Design of Dynamic Algorithms via Primal-Dual Method

Sayan Bhattacharya*, Monika Henzinger**, Giuseppe F. Italiano***

Abstract. In this paper, we develop a dynamic version of the primal-dual method for optimization problems, and apply it to obtain the following results. (1) For the dynamic set-cover problem, we maintain an $O(f^2)$-approximately optimal solution in $O(f \cdot \log(m+n))$ amortized update time, where f is the maximum "frequency" of an element, n is the number of sets, and m is the maximum number of elements in the universe at any point in time. (2) For the dynamic b-matching problem, we maintain an $O(1)$-approximately optimal solution in $O(\log^3 n)$ amortized update time, where n is the number of nodes in the graph.

1 Introduction

The primal-dual method lies at the heart of the design of algorithms for combinatorial optimization problems. The basic idea, contained in the "Hungarian Method" [8], was extended and formalized by Dantzig et al. [5] as a general framework for linear programming, and thus it became applicable to a large variety of problems. Few decades later, Bar-Yehuda et al. [1] were the first to apply the primal-dual method to the design of approximation algorithms. Subsequently, this paradigm was applied to obtain approximation algorithms for a wide collection of NP-hard problems [11]. When the primal-dual method is applied to approximation algorithms, an approximate solution to the problem and a feasible solution to the dual of an LP relaxation are constructed simultaneously, and the performance guarantee is proved by comparing the values of both solutions. The primal-dual method was also extended to online problems [4]. Here, the input is revealed only in parts, and an online algorithm is required to respond to each new input upon its arrival (without being able to see the future). The algorithm's performance is compared against the benchmark of an optimal omniscient algorithm that can view the entire input sequence in advance.

In this paper, we focus on dynamic algorithms for optimization problems. In the dynamic setting, the input of a problem is being changed via a sequence of updates, and after each update one is interested in maintaining the solution to the problem much faster than recomputing it from scratch. We remark that the dynamic and the online setting are completely different: in the dynamic scenario one is concerned more with guaranteeing fast (worst-case or amortized) update

* Institute of Mathematical Sciences, Chennai, India. E-mail: bsayan@imsc.res.in.
** University of Vienna, Austria. E-mail: monika.henzinger@univie.ac.at.
*** Università di Roma "Tor Vergata", Italy. E-mail: giuseppe.italiano@uniroma2.it.

times rather than comparing the algorithms' performance against optimal offline algorithms. As a main contribution of this paper, we develop a dynamic version of the primal-dual method, thus opening up a completely new area of application of the primal-dual paradigm to the design of dynamic algorithms. With some careful insights, our recent algorithms for dynamic matching and dynamic vertex cover [3] can be reinterpreted in this new framework. In this paper, we show how to apply the new dynamic primal-dual framework to the design of two other optimization problems: the dynamic set-cover problem and the dynamic b-matching problem.

Definition 1 (Set-Cover). *We are given a universe $\mathcal{U}$ of at most m elements, and a collection $\mathcal{S}$ of n sets $S \subseteq \mathcal{U}$. Each set $S \in \mathcal{S}$ has a (polynomially bounded by n) "cost" $c_S > 0$. The goal is to select a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that each element in $\mathcal{U}$ is covered by some set $S \in \mathcal{S}'$ and the total cost $\sum_{S \in \mathcal{S}'} c(S)$ is minimized.*

Definition 2 (Dynamic Set-Cover). *Consider a dynamic version of the problem specified in Definition 1, where the collection $\mathcal{S}$, the costs $\{c_S\}, S \in \mathcal{S}$, the upper bound f on the maximum frequency $\max_{u \in \mathcal{U}} |\{S \in \mathcal{S} : u \in S\}|$, and the upper bound m on the maximum size of the universe $\mathcal{U}$ remain fixed. The universe $\mathcal{U}$, on the other hand, keeps changing dynamically. In the beginning, we have $\mathcal{U} = \emptyset$. At each time-step, either an element u is inserted into the universe $\mathcal{U}$ and we get to know which sets in $\mathcal{S}$ contain u, or some element is deleted from the universe. The goal is to maintain an approximately optimal solution to the set-cover problem in this dynamic setting.*

Definition 3 (b-Matching). *We are given an input graph $G = (V, E)$ with $|V| = n$ nodes, where each node $v \in V$ has a capacity $c_v \in \{1, \ldots, n\}$. A b-matching is a subset $E' \subseteq E$ of edges such that each node v has at most c_v edges incident to it in E'. The goal is to select the b-matching of maximum cardinality.*

Definition 4 (Dynamic b-Matching). *Consider a dynamic version of the problem specified in Definition 3, where the node set V and the capacities $\{c_v\}, v \in V$ remain fixed. The edge set E, on the other hand, keeps changing dynamically. In the beginning, we have $E = \emptyset$. At each time-step, either a new edge is inserted into the graph or some existing edge is deleted from the graph. The goal is to maintain an approximately optimal solution to the b-matching problem in this dynamic setting.*

As stated in [4, 11], the set-cover problem has played a pivotal role both for approximation and for online algorithms, and thus it seems a natural problem to consider in our dynamic setting. Our definition of dynamic set-cover is inspired by the standard formulation of the online set-cover problem [4], where the elements arrive online.

Our Techniques. Roughly speaking, our dynamic version of the primal-dual method works as follows. We start with a feasible primal solution and an infeasible dual solution for the problem at hand. Next, we consider the following process: gradually increase all the primal variables at the same rate, and

whenever a primal constraint becomes tight, stop the growth of all the primal variables involved in that constraint, and update accordingly the corresponding dual variable. This primal growth process is used to define a suitable data structure based on a hierarchical partition. A level in this partition is a set of the dual variables whose corresponding primal constraints became (approximately) tight at the same time-instant. To solve the dynamic problem, we maintain the data structure, the hierarchical partition and the corresponding primal-dual solution dynamically using a simple greedy procedure. This is sufficient for solving the dynamic set-cover problem. For the dynamic b-matching problem, we need some additional ideas. We first get a fractional solution to the problem using the previous technique. To obtain an integral solution, we perform randomized rounding on the fractional solution in a dynamic setting. This is done by sampling the edges with probabilities that are determined by the fractional solution.

Our Results. Our new dynamic primal-dual framework yields efficient dynamic algorithms for both the dynamic set-cover problem and the dynamic b-matching problem. In particular, for the dynamic set-cover problem we maintain a $O(f^2)$-approximately optimal solution in $O(f \cdot \log(m+n))$ amortized update time (see Corollary 1 in Section 2). On the other hand, for the dynamic b-matching problem, we maintain a $O(1)$-approximation in $O(\log^3 n)$ amortized time per update (see Theorem 7 in Section 3). Further, we can show that an edge insertion/deletion in the input graph, on average, leads to $O(\log^2 n)$ changes in the set of matched edges maintained by our algorithm.

Related Work. The design of dynamic algorithms is one of the classic areas in theoretical computer science with a countless number of applications. Dynamic graph algorithms have received special attention, and there have been many efficient algorithms for several dynamic graph problems, including dynamic connectivity, minimum spanning trees, transitive closure, shortest paths and matching problems (see, e.g., the survey in [6]). The b-matching problem contains as a special case matching problems, for which many dynamic algorithms are known [2, 3, 7, 9, 10]. Unfortunately, none of the results on dynamic matching extends to the dynamic b-matching problem. To the best of our knowledge, no previous result was known for dynamic set-cover problem.

2 Maintaining a Set-Cover in a Dynamic Setting

We define a problem called "fractional hypergraph b-matching" (Definition 5). Later, we show that this generalizes the well-known set-cover problem (Lemma 1). Our main result is Theorem 2, which, along with Lemma 1, implies Corollary 1.

Definition 5 (Fractional Hypergraph b-Matching). *We are given an input hypergraph $G = (V, E)$ with $|V| = n$ nodes and* at most *$m \geq |E|$ edges. Let $\mathcal{E}_v \subseteq E$ denote the set of edges incident upon a node $v \in V$, and let $\mathcal{V}_e = \{v \in V : e \in \mathcal{E}_v\}$ denote the set of nodes an edge $e \in E$ is incident upon. Let $c_v > 0$ denote the "capacity" of a node $v \in V$, and let $\mu \geq 1$ denote the "multiplicity" of an edge. We assume that the μ and the c_v values are polynomially bounded by n.*

Our goal is to assign a "weight" $x(e) \in [0, \mu]$ to each edge $e \in E$ in such a way that (a) $\sum_{e \in \mathcal{E}_v} x(e) \leq c_v$ for all nodes $v \in V$, and (b) the sum of the weights of all the edges is maximized.

Below, we write a linear program for the above problem and its dual.

$$\textbf{Primal LP:} \quad \text{Maximize} \quad \sum_{e \in E} x(e) \tag{1}$$

$$\text{subject to:} \quad \sum_{e \in \mathcal{E}_v} x(e) \leq c_v \qquad \forall v \in V. \tag{2}$$

$$0 \leq x(e) \leq \mu \qquad \forall \text{e} \in E. \tag{3}$$

$$\textbf{Dual LP:} \quad \text{Minimize} \quad \sum_{v \in V} c_v \cdot y(v) + \sum_{e \in E} \mu \cdot z(e) \tag{4}$$

$$\text{subject to:} \quad z(e) + \sum_{v \in \mathcal{V}_e} y(v) \geq 1 \qquad \forall e \in E. \tag{5}$$

$$y(v), z(e) \geq 0 \qquad \forall v \in V, e \in E. \tag{6}$$

Definition 6. *A feasible solution to LP (1) is λ-maximal, $\lambda \geq 1$, iff for every edge $e \in E$ with $x(e) < \mu$, there is some node $v \in \mathcal{V}_e$ with $\sum_{e' \in \mathcal{E}_v} x(e') \geq c_v/\lambda$.*

Theorem 1. *Let $f \geq \max_{e \in E} |\mathcal{V}_e|$ be an upper bound on the maximum possible "frequency" of an edge. Let OPT be the optimal objective value of LP (1). Any λ-maximal solution to LP (1) has an objective value that is at least $OPT/(\lambda f + 1)$.*

Proof (Sketch). Follows from LP duality. □

Definition 7 (Dynamic Fractional Hypergraph b-Matching). *Consider a dynamic version of the problem specified in Definition 5, where the node-set V, the capacities $\{c_v\}, v \in V$, the upper bound f on the maximum frequency $\max_{e \in E} |\mathcal{V}_e|$, and the upper bound m on the maximum number of edges remain fixed. The edge-set E, on the other hand, keeps changing dynamically. In the beginning, we have $E = \emptyset$. At each time-step, either an edge is inserted into the graph or an edge is deleted from the graph. The goal is to maintain an approximately optimal solution to the problem in this dynamic setting.*

Theorem 2. *We can maintain a $(f + 1 + \epsilon f)$-maximal solution to dynamic fractional hypergraph b-matching in $O(f \cdot \log(m+n)/\epsilon^2)$ amortized update time.*

We now compare fractional hypergraph b-matching with set-cover.

Lemma 1. *The dual LP (4) is an LP-relaxation of the set-cover problem.*

Proof (Sketch). Given an instance of the set-cover problem, we create an instance of the hypergraph b-matching problem as follows. For each element $u \in \mathcal{U}$ create an edge $e(u) \in E$, and for each set $S \in \mathcal{S}$, create a node $v(S) \in V$ with capacity $c_{v(S)} = c_S$. Ensure that an element u belongs to a set S iff $e(u) \in \mathcal{E}_{v(S)}$. Set

$\mu = \max_{v \in V} c_v + 1$. Since $\mu > \max_{v \in V} c_v$, it can be shown that an optimal solution to the dual LP (4) will set $z(e) = 0$ for every edge $e \in E$. Thus, we can remove the variables $\{z(e)\}$ from the constraints and the objective function of LP (4) to get a new LP with the same optimal objective value. This new LP is an LP-relaxation for the set-cover problem. □

Corollary 1. *We can maintain an $(f^2+f+\epsilon f^2)$-approximately optimal solution to the dynamic set cover problem in $O(f \cdot \log(m+n)/\epsilon^2)$ amortized update time.*

Proof (Sketch). We map the set cover instance to a fractional hypergraph b-matching instance as in the proof of Lemma 1. By Theorem 2, in $O(f \log(m+n)/\epsilon^2)$ amortized update time, we can maintain a feasible solution $\{x^*(e)\}$ to LP (1) that is λ-maximal, where $\lambda = f+1+\epsilon f$. Consider a collection of sets $\mathcal{S}^* = \{S \in \mathcal{S} : \sum_{e \in \mathcal{E}_{v(S)}} x(e) \geq c_{v(S)}/\lambda\}$. Since we can maintain the fractional solution $\{x^*(e)\}$ in $O(f \log(m+n)/\epsilon^2)$ amortized update time, we can also maintain $\mathcal{S}^*$ without incurring any additional overhead in the update time. Now, using complementary slackness conditions, we can show that each element $e \in \mathcal{U}$ is covered by some $S \in \mathcal{S}^*$, and the sum $\sum_{S \in \mathcal{S}^*} c_S$ is at most (λf)-times the size of the primal solution $\{x^*(e)\}$. The corollary follows from LP duality. □

For the rest of this section, we focus on proving Theorem 2. First, in the static setting, inspired by the primal-dual method for set-cover we consider the following algorithm for the fractional hypergraph b-matching problem.

- Consider a primal solution with $x(e) \leftarrow 0$ for all $e \in E$, and let $F \leftarrow E$.
- WHILE there is some primal constraint that is not tight:
  - Keep increasing the primal variables $\{x(e)\}, e \in F$, uniformly at the same rate till some primal constraint becomes tight. At that instant, "freeze" all the primal variables involved in that constraint and delete them from the set F, and set the corresponding dual variable to one.

Figure 1 defines a variant of the above procedure that happens to be easier to maintain in a dynamic setting. The main idea is to discretize the continuous primal growth process. Define $c_{\min} = \min_{v \in V} c_v$, and without any loss of generality, assume that $c_{\min} > 0$. Fix $\alpha, \beta > 1$, and define $L = \lceil \log_\beta(m\mu\alpha/c_{\min}) \rceil$.

Claim 3 *If $x(e) = \mu \cdot \beta^{-L}$ for all $e \in E$, then we have a feasible primal solution.*

Proof. Clearly, $x(e) \leq \mu$ for all $e \in E$. Now, consider any node $v \in V$. We have $\sum_{e \in \mathcal{E}_v} x(e) = |\mathcal{E}_v| \cdot \mu \cdot \beta^{-L} \leq |E| \cdot \mu \cdot \beta^{-L} \leq m \cdot \mu \cdot \beta^{-L} \leq m \cdot \mu \cdot (c_{\min}/(m\mu\alpha)) = c_{\min}/\alpha < c_v$. Hence, all the primal constraints are satisfied. □

Our new algorithm is described in Figure 1. We initialize our primal solution by setting $x(e) \leftarrow \mu\beta^{-L}$ for every edge $e \in E$, as per Claim 3. Say that a node v is *nearly-tight* if its corresponding primal constraint is tight within a factor of $f\alpha\beta$, and *slack* otherwise. Say that an edge is *nearly-tight* if it is incident upon some *nearly-tight* node, and *slack* otherwise. Let $V_L \subseteq V$ and $E_L \subseteq E$

01. Set $x(e) \leftarrow \mu \cdot \beta^{-L}$ for all $e \in E$, and define $c_v^* = c_v/(f\alpha\beta)$ for all $v \in V$.
02. Set $V_L \leftarrow \{v \in V : \sum_{e \in \mathcal{E}_v} x(e) \geq c_v^*\}$, and $E_L \leftarrow \bigcup_{v \in V_L} \mathcal{E}_v$.
03. For $i = L - 1$ to 1:
04. Set $x(e) \leftarrow x(e) \cdot \beta$ for all $e \in E \setminus \bigcup_{k=i+1}^{L} E_i$.
05. Set $V_i \leftarrow \left\{v \in V \setminus \bigcup_{k=i+1}^{L} V_k : \sum_{e \in \mathcal{E}_v} x(e) \geq c_v^*\right\}$.
06. Set $E_i \leftarrow \bigcup_{v \in V_i} \mathcal{E}_v$.
07. Set $V_0 \leftarrow V \setminus \bigcup_{k=1}^{L} V_i$, and $E_0 \leftarrow \bigcup_{v \in V_0} \mathcal{E}_v$.
08. Set $x(e) \leftarrow x(e) \cdot \beta$ for all $e \in E_0$.

Fig. 1. DISCRETE-PRIMAL-DUAL().

respectively denote the sets of *nearly-tight* nodes and edges, immediately after the initialization step. The algorithm then performs $L - 1$ iterations.

At iteration $i \in \{L - 1, \ldots, 1\}$, the algorithm increases the weight $x(e)$ of every *slack* edge e by a factor of β. Since the total weight received by every *slack* node v (from its incident edges) never exceeds $c_v/(f\alpha\beta)$, this weight-increase step does not violate any primal constraint. The algorithm then defines V_i (resp. E_i) to be the set of new nodes (resp. edges) that become *nearly-tight* due to this weight-increase step.

Finally, the algorithm defines V_0 (resp. E_0) to be the set of nodes (resp. edges) that are *slack* at the end of iteration $i = 1$. It terminates after increasing the weight of every edge in E_0 by a factor of β.

When the algorithm terminates, it is easy to check that $x(e) = \mu \cdot \beta^{-i}$ for every edge $e \in E_i$, $i \in \{0, \ldots, L\}$. We also have $c_v^* \leq \sum_{e \in \mathcal{E}_v} x(e) \leq \beta \cdot c_v^*$ for every node $v \in \bigcup_{k=1}^{L} V_k$, and $\sum_{e \in \mathcal{E}_v} x(e) \leq c_v^*$ for every node $v \in V_0$. Furthermore, at the end of the algorithm, every edge $e \in E$ is either *nearly-tight*, or it has weight $x(e) = \mu$. We, therefore, reach the following conclusion.

Claim 4 *The algorithm described in Figure 1 returns an $(f\alpha\beta)$-maximal solution to the fractional hypergraph b-matching problem.*

Our goal is to make a variant of the procedure in Figure 1 work in a dynamic setting. Towards this end, we introduce the concept of an (α, β)-partition (see Definition 8) satisfying a certain invariant (see Invariant 5). The reader is encouraged to notice the similarities between this construct and the output of the procedure in Figure 1.

Definition 8. *Fix any two parameters $\alpha, \beta > 1$ and let $c_{\min} = \min_{v \in V} c_v > 0$. An* (α, β)-partition *of the hypergraph G partitions its node-set V into subsets $V_0 \ldots V_L$, where $L = \lceil \log_\beta(m\mu\alpha/c_{\min}) \rceil$. For $i \in \{0, \ldots, L\}$, we identify the subset V_i as the i^{th} "level" of this partition, and denote the* level of a node v *by $\ell(v)$. Thus, we have $v \in V_{\ell(v)}$ for all $v \in V$. We also define the level of each edge $e \in E$ as $\ell(e) = \max_{v \in \mathcal{V}_e} \{\ell(v)\}$, and assign a "weight" $w(e) = \mu \cdot \beta^{-\ell(e)}$ to e.*

Given an (α, β)-partition, let $\mathcal{E}_v(i) = \{e \in \mathcal{E}_v : \ell(e) = i\}$ denote the set of edges incident to v that are in the i^{th} level, and let $\mathcal{E}_v(i, j) = \bigcup_{k=i}^{j} \mathcal{E}_v(k)$ denote

the set of edges incident to v whose levels are in the range $[i, j]$. Similarly, we define the notations $\mathcal{D}_v = |\mathcal{E}_v|$ and $\mathcal{D}_v(i,j) = |\mathcal{E}_v(i,j)|$. Let $W_v = \sum_{e \in \mathcal{E}_v} w(e)$ denote the total weight a node $v \in V$ receives from the edges incident to it. We also define the notation $W_v(i) = \sum_{e \in \mathcal{E}_v} \mu \cdot \beta^{-\max(\ell(e), i)}$. It gives the total weight the node v would receive from the edges incident to it, *if the node v itself were to go to the i^{th} level*. It is easy to check that an (α, β)-partition satisfies the following conditions for all nodes $v \in V$.

$$W_v(L) \leq c_{\min}/\alpha \tag{7}$$

$$W_v(L) \leq \cdots \leq W_v(i) \leq \cdots \leq W_v(0) \tag{8}$$

$$W_v(i) \leq \beta \cdot W_v(i+1) \quad \forall i \in \{0, \ldots, L-1\}. \tag{9}$$

Invariant 5 *Define $c^*_v = c_v/(f\alpha\beta)$. For every node $v \in V$, if $\ell(v) = 0$, then $W_v \leq f\alpha\beta \cdot c^*_v$. Else if $\ell(v) \geq 1$, then $c^*_v \leq W_v \leq f\alpha\beta \cdot c^*_v$.*

Lemma 2. *Consider an (α, β)-partition that satisfies Invariant 5. The edge-weights $\{w(e)\}, e \in E$, give an $(f\alpha\beta)$-maximal solution to LP (1).*

Proof (Sketch). Similar to the proof of Claim 4. □

Handling an edge insertion/deletion. Consider an (α, β)-partition in the graph G. A node is called *dirty* if it violates Invariant 5, and *clean* otherwise. Since the edge-set E is initially empty, every node is clean and at level zero before the first update. Now consider the time instant just prior to the t^{th} update. By induction hypothesis, at this instant every node is clean. Then the t^{th} update takes place, which inserts (resp. deletes) an edge e in E with weight $w(e) = \mu\beta^{-\ell(e)}$. This increases (resp. decreases) the weights $\{W_x\}, x \in \mathcal{V}_e$. Due to this change, the nodes $x \in \mathcal{V}_e$ might become dirty. To recover from this, we call the subroutine in Figure 2.

```
01. WHILE there exists a dirty node v
02.     IF W_v > fαβc*_v, THEN
            // If true, then by equation 7, we have ℓ(v) < L.
03.         Increment the level of v by setting ℓ(v) ← ℓ(v) + 1.
04.     ELSE IF (W_v < c*_v and ℓ(v) > 0), THEN
05.         Decrement the level of v by setting ℓ(v) ← ℓ(v) − 1.
```

Fig. 2. RECOVER().

Consider any node $v \in V$ and suppose that $W_v > f\alpha\beta c^*_v = c_v \geq c_{\min}$. In this event, since $\alpha > 1$, equation 7 implies that $W_v(L) < W_v(\ell(v))$ and hence we have $L > \ell(v)$. In other words, when the procedure described in Figure 2 decides to increment the level of a dirty node v (Step 02), we know for sure that the current level of v is strictly less than L (the highest level in the (α, β)-partition).

Next, consider an edge $e \in \mathcal{E}_v$. If we change $\ell(v)$, then this may change the weight $w(e)$, and this in turn may change the weights $\{W_z\}, z \in \mathcal{V}_e$. Thus, a

single iteration of the WHILE loop in Figure 2 may lead to some clean nodes becoming dirty, and some other dirty nodes becoming clean. If and when the WHILE loop terminates, however, we are guaranteed that every node is clean and that Invariant 5 holds.

Analyzing the amortized update time For each node $v \in V$ and each $i \in \{0, \ldots, L\}$, we store the set of edges $\{e \in \mathcal{E}_v : \ell(e) = i\}$ in a doubly linked list NEIGHBORS$[v, i]$. The update time of our algorithm is dominated by the time taken to update these lists. Next, note that each time the level of an edge changes, we have to update at most f lists (one corresponding to each node $v \in \mathcal{V}_e$). Hence, the time taken to update the lists is given by $f \cdot \delta_l$, where δ_l is the number of times the procedure in Figure 2 changes the level of an edge. Using a carefully chosen potential function, in the full version of the paper we show that $\delta_l \leq t \cdot O(L/\epsilon)$ after t edge insertions/deletions in G starting from an empty graph, for $\alpha = 1 + 1/f + 3\epsilon$ and $\beta = 1 + \epsilon$. This gives the required $O(f\delta_l/t) = O(f \log(m+n)/\epsilon^2)$ bound on the amortized update time.

3 Maintaining a *b*-Matching in a Dynamic Setting

In this section, we will present a dynamic algorithm for the problem specified in Definitions 3, 4 (see Theorem 7). Given any subset of edges $E' \subseteq E$ and any node $v \in V$, let $\mathcal{N}(v, E') = \{u \in V : (u, v) \in E'\}$ denote the set of neighbors of v with respect to the edge-set E', and let $\deg(v, E') = |\mathcal{N}(v, E')|$. Next, consider any "weight" function $w : E' \to \mathbf{R}^+$. For every node $v \in V$, we define $W_v = \sum_{u \in \mathcal{N}(v,E)} w(u, v)$. Finally, for every subset of edges $E' \subseteq E$, we define $w(E') = \sum_{e \in E'} w(e)$. Next, we show how to maintain a "fractional" b-matching.

Theorem 6. *Fix a constant $\epsilon \in (0, 1/4)$, and let $\lambda = 4$. In $O(\log n)$ amortized update time, we can maintain a fractional b-matching $w : E \to [0, 1]$ in $G = (V, E)$ such that:*

$$W_v \leq c_v/(1+\epsilon) \text{ for all nodes } v \in V. \tag{10}$$

$$w(u, v) = 1 \text{ for each edge } (u, v) \in E \text{ with } W_u, W_v < c_v/\lambda. \tag{11}$$

Further, the size of the optimal b-matching in G is $O(1)$ times the sum $\sum_{e \in E} w(e)$.

Proof (Sketch). Note that the fractional b-matching problem is a special case of fractional hypergraph b-matching (Definitions 5, 7) where $\mu = 1$, $m = n^2$, and $f = 2$.

We scale down the capacity of each node $v \in V$ by a factor of $(1 + \epsilon)$, by defining $\tilde{c}_v = c_v/(1 + \epsilon)$ for all $v \in V$. Next, we apply Theorem 2 on the input graph $G = (V, E)$ with $\mu = 1$, $m = n^2$, $f = 2$, and the reduced capacities $\{\tilde{c}_v\}, v \in V$. Let $\{w(e)\}, e \in E$, be the resulting $(f + 1 + \epsilon f)$-maximal matching (see Definition 6). Since $\epsilon < 1/3$ and $f = 2$, we have $\lambda \geq f + 1 + \epsilon f$. Since ϵ is a constant, the amortized update time for maintaining the fractional b-matching becomes $O(f \cdot \log(m+n)/\epsilon^2) = O(\log n)$. Finally, by Theorem 1, the fractional b-matching $\{w(e)\}$ is an $(\lambda f + 1) = 9$-approximate optimal b-matching in G in the

presence of the reduced capacities $\{\tilde{c}_v\}$. But scaling down the capacities reduces the objective of LP (1) by at most a factor of $(1+\epsilon)$. Hence, the size of the optimal b-matching in G is at most $9(1+\epsilon) = O(1)$ times the sum $\sum_{e \in E} w(e)$. This concludes the proof. □

Set $\lambda = 4$ for the rest of this section. We will show how to dynamically convert the fractional b-matching $\{w(e)\}$ from Theorem 6 into an integral b-matching, by losing a constant factor in the approximation ratio. The main idea is to randomly sample the edges $e \in E$ based on their $w(e)$ values. But, first we introduce the following notations.

Say that a node $v \in V$ is "nearly-tight" if $W_v \geq c_v/\lambda$ and "slack" otherwise. Let T denote the set of all nearly-tight nodes. We also partition the node-set V into two subsets: $B \subseteq V$ and $S = V \setminus B$. Each node $v \in B$ is called "big" and has $\deg(v, E) \geq c \log n$, for some large constant $c > 1$. Each node $v \in S$ is called "small" and has $\deg(v, E) < c \log n$. Define $E_B = \{(u, v) \in E : \text{either } u \in B \text{ or } v \in B\}$ to be the subset of edges with at least one endpoint in B, and let $E_S = \{(u, v) \in E : \text{either } u \in S \text{ or } v \in S\}$ be the subset of edges with at least one endpoint in S. We define the subgraphs $G_B = (V, E_B)$ and $G_S = (V, E_S)$.

Overview of our approach. We maintain the following quantitates. (1) A random subset $H_B \subseteq E_B$, and a weight function $w^B : H_B \to [0, 1]$ in the subgraph $G_B(H) = (V, H_B)$, as per Definition 9. (2) A random subset $H_S \subseteq E_S$, and a weight function $w^S : H_S \to [0, 1]$ in the subgraph $G_S(H) = (V, H_S)$, as per Definition 10. (3) A maximal b-matching $M_S \subseteq H_S$ in the subgraph $G_S(H)$, that is, for every edge $(u, v) \in H_S \setminus M_S$, there is a node $q \in \{u, v\}$ such that $\deg(q, M_S) = c_q$. (4) The set of edges $E^* = \{e \in E : w(e) = 1\}$.

We will show that with high probability, one of the edge-sets H_B, M_S, E^* is an $O(1)$-approximation to the optimal b-matching in G. The rest of this section is organized as follows. In Lemma 3 (resp. Lemma 4), we prove some properties of the random set H_B (resp. H_S) and the weight function w^B (resp. w^S). In Lemma 5, we show that the edge-sets H_B, H_S, M_S and E^* can be maintained in a dynamic setting in $O(\log^3 n)$ amortized update time. We prove our main result in Theorem 7.

Definition 9. *Let $Z_B(e) \in \{0, 1\}$ be a random variable such that (a) it is set to one if $e \in H_B$ and zero otherwise, and (b) the following properties are satisfied.*

$$\text{With probability one, } \deg(v, H_B) \leq c_v \text{ for every small node } v \in S. \quad (12)$$

$$\Pr[e \in H_B] = \mathbf{E}[Z_B(e)] = w(e) \text{ for every edge } e \in E_B. \quad (13)$$

$$\forall v \in B, \text{ variables } \{Z_B(u, v)\}, u \in \mathcal{N}(v, E_B), \text{ are mutually independent.} \quad (14)$$

$$\text{For each edge } e \in H_B, \text{ we have } w^B(e) = 1 \quad (15)$$

Definition 10. *Let $Z_S(e) \in \{0, 1\}$ be a random variable such that (a) it is set to one if $e \in H_S$ and zero otherwise, and (b) the following properties hold.*

$$\Pr[e \in H_S] = \mathbf{E}[Z_S(e)] = p_e = \min(1, w(e) \cdot (c\lambda \log n/\epsilon)) \quad \forall e \in E_S. \quad (16)$$

$$\text{The variables } \{Z_S(e)\}, e \in E_S, \text{ are mutually independent.} \quad (17)$$

$$\text{For each edge } e \in H_S, \text{ we have } w^S(e) = \begin{cases} w(e) & \text{if } p_e \geq 1; \\ \epsilon/(c\lambda \log n) & \text{if } p_e < 1. \end{cases} \tag{18}$$

Lemma 3. *For every node $v \in V$, define $W_v^B = \sum_{u \in \mathcal{N}(v,H_B)} w^B(u,v)$. The following conditions hold with high probability. (a) For every node $v \in V$, we have $W_v^B \leq c_v$. (b) For every node $v \in B \cap T$, we have $W_v^B \geq (1-\epsilon) \cdot (c_v/\lambda)$.*

Proof (Sketch). Consider any small node $v \in S$. By equations 12, 15, we have $W_v^B = \deg(v, H_B) \leq c_v$ with high probability. Now, consider any big node $v \in B$. By equations 13, 15 and linearity of expectation, we have $\mathbf{E}[W_v^B] = W_v \leq c_v/(1+\epsilon)$. Furthermore, if $v \in B \cap T$, then we have $\mathbf{E}[W_v^B] = W_v \geq c_v/\lambda$. Since $c_v \geq c \log n$, the Lemma now follows from equation 14 and Chernoff bound. □

Lemma 4. *For every node $v \in V$, define $W_v^S = \sum_{u \in \mathcal{N}(v,H_S)} w^S(u,v)$. The following conditions hold with high probability. (a) For each node $v \in V$, we have $W_v^S \leq c_v$. (b) For each node $v \in S$, we have $deg(v, H_S) = O(\log^2 n)$. (c) For each node $v \in S \cap T$, we have $W_v^S \geq (1-\epsilon) \cdot (c_v/\lambda)$.*

Proof (Sketch). In order to highlight the main idea subject to space constraints, we assume that $p_e < 1$ for every edge $e \in E_S$. First, consider any small node $v \in S$. Since $\mathcal{N}(v, E_S) = \mathcal{N}(v, E)$, from equations 10, 16, 18 and linearity of expectation, we infer that $\mathbf{E}[\deg(v, H_S)] = (c\lambda \log n/\epsilon) \cdot W_v \leq (c\lambda \log n/\epsilon) \cdot (c_v/(1+\epsilon))$. Since $c_v \in [1, c \log n]$, from equation 17 and Chernoff bound we infer that $\deg(v, H_S) \leq (c\lambda \log n/\epsilon) \cdot c_v = O(\log^2 n)$ with high probability. Next, note that $W_v^S = \deg(v, H_S) \cdot (\epsilon/(c\lambda \log n))$. Hence, we also get $W_v^S \leq c_v$ with high probability. Next, suppose that $v \in S \cap T$. In this case, we have $\mathbf{E}[\deg(v, H_S)] = (c\lambda \log n/\epsilon) \cdot W_v \geq (c\lambda \log n/\epsilon) \cdot (c_v/\lambda)$. Again, since this expectation is sufficiently large, applying Chernoff bound we get $\deg(v, H_S) \geq (c\lambda \log n/\epsilon) \cdot (1-\epsilon) \cdot (c_v/\lambda)$ with high probability. It follows that $W_v^S = (\epsilon/(c\lambda \log n)) \cdot \deg(v, H_S) \geq (1-\epsilon) \cdot (c_v/\lambda)$ with high probability.

Finally, applying a similar argument we can show that for every big node $v \in B$, we have $W_v^S \leq c_v$ with high probability. □

Lemma 5. *With high probability, we can maintain the edge-sets H_B, E^*, H_S, and a maximal b-matching M_S in $G_S(H) = (V, H_S)$ in $O(\log^3 n)$-amortized update time.*

Proof (Sketch). We maintain the fractional b-matching $\{w(e)\}$ as per Theorem 6. This requires $O(\log n)$ amortized update time, and starting from an empty graph, t edge insertions/deletions in G lead to $O(t \log n)$ many changes in the edge-weights $\{w(e)\}$ (see Section 2). Thus, we can easily maintain the edge-set $E^* = \{e \in E : w(e) = 1\}$ in $O(\log n)$ amortized update time.

Next, we show to maintain the edge-set H_S. We do this by independently sampling each edge $e \in E_S$ with probability p_e. This probability is completely determined by the weight $w(e)$. So we need to resample the edge each time its weight changes. Thus, the amortized update time for maintaining H_S is $O(\log n)$.

Next, we show how to maintain the maximal b-matching M_S in H_S. Every edge $e \in H_S$ has at least one endpoint in S, and each node $v \in S$ has $\deg(v, H_S) = O(\log^2 n)$ with high probability (see Lemma 4). Due to this fact, for each node $v \in B$, we can maintain the set of its free neighbors $F_v(S) = \{u \in \mathcal{N}(v, H_S) : u \text{ is unmatched in } M_S\}$ in $O(\log^2 n)$ worst case time per update in H_S, with high probability (w.h.p.). Using the sets $\{F_v(S)\}, v \in B$, after each edge insertion/deletion in H_S, we can update the maximal b-matching M_S in $O(\log^2 n)$ worst case time w.h.p. [9]. Since each edge insertion/deletion in G, on average, leads to $O(\log n)$ edge insertions/deletions in H_S, we spend $O(\log^3 n)$ amortized update time for maintaining M_S, w.h.p..

Finally, we show how to maintain the set H_B. The edges $(x, y) \in E_B$ with both endpoints $x, y \in B$ are sampled independently with probability $w(x, y)$. This requires $O(\log n)$ amortized update time. Next, each small node $v \in S$ randomly selects some neighbors $u \in \mathcal{N}(v, E_B)$ and adds the corresponding edges (u, v) to the set H_B, ensuring that $\Pr[(u, v) \in H_B] = w(u, v)$ for all $u \in \mathcal{N}(v, E_B)$ and that $\deg(v, H_B) \leq c_v$. The random choices made by the different small nodes are mutually independent, which implies equation 14. But, for a given node $v \in S$, the random variables $\{Z_B(u, v)\}, u \in \mathcal{N}(v, E_B)$, are completely correlated. They are determined as follows.

In the beginning, we pick a number η_v uniformly at random from the interval $[0, 1)$, and, in a predefined manner, label the set of big nodes as $B = \{v_1, \ldots, v_{|B|}\}$. For each $i \in \{1, \ldots, |B|\}$, we define $a_i(v) = w(v, v_i)$ if $v_i \in \mathcal{N}(v, E_B)$ and zero otherwise. We also define $A_i(v) = \sum_{j=1}^{i} a_j(v)$ for each $i \in \{1, \ldots, |B|\}$ and set $A_0(v) = 0$. At any given point in time, we define $\mathcal{N}(v, H_B) = \{v_i \in B : A_{i-1}(v) \leq k + \eta_v < A_i(v) \text{ for some nonnegative integer } k < c_v\}$. Under this scheme, for every node $v_i \in B$, we have $\Pr[v_i \in \mathcal{N}(v, H_B)] = A_i(v) - A_{i-1}(v) = a_i(v)$. Thus, we get $\Pr[v_i \in \mathcal{N}(v, H_B)] = w(v, v_i)$ for all $v_i \in \mathcal{N}(v, E_B)$, and $\Pr[v_i \in \mathcal{N}(v, H_B)] = 0$ for all $v_i \neq \mathcal{N}(v, E_B)$. Also note that $\deg(v, H_B) \leq \lceil \sum_{v_i \in \mathcal{N}(v, E_B)} w(v, v_i) \rceil \leq \lceil W_v \rceil \leq \lceil c_v/(1+\epsilon) \rceil \leq c_v$. Hence, equations 12, 13 are satisfied.

In the full paper, we show that the sums $\{A_i(v)\}, v \in S, i$, and the sets $\{\mathcal{N}(v, H_B)\}$, $v \in S$, can be maintained using a balanced binary tree data structure in $O(\log^3 n)$ amortized update time. This means that the set H_B can also be maintained in $O(\log^3 n)$ amortized update time. □

Theorem 7. *With high probability, we can maintain an $O(1)$-approximately optimal b-matching in the input graph G in $O(\log^3 n)$ amortized update time.*

Proof (Sketch). We maintain the random sets of edges H_B and H_S, a maximal b-matching M_S in the subgraph $G_S(H) = (V, H_S)$, and the set of edges $E^* = \{e \in E : w(e) = 1\}$ as per Lemma 5. This requires $O(\log^3 n)$ amortized update time with high probability (w.h.p.). We will show that w.h.p., one of the edge-sets E^*, H_B and M_S is an $O(1)$-approximately optimal b-matching in G. But, first, we claim that:

$$w(E^*) + \sum_{v \in B \cap T} W_v + \sum_{v \in S \cap T} W_v \geq w(E) \tag{19}$$

Equation 19 holds since each edge $e \in E \setminus E^*$ has at least one endpoint in T (by equation 11), and hence each edge $e \in E$ contributes at least $w(e)$ to the left hand side and exactly $w(e)$ to the right hand side. Next, note that by Lemmas 3, 4, the weight functions w^B, w^S are fractional b-matchings in G with high probability. For the rest of proof, we condition on this event, and consider three possible cases based on equation 19.

Case 1. $w(E^*) \geq (1/3) \cdot w(E)$. In this case, since $w(e) = 1$ for all $e \in E^*$, Theorem 6 imply that E^* is an $O(1)$-approximately optimal b-matching in G.

Case 2. $\sum_{v \in B \cap T} W_v \geq (1/3) \cdot w(E)$. Here, Lemma 3 and equation 15 imply that: $\sum_{v \in B \cap T} W_v \leq \sum_{v \in B \cap T} c_v \leq \sum_{v \in B \cap T} O(1) \cdot W_v^B \leq O(1) \cdot 2 \cdot w^B(H^B) = O(1) \cdot |H_B|$. Since $O(1) \cdot |H_B| \geq \sum_{v \in B \cap T} W_v \geq (1/3) \cdot w(E)$, Theorem 6 implies that the edge-set H_B is an $O(1)$-approximately optimal b-matching in G.

Case 3. $\sum_{v \in S \cap T} W_v \geq (1/3) \cdot w(E)$. Here, Lemma 4 implies that: $\sum_{v \in S \cap T} W_v \leq \sum_{v \in S \cap T} c_v \leq O(1) \cdot \sum_{v \in S \cap T} W_v^S \leq O(1) \cdot 2 \cdot w^S(H_S) \leq O(1) \cdot |M_S|$. The last inequality holds since M_S is a 1-maximal b-matching in $G_S(H) = (V, H_S)$, and hence we have $w^S(H_S) \leq 3 \cdot |M_S|$ (see Theorem 1). Finally, since $O(1) \cdot |M_S| \geq \sum_{v \in S \cap T} W_v \geq (1/3) \cdot w(E)$, Theorem 6 implies that the edge-set M_S is an $O(1)$-approximately optimal b-matching in G. □

References

1. R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2:198–203, 1981.
2. S. Baswana, M. Gupta, and S. Sen. Fully dynamic maximal matching in $O(\log n)$ update time. In *FOCS*, pages 383–392, 2011.
3. S. Bhattacharya, M. Henzinger, and G. F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Procs. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 785–804, 2015.
4. N. Buchbinder and J. Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009.
5. G. B. Dantzig, L. R. Ford, and D. R. Fulkerson. A primal-dual algorithm for linear programs. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*, pages 171–181. Princeton University Press, 1956.
6. D. Eppstein, Z. Galil, and G. F. Italiano. Dynamic graph algorithms. In M. J. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook, 2nd Edition, Vol. 1*, pages 9.1–9.28. CRC Press, 2009.
7. M. Gupta and R. Peng. Fully dynamic $(1+\epsilon)$-approximate matchings. In *FOCS*, pages 548–557, 2013.
8. H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
9. O. Neiman and S. Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *STOC*, pages 745–754, 2013.
10. K. Onak and R. Rubinfeld. Maintaining a large matching and a small vertex cover. In *STOC*, pages 457–464, 2010.
11. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, NY, USA, 2001.