2k) = (1-5\lambda)/2$

Our transformation thus preserves properties 1, 2, and 3. It now remains to verify that the resulting sets $X$ and $Y$ satisfy property 4. To see this, note that we took a maximal matching $M^H_X$ of the set $X_1$ among edges in $H$, and moved all the matched vertices in $M^H_X$ to $Y$. Thus all the vertices that remain in $X$ are free in $M^H_X$, and so by definition of a maximal matching, there are no edges in $H$ between vertices in $X$ after the transformation. There are also no edges in $H$ between $X$ and $V \setminus (X \cup Y)$ because there no such edges originally when $X = X_1$, and our transformation only moved edges from $X$ to $Y$, which cannot create new edges between $X$ and $V \setminus (X \cup Y)$.

We now want to construct (for the proof) a $\epsilon$-restricted fractional matching $M^H_f$ using the edges in $H$ such that $\text{VAL}(M^H_f) \geq (2/3 - \epsilon)\mu(G)$. We start by finding two sets $|X|$ and $|Y|$ that satisfy the properties of Lemma B.2. Now, by property 2 of Lemma B.2, $|Y|$ contains a perfect matching $M^H_Y$ using edges in $H$. Let $Y^-$ be a subset of $Y$ obtained by randomly sampling exactly half the edges of $M^H_Y$ and adding their endpoints to $Y^-$. Let $Y^* = Y \setminus Y^-$, and observe that $|Y^-| = |Y^*| = |Y|/2$.

Let $H^*$ be the subgraph of $H$ (not of $G$) induced by $X \cup Y^*$. We define a fractional matching $M^{H*}_f$ on the edges of $H^*$ in which all edges have value at most $\epsilon$. We will then let our final fractional matching $M^H_f$ be the fractional matching $M^{H*}_f$ joined with the perfect matching in $H$ of $Y^-$ (so $M^H_f$ assigns value 1 to the edges in this perfect matching). $M^H_f$ is, by definition, a $\epsilon$-restricted fractional matching.

We now give the details for the construction of $M^{H*}_f$. Let $V^* = X \cup Y^*$ be the vertices of $H^*$, and let $E^*$ be its edges. For any vertex $v \in V^*$, define $d^*_H(v)$ to be the degree of $v$ in $H^*$. Recall that by property 4 of Lemma B.2, if $x \in X$ then all the edges of $H$ incident to $x$ go to $Y$ (but some might go to $Y^-$); thus, for $x \in X$, we clearly have $E[d^*_H(x)] = d_H(x)/2$.

We now define $M^{H*}_f$ as follows. For every $x \in X$, we arbitrarily order the edges of $H$ incident to $x$, and then we assign a value of $\min\left\{\epsilon, \frac{1}{\beta - d_H(x)}\right\}$ to the edges one by one, stopping when either $\text{VAL}(x)$ reaches 1 or there are no more edges in $H$ incident to $x$, whichever comes first (in the case that $\text{VAL}(x)$ reaches 1 the last edge might have value less than $\min\left\{\epsilon, \frac{1}{\beta - d_H(x)}\right\}$). We now verify that $M^{H*}_f$ is a valid fractional matching in that all vertices have value at most 1. This is clearly true of vertices $x \in X$ by construction. For a vertex $y \in Y^*$, it suffices to show that each edge incident to $y$ receives a value of at most $1/d_H(y) \leq 1/d^*_H(y)$. To see this, first note that the only edge edges to which $M^{H*}_f$ assigns non-zero values are in $X \times Y^*$. Any such edge $(x, y)$ receives value at most $1/(\beta - d_H(x))$, but since $(x, y)$ is in $M^{H*}_f$ and so in $H$, we have by property $P1$ of an EDCS that $d_H(y) \leq \beta - d_H(x)$, and so $1/(\beta - d_H(x)) \leq 1/d_H(y)$, as desired.

By construction, for any $x \in X$, we have that in $M^{H*}_f$
(B.5)
$$\text{VAL}(x) = \min\left\{1, d^*_H(x) \cdot \min\left\{\epsilon, \frac{1}{\beta - d_H(x)}\right\}\right\}$$

Consider the simple accounting on edges in $E^*$ where for every edge $(x, y) \in X \times Y^*$ we set $a_x(x, y) = 1$ and $a_y(x, y) = 0$, while for other edges $(y, y')$ we effectively ignore the edge by setting $a_y(y, y') = a_{y'}(y, y') = 0$ In this accounting, we clearly have $\rho(x) = \text{VAL}(x)$ for

$x \in X$ and $\rho(y) = 0$ for $y \in Y^*$. By Observation 1 we can lower bound $M_f^{H*}$ by summing over all $\rho(x)$ for $x \in X$. To this end, we prove the lemma below; recall that $E[d_H^*(x)] = d_H(x)/2$, that $\beta \geq 32\lambda^{-3} >> \epsilon^{-1}$, and the definition of $\phi(x)$ from Equation B.3.

LEMMA B.3. *For any $x \in X$, $E[\rho(x)] \geq (1 - \lambda)\phi(d_H(x))$.*

*Proof.* The proof is simply algebraic manipulation combined with the Chernoff bound. First consider the case in which $d_H(x) \leq \beta/2$. In this case $\frac{1}{\beta - d_H(x)} \leq 2/\beta < \epsilon$, and $d_H^*(x) \leq d_H(x) \leq \beta - d_H(x)$, so $\rho(x) = \frac{d_H^*(x)}{\beta - d_H(x)}$. Thus since $E[d_H^*(x)] = d_H(x)/2$, we have by definition of $\Phi$ that:

$$E[\rho(x)] = \frac{E[d_H^*(x)]}{\beta - d_H(x)} = \frac{d_H(x)/2}{\beta - d_H(x)} = \Phi(d_H(x))$$

Now consider the case in which $d_H(x) > \beta/2$. Then,

$$E[d_H^*(x)] = \frac{d_H(x)}{2} > \frac{\beta}{4} \geq 8\lambda^{-3} \ .$$

Thus by the Chernoff bound

$$\begin{aligned} Pr[d_H^*(x) &< (1 - \frac{\lambda}{2})(\frac{d_H(x)}{2})] \\ &< e^{-E[d_H^*(x)](\frac{\lambda}{2})^2/2} \\ &\leq e^{-\lambda^{-1}} < \frac{\lambda}{2} \ , \end{aligned} \tag{B.6}$$

where the last inequality follows from simple calculus and the fact that $0 \leq \lambda \leq 1$. Now, we start by considering the fringe case where $d_H(x) \geq \beta - \epsilon^{-1}$, and so $\min\left\{\epsilon, \frac{1}{\beta - d_H(x)}\right\} = \epsilon$. By Equation B.6 with probability at least $(1 - \frac{\lambda}{2})$, we have that $d_H^*(x) \geq \frac{(\beta - \frac{1}{\epsilon})(1 - \frac{\lambda}{2})}{2} >> \epsilon^{-1}$. So with probability at least $(1 - \frac{\lambda}{2})$ we have $d_H^*(x)\epsilon > 1$, so $E[\rho(x)] \geq (1 - \frac{\lambda}{2}) \geq (1 - \frac{\lambda}{2})\phi(x)$.

We are thus left with the case where $d_H(x) > \beta/2$, and $\rho(x) = \min\left\{1, \frac{d_H^*(x)}{\beta - d_H(x)}\right\}$. Again by Equation B.6 we have that with probability at least $(1 - \lambda/2)$,

$$\frac{d_H^*(x)}{\beta - d_H(x)} \geq \frac{d_H(x)(1 - \lambda/2)}{2(\beta - d_H(x))} \geq \left(1 - \frac{\lambda}{2}\right)\phi(d_H(x)) \ .$$

In other words, with probability at least $(1 - \lambda/2)$ we have $\rho(x) \geq (1 - \frac{\lambda}{2})\phi(d_H(x))$, so $E[\rho(x)] \geq (1 - \frac{\lambda}{2})^2\phi(d_H(x)) > (1 - \lambda)\phi(d_H(x))$, as desired.

We have almost completed the proof of Lemma 5.1. By Lemma B.3 and Observation 1, we are able to lower bound $\text{VAL}(M_f^{H*})$. In particular,

$$\text{VAL}(M_f^{H*}) \geq \sum_{x \in X} \rho(x) \geq (1 - \lambda) \sum_{x \in X} \phi(d_H(x)) \ .$$

Recall that we constructed $M_f^H$ by taking the fractional value $M_f^{H*}$ and adding in the half of the edges from $Y$ that we had removed (i.e. the edges in $Y^-$). There are in total $|Y^-|/2 = |Y|/4$ such edges so

$$\text{VAL}(M_f^H) \geq (1 - \lambda) \sum_{x \in X} \phi(d_H(x)) + \frac{|Y|}{4} \ .$$

We now lower bound this quantity using property 3 of Lemma B.2.

$$\begin{aligned} \text{VAL}(M_f^H) &\geq (1 - \lambda) \sum_{x \in X} \phi(d_H(x)) + \frac{|Y|}{4} \\ &= (1 - \lambda) \sum_{x \in X} \phi(d_H(x)) + \frac{|Y|}{2} - \frac{|Y|}{4} \\ &\geq (1 - \lambda)\mu(G)(1 - 5\lambda) - \frac{|Y|}{4} \\ &\geq (1 - 6\lambda)\mu(G) - \frac{|Y|}{4} \ . \end{aligned} \tag{B.7}$$

To complete the proof, recall that $Y$ contains a perfect matching in $H$ of $|Y|/2$ edges, so if $|Y| \geq 4\mu(G)/3$ then there already exists a matching in $H$ of size $2\mu(G)/3$, so the main lemma we are trying to prove (Lemma 5.1) is trivially true. We can thus assume that $|Y| < 4\mu(G)/3$, in which case Equation B.7 yields that

$$\begin{aligned} \text{VAL}(M_f^H) &\geq (1 - 6\lambda)\mu(G) - \frac{|Y|}{4} \\ &> (1 - 6\lambda)\mu(G) - \frac{\mu(G)}{3} \\ &= \left(\frac{2}{3} - 6\lambda\right)\mu(G) \ . \end{aligned}$$

This completes the proof because in Lemma 5.1 we set $\lambda = \epsilon/6$.

## C A Violation Oracle: proof of Lemma 6.3

In our earlier proof or Lemma 6.3 we achieved all the desired bounds to within a $\log n$ factor. This $\log n$ factor came from the fact that we used balanced binary search trees for $N_v^{G \setminus H}$ and $N_v^H$. To remove the $\log n$ factor, we maintain exactly the same oracle as in the earlier proof, except that we change the data structures $N_v^{G \setminus H}$ and $N_v^H$ to allow constant time per operation. The new data structures are extremely simple. Recall that a data structure for vertex $v$ will include all its neighbors $w$. To avoid recourse to hash tables (and the resulting randomized algorithm), all the different data structures for all vertices $v$ will use pointers to the global list of vertices $V$.

LEMMA C.1. *There exists a data structure which we call a* High Threshold Table *(*HTT*), which contains*

*some subset of $V$ of elements $w$ each with an associated* KEY*$(w)$. The* HTT *also has a threshold parameter* HIGH*, and supports the following operations in constant time.*

1. *Insert or delete some element $w$ (a key change can be implemented as a deletion followed by an insertion).*
2. *Return some element $w$ with* KEY$(w) >$ HIGH *(or* NIL *if none exists).*
3. *Increase or Decrease* HIGH *by* 1.

*There also exists an analogous data structure* Low Threshold Table *(*LTT*) that is identical except it stores a threshold* LOW*, and operation 2 requires finding an element $w$ with* KEY$(w) <$ LOW*.*

*Proof.* We group all elements $w$ into buckets according to KEY(w), so bucket $B_k$ contains all $w$ for which KEY$(w) = k$. We can store the buckets as doubly linked lists, with pointers from element $w$ in the list to the vertex $w$ in $V$, and vice versa. We then keep an *unordered* list $L$ of all indices $k$ such that $k >$ HIGH and $B_k$ is non-empty. We maintain a pointer from each bucket $B_k$ to its position in $L$ (if $k \in L$).

Operation 1 above can be implemented as follows: to insert some vertex $w$ into $N_v^H$, the algorithm simply puts $w$ in bucket $B_{\text{KEY}(w)}$. If $w$ is the only vertex in $B_{\text{KEY}(w)}$ and KEY$(w) >$ HIGH then it also adds index KEY$(w)$ to $L$. To delete some vertex $w$ from the HTT, the algorithm deletes $w$ from $B_{\text{KEY}(w)}$, and if $B_{\text{KEY}(w)}$ is now empty it deletes index KEY$(w)$ from $L$. For operation 2, the algorithm finds the first element $k$ of $L$, and then picks an arbitrary element $w$ from $B_k$; by definition of $L$ this will guarantee that KEY$(w) >$ HIGH. Finally, for operation 3, if HIGH moves by 1, let $k$ be the original HIGH. If HIGH decreased by 1, the algorithm simply adds $k$ to $L$ if bucket $B_k$ is non-empty. If HIGH increased by 1, the algorithm removes $(k+1)$ from $L$ if $(k+1)$ was in $L$. All three operations can thus be done in constant time.

Going back to the proof of Lemma 6.3, for $N_v^H$ we can use an HTT that includes the same elements $w$ as in the earlier proof in Section 6 (where we used a balanced binary search tree for $N_v^H$). We set KEY$(w) = d_H(w)$ and HIGH $= \beta - d_H(v)$. The implementation of the operations VO.find(v) and VO.reorient(v) are then exactly the same as in the earlier proof. To implement VO.change-status(v,w), we might have to increase/decrease HIGH by 1 if the update to edge $(v, w)$ increased/decreased $d_H(v)$. Similarly, for $N_v^{G \setminus H}$ we use an LTT with LOW $= \beta(1-\lambda) - d_H(v)$.

## D Maintaining an EDCS in general graphs

In this section we prove Theorem 3.2. The first half of the proof is nearly identical to that in Section 6 modulo some parameter changes, but for the sake of clarity we repeat the whole proof with the new parameters. Recall that updates to $H$ (the EDCS) can be external or internal. An external update is an update made by the dynamic adversary which inserts or deletes an edge in $G$. This external update can lead certain edges to violate the EDCS properties, so the algorithm that maintains $H$ can also make internal updates which add or remove edges from $H$ to maintains these properties. Recall that an external deletion of edge $(u, v)$ also removes the edge from $H$, while an external insertion of $(u, v)$ only affects $G$, not $H$ (though the external insertion may be followed up by an internal insertion which adds the edge to $H$).

Intuitively, the algorithm only needs to perform an internal update on an edge if that edge violates one of the EDCS properties. In Section 6 we argued that such an algorithm only performs a small number of internal updates (Lemma 6.1). The problem is that in general graphs there could be many edges and finding a violating edge is difficult. If the algorithm examines a large number of edges that are not violating, we need to somehow guarantee that progress is still being made. To this end we observe that if an algorithm comes across an edge that is *close* to violating one of the properties, it makes sense to fix it on the spot. This motivates a generalization of the definition of locally repairing (Definition 4) in Section 6. Recall that $H$ is an (unweighted) EDCS$(G, \beta, \beta(1-\lambda))$, and that by the statement of Theorem 3.2, $\beta \geq 36\lambda^{-1}$.

DEFINITION 5. *We say that algorithm $A$ for maintaining $H$ is* locally balancing *if:*

1. *Algorithm $A$ only internally deletes edge $(u, v)$ from $H$ if before the deletion $d_H(u)+d_H(v) > \beta(1-4\lambda/9)$ (i.e. the edge was close to violating property P1).*
2. *Algorithm $A$ only internally inserts edge $(u, v)$ into $H$ if before the insertion $d_H(u) + d_H(v) < \beta(1 - 5\lambda/9)$ (i.e., the edge was close to violating property P2).*

Let us say that an edge is unbalanced if it satisfies one of the inequalities above, i.e. if it is a candidate for being added or removed from $H$ by a locally balancing algorithm. Note that *after* the locally balancing algorithm updates the edge, it is no longer unbalanced. If before the update edge $(u, v)$ was in $H$ and $d_H(u) + d_H(v) > \beta(1 - 4\lambda/9)$, then since removing the edge only decreases its edge degree by 2, after the update $(u, v) \in G \setminus H$ and $d_H(u) + d_H(v) >$

$\beta(1-4\lambda/9)-2 \geq \beta(1-5\lambda/9)$, so the edge is no longer unbalanced (The last inequality follows from $\beta \geq 36\lambda^{-1}$). Similarly, if before the update $(u,v)$ was in $G \setminus H$ and $d_H(u) + d_H(v) < \beta(1-5\lambda/9)$ then after the update $(u,v) \in H$ and $d_H(u) + d_H(v) \leq \beta(1-5\lambda/9) + 2 \leq \beta(1-4\lambda/9)$.

LEMMA D.1. *If an algorithm $A$ for maintaining $H$ is locally balancing, then $A$ performs an amortized $O(1/\lambda)$ internal updates to $H$ for each update to $G$.*

*Proof.* The proof is very similar to that of Lemma 6.1. We use the same potential function

$$\Phi = \sum_{(u,v)\in H} (d_H(u) + d_H(v)) - \beta(1-\lambda/2) \sum_{v \in V} d_H(v)$$

We will show that an internal insertion or deletion to $H$ *decreases* $\Phi$ by at least $\beta\lambda/18$, while an external deletion to $G$ *increases* $\Phi$ by at most $2\beta$, and an external insertion has no effect on $\phi$; together these facts clearly imply the lemma. Given any update to edge $(x,y)$, we let $d^{\mathrm{O}}(x), d^{\mathrm{O}}(y), d^{\mathrm{N}}(x), d^{\mathrm{N}}(y)$ ($O$ for old, $N$ for new) denote the degrees of $x$ and $y$ before and after the edge update. Let $\Delta\Phi$ denote the change to $\Phi$ due to the update.

Consider first an internal insertion of edge $(x,y)$. Clearly $\sum_{v\in V} d_H(v)$ increases by 2. To bound the total change to $\sum_{(u,v)\in H}(d_H(u) + d_H(v))$, observe that all the edge degrees of edges incident to $x$ and $y$ go up by one, so the total increase in edge degree of all these edges is precisely $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)$. We also added a new edge of edge degree $d^{\mathrm{N}}(x) + d^{\mathrm{N}}(y) = d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y) + 2$; thus $\Delta\Phi = 2(d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)) + 2 - 2\beta(1-\lambda/2)$. But because our algorithm is locally balancing we know that for an internal insertion $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y) < \beta(1-5\lambda/9)$, so $\Delta\Phi < 2\beta(1-5\lambda/9) + 2 - 2\beta(1-\lambda/2) = -\beta\lambda/9 + 2 \leq -\beta\lambda/18$, as desired. (The last inequality follows from $\beta \geq 36\lambda^{-1}$.)

Consider an internal deletion of edge $(x,y)$. The total change to $\sum_{v\in V} d_H(v)$ is now $-2$. The deletion causes all edge degrees of edges incident to $x$ and $y$ to go down by one, so the total change to all these edge degrees is $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)$. We also deleted an edge of edge degree $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)$. Thus, $\Delta\Phi = -2(d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)) + 2\beta(1-\lambda/2)$. Since our algorithm is locally balancing we know that for an internal deletion $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y) > \beta(1-4\lambda/9)$, so $\Delta\Phi \leq -\beta\lambda/9$, as desired.

Consider finally an external deletion of edge $(x,y)$. The same argument as for internal deletions yields $\Delta\Phi = -2(d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)) + 2\beta(1-\lambda/2)$. Now however, we have no lower bound on $d^{\mathrm{O}}(x) + d^{\mathrm{O}}(y)$ except that it is clearly positive. This yields $\Delta\Phi \leq 2\beta(1-\lambda/2) < 2\beta$, as desired.

An external insertion only changes $G$ and not $H$, so it has no effect on $\phi$.

The main difference between maintaining an EDCS in general graphs compared to small arboricity graphs is that because we can have many edges, we cannot afford to alert all the owned neighbors of some vertex $v$ *every* time $d_H(v)$ changes. The basic idea is that instead of directly working with $d_H(v)$, each vertex $v$ will store a public value $\textsc{pub}(v)$, and when the algorithm determines what to do with an edge $(v,w)$, it will only look at its *public* edge degree, $\textsc{pub}(v) + \textsc{pub}(w)$. The algorithm will avoid excessive computation by only occasionally changing $\textsc{pub}(v)$, while guaranteeing that the algorithm nonetheless functions property by ensuring that $\textsc{pub}(v)$ is always not too far off from $d_H(v)$.

DEFINITION 6. *We say that an edge $(v,w)$ is* violating *if $(v,w)$ violates one of the* EDCS *properties: i.e. if $(v,w) \in H$ and $d_H(v) + d_H(w) > \beta$, or $(v,w) \in G \setminus H$ and $d_H(v) + d_H(w) < \beta(1-\lambda)$. We say that an edge $(v,w)$ is* publicly unbalanced *if $(v,w) \in H$ and $\textsc{pub}(v) + \textsc{pub}(w) > \beta(1-2\lambda/9)$ or if $(v,w) \in G \setminus H$ and $\textsc{pub}(v) + \textsc{pub}(w) < \beta(1-7\lambda/9)$. We say that an edge is publicly balanced if it is not publicly unbalanced.*

Our algorithm will always maintain the following three invariants:

**Publicity Invariant:** $|\textsc{pub}(v) - d_H(v)| \leq \beta\lambda/9$.

**Correctness Invariant:** After the algorithm finishes processing any update to $G$, all edge are publicly balanced.

**Balancing Invariant:** The algorithm only performs internal insertions/deletions on edges in $H$ that are publicly unbalanced.

It is is not hard to see that invariants 1 and 2 together guarantee that after the algorithm finishes processing an update to $G$, there are no violating edges in the EDCS. Invariants 1 and 3 together guarantee that the algorithm is locally balancing (Definition 5).

To find locally unbalanced edges, we present an analogue of the violation oracle of Lemma 6.3. One of the main reasons it is easier to work with $\textsc{pub}(v)$ instead of the real $d_H(v)$ is that although updating an edge $(u,v)$ can change $d_H(u)$ and $d_H(v)$, it cannot in and of itself create new publicly unbalanced edges; *a publicly unbalanced edge can only be created by changing* $\textsc{pub}(v)$ *or* $\textsc{pub}(w)$. That being said, by the publicity invariant updating an edge $(u,v)$ can indirectly force a change to $\textsc{pub}(v)$ or $\textsc{pub}(w)$.

Note that in general graphs, we use Theorem 2.1 to maintain an orientation of the edges of $G$ such that

each vertex owns $O(\sqrt{m})$ edges. But unlike in the small arboricity case, our algorithm does not need a separate procedure for handling edge reorientations because by Theorem 2.1 every update to $G$ only causes $O(1)$ reorientations, so we can bluntly model a reorientation as a deletion of the edge followed by an insertion of the same edge pointing in the other direction.

LEMMA D.2. *Suppose that we have a graph $G$ for which we maintain a dynamic orientation in which each vertex owns $O(\sqrt{m})$ edges. There exists a data structure* PBO *(public balancing oracle) on the graph $G$ that supports the following operations:*

- PBO.*find(v) returns* all *publicly unbalanced edges incident to $v$ in time $O(\sqrt{m} + (\text{number of publicly unbalanced edges returned}))$.*
- PBO.*change-status$(v, w)$ adds/removes edge $(v, w)$ from $H$, or inserts/deletes $(v, w)$ from $G$ in $O(1)$ time.*
- PBO.*change-public$(v, p)$ changes the value of* PUB$(v)$ *to $p$ in time $O(\sqrt{m}+|p-(\text{original PUB}(v))|)$.*

*Proof.* The proof is similar to that of Lemma 6.3. We use the data structures High Threshold Table (HTT) and Low Threshold Table (LTT) from Lemma C.1. For every vertex $v$, we build a HTT $N_v^H$ which contains all vertices $w$ for which $(v, w) \in H$ and $(v, w)$ is *not* owned by $v$; we set KEY$(w)$ = PUB$(w)$ and HIGH $= \beta(1-2\lambda/9)-$PUB$(v)$. We also build an LTT called $N_v^{G\setminus H}$ which contains all vertices $w$ for which $(v, w) \in G \setminus H$ and $(v, w)$ is not owned by $v$; we set KEY$(w)$ = PUB$(w)$ and LOW = $\beta(1 - 7\lambda/9) -$ PUB$(v)$.

To implement PBO.find(v), we first find all the publicly unbalanced edges that are owned by $v$ in $O(\sqrt{m})$ time by simply scanning the $O(\sqrt{m})$ owned edges of $v$. We now need to find all publicly unbalanced edges $(v, w)$ that are not owned by $v$. Let us first find all edges $(v, w) \in H$ that are not owned by $v$ and for which PUB$(v)+$PUB$(w) > \beta(1-2\lambda/9)$, which is precisely the set of elements $w \in N_v^H$ for which KEY$(w) >$ HIGH. Since $N_v^H$ is an HTT we can find one such element $w$ in $O(1)$ time. To find all of them, we repeatedly find such an element $w$ and then temporarily remove it from the HTT until we can no find no more such elements $w$: we then insert all the temporarily removed elements back into the HTT. Finding the publicly unbalanced edges not owned by $v$ thus requires a total time of $O(1 + (\text{number of elements } w \text{ found}))$. We can similarly find all edges $(v, w) \in G \setminus H$ that are not owned by $v$ and for which PUB$(v)+$PUB$(w) < \beta(1-7\lambda/9)$ by looking for elements $w \in N_v^{G\setminus H}$ for which KEY$(w) <$ LOW.

Implementing PBO.*change − status*$(v, w)$ is easy, because the values PUB$(v)$ and PUB$(w)$ do not change. Thus, the most we would have to do is add or remove $(v, w)$ from one of $N_v^H, N_v^{G\setminus H}, N_w^H$, or $N_w^{G\setminus H}$, which can be done in $O(1)$ time.

To implement PBO.*change − public*$(v, p)$, let $p_0(v)$ be the value of PUB$(v)$ before the change. First, for all edges $(v, w)$ that are owned by $v$, the change to PUB$(v)$ will change KEY$(v)$ from $p_0$ to $p$ in $N_w^H$ or $N_w^{G\setminus H}$. A key change in an HTT or LTT can be implemented in $O(1)$ time, and our orientation guarantees that $v$ owns $O(\sqrt{m})$ edges, so this operation takes $O(\sqrt{m})$ time. Secondly, we need to change HIGH and LOW in $N_v^H$ and $N_v^{G\setminus H}$ since they are defined in terms of PUB$(v)$. In particular, we have to add $(p_0 - p)$ to HIGH and LOW (note that this expression might be negative). We know that in an HTT or LTT we can change HIGH or LOW by 1 in $O(1)$ time, so we can change it by $(p_0 - p)$ in time $O(|p_0 - p|)$, as desired.

LEMMA D.3. *There exists a locally balancing algorithm $A$ for maintaining $H$ such that the amortized update time of $A$ per update to $G$ is $O((m\lambda^{-1}/\beta)(1+$ number of internal updates performed by $A))$*

*Proof.* Just as in Lemma 6.2, we do not need to bound the number of internal updates: as long as the algorithm is locally balancing, Lemma D.1 does the bounding for us. As discussed above, the algorithm $A$ will satisfy the conditions of the lemma as long as it maintains the three invariants above (publicity, correctness, balancing). The algorithm maintains a stack (STACK) of all vertices that might potentially violate the publicity invariant; this invariant can only become violated when $d_H(v)$ changes, which in turn only occurs when we add/remove edge $(v, w)$ from $H$, so whenever we add/remove an edge $(v, w)$ we put $v$ and $w$ on STACK. If a vertex $v$ on the stack violates the publicity invariant, we change PUB$(v)$ to $d_H(v)$; changing PUB$(v)$ can create publicly unbalanced edges, so we use PBO.find$(v)$ to detect and fix all of these.

The full algorithm is defined in Figure 2. We first verify that the invariants are maintained. As discussed above, the publicity invariant is maintained because whenever $d_H(v)$ changes due to an edge change in $H$ the algorithm algorithm adds $v$ to STACK, and explicitly maintains the publicity invariant for all vertices on STACK.

To see that the correctness invariant is maintained, note that an edge $(v, w)$ can only become unbalanced for two reasons: it can be a new edge added to $G$, or one of PUB$(v)$ or PUB$(w)$ must have changed. The first lines of the update procedure in Figure 2 explicitly handle

Update $(u, v)$ in $G$
if $(u, v)$ is inserted into $G$
    if PUB$(u)$ + PUB$(v) < \beta(1 - 7\lambda/9)$
        add $(u, v)$ to $H$
if $(u, v)$ is deleted from $G$
    if $(u, v) \in H$
        remove $(u, v)$ from $H$
PBO.*change-status*$(u, v)$
STACK.*push*$(u)$; STACK.*push*$(v)$
While !STACK.*Empty*()
- $w =$ STACK.*pop*()
- if $|$PUB$(w) - d_H(w)| > \beta\lambda/9$
        *balance*$(w)$

Balance $(v)$
PBO.*change-public*$(v, d_H(v))$
$S =$ PBO.*Find*$(v)$  \\ uses new PUB$(v)$
For each $(v, w) \in S$
- if $(v, w) \in H$,
        remove $(v, w)$ from $H$
- if $(v, w) \notin H$,
        add $(v, w)$ to $H$
- PBO.*change-status*$(v, w)$
- STACK.*push*$(w)$

STACK.*push*$(v)$

Figure 2: How the algorithm of Lemma D.3 handles an update to $G$

any unbalanced edge inserted into $G$ by adding it to $H$. The only time we ever change PUB$(v)$ is in the balance procedure, which after changing PUB$(v)$ finds and fixes all unbalanced edges incident to $v$.

The balancing invariant is clearly maintained, since all internal updates occur through balance(v), which only updates the unbalanced edges found through PBO.find$(v)$.

For running time analysis, we are allowed $O(m\lambda^{-1}/\beta)$ time per external update, plus an additional $O(m\lambda^{-1}/\beta)$ time per internal update. We thus need a charging scheme which shows that all the steps of the algorithm can be executed in time $O(m\lambda^{-1}/\beta)$ per update to $H$. In particular, we will think of every update to an edge $(x, y)$ in $H$ as giving $O(m\lambda^{-1}/\beta)$ credits to both $x$ and $y$.

The only non-constant operations performed by the algorithm are PBO.find$(v)$ and PBO.change-public$(v, d_H(v))$, both of which only occur during the execution of balance$(v)$. The key observation is that balance$(v)$ only occurs when the publicity invariant is violated for $v$, and since $d_H(v)$ only changes due to an update of an edge in $H$ incident to $v$, at least $\beta\lambda/9$ such updates must have occurred since the last time balance$(v)$ was executed. Thus, by the time balance$(v)$ is executed again, $v$ has accrued $O((m\lambda^{-1}/\beta)\cdot(\beta\lambda/9)) = O(\sqrt{m})$ credits, which pays for the $O(\sqrt{m})$ term in PBO.find$(v)$ and PBO.change-public$(v, d_H(v))$.

The other terms in pbo.find$(v)$ and PBO.change-public$(v, d_H(v))$ only require and additional $O(1)$ time per update to $H$. In PBO.find$(v)$ the additional term is O(number of unbalanced edges found), and each unbalanced edge found leads to an internal update to $H$. In PBO.change-public$(v, d_H(v))$, the additional term is $O(|d_H(v) - p_0|)$, where $p_0$ was the original PUB$(v)$. For $p_0$ and $d_H(v)$ to differ, at least $|p_0 - d_H(v)|$ updates to edges in $H$ incident to $v$ must have occurred since the last execution of balance$(v)$ (when PUB$(v)$ was set to $p_0$), so if each of these updates gives $O(1)$ credits to $v$, it will be enough to implement PBO.change-public$(v, d_H(v))$.

**Proof of Theorem 3.2** We use the algorithm of Lemma D.3. Since this algorithm is locally balancing, by Lemma D.1 each update to $G$ leads to amortized $O(\lambda^{-1})$ internal updates, yielding the $O(\lambda^{-1})$ bound for amortized update ratio. Thus, by Lemma D.3, an update to $G$ can be processed in amortized time $O(\sqrt{m}\lambda^{-2}/\beta)$. By Theorem 2.1, the time to maintain an orientation in $G$ only causes a constant overhead on top of this. □