Maintaining Approximate Maximum Matching in an Incremental Bipartite Graph in Polylogarithmic Update Time

Manoj Gupta*

Xerox Research, India
manoj.gupta@xerox.com, gmanoj@cse.iitd.ac.in

Abstract

A sparse subgraph G' of G is called a matching sparsifier if the size or weight of matching in G' is approximately equal to the size or weight of maximum matching in G. Recently, algorithms have been developed to find matching sparsifiers in a static bipartite graph. In this paper, we show that we can find matching sparsifier even in an incremental bipartite graph.

This observation leads to following results:

1. We design an algorithm that maintains a $(1+\epsilon)$ approximate matching in an incremental bipartite graph in $O(\frac{\log^2 n}{\epsilon^4})$ update time.
2. For weighted graphs, we design an algorithm that maintains $(1+\epsilon)$ approximate weighted matching in $O(\frac{\log n \log(nN)}{\epsilon^4})$ update time where N is the maximum weight of any edge in the graph.

1998 ACM Subject Classification E.1 [Data Structures]: Graphs and Networks, F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems, G.2.2 [Graph Theory]: Graph Algorithms

Keywords and phrases Graph Algorithm, Dynamic Graph

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.227

1 Introduction

A matching is a set of vertex disjoint edges in the graph. Finding a matching of maximum size in a graph is one of the most important question in combinatorial optimization. For static graph, Hopcroft and Karp [12] designed an algorithm that found maximum matching in a bipartite graph in $O(m\sqrt{n})$ time. However, extending this result to general graph turned out to be a challenging problem. Micali and Vazirani [14, 18] were the first to show that a maximum matching in a general graph can be found in $O(m\sqrt{n})$ time.

In recent years, there has been a lot of activity for maintaining approximate/exact matching in a *dynamic* graph. In a dynamic graph, at each update step an edge is added or deleted from the graph. If only insertions are allowed, then the graph is said to be an *incremental* dynamic graph. If only deletions are allowed, then the graph is said to be a *decremental* dynamic graph. If both insertions and deletions are allowed, then the graph is said to be *fully* dynamic graph.

For the analysis, we assume that an adversary executes a sequence of addition and deletion of edges in a graph with the objective of maximizing the update time of a given algorithm. An adversary is *oblivious* if he/she knows the code of the algorithm but does

* The work was done when the author was a student at IIT Delhi.

34th Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2014).
Editors: Venkatesh Raman and S. P. Suresh; pp. 227–239
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

not have access to the random bits used in the algorithm. In the literature, there are some randomized algorithm that assumes an *oblivious* adversary which means that the adversary has no knowledge of the matched edges maintained by the algorithm. If the algorithm is *deterministic*, then the adversary can run this algorithm on an input sequence and find the matched edges at each update step of the algorithm. In the following literature survey, all the randomized algorithm assume *oblivious* adversary model.

Ivković and Llyod[13] were the first to investigate matching in dynamic graphs. They designed a deterministic algorithm that maintains a maximal matching with $O((n+m)^{\frac{\sqrt{2}}{2}})$ update time. Sankowski[17] designed a deterministic algorithm that maintains maximum matching in $O(n^{1.495})$ update time. Onak and Rubinfeld[16] designed a randomized algorithm that maintains a c-approximation of maximum matching in $O(\log^2 n)$ update time, where c is a large unspecified constant. Baswana, Gupta and Sen[5] showed that maximal matching can be maintained in a dynamic graph in an amortized $O(\log n)$ update time with high probability. Subsequently, Anand et al. {[2, 3]} extended this work to the weighted case, and designed a randomized algorithm that maintains a matching with a weight that is *expected* to be at least $1/4.9108 \approx 0.2036$ of the optimum. Neiman and Solomon[15] designed a deterministic algorithm that maintains a matching of size at least $2/3$ of the size of optimum matching in $O(\sqrt{m})$ time per update in general graphs. Gupta and Peng [11] generalized this result — they designed a deterministic algorithm that maintains a $(1+\epsilon)$ approximate maximum matching in $O(\sqrt{m}\epsilon^{-2})$ update time. They also extended this result to a weighted graph by designing a deterministic algorithm that maintains a $(1+\epsilon)$-approximate weighted matching in $O(m\epsilon^{-O(1/\epsilon)} \log N)$ update time where the edges have weights in the range $[1, N]$.

We investigate the problem of finding maximum matching in an incremental bipartite graph. To the best of our knowledge, there are no results for maintaining near approximate maximum matching in incremental bipartite graphs. However, the results of Gupta and Peng [11] also applies to a bipartite graph. Note that their result applies for matching in *fully* dynamic graph which seems to be a harder problem than maintaining matching in an incremental graph. However, the running time they obtain have a dependence on $(\sqrt{m})$ where m is the maximum number of edges in the graph at any point of time.

To obtain better bounds, we look for inspiration from the field of streaming algorithms. Recently, there has been much interest[9, 10, 8] in the study of graph problems in a *semi-streaming* environment. In this model, the algorithm has to work with $O(n\ poly \log n)$ space and the one of the aim of the algorithm is to minimize the total number of passes over the stream. Ahn and Guha [1] showed that there exists a semi-streaming algorithm that find a $(1+\epsilon)$-approximate bipartite matching in an unweighted/weighted graph in $O(\log n/\epsilon^3)$ passes. One of the ingredients they use is what we call as *matching sparsifier*.

▶ **Definition 1.** A subgraph G' of G is said to be a (ϵ, β)-sparsifier if the size or weight of matching in G' is at least $\frac{\beta}{1+\epsilon}$ whenever the size or weight of maximum matching in G is equal to β.

Ahn and Guha [1] showed that a $(O(\epsilon), \beta)$-sparsifier can be found in $O(m \log n/\epsilon^3)$ time. We show that this algorithm works even for an incremental bipartite graph. This observation helps us in showing the following results:

▶ **Theorem 2.** *For any $\epsilon \leq 1/2$, there exists an algorithm that maintains a $(1+\epsilon)$ approximate maximum cardinality matching in an incremental bipartite graph in an amortized $O(\frac{\log^2 n}{\epsilon^4})$ update time.*

We extend the above result to weighted graphs:

▶ **Theorem 3.** *For any $\epsilon \leq 1/2$, there exists an algorithm that maintains a $(1+\epsilon)$ approximate maximum weighted matching in an incremental weighted bipartite graph in an amortized $O(\frac{\log n \log(nN)}{\epsilon^4})$ update time where each edge has weight in the range $[1, N]$.*

2 Preliminaries

An undirected graph is represented by $G = (V, E)$, where V represents the set of vertices and E represents the set of edges in the graph. We will use n to denote the number of vertices $|V|$ and m to denote the number of edges $|E|$.

A *matching* in a graph is a set of independent edges in the graph. The maximum cardinality matching(MCM) in a graph is the matching of maximum size. Let $\mathcal{M}$ denote a maximum matching in the graph. Similarly, given a set of weights $w : E \to [1, N]$, we can denote the weight of a matching M as $w(M) = \sum_{e \in M} w(e)$. The maximum weight matching(MWM) in a graph is in turn the matching of maximum weight.

For measuring the quality of approximate matching, we will use the notation of α-approximation, which indicates that the objective (either cardinality or weight) given by the current solution is at least $1/\alpha$ of the optimum. Specifically, a matching M is called α-MCM if $|M| \geq \frac{1}{\alpha}$(size of MCM), and α-MWM if $w(M) \geq \frac{1}{\alpha}$(weight of MWM).

Finding or approximating MCMs and MWMs in the static setting have been intensely studied. Near linear time algorithms have been developed for finding $(1+\epsilon)$ approximations and we will make crucial use of these algorithms in our data structure. For maximum cardinality matching, such an algorithm for bipartite graph was introduced by Hopcroft and Karp[12].

▶ **Lemma 4.** [12] *For any $\epsilon < 1$, there exists an algorithm* ApproxMCM *that finds a $(1+\epsilon)$-MCM in a static unweighted bipartite graph G in $O(m\epsilon^{-1})$ time where there are m edge in G.*

For approximate MWM, there has been some recent progress. Duan et al.[6, 7] designed an algorithm that finds a $(1+\epsilon)$ approximate maximum weighted matching in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ time.

▶ **Lemma 5.** [6, 7] *For any $\epsilon < 1$, there exists an algorithm* ApproxMWM *that finds a $(1+\epsilon)$-MWM in a static weighted graph G in $O(m\epsilon^{-1}\log(\epsilon^{-1}))$ time where there are m edge in G.*

In Section 3, we reproduce the results of [1] that finds a $(O(\epsilon), \beta)$-sparsifier in a static bipartite graph. In section 4, we design an algorithm that finds a $(O(\epsilon), \beta)$-sparsifier in an incremental bipartite graph and use this algorithm to maintain a $(1+\epsilon)$-MCM. In Section 5, we reproduce the algorithm in [1] that finds a $(O(\epsilon), \beta)$-sparsifier in a static weighted bipartite graph. In Section 6, we design an algorithm that finds a $(O(\epsilon), \beta)$-sparsifier in an incremental weighted bipartite graph and use this algorithm to maintain a $(1+\epsilon)$-MWM.

3 Background

Consider the following primal-dual for bipartite matching. Here the dual(LP2) is the linear program for bipartite matching and the primal(LP1) is the linear program for vertex cover.

$u_i^1 = 1 \ \forall i \in V$;
for $t = 1$ *to* T **do**
  Call FIND-ADMISSIBLE-SOLUTION(t)
  Let $M(i, \mathbf{y^t}) = \sum_{j:(i,j)\in E} y_{ij} - 1 \ \forall i$
  $\forall i$, set $u_i^{t+1} = \begin{cases} u_i^t(1+\epsilon)^{M(i,\mathbf{y^t})/\rho} & \text{if } M(i,\mathbf{y^t}) \geq 0 \\ u_i^t(1-\epsilon)^{-M(i,\mathbf{y^t})/\rho} & \text{if } M(i,\mathbf{y^t}) < 0 \end{cases}$
Output $\mathbf{y} = \left(\frac{1}{1+4\delta}\right)\frac{1}{T}\sum_t \mathbf{y^t}$;

Figure 1 MULTIPLICATIVE-WEIGHT-UPDATE(): The Multiplicative weight update method.

Primal

min	$\sum_i x_i$		
s. t.	$x_i + x_j \geq 1$	$\forall (i,j) \in E$	LP1
	$x_i \geq 0$		

Dual

max	$\sum_i y_{ij}$		
s. t.	$\sum_{j:(i,j)\in E} y_{ij} \leq 1$	$\forall i \in V$	LP2
	$y_{ij} \geq 0$		

We follow the algorithm of Ahn and Guha [1] in this section. We will use the multiplicative weight update method of Arora, Hazan and Kale [4]. Consider the multiplicative weight update method MULTIPLICATIVE-WEIGHT-UPDATE in Figure 1. The aim of MULTIPLICATIVE-WEIGHT-UPDATE is to find a feasible solution $\mathbf{y}$ such that $\sum \mathbf{y}_{ij} \approx \alpha$, i.e, the aim of our algorithm is to find a matching of a size approximately α. The algorithm runs for T iterations where we will calculate T at the end of the analysis. For each constraint associated with a vertex i in the dual LP, we associate it with a value u_i. Initially $u_i^1 = 1$. In each iteration of the algorithm, we find an *admissible* dual solution (by calling the procedure FIND-ADMISSIBLE-SOLUTION). We will define the notion of *admissibility* in Definition 6. Let $\mathbf{y^t}$ be an *admissible* solution in iteration t. Define $M(i, \mathbf{y}^t) = \sum_{j:(i,j)\in E} y_{ij}^t - 1$ for each constraint associated with vertex i. Then comes the step which justifies the name of the method, i.e., we update the weights of each u_i based on the value of $M(i, \mathbf{y^t})$. Note that ρ and ϵ are parameters in this update step which we will calculate in the analysis.

We now define the notion of *admissibility*.

▶ **Definition 6.** Define $E(\mathbf{y^t})$ to be the expected value of $M(i, \mathbf{y^t})$ if constraint i is chosen with probability $\frac{u_i^t}{\sum_j u_j^t}$, i.e, $E(\mathbf{y^t}) = \sum_i \frac{u_i^t}{\sum_j u_j^t} M(i, \mathbf{y^t})$. The dual solution $\mathbf{y^t}$ is admissible if $E(\mathbf{y^t}) \leq \delta$ and $\sum_{i,j} y_{ij} \geq \alpha$ and $M(i, \mathbf{y^t}) \in [l, \rho]$ where δ, ρ, l are parameter dependent on ϵ.

In [1], the following theorem calculates the number of iterations for which MULTIPLICATIVE-WEIGHT-UPDATE needs to run.

▶ **Theorem 7.** *Let* $\epsilon \leq 1/2, \delta = 4\epsilon l$. *If* FIND-ADMISSIBLE-SOLUTION *returns an* admissible *solution in all* $T = \frac{2\rho \log n}{\delta\epsilon}$ *iterations , then for all constraints* $i, M(i, \mathbf{y}) \leq 1$

Theorem 7 implies that if FIND-ADMISSIBLE-SOLUTION returns an *admissible* solution for T iteration, then we can find a feasible solution $\mathbf{y}$. Since $\mathbf{y} = \left(\frac{1}{1+4\delta}\right)\frac{1}{T}\sum_t \mathbf{y^t}$ and $\sum_{ij} y_{ij}^t \geq \alpha$, $\sum_{ij} y_{ij} \geq \left(\frac{1}{1+4\delta}\right)\alpha$. So we obtain a feasible *fractional* solution of a value approximately equal to α.

$\forall i$, let $x_i^t = \frac{\alpha u_i^t}{\sum_j u_j^t}$;
Let $E_{violated}^t = \{(i,j) : x_i^t + x_j^t < 1\}$
Find a maximal matching S_t in $E_{violated}^t$.
if $|S_t| < \delta\alpha$ **then**
  For each $(i,j) \in S_t$, increase x_i^t and x_j^t by 1
  return failure
else
  Return $y_{ij}^t = \frac{\alpha}{|S_t|}$ for $(i,j) \in S_t$ and 0 otherwise

Figure 2 FIND-ADMISSIBLE-SOLUTION(t): The procedure that finds an *admissible* solution.

3.1 MCM

The multiplicative weight update method mandates that an *admissible* solution is found at each iteration. We now reproduce FIND-ADMISSIBLE-SOLUTION designed by Ahn and Guha [1] which finds an *admissible* solution.

FIND-ADMISSIBLE-SOLUTION starts by setting x_i values for all vertices in the graph. The value of x_i^t is α times the probability of choosing a vertex i under the probability distribution $\mathbf{u^t}$. We look at the edges which violated the dual LP constraint using the assignment $\mathbf{x^t}$, i.e., all the edges $e = (i,j)$ such that $x_i^t + x_j^t < 1$. Let the set of all violated edge be denoted by $E_{violated}$. We find a maximal matching S_t in the set $E_{violated}$. If $|S_t| < \delta\alpha$, we return a failure, else we return $\mathbf{y^t}$ by setting $y_{ij}^t = \alpha/|S_t|$ for all the edges in the matching.

We reproduce the following lemma from [1]

▶ **Lemma 8.** [1] *If $|S_t| < \delta\alpha$, then* FIND-ADMISSIBLE-SOLUTION *returns a feasible solution for LP 2 with a value at most $(1+2\delta)\alpha$.*

Proof. For each edge (i,j) such that $x_i^t + x_j^t < 1$, at least one of the endpoint is in the maximal matching (since S_t is a maximal matching). We increase x_i^t and x_j^t by 1 to satisfy this constraint. So all the violated constraints are satisfied. Initially, $\sum_i x_i^t = \alpha$ and since we have increased the value of all the vertices in the maximal matching, the total increase is $< 2\delta\alpha$. So the total value of the solution of LP1 is at most $(1+2\delta)\alpha$. ◀

▶ **Lemma 9.** [1] *If $|S_t| \geq \delta\alpha$, then* FIND-ADMISSIBLE-SOLUTION *returns an* admissible *solution with $l = 1$ and $\rho = 1/\delta$ and $E(\mathbf{y^t}) \leq \delta$.*

Proof. Since $y_{i,j}^t = \alpha/|S_t|$, for all $(i,j) \in S_t$, so

$$\begin{array}{lcll}
\sum_{(i,j)\in S_t} y_{i,j}^t & = & \alpha & \\
\sum_{(i,j)\in S_t} y_{i,j}^t (x_i^t + x_j^t) & < & \alpha & \{ \text{ For each } (i,j) \in S_t,\ x_i^t + x_j^t < 1 \ \} \\
\sum_{(i,j)\in E} y_{i,j}^t (x_i^t + x_j^t) & = & \alpha & \{ \text{ Since } y_{i,j}^t = 0 \text{ for all other edges } \} \\
\sum_i x_i^t \sum_{j:(i,j)\in E} y_{i,j}^t & = & \sum_i x_i^t & \\
\sum_i x_i^t \left(\sum_{j:(i,j)\in E} y_{i,j}^t - 1 \right) & = & 0 & \\
\sum_i x_i^t M(i, y^t) & = & 0 & \\
\sum_i \frac{x_i^t}{\sum_j x_j^t} M(i, y^t) & = & 0 &
\end{array}$$

This implies $E(\mathbf{y^t}) \leq 0 \leq \delta$. Also, if $(i,j) \in S_t$, $y_{ij}^t = 1/\delta$, this implies $M(i, \mathbf{y^t}) = 1/\delta$, so $\rho = 1/\delta$. If $(i,j) \notin S_t$, $M(i, \mathbf{y^t}) = -1$, so $l = 1$. ◀

Now comes the crucial step of the algorithm. We never want FIND-ADMISSIBLE-SOLUTION to fail. This implies that the size of the maximal matching found by our algorithm should

always be greater than or equal to $\delta\alpha$. To achieve this, we set a suitable value of α. Let M be *any* matching in the graph such that $|M| = \beta$. Suppose that we set $\alpha_i = (1+\epsilon)^i$ for $i \geq 0$. Let α_j be the smallest value above β. So $\alpha_j \geq \beta \geq \alpha_{j-1}$. We now prove the following lemma:

▶ **Lemma 10.** *If $\beta \geq \alpha_{j-1}$ and if we set $\alpha = \alpha_j/(1+\epsilon)^9$ in* MULTIPLICATIVE-WEIGHT-UPDATE *procedure, then* FIND-ADMISSIBLE-SOLUTION *never fails.*

Proof. Suppose the algorithm fails. By Lemma 8, this implies that there exists a feasible solution of LP1 with value $(1+2\delta)\alpha$. Since $\epsilon = \delta/4$ (since $\delta = 4\epsilon l$ and $l = 1$), this value is $\leq (1+8\epsilon)\alpha = (1+8\epsilon)\alpha_j/(1+\epsilon)^9 \leq (1+8\epsilon)\beta/(1+\epsilon)^8 < \beta$. This leads to a contradiction since the minimum value of LP1 is $\geq \beta$ (as the size of maximum matching is $\geq \beta$). This implies that FIND-ADMISSIBLE-SOLUTION never fails. ◀

Using Theorem 7, if we set $\alpha = \alpha_j/(1+\epsilon)^9$, then after T iterations, we will find a feasible *fractional* solution $\mathbf{y}$. In MULTIPLICATIVE-WEIGHT-UPDATE procedure, we set $\mathbf{y} = \left(\frac{1}{1+4\delta}\right)\frac{1}{T}\sum_t \mathbf{y^t}$. Since FIND-ADMISSIBLE-SOLUTION returns an *admissible* solution of value $= \alpha$ in each iteration, we have $\sum y^t_{ij} = \left(\frac{1}{1+4\delta}\right)\alpha$. Since $\delta = 4\epsilon l$ and $l = 1$, $\delta = 4\epsilon$. So,

$$\begin{array}{lcll} \sum \mathbf{y}_{ij} & = & \left(\frac{1}{1+16\epsilon}\right)\alpha & \\ & = & \left(\frac{1}{1+16\epsilon}\right)\alpha_j/(1+\epsilon)^9 & \\ & \geq & \frac{1}{(1+16\epsilon)(1+\epsilon)^9}\alpha_j & \\ & \geq & \frac{1}{(1+16\epsilon)(1+75\epsilon)}\alpha_j & \left(\text{ if } \epsilon \leq 1/2, \text{ then } \frac{1}{(1+\epsilon)^9} \geq \frac{1}{1+75\epsilon}\right) \\ & \geq & \frac{1}{(1+1200\epsilon)}\alpha_j & \\ & \geq & \frac{1}{(1+1200\epsilon)}\beta & \left(\text{ since } \beta \leq \alpha_j\right) \end{array}$$

Thus, we have proved the following lemma:

▶ **Lemma 11.** *If* FIND-ADMISSIBLE-SOLUTION *returns an* admissible *solution for T iterations, then the size of the fractional matching returned by* MULTIPLICATIVE-WEIGHT-UPDATE *is* $\geq \frac{1}{1+1200\epsilon}\alpha_j \geq \frac{1}{(1+1200\epsilon)}\beta$

So if we set $\epsilon' = 1/1200\epsilon$, we get a feasible solution $\mathbf{y}$ such that the size of this fractional solution $\mathbf{y}$ is $\geq (1-\epsilon')\beta$. Also note that at each iteration t, FIND-ADMISSIBLE-SOLUTION selects at most n edges and sets $y^t_{ij} > 0$ for these n edges. Since there are at most $T = \frac{2\rho\log n}{\delta\epsilon}$ iterations, the total number of edges selected by FIND-ADMISSIBLE-SOLUTION with $\mathbf{y}_{ij} > 0$ is $\leq \frac{2\rho\log n}{\delta\epsilon}n$. We have the following lemma:

▶ **Theorem 12.** *There exists an algorithm which finds a $(O(\epsilon), \beta)$-sparsifier G' of G of size $O(n\log n/\epsilon^3)$. Moreover, the time taken to find such a graph is $O(m\log n/\epsilon^3)$,*

Proof. From the above discussion, we claim the total number of edges selected by FIND-ADMISSIBLE-SOLUTION is $O(\frac{2\rho\log n}{\delta\epsilon}n) = O(\frac{n\log n}{\epsilon^3})$ (since $\delta = 4\epsilon$ and $\rho = 1/\delta$). By Lemma 11, the size of fractional matching in G', i.e., $\sum \mathbf{y}_{ij}$ is $\geq \frac{1}{(1+1200\epsilon)}\beta$. Using the integrality of bipartite matching polytope, we claim that the size of maximum matching in G' is $\geq \frac{1}{(1+1200\epsilon)}\beta$. Regarding the running time, note that each iteration is dominated by the running time of FIND-ADMISSIBLE-SOLUTION — which is $O(m)$. Since $\rho = 1/\delta$ and $\delta = 4\epsilon$, there are at most $T = \frac{2\rho\log n}{\delta\epsilon} = O(\frac{\log n}{\epsilon^3})$ iterations and the total running time is $O(m\log n/\epsilon^3)$. ◀

4 Incremental MCM

Overview

In this section, we show that we can find a $(O(\epsilon), \beta)$-sparsifier in an incremental bipartite graph. This observation is then used to maintain a $(1+\epsilon)$-MCM in the following way: We run many versions of the algorithm in Theorem 12 in parallel such that in the kth run, we set $\alpha = \alpha_k/(1+\epsilon)^9$ in the MULTIPLICATIVE-WEIGHT-UPDATE procedure where $\alpha_k = (1+\epsilon)^k$ and $k \geq 9$. Since the size of maximum matching is $\leq n$, $k \leq \frac{\log n}{\log(1+\epsilon)} = O(\frac{\log n}{\epsilon})$. In the kth run, we want to find a $(O(\epsilon), \alpha_k)$-sparsifier. Note that initially a $(O(\epsilon), \alpha_k)$-sparsifier may not exist as the size of maximum matching itself may be $< \alpha_k$. So our algorithm returns failure till the size of maximum matching is approximately equal to α_k. At any given update step, say l, let i be the highest numbered version for which MULTIPLICATIVE-WEIGHT-UPDATE has not failed. We find a $(1+\epsilon)$-MCM M_i in the $(O(\epsilon), \alpha^i)$-sparsifier found in the ith run. We will show that M_i is a $(1+O(\epsilon))$-MCM in G_l, i.e., the ratio between the size of M_i and the maximum matching at the lth update step is $1+O(\epsilon)$.

Consider an incremental graph where at the update step l, an edge e_l is added to the graph, i.e., the graph at lth update step $G_l = G_{l-1} \cup e_l$. We use the incremental version of MULTIPLICATIVE-WEIGHT-UPDATE and FIND-ADMISSIBLE-SOLUTION (see Figure 3 and 4).

Fix a value of k. We describe our adaptation of the algorithm in the previous section for kth run of the algorithm when $\alpha = \alpha_k/(1+\epsilon)^9$. Consider the procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION. Before calling this procedure, we set $u_i^1 = 1$ for all the constraints of LP1. Since the graph is empty initially, the procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION fails in the first iteration. At update step l, e_l is added to the graph G_{l-1}, and the procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE is run. The procedure finds the iteration (say t), where INCREMENTAL-FIND-ADMISSIBLE-SOLUTION has failed. Since a new edge is added to the graph, the procedure calls INCREMENTAL-FIND-ADMISSIBLE-SOLUTION hoping that there exists an *admissible* solution after the addition of this new edge. If INCREMENTAL-FIND-ADMISSIBLE-SOLUTION returns failure, then there is still no *admissible* solution found at iteration t. Else, INCREMENTAL-FIND-ADMISSIBLE-SOLUTION successfully finds an *admissible* solution. We increment t and try to find an *admissible* solution in iteration $t+1$. If $t = T+1$, then using Lemma 11, we claim that the size of maximum matching in our sparsifier is at least $\frac{1}{(1+1200\epsilon)}\alpha_k$. This implies that we have found a $(O(\epsilon), \alpha^k)$-sparsifier at this update step. We then run ApproxMCM on this $(O(\epsilon), \alpha^k)$-sparsifier. We then *stop* the kth run of our algorithm.

We now describe the incremental version of the algorithm FIND-ADMISSIBLE-SOLUTION (see Figure 4). If procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE calls procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION for the first time in the iteration t, then we initialize all x_u^t's and find a maximal matching S_t as in procedure FIND-ADMISSIBLE-SOLUTION. Else we need to update S_t with respect to this newly added edge $e_l = (u, v)$. If $x_u^t + x_v^t$ is less than 1 and u and v are not adjacent to any edge in S_t, then the edge e_l is added to S_t. If $S_t < \delta\alpha$, then FIND-ADMISSIBLE-SOLUTION was unable to find an *admissible* solution and returns failure. Else we return an *admissible* solution $\mathbf{y^t}$(this part is same as in the FIND-ADMISSIBLE-SOLUTION). The important thing to note is that we return an *admissible* solution as soon as $|S_t|$ is equal to $\delta\alpha$.

We now prove the following lemma:

▶ **Lemma 13.** *If the size of maximum matching crosses α_{k-1} at update step l, then the kth run of* INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE stops *before or at update step l.*

Let the current update step be l with e_l being added to graph G_{l-1}.
Let t be the iteration in which INCREMENTAL-FIND-ADMISSIBLE-SOLUTION has previously failed.
while INCREMENTAL-FIND-ADMISSIBLE-SOLUTION*(t) does not return failure* **do**
  $t \leftarrow t+1$
  if *t = T+1* **then**
    Run APPROXMCM on the sparsifier found by the kth run
    Stop the kth run of the algorithm
  else
    Let $M(i,\mathbf{y^t}) = \sum_{j:(i,j)\in E} y_{ij} - 1 \ \forall i$
    $\forall i$, set $u_i^{t+1} = \begin{cases} u_i^t(1+\epsilon)^{M(i,\mathbf{y^t})/\rho} & \text{if } M(i,\mathbf{y^t}) \geq 0 \\ u_i^t(1-\epsilon)^{-M(i,\mathbf{y^t})/\rho} & \text{if } M(i,\mathbf{y^t}) < 0 \end{cases}$

Figure 3 INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE(): The incremental version of Multiplicative Weight Update Method.

if *this is the first call to* INCREMENTAL-FIND-ADMISSIBLE-SOLUTION *in iteration t* **then**
  $\forall w$, let $x_w^t = \frac{\alpha u_w^t}{\sum_j u_j^t}$;
  Let $E^t_{violated} = \{(u,v) : x_u^t + x_v^t < 1\}$
  Find a maximal matching S_t in $E^t_{violated}$.
else
  if $x_u^t + x_v^t < 1$ *and* u *and* v *are free with respect to* S_t **then**
    $S_t \leftarrow S_t \cup e_i$
if $|S_t| < \delta\alpha$ **then**
  return failure
else
  Return $y_{ij}^t = \frac{\alpha}{|S|}$ for $(i,j) \in S_t$ and 0 otherwise

Figure 4 INCREMENTAL-FIND-ADMISSIBLE-SOLUTION(t): The incremental version FIND-ADMISSIBLE-SOLUTION that finds an *admissible* solution.

Proof. Suppose that the procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE does not stop at or before the update step l. This implies that there exists an iteration t at which INCREMENTAL-FIND-ADMISSIBLE-SOLUTION is unable to find an *admissible* solution. Note that INCREMENTAL-FIND-ADMISSIBLE-SOLUTION incrementally maintains a maximal matching S_t. This implies that $|S_t| < \delta\alpha$. At the lth update step, the size of maximum matching is $\geq \alpha_{k-1}$, so Lemma 10 mandates that FIND-ADMISSIBLE-SOLUTION (and therefore INCREMENTAL-FIND-ADMISSIBLE-SOLUTION) never fails. This lead to a contradiction thus proving the lemma. ◀

At an update step l, let i be the highest numbered version for which the procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE has stopped. This implies that the size of maximum matching at this update step is less than α_i — if not then by Lemma 13, even $(i+1)$th run of INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE should have stopped. After the ith run stops, we use APPROXMCM to find a $(1+\epsilon)$-MCM in the sparsifier found at the ith run. Using Lemma 11, we claim that the size of matching in the sparsifier is $\frac{1}{(1+1200\epsilon)}\alpha^i$.

$\forall i$, let $x_i^t = \frac{\alpha u_i^t}{\sum_j u_j^t}$;
Let $E^t_{violated,k} = \{(i,j) : x_i^t + x_j^t < w_{ij}, \alpha/2^k \leq w_{ij} \leq \alpha/2^{k-1}\}$
Find a maximal matching S_t^k in $E^t_{violated,k}$ for each $k = 1, 2, \ldots, \lceil \log \frac{n}{\delta} \rceil = O(\log n)$.
Let $S_t = \cup_k S_k$, $\Delta = w(S_t)$
if $\Delta < \delta\alpha$ **then**
  For each $(i,j) \in S_t$, increase x_i and x_j by $2w_{ij}$.
  Further increase every x_i by $\delta\alpha$
  Return $\mathbf{x}$ and report failure.
else
  $S' \leftarrow \emptyset$
  while $S_t \neq \emptyset$ **do**
    Pick the heaviest edge (i,j) from S_t and add it to S'
    Remove all the edges adjacent to i and j from S_t
  Return $y_{ij}^t = \frac{\alpha}{w(S')}$ for $(i,j) \in S'$ and 0 otherwise

Figure 5 FIND-ADMISSIBLE-SOLUTION(t): The oracle which finds an *admissible* solution.

So, ApproxMCM finds a matching of size at least $\frac{\alpha^i}{(1+1200\epsilon)(1+\epsilon)} \geq \frac{1}{(1+1202\epsilon)}\alpha^i$. If we use the matching obtained at the ith run as our current matching, the approximation ratio of our matching with respect to the maximum matching is $\leq \frac{\alpha_i}{\alpha_i/(1+1202\epsilon)} = 1 + 1202\epsilon$.

Now we analyze the running time of the kth run of INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE. The running time of INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE is dominated by the procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION. We claim that the running time of this procedure at the tth iteration is at most $O(m)$ where m is the number of edges at the end of all updates. This is true because the initialization step takes at most $O(m)$ time and processing each update e_l takes $O(1)$ time. Since there are $T = O(\log n/\epsilon^3)$ iterations, the total running time for the kth run of our algorithm is $O(m \log n/\epsilon^3)$.

Since we run $O(\log n/\epsilon)$ versions of INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE in parallel, the total time taken is $O(m \log^2 n/\epsilon^4)$. This implies an amortized update time of $O(\frac{\log^2 n}{\epsilon^4})$. Thus, we have proved the following theorem:

▶ **Theorem 14.** *For any $\epsilon \leq 1/2$, there exists an algorithm that maintains a $(1+\epsilon)$-MCM in an incremental bipartite graph in amortized $O(\frac{\log^2 n}{\epsilon^4})$ update time.*

5 MWM

In this section, we follow the algorithm in [1] that finds a $(1+\epsilon)$-MWM in a static weighted graph. Consider the maximum weighted matching problem where each edge (u,v) has weight w_{uv} such that the weight of every edge in the graph is $\leq N$. As before we use the Multiplicative-Weight-Update procedure in Figure 1. If the procedure FIND-ADMISSIBLE-SOLUTION returns an admissible solution in each of the T iterations, then the procedure Multiplicative-Weight-Update finds a feasible solution. We reproduce the procedure FIND-ADMISSIBLE-SOLUTION for weighted graphs from [1].

The procedure starts by partitioning the edges across $\log n$ levels. If an edge (i,j) has weight between $\alpha/2^{k-1}$ and $\alpha/2^k$, then it is at level k where $k \leq \lceil \log \frac{n}{\delta} \rceil$. We ignore the edges with weight $\leq \alpha/2^{\log \frac{n}{\delta}}$. We find a maximal matching at all the k levels and let S_t be the union of these matchings. Let Δ be the sum of weight of all the edges in S_t. If $\Delta < \delta\alpha$,

then we report failure, else we find a matching S' from S_t. This procedure adds the heaviest edge (i, j) in S_t in S' and remove all the edges adjacent to i and j from S_t. The procedure then returns an *admissible* solution $\mathbf{y_t}$.

Ahn and Guha [1] proved the following:

▶ **Lemma 15.** [1] *If $\Delta > \delta\alpha$, then* FIND-ADMISSIBLE-SOLUTION *returns an admissible solution with $\rho = \frac{5}{\delta}$ and $l = 1$.*

▶ **Lemma 16.** [1] *If $\Delta \leq \delta\alpha$, then* FIND-ADMISSIBLE-SOLUTION *finds a feasible solution of vertex cover with value $(1 + 5\delta)\alpha$ and returns failure.*

Now comes the crucial step of the algorithm. We never want FIND-ADMISSIBLE-SOLUTION to fail. This implies that the weight of S_t found by our algorithm should always be greater than equal to $\delta\alpha$. Now we set a suitable value of α to achieve this. Let M be *any* matching in the graph such that the weight of M, $w(M) = \beta$. Suppose that we set $\alpha_i = (1 + \epsilon)^i$ for $i \geq 0$. Let α_j be the smallest value above β. So $\alpha_j \geq \beta \geq \alpha_{j-1}$. We now prove the following lemma:

▶ **Lemma 17.** *If $\beta \geq \alpha_{j-1}$ and if we set $\alpha = \alpha_j/(1+\epsilon)^{21}$, then* FIND-ADMISSIBLE-SOLUTION *never fails*

Proof. Suppose that FIND-ADMISSIBLE-SOLUTION fails. By Lemma 16, this implies that there exists a feasible solution of vertex cover with value $(1 + 5\delta)\alpha$. Since $\delta = 4\epsilon l$ and $l = 1$, this value is $\leq (1 + 20\epsilon)\alpha = (1 + 20\epsilon)\alpha_j/(1 + \epsilon)^{21} = (1 + 20\epsilon)\beta/(1 + \epsilon)^{20} < \beta$. This leads to a contradiction since the minimum weight of the vertex cover is $\geq \beta$ (as the weight of maximum matching is $\geq \beta$). This implies that the oracle FIND-ADMISSIBLE-SOLUTION never fails. ◀

Using Lemma 7, if we set $\alpha = \alpha_j/(1+\epsilon)^{21}$, then after T iterations, we will find a feasible solution $\mathbf{y}$. In the MULTIPLICATIVE-WEIGHT-UPDATE procedure, we set $\mathbf{y} = \left(\frac{1}{1+4\delta}\right)\frac{1}{T}\sum_t \mathbf{y^t}$. Since FIND-ADMISSIBLE-SOLUTION returns an *admissible* solution of value $= \alpha$ in each iteration, we have $\sum y^t_{ij} = \left(\frac{1}{1+4\delta}\right)\alpha$. Since $\delta = 4\epsilon l$ and $l = 1$, $\delta = 4\epsilon$. So,

$$\begin{aligned}
\sum \mathbf{y}_{ij} &= \left(\tfrac{1}{1+16\epsilon}\right)\alpha \\
&= \left(\tfrac{1}{1+16\epsilon}\right)\tfrac{\alpha_j}{(1+\epsilon)^{21}} \\
&\geq \tfrac{1}{(1+16\epsilon)(1+10000\epsilon)}\alpha_j \qquad \left(\text{ if } \epsilon \leq 1/2, \text{ then } \tfrac{1}{(1+\epsilon)^{21}} \geq \tfrac{1}{1+10000\epsilon}\right) \\
&\geq \tfrac{1}{(1+160000\epsilon)}\alpha_j \qquad (2) \\
&\geq \tfrac{1}{(1+160000\epsilon)}\beta \qquad \left(\text{ since } \beta \leq \alpha_j\right)
\end{aligned}$$

We can state the following lemma:

▶ **Lemma 18.** *If* FIND-ADMISSIBLE-SOLUTION *returns an admissible solution for T iterations then the size of fractional matching return by our algorithm is $\geq \frac{1}{1+160000\epsilon}\alpha_j \geq \frac{1}{(1+160000\epsilon)}\beta$*

If we set $\epsilon' = 1/160000\epsilon$, we get a feasible solution $\mathbf{y}$ such that the size of this fractional solution $\mathbf{y}$ is $\geq (1 - \epsilon')\beta$. Also note that at each iteration t, FIND-ADMISSIBLE-SOLUTION selects at most n edges and sets $y^t_{ij} > 0$ for these n edges. Since there are at most $T = \frac{2\rho \log n}{\delta\epsilon}$ iterations, the total number of edges selected by FIND-ADMISSIBLE-SOLUTION with $\mathbf{y}_{ij} > 0$ is $\leq \frac{2\rho \log n}{\delta\epsilon}n$. Since $\rho = 5/\delta$ and $\delta = 4\epsilon l$, we have the following lemma:

▶ **Theorem 19.** *There exists an algorithm which finds a $(O(\epsilon), \beta)$-sparsifier G' of G of size $O(n \log n/\epsilon^3)$. Moreover, the time taken to find such a graph is $O(m \log n/\epsilon^3)$.*

Proof. The proof is identical to the proof of Theorem 12. ◀

6 Incremental MWM

In this section, we show that we can find a $(O(\epsilon), \beta)$-sparsifier in an incremental weighted bipartite graph where the weight of any edge is $\leq N$. This observation is then used to maintain a $(1+\epsilon)$-MWM in the following way: We run many versions of algorithm in Theorem 19 in parallel such that in the kth run, we set $\alpha = \alpha_k/(1+\epsilon)^{21}$ in the MULTIPLICATIVE-WEIGHT-UPDATE where $\alpha_k = (1+\epsilon)^k$ and $k \geq 21$. Since the size of maximum matching is $\leq nN$, $k \leq \frac{\log(nN)}{\log(1+\epsilon)} = O(\frac{\log(nN)}{\epsilon})$. In the kth run, we want to find a $(O(\epsilon), \alpha_k)$-sparsifier. Note that initially a $(O(\epsilon), \alpha_k)$-sparsifier may not exist as the size of maximum weighted matching itself may be $< \alpha_k$. So our algorithm returns failure till the size of maximum matching is approximately equal to α_k. At any given update step, say l, let i be the highest numbered version for which MULTIPLICATIVE-WEIGHT-UPDATE has not failed. We find a $(1+\epsilon)$-MWM M_i in the $(O(\epsilon), \alpha^i)$-sparsifier found in the ith run. We will show that the ratio between the weight of M_i and the weight of maximum matching at the lth update step is $1 + O(\epsilon)$.

Consider an incremental weighted graph where at the update step l, an edge $e_l = (i, j)$ with weight w_{ij} is added to the graph, i.e., the graph at lth update step $G_l = G_{l-1} \cup e_l$. We use the incremental version of MULTIPLICATIVE-WEIGHT-UPDATE in Figure 3. Now, we design an incremental version of FIND-ADMISSIBLE-SOLUTION (see Figure 6).

Fix a value of k. We describe our adaptation of the algorithm in the previous section for kth run of the algorithm when $\alpha = \alpha_k/(1+\epsilon)^{21}$. If procedure INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE calls procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION for the first time in the iteration t, we initialize all x_u^t's and find a maximal matching S_t as in procedure FIND-ADMISSIBLE-SOLUTION. Else we need to update S_t with respect to this newly added edge $e_l = (u, v)$. If $(u, v) \in E_{violated,k}$ and u and v are free with respect to maximal matching in $E_{violated,k}$, edge e_l is added to S_t. If $S_t < \delta\alpha$, then the oracle was unable to find an *admissible* solution and returns failure. Else we return an *admissible* solution $\mathbf{y^t}$ (this part is same as in FIND-ADMISSIBLE-SOLUTION). Again note that we return an *admissible* solution as soon as $|S_t|$ is equal to $\delta\alpha$.

We now prove the following lemma:

▶ **Lemma 20.** *If the size of maximum matching crosses α_{k-1} at update step l, then the kth run of* INCREMENTAL-MULTIPLICATIVE-WEIGHT-UPDATE stops *before or at update step l.*

Proof. Similar to the proof of Lemma 13. ◀

Similar to our analysis in Section 4, we claim that our algorithm maintains a $(1+\epsilon)$-MWM at every update step.

Now we analyze the running time of our algorithm. Consider the ith run of the algorithm. The running time of the algorithm is dominated by the procedure INCREMENTAL-FIND-ADMISSIBLE-SOLUTION. We claim that the running time of this procedure at the tth iteration is at most $O(m)$ where m is the number of edges at the end of all updates. This is true because the initialization step takes at most $O(m)$ time and processing each update e_l takes $O(1)$ time. Since there are $T = O(\frac{\log n}{\epsilon^3})$ iterations, the total running time for the ith run of our algorithm is $O(m \log n/\epsilon^3)$. Since there are $O(\log(nN)/\epsilon)$ version of our algorithm, the total time taken by the algorithm is $O(m \log n \log(nN)/\epsilon^4)$. This implies an amortized update time of $O(\log n \log(nN)/\epsilon^4)$.

We claim the following theorem:

▶ **Theorem 21.** *For any $\epsilon \leq 1/2$, there exists an algorithm that maintains a $(1+\epsilon)$-MWM in an incremental weighted bipartite graph in amortized $O(\frac{\log n \log(nN)}{\epsilon^4})$ update time where each edge has weight in the range $[1, N]$.*

if *this is the first call to* INCREMENTAL-FIND-ADMISSIBLE-SOLUTION *in iteration* t **then**
  $\forall i$, let $x_i^t = \frac{\alpha u_i^t}{\sum_j u_j^t}$;
  Let $E^t_{violated,k} = \{(i,j) : x_i^t + x_j^t < w_{ij}, \alpha/2^k \leq w_{ij} \leq \alpha/2^{k-1}\}$
  Find a maximal matching S_t^k in $E^t_{violated,k}$ for each $k = 1, 2, \ldots, \lceil \log \frac{n}{\delta} \rceil = O(\log n)$.
  Let $S_t = \cup_k S_k$, $\Delta = w(S_t)$
else
  if $(u,v) \in E^t_{violated,k}$ *and* u *and* v *are free with respect to the maximal matching in* $E^t_{violated,k}$ **then**
    $S_t \leftarrow S_t \cup e_l$
if $|S_t| < \delta\alpha$ **then**
  return failure
else
  $S' \leftarrow \emptyset$
  while $S_t \neq \emptyset$ **do**
    Pick the heaviest edge (i,j) from S and add it to S'
    Remove all the edges adjacent to i and j from S_t
  Return $y_{ij}^t = \frac{\alpha}{w(S')}$ for $(i,j) \in S'$ and 0 otherwise

Figure 6 INCREMENTAL-FIND-ADMISSIBLE-SOLUTION(e_l, t): The incremental version of the oracle that finds an *admissible* solution.

References

1 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.*, 222:59–79, 2013.

2 Abhash Anand, Surender Baswana, Manoj Gupta, and Sandeep Sen. Maintaining Approximate Maximum Weighted Matching in Fully Dynamic Graphs. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012)*, volume 18 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 257–266, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

3 Abhash Anand, Surender Baswana, Manoj Gupta, and Sandeep Sen. Maintaining approximate maximum weighted matching in fully dynamic graphs. *CoRR*, abs/1207.3976, 2012.

4 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

5 S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 383–392. IEEE, 2011.

6 Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, January 2014.

7 Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for approximate and exact maximum weight matching. *CoRR*, abs/1112.0790, 2011.

8 Sebastian Eggert, Lasse Kliemann, Peter Munstermann, and Anand Srivastav. Bipartite matching in the semi-streaming model. *Algorithmica*, 63(1-2):490–508, June 2012.

9 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2):207–216, December 2005.

10 Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, December 2008.

11 Manoj Gupta and Richard Peng. Fully dynamic $(1 + \epsilon)$-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science*, 2013.

12 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.

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

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

15 Ofer Neiman and Shay Solomon. Deterministic algorithms for fully dynamic maximal matching. *CoRR*, abs/1207.1277, 2012.

16 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC*, pages 457–464, 2010.

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

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