Information and Computation 261 (2018) 219–239

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Dynamic algorithms via the primal-dual method ☆

Sayan Bhattacharya [a,*], Monika Henzinger [b], Giuseppe Italiano [c]

[a] University of Warwick, Coventry, UK
[b] University of Vienna, Austria
[c] University of Rome Tor Vergata, Italy

ARTICLE INFO

Article history:
Received 28 May 2015
Available online 8 February 2018

Keywords:
Dynamic algorithms
Primal-dual method
Data structures

ABSTRACT

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" [1], was extended and formalized by Dantzig et al. [2] as a general framework for linear programming, and thus it became applicable to a large variety of problems. A few decades later, Bar-Yehuda et al. [3] 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 [4,5]. 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 [6]. 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 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

☆ An extended abstract of this paper appeared in the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015).

* Corresponding author.
E-mail addresses: S.Bhattacharya@warwick.ac.uk (S. Bhattacharya), monika.henzinger@univie.ac.at (M. Henzinger), giuseppe.italiano@uniroma2.it (G. Italiano).

https://doi.org/10.1016/j.ic.2018.02.005
0890-5401/

matching and dynamic vertex cover [7] 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 and the dynamic b-matching. Before proceeding any further, we formally define these problems.

Definition 1.1 *(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 1.2 *(Dynamic Set-Cover).* Consider a dynamic version of the problem specified in Definition 1.1, 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 1.3 *(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 1.4 *(Dynamic b-Matching).* Consider a dynamic version of the problem specified in Definition 1.3, 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 [6,8], 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 [6], where the elements arrive online. There exists algorithms for online set cover that achieve a competitive ratio of $O(\log n \log m)$ [6], and it is also known that this bound is asymptotically tight [9].

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 Theorem 3.2 in Section 3). 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 4.8 in Section 4). 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 [10]). The b-matching problem contains as a special case matching problems, for which many dynamic algorithms are known [11,7,12–14]. 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.

In the static setting, a simple greedy algorithm for the set-cover problem gives an $O(\log n)$ approximation [15], whereas a primal-dual algorithm gives an f-approximation [3]. Both the algorithms run in $O(f \cdot (m+n))$-time. On the other hand, there exists some constant $c > 0$ such that obtaining a $c \log n$-approximation to the set cover problem in polynomial time will imply $P = NP$ [16]. Similarly, under the Unique-Games conjecture, one cannot obtain a better than f-approximation to the set cover problem in polynomial time [17].

For the maximum b-matching problem, the best known exact algorithm runs in $O(mn\log n)$-time [18] in the static setting, where n (resp. m) is the number of nodes (resp. edges) in the graph. Very recently, Ahn and Guha [19] presented another static algorithm that runs in $O(m\cdot \text{poly}(\delta^{-1},\log n))$-time and returns a $(1+\delta)$-approximation for maximum b-matching, for any $\delta > 0$.

Roadmap for the rest of the paper. We first define a problem called "fractional hypergraph b-matching" (see Definitions 1.5 and 1.6). In Section 2, we show how to maintain a fractional hypergraph b-matching in a dynamic setting. In Section 3, we use our result from Section 2 to design a dynamic algorithm for set cover. Finally, in Section 4 we present our result for dynamic b-matching.

Definition 1.5 *(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.

Definition 1.6 *(Dynamic Fractional Hypergraph b-Matching).* Consider a dynamic version of the problem specified in Definition 1.5, 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.

2. Maintaining a fractional hypergraph *b*-matching in a dynamic setting

2.1. Preliminaries

We first define a linear program for fractional hypergraph b-matching (Definition 1.5). Next, we define the concept of a "λ-maximal" solution of this LP (Definition 2.1) and prove the approximation guarantee for such a solution (Theorem 2.2). Our main result is summarized in Theorem 2.3 and Corollary 2.4.

Below, we write a linear program for a fractional hypergraph b-matching.

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

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

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

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

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

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

We next define the concept of a "λ-maximal" solution.

Definition 2.1. A feasible solution to LP (1) is λ-maximal (for $\lambda \geq 1$) iff for every edge $e \in E$ with $x(e) < \mu$, there is some node $v \in \mathcal{V}_e$ such that $\sum_{e'\in\mathcal{E}_v} x(e') \geq c_v/\lambda$.

Theorem 2.2. *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 $\text{OPT}/(\lambda f + 1)$.*

Proof. Let $\{x^*(e)\}$ be a λ-maximal solution to the primal LP. Construct a dual solution $\{y^*(v), z^*(e)\}$, as follows. For every $v \in V$, set $y^*(v) = 1$ if $\sum_{e\in\mathcal{E}_v} x^*(e) \geq c_v/\lambda$, and $y^*(v) = 0$ otherwise. For every $e \in E$, set $z^*(e) = 1$ if $x^*(e) = \mu$ and $z^*(e) = 0$ otherwise.

Consider the dual constraint corresponding to any edge $e' \in E$. Since the primal solution $\{x^*(e)\}$ is λ-maximal, either $x^*(e) = \mu$ or there is some $v' \in \mathcal{V}_{e'}$ for which $y^*(v') = 1$. In the former case we have $z^*(e) = 1$, whereas in the latter case we have $y^*(v') = 1$. Hence, the dual constraint under consideration is satisfied. This shows that the values $\{y^*(v), z^*(e)\}$, constitute a feasible dual solution. Next, we infer that:

$$\begin{aligned}
&\sum_{v\in V} c_v \cdot y^*(v) + \sum_{e\in E} \mu \cdot z^*(e) \\
&= \sum_{v\in V: y^*(v)=1} c_v + \sum_{e\in E: z^*(e)=1} \mu && (7)\\
&\le \sum_{v\in V: y^*(v)=1} \lambda \cdot \sum_{e\in \mathcal{E}_v} x^*(e) + \sum_{e\in E: z^*(e)=1} x^*(e) && (8)\\
&\le \sum_{v\in V} \lambda \cdot \sum_{e\in \mathcal{E}_v} x^*(e) + \sum_{e\in E} x^*(e) \\
&\le \lambda \cdot f \cdot \sum_{e\in E} x^*(e) + \sum_{e\in E} x^*(e) && (9)\\
&= (\lambda f + 1) \cdot \sum_{e\in E} x^*(e)
\end{aligned}$$

Equation (7) holds since $y^*(v) \in \{0,1\}$ for all $v \in V$ and $z^*(e) \in \{0,1\}$ for all $e \in E$. Equation (8) holds since $y^*(v) = 1$ only if $\sum_{e\in\mathcal{E}_v} x^*(e) \ge c_v/\lambda$, and since $x^*(e) = \mu$ for all $e \in E$ with $z^*(e) = 1$. Equation (9) holds since each edge can be incident upon at most f nodes.

Thus, we have constructed a feasible dual solution whose objective is at most $(\lambda f + 1)$-times the objective of the λ-maximal primal solution. The theorem now follows from weak duality.

Our main result is summarized below. For the rest of Section 2, we focus on proving Theorem 2.3.

Theorem 2.3. *We can maintain a $(f + 1 + \epsilon f)$-maximal solution to the dynamic fractional hypergraph b-matching problem in $O(f \cdot \log(m+n)/\epsilon^2)$ amortized update time.*

Corollary 2.4. *We can maintain an $O(f^2)$-approximate solution to the dynamic hypergraph b-matching problem in $O(f \log(m+n)/\epsilon^2)$ amortized update time.*

Proof. Follows from Theorem 2.2 and Theorem 2.3.

2.2. The (α, β)-partition and its properties

For the time being, we restrict ourselves to 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 an initial primal solution with $x(e) \leftarrow 0$ for all $e \in E$, and define $F \leftarrow E$.
- 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.

In Fig. 1, we define 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 $c_{\min} = \min_{v\in V} c_v$ to be the minimum capacity over all the nodes. Without any loss of generality, assume that $0 < c_{\min} < m\mu$. We can make this assumption because of the following two reasons. (1) If $c_{min} = 0$, then we can delete all the edges incident on the nodes $v \in V$ with $c_v = 0$ and then solve the new instance of the problem. (2) If $c_{\min} \ge m\mu$, then we can set $x(e) \leftarrow \mu$ for all edges $e \in E$ and get an optimal primal solution.

Henceforth, we fix two parameters $\alpha \ge \beta > 1$, and define $L = \lceil \log_\beta(m\mu\alpha/c_{\min}) \rceil$. Note that $L \ge 2$.

Claim 2.5. *If we set $x(e) \leftarrow \mu \cdot \beta^{-L}$ for all $e \in E$, then we get a feasible primal solution.*

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

Our new algorithm is described in Fig. 1. We initialize our primal solution by setting $x(e) \leftarrow \mu\beta^{-L}$ for every edge $e \in E$, as per Claim 2.5. We call a node v *nearly-tight* if its corresponding primal constraint is tight within a factor of $f\alpha\beta$, and *slack* otherwise. Furthermore, we call an edge *nearly-tight* if it is incident upon some nearly tight node, and *slack* otherwise.

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().

Let $V_L \subseteq V$ and $E_L \subseteq E$ 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 $\sum_{e\in\mathcal{E}_v} x(e) \leq \beta \cdot c^*_v$ for every node $v \in V_0$. Furthermore, at the end of the algorithm, every edge $e \in E \setminus E_0$ is nearly-tight, and every edge $e \in E_0$ has weight $x(e) = \mu$. We, therefore, reach the following conclusion.

Claim 2.6. *The algorithm described in Fig. 1 returns an $(f\alpha\beta)$-maximal solution to the fractional hypergraph b-matching problem with the additional property that $c^*_v \leq \sum_{e\in\mathcal{E}_v} x(e) \leq \beta \cdot c^*_v$ for every node $v \in \bigcup_{k=1}^{L} V_k$, and $\sum_{e\in\mathcal{E}_v} x(e) \leq \beta \cdot c^*_v$ for every node $v \in V_0$.*

Our goal is to make a variant of the procedure in Fig. 1 work in a dynamic setting. Towards this end, we introduce the concept of an (α, β)-partition (see Definition 2.7) satisfying a certain invariant (see Invariant 2.9). The reader is encouraged to notice the similarities between this construct and the output of the procedure in Fig. 1.

Definition 2.7. An (α, β)-*partition* of the graph G partitions its node-set V into subsets $V_0 \ldots V_L$, where $L = \lceil \log_\beta(m\mu\alpha/c_{\min})\rceil$ and $\alpha, \beta > 1$. For $i \in \{0, \ldots, L\}$, we identify the subset V_i as the i^{th} "level" of this partition, and call i the *level* $\ell(v)$ of a node 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 the edge e.

Given an (α, β)-partition, let $\mathcal{E}_v(i) \subseteq \mathcal{E}_v$ denote the set of edges incident to v that are in the i^{th} level, and let $\mathcal{E}_v(i, j) \subseteq \mathcal{E}_v$ denote the set of edges incident to v whose levels are in the range $[i, j]$.

$$\mathcal{E}_v(i) = \{e \in \mathcal{E}_v : \ell(e) = i\} \quad \forall v \in V; i \in \{0, \ldots, L\} \tag{10}$$

$$\mathcal{E}_v(i, j) = \bigcup_{k=i}^{j} \mathcal{E}_v(k) \quad \forall v \in V; i, j \in \{0, \ldots, L\}, i \leq j. \tag{11}$$

Similarly, we define the notations $\mathcal{D}_v$ and $\mathcal{D}_v(i, j)$.

$$\mathcal{D}_v = |\mathcal{E}_v| \tag{12}$$

$$\mathcal{D}_v(i) = |\mathcal{E}_v(i)| \tag{13}$$

$$\mathcal{D}_v(i, j) = |\mathcal{E}_v(i, j)| \tag{14}$$

Given an (α, β)-partition, 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 $W_v(i)$. 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*. Thus, we have $W_v = W_v(\ell(v))$. Since the weight of an edge e in the hierarchical partition is given by $w(e) = \mu \cdot \beta^{-\ell(e)}$, we derive the following equations for all nodes $v \in V$.

$$W_v = \sum_{e\in\mathcal{E}_v} \mu \cdot \beta^{-\ell(e)}. \tag{15}$$

$$W_v(i) = \sum_{e\in\mathcal{E}_v} \mu \cdot \beta^{-\max(\ell(e), i)} \quad \forall i \in \{0, \ldots, L\}. \tag{16}$$

Fix any node $v \in V$, and focus on the value of $W_v(i)$ as we go down from the highest level $i = L$ to the lowest level $i = 0$. Lemma 2.8 states that $W_v(i) \leq c_{\min}/\alpha$ when $i = L$, that $W_v(i)$ keeps increasing as we go down the levels one after another, and that $W_v(i)$ increases by at most a factor of β between consecutive levels.

Lemma 2.8. *An (α, β)-partition satisfies the following conditions for all nodes $v \in V$.*

$$W_v(L) \leq c_{\min}/\alpha \tag{17}$$

$$W_v(L) \leq \cdots \leq W_v(i) \leq \cdots \leq W_v(0) \tag{18}$$

$$W_v(i) \leq \beta \cdot W_v(i+1) \ \ \forall i \in \{0, \ldots, L-1\}. \tag{19}$$

Proof. Fix any (α, β)-partition and any node $v \in V$. We prove the first part of the lemma as follows.

$$W_v(L) = \sum_{e \in \mathcal{E}_v} \mu \cdot \beta^{-\max(\ell(e), L)} = \sum_{e \in \mathcal{E}_v} \mu \cdot \beta^{-L} \leq m\mu \cdot \beta^{-L} \leq m\mu \cdot \beta^{-\log_\beta(m\mu\alpha/c_{\min})} = c_{\min}/\alpha.$$

We now fix any level $i \in \{0, \ldots, L-1\}$ and show that the (α, β)-partition satisfies equation (18).

$$W_v(i+1) = \sum_{e \in \mathcal{E}_v} \mu \cdot \beta^{-\max(\ell(e), i+1)} \leq \sum_{e \in \mathcal{E}_v} \mu \cdot \beta^{-\max(\ell(e), i)} = W_v(i).$$

Finally, we prove equation (19).

$$\begin{aligned} W_v(i) &= \sum_{e \in \mathcal{E}_v} \mu \cdot \beta^{-\max(\ell(e), i)} = \mu \cdot \beta \cdot \sum_{e \in \mathcal{E}_v} \beta^{-1-\max(\ell(e), i)} \\ &\leq \mu \cdot \beta \cdot \sum_{e \in \mathcal{E}_v} \beta^{-\max(\ell(e), i+1)} = \beta \cdot W_v(i+1) \end{aligned}$$

We will maintain a specific type of (α, β)-partition, where each node is assigned to a level in a way that satisfies the following Invariant 2.9. This invariant is a relaxation of the bounds on $\sum_{e \in \mathcal{E}_v} x(e)$ for every node v stated in Claim 2.6.

Invariant 2.9. Define $c^*_v = c_v/(f\alpha\beta)$. For every node $v \in V \setminus V_0$, it holds that $c^*_v \leq W_v \leq f\alpha\beta \cdot c^*_v$ and for every node $v \in V_0$ it holds that $W_v \leq f\alpha\beta \cdot c^*_v$.

Theorem 2.10. *Consider an (α, β)-partition that satisfies Invariant 2.9. The edge-weights $\{w(e)\}, e \in E$, give an $(f\alpha\beta)$-maximal solution to LP (1).*

Proof. By Invariant 2.9, we have $W_v \leq (f\alpha\beta) \cdot c^*_v = c_v$ for every node $v \in V$. Next, note that $w(e) \leq \mu$ for every edge $e \in E$. Thus, the weights $\{w(e)\}, e \in E$, define a feasible solution to LP (1).

We claim that for every edge $e \in E$ with $w(e) < \mu$, there is some node $v \in \mathcal{V}_e$ for which $W_v \geq c_v/(f\alpha\beta)$. This will imply that the weights $\{w(e)\}, e \in E$, form an $(f\alpha\beta)$-maximal feasible solution to the primal LP.

To prove the claim, consider any edge $e \in E$ with $w(e) < \mu$. Since $w(e) = \mu\beta^{-\ell(e)}$, this implies that $\ell(e) > 0$. Let $v \in \arg\max_{u \in \mathcal{V}_e} \{\ell(u)\}$. Note that $\ell(e) = \ell(v)$. This implies that $\ell(v) > 0$. Hence, by Invariant 2.9, we have $W_v \geq c^*_v = c_v/(f\alpha\beta)$. This concludes the proof of the theorem.

2.3. The algorithm: handling the insertion/deletion of an edge

We now show how to maintain an (α, β)-partition under edge insertions and deletions. A node is called *dirty* if it violates Invariant 2.9, and *clean* otherwise. At the beginning of the algorithm the edge-set E is empty, and, thus, every node is initially clean and at level zero. 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 $\{W_v\}, v \in \mathcal{V}_e$. Due to this change, the nodes $v \in \mathcal{V}_e$ might become dirty. To recover from this, we call the subroutine in Fig. 2, which works as follows

Consider any node $v \in V$ and suppose that $W_v > f\alpha\beta c^*_v = c_v \geq c_{\min}$. In this event, the algorithm increments the level of the node. Since $\alpha > 1$, equation (17) implies that $W_v(L) < W_v(\ell(v))$ and, hence, we have $L > \ell(v)$. In other words, when the procedure described in Fig. 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 Fig. 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 2.9 holds.

```
01. While there exists a dirty node v
02.     If W_v > f α β c*_v, Then
            // If true, then by equation (17), 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().

2.4. Data structures used by our algorithm

We now describe the relevant data structures. We maintain for each node $v \in V$:

- A counter Level[v] to keep track of the current level of v. Thus, we set Level[v] $\leftarrow \ell(v)$.
- A counter Weight[v] to keep track of the weight of v. Thus, we set Weight[v] $\leftarrow W_v$.
- For every level $i >$ Level[v], we store the set of edges $\mathcal{E}_v(i)$ in the form of a doubly linked list Incident-Edges$_v[i]$. For every level $i \leq$ Level[v], the list Incident-Edges$_v[i]$ is empty.
- For level $i =$ Level[v], we store the set of edges $\mathcal{E}_v(0, i)$ in the form of a doubly linked list Incident-Edges$_v[0, i]$. For every level $i \neq$ Level[v], the list Incident-Edges$_v[0, i]$ is empty.

When the graph gets updated due to an edge insertion/deletion, we may discover that a node violates Invariant 2.9. Recall that such a node is called *dirty*, and we store the set of such nodes as a doubly linked list Dirty-nodes. For every node $v \in V$, we maintain a bit Status[v] $\in$ {dirty, clean} that indicates if the node is dirty or not. Every dirty node stores a pointer to its position in the list Dirty-nodes.

The collection of linked lists $\bigcup_{i=0}^{L}$ {Incident-Edges$_v[0, i]$, Incident-Edges$_v[i]$} is denoted by the phrase *"incidence lists of v"*. For every edge $e \in E$, we maintain a counter Level[e] to keep track of $\ell(e)$. Furthermore, for every edge $e \in E$, we maintain $|\mathcal{V}_e|$ bidirectional pointers corresponding to the nodes in $\mathcal{V}_e$. The pointer corresponding to a node $v \in \mathcal{V}_e$ points to the position of e in the incidence lists of v. Using these pointers, we can update the incidence lists of the relevant nodes when the edge e is inserted into (resp. deleted from) the graph, or when some node $v \in \mathcal{V}_e$ increases (resp. decreases) its level by one. Thus, we can always efficiently find a dirty node in step (01) of Fig. 2, and quickly recalculate the value of W_v.

2.5. Bounding the amortized update time

We devote this section to the proof of the following theorem.

Theorem 2.11. *Fix any $\epsilon \in (0, 1)$, $\alpha = 1 + 1/f + 3\epsilon$ and $\beta = 1 + \epsilon$. Starting from an empty graph, we can maintain an (α, β) partition in G satisfying Invariant 2.9 in $O(f \log(m+n)/\epsilon^2)$ amortized update time.*

The main idea is as follows. After an edge insertion or deletion the data structure can be updated in time $O(1)$, plus the time to adjust the levels of the nodes, i.e., the time for procedure RECOVER. To bound the latter quantity we note that each time the level of an edge $e \in E$ 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 Fig. 2 changes the level of an edge. Below, we show that $\delta_l \leq t \cdot O(L/\epsilon) = t \cdot O(\log(m+n)/\epsilon^2)$ after t edge insertions/deletions in G starting from an empty graph. This gives the required $O(f\delta_l/t) = O(f \log(m+n)/\epsilon^2)$ bound on the amortized update time.

Hence, to complete the proof of Theorem 2.11, we need to give an amortized bound on *the number of times we have to change the level (or, equivalently, the weight) of an already existing edge*. During a single iteration of the WHILE loop in Fig. 2, this number is exactly $\mathcal{D}_v(0, i)$ when node v goes from level i to level $i + 1$, and at most $\mathcal{D}_v(0, i)$ when node v goes from level i to level $i - 1$.

Specifically, we devote the rest of this section to the proof of Theorem 2.12, which implies that on average we change the weights of $O(L/\epsilon) = O(\log(m+n)/\epsilon^2)$ edges per update in G.

Theorem 2.12. *Set $\alpha \leftarrow 1 + 1/f + 3\epsilon$ and $\beta \leftarrow 1 + \epsilon$. In the beginning, when G is an empty graph, initialize a counter* Count $\leftarrow 0$. *Subsequently, each time we change the weight of an already existing edge in the hierarchical partition, set* Count $\leftarrow$ Count $+ 1$. *Then* Count $= O(tL/\epsilon)$ *just after we handle the t^{th} update in G.*

Recall that the level of an edge e is defined as $\ell(e) = \max_{v \in \mathcal{V}_e}(\ell(v))$. Consider the following thought experiment. We have a *bank account*, and initially, when there are no edges in the graph, the bank account has a balance of zero dollars. For each subsequent edge insertion/deletion, at most $3L/\epsilon$ dollars are deposited to the bank account; and every time our algorithm changes the level of an already existing edge, one dollar is withdrawn from it. We show that the bank account never runs out of money, and this implies that Count $= O(tL/\epsilon)$ after t edge insertions/deletions starting from an empty graph.

Let $\mathcal{B}$ denote the total amount of money (or potential) in the bank account at the present moment. We keep track of $\mathcal{B}$ by distributing an ϵ-fraction of it among the nodes and the current set of edges in the graph.

$$\mathcal{B} = (1/\epsilon) \cdot \left(\sum_{e \in E} \Phi(e) + \sum_{v \in V} \Psi(v) \right) \tag{20}$$

In the above equation, the amount of money (or potential) associated with an edge $e \in E$ is given by $\Phi(e)$, and the amount of money (or potential) associated with a node $v \in V$ is given by $\Psi(v)$. At every point in time, the potentials $\{\Phi(e), \Psi(v)\}$ will be determined by two invariants. But, before stating the invariants, we need to define the concepts of "active" and "passive" nodes.

Definition 2.13. Consider any node $v \in V$. In the beginning, there is no edge incident upon the node v, and we initialize a counter $\kappa_v \leftarrow 0$. Subsequently, whenever an edge-insertion occurs in the graph, if the inserted edge is incident upon v, then we set $\kappa_v \leftarrow \kappa_v + 1$. At any given time-step, we say that a node $v \in V$ is *active* if $\mu\kappa_v \geq c_v$ and *passive* otherwise.

It is easy to check that if a node is active at time-step t, then it will remain active at every time-step $t' > t$. A further interesting consequence of the above definition is that a passive node is always at level zero, as shown in the lemma below.

Lemma 2.14. *At any given time-step, if a node $v \in V$ is passive, then we have $\ell(v) = 0$.*

Proof. We prove this by induction. Let $\ell^{(t)}(v)$ and $\kappa_v^{(t)}$ respectively denote the level of the node v and the value of the counter κ_v at time-step t. Further, let $W_v^{(t)}$ denote the value of W_v at time-step t. Initially, at time-step $t = 0$, the graph is empty, we have $W_v^{(0)} = 0$, and hence $\ell^{(0)}(v) = 0$. Now, by induction hypothesis, suppose that at time-step t the node v is passive and $\ell^{(t)}(v) = 0$, and, furthermore, suppose that the node v remains passive at time-step $(t+1)$. Given this hypothesis, we claim that $\ell^{(t+1)}(v) = 0$. The lemma will follow if we can prove the claim.

To prove the claim, note that since the node v is passive at time-step $(t+1)$, we have $\kappa_v^{(t+1)}\mu < c_v = f\alpha\beta c_v^*$. Since the node v has at most $\kappa_v^{(t+1)}$ edges incident to it at time-step $(t+1)$, and since each of these edges has weight at most μ, we have $W_v^{(t+1)} \leq \kappa_v^{(t+1)}\mu < f\alpha\beta c_v^*$. Now, recall Fig. 2. Since $\ell^{(t)}(v) = 0$ and since $W_v^{(t+1)} < f\alpha\beta c_v^*$, the node v can never become dirty during the execution of the procedure in Fig. 2 after the edge insertion/deletion that occurs at time-step $(t+1)$. Thus, the node v will not change its level, and we will have $\ell^{(t+1)}(v) = 0$. This concludes the proof.

We are now ready to state the invariants that define edge and node potentials.

Invariant 2.15. For every edge $e \in E$, we have:

$$\Phi(e) = (1+\epsilon) \cdot (L - \ell(e))$$

Invariant 2.16. Recall Definition 2.13. For every node $v \in V$, we have:

$$\Psi(v) = \begin{cases} \left(\beta^{\ell(v)+1}/(f\mu(\beta-1))\right) \cdot \max\left(0, f\alpha \cdot c_v^* - W_v\right) & \text{if } v \text{ is active;} \\ (\beta/(f(\beta-1)) \cdot \kappa_v & \text{otherwise.} \end{cases}$$

When the algorithm starts, the graph has zero edges, and all the nodes $v \in V$ are passive and at level 0 with $W_v = 0$ and $\kappa_v = 0 < c_v/\mu$. At that moment, Invariant 2.16 sets $\Psi(v) = 0$ for all nodes $v \in V$. Consequently, equation (20) implies that $\mathcal{B} = 0$. Theorem 2.12, therefore, will follow if we can prove the next two lemmas. Their proofs appear in Section 2.6 and Section 2.7 respectively.

Lemma 2.17. *Consider the insertion (resp. deletion) of an edge e in E. It creates (resp. destroys) the weight $w(e) = \mu \cdot \beta^{-\ell(e)}$, creates (resp. destroys) the potential $\Phi(e)$, and changes the potentials $\{\Psi(v)\}$, $v \in \mathcal{V}_e$. Due to these changes, the total potential $\mathcal{B}$ increases by at most $3L/\epsilon$.*

Lemma 2.18. *During every single iteration of the* While *loop in Fig. 2, the total increase in* Count *is no more than the net decrease in the potential $\mathcal{B}$.*

2.6. Proof of Lemma 2.17

Edge-insertion. Suppose that an edge e is inserted into the graph at time-step t. Then the potential $\Phi(e)$ is created and gets a value of at most $(1+\epsilon)L$ units. Now, fix any node $v \in \mathcal{V}_e$, and consider three possible cases.

Case 1. The node v was passive at time-step $(t-1)$ and remains passive at time-step t. In this case, due to the edge-insertion, the potential $\Psi(v)$ increases by $\beta/(f(\beta-1))$.

Case 2. The node v was passive at time-step $(t-1)$ and becomes active at time-step t. In this case, we must have: $c_v - \mu \leq \mu\kappa_v^{(t-1)} < c_v \leq \mu\kappa_v^{(t)}$. By Invariant 2.16, just before the insertion of the edge e we had:

$$\begin{aligned}\Psi(v) &= \{\beta/(f\mu(\beta-1))\} \cdot \mu\kappa_v^{(t-1)} \\ &\geq \{\beta/(f\mu(\beta-1))\} \cdot (c_v - \mu) \end{aligned} \tag{21}$$

Since the node v was passive at time-step $(t-1)$, by Lemma 2.14 we infer that $\ell^{(t-1)}(v) = 0$. Hence, by Invariant 2.16, just after the insertion of the edge e we get:

$$\begin{aligned}\Psi(v) &= \{\beta/(f\mu(\beta-1))\} \cdot \max\left(0,\, f\alpha \cdot c_v^* - W_v\right) \\ &\leq \{\beta/(f\mu(\beta-1))\} \cdot (f\alpha c_v^*) \\ &\leq \{\beta/(f\mu(\beta-1))\} \cdot c_v \end{aligned} \tag{22}$$

By equations (21), (22), the potential $\Psi(v)$ increases by at most $\{\beta/(f\mu(\beta-1))\} \cdot (c_v - (c_v - \mu)) = \{\beta/(f(\beta-1))\}$.

Case 3. The node v was active at time-step $(t-1)$. In this case, clearly the node v remains active at time-step t, the weight W_v increases, and hence the potential $\Psi(v)$ can only decrease.

From the above discussion, we conclude that the potential $\Psi(v)$ increases by at most $\beta/(f(\beta-1))$ for every node $v \in \mathcal{V}_e$. Since $|\mathcal{V}_e| \leq f$, this accounts for a net increase of at most $f \cdot \beta/(f(\beta-1)) = \beta/(\beta-1) = \beta/\epsilon \leq L/\epsilon$. Finally, recall that the potential $\Phi(e)$ is created and gets a value of at most $(1+\epsilon)L \leq 2L/\epsilon$ units. Thus, the net increase in the potential $\mathcal{B}$ is at most $L/\epsilon + 2L/\epsilon = 3L/\epsilon$.

Edge-deletion. If an edge e is deleted from E, then the potential $\Phi(e)$ is destroyed. The weight W_v of each node $v \in \mathcal{V}_e$ decreases by at most $\mu \cdot \beta^{-\ell(v)}$. Furthermore, no passive node becomes active due to this edge-deletion, and, in particular, the counter κ_v remains unchanged for every node $v \in V$. Hence, each of the potentials $\{\Psi(v)\}$, $v \in \mathcal{V}_e$, increases by at most $\beta^{\ell(v)+1}/(f\mu(\beta-1)) \cdot \mu\beta^{-\ell(v)} = \beta/(f(\beta-1)) = ((1+1/\epsilon)/f) \leq 2L/(\epsilon f)$. The potentials of the remaining nodes and edges do not change. Since $|\mathcal{V}_e| \leq f$, by equation (20), the net increase in $\mathcal{B}$ is at most $2L/\epsilon \leq 3L/\epsilon$.

2.7. Proof of Lemma 2.18

Throughout this section, fix a single iteration of the While loop in Fig. 2 and suppose that it changes the level of a dirty node v by one unit. We use the superscript 0 (resp. 1) on a symbol to denote its state at the time instant immediately prior to (resp. after) that specific iteration of the While loop. Further, we preface a symbol with δ to denote the net decrease in its value due to that iteration. For example, consider the potential $\mathcal{B}$. We have $\mathcal{B} = \mathcal{B}^0$ immediately before the iteration begins, and $\mathcal{B} = \mathcal{B}^1$ immediately after iteration ends. We also have $\delta\mathcal{B} = \mathcal{B}^0 - \mathcal{B}^1$.

A change in the level of node v does not affect the potentials of the edges $e \in E \setminus \mathcal{E}_v$. This observation, coupled with equation (20), gives us the following guarantee.

$$\delta\mathcal{B} = (1/\epsilon) \cdot \left(\delta\Psi(v) + \sum_{e \in \mathcal{E}_v} \delta\Phi(e) + \sum_{u \in V \setminus \{v\}} \delta\Psi(u) \right) \tag{23}$$

Remark. Since the node v is changing its level, it must be active. Hence, by Invariant 2.16, we must have $\Psi(v) = \beta^{\ell(v)+1}/(f\mu(\beta-1)) \cdot \max(0, f\alpha c_v^* - W_v)$. We will use this observation multiple times throughout the rest of this section.

We divide the proof of Lemma 2.18 into two possible cases, depending upon whether the concerned iteration of the While loop increments or decrements the level of v. The main approach to the proof remains the same in each case. We first give an upper bound on the increase in Count due to the iteration. Next, we separately lower bound each of the following quantities: $\delta\Psi(v)$, $\delta\Phi(e)$ for all $e \in \mathcal{E}_v$, and $\delta\Psi(u)$ for all $u \in V \setminus \{v\}$. Finally, applying equation (23), we derive that $\delta\mathcal{B}$ is sufficiently large to pay for the increase in Count.

Remark. Note that $\ell^0(u) = \ell^1(u)$ for all nodes $u \in V \setminus \{v\}$, and $\mathcal{E}_u^0 = \mathcal{E}_u^1$ for all nodes $u \in V$. Thus, we will use the symbols $\ell(u)$ and $\mathcal{E}_u$ without any ambiguity for all such nodes.

Case 1: The level of the node v increases from k to $(k+1)$.

Claim 2.19. *We have $\ell^0(e) = k$ and $\ell^1(e) = k+1$ for every edge $e \in \mathcal{E}_v^0(0,k)$.*

Proof. Consider edge $e \in \mathcal{E}_v^0(0,k)$. Since $e \in \mathcal{E}_v^0(0,k)$, we have $\ell^0(e) \leq k$. Since $\ell^0(v) = k$ and $e \in \mathcal{E}_v$, we must have $\ell^0(e) = k$. Finally, since $\ell^1(u) = \ell^0(u)$ for all nodes $u \in V \setminus \{v\}$, we conclude that $\ell^1(e) = \ell^1(v) = k+1$.

Claim 2.20. *We have $\ell^0(e) = \ell^1(e)$ for every edge $e \in \mathcal{E}_v^0(k+1, L)$.*

Proof. Consider any edge $e \in \mathcal{E}_v^0(k+1, L)$. Since $\ell^0(e) \geq k+1$ and $\ell^0(v) = k$, there must be some node $u \in V \setminus \{v\}$ such that $\ell^0(u) \geq k+1$, $e \in \mathcal{E}_u$ and $\ell^0(e) = \ell^0(u)$. Since $\ell^1(u) = \ell^0(u) \geq k+1$ and $\ell^1(v) = k+1$, we infer that $\ell^1(e) = \ell^1(u) = \ell^0(e)$.

Claim 2.21. *We have $\textsc{Count}^1 - \textsc{Count}^0 = \mathcal{D}_v^0(0,k)$.*

Proof. This follows immediately from Claims 2.19 and 2.20.

Claim 2.22. *We have $\delta\Psi(v) = 0$.*

Proof. Step 03 (Fig. 2) is executed only when $W_v^0 = W_v^0(k) > f\alpha\beta \cdot c_v^*$. Next, from Lemma 2.8 we infer that $W_v^1 = W_v^0(k+1) \geq \beta^{-1} \cdot W_v^0(k) > f\alpha c_v^*$. Since both $W_v^0, W_v^1 > f\alpha c_v^*$, we get: $\Psi^0(v) = \Psi^1(v) = 0$. It follows that $\delta\Psi(v) = \Psi^0(v) - \Psi^1(v) = 0$.

Claim 2.23. *For every edge $e \in \mathcal{E}_v$, we have:*

$$\delta\Phi(e) = \begin{cases} 1+\epsilon & \text{if } e \in \mathcal{E}_v^0(0,k); \\ 0 & \text{if } e \in \mathcal{E}_v^0(k+1, L). \end{cases}$$

Proof. If $e \in \mathcal{E}_v^0(0,k)$, then we have $\ell^0(e) = k$ and $\ell^1(e) = k+1$ (see Claim 2.19). Hence, we have $\Phi^0(e) = (1+\epsilon) \cdot (L-k)$ and $\Phi^1(e) = (1+\epsilon) \cdot (L-k-1)$. It follows that $\delta\Phi(e) = \Phi^0(e) - \Phi^1(e) = (1+\epsilon)$.

In contrast, if $e \in \mathcal{E}_v^0(k+1, L)$, then Claim 2.20 implies that $\ell^0(e) = \ell^1(e)$. Accordingly, we have $\Phi^0(e) = \Phi^1(e) = (1+\epsilon) \cdot (L - \ell^0(e))$. Hence, we get $\delta\Phi(e) = \Phi^0(e) - \Phi^1(e) = 0$.

Claim 2.24. *For every node $u \in V \setminus \{v\}$, we have:*

$$\delta\Psi(u) \geq -(1/f) \cdot |\mathcal{E}_u \cap \mathcal{E}_v^0(0,k)|$$

Proof. Consider any node $u \in V \setminus \{v\}$. If the node u is passive, then we have $\delta\Psi(u) = 0$, and the claim is trivially true. Thus, for the rest of the proof we assume that the node u is active.

Clearly, we have $\ell^0(e) = \ell^1(e)$ for each edge $e \in \mathcal{E}_u \setminus \mathcal{E}_v$. Hence, we get $\delta w(e) = 0$ for each edge $\mathcal{E}_u \setminus \mathcal{E}_v$. Next, by Claim 2.20, we have $\ell^0(e) = \ell^1(e)$ for each edge $e \in \mathcal{E}_u \cap \mathcal{E}_v^0(k+1, L)$. Thus, we get $\delta w(e) = 0$ for each edge $e \in \mathcal{E}_u \cap \mathcal{E}_v^0(k+1, L)$. We therefore conclude that:

$$\begin{aligned}
\delta W_u &= \sum_{e \in \mathcal{E}_u \setminus \mathcal{E}_v} \delta w(e) + \sum_{e \in \mathcal{E}_u \cap \mathcal{E}_v^0(k+1,L)} \delta w(e) + \sum_{e \in \mathcal{E}_u \cap \mathcal{E}_v^0(0,k)} \delta w(e) \\
&= \sum_{e \in \mathcal{E}_u \cap \mathcal{E}_v^0(0,k)} \delta w(e) \\
&= |\mathcal{E}_u \cap \mathcal{E}_v^0(0,k)| \cdot \mu \cdot (\beta^{-k} - \beta^{-(k+1)}) \\
&= |\mathcal{E}_u \cap \mathcal{E}_v^0(0,k)| \cdot \mu \cdot (\beta - 1)/\beta^{k+1}
\end{aligned}$$

Using this observation, we infer that:

$$\begin{aligned}
\delta\Psi(u) &\geq -\left(\beta^{\ell(u)+1}/(f\mu(\beta-1))\right) \cdot \delta W_u \\
&= -\left(\beta^{\ell(u)+1}/(f\mu(\beta-1))\right) \cdot |\mathcal{E}_u \cap \mathcal{E}_v^0(0,k)| \cdot \mu \cdot (\beta-1)/\beta^{k+1} \\
&= -\beta^{\ell(u)-k} \cdot (1/f) \cdot |\mathcal{E}_u \cap \mathcal{E}_v^0(0,k)| \\
&\geq -(1/f) \cdot |\mathcal{E}_u \cap \mathcal{E}_v^0(0,k)| \qquad (24)
\end{aligned}$$

Equation (24) holds since either $|\mathcal{E}_u \cap \mathcal{E}_v^0(0,k)| = 0$, or there is an edge $e \in \mathcal{E}_u \cap \mathcal{E}_v^0(0,k)$. In the former case, equation (24) is trivially true. In the latter case, by Claim 2.19 we have $\ell^0(e) = k$, and since $\ell^0(e) \geq \ell(u)$, we infer that $\ell(u) \leq k$ and $\beta^{\ell(u)-k} \leq 1$.

Claim 2.25. *We have:*

$$\sum_{u \in V \setminus \{v\}} \delta\Psi(u) \geq -\mathcal{D}_v^0(0,k)$$

Proof. For every node $u \in V$ such that $\mathcal{E}_u \cap \mathcal{E}_v^0(0,k) = \emptyset$, we have $\delta\Psi(u) = 0$. Hence, we infer that:

$$\begin{aligned}
\sum_{u \in V \setminus \{v\}} \delta\Psi(u) &= \sum_{u \in V \setminus \{v\}: \mathcal{E}_u \cap \mathcal{E}_v^0(0,k) \neq \emptyset} \delta\Psi(u) && (25)\\
&\geq \sum_{u \in V \setminus \{v\}: \mathcal{E}_u \cap \mathcal{E}_v^0(0,k) \neq \emptyset} -(1/f) \cdot |\mathcal{E}_u \cap \mathcal{E}_v^0(0,k)| && (26)\\
&\geq \sum_{e \in \mathcal{E}_v^0(0,k)} f \cdot (-1/f) && (27)\\
&= -\mathcal{D}_v^0(0,k)
\end{aligned}$$

Equation (26) follows from Claim 2.24. Equation (27) follows from a simple counting argument and the fact that the maximum frequency of an edge is f.

From Claims 2.22, 2.23, 2.25 and equation (23), we derive the following bound.

$$\begin{aligned}
\delta\mathcal{B} &= (1/\epsilon) \cdot \left(\delta\Psi(v) + \sum_{e \in \mathcal{E}_v} \delta\Phi(e) + \sum_{u \in V \setminus \{v\}} \delta\Psi(u) \right)\\
&\geq (1/\epsilon) \cdot \Big(0 + (1+\epsilon) \cdot D_v^0(0,k) - D_v^0(0,k) \Big)\\
&= D_v^0(0,k)
\end{aligned}$$

Thus, Claim 2.21 implies that the net decrease in the potential $\mathcal{B}$ in no less than the increase in COUNT. This proves Lemma 2.18 for Case 1.

Case 2: The level of the node v decreases from k to k − 1.

The claim below bounds the increase in COUNT.

Claim 2.26. *We have* $\text{COUNT}^1 - \text{COUNT}^0 \leq c_v^* \beta^k / \mu$.

Proof. The node v decreases its level from k to $k-1$. We first claim that due to this event, the level of an edge changes only if it belongs to the set $\mathcal{E}_v^0(0,k)$. To see why this is true, note that the levels of the edges *not* incident on v clearly do not change. Hence, it suffices to focus on the edges $e \in \mathcal{E}_v^0(0,L)$. Partition these edges into two subsets: $\mathcal{E}_v^0(0,k)$ and $\mathcal{E}_v^0(k+1,L)$. Every edge $e \in \mathcal{E}_v^0(k+1,L)$ is incident on some node $u \neq v$ that is on or above level $k+1$. Hence, the level of an edge $e \in \mathcal{E}_v^0(k+1,L)$ does not change as the node v moves down from level k to level $k-1$. We conclude that only the edges in $\mathcal{E}_v^0(0,k)$ can potentially change their levels due to this event.

Consider any edge $e \in \mathcal{E}_v^0(0,k)$. By definition, such an edge has $\ell^0(e) \leq k$. Since the edge e is incident on v and $\ell^0(v) = k$, we get: $\ell^0(e) = k$ and $w^0(e) = \mu\beta^{-k}$. To summarize, we get:

$$w^0(e) = \mu\beta^{-k} \text{ for all } e \in \mathcal{E}_v^0(0,k). \tag{28}$$

Since the node v decreases its level from k to $(k-1)$, Step 04 (Fig. 2) ensures that $W_v^0 = W_v^0(k) < c_v^*$. Applying equation (28), we now infer that:

$$c_v^* > W_v^0 \geq \sum_{e \in \mathcal{E}_v^0(0,k)} w^0(e) = \mu\beta^{-k} \cdot D_v^0(0,k).$$

Rearranging the terms in the above inequality, we get $D_v^0(0,k) \leq c_v^* \beta^k / \mu$. Finally, recall that only the edges in $\mathcal{E}_v^0(0,k)$ can potentially change their levels as the node v moves down from level k to level $k-1$. Thus, we get: $\text{COUNT}^1 - \text{COUNT}^0 \leq D_v^0(0,k) \leq c_v^* \beta^k / \mu$. This concludes the proof of the claim.

Claim 2.27. *For all* $u \in V \setminus \{v\}$, *we have* $\delta\Psi(u) \geq 0$.

Proof. Fix any node $u \in V \setminus \{v\}$. If the node u is passive, then we have $\delta\Psi(u) = 0$, and the claim is trivially true. Thus, for the rest of the proof we assume that the node u is active.

If $\mathcal{E}_u \cap \mathcal{E}_v^0(0,k) = \emptyset$, then we have $W_u^0 = W_u^1$, and hence, $\delta\Psi(u) = 0$. Otherwise, as the level of the node v decreases from k to $k-1$, we infer that $w^0(e) \leq w^1(e)$ for all $e \in \mathcal{E}_u \cap \mathcal{E}_v^0(0,k)$, and accordingly we get $W_u^0 \leq W_u^1$. This implies that $\Psi^0(u) \geq \Psi^1(u)$.

We now partition the edge-set $\mathcal{E}_v$ into two subsets, X and Y, according to the level of the other endpoint.

$$X = \left\{ e \in \mathcal{E}_v : \max_{u \in \mathcal{V}_e \setminus \{v\}} \{\ell(u)\} < k \right\} \text{ and } Y = \mathcal{E}_v \setminus X.$$

Claim 2.28. *For every edge $e \in \mathcal{E}_v$, we have:*

$$\delta\Phi(e) = \begin{cases} 0 & \text{if } e \in Y; \\ -(1+\epsilon) & \text{if } e \in X. \end{cases}$$

Proof. Fix any edge $e \in \mathcal{E}_v$. We consider two possible scenarios.

1. We have $e \in Y$. Since e is incident on a vertex $u \neq v$ that is on or above level k, we infer that $\ell^0(e) = \ell^1(e)$, and accordingly, $\Phi^0(e) = \Phi^1(e)$.
2. We have $e \in X$. Since the level of node v decreases from k to $k-1$, we infer that $\ell^0(e) = k$ and $\ell^1(e) = k-1$, and accordingly, $\Phi^0(e) = (1+\epsilon) \cdot (L-k)$ and $\Phi^1(e) = (1+\epsilon) \cdot (L-k+1)$.

This concludes the proof of the Claim.

Next, we partition W_v^0 into two parts: x and y. The first part denotes the contributions towards W_v^0 by the edges $e \in X$, while the second part denotes the contribution towards W_v^0 by the edges $e \in Y$. Note that $X \subseteq \mathcal{E}_v^0(0,k)$, which implies that $x = \sum_{e \in X} w^0(e) = \mu\beta^{-k} \cdot |X|$. Thus, we get the following equations.

$$W_v^0 = x + y < c_v^* \tag{29}$$

$$x = \mu\beta^{-k} \cdot |X| \tag{30}$$

$$y = \sum_{e \in Y} w^0(e) \tag{31}$$

Equation (29) holds due to Step 04 in Fig. 2.

Claim 2.29. *We have $\sum_{e \in \mathcal{E}_v} \delta\Phi(e) = -(1+\epsilon) \cdot x \cdot \beta^k/\mu$.*

Proof. Claim 2.28 implies that $\sum_{e \in \mathcal{E}_v} \delta\Phi(e) = -(1+\epsilon) \cdot |X|$. Applying equation (30), we infer that $|X| = x \cdot \beta^k/\mu$.

Claim 2.30. *We have:*

$$\delta\Psi(v) = (f\alpha c_v^* - x - y) \cdot \frac{\beta^{k+1}}{f\mu(\beta-1)} - \max\left(0, f\alpha c_v^* - \beta x - y\right) \cdot \frac{\beta^k}{f\mu(\beta-1)}.$$

Proof. Equation (29) states that $W_v^0 = x + y < c_v^*$. Since $\ell^0(v) = k$, we get:

$$\Psi^0(v) = (f\alpha c_v^* - x - y) \cdot \frac{\beta^{k+1}}{f\mu(\beta-1)} \tag{32}$$

As the node v decreases its level from k to $k-1$, we have:

$$w^1(e) = \begin{cases} \beta \cdot w^0(e) & \text{if } e \in X; \\ w^0(e) & \text{if } u \in Y \end{cases}$$

Accordingly, we have $W_v^1 = \beta \cdot x + y$, which implies the following equation.

$$\Psi^1(v) = \max(0,\, f\alpha c^*_v - \beta x - y) \cdot \frac{\beta^k}{f\mu(\beta-1)} \tag{33}$$

Since $\delta\Psi(v) = \Psi^0(v) - \Psi^1(v)$, the Claim follows from equations (32) and (33).

We now consider two possible scenarios depending upon the value of $(f\alpha c^*_v - \beta x - y)$. We show that in each case $\delta\mathcal{B} \geq c^*_v\beta^k/\mu$. This, along with Claim 2.26, implies that $\delta\mathcal{B} \geq \textsc{Count}^1 - \textsc{Count}^0$. This proves Lemma 2.18 for Case 2.

1. Suppose that $(f\alpha c^*_v - \beta x - y) < 0$. From Claims 2.27, 2.29, 2.30 and equation (23), we derive:

$$\begin{aligned}
\epsilon \cdot \delta\mathcal{B} &= \sum_{u \in V\setminus\{v\}} \delta\Psi(u) + \sum_{e\in\mathcal{E}_v} \delta\Phi(e) + \delta\Psi(v) \\
&\geq -(1+\epsilon)\cdot x \cdot \frac{\beta^k}{\mu} + (f\alpha c^*_v - x - y)\cdot \frac{\beta^{k+1}}{f\mu(\beta-1)} \\
&\geq -(1+\epsilon)\cdot c^*_v \cdot \frac{\beta^k}{\mu} + (f\alpha - 1)c^*_v \cdot \frac{\beta^{k+1}}{f\mu(\beta-1)} \qquad (34)\\
&= \frac{c^*_v\beta^k}{\mu}\left\{-(1+\epsilon) + (\alpha - 1/f)\cdot\frac{\beta}{(\beta-1)}\right\} \\
&= \frac{c^*_v\beta^k}{\mu}\left\{-(1+\epsilon) + (1+3\epsilon)\cdot\frac{(1+\epsilon)}{\epsilon}\right\} \qquad (35)\\
&\geq \epsilon \cdot c^*_v \cdot \frac{\beta^k}{\mu}
\end{aligned}$$

   Equation (34) follows from equation (29). Equation (35) holds since $\alpha = 1 + 1/f + 3\epsilon$ and $\beta = 1+\epsilon$.
2. Suppose that $(f\alpha c^*_v - \beta x - y) \geq 0$. From Claims 2.27, 2.29, 2.30 and equation (23), we derive:

$$\begin{aligned}
\epsilon \cdot \delta\mathcal{B} &= \sum_{u \in V\setminus\{v\}} \delta\Psi(u) + \sum_{e\in\mathcal{E}_v} \delta\Phi(u,v) + \delta\Psi(v) \\
&\geq -(1+\epsilon)\cdot x \cdot \frac{\beta^k}{\mu} + (f\alpha c^*_v - x - y)\cdot \frac{\beta^{k+1}}{f\mu(\beta-1)} - (f\alpha c^*_v - \beta x - y)\cdot\frac{\beta^k}{f\mu(\beta-1)} \\
&= \frac{\beta^k}{\mu(\beta-1)}\cdot\left\{(f\alpha c^*_v - x - y)\cdot\frac{\beta}{f} - (f\alpha c^*_v - \beta x - y)\cdot\frac{1}{f} - (1+\epsilon)\cdot x\cdot(\beta-1)\right\} \\
&= \frac{\beta^k}{\mu(\beta-1)}\cdot\left\{\alpha c^*_v\beta - \alpha c^*_v - \frac{(\beta x + \beta y - \beta x - y)}{f} - (1+\epsilon)\cdot x\cdot(\beta-1)\right\} \\
&= \frac{\beta^k}{\mu(\beta-1)}\cdot\left\{\alpha c^*_v\cdot(\beta-1) - \frac{y(\beta-1)}{f} - (1+\epsilon)\cdot x\cdot(\beta-1)\right\} \\
&= \frac{\beta^k}{\mu}\cdot\left\{\alpha c^*_v - \frac{y}{f} - (1+\epsilon)\cdot x\right\} \\
&\geq \frac{\beta^k}{\mu}\cdot\left\{\alpha c^*_v - \beta(y+x)\right\} \qquad (36)\\
&\geq \frac{\beta^k}{\mu}\cdot(\alpha-\beta)\cdot c^*_v \qquad (37)\\
&\geq \epsilon\cdot c^*_v\cdot\frac{\beta^k}{\mu} \qquad (38)
\end{aligned}$$

   Equation (36) holds since $\beta = 1+\epsilon$ and $f \geq 1$. Equation (37) follows from Equation (29). Equation (38) holds since $\alpha = 1 + 1/f + 3\epsilon$ and $\beta = 1+\epsilon$.

3. Maintaining a set-cover in a dynamic setting

We first show the link between the fractional hypergraph b-matching and set-cover.

Lemma 3.1. *The dual LP (4) is an LP-relaxation of the set-cover problem (Definition 1.1).*

Proof. 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 cost $c_{v(S)} = c_S$. Ensure that an element u belongs to a set S iff $e(u) \in \mathcal{E}_{v(S)}$. Finally, set $\mu = \max_{v \in V} c_v + 1$.

On this instance of the hypergraph b-matching problem, we claim that an optimal solution to the dual LP (4) will set $z(e) = 0$ for every edge $e \in E$. To see why the claim holds, consider any feasible solution to the dual LP (4) where $z(e') = q > 0$ (say) for some edge $e' \in E$. Then we can identify any node $v' \in \mathcal{V}_e$, and get a new feasible dual solution by setting $y(v') := y(v') + q$ and $z(e') := 0$. Since $\mu > c_{v'}$, the objective of the dual solution will decrease by $(\mu \cdot q - c_{v'} \cdot q) > 0$ due to this transformation. It follows that the initial dual solution was not optimal.

Since an optimal solution to the dual LP (4) will set $z(e) = 0$ for every edge $e \in E$, 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.

We now present the main result of this section.

Theorem 3.2. *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. We map the set cover instance to a fractional hypergraph b-matching instance as in the proof of Lemma 3.1. By Theorem 2.3, 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.

4. Maintaining a *b*-matching in a dynamic setting

We will present a dynamic algorithm for maintaining an $O(1)$-approximation to the maximum b-matching (see Definitions 1.3, 1.4). Our main result is summarized in Theorem 4.8. We use the following approach. First, we note that the fractional b-matching problem is a special case of the fractional hypergraph b-matching problem (see Definition 1.5) with $f = 2$ (for each edge is incident upon exactly two nodes). Hence, by Theorems 2.2 and 2.3, we can maintain a $O(f^2) = O(1)$ approximate "fractional" solution to the maximum b-matching problem in $O(f \log(m+n)) = O(\log n)$ amortized update time. Next, we perform randomized rounding on this fractional solution in the dynamic setting, whereby we select each edge in the solution with some probability that is determined by its fractional value. This leads to Theorem 4.8.

Notations. Let $G = (V, E)$ be the input graph to the b-matching problem. 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 $w : E' \to \mathbf{R}^+$ that assigns a weight $w(e)$ to every edge $e \in E'$. 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 $w(E') = \sum_{e \in E'} w(e)$.

Recall that in the b-matching problem, we are given an "input graph" $G = (V, E)$ with $|V| = n$ nodes, where each node $v \in V$ has a "capacity" $c_v \in \{1, \ldots, n\}$. We want to select a subset $E' \subseteq E$ of edges of maximum size such that each node v has at most c_v edges incident to it in E'. We will also be interested in "fractional" b-matchings. In the fractional b-matching problem, we want to assign a weight $w(e) \in [0, 1]$ to every edge $e \in E$ such that $\sum_{u \in \mathcal{N}(v,E)} w(u, v) \leq c_v$ for every node $v \in V$, and the sum of the edge-weights $w(E)$ is maximized. In the dynamic version of these problems, the node-set V remains fixed, and at each time-step the edge-set E gets updated due to an edge insertion or deletion. We now show how to efficiently maintain an $O(1)$-approximate fractional b-matching in the dynamic setting.

Theorem 4.1. *Fix a constant $\epsilon \in (0, 1/4)$, and let $\lambda = 4$, and $\gamma = 1 + 4\epsilon$. 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:*

$$W_v \leq c_v/\gamma \text{ for all nodes } v \in V. \tag{39}$$

$$w(u, v) = 1 \text{ for each edge } (u, v) \in E \text{ with } W_u, W_v < c_v/\lambda. \tag{40}$$

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

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

We scale down the capacity of each node $v \in V$ by a factor of γ, by defining $\tilde{c}_v = c_v/\gamma$ for all $v \in V$. Next, we apply Theorem 2.3 on the input simple 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 2.1). 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 2.2, 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 γ. Hence, the size of the optimal b-matching in G is at most $9\gamma = O(1)$ times the sum $\sum_{e \in E} w(e)$. This concludes the proof.

For the rest of this section, we set $\lambda = 4$, $\gamma = 1+4\epsilon$ and $\epsilon \in (0, 1/4)$, as defined in the statement of Theorem 4.1. We will show how to dynamically convert the fractional b-matching $\{w(e)\}$ from Theorem 4.1 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 a few notations.

4.1. Notations

Say that a node $v \in V$ is "nearly-tight" if $W_v \geq c_v/\lambda$ and "slack" otherwise. Let T be the set of all nearly-tight nodes. We also partition of 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 $G_S = (V, E_S)$. Let $E^* = \{e \in E : w(e) = 1\}$ be the subset of edges with weight one in the fractional b-matching $\{w(e)\}$.

Observation 4.2. We have $\mathcal{N}(v, E) = \mathcal{N}(v, E_B)$ for all $v \in B$, and $\mathcal{N}(u, E) = \mathcal{N}(u, E_S)$ for all $u \in S$.

4.2. An overview of our algorithm

Our algorithm maintains the following structures.

- A fractional b-matching as per Theorem 4.1.
- A random subset of edges $H_B \subseteq E_B$ and a weight function $w^B : H_B \to [0, 1]$ in the subgraph $G_B(H_B) = (V, H_B)$. The algorithm ensures that they satisfy Property 4.3.
- A random subset of edges $H_S \subseteq E_S$ and a weight function $w^S : H_S \to [0, 1]$ in the subgraph $G_S(H_S) = (V, H_S)$. The algorithm ensures that they satisfy Property 4.4.
- A maximal b-matching $M_S \subseteq H_S$ in the subgraph $G_S(H_S) = (V, H_S)$, that is, for every edge $(u, v) \in H_S \setminus M_S$, there is a node $q \in \{u, v\}$ such that $\deg(q, M_S) = c_q$.
- The set of edges $E^* = \{e \in E : w(e) = 1\}$.

Property 4.3. Let $Z_B(e) \in \{0, 1\}$ be an indicator random variable that is set to one if $e \in H_B$ and zero otherwise. Then the following conditions are satisfied.

$$\text{We always have } \deg(v, H_B) \leq c_v \text{ for every small node } v \in S. \tag{41}$$

$$\mathbf{E}[Z_B(e)] = \Pr[e \in H_B] = w(e) \text{ for every edge } e \in E_B. \tag{42}$$

$$\forall v \in B, \text{ the events } \{Z_B(e) = 1\}, u \in \mathcal{N}(v, E_B), \text{ are mutually independent.} \tag{43}$$

$$\text{For each edge } e \in H_B, \text{ we have } w^B(e) = 1 \tag{44}$$

Property 4.4. Let $Z_S(e) \in \{0, 1\}$ be an indicator random variable that is set to one if $e \in H_S$ and zero otherwise. Then the following conditions are satisfied.

$$\mathbf{E}[Z_S(e)] = \Pr[e \in H_S] = p_e = \min(1, w(e) \cdot (c\lambda \log n/\epsilon)) \ \forall e \in E_S. \tag{45}$$

$$\text{The events } \{Z_S(e) = 1\}, e \in E_S, \text{ are mutually independent.} \tag{46}$$

$$\text{For each edge } e \in H_S, \text{ we have } w^S(e) = \begin{cases} w(e) & \text{if } p_e \geq 1; \\ \epsilon/(c\lambda \log n) & \text{if } p_e < 1. \end{cases} \tag{47}$$

In Section 4.3, we describe some important properties satisfied by the weight functions w^B and w^S. We use these properties while proving the three main lemmas stated below. The proofs of Lemmas 4.5, 4.6 and 4.7 appear in Sections 4.4, 4.5 and 4.6 respectively.

Lemma 4.5. *With high probability, we can maintain the random sets of edges H_B and H_S, a maximal b-matching M_S in $G_S(H_S) = (V, H_S)$, and the set of edges E^* in $O(\log^3 n)$-amortized update time.*

Lemma 4.6. *With high probability, each of the edge-sets H_B, M_S and E^* is a valid b-matching in G.*

Lemma 4.7. *We have $w(E) \leq O(1) \cdot \max(|E^*|, |H_B|, |M_S|)$ with high probability.*

The three lemmas stated above implies our main result, which is summarized in Theorem 4.8.

Theorem 4.8. *With high probability, we can maintain a $O(1)$-approximate b-matching in the input graph $G = (V, E)$ in $O(\log^3 n)$ amortized update time.*

Proof. We maintain a fractional b-matching $w : E \to [0, 1]$ as per Theorem 4.1. Next, we maintain the random sets of edges H_B and H_S, a maximal b-matching M_S in $G_S(H_S) = (V, H_S)$, and the set of edges $E^* = \{e \in E : w(e) = 1\}$ as per Lemma 4.5. This requires $O(\log^3 n)$ amortized update time with high probability. Lemmas 4.6 and 4.7 imply that at least one of the subsets $H_B, M_S, E^* \subseteq E$ is a $O(1)$-approximate maximum b-matching in $G = (V, E)$. This concludes the proof of the theorem.

4.3. Some useful properties of the weight functions w^B and w^S

For ease of exposition, we defer the proofs of Lemmas 4.9 and 4.10 to Sections 4.7 and 4.8.

Lemma 4.9. *Let $W_v^B = \sum_{u \in \mathcal{N}(v, H_B)} w^B(u, v)$ denote the total weight received by a node $v \in V$ from its incident edges in H_B under the weight function w^B. The following conditions hold with high probability.*

- *For every node $v \in V$, we have $W_v^B \leq c_v$.*
- *For every node $v \in B \cap T$, we have $W_v^B \geq (1 - \epsilon) \cdot (c_v/\lambda)$. The set T is defined as in Section 4.1.*

Lemma 4.10. *Let $W_v^S = \sum_{u \in \mathcal{N}(v, H_S)} w^S(u, v)$ denote the total weight received by a node $v \in V$ from its incident edges in H_S under the weight function w^S. The following conditions hold with high probability.*

- *For each node $v \in V$, we have $W_v^S \leq c_v$.*
- *For each node $v \in S$, we have $deg(v, H_S) = O(\log^2 n)$.*
- *For each node $v \in S \cap T$, we have $W_v^S \geq (1 - \epsilon) \cdot (c_v/\lambda)$. The set T is defined as in Section 4.1.*

4.4. Proof of Lemma 4.5

We maintain the fractional b-matching $\{w(e)\}$ as per Theorem 4.1. 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 $\{w(e)\}$. Thus, we can easily maintain the edge-set $E^* = \{e \in E : w(e) = 1\}$ in $O(\log n)$ amortized update time. Specifically, we store the edge-set E^* as a doubly linked list. For every edge $(u, v) \in E^*$, we maintain a pointer that points to the position of (u, v) in this linked list. For every edge $(u, v) \in E \setminus E^*$, the corresponding pointer is set to NULL. An edge (u, v) is inserted into/deleted from the set E^* only when its weight $w(e)$ is changed. Thus, maintaining the linked list for E^* does not incur any additional overhead in the update time.

Next, we show to maintain the edge-set H_S 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 also $O(\log n)$. Similar to the case of the edge-set E^*, we store the edge-set H_S as a doubly linked list.

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 4.10). Exploiting this fact, for each node $v \in B$, we can maintain the set of its free (unmatched) neighbors $F_v(S) = \{u \in \mathcal{N}(v, H_S) : u \text{ is unmatched in } M_S\}$ in $O(\log^2 n)$ amortized time per update in H_S, with high probability. This is done as follows. Since $v \in B$, the onus of maintaining the set $F_v(S)$ falls squarely upon the nodes in $\mathcal{N}(v, H_S) \subseteq S$. Specifically, each small node $u \in S$ maintains a "status-bit" indicating if it is free or not. Whenever a matched small node u changes its status-bit, it communicates this information to its neighbors in $\mathcal{N}(u, H_S) \cap B$ in $O(\deg(u, H_S)) = O(\log^2 n)$ time. Using the lists $\{F_v(S)\}, v \in B$, and the status-bits of the small nodes, 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, with high probability. 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, with high probability, for maintaining the matching M_S.

Finally, we show how to maintain the set H_B. The edges $(u,v) \in E_B$ with both endpoints $u, v \in B$ are sampled independently with probability $w(u,v)$. 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 $\deg(v, H_B) \le c_v$. The random choices made by the different small nodes are mutually independent, which implies equation (43). 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 $\mathcal{N}(v, H_B) = \{v_i \in B : A_{i-1}(v) \le k + \eta_v < A_i(v)$ 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 $v_i \ne \mathcal{N}(v, E_B)$. Also note that $\deg(v, H_B) \le \lceil \sum_{v_i \in \mathcal{N}(v, E_B)} w(v, v_i) \rceil \le \lceil W_v \rceil \le \lceil c_v/(\gamma) \rceil \le c_v$. Hence, equations (41), (42) are satisfied. We maintain the sums $\{A_i(v)\}, i$, and the set $\mathcal{N}(v, H_B)$ using a balanced binary tree data structure, as described below.

We store the ordered sequence of $|B|$ numbers $a_1(v), \ldots, a_{|B|}(v)$ in the leaves of a balanced binary tree from left to right. Let x_i denote the leaf node that stores the value $a_i(v)$. Further, at each internal node x of the balanced binary tree, we store the sum $S_x = \sum_{i: x_i \in T(x)} a_i(v)$, where $T(x)$ denotes the set of nodes in the subtree rooted at x. This data structure can support the following operations.

INCREMENT(i, δ): This asks us to set $a_i(v) \leftarrow a_i(v) + \delta$, where δ is any real number. To perform this update, we first change the value stored at the leaf node x_i. Then starting from the node x_i, we traverse up to the root of the tree. At each internal node x in this path from x_i to the root, we set $S_x \leftarrow S_x + \delta$. The S_x values at every other internal node remains unchanged. Since the tree has depth $O(\log n)$, the total time required to update the data structure is also $O(\log n)$.

RETURN-INDEX(y): Given a number $0 \le y < c_v$, this asks us to return an index i (if it exists) such that $A_{i-1}(v) \le y < A_i(v)$. We can answer this query in $O(\log n)$ time by doing binary search. Specifically, we perform the following operations. We initialize a counter $C \leftarrow 0$ and start our binary search at the root of the tree. At an intermediate stage of the binary search, we are at some internal node x and we know that $y < C + S_x$. Let $x(l)$ and $x(r)$ respectively be the left and right child of x. Note that $S_x = S_{x(l)} + S_{x(r)}$. If $y < C + S_{x(l)}$, then we move to the node $x(l)$. Otherwise, we set $C \leftarrow C + S_{x(l)}$ and move to the node $x(r)$. We continue this process until we reach a leaf node, which gives us the required answer. The total time taken by the procedure is $O(\log n)$.

We use the above data structure to maintain the sets $\mathcal{N}(v, H_B), v \in S$. Whenever the weight of an edge (u,v), $v \in S$, changes, we can update the set $\mathcal{N}(v, H_B)$ by making one call to the INCREMENT(i, δ), and c_v calls to RETURN-INDEX(y), one for each $y = k + \eta_v$, where $k < c_v$ is a nonnegative integer. Since $c_v = O(\log n)$, the total time required is $O(\log^2 n)$ per change in the edge-weights $\{w(e)\}$.

Since each edge insertion/deletion in G, on average, leads to $O(\log n)$ changes in the edge-weights $\{w(e)\}$, the overall amortized update time for maintaining the edge-set H_B is $O(\log^3 n)$.

Similar to the edge-sets E^* and H_S, we store the edge-set H_B as a doubly linked list. Each edge $(u,v) \in H_B$ maintains a pointer to its position in this list. Each edge $(u,v) \in E \setminus H_B$ sets the corresponding pointer to NULL. It is easy to check that this does not incur any additional overhead in the update time. This concludes the proof of the lemma.

4.5. Proof of Lemma 4.6

Since $w^B(e) = 1$ for every edge $e \in H_B$ (see Definition 4.3), Lemma 4.9 implies that the edge-set H_B is a b-matching in G with high probability. Next, by definition, the edge-set M_S is a b-matching in $G_S(H_S) = (V, H_S)$. Since $H_S \subseteq E$, the edge-set M_S is also a b-matching in G. Finally, since $w : E \to [0,1]$ is a fractional b-matching in G, the set of edges E^* is also a b-matching in G.

4.6. Proof of Lemma 4.7

Consider any edge $(u,v) \in E$. If $u \notin T$ and $v \notin T$, then by equation (40), we must have $(u,v) \in E^*$. In contrast, if there is some node $x \in \{u,v\}$ such that $x \in T$, then we must have either $x \in B \cap T$ or $x \in S \cap T$. In other words, every edge (u,v) satisfies this property: Either $(u,v) \in E^*$, or it is incident upon some node in $B \cap T$, or it is incident upon some node $S \cap T$. Thus, each edge $e \in E$ contributes at least $w(e)$ to the sum $w(E^*) + \sum_{v \in B \cap T} W_v + \sum_{v \in S \cap T} W_v$. Hence, we get:

$$w(E^*) + \sum_{v \in B \cap T} W_v + \sum_{v \in S \cap T} W_v \ge w(E) \tag{48}$$

Note that $w(E^*) = |E^*|$. We now consider three possible cases, based on equation (48).

Case 1. $w(E^*) \ge (1/3) \cdot w(E)$. In this case, clearly $w(E) \le 3 \cdot \max(|E^*|, |H_B|, |M_S|)$.

Case 2. $\sum_{v \in B \cap T} W_v \ge (1/3) \cdot w(E)$. In this case, we condition on the event under which Lemma 4.9 holds. Thus, we get:

$$w(E) \le \sum_{v \in B \cap T} 3 \cdot W_v \le \sum_{v \in B \cap T} 3 \cdot c_v \le \sum_{v \in B \cap T} (3\lambda/(1-\epsilon)) \cdot W_v^B$$

$$\leq (3\lambda/(1-\epsilon)) \cdot \sum_{e \in H_B} 2 \cdot w^B(e) = (6\lambda/(1-\epsilon)) \cdot |H_B|$$

Case 3. $\sum_{v \in S \cap T} W_v \geq (1/3) \cdot w(E)$. In this case, we condition on the event under which Lemma 4.10 holds. Thus, we get:

$$\begin{aligned} w(E) &\leq \sum_{v \in S \cap T} 3 \cdot W_v \leq \sum_{v \in S \cap T} 3 \cdot c_v \leq \sum_{v \in S \cap T} (3\lambda/(1-\epsilon)) \cdot W_v^S \\ &\leq (3\lambda/(1-\epsilon)) \cdot \sum_{e \in H_S} 2 \cdot w^S(e) = (6\lambda/(1-\epsilon)) \cdot \sum_{e \in H_S} w^S(e) \\ &\leq (12\lambda/(1-\epsilon)) \cdot |M_S|. \end{aligned}$$

The last inequality holds since M_S is a maximal b-matching in $G_S(H_S) = (V, H_S)$, and since every maximal b-matching is a 2-approximation to the maximum fractional b-matching (this follows from LP duality). Accordingly, we have $\sum_{e \in H_S} w^S(e) \leq 2 \cdot |M_S|$.

4.7. Proof of Lemma 4.9

Lemma 4.9 follows from Lemmas 4.11 and 4.12.

Lemma 4.11. *With high probability, we have $W_v^B \geq (1-\epsilon) \cdot (c_v/\lambda)$ for every node $v \in B \cap T$.*

Proof. Fix any node $v \in B \cap T$. Note that $\mathcal{N}(v, E_B) = \mathcal{N}(v, E)$, $W_v \geq c_v/\lambda$, and $c_v \geq c\lambda \log n/\epsilon$. Linearity of expectation, in conjunction with equations (42), (44) and Observation 4.2 imply that we have $\mathbf{E}[W_v^B] = \sum_{u \in \mathcal{N}(v,E_B)} \mathbf{E}[Z_B(u,v)] = \sum_{u \in \mathcal{N}(v,E_B)} w(u,v) = \sum_{u \in \mathcal{N}(v,E)} w(u,v) = W_v \geq c_v/\lambda \geq c \log n/\epsilon$. Thus, applying Chernoff bound, we infer that $\mathbf{E}[W_v^B] \geq (1-\epsilon) \cdot (c_v/\lambda)$ with high probability. The lemma follows if we take a union bound over all nodes $v \in B \cap T$.

Lemma 4.12. *With high probability, we have $W_v^B \leq c_v$ for every node $v \in V$.*

Proof. Consider any node $v \in V$. If $v \in S$, then we have $W_v^B \leq c_v$ with probability one (see equations (41), (44)). For the rest of the proof, suppose that $v \in B$. Applying an argument similar to the one used in the proof of Lemma 4.11, we infer that $\mathbf{E}[W_v^B] = W_v \leq c_v/\gamma$. The last inequality holds due to equation (39). Since $\gamma > (1+\epsilon)$ and $c_v \geq c\lambda \log n/\epsilon$, applying Chernoff bound we derive that $W_v^B \leq c_v$ with high probability. Thus, for each node $v \in V$, we have $W_v^B \leq c_v$ with high probability. The lemma now follows if we take a union bound over all nodes $v \in B$.

4.8. Proof of Lemma 4.10

High Level Overview. In order to highlight the main idea, 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 (39), (45), (47) 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 (46) 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.

4.8.1. Full details

For every node $v \in V$, we partition the set $\mathcal{N}(v, E_S)$ into two subsets – $X(v)$, $Y(v)$ – as defined below.

$$X(v) = \{u \in \mathcal{N}(v, E_S) : p_{(u,v)} = 1\} \tag{49}$$

$$Y(v) = \{u \in \mathcal{N}(v, E_S) : p_{(u,v)} < 1\} \tag{50}$$

Next, for every node $v \in V$, we define:

$$\delta_X(v) = \sum_{u \in X(v)} w(u, v) \tag{51}$$

$$\delta_Y(v) = \sum_{u \in Y(v)} w(u, v) \tag{52}$$

Since $\mathcal{N}(v, E_S) \subseteq \mathcal{N}(v, E)$ for every node $v \in V$, by equation (39) we have:

$$\sum_{u \in \mathcal{N}(v, E_S)} w(u, v) = \delta_X(v) + \delta_Y(v) \leq c_v/\gamma \tag{53}$$

Since $X(v) \subseteq \mathcal{N}(v, E_S)$ and $w^S(u, v) = w(u, v)$ for every node $u \in X(v)$, we get:

$$\sum_{u \in X(v)} w^S(u, v) = \delta_X(v). \tag{54}$$

Lemma 4.13. *For every node $v \in V$, if $\delta_Y(v) \leq \epsilon/\lambda$, then with high probability, we have:*

$$|Y(v) \cap \mathcal{N}(v, H_S)| \leq (1+\epsilon) \cdot c \log n; \text{ and}$$
$$\sum_{u \in Y(v) \cap \mathcal{N}(v, H_S)} w^S(u, v) \leq 2\epsilon/\lambda.$$

Proof. Recall that for every node $u \in Y(v)$, we have defined $Z_S(u, v) \in \{0, 1\}$ to be an indicator random variable that is set to one if $(u, v) \in H_S$ and zero otherwise. Clearly, we have $\mathbf{E}[Z_S(u, v)] = (c\lambda \log n/\epsilon) \cdot w(u, v)$ for all $u \in Y(v)$. Applying linearity of expectation, we get:

$$\mathbf{E}[|Y(v) \cap \mathcal{N}(v, H_S)|] = E\left[\sum_{u \in Y(v)} Z_S(u, v)\right] = (c\lambda \log n/\epsilon) \cdot \sum_{u \in Y(v)} w(u, v)$$
$$= (c\lambda \log n/\epsilon) \cdot \delta_Y(v) \leq c \log n.$$

Since $\mathbf{E}[|Y(v) \cap \mathcal{N}(v, H_S)|] \leq c \log n$, applying Chernoff bound we infer that $|Y(v) \cap \mathcal{N}(v, H_S)| \leq (1+\epsilon)c \log n$ with high probability.

Finally, note that each node $u \in Y(v) \cap \mathcal{N}(v, H_S)$ has $w^S(u, v) = \epsilon/(c\lambda \log n)$. This implies that

$$\sum_{u \in Y(v) \cap \mathcal{N}(v, H_S)} w^S(u, v) = \epsilon/(c\lambda \log n) \cdot |Y(v) \cap H_S|.$$

Since $|Y(v) \cap H_S| \leq (1+\epsilon)c \log n$ with high probability, we get: $\sum_{u \in Y(v) \cap \mathcal{N}(v, H_S)} w^S(u, v) \leq (1+\epsilon)\epsilon/\lambda \leq 2\epsilon/\lambda$ with high probability. This concludes the proof of the lemma.

Lemma 4.14. *For every node $v \in V$, if $\delta_Y(v) \geq \epsilon/\lambda$, then with high probability, we have:*

$$(c\lambda \log n/\epsilon) \cdot \frac{\delta_Y(v)}{(1+\epsilon)} \leq |Y(v) \cap \mathcal{N}(v, E_S)| \leq (c\lambda \log n/\epsilon) \cdot (1+\epsilon)\delta_Y(v); \text{ and}$$
$$\frac{\delta_Y(v)}{(1+\epsilon)} \leq \sum_{u \in Y(v) \cap \mathcal{N}(v, H_S)} w^S(u, v) \leq (1+\epsilon)\delta_Y(v).$$

Proof. Let $\mu = E[|Y(v) \cap \mathcal{N}(v, H_S)|]$. Applying an argument as in the proof of Lemma 4.13, we get: $\mu = (c\lambda \log n/\epsilon) \cdot \delta_Y(v) \geq c \log n$. Hence, applying Chernoff bound, we infer that $\mu/(1+\epsilon) \leq |Y(v) \cap \mathcal{N}(v, H_S)| \leq (1+\epsilon)\mu$ with high probability. This proves the first part of the lemma.

To prove the second part of the lemma, we simply note that, as in the proof of Lemma 4.13, we have

$$\sum_{u \in Y(v) \cap \mathcal{N}(v, H_S)} w^S(u, v) = (\epsilon/(c\lambda \log n)) \cdot |Y(v) \cap \mathcal{N}(v, H_S)|.$$

Lemma 4.15. *For every node $v \in V$, we have $deg(v, H_S) = O((\log n/\epsilon) \cdot c_v)$ with high probability.*

Proof. Fix any node $v \in V$. Note that $X(v) \subseteq \mathcal{N}(v, H_S)$ and $w(u, v) = w^S(u, v) \geq \epsilon/(c\lambda \log n)$ for every node $u \in X(v)$. By equation (54), we have $\sum_{u \in X(v)} w^S(u, v) = \delta_X(v)$ for every node $v \in V$. Thus, we get:

$$|X(v)| \leq (c\lambda \log n/\epsilon) \cdot \delta_X(v) = O((\log n/\epsilon) \cdot \delta_X(v)) \tag{55}$$

Lemmas 4.13 and 4.14 imply that with high probability, we have:

$$|Y(v) \cap H_S| \leq \max(c \log n, (c\lambda \log n/\epsilon)(1+\epsilon)\delta_Y(v))$$
$$= O((\log n/\epsilon) \cdot \delta_Y(v)) \tag{56}$$

Since $\deg(v, H_S) = |X(v)| + |Y(v) \cap \mathcal{N}(v, H_S)|$, the lemma follows if we add equations (55) and (56), and recall that $\delta_X(v) + \delta_Y(v) \leq c_v$ (see equation (53)).

Lemma 4.16. *For every node $v \in V$, we have $W_v^S \leq c_v$ with high probability.*

Proof. Lemmas 4.13 and 4.14 imply that with high probability, we have:

$$\sum_{u \in Y(v) \cap \mathcal{N}(v, H_S)} w^S(u, v) \leq \max(2\epsilon/\lambda, (1+\epsilon)\delta_Y(v)) \tag{57}$$

Since the node-set $\mathcal{N}(v, H_S)$ is partitioned into $X(v)$ and $Y(v) \cap \mathcal{N}(v, H_S)$, we get:

$$\begin{aligned} W_v^S &= \sum_{u \in X(v)} w^S(u, v) + \sum_{u \in Y(v) \cap \mathcal{N}(v, H_S)} w^S(u, v) \\ &\leq (1+\epsilon) \cdot \delta_X(v) + \max(2\epsilon/\lambda, (1+\epsilon)\delta_Y(v)) && (58) \\ &\leq (1+\epsilon) \cdot (\delta_X(v) + \delta_Y(v)) + 2\epsilon/\lambda \\ &\leq (1+\epsilon) \cdot (c_v/\gamma) + (2\epsilon/\lambda) \cdot c_v && (59) \\ &\leq (1+\epsilon) \cdot (c_v/\gamma) + 2\epsilon \cdot (c_v/\gamma) && (60) \\ &\leq c_v && (61) \end{aligned}$$

Equation (58) follows from equations (54) and (57), and it holds with high probability. Equation (59) follows from equation (53) and the fact that $c_v \geq 1$. Equation (60) holds since $\gamma < \lambda$ (see Theorem 4.1). Equation (61) holds since $\gamma > 1 + 3\epsilon$ (see Theorem 4.1).

Lemma 4.17. *For every node $v \in S \cap T$, we have $W_v^S \geq (1-\epsilon) \cdot (c_v/\lambda)$.*

Proof. Fix any node $v \in S \cap T$. Since $v \in S$, we have $\mathcal{N}(v, E) = \mathcal{N}(v, E_S)$. Since $v \in T$, we have $W_v = \sum_{u \in \mathcal{N}(v, E_S)} w(u, v) \geq c_v/\lambda$. Since $\sum_{u \in \mathcal{N}(v, E_S)} w(u, v) = \delta_X(v) + \delta_Y(v)$, we get:

$$\delta_X(v) + \delta_Y(v) \geq c_v/\lambda \tag{62}$$

We also recall that by equation (54) we have:

$$\sum_{u \in X(v)} w^S(u, v) = \delta_X(v) \tag{63}$$

We now consider two possible cases, based on the value of $\delta_Y(v)$.

Case 1. We have $\delta_Y(v) \leq \epsilon/\lambda$. Since $c_v \geq 1$, in this case, we have $\delta_X(v) \geq c_v/\lambda - \delta_Y(v) \geq c_v(1-\epsilon)/\lambda$. By equation (63), we infer that $W_v^S \geq \sum_{u \in X(v)} w^S(u, v) = \delta_X(v) \geq c_v(1-\epsilon)/\lambda$. This concludes the proof of the lemma for Case 1.

Case 2. We have $\delta_Y(v) > \epsilon/\lambda$. In this case, Lemma 4.14 implies that with high probability we have:

$$\sum_{u \in Y(v) \cap \mathcal{N}(v, H_S)} w^S(u, v) \geq \delta_Y(v)/(1+\epsilon).$$

Since the node-set $\mathcal{N}(v, H_S)$ is partitioned into $X(v)$ and $Y(v) \cap \mathcal{N}(v, H_S)$, we get:

$$\begin{aligned} W^S(u, v) &= \sum_{u \in X(v)} w^S(u, v) + \sum_{u \in Y(v) \cap \mathcal{N}(v, H_S)} w^S(u, v) \geq \delta_X(v) + \delta_Y(v)/(1+\epsilon) \\ &\geq (\delta_X(v) + \delta_Y(v))/(1+\epsilon) \geq (c_v/\lambda) \cdot (1/(1+\epsilon)) \geq (1-\epsilon) \cdot (c_v/\lambda) \end{aligned}$$

This concludes the proof of the lemma for Case 2.

Lemma 4.10 follows from Lemmas 4.15, 4.16, 4.17, and the fact that $c_v = O(\log n)$ for all $v \in S$.

5. Conclusion and open problems

In this paper, we introduced a dynamic version of the primal-dual method. Applying this framework, we obtained the first nontrivial dynamic algorithms for the set cover and b-matching problems. Specifically, we presented a dynamic algorithm for set cover that maintains a $O(f^2)$-approximation in $O(f \cdot \log(m+n))$ update time, where f is the maximum frequency of an element, m is the number of sets and n is the number of elements. On the other hand, for the b-matching problem, we presented a dynamic algorithm that maintains a $O(1)$-approximation in $O(\log^3 n)$ update time. Our work leaves several interesting open questions. We conclude the paper by stating a couple of such problems.

- Recall that in the static setting the set cover problem admits $O(\min(f, \log n))$-approximation in $O(f \cdot (m+n))$-time. Can we match this approximation guarantee in the dynamic setting in $O(f \cdot \operatorname{poly}\log(m+n))$ update time? As a first step, it will be interesting to design a dynamic algorithm for fractional hypergraph b-matching that maintains a $O(f)$-approximation and has an update time of $O(f \cdot \operatorname{poly}\log(m+n))$.
- Are there other well known problems (such as facility location, Steiner tree etc.) that can be solved in the dynamic setting using the primal-dual framework?

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 340506. Monika Henzinger was also supported by the EU 7th Framework Programme under Grant Agreement 317532. Giuseppe Italiano was partially supported by MIUR (the Italian Ministry of Education, University and Research) under Project AMANDA.

References

[1] H.W. Kuhn, The Hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1955) 83–97.
[2] G.B. Dantzig, L.R. Ford, D.R. Fulkerson, A primal-dual algorithm for linear programs, in: H.W. Kuhn, A.W. Tucker (Eds.), Linear Inequalities and Related Systems, Princeton University Press, Princeton, NJ, 1956, pp. 171–181.
[3] R. Bar-Yehuda, S. Even, A linear time approximation algorithm for the weighted vertex cover problem, J. Algorithms 2 (1981) 198–203.
[4] M. Goemans, D.P. Williamson, A general approximation technique for constrained forest problems, SIAM J. Comput. 24 (1992) 296–317.
[5] M.X. Goemans, D.P. Williamson, The primal-dual method for approximation algorithms and its application to network design problems, in: D.S. Hochbaum (Ed.), Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, 1997, pp. 144–191.
[6] N. Buchbinder, J. Naor, The design of competitive online algorithms via a primal-dual approach, Found. Trends Theor. Comput. Sci. 3 (2–3) (2009) 93–263, https://doi.org/10.1561/0400000024.
[7] S. Bhattacharya, M. Henzinger, 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), 2015, pp. 785–804.
[8] V.V. Vazirani, Approximation Algorithms, Springer-Verlag, New York, NY, USA, 2001.
[9] S. Korman, On the Use of Randomization in the Online Set Cover Problem, Weizmann Institute of Science, 2004.
[10] D. Eppstein, Z. Galil, G.F. Italiano, Dynamic graph algorithms, in: M.J. Atallah, M. Blanton (Eds.), 2nd edition, Algorithms and Theory of Computation Handbook, vol. 1, CRC Press, 2009, pp. 9.1–9.28.
[11] S. Baswana, M. Gupta, S. Sen, Fully dynamic maximal matching in $O(\log n)$ update time, in: 52nd IEEE Symposium on Foundations of Computer Science, 2011, pp. 383–392.
[12] M. Gupta, R. Peng, Fully dynamic $(1+\epsilon)$-approximate matchings, in: 54th IEEE Symposium on Foundations of Computer Science, 2013, pp. 548–557.
[13] O. Neiman, S. Solomon, Simple deterministic algorithms for fully dynamic maximal matching, in: 45th ACM Symposium on Theory of Computing, 2013, pp. 745–754.
[14] K. Onak, R. Rubinfeld, Maintaining a large matching and a small vertex cover, in: 42nd ACM Symposium on Theory of Computing, 2010, pp. 457–464.
[15] D.S. Johnson, Approximation algorithms for combinatorial problems, J. Comput. Syst. Sci. 9 (1974) 256–278.
[16] U. Feige, A threshold of ln n for approximating set cover, J. ACM 45 (1998) 634–652.
[17] S. Khot, O. Regev, Vertex cover might be hard to approximate to within $2-\epsilon$, J. Comput. Syst. Sci. 74 (3) (2008) 335–349.
[18] H.N. Gabow, An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems, in: Proceedings of the 15th Annual ACM Symposium on Theory of Computing, Boston, Massachusetts, USA, 25–27 April, 1983, 1983, pp. 448–456.
[19] K.J. Ahn, S. Guha, Near linear time approximation schemes for uncapacitated and capacitated b-matching problems in nonbipartite graphs, in: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5–7, 2014, 2014, pp. 239–258.