Incremental $(1-\varepsilon)$-Approximate Dynamic Matching in $O(\text{poly}(1/\varepsilon))$ Update Time

Joakim Blikstad
KTH Royal Institute of Technology, Stockholm, Sweden

Peter Kiss
Max-Planck-Institut für Informatik, Saarbrücken, Germany
University of Warwick, Coventry, UK

Abstract

In the dynamic approximate maximum bipartite matching problem we are given bipartite graph G undergoing updates and our goal is to maintain a matching of G which is large compared the maximum matching size $\mu(G)$. We define a dynamic matching algorithm to be α (respectively (α, β))-approximate if it maintains matching M such that at all times $|M| \geq \mu(G) \cdot \alpha$ (respectively $|M| \geq \mu(G) \cdot \alpha - \beta$).

We present the first deterministic $(1-\varepsilon)$-approximate dynamic matching algorithm with $O(\text{poly}(\varepsilon^{-1}))$ amortized update time for graphs undergoing edge insertions. Previous solutions either required super-constant [Gupta FSTTCS'14, Bhattacharya-Kiss-Saranurak SODA'23] or exponential in $1/\varepsilon$ [Grandoni-Leonardi-Sankowski-Schwiegelshohn-Solomon SODA'19] update time. Our implementation is arguably simpler than the mentioned algorithms and its description is self contained. Moreover, we show that if we allow for additive $(1, \varepsilon \cdot n)$-approximation our algorithm seamlessly extends to also handle vertex deletions, on top of edge insertions. This makes our algorithm one of the few small update time algorithms for $(1-\varepsilon)$-approximate dynamic matching allowing for updates both increasing and decreasing the maximum matching size of G in a fully dynamic manner.

Our algorithm relies on the weighted variant of the celebrated Edge-Degree-Constrained-Subgraph (EDCS) datastructure introduced by [Bernstein-Stein ICALP'15]. As far as we are aware we introduce the first application of the weighted-EDCS for arbitrarily dense graphs. We also present a significantly simplified proof for the approximation ratio of weighed-EDCS as a matching sparsifier compared to [Bernstein-Stein], as well as simple descriptions of a fractional matching and fractional vertex cover defined on top of the EDCS. Considering the wide range of applications EDCS has found in settings such as streaming, sub-linear, stochastic and more we hope our techniques will be of independent research interest outside of the dynamic setting.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Approximation algorithms analysis

Keywords and phrases Bipartite Matching, Incremental Matching, Dynamic Algorithms, Approximation Algorithms, EDCS

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.22

Related Version *Full Version*: `https://arxiv.org/abs/2302.08432` [27]

Funding *Joakim Blikstad*: Partially supported by the Swedish Research Council (Reg. No. 2019-05622).
Peter Kiss: Partially supported by Engineering and Physical Sciences Research Council, UK (EPSRC) Grant EP/S03353X/1.

Acknowledgements We thank Danupon Nanongkai for insightful discussions and valuable comments throughout the development of this work. Part of this work was done while the authors visited the Max Planck Institute of Informatics, Saarbrücken.

1 Introduction

Matchings are fundamental objects of combinatorical optimization with a wide range of practical applications. The first polynomial time algorithm for finding a maximum matching was published by Kuhn[1] [39] in 1955 which ran in $O(n^4)$ time. A long line of papers have focused on improving this polynomial complexity. Notably Edmonds and Karp [30] showed the first $O(n^3)$ time algorithm for the problem which was later improved to $O(n^{2.5})$ [36]. Mucha and Sankowski [40] showed maximum matching can be solved in matrix multiplication time, that is in $O(n^\omega)$ where ω is currently around 2.38. In the very recent breakthrough result of Chen-Kyng-Liu-Peng-ProbstGutenberg-Sachdeva [29], they showed an $O(m^{1+o(1)})$ time algorithm for the maximum flow problem (which generalizes bipartite matching) providing the first near-linear time algorithm, essentially settling the problem in the sequential setting.

Dynamic Model. This paper focuses on the matching problem in the dynamic model where it has received extensive research attention in recent years, see e.g [1, 2, 10, 14, 17, 20, 22, 28, 41, 43, 45, 47] and many more. In this setting our task is to maintain a large matching as the graph undergoes updates. We will refer to updates being fully dynamic if they concern both insertions and deletions and partially dynamic if only one of the two is allowed. The objective is to minimize the time spent maintaining the output after each update. Throughout the paper we will always refer to update time in the amortized sense – averaged over the sequence of updates. In [45] Sankowski has shown the fist improvement for the fully dynamic maximum matching problem in terms of update time ($O(n^{1.45})$) compared to static recomputation after each update. Unfortunately, based on well accepted hardness conjectures no dynamic algorithm for the maximum matching problem may achieve update time sub-linear in n [35]. Most works focused on the relaxed approximate version of the problem where the goal is to maintain a large matching in G with respect to the maximum matching size $\mu(G)$. We will refer to matching algorithms as α-approximate (respectively (α, β)-approximate) if it maintains matching M such that at all times $|M| \geq \mu(G) \cdot \alpha$ (respectively $|M| \geq \mu(G) \cdot \alpha - \beta$).

Fully Dynamic Approximate Matching. The holy grail of dynamic algorithm design is to achieve an update time of $O(\text{polylog}(n))$ or ideally even constant. For the fully dynamic approximate matching problem, a long line of research [6, 9, 18, 19, 21, 26, 42, 46] has lead to algorithms achieving $\approx \frac{1}{2}$-approximation with $O(\text{polylog}(n))$ and constant update time. No fully dynamic algorithm has been found achieving better than $\frac{1}{2}$-approximation in $O(\text{polylog}(n))$ update time for the problem, and this challenge appears to be one of the most celebrated problem within the dynamic matching literature. A set of interesting papers focused on $\approx \frac{2}{3}$-approximation in $\tilde{O}(\sqrt{n})$ update time [16, 17, 32, 38] and other approximation-ratio to polynomial-update-time trade-offs in the better-than-$\frac{1}{2}$-approximation regime were achieved by [10, 11, 44]. Note that through periodic recomputation of the matching (roughly every $\varepsilon\mu(G)$ updates) we can achieve fully dynamic $(1-\epsilon)$-approximation in $\tilde{O}(n)$ update time [34]. Very recently [12] has shown that $(1-\varepsilon)$-approximation is possible in slightly sublinear update time $O(n/\log^*(n)^{O(1)})$ suggesting that there might exist efficient non-trivial algorithms for the problem. Note that very recently [8, 25] have independently shown that if our goal is to maintain the *size* of the maximum matching (and not the edge-set) then sub-$\frac{1}{2}$ approximation is achievable in polylogarithmic update time.

[1] However, this result is usually attributed to Kőnig and Egerváry.

Partially Dynamic Matching Algorithms. For small approximation ratios, achieving poly-logarithmic update time for fully dynamic matching seems far out of reach with current techniques, or perhaps even impossible. Hence, a long line of papers have focused on maintaining a $(1-\varepsilon)$-approximate matching in partially dynamic graphs: either incremental (edge insertions) or decremental (edge deletions). The first $O(\text{poly}(\log(n), \varepsilon^{-1}))$ algorithm for maintaining a $(1-\varepsilon)$-approximate matching under edge insertions is due to Gupta [33], with amortized update time $O(\log^2(n)/\varepsilon^4)$. Their algorithm models the bipartite matching problem as a linear program, and uses the celebrated multiplicative-weights-updates framework. Generalizing the result of [33] recently Bhattacharya-Kiss-Saranurak [23] has shown that an arbitrarily close approximation to the solution of a linear program undergoing updates either relaxing or restricting (but not both) its solution polytope can be maintained in $O(\text{poly}(\log n, \varepsilon^{-1}))$ update time. Hence, the algorithm of [23] shows how to maintain a $(1-\varepsilon)$-approximate matching under either decremental or incremental updates with a unified approach. As both of these papers rely on static linear program solver sub-routines their update times inherently carry $\log(n)$ factors, and it seems implausible that these techniques can achieve constant update time independent of n.

The decremental algorithms of [15,37] focus on maintaining "evenly spread out" fractional matchings so that they are robust against edge-deletions. These algorithms rely on either maximum-flow computation or convex optimization sub-routines which similarly to LP-solvers carry $\log(n)$-factors into the update time.

The first constant time[2] partially dynamic matching algorithm is due to [31] who solve $(1-\varepsilon)$-approximate matching in incremental graphs with an update time of $(1/\varepsilon)^{O(1/\varepsilon)}$. Their solution relies on augmenting path elimination, a technique used commonly for the matching problem in the static setting. However, enumerating augmenting paths seems to inherently carry an exponential dependency on $1/\varepsilon$ due to the number of possible such paths present in the graph.

As far as we are aware partially dynamic $(1-\epsilon)$-approximate matching algorithms with update time independent of n and faster than some exponential in ϵ are all restricted to a relaxed version of the problem where the graph may undergo vertex insertions/deletions. Such an algorithm can simply be obtained through straightforward periodic rebuilds (if we allow for additive $\epsilon \cdot n$ slack) or as shown by Zheng-Henzinger [48][3]. Hence, it we can observe the following apparent gap in the literature of partially dynamic matching algorithms:

▶ **Question 1.** *Can we maintain a $(1-\varepsilon)$-approximate maximum matching of a partially dynamic bipartite graph in $O(\text{poly}(\varepsilon^{-1}))$ update time?*

Based on the current landscape of the dynamic algorithms literature, achieving $(1-\varepsilon)$-approximation under fully dynamic updates in small update times seems to be far out of reach. Contrary to the extensive research effort, no fully dynamic algorithm with $\text{poly}(\log(n), \varepsilon^{-1})$ has been found for maintaining matchings with better than $\frac{1}{2}$-approximation. This apparent difficulty proposes the research of dynamic models somewhere between fully and partially dynamic updates. The previously mentioned paper by Zheng-Henzinger [48] implements a

[2] That is constant-time whenever ε is constant, i.e. the update time should be independent of n.

[3] A very recent paper of Zheng-Henzinger [48] has initially claimed an algorithm which can maintain a $(1-\epsilon)$-approximate matching in $O(1/\varepsilon)$ update time under edge deletions. However, the authors have found a mistake in their paper and claim that their algorithm only works under specific vertex updates

$(1-\epsilon)$-approximate algorithm with $O(1/\varepsilon)$-update time which can support vertex insertions and deletions on separate sides of the bipartition. The existence of this new result proposes the following natural question:

▶ **Question 2.** *Under what kind of non-partially dynamic updates can we maintain a $(1-\varepsilon)$-approximate maximum matching of a bipartite graph?*

1.1 Our Contribution

In this paper we provide a positive answer to **Question 1** and make progress towards understanding **Question 2**. Our main result is the first $O(\text{poly}(1/\varepsilon))$ update time $(1-\varepsilon)$-approximate dynamic matching algorithm for bipartite graphs undergoing edge insertions:

▶ **Theorem 1.** *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1-\varepsilon)$-approximate maximum matching of a bipartite graph undergoing edge insertions in total update time $O(n/\varepsilon^6 + m/\varepsilon^5)$.*

Previous algorithms for $(1-\varepsilon)$ approximate dynamic matching under edge updates required update times which were either super-constant [23, 33] or had an exponential dependency on ε^{-1} [31]. Furthermore, our algorithm is arguably simpler then previous implementations and it is self contained (except for the static computation of $(1-\varepsilon)$-approximate maximum matchings) where as most dynamic matching algorithms either rely on heavy machinery from previous papers or use black-box tools like multiplicative weight updates or flow-subroutines.

We further show that if we allow for (additive) $(1, \varepsilon \cdot n)$-approximation[4] our algorithm seamlessly extends to a wider range of updates:

▶ **Theorem 2.** *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1, \epsilon \cdot n)$-approximate maximum matching of a bipartite graph undergoing edge insertions and vertex deletions in total update time of $O(n/\varepsilon^8 + m/\varepsilon^5)$.*

In contrast to the similar update time result of [48] which allows for edge deletions and vertex insertions on one side of the bipartition our algorithm allows from arbitrary vertex deletions. Our algorithm maintains a $(1-\varepsilon)$-approximate maximum matching of the graph throughout updates which can both increase and decrease the maximum matching size of the graph. Hence, we hope our techniques provide useful insight towards fully dynamizing $(1-\varepsilon)$-approximate algorithms for the matching problem.

Our algorithm relies on the weighted variant of the celebrated Edge-Degree-Constrained-Subgraph (EDCS) matching sparsifier. The unweighted EDCS (first introduced by Bernstein-Stein [16]) has found applications in a number of different computational settings: streaming [3, 4, 13], stochastic, one way communication, fault tolerant [5], sub-linear [8, 12, 24, 25] and dynamic [8, 16, 17, 32, 38]. On the other hand the *weighted* EDCS variant which provides a tighter approximation has only found applications in small arboricity graphs [16]. Hence, we initiate the study of the weighted EDCS in dense graphs.

Furthermore, we show a significantly simplified proof for the approximation ratio of the weighted EDCS datastructure with respect to the maximum matching size. In our proof, we identify simple and explicit descriptions of a fractional matching and fractional vertex covers

[4] Recall that this means that we maintain a matching of size at least $\mu - \varepsilon n$, as opposed to $\mu - \varepsilon\mu$, where μ denotes the size of the maximum matching.

defined on top of the weighted EDCS, which might be of independent interest. Moreover, we show that the dependence on the slack parameter on the maximum degree of the weighted EDCS is exactly quadratic. This in sharp contrast to the unweighted EDCS where the same relationship have been proven to be linear [7]. While within the dynamic matching algorithm literature papers don't tend to focus on exact ϵ complexities but rather n dependence, in models such as semi-streaming and distributed the ϵ dependency usually gains more focus. Our hard example (most likely) rules out applications of weighted EDCS in these models for obtaining sub ϵ^{-2} round/pass complexity algorithms. We hope that these observations will be of independent research interest due to the wide-spread popularity of the EDCS datastructure for solving matching problems.

1.2 Our Techniques

Assume H is a multi-graph defined on the vertex set of G and let $\deg_H(v)$ stand for the degree of vertex v in H. We define the degree of an edge to be the sum of the degrees of its endpoints.[5]

▶ **Definition 3** (Weighted EDCS [16]). *Given a graph $G = (V, E)$, a multiset H is called a weighted EDCS with parameter β if*[a]*:*

(i) $\deg_H(u) + \deg_H(v) \leq \beta$ *for all edges* $(u, v) \in H$.

(ii) $\deg_H(u) + \deg_H(v) \geq \beta - 1$ *for all edges* $(u, v) \in E$.

If H is not a weighted EDCS, we call an edge $e \in H$ *overfull* if it violates (i), and an edge $e \in E$ *underfull* if it violates (ii).

[a] Some authors use an additional parameter $\beta^- < \beta$ which replaces the "$\beta - 1$" in the degree-constraint. For our purposes, we will always have $\beta^- = \beta - 1$.

If $\beta = \Omega(\varepsilon^{-2})$ and H is a β-WEDCS of G then $\mu(H) \geq \mu(G) \cdot (1-\varepsilon)$ ([16], Theorem 13). In order to derive our incremental result we show that a β-WEDCS can be efficiently maintained greedily under edge insertions. In turn we can efficiently maintain a $(1 - \epsilon)$-approximate matching within the support of H through periodic recomputation.

Define a valid update of H to be one of the following: (i) and edge $e \in H$ which is overfull with respect to H gets deleted from H; and (ii) a copy of an edge $e \in E$ which is underfull with respect to H is added to H. In Lemma 7 (slightly improving on the similar lemma's of [5, 13, 16]) we show that if H is initialized as the empty graph and only undergoes valid updates, then it there are at most $O(\mu(G) \cdot \beta^2)$ many updates.

Fix some $\beta = \Theta(\varepsilon^{-2})$. Assume G is initially empty and initialize H to be an empty edge set (note that by definition initially H is a β-WEDCS of G). Assume edge e is inserted into G. If at this point e is not underfull with respect to H there is nothing to be done as H remained a valid WEDCS of G. If e is underfull with respect to H we add copies of it to H until it is not. This process of adding e to H has increased the edge degree of edges neighbouring e in H and some of them might have became overfull. To counteract this we iterate through the neighbours of e in an arbitrary order and if we find an overfull edge e' we remove it from H. This edge removal decreases the edge degrees in the neighbourhood of e'. To counteract this we recurse and look for underfull edges in the neighbourhood of

[5] Note that we are sticking to the notation weighted-EDCS instead of multi-EDCS to be in line with the naming convention of [16] which defined H to be a weighted graph with integer edge weights.

e'. If such an edge e'' is found we add copies of it to H until e'' is not underfull and repeat the same steps as if e'' was just inserted into G. This defines a natural recursive process for restoring the WEDCS properties after each edge insertion in a local and greedy way.

Whenever we have to explore the neighbourhood of an edge in $O(\Delta)$ time (where Δ is the max-degree) to either check for underfull or overfull edges we do so because H underwent a valid update. By Lemma 7 this may only happen at most $O(\mu(G) \cdot \beta^2)$ times. Hence, naively the total work spent greedily fixing the WEDCS properties is $O(\mu(G) \cdot \Delta \cdot \beta^2)$. For some graphs this value might be significantly larger then m. In order to improve the update time to $O_\beta(m)$ we assign a counter c_v to each vertex v measuring the number of valid updates of H the neighbourhood of v has underwent. Once c_v grows to $\Omega(\beta^2 \cdot \varepsilon^{-1})$ we mark v dirty and ignore further edges inserted in the neighbourhood of v. By marking a single vertex dirty and ignoring some edges incident on it we may loose out only on a single edge of any maximum matching. However, whenever we mark a vertex dirty we can charge $\Omega(\beta^2 \cdot \varepsilon^{-1})$ valid updates of H to that vertex. As there may be at most $O(\mu(G) \cdot \beta^2)$ valid updates of H in total we may only mark $O(\mu(G) \cdot \varepsilon)$ vertices dirty hence we will only ignore an $O(\epsilon)$-fraction of any maximum matching within the graph through ignoring edges incident on dirty vertices. As we may scan the neighbourhood of vertex v at most $O(\beta^2 \cdot \varepsilon^{-1}) = \text{poly}(\varepsilon^{-1})$ times until v is marked dirty we ensure that each edge is explored $\text{poly}(\varepsilon^{-1})$ times guaranteeing a total running time of $O(m \cdot \text{poly}(\varepsilon^{-1}))$. Full details can be found in Section 3.

Towards Full Dynamization. The algorithm almost seamlessly adopts to vertex deletions if we allow for $(1, \varepsilon \cdot n)$-approximation[6]. Whenever a vertex gets deleted from the graph our WEDCS H might be locally affected. This means that over the full run of the algorithm, H may undergo further valid updates then the $O(\mu(G) \cdot \beta^2)$ bound provided by Lemma 7. A potential function based argument allows us to claim that each vertex deletion may increase the total number valid updates H may undergo by $O(\beta^2)$. As each vertex may be deleted at most once this means that the total number of valid updates we might make to restore H is $O(n \cdot \beta^2)$, each update requiring $O(\Delta)$ time if naively implemented. By marking vertices as dirty as before we can guarantee amortized $O(\text{poly}(1/\varepsilon))$ update time. However, now we must mark up to $\approx \varepsilon \cdot n$ vertices as dirty (as opposed to $\approx \varepsilon \cdot \mu(G)$ like before), which means we may miss out on $\varepsilon \cdot n$ edges of the maximum matching.

2 Preliminaries

Matching Notation. Let $N_E(v)$ stand for the edges neighbouring vertex v in E. A fractional matching f of a graph G is an assignment of the edges of G to values in the range $[0, 1]$ such that for all vertices $v \in V$ it holds that $\sum_{e \in N_E(v)} f_e \leq 1$. The *size* of a fractional matching is simply the sum of the fractional values over its edges. That is a maximum fractional matching is the solution to the linear program $\max\{\sum_{e \in E} f_e : \sum_{e \in N_E(v)} f_e \leq 1 \text{ for all } v \in V, f \geq 0\}$. A solution x to the dual of this program $\min\{\sum_{v \in V} x_v : x_u + x_v \geq 1 \text{ for all } (u, v) \in E, x \geq 0\}$ is a fractional vertex cover.

[6] Readers may reasonable argue that the additive slack is not necessary as a number of vertex-sparsification techniques exist in literature allowing us to improve the approximation to purely multiplicative slack in the dynamic setting. Unfortunately, these techniques don't appear to be robust against vertex-wise updates.

Approximation with respect to a fractional matching is defined simmilarly as with respect to integral matchings. For a graph $G = (V, E)$ we use $\mu(G)$ to denote the size of the maximum matching in G. Likewise, we use $\mu^*(G)$ to denote the size of the maximum *fractional* matching. It is well-known that $\mu(G) \leq \mu^*(G) \leq \frac{3}{2}\mu(G)$ for any graph, and that $\mu^*(G) = \mu(G)$ in bipartite graphs.

▶ **Theorem 4** (Hopcroft-Karp [36]). *There exists a deterministic static algorithm which finds a $(1-\varepsilon)$-approximate maximum matching of a graph G on m edges in $O(m/\varepsilon)$ time.*

3 Incremental Approximate Matching

We start by showing our incremental fractional matching algorithm, and then show how to extend it (for bipartite graphs) to also maintain an integral matching.

3.1 Weighted EDCS & Fractional Matchings

In this section we show our algorithm to (almost[7]) maintain a weighted EDCS H in an incremental graph. It is well-known that such an H will be a $(1-\varepsilon)$-matching sparsifier on *bipartite* graphs, that is a "sparse" subgraph with $\mu(H) \geq (1-\varepsilon)\mu(G)$ [16].

As we show later in Section 5.1 (Theorem 13), we even known something stronger: there is an explicit fractional matching in H of size at least $(1-\varepsilon)\mu^*(G)$, defined as

$$f_{(u,v)} = \min\left(\frac{1}{\deg_H(v)}, \frac{1}{\deg_H(u)}\right) \text{ on each } (u,v) \in H. \tag{1}$$

Note that [17] similarly (implicitly) defines a large fractional matching on the support of a weighted EDCS, however our construction and analysis are arguably simpler. This fractional matching is also valid for general (non-bipartite) graphs. Hence our incremental algorithm will also maintain this explicit $(1-\varepsilon)$-approximate fractional matching (even in non-bipartite graphs). Formally we prove the following theorem.

▶ **Theorem 5.** *For any $\varepsilon \in (0,1)$, there is an algorithm (Algorithm 1) that maintains a $(1-\varepsilon)$-approximate maximum* fractional *matching in an incremental graph in total update time $O(n/\varepsilon^6 + m/\varepsilon^5)$. Additionally, this fractional matching is always supported on a set of edges H of size $|H| \leq \Theta(\mu(G)/\varepsilon^2)$ and maximum degree $O(1/\varepsilon^2)$.*

First we need two standard facts about weighted EDCS. For completeness, we prove these in Section 7. Lemma 7 has only been shown before for unweighted EDCS [5, 13, 16] and not weighted (but the arguments are very similar).

▶ **Lemma 6.** *In a β-WEDCS H, the maximum degree is at most β and $|H| \leq \beta\mu^*(G)$.*

▶ **Lemma 7.** *If a multiset of edges H is only ever changed by removing overfull edges and adding underfull edges, then there are at most $\beta^2\mu^*(G)$ such insertions/deletions to H.*

[7] As we will see later in this section, our sparsifier H will be a weighted EDCS for $G \setminus R$, where R is a subgraph of G with very small maximum matching size $\mu(R) = O(\varepsilon\mu(G))$.

Overview. Our algorithm (see Algorithm 1) will maintain a weighted EDCS H with $\beta = \Theta(1/\varepsilon^2)$. We also maintain the $(1-\varepsilon)$-approximate fractional matching f as in Equation (1) and Theorem 13.

When we get an edge-insertions (u, v), we need to reestablish the property that H is an EDCS. If (u, v) is underfull ($\deg_H(u) + \deg_H(v) < \beta - 1$), we add it (maybe multiple times) to H. This means that $\deg_H(u)$ (similarly $\deg_H(v)$) increases, which can potentially make some incident edge $(u, w) \in H$ overfull ($\deg_H(u) + \deg_H(w) > \beta$), so we must remove one such edge. This might in turn lead to some edge $(w, z) \in E$ being underfull (as now $\deg_H(w)$ decreased), so we add this edge to H. This process continues, so both from u and v we need to search for alternating paths of underfull and overfull edges (as is standard in EDCS-based algorithms). In total, Lemma 7 says there are $O(n/\varepsilon^4)$ updates to H over the full run of the algorithm.

We note that searching for an *overfull* edge is cheap: the maximum degree in H is just $O(\beta)$ (Lemma 6), so we can afford to, in $\Theta(1/\varepsilon^2)$ time, check all incident edges. However, searching for *underfull* edges is more expensive: this time we cannot afford to just go through all neighboring edges in E, as we no longer have a bound on the maximum degree.

To overcome this we use an amortization trick which allows us to ignore a vertex if we touched it too many times. There are only $\beta^2\mu^*(G)$ updates to H in total (Lemma 7), so there will only be $\varepsilon\mu^*(G)$ many vertices incident to more than $2\beta^2/\varepsilon$ of these updates. Any edges incident to these "update-heavy" vertices we may ignore, as this may only decrease the maximum matching size by an ε fraction. We thus only need to check each edge a total of $O(1/\varepsilon^5)$ times over the run of the algorithm, except when it is in H already. Note that H is no longer a weighted EDCS of $G = (V, E)$, but rather of $G' = (V, (E \setminus R) \cup H)$ where R is this set of edges we ignored (with $\mu^*(R) \leq \varepsilon\mu^*(G)$).

Algorithm 1 Incremental Weighted EDCS & Frasctional Matching.

```
// Initially E = H = ∅ and deg_H(v) = visits[v] = 0 for all v ∈ V.
// When an edge insertion e appears, add it to E and call FixEdge(e).

function FixEdge(e = (u, v))
    if deg_H(u) + deg_H(v) > β and (u, v) ∈ H then                  // overfull
        Remove (one copy of) the edge (u, v) from H
    if deg_H(u) + deg_H(v) < β − 1 then                             // underfull
        Add (one copy of) the edge (u, v) to H
    if the edge was added or removed then
        Update deg_H(u), deg_H(v), and the fractional matching accordingly
        FixVertex(u), FixVertex(v)

function FixVertex(v)
    visits[v] ← visits[v] + 1
    if visits[v] < 2β²/ε then
        for edge e ∈ E incident to v do
            FixEdge(e)
    else
        for edge e ∈ H incident to v do
            FixEdge(e)
```

Running Time. We analyse the total update time spent in different parts of our algorithm.

- We first note that `FixEdge` runs in constant time whenever it does not update H. It is called once for each edge-insertion (total $O(m)$ times), and also some number of times from `FixVertex`.
- Now consider the case when `FixEdge` does update H (which happens at most $\beta^2\mu^*(G)$ times per Lemma 7). Now the algorithm uses $O(\beta)$ time for the update of the fractional matching and insertion/removal in H, in addition to exactly two calls to `FixVertex`. Except for these calls to `FixVertex`, over the run of the algorithm we hence spend a total of $O(\mu(G)\beta^3) = O(n/\varepsilon^6)$ time.
- By the previous point, we will call `FixVertex` at most $2\beta^2\mu^*(G)$ times. In each call, we either loop through all incident edges in H or E. If we loop through H, we visit β many edges (by Lemma 6). Either these `FixEdge` calls take constant time, or they are already accounted for in the previous point. In total, this thus accounts for another $O(\mu(G)\beta^3) = O(n/\varepsilon^6)$ running time.
- We account for the case when `FixVertex` loops through all incident edges in E differently. Consider how often a specific edge e appears in the for-loop at line 13. Each endpoint vertex of e will reach this line at most $O(\beta^2/\varepsilon)$ times. Hence, in total for all edges, line 13 is run at most $O(m\beta^2/\varepsilon) = O(m/\varepsilon^5)$ times.

Approximation Guarantee. We now argue the approximation ratio. We will show that the fractional matching supported on H is a $(1-2\varepsilon)$-approximation of maximum fractional matching in G. If one want a $(1-\varepsilon')$-approximation, then one can run the algorithm in the same asymptotic update time setting $\varepsilon = \varepsilon'/2$, and changing β accordingly.

Define R_V to be the set of "dirty"/"update-heavy" vertices: that is vertices v for which `FixVertex(`v`)` has been called at least $2\beta^2/\varepsilon$ many times (i.e. $\mathsf{visits}[v] \geq 2\beta^2/\varepsilon$). By a counting argument $|R_V| \leq 2\beta^2\mu^*(G)/(2\beta^2/\varepsilon) = \varepsilon\mu^*(G)$, since by Lemma 7 in total there are only $\beta^2\mu^*(G)$ many updates to H, each issuing exactly two calls to `FixVertex`. If R_E is the set of edges incident to R_V, then $\mu^*(R_E) \leq |R_V| \leq \varepsilon\mu^*(G)$ since R_V is a vertex cover of R_E.

Define $G' = (V, E \setminus (R_E \setminus H))$. By the above, $\mu^*(G') \geq (1-\varepsilon)\mu^*(G)$. We will finish the proof by arguing that H is a weighted EDCS of G', and thus that our fractional matching is of value at least $(1-\varepsilon)\mu^*(G') \geq (1-\varepsilon)^2\mu^*(G) \geq (1-2\varepsilon)\mu^*(G)$. Whenever an edge is added or removed to H, we call `FixVertex` on its endpoints, and no other degrees $\deg_H$ have changes. Every time `FixVertex`(v) is called for $v \notin R_V$, we make sure that all edges $e \in E$ incident to it satisfy the definition of an EDCS, and when `FixVertex(`v`)` is called for some $v \in R_v$, we check the edges incident to H. We note that when a vertex becomes "update-heavy" (added to R_v), then we do **not** immediately remove all incident edges from H (as then we no longer have the same bound on the number of updates to H since Lemma 7 no longer applies).

▶ Remark 8. We note that our algorithm runs in time $O(n\beta^3 + m\beta^2/\varepsilon)$, and a valid question is whether setting $\beta = \Theta(1/\varepsilon^2)$ is actually necessary? Recently it was shown that for unweighted EDCS $\beta = \Theta(1/\varepsilon)$ is enough [7]. However, for weighted EDCS the ε^2 dependency is indeed necessary, as we show by an example where this is asymptotically tight in Section 5.2 (Theorem 16).

3.2 Integral Matchings in Bipartite Graphs

In this section we argue how to extend our fractional matching algorithm (Theorem 5) to maintain an integral matching instead (for bipartite graphs), in the same asymptotic update time. We cannot use known dynamic rounding techniques [23, 47], since all these incur

polylog(n) factors or require randomization, and we are aiming for update time independent of n. In fact, our technique is simple and combinatorial; and only relies on the standard Hopcroft-Karp algorithm for finding a $(1-\varepsilon)$-approximate matching in the static setting [36].

▶ **Theorem 1.** *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1-\varepsilon)$-approximate maximum matching of a bipartite graph undergoing edge insertions in total update time $O(n/\varepsilon^6 + m/\varepsilon^5)$.*

▶ Remark 9. Before proving the above theorem, we briefly explain how to achieve a slightly less efficient version (amortized $O(1/\varepsilon^8)$ update time) by using the fully dynamic $(1-\varepsilon)$-approximate matching algorithm of Gupta-Peng [34] as a black box. The idea is to run the fully dynamic algorithm on our sparsifier – the weighted EDCS H. This way we maintain a matching M of size $|M| \geq (1-\varepsilon)\mu(H) \geq (1-\varepsilon)^2\mu(G) \geq (1-2\varepsilon)\mu(G)$.

Gupta-Peng [34] state that their algorithm runs in $O(\sqrt{m}/\varepsilon^2)$ time per update. However, as previously pointed out by e.g. [16, Lemma 1], it is in fact more efficient when the max-degree Δ is low, in which case the update time is only $O(\Delta/\varepsilon^2)$. Since H always has max-degree $\beta = \Theta(1/\varepsilon^2)$, we can maintain the integral matching M in $O(1/\varepsilon^4)$ time *per update to* H. Over the run of the algorithm, we only perform $O(\mu(G)/\varepsilon^4)$ updates to H (see Lemma 7), hence the total additional update time spent maintaining the integral matching will be $O(n/\varepsilon^8)$.

Proof of Theorem 1. To prove Theorem 1, we need a slightly more refined analysis than the one above. We still run our incremental algorithm (Algorithm 1 and Theorem 5) to maintain a weighted EDCS H together with a fractional matching supported on H. Similarly to above, we additionally maintain an $(1-\varepsilon)$-approximate (integral) matching M of H.

The main idea of the fully dynamic algorithm of Gupta-Peng [34] is to lazily recompute (in $O(|H|/\varepsilon)$ time via Hoproft-Karp Theorem 4) M every $\approx \varepsilon\mu$ updates to H (indeed, during this few updates, the matching size cannot change its value by more than $\varepsilon\mu$). There are two observations which helps us to do better:

- **(i)** The graph G (but not the sparsifier H) is incremental, so $\mu(G)$ can only grow.
- **(ii)** We know a good estimate of $\mu(G)$, namely the size of our fractional matching. Denote by $\tilde{\mu}$ the value of the maintained fractional matching, so that $(1-\varepsilon)\mu(G) \leq \tilde{\mu} \leq \mu(G)$.

The above two observations mean that we only need to recompute the matching M whenever $\mu(G)$ actually have increased significantly (namely by a $(1+\Theta(\varepsilon))$-factor), and not just every $\varepsilon\mu$ updates.

Formally, whenever $|M| \geq (1-\varepsilon)^2\tilde{\mu}$ we know that M is still a $(1-3\varepsilon)$-approximation since then $|M| \geq (1-\varepsilon)^2\tilde{\mu} \geq (1-\varepsilon)^3\mu(G) \geq (1-3\varepsilon)\mu(G)$. Conversely, whenever $|M| < (1-\varepsilon)^2\tilde{\mu}$, we recompute M in time $O(|H|/\varepsilon)$ (Theorem 4) so that it is a $(1-\varepsilon)$-approximation of the maximum matching in H (and thus also a $(1-2\varepsilon)$-approximation of the maximum matching in G).

Let us now bound the total time spent recomputing M. Let $M_1, M_2, \ldots, M_t$ be the different approximate matchings we compute during the run of the algorithm. We first note that at the time when we compute M_{i+1}:

$$|M_i| \leq (1-\varepsilon)^2\tilde{\mu} \leq (1-\varepsilon)\left((1-\varepsilon)\mu(H)\right) \leq (1-\varepsilon)|M_{i+1}| \tag{2}$$

This in turn means that $|M_i| \leq (1-\varepsilon)^{t-i}\, n$ (since $|M_t| \leq n$), and hence that $\sum_{i=1}^{t} |M_i| \leq n\sum_{i=0}^{\infty}(1-\varepsilon)^i \leq n/\varepsilon$, by a geometric sum.

Finally we note that we spend $O(|M_i|/\varepsilon^3)$ time in order to compute M_i. Indeed, when we compute M_i, we did so in $O(|H|/\varepsilon)$ time, and $|H| = O(\mu(G)/\varepsilon^2)$ by Lemma 6. This means that in total, over the run of the algorithm, we spend $O(\sum |M_i|/\varepsilon^3) = O(n/\varepsilon^4)$ time

maintaining the integral matchings M_i. This is in addition to the time spent maintaining the weighted EDCS H and the fractional matching (see Theorem 5). This concludes the proof of Theorem 1. ◀

4 Vertex Deletions

In this section we will observe that our algorithm can also handle vertex deletions (simultaneously to handling edge insertions) in similar total update time. However, this comes with one caveat: we instead get additive approximation error proportional to εn (that is we maintain a matching of size $\mu^*(G) - \varepsilon n$, instead of $\mu^*(G) - \varepsilon\mu^*(G)$ as before).

▶ **Theorem 10.** *For any $\varepsilon \in (0,1)$, there is an algorithm that maintains a* fractional *matching of size at least $\mu^*(G) - \varepsilon n$ in an graph G undergoing edge insertions and vertex deletions. The total update time is $O(n/\varepsilon^6 + m/\varepsilon^5)$.*

Proof. The algorithm (Algorithm 1) remains the same as in Section 3.1. When a vertex is deleted, we simply remove all it's incident edges from E and H, and call `FixVertex` on all neighboring vertices (in H) who now changed their degree. The only thing which changes in the analysis is the $\beta^2\mu^*(G)$-bound on the number of updates to H (Lemma 7), which no longer applies. However, we can still get a weaker version of Lemma 7 with a $\beta^2 n$ total update bound instead:

▶ **Lemma 11.** *If a multiset of edges H is only ever changed by (i) removing overfull edges, (ii) adding underfull edges, and (iii)* removing all edges incident to a vertex when no edges are underfull or overfull*, then there are at most $3\beta^2 n$ insertions/deletions to H.*

Given Lemma 11 (which we prove in Section 7), we see that the running time analysis of Algorithm 1 can remain exactly the same! In the approximation guarantee analysis, we now have more "update-heavy" vertices $|R_V| \leq 6\beta^2 n/(2\beta^2/\varepsilon) = 3\varepsilon n$, which is why we now can lose up to $O(n\varepsilon)$ edges from the matching. Otherwise, the approximation guarantee analysis remains the same, and so does the rest of the analysis of the algorithm. ◀

Rounding in Bipartite Graphs. Similar as in Section 3.2, we can round the fractional matching to an integral one in bipartite graphs, also while supporting edge-insertions and vertex-deletions simultaneously.

▶ **Theorem 2.** *There exists a deterministic dynamic algorithm which for arbitrary small constant $\varepsilon > 0$ maintains a $(1, \epsilon \cdot n)$-approximate maximum matching of a bipartite graph undergoing edge insertions and vertex deletions in total update time of $O(n/\varepsilon^8 + m/\varepsilon^5)$.*

Proof. Unlike in Section 3.2, we cannot argue that $\mu(G)$ is increasing when we have vertex deletions. So instead we resort to the Gupta-Peng [34] framework discussed in Remark 9 (together with Lemma 11), which has the additional cost of $O(n/\varepsilon^8)$ total update time to maintain an approximate integral matching on our sparsifier H. ◀

▶ Remark 12. We note that normal vertex-sparsification techniques (such as the one shown in [38] against oblivious adversaries) do not apply here in order to assume $n = \tilde{\Theta}(\mu(G))$ so that the additive error becomes multiplicative again. This is because vertex deletions in the original graph might become edge deletions in the vertex-sparsified graph. We also note that one can achieve similar guarantees of supporting vertex deletions with additive εn slack using *any* edge-incremental algorithm (also for non-bipartite graphs) as a black-box: see Section 6 for a discussion on how this can be done. The general approach in Section 6 will give worse dependency on $1/\varepsilon$ (for dense graphs), compared to Theorems 2 and 10 above.

5 Tight Bounds of the of Approximation Ratio of a Weighted EDCS

5.1 Explicit Fractional Matching

In this section, we give explicit formulas only based on the degrees in H, for a $(1-\varepsilon)$-approximate fractional matching. We prove this by also providing an explicit approximate fractional vertex cover, and showing that these satisfy approximate complimentary slackness. This also significantly simplifies the previous proof [16] that H is $(1-\varepsilon)$-matching sparsifier in bipartite graphs.

▶ **Theorem 13.** *Suppose H is a weighted EDCS of a graph G, with parameter $\beta \geq 25/\varepsilon^2$. Let $f_{(u,v)} = \min\{\frac{1}{\deg_H(v)}, \frac{1}{\deg_H(u)}\}$ for each[8] $(u,v) \in H$. Then f is a $(1-\epsilon)$-approximate fractional matching of G.*

Define $r_v := \deg_H(v) - \frac{\beta-1}{2}$ for a vertex $v \in H$. We define the fractional matching f as in the statement of the theorem, together with the fractional vertex cover x:

$$f_{(u,v)} = \min\left(\frac{1}{\deg_H(u)}, \frac{1}{\deg_H(v)}\right) \quad \text{for edge } (u,v) \in H \tag{3}$$

$$x_v = \begin{cases} \min(1, \frac{1}{2} + \frac{r_v^2}{\beta}) & \text{if } r_v \geq 0 \\ \max(0, \frac{1}{2} - \frac{r_v^2}{\beta}) & \text{if } r_v < 0 \end{cases} \tag{4}$$

It is now relatively straightforward (albeit a bit calculation-heavy) to argue that f and x are indeed feasible solutions and that they satisfy approximate complimentary slackness.

▷ Claim 14. Our f is a fractional matching and our x is a fractional vertex cover of G.

Proof. Our f is feasible since no vertex v is overloaded by the matching: at most $\deg_H(v)$ many incident edges to v contribute at most $1/\deg_H(v)$ each.

To argue that x is feasible, consider some edge $(u,v) \in E$. Without loss of generality we may assume that $\deg_H(v) \geq \deg_H(u)$ and $\deg_H(v) \geq \frac{\beta-1}{2}$, i.e. $r_v \geq r_u$ and $r_v \geq 0$ (since $\deg_H(u) + \deg_H(v) \geq \beta - 1$ as H is a weighted EDCS). If $r_v^2 \geq \beta/2$, $x_v = 1$ so (u,v) is covered. In the case $r_v^2 < \beta/2$, we instead have $x_v = \frac{1}{2} + \frac{r_v^2}{\beta}$. It is always the case that $x_u \geq \frac{1}{2} - \frac{r_u^2}{\beta}$. Additionally we note that $r_u + r_v \geq 0$ (so $r_u^2 \leq r_v^2$) since $\deg_H(u) + \deg_H(v) \geq \beta - 1$, so we conclude that $x_u + x_v \geq 1$, and hence that (u,v) is covered. ◁

▷ Claim 15. The fractional matching f and fractional vertex cover x satisfy $(1 - \frac{3}{\sqrt{\beta}}, 1 + \frac{2}{\sqrt{\beta}} + \frac{2}{\beta})$-approximate complementary slackness[9]; in particular:

- **(i)** Whenever $f_{(u,v)} > 0$, then $x_u + x_v \leq 1 + \frac{2}{\sqrt{\beta}} + \frac{1}{\beta}$.
- **(ii)** Whenever $x_v > 0$, then $\sum_{u:(u,v)\in H} f_{(u,v)} \geq 1 - \frac{4}{\sqrt{\beta}}$.

Proof. We verify (i) and (ii).

- **(i)** Suppose $f_{(u,v)} > 0$, then $(u,v) \in H$, so $\deg_H(u) + \deg_H(v) \in \{\beta - 1, \beta\}$. This means that $0 \leq r_v + r_u \leq 1$. If both r_v and r_u are non-negative, we have that $x_u + x_v \leq 2(\frac{1}{2} + \frac{1^2}{\beta}) \leq 1 + \frac{2}{\beta}$. Now, without loss of generality $r_u < 0 \leq r_v$. In case

[8] We note that edges e appearing multiple time in H all contribute towards f_e: if e appears ϕ_e times in H, the value of f_e is naturally scaled by ϕ_e.

[9] For completeness, we define the approximate complimentary slackness conditions in Section 7 and prove them in Lemma 20.

$r_u^2 \geq \beta/2$, we know $x_u = 0$ so $x_u + x_v \leq 1$. In the case when $r_u^2 < \beta/2$, we know $x_u = \frac{1}{2} + \frac{r_u^2}{\beta}$ and $x_v \leq \frac{1}{2} + \frac{r_v^2}{\beta}$. Since $r_u + r_v \leq 1$, we known that $r_v \leq |r_u| + 1$. Concluding:

$$x_v + x_u \leq 1 + \frac{(|r_u|+1)^2 - r_u^2}{\beta} = 1 + \frac{2|r_u|+1}{\beta} < 1 + \frac{2\sqrt{\beta}+1}{\beta} = 1 + \frac{2}{\sqrt{\beta}} + \frac{1}{\beta}.$$

(ii) Suppose $x_v > 0$. Hence $r_v > -\sqrt{\beta/2}$ (else $x_v = 0$), that is $\deg_H(v) > \frac{\beta-1}{2} - \sqrt{\beta/2}$. For any incident edge $(u,v) \in H$, we must have $\deg_H(v) + \deg_H(u) \leq \beta$, so $\deg_H(u) \leq \frac{\beta-1}{2} + (1 + \sqrt{\beta/2})$. Now, we see that we assign weight at least $1/(\frac{\beta-1}{2} + (1+\sqrt{\beta/2}))$ to the edge (u,v) in f. Since this holds for all the $\deg_H(v) > \frac{\beta-1}{2} - \sqrt{\beta/2}$ incident edges we know that v will in total receive, from the fractional matching f, at least:

$$\sum_{u:(u,v)\in H} f_{(u,v)} \geq \frac{\frac{\beta-1}{2} - \sqrt{\beta/2}}{\frac{\beta-1}{2} + 1 + \sqrt{\beta/2}} = 1 - \frac{\sqrt{8\beta}+2}{\beta + \sqrt{2\beta}+1} \geq 1 - \frac{3}{\sqrt{\beta}}.$$ ◀

Proof of Theorem 13. By the above claims and approximate complimentary slackness (see Lemma 20 in Section 7) we know that $(1 - \frac{3}{\sqrt{\beta}})|x| \leq (1 + \frac{2}{\sqrt{\beta}} + \frac{2}{\beta})|f|$. Since $(1 - \frac{3}{\sqrt{\beta}})/(1 + \frac{2}{\sqrt{\beta}} + \frac{2}{\beta}) > 1 - \frac{5}{\sqrt{\beta}}$, we get that $|f| \geq (1 - \frac{5}{\sqrt{\beta}})|x| \geq (1-\varepsilon)|x| \geq (1-\varepsilon)\mu^*(G)$ whenever $\beta \geq 25/\varepsilon^2$. ◀

5.2 Lower Bound

In this section we show that Theorem 13 is tight up to a constant, i.e. that one must set $\beta = \Theta(1/\varepsilon^2)$ in order to guarantee that a weighted EDCS preserves a $(1-\varepsilon)$-fraction of the matching. This might be a bit surprising, considering that for the *unweighted* EDCS, it is known that $\beta = \Theta(1/\varepsilon)$ suffices (to preserve a $(\frac{2}{3} - \varepsilon)$-approximation to the matching [7]).

▶ **Theorem 16.** *For any $\beta \geq 2$, there exists a (bipartite) graph G together with a weighted EDCS H for which $\mu(H) = (1 - \Theta(1/\sqrt{\beta}))\mu(G)$.*

We show our construction in Figure 1, and also describe it here formally in words. For simplicity, we will assume that $\beta = 2\gamma^2$ for some integer γ (but it is not difficult to adapt the proof for when β is not twice a square). In our construction, each edge appears at most once in H, and all edges $e \in H$ have $\deg_H(e) = \beta$; all edges $e \in E \setminus H$ have $\deg_H(e) = \beta - 1$.

Define the gadget $G_i = (S_i, L_i, E_i)$ to be a complete bipartite graph in which $|S_i| = \gamma^2 + i$ and $|L_i| = \gamma^2 - i$ (S is for vertices with *small* degree, and L for vertices with *large* degree). The subgraph H will consist of many of these gadgets G_i, so we start by noting a few properties about them. Firstly, vertices in L_i have degree $\frac{\beta}{2} + i$ while those in S_i have degree $\frac{\beta}{2} - i$. This means that any edge in $(u,v) \in E_i$ has degree exactly $\deg_{G_i}(u) + \deg_{G_i}(v) = \beta$. We also note that the maximum matching inside G_i is of size $|L_i| = \gamma^2 - i$.

Subgraph H. We begin by describing how the weighted EDCS H looks like, and later we will define what additional edges are also in the full graph G. The subgraph H will exactly consist of:

- One copy of G_0.
- Two copies each of $G_1, G_2, \ldots, G_{\gamma-1}$. Call the copies $G_i^{(1)}$ and $G_i^{(2)}$.
- 2γ copies of G_γ. Call the copies $G_\gamma^{(1,j)}$ and $G_\gamma^{(2,j)}$ for $j = 1, 2, \ldots, \gamma$.

▷ Claim 17. $\mu(H) = 4\gamma^3 - 4\gamma^2 + \gamma$.

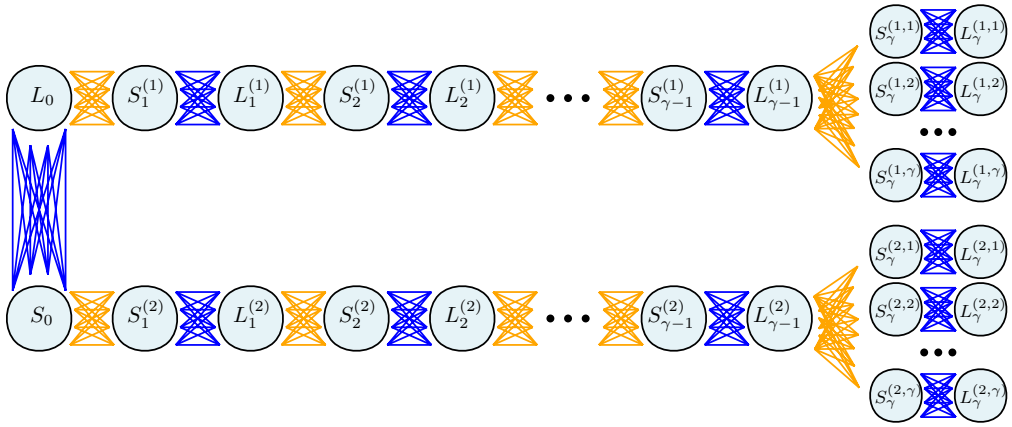

Figure 1 The lower bound construction. The blue edges are part of H, while the yellow are not. We have $\gamma = \sqrt{\beta/2}$, and each set S_i, L_i indicates an independent set of vertices of size $|S_i| = \gamma^2 + i$ and $|L_i| = \gamma^2 - i$ (so in H they have degrees $\frac{\beta}{2} - i$ and $\frac{\beta}{2} + i$ respectively). The maximum matching in H matches all vertices in L_i to the corresponding S_i. The maximum matching in G however, matches L_i to S_{i+1} (and S_0 to $S_1^{(2)}$), in addition to L_γ which can also be matched to S_γ.

Proof. Since the maximum matching size in G_i is $|L_i| = \gamma^2 - i$ we get:

$$\begin{aligned}\mu(H) &= |L_0| + 2\gamma|L_\gamma| + 2(|L_1| + |L_2| + \cdots + |L_{\gamma-1}|) \\ &= \gamma^2 + 2\gamma(\gamma^2 - \gamma) + 2\sum_{i=1}^{\gamma-1}(\gamma^2 - i) \\ &= 4\gamma^3 - 4\gamma^2 + \gamma\end{aligned}$$

◁

Full graph G. Now we describe the additional edges which are part of G but not already in H (see also Figure 1):

- For $k \in \{1, 2\}$, we connect $G_1^{(k)}, G_2^{(k)}, \ldots, G_{\gamma-1}^{(k)}$ in a chain as follows: every pair (u, v) with $u \in L_i^{(k)}$ and $v \in S_{i+1}^{(k)}$ is an edge (so that the induced subgraph on these two sets of vertices forms a complete bipartite graph).
- At the end of these two chains, we connect all the gadgets $G_\gamma^{(k,j)}$ as follows: every pair (u, v) with $u \in L_{\gamma-1}^{(k)}$ and $v \in S_\gamma^{(k,j)}$ for some j, is an edge.
- Finally we connect these two chains using $G_0 = (S_0, L_0, E_0)$ as follows: every pair (u, v) with $u \in L_0$ and $v \in S_1^{(1)}$ is an edge; and every pair (u, v) with $u \in S_0$ and $v \in S_1^{(2)}$ is an edge.

We note that G is bipartite and all above edges have degree exactly $\deg_H(u) + \deg_H(v) = \beta - 1$, so indeed H is a weighted EDCS of G.

▷ Claim 18. $\mu(G) \geq 4\gamma^3 - 3\gamma^2 + \gamma$.

Proof. We argue that a matching of this size exists in G. In fact the only edges of H we will use as part of this matching are those in the gadgets $G_\gamma^{(k,j)}$.

- We pick a matching between $L_i^{(k)}$ and $S_{i+1}^{(k)}$ of size $|L_i^{(k)}| = \gamma - i$ for all $k \in \{1, 2\}$ and $i = 1, 2, \ldots, \gamma - 2$.
- In $G_\gamma^{(k,j)}$ we pick a matching of size $|L_\gamma^{(k,j)}| = \gamma^2 - \gamma$. Note that exactly 2γ vertices in $S_\gamma^{(k,j)}$ are left unmatched.

- Denote by $U^{(k)}$ the set of unmatched vertices in $S_\gamma^{(k,1)}, S_\gamma^{(k,2)}, \ldots, S_\gamma^{(k,\gamma)}$, for $k \in \{1,2\}$. Note that $|U^{(k)}| = 2\gamma^2$ and that $(L_{\gamma-1}^{(k)}, U^{(k)})$ forms a complete bipartite graph, so we pick a matching of size $|L_{\gamma-1}^{(k)}| = \gamma^2 - \gamma + 1$ from there.
- Finally we pick matchings between L_0 and $S_1^{(1)}$ (respectively S_0 and $S_1^{(2)}$) of size $|L_0| = \gamma$ (respectively $|S_0| = \gamma$).

In total we see that the above matching is exactly $|S_0| = \gamma$ larger than in Claim 17, which concludes the proof of the claim. ◁

Approximation ratio. To conclude the proof of Theorem 16, we see that $\mu(G) - \mu(H) = \gamma^2 \geq \frac{1}{4\gamma}\mu(G)$ whenever $\gamma \geq 1$. Hence H preserves at most a $(1 - \frac{1}{4\gamma}) = (1 - \frac{1}{2\sqrt{2\beta}})$ fraction of the maximum matching of G.

6 Black-Box Vertex Deletions

Here we briefly explain how one can convert any incremental $(1-\varepsilon)$-approximate maximum matching algorithm to also support vertex deletions, if allowing additive εn approximation instead, in a black-box fashion. The reduction is simple and also works in general (non-bipartite) graphs. Hence, as an immediate application, we can get a $(1, \varepsilon n)$-approximate matching algorithm for general graphs undergoing both edge-insertions and vertex-deletions, with amortized $1/\varepsilon^{O(1/\varepsilon)}$ update time, if using the incremental algorithm of [31].

▶ **Lemma 19.** *Suppose $\mathcal{A}$ is an algorithm which maintains a $(1-\varepsilon/2)$-approximate maximum matching for a graph undergoing edge insertions, running in total time T. Then there exists an algorithm which maintains a $(1, \varepsilon n)$-approximate matching, in total time $O(T/\varepsilon)$, on a graph undergoing both edge insertions and vertex deletions.*

Proof. We run $\mathcal{A}$, and whenever we get a vertex deletion we ignore it and keep the vertex in the graph. In the outputted matching from the algorithm we remove any edges incident to deleted vertices. When $\varepsilon n/2$ vertices have been deleted, we actually delete them from the graph and rerun the algorithm from scratch (starting on the empty graph). This will only happen $\frac{2}{\varepsilon}$ times, which is the running time blow-up. At each point, the algorithm maintains a matching of size at least $\mu - (\varepsilon n/2 + \varepsilon n/2) = \mu - \varepsilon n$, since only one edge can be removed per deleted vertex still remaining in the graph. ◀

7 Omitted Proofs

EDCS properties

▶ **Lemma 6.** *In a β-WEDCS H, the maximum degree is at most β and $|H| \leq \beta\mu^*(G)$.*

Proof. If a vertex u has $\deg_H(u) > \beta$, then any incident edge $(u,v) \in H$ is overfull: $\deg_H(u) + \deg_H(v) > \beta$, leading to a contradiction. Hence the maximum degree is at most β. Now we construct a fractional matching by assigning a weight of $1/\beta$ to every edge in H (so an edge appearing with multiplicity ϕ in H gets weight ϕ/β). Clearly this is a feasible fractional matching of G, since no vertex is overloaded. On the other hand, the size of this fractional matching is $|H|/\beta$, implying that $|H| \leq \beta\mu^*(G)$. ◀

▶ **Lemma 7.** *If a multiset of edges H is only ever changed by removing overfull edges and adding underfull edges, then there are at most $\beta^2\mu^*(G)$ such insertions/deletions to H.*

Proof. We use a potential function argument. Define

$$\Phi(H) := |H|(2\beta-1) - \sum_{(u,v)\in H} (\deg_H(v)+\deg_H(u)) = \sum_{(u,v)\in H} (2\beta-1-\deg_H(u)-\deg_H(v))$$

We note that if an edge (u,v) appears multiple times in H, it appears multiple times in the above sum as well. We first note that $\Phi(\emptyset)=0$ and $\Phi(H) \leq \beta|H| \leq \beta^2\mu^*(G)$, by Lemma 6 and since $(2\beta-1-\deg_H(u)-\deg_H(v)) \leq \beta$ when H is a valid EDCS. Now we verify that updates to H increase the potential by at least 1:

- Insertion of an underfull edge $(u,v) \in E$.
  That is $\deg_H(u)+\deg_H(v) \leq \beta-2$ before adding the edge. The term $|H|(2\beta-1)$ will increase by $2\beta-1$. $\sum_{(u,v)\in H}(\deg_H(v)+\deg_H(u))$ will increase by at most $2\beta-2$, since one term of value $\deg_H(v)+\deg_H(v) \leq \beta-2+2$ (the $+2$ comes from $\deg_H(v)$ and $\deg_H(u)$ increasing by one when we add (u,v) to H) is added, and at most $\beta-2$ other terms decrease in value by one.
- Deletion of an overfull edge $(u,v) \in H$.
  That is $\deg_H(u)+\deg_H(v) \geq \beta+1$ before removing the edge. The term $|H|(2\beta-1)$ will decrease by $2\beta-1$. $\sum_{(u,v)\in H}(\deg_H(v)+\deg_H(u))$ will decrease by at least 2β, since one term of value $\deg_H(v)+\deg_H(v) \geq \beta+1-2$ (the -2 comes from $\deg_H(v)$ and $\deg_H(u)$ decreasing when we remove (u,v)) is erased, and at least $\beta+1$ other terms increase in value by one. ◀

▶ **Lemma 11.** *If a multiset of edges H is only ever changed by (i) removing overfull edges, (ii) adding underfull edges, and (iii)* removing all edges incident to a vertex when no edges are underfull or overfull*, then there are at most $3\beta^2 n$ insertions/deletions to H.*

Proof. We continue the potential function argument from the proof of Lemma 7 above. When we delete, from H, all edges incident to some vertex u, we know that we deleted at most β many edges from H (as the degree of this vertex was at most β). For each such incident edge (u,v), we bound how much its deletion could have decreased the potential function. The $|H|(2\beta-1)$ term decreased by exactly $2\beta-1$, and the $-\sum_{(u,v)\in H}(\deg_H(u)+\deg_H(v))$ term can only increase. So the total decrease in potential, over all up to β incident edges which were deleted, is at most $2\beta^2-\beta$.

Since we can only delete up to n vertices in total, and the potential is always bounded by $\beta^2\mu^*(G) \leq \beta^2 n$, it follows that the total increase in the potential function, over the run of the algorithm, is at most $3\beta^2 n - n\beta$ (and thus this many updates to H from insertions/deletions of underfull/overfull edges). In total we deleted at most $n\beta$ edges in H incident to deleted vertices, so the total number of updates to H is thus bounded by $3\beta^2 n - \beta n + \beta n = 3\beta^2 n$. ◀

Approximate Complimentary Slackness

▶ **Lemma 20.** *Suppose we have the primal linear program $\max\{c^T x : Ax \leq b, x \geq 0\}$, and its dual $\min\{b^T y : A^T y \geq c, y \geq 0\}$. We say that feasible primal solution x and dual solution y satisfy (α,γ)-approximate complementary slackness (for $\alpha \leq 1 \leq \gamma$) if: (i) if $x_i = 0$ then $(A^T)_i y \leq \gamma c_i$, and (ii) if $y_j = 0$ then $(A)_j x \geq \alpha b_j$. When this is the case, then $\alpha b^T y \leq \gamma c^T x$ (i.e. x and y are $\frac{\gamma}{\alpha}$-approximate optimal).*

Proof. We see that $\gamma c^T x - \alpha b^T y = x^T(\gamma c - A^T y) + y^T(Ax - \alpha b)$. Now either $x_i = 0$ or $(\gamma c - A^T y)_i \geq 0$; and either $y_j = 0$ or $(Ax-\alpha b)_j \geq 0$. Hence $\gamma c^T x - \alpha b^T y \geq 0$. ◀

References

1 Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In *STOC*, pages 114–125. ACM, 2019. doi:10.1145/3313276.3316376.

2 Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In *ICALP*, volume 107 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.7.

3 Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *SODA*, pages 1616–1635. SIAM, 2019. doi:10.1137/1.9781611975482.98.

4 Sepehr Assadi and Soheil Behnezhad. Beating two-thirds for random-order streaming matching. In *ICALP*, volume 198 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.19.

5 Sepehr Assadi and Aaron Bernstein. Towards a unified theory of sparsification for matching problems. In *SOSA*, volume 69 of *OASIcs*, pages 11:1–11:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/OASICS.SOSA.2019.11.

6 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in O(log n) update time (corrected version). *SIAM J. Comput.*, 47(3):617–650, 2018. Announced at FOCS'11. doi:10.1137/16M1106158.

7 Soheil Behnezhad. Improved analysis of EDCS via gallai-edmonds decomposition. *CoRR*, abs/2110.05746, 2021.

8 Soheil Behnezhad. Dynamic algorithms for maximum matching size. In *SODA*, pages 129–162. SIAM, 2023. doi:10.1137/1.9781611977554.CH6.

9 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *FOCS*, pages 382–405. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00032.

10 Soheil Behnezhad and Sanjeev Khanna. New trade-offs for fully dynamic matching via hierarchical EDCS. In *SODA*, pages 3529–3566. SIAM, 2022. doi:10.1137/1.9781611977073.140.

11 Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. Fully dynamic matching: Beating 2-approximation in Δ^ϵ update time. In *SODA*, pages 2492–2508. SIAM, 2020. doi:10.1137/1.9781611975994.152.

12 Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. Sublinear time algorithms and complexity of approximate maximum matching. In *STOC*, pages 267–280. ACM, 2023. doi:10.1145/3564246.3585231.

13 Aaron Bernstein. Improved bounds for matching in random-order streams. In *ICALP*, volume 168 of *LIPIcs*, pages 12:1–12:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ICALP.2020.12.

14 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. *ACM Trans. Algorithms*, 17(4):29:1–29:51, 2021. Announced at SODA'19. doi:10.1145/3469833.

15 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In *FOCS*, pages 1123–1134. IEEE, 2020. doi:10.1109/FOCS46700.2020.00108.

16 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In *ICALP (1)*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015. doi:10.1007/978-3-662-47672-7_14.

17 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In *SODA*, pages 692–711. SIAM, 2016. doi:10.1137/1.9781611974331.CH50.

18 Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In *IPCO*, volume 10328 of *Lecture Notes in Computer Science*, pages 86–98. Springer, 2017. doi:10.1007/978-3-319-59250-3_8.

19 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM J. Comput.*, 47(3):859–887, 2018. Announced at SODA'15. doi:10.1137/140998925.

20 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *STOC*, pages 398–411. ACM, 2016. doi:10.1145/2897518.2897568.

21 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time. In *SODA*, pages 470–489. SIAM, 2017. doi:10.1137/1.9781611974782.30.

22 Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. In *ICALP*, volume 198 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.27.

23 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic algorithms for packing-covering lps via multiplicative weight updates. In *SODA*, pages 1–47. SIAM, 2023. doi:10.1137/1.9781611977554.CH1.

24 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Sublinear algorithms for $(1.5+\epsilon)$-approximate matching. In *STOC*, pages 254–266. ACM, 2023. doi:10.1145/3564246.3585252.

25 Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. In *SODA*, pages 100–128. SIAM, 2023. doi:10.1137/1.9781611977554.CH5.

26 Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $O(1/\epsilon^2)$ amortized update time. In *SODA*, pages 1872–1885. SIAM, 2019. doi:10.1137/1.9781611975482.113.

27 Joakim Blikstad and Peter Kiss. Incremental (1-(epsilon))-approximate dynamic matching in O(poly(1/(epsilon))) update time. *CoRR*, abs/2302.08432, 2023. doi:10.48550/arXiv.2302.08432.

28 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In *ICALP*, volume 107 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.33.

29 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *FOCS*, pages 612–623. IEEE, 2022. doi:10.1109/FOCS54457.2022.00064.

30 Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972. doi:10.1145/321694.321699.

31 Fabrizio Grandoni, Stefano Leonardi, Piotr Sankowski, Chris Schwiegelshohn, and Shay Solomon. $(1+\epsilon)$-approximate incremental matching in constant deterministic amortized time. In *SODA*, pages 1886–1898. SIAM, 2019. doi:10.1137/1.9781611975482.114.

32 Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzrad. Maintaining an EDCS in general graphs: Simpler, density-sensitive and with worst-case time bounds. In *SOSA*, pages 12–23. SIAM, 2022. doi:10.1137/1.9781611977066.2.

33 Manoj Gupta. Maintaining approximate maximum matching in an incremental bipartite graph in polylogarithmic update time. In *FSTTCS*, volume 29 of *LIPIcs*, pages 227–239. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICS.FSTTCS.2014.227.

34 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *FOCS*, pages 548–557. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.65.

35 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *STOC*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

36 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973. doi:10.1137/0202019.

37 Arun Jambulapati, Yujia Jin, Aaron Sidford, and Kevin Tian. Regularized box-simplex games and dynamic decremental bipartite matching. In *ICALP*, volume 229 of *LIPIcs*, pages 77:1–77:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ICALP.2022.77.

38 Peter Kiss. Deterministic dynamic matching in worst-case update time. In *ITCS*, volume 215 of *LIPIcs*, pages 94:1–94:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITCS.2022.94.

39 Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 1955.

40 Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *FOCS*, pages 248–255. IEEE Computer Society, 2004. doi:10.1109/FOCS.2004.40.

41 Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. *ACM Trans. Algorithms*, 12(1):7:1–7:15, 2016. Announced at STOC'13. doi:10.1145/2700206.

42 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC*, pages 457–464. ACM, 2010. doi:10.1145/1806689.1806753.

43 David Peleg and Shay Solomon. Dynamic $(1 + \epsilon)$-approximate matchings: A density-sensitive approach. In *SODA*, pages 712–729. SIAM, 2016. doi:10.1137/1.9781611974331.CH51.

44 Mohammad Roghani, Amin Saberi, and David Wajc. Beating the folklore algorithm for dynamic matching. In *ITCS*, volume 215 of *LIPIcs*, pages 111:1–111:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ITCS.2022.111.

45 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126. SIAM, 2007.

46 Shay Solomon. Fully dynamic maximal matching in constant update time. In *FOCS*, pages 325–334. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.43.

47 David Wajc. Rounding dynamic matchings against an adaptive adversary. In *STOC*, pages 194–207. ACM, 2020. doi:10.1145/3357713.3384258.

48 Da Wei Zheng and Monika Henzinger. Multiplicative auction algorithm for approximate maximum weight bipartite matching. In *IPCO*, volume 13904 of *Lecture Notes in Computer Science*, pages 453–465. Springer, 2023. doi:10.1007/978-3-031-32726-1_32.