Fully Dynamic Matching in Bipartite Graphs

Aaron Bernstein[1] and Cliff Stein[2(✉)]

[1] Department of Computer Science, Columbia University, New York, NY, USA
bernstei@gmail.com
[2] Department of IEOR and Computer Science, Columbia University, New York, NY, USA
cliff@ieor.columbia.edu

Abstract. We present two fully dynamic algorithms for maximum cardinality matching in bipartite graphs. Our main result is a *deterministic* algorithm that maintains a $(3/2 + \epsilon)$ approximation in *worst-case* update time $O(m^{1/4}\epsilon^{-2.5})$. This algorithm is polynomially faster than all previous *deterministic* algorithms for *any* constant approximation, and faster than all previous algorithms (randomized included) that achieve a better-than-2 approximation. We also give stronger results for bipartite graphs whose arboricity is at most α, achieving a $(1+\epsilon)$ approximation in worst-case update time $O(\alpha(\alpha+\log(n))+\epsilon^{-4}(\alpha+\log(n))+\epsilon^{-6})$, which is $O(\alpha(\alpha+\log n))$ for constant ϵ. Previous results for small arboricity graphs had similar update times but could only maintain a maximal matching (2-approximation). All these previous algorithms, however, were not limited to bipartite graphs.

1 Introduction

Finding a maximum cardinality matching in a bipartite graph is a classic problem in computer science and combinatorial optimization. There are efficient polynomial time algorithms (e.g. [11]), and well-known applications, ranging from early algorithms to minimize transportation costs (e.g. [10,13]) to recent applications in on-line advertising and social media (e.g. [7,15]). For matching, the restriction to bipartite graphs is natural and models many real-world applications. Furthermore, in many of these applications, the graph is actually changing over time. We study the *fully dynamic* variant of bipartite matching in which the goal is to maintain a near-maximum matching in a graph subject to a sequence of edge insertions and deletions. When an edge change occurs, the goal is to maintain the matching in time significantly faster than simply recomputing it from scratch.

One of our results is for bipartite *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 at most $O(\sqrt{m})$.

A. Bernstein—Supported in part by an NSF Graduate Fellowship and a Simons Foundation Graduate Fellowship.

C. Stein—Supported in part by NSF grants CCF-1349602 and CCF-1421161.

M.M. Halldórsson et al. (Eds.): ICALP 2015, Part I, LNCS 9134, pp. 167–179, 2015.
DOI: 10.1007/978-3-662-47672-7_14

1.1 Previous Work

In addition to exact algorithms on static graphs, there is previous work on approximating matching and on finding 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. Motivated partly by online advertising, there has also been significant work on "online matching" (e.g. [7,15]), both exact and approximate. In most online matching work, the graph is dynamic, but with a restricted set of updates. Typically, one side of the bipartite graph is fixed at the beginning of the algorithm. The vertices on the other side arrive, one at a time, and when a vertex arrives, we learn about all of its incident edges. Deletions are not allowed, nor typically are changes to the matching, although some work also studies models that measure the number of changes needed to maintain a matching [4,5,8].

We now turn to fully dynamic matchings. 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 here as all of the existing randomized algorithms require the assumption of an *oblivious* adversary that does not see the algorithm's random bits; thus, in addition to working only with high probability, randomized dynamic algorithms must make an extra assumption on the model which makes them inadequate in certain settings.

For maintaining an *exact* maximum matching, 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. If we restrict the model to bipartite graphs and to the incremental or decremental setting – where we allow only edge insertions or only edge deletions (but not both) – Bosek *et al.*([4]) show that we can achieve total update time (over all insertions or all deletions) $m\sqrt{n}$ for an exact matching and $m\epsilon^{-1}$ for a $(1+\epsilon)$-matching, which is optimal in that it matches the best known bounds for the static case. For the special case of *convex* bipartite graphs in the fully dynamic setting, Brodal *et al.* showed how to maintain an *implicit* (exact) matching with very fast update but slow query time.

Going back to the general problem of maintaining an explicit matching in a fully dynamic setting, we can achieve a much faster update time than $O(n^{1.495})$ if we allow approximation. One can trivially maintain a *maximal* (and so 2-approximate) matching in $O(n)$ time per update. Ivkovic and Lloyd [12] showed how to improve the update time to $O((m+n)^{\sqrt{2}/2})$. Onak and Rubinfeld [18] were to first to achieve truly fast update times, presenting 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.*[2] improved upon this with a 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 suffer from being amortized and inherently randomized, and also from the fact their techniques focus on local changes, and so seem unable to break through the barrier of a 2-approximation.

The first result to achieve a better-than-2 approximation was by Neiman and Solomon [17], who presented a *deterministic, worst-case* algorithm for maintaining a 3/2-approximate matching. However, the price of this improvement was a huge increase in update time: from $O(\log n)$ to $O(\sqrt{m})$. 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 two deterministic algorithms are strongly tethered to the $\sqrt{m}$ bound and do not seem to contain any techniques for breaking past it. An important open question was thus: can we achieve $o(\sqrt{m})$ update with a deterministic algorithm? (In fact Onak and Rubinfeld [18] presented a deterministic algorithm with amortized update time $O(\log^2 n)$, but it only achieves a $\log(n)$-approximation.) Very 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. The same paper presents 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 $O(\log(n)/\log\log(n))$; using a recent dynamic orientation algorithm of Kopelowitz *et al.*[14], this algorithm yields a $O(\log(n))$ *worst-case* update time.

Abboud and Williams [1] recently showed a conditional lower bound for dynamic matching in general graphs assuming that 3-sum cannot be solved in $o(n^2)$ time; they show that there exists a constant $k \in [2, 10]$ with the following property: any algorithm that maintains an approximate matching in which every augmenting path has length at least $2k-1$ has amortized update time $\Omega(m^{1/3})$.

1.2 Results

If we disregard special cases such as small arboricity or fractional matchings, we see that existing algorithms for dynamic matching seem to fall into two groups: there are fast (mostly randomized) algorithms that do not break through the 2-approximation barrier, and there are slow algorithms with $O(\sqrt{m})$ update that achieve a better-than-2 approximation. Thus the obvious question is whether we can design an algorithm – deterministic or randomized – that achieves a tradeoff between these two: a $o(\sqrt{m})$ update and a better-than-2 approximation. We answer this question in the affirmative for bipartite graphs.

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

This theorem achieves a new trade-off even if one considers existing randomized algorithms. Focusing on only deterministic algorithms the improvement is even more drastic: our algorithm improves upon not just $\sqrt{m}$ but $m^{1/3}$, and so achieves the fastest known deterministic update time (excluding the $\log(n)$-approximation of [18]), while still maintaining a better-than-2 approximation.

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. Of course, our algorithm has the disadvantage of only working on bipartite graphs.

For small arboricity graphs we also show how to break through the maximal matching (2-approximation) barrier and achieve a $(1+\epsilon)$-approximation.

Theorem 2. *Let G be a bipartite graph subject to a series of edge insertions and deletions, and let ϵ be < 1. Say that at all times G has arboricity at most α. Then, we can maintain a $(1+\epsilon)$-approximate matching in G in deterministic worst-case update time $O(\alpha(\alpha+\log(n))+\epsilon^{-4}(\alpha+\log(n))+\epsilon^{-6})$ For constant α and ϵ the update time is $O(\log(n))$, and for α and ϵ polylogarithmic the update time is polylogarithmic.*

Note that a $(1+\epsilon)$-approximation with polylog update time is pretty much the best we can hope for. The conditional lower bound of Abboud and Williams [1] provides an indication that such a result might not be possible for general graphs, but we have presented the first class of graphs (bipartite, polylog arboricity) for which it is achievable.

1.3 Techniques

We can think of the dynamic matching problem as follows: We are given a dynamic graph G and want to maintain a large subgraph M of maximum degree 1. This task turns out to be quite hard because, as the graph evolves, M is unstable and has few appropriate structural properties.

Very recently, Bhattacharya *et al.*[3] presented the idea of using a transition subgraph H, which they refer to as a *kernel* of G: the idea is to maintain H as G changes, and then maintain M in H. Maintaining an approximate matching M is significantly easier in a bounded degree graph, so we need a graph H that has the following properties: it should have bounded degree, it should be easy to maintain in G, and most importantly, a large matching using edges in H should be a good approximation to the maximum matching in G.

Our algorithm uses the same basic idea of transition subgraph with bounded degree, but the details are entirely different from those in [3] . Their subgraph H is just a maximal B-matching with B around $m^{1/3}$, that allows some slack on the maximality constraint. The use of a maximal matching is a natural choice in a dynamic setting because maximality is a purely *local* constraint, and so easier to maintain dynamically. The downside is that as long as one relies on maximality, one can never achieve a better-than-2 approximation; due to other difficulties, their paper in fact only achieves a $(3+\epsilon)$-approximation.

The main technical contribution of this paper is to present a new type of bounded-degree subgraph, which we call an *edge degree constrained subgraph* (EDCS). The problem with a simple B-matching is that the edges are not sufficiently "spread out" to all the vertices: imagine that G consists of 4 sets L_1, L_2, R_1, R_2, each of size $n/2$, where the edges form a complete graph except that there are no edges between L_2 and R_2. One possible maximal B-matching

includes many edges between L_1 and R_1 while leaving L_2 and R_2 completely isolated. The resulting matching is only 2-approximate, which is what we are trying to overcome. Our EDCS circumvents this problem by trying to spread out edges. For each edge, instead of separately upper bounding the matching-degree of both endpoints (B-matching) it upper bounds the *sum* of the matching-degrees of the endpoints, and then captures the notion of maximality by also lower bounding this sum for edges not in the matching. Using an EDCS prevents the above scenario as the sum of the matching-degrees of edges from L_1 to R_2 will be illegally small unless the matching-degree of R_2 is raised by adding some of those edges to the graph, thus ensuring a larger matching in H.

Although the definition is somewhat similar, the structure of an edge degree constrained subgraph is entirely different from that of a maximal B-matching, and for this reason both our analysis of the approximation factor and our algorithm for maintining this subgraph are entirely different from those in [3]. In particular, while the constraints in an EDCS seem purely local in that they concern only the degrees of the endpoints of an edge, they in fact have a global effect in a way that they do not in a maximal B-matching. In the latter, as long as an edge does not directly violate the degree constraints, it can *always* be added to the maximal B-matching, without concern for the edges elsewhere in the graph. But as seen from the above example, this is not true in an EDCS: although the edges from L_1 and R_1 do not themselves violate any constraints, they prevent the constraints between L_1 and R_2 or L_2 and R_1 from being satisfied. An analysis of this global structure is what allows us to go beyond the 2-approximation. On the other hand, the same global structure makes the EDCS more difficult to maintain dynamically; we end up showing that an EDCS contains something akin to augmenting paths, although more locally well behaved. We also develop a general new technique for maintaining a transition subgraph based on dynamic graph orientation, which allows us to reduce the update time from $O(m^{1/3})$ to $O(m^{1/4})$. That being said, the additional complications inherent in an EDCS have so far prevented us from extending our results to non-bipartite graphs.

We omit many details in this extended abstract and refer the reader to the full paper for details.

2 Preliminaries

Let $G = (L \bigcup R, E)$ be an undirected, unweighted bipartite graph where $|L| = |R| = n$ and $|E| = m$. Unless otherwise specified, "graph" will always refer to a bipartite graph. In general, we will often be dealing 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.

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 sets in the matching, and *free* or *unmatched* otherwise. We now state a simple corollary of an existing result of [9].

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

Proof. This lemma immediately follows from a simple algorithm presented in Sect. 3.2 of [9] which shows how to achieve update time $|E(G)|\epsilon^{-2}/\mu(G)$ (for the transition from amortized to worst-case see appendix A.3 of the same paper), as well as the fact that we always have $|E(G)|/\mu(G) \leq 2B$ because all edges must be incident to one of the $2\mu(G)$ matched vertices in the maximum matching, and each of those vertices have degree at most B.

Orientations An orientation 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 say that u *owns* edge (u, v) and will define the *load* of a vertex v to be the number of edges owned by v. Orientations of small max load are closely linked to arboricity: every graph with arboricity α has an α-orientation [16]. Our algorithms will at all times maintain an orientation of the *dynamic* graph G. We rely on two results to do this: one by Kopelowitz *et al.*[14], and a second simple result new to this paper whose proof we leave for the full version.

Theorem 3. *[14] Given a dynamic graph G that at all times has arboricity $\leq \alpha$, there exists an algorithm that maintains an orientation with max load $O(\alpha \log(n))$ such that every insertion/deletion to G is processed in worst-case update time $O(\alpha(\alpha + \log(n)))$ and requires at most $O(\alpha + \log(n))$ edge reorientations.*

Theorem 4. *Given a dynamic graph G, we can maintain an orientation with max load $O(\sqrt{m})$ in worst-case update time $O(1)$ per insertion/deletion to G.*

3 The Framework

We now define the transition subgraph H mentioned in Sect. 1.3.

Definition 1. *An unweighted edge degree constrained subgraph(*EDCS*) (G, β, β^-) is a subset of the edges $H \subseteq E$ with the following properties:*

(P1) if (u, v) is used (in H) then $d_H(u) + d_H(v) \leq \beta$,
(P2) if (u, v) is unused (in $G - H$) then $d_H(u) + d_H(v) \geq \beta^-$.

We also define a similar subgraph where edges in H have weights, effectively allowing them to be used more than once. The properties change somewhat as now used edges can always take more weight, so it makes sense to lower bound the degrees of used edges as well. Recall that the degree of a vertex in a weighted graph is the sum of the weights of the incident edges. returnpoint

Definition 2. *A weighted edge degree constrained subgraph(*EDCS*)* (G, β, β^-) *is a subset of the edges* $H \subseteq E$ *with positive integer weights that has properties:*

(P1) if (u, v) *is used then* $d_H(u) + d_H(v) \leq \beta$
(P2) for all *edges* (u, v)*, we have* $d_H(u) + d_H(v) \geq \beta^-$

Algorithm Outline: To process an edge insertion/deletion in G: First, we update the small-max-load edge orientation (Theorem 3 or 4. Second, we update the subgraph H so it remains a valid EDCS of the changed graph G (Sect. 5); this relies on the graph orientation for efficiency. Third, we update the $(1+\epsilon)$-approximate matching in H with respect to the changes to H from the previous step (See Lemma 1). The maintained $(1+\epsilon)$-approximate matching of H is also our final matching in G; the central claim of this paper is that because H is an EDCS, $\mu(H)$ is not too far from $\mu(G)$, so a good approximation to $\mu(H)$ is also a decent approximation to $\mu(G)$ (see Sectionřefsec:matching).

There is a subtle difficulty that arises from using a transition graph in a dynamic algorithm. By Lemma 1, as long as H has degree bounded by Δ_H, we can maintain a $(1+\epsilon)$-approximate matching in H in time $O(\Delta_H)$ *per update in* H. But a single change in G could in theory causes many changes in H, each of which would take $O(\Delta_H)$ time to process. This motivates the following definition: given an algorithm A that maintains a subgraph H in a dynamic graph G, we define the *update ratio* of A to be the maximum number of edge changes (insertions or deletions) that A could make to H given a single edge change in G.

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 ϵ corresponds to the desired approximation ratio (either $(1+\epsilon)$ or $(3/2+\epsilon)$).

3.1 General Bipartite Graphs

For the sake of intuition, think of β in the two theorems below as roughly $m^{1/4}$.

Theorem 5. *Let* G *be a bipartite graph, and let* $\lambda = \epsilon/4$*. Let* H *be an unweighted* EDCS *with* $\beta^- = \beta(1-\lambda)$*, where* β *is a parameter we will choose later. Then* $\mu(H) \geq (2/3 - \epsilon)\mu(G)$.

Theorem 6. *Let* G *be a bipartite graph. Let* H *be an unweighted* EDCS *with* $\beta^- = \beta(1-\lambda)$*, where* λ *is a positive constant less than* 1*. There is an algorithm that maintains* H *over updates in* G *(i.e. maintains* H *as a valid edge degree constrained subgraph) with the following properties:*

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

Proof of Theorem 1 We use the algorithm outline presented near the beginning of Sect. 3. We let be transition subgraph H be an unweighted EDCS$(G, \beta, \beta(1-\lambda))$ with $\lambda = 4\epsilon^{-1} = O(\epsilon^{-1})$ and $\beta = m^{1/4}\epsilon^{1/2}$. By Theorem 6 we can maintain H in worst-case update time $O((\frac{1}{\lambda})\left(\beta + \frac{\sqrt{m}}{\lambda\beta}\right)) = O(m^{1/4}\epsilon^{-2.5} + m^{1/4}\epsilon^{-.5}) = O(m^{1/4}\epsilon^{-2.5})$. The update ratio is $O(\lambda^{-1}) = O(\epsilon^{-1})$. Since degrees in H are clearly bounded by β, by Lemma 1 we can maintain a $(1+\epsilon)$-approximate 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 7, $\mu(H)$ is a $(3/2+\epsilon)$-approximation to $\mu(G)$, so our matching is a $(3/2+\epsilon)(1+\epsilon) = (3/2+\epsilon)$-approximate matching in G. □

3.2 Small Arboricity Graphs

Theorem 7. *Let G be a bipartite graph, and let $\beta > 4\epsilon^{-2}$. Let H be a weighted* EDCS *with $\beta^- = \beta - 1$. Then $\mu(H) \geq \mu(G)(1-\epsilon)$.*

Theorem 8. *Let G be a bipartite graph with arboricity α. Let H be a weighted* EDCS *with $\beta^- = \beta - 1$. There is an algorithm that maintains H over updates in G with the following properties:*

- *The algorithm has worse-case update time $O(\beta^2(\alpha + \log n) + \alpha(\alpha + \log n))$.*
- *The update ratio of the algorithm is $O(\beta)$.*

The proof of Theorem 2 is analogous to that of Theorem 1 with β set to ϵ^{-2}.

4 An EDCS Contains an Approximate Matching

In this section we prove Theorems 5 and 7. Both proofs will be by contradiction; for example, for Theorem 5 to be false, there must be an unweighted EDCS$(G, \beta, \beta(1-\lambda))$ H such that $\mu(H) < (2/3 - \epsilon)\mu(G)$. To exhibit the contradiction, we start by establishing a property that must hold of *any* subgraph H defined on the full vertex set of G for which $\mu(H)$ is smaller than $\mu(G)$; the smaller $\mu(H)$, the more constraining the property. Loosely speaking, the property is a generalization of the fact that the maximum matching on H establishes an (S, T) cut with no edges crossing in H, but at least $\mu(G) - \mu(H)$ edges crossing in G. We use the convention that the subscript L or R refer to the side of the bipartition in which the vertices lie. The proof of the following lemma involves a careful accounting of augmenting paths and is left for the full version.

Lemma 2. *Let $G = (V, E_G)$ be a bipartite graph, and let $H = (V, E_H)$ be a subgraph of G. Then, there exist vertex sets $S_L^*, S_L, S_R, T_R^*, T_R, T_L$ with the following properties:*

1. *$|S_L| + |T_L| = |S_R| + |T_R| = \mu(H)$.*
2. *In E_H, all edges incident to $S_L \bigcup S_L^*$ go to S_R and all edges incident to $T_R \bigcup T_R^*$ go to T_L.*

3. *G contains a perfect matching between S_L and S_R and between T_L and T_R ($|S_L| = |S_R|, |T_L| = |T_R|$).*
4. *$|S_L^*| = |T_R^*| = \mu(G) - \mu(H)$ and G contains a perfect matching between these sets.*

Let us say, for contradiction, that $\mu(H)$ is much smaller than $\mu(G)$. Then according to Lemma 2, there is a perfect matching between S_L^* and T_R^* in G but not H. Thus, by property P2 of an EDCS, for every edge (v, w) on that matching $d_H(v) + d_H(w)$ must be almost β. This implies that the average degree in H of vertices in S_L^* and T_R^* must be at least around $\beta/2$. But all the edges in H incident to S_L^* and T_R^* can only go to S_L and T_R, which are relatively small if $\mu(H)$ is much smaller than $\mu(G)$. To close the contradiction we argue that because of property P1 of an EDCS, we simply won't be able to fit all those edges from S_L^* to S_R and T_R^* to T_L. We argue this by bounding how high degrees can get in an EDCS. Intuitively, if U and V have equal size and all edges are between U and V, we expect the average degree on each side to be no more than $\beta/2$, as if each vertex had degree $\beta/2$ then all edge degrees would be β – the maximum allowed by property P1. We now state a generalization of this intuition which shows that if one of the sets U, V is larger than the other, it will have average degree below $\beta/2$; the proof is left for the full version.

Lemma 3. *Let us say that in some graph we have disjoint sets (U, V) such that $|U| = c|V|$, and all edges incident to U go to V (but there may be edges incident to V which do not go to U). Let $d(v)$ be the degree of vertex v in this graph, and say that for every edge (u, v) in the graph $d(u) + d(v) \leq \beta$ for some parameter β. Then, the average degree of vertices in U is at most $\frac{\beta}{c+1}$.*

Proof of Theorem 5: Let us say, for the sake of contradiction, that we had $\mu(H) < (2/3 - \epsilon)\mu(G)$. Then, we have sets $S_L^*, S_L, S_R, T_R^*, T_R, T_L$ as in Lemma 2. By property 4 of this lemma, S_L^* and T_R^* have a perfect matching between them consisting of $\mu(G) - \mu(H)$ edges in $E_G - E_H$ – that is, a perfect matching of *unused* edges. Thus, by the property P2 of an EDCS, for each edge (u, v) in this matching we have $d_H(u) + d_H(v) \geq \beta(1 - \lambda)$, which implies that the total degree of vertices in $S_L^* \bigcup T_R^*$ is at least $\beta(1 - \lambda)(\mu(G) - \mu(H))$. Now, by property 4 of Lemma 2 we know that $|S_L^*| = |T_R^*| = \mu(G) - \mu(H)$, so $|S_L^* \bigcup T_R^*| = 2(\mu(G) - \mu(H))$, so we have:

$$\text{average degree of } S_L^* \bigcup T_R^* \geq \frac{\beta(1-\lambda)(\mu(G) - \mu(H))}{2(\mu(G) - \mu(H))} = \beta\frac{(1-\lambda)}{2}. \tag{1}$$

We argue such a high average degree is not possible. Since $\mu(H) < (2/3 - \epsilon)\mu(G)$:

$$|S_L^* \bigcup T_R^*| = 2(\mu(G) - \mu(H)) > \mu(H)(1 + \epsilon). \tag{2}$$

Observe that we are now in the situation described in Lemma 3: $S_L^* \bigcup T_R^*$ corresponds to U, and $S_R \bigcup T_L$ corresponds to V. Property 2 of Lemma 2 precisely tells us that all edges from U go to V, as needed in Lemma 3. We know from

properties 3 and 1 of Lemma 2 that $|V| = |S_R \bigcup T_L| = |S_R| + |T_L| = |S_L| + |T_L| = \mu(H)$ so by Eq. 2 we have $|U| = |S_L^* \bigcup T_R^*| = c|V|$ for some $c > (1+\epsilon)$. Thus Lemma 3 tells us that the average degree of U is at most $\beta/(1+c) \leq \beta/(2+\epsilon)$, which some simple algebra shows is strictly less than $\beta(1-\lambda)/2$ because we set $\lambda = \epsilon/4$. We have thus arrived at a contradiction with Eq. 1, so our original assumption that $\mu(H) < (2/3-\epsilon)\mu(G)$ must be false. □

Small Arboricity Graphs: We now turn to Theorem 7. The full proof is left for the full version, but we give some intuition here. The statement is very similar to Theorem 5, but with two crucial differences: we are now dealing with a *weighted* EDCS H, and the approximation we need to guarantee is $1-\epsilon$ instead of $2/3-\epsilon$. (Note that Theorem 7 is true of general graphs as well; we only use it for small arboricity graphs, however, because a weighted EDCS is difficult to maintain in general graphs.) It may seem unintuitive that a weighted EDCS contains a better matching than an unweighted one since it will in fact have fewer total edges to work with. To show why a weighted EDCS is better, see for a simple example where an unweighted EDCS only contains a (3/2)-approximate matching, but a weighted one does not suffer the same issues.

In the proof of Theorem 5 we constructed the sets S_L^*, S_L, S_R, T_R^*, T_R, T_L from Lemma 2 and then argued that S_L^* (and analogously T_R^*) must have low average degree because all of its edges go to S_R, so we simply cannot fit that many edges before violating property P1 of an EDCS. Now, we could upper bound the average degree of S_L^* even better if we could argue that there also had to be other edges coming into S_R, taking up space. The natural candidate would be the edges on the matching from S_L to S_R guaranteed by property 3 of Lemma 2. In Theorem 5 we were unable to take advantage of these edges because we were dealing with an *unweighted* EDCS, so a single matching worth of edges did not count for much. The properties of a weighted EDCS, however, can force this single matching to be used multiple times, thus leaving even less space for edges leaving S_L^*. The proof of Theorem 7 is thus analogous to that of Theorem 5 but requires a stronger version of Lemma 3.

5 Maintaining an Edge Degree Constrained Subgraph

In this section, we outline the proofs of Theorems 6 and 8, leaving the details for the full version of the paper.

Recall that $\delta(u,v)$ denotes the edge degree of (u,v), $d_H(u)+d_H(v)$. We define an edge to be *full* if it is in H and has edge degree β. We define it to be *deficient* if it is not in H and has the minimum allowable edge degree β^-: this is $\beta-1$ for the weighted EDCS in Theorem 8 and $\beta(1-\lambda)$ for the unweighted EDCS of Theorem 6. We define a vertex to be *increase-safe* if it has no incident full edges and *decrease-safe* if it has no incident deficient edges; it is easy to see that increasing (decreasing) the degree of an increase-safe (decrease-safe) vertex by one does not lead to a violation of any EDCS constraints.

Now, let us say that we delete some edge (u,v) from G. If (u,v) was not in the EDCS H then all constraints remain satisfied. Otherwise, deleting (u,v) causes

the degree of u and v to decrease by one. Let us focus on fixing up vertex v; vertex u can then be handled analogously. If v was decrease-safe, then all constraints relating to v remain satisfied and we are done. Otherwise, it must have had some incident deficient edge (v, v_2). Adding this edge to H rebalances the degree of v to what it was before the deletion, but now the degree of v_2 has increased by one. If v_2 was increase-safe, the degree increase does not violate any constraints, and we are done. Otherwise, v_2 must have an incident full edge (v_2, v_3) which we delete from the graph; this rebalances v_2 but decreases the degree of v_3, so we look for an incident deficient edge. We continue in this fashion until we end on an increase/decrease-safe vertex.

We can thus fix up an edge deletion by finding an alternating path of full and deficient edges that ends in an increase/decrease-safe vertex. Insertions are handled analogously. This is similar to finding an augmenting path in a matching except that this latter case is much harder because we might hit a dead end and have to back track; but we can fix up an EDCS by following *any* sequence of full/deficient edges. Moreover, the resulting alternating path is always simple and contains few edges: for the small arboricity case (Theorem 8) where $\beta^- = \beta - 1$, it is not hard to see that in any such alternating path the vertex degrees $d_H(v)$ on either side of the bipartition are either increasing or decreasing by 1, so since $d_H(v)$ is always between 0 and β, the path has length $O(\beta)$; in the small arboricity case, $O(\beta)$ is small because we set $\beta = O(1/\epsilon^2)$. In the general case (Theorem 6), β is large but the gap between β and β^- is $\beta\lambda$, so degrees on either side change by $\beta\lambda$ and the path has length only $O(1/\lambda)$.

To find such an alternating path of full and deficient edges we maintain a data structure that for any vertex v can return an incident full or deficient edge (whichever is asked for), or indicate that none exists. Since the alternating path will always be short, this data structure will only be queried a small number of times per insertion/deletion in G. We maintain this data structure using a dynamic orientation, in which each edge is owned by one of its endpoints (see end of Sect. 2). Let us focus on the small arboricity case, where the dynamic orientation maintains a small max load. Each vertex will maintain fullness/deficiency information about the edges it does *not* own, storing each category of edge (full/deficient) in its own list. To find a full/deficient edge incident to some vertex v, the data structure simply picks an edge from the corresponding list in $O(1)$ time; if the list is empty, the data structure then manually checks all the edges that v *does* own: since the max load is small, this can be done efficiently. When the status of a vertex v changes, to maintain itself the data structure must transfer this information along all edges (v, u) that are *not* owned by u, but since these are precisely the edges owned by v, there can only be a small number of them.

The basic idea is the same for general bipartite graphs (Theorem 6), except that now the max load is $O(\sqrt{m})$, and we cannot afford to spend $O(\sqrt{m})$ per update. Note that in this case, however, there is a gap of $\beta\lambda$ between full and deficient edges, so intuitively, the degree of a vertex has to change $\beta\lambda$ time before it must be updated in the data structure. This leads to an update time

of around $\sqrt{m}/(\beta\lambda)$, as needed in Theorem 6. The details, however, are quite involved, especially since we need a *worst-case* update time.

6 Conclusion

We have presented the first fully dynamic matching algorithm to achieve a $o(\sqrt{m})$ update time while maintaining a better-than-2-approximate bipartite matching. It is also the fastest known deterministic algorithm for achieving *any* constant approximation, and certainly any better-than-2 approximation. The main open questions are in how far we can push this tradeoff. Can we achieve a *randomized* better-than-2 approximation with update time polylog(n)? For *deterministic* algorithms, can we achieve a constant approximation with update time polylog(n), or a $(1+\epsilon)$-approximation with update time $o(\sqrt{m})$?

The other natural question is whether our results can be extended to general (non-bipartite) graphs and non-bipartite graphs of small arboricity. The definition of an edge degree constrained subgraph does not inherently rely on bipartiteness, and neither do many of the techniques in this paper. The main obstruction to the generalization seems to lie in the structural property exhibited in Lemma 2. Is there an analogue for non-bipartite graphs?

Acknowledgments. We thank Tsvi Kopelowitz for several helpful discussions and for pointing us towards useful information about orientations.

References

1. Abboud, A., Williams, V.V.: Popular conjectures imply strong lower bounds for dynamic problems. In: Proceedings of FOCS 2014, pp. 434–443 (2014)
2. Baswana, S., Gupta, M., Sen, S.: Fully dynamic maximal matching in O (log n) update time. In: Proceedings of FOCS 2011, pp. 383–392 (2011)
3. Bhattacharya, S., Henzinger, M., Italiano, G.F.: Deterministic fully dynamic data structures for vertex cover and matching. In: SODA, pp. 785–804 (2015)
4. Bosek, B., Leniowski, D., Sankowski, P., Zych, A.: Online bipartite matching in offline time. In: Proceedings of FOCS 2014, pp. 384–393 (2014)
5. Chaudhuri, K., Daskalakis, C., Kleinberg, R.D., Lin, H.: Online bipartite perfect matching with augmentations. In: INFOCOM, pp. 1044–1052 (2009)
6. Duan, R., Pettie, S.: Linear-time approximation for maximum weight matching. J. ACM **61**(1), 1 (2014)
7. Feldman, J., Henzinger, M., Korula, N., Mirrokni, V.S., Stein, C.: Online stochastic packing applied to display ad allocation. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 182–194. Springer, Heidelberg (2010)
8. Gupta, A., Kumar, A., Stein, C.: Maintaining assignments online: matching, scheduling, and flows. In: SODA, pp. 468–479 (2014)
9. Gupta, M., Peng, R.: Fully dynamic (1+ e)-approximate matchings. In: Proceedings of FOCS 2013, pp. 548–557 (2013)
10. Hitchcock, F.: The distribution of a product from several sources to numberous localities. J. Math Phys. **20**, 224–230 (1941)

11. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matching in bipartite graphs. SIAM Journal on Computing **2**, 225–231 (1973)
12. Ivković, Z., Lloyd, E.L.: Fully dynamic maintenance of vertex cover. In: van Leeuwen, Jan (ed.) WG 1993. LNCS, vol. 790, pp. 99–111. Springer, Heidelberg (1994)
13. Kantorovitch, L.: On the translocation of masses. Doklady Akad. Nauk SSSR **37**, 199–201 (1942)
14. Kopelowitz, T., Krauthgamer, R., Porat, E., Solomon, S.: Orienting fully dynamic graphs with worst-case time bounds. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014, Part II. LNCS, vol. 8573, pp. 532–543. Springer, Heidelberg (2014)
15. Mehta, A., Saberi, A., Vazirani, U., Vazirani, V.: Adwords and generalized on-line matching. In: Proceedings of FOCS 2005, pp. 264–273 (2005)
16. Nash-Williams, C.S.J.A.: Edge disjoint spanning trees of finite graphs. Journal of the London Mathematical Society **36**, 445–450 (1961)
17. Neiman, O., Solomon, S.: Simple deterministic algorithms for fully dynamic maximal matching. In: Proceedings of STOC 2013, pp. 745–754 (2013)
18. Onak, K., Rubinfeld, R.: Maintaining a large matching and a small vertex cover. In: Proceedings of STOC 2010, pp. 457–464 (2010)
19. Sankowski, P.: Faster dynamic matchings and vertex connectivity. In: Proceedings of SODA 2007, pp. 118–126 (2007)