A Framework for Sampling-Based XML Data Pricing

Ruiming Tang[1], Antoine Amarilli[2], Pierre Senellart[1,2], and Stéphane Bressan[1]

[1] National University of Singapore, Singapore
tangruiming1987@gmail.com,steph@nus.edu.sg
[2] Institut Mines–Télécom; Télécom ParisTech; CNRS LTCI. Paris, France
{antoine.amarilli,pierre.senellart}@telecom-paristech.fr

Abstract. While price and data quality should define the major trade-off for consumers in data markets, prices are usually prescribed by vendors and data quality is not negotiable. In this paper we study a model where data quality can be traded for a discount. We focus on the case of XML documents and consider completeness as the quality dimension.
In our setting, the data provider offers an XML document, and sets both the price of the document and a weight to each node of the document, depending on its potential worth. The data consumer proposes a price. If the proposed price is lower than that of the entire document, then the data consumer receives a sample, i.e., a random rooted subtree of the document whose selection depends on the discounted price and the weight of nodes. By requesting several samples, the data consumer can iteratively explore the data in the document.
We present a *pseudo-polynomial time* algorithm to select a rooted subtree with prescribed weight uniformly at random, but show that this problem is unfortunately intractable. Yet, we are able to identify several practical cases where our algorithm runs in polynomial time. The first case is uniform random sampling of a rooted subtree with prescribed size rather than weights; the second case restricts to binary weights.
As a more challenging scenario for the sampling problem, we also study the uniform sampling of a rooted subtree of prescribed weight and prescribed height. We adapt our pseudo-polynomial time algorithm to this setting and identify tractable cases.

1 Introduction

There are three kinds of actors in a data market: data consumers, data providers, and data market owners [14]. A data provider brings data to the market and sets prices on the data. A data consumer buys data from the market and pays for it. The owner is the broker between providers and consumers, who negotiates pricing schemes with data providers and manages transactions to trade data.

In most of the data pricing literature [4–6, 9], data prices are prescribed and not negotiable, and give access to the best data quality that the provider can achieve. Yet, data quality is an important axis which should be used to price documents in data markets. Wang et al. [15, 19] define dimensions to assess data

quality following four categories: intrinsic quality (believability, objectivity, accuracy, reputation), contextual quality (value-added, relevancy, timeliness, ease of operation, appropriate amount of data, completeness), representational quality (interpretability, ease of understanding, concise representation, consistent representation), and accessibility quality (accessibility, security).

In this paper, we focus on contextual quality and propose a data pricing scheme for *XML trees* such that *completeness* can be traded for discounted prices. This is in contrast to our previous work [18] where the *accuracy* of *relational data* is traded for discounted prices. Wang et al. [15, 19] define completeness as "the extent to which data includes all the values, or has sufficient breadth and depth for the current task". We retain the first part of this definition as there is no current task defined in our setting. Formally, the data provider assigns, in addition to a price for the entire document, a *weight* to each node of the document, which is a function of the potential worth of this node: a higher weight is given to nodes that contain information that is more valuable to the data consumer. We define the completeness of a rooted subtree of the document as the total weight of its nodes, divided by the total weight of the document. A data consumer can then offer to buy an XML document for less than the provider's set price, but then can only obtain a rooted subtree of the original document, whose completeness depends on the discount granted.

A data consumer may want to pay less than the price of the entire document for various reasons: first, she may not be able to afford it due to limited budget but may be satisfied by a fragment of it; second, she may want to explore the document and investigate its content and structure before purchasing it fully.

The data market owner negotiates with the data provider a pricing function, allowing them to decide the price of a rooted subtree, given its completeness (i.e., the weight). The pricing function should satisfy a number of axioms: the price should be non-decreasing with the weight, be bounded by the price of the overall document, and be *arbitrage-free* when repeated requests are issued by the same data consumer (arbitrage here refers to the possibility to strategize the purchase of data). Hence, given a proposed price by a data consumer, the inverse of the pricing function decides the completeness of the sample that should be returned. To be fair to the data consumer, there should be an equal chance to explore every possible part of the XML document that is worth the proposed price. Based on this intuition, we sample a rooted subtree of the XML document of a certain weight, according to the proposed price, uniformly at random.

The data consumer may also issue repeated requests as she is interested in this XML document and wants to explore more information inside in an iterative manner. For each repeated request, a new rooted subtree is returned. A principle here is that the information (document nodes) already paid for should not be charged again. Thus, in this scenario, we sample a rooted subtree of the XML document of a certain weight uniformly at random, without counting the weight of the nodes already bought in previously issued requests.

The present article brings the following contributions:

- We propose to realize the trade-off between quality and discount in data markets. We propose a framework for pricing the completeness of XML data, based on uniform sampling of rooted subtrees of prescribed weight in weighted XML documents. (Section 3)
- We show that the general uniform sampling problem in weighted XML trees is intractable. In this light, we propose two restrictions: sampling based on the number of nodes, and sampling when weights are binary (i.e., weights are 0 or 1). (Section 4)
- We propose a pseudo-polynomial time algorithm for the general uniform sampling problem on prescribed weight, with the proof of its correctness and complexity. (Section 5)
- We show that the two restricted problem variants are tractable by showing that the pseudo-polynomial time algorithm for the general sampling problem runs in polynomial time for uniform sampling based on the size of a rooted subtree, or on 0/1-weights. (Section 6)
- We extend our framework to the case of repeated sampling requests with the requirement that the data consumer is never charged twice for the same nodes. Again, we obtain tractability when the weight of a subtree is its size. (Section 7)
- As a more challenge scenario, we study the unform sampling problem on both prescribed weight and height. We devise a pseudo-polynomial time to solve this sampling problem and also identify tractable cases for which the pseudo-polynomial time sampling algorithm performs in polynomial-time. (Section 8)

This article is the journal version of our previous work [17], extended with the pseudo-polynomial time algorithm for the general weighted sampling problem, and the problem of sampling for prescribed weight and height.

2 Related Work

Data Pricing. The basic structure of data markets and different pricing schemes were introduced in [14]. The notion of "query-based" pricing was introduced in [4, 6] to define the price of a query as the price of the cheapest set of pre-defined views that can determine the query. It makes data pricing more flexible, and serves as the foundation of a practical data pricing system [5]. The price of aggregate queries has been studied in [9]. Different pricing schemes are investigated and multiple pricing functions are proposed to avoid several pre-defined arbitrage situations in [10]. However, none of the works above takes data quality into account, and those works do not allow the data consumer to propose a price less than that of the data provider, which is the approach that we study here.

The idea of trading off price for data quality has been explored in the context of privacy in [8], which proposes a theoretic framework to assign prices to noisy query answers. If a data consumer cannot afford the price of a query, she can choose to tolerate a higher standard deviation to lower the price. However, this work studies pricing on accuracy for linear relational queries, rather than

pricing XML data based on completeness. In [18], we propose a relational data pricing framework in which data accuracy can be traded for discounted prices. By contrast, this paper studies pricing for XML data, and proposes a tradeoff based on data completeness rather than accuracy.

Subtree/Subgraph Sampling. The main technical result of this paper is the tractability of uniform subtree sampling under a certain requested size. This question is related to the general topic of subtree and subgraph sampling, but, to our knowledge, it has not yet been adequately addressed.

Subgraph sampling works such as [3, 7, 13, 16] have proposed algorithms to sample small subgraphs from an original graph while attempting to preserve selected metrics and properties such as degree distribution, component distribution, average clustering coefficient and community structure. However, the distribution from which these random graphs are sampled is not known and cannot be guaranteed to be uniform.

Other works have studied the problem of uniform sampling [2, 11]. However, [2] does not propose a way to fix the size of the samples. The authors of [11] propose a sampling algorithm to sample a connected sub-graph of size k under an approximately uniform distribution; note that this work provides no bound on the error relative to the uniform distribution.

Sampling approaches are used in [20, 12] to estimate the selectivity of XML queries (containment join and twig queries, respectively). Nevertheless, the samples in [20] are specific to containment join queries, while those in [12] are representatives of the XML document for any twig queries. Neither of those works controls the distribution from which the subtrees are sampled.

In [1], Cohen and Kimelfeld show how to evaluate a deterministic tree automaton on a probabilistic XML document. This has applications to sampling possible worlds that satisfy a given constraint, e.g., expressed in monadic second-order logic and then translated into a tree automaton. Note that the translation of constraints to tree automata itself is not tractable in general; in this respect, our approach can be seen as a specialization of [1] to the simpler case of fixed-size, fixed-weight, or fixed-height tree sampling, and as an application of it to data pricing.

3 Pricing Function and Sampling Problem

This paper studies data pricing for tree-shaped documents. Let us first formally define the terminology that we use for such documents.

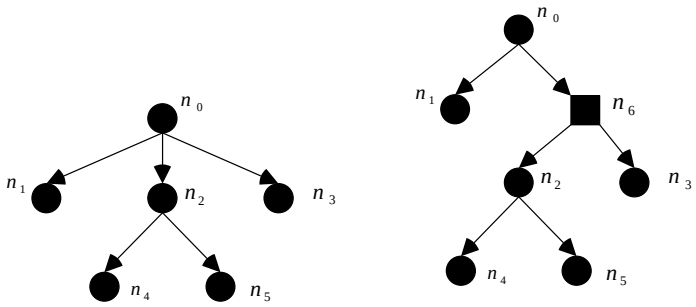
We consider trees that are unordered, directed, rooted, and weighted. Formally, a tree t consists of a set of nodes $\mathsf{V}(t)$ (which are assumed to carry unique identifiers), a set of edges $\mathsf{E}(t)$, and a function w mapping every node $n \in \mathsf{V}(t)$ to a non-negative rational number $\mathsf{w}(n)$ which is the *weight* of node n. We write $\mathsf{root}(t)$ for the root node of t. Any two nodes n_1, $n_2 \in \mathsf{V}(t)$ such that $(n_1, n_2) \in \mathsf{E}(t)$ are in a *parent-child relationship*, that is, n_1 is the parent of n_2 and n_2 is a child of n_1.

By $\mathsf{children}(n)$, we represent the set of nodes that have parent n. A tree is said to be *binary* if each node of the tree has at most two children, otherwise it is *unranked*. Throughout this paper, for ease of presentation, we may call such trees "XML documents".

We now introduce the notion of *rooted subtree* of an XML document:

Definition 1. *(Subtree, rooted subtree) A tree t' is a* subtree *of a tree t if $\mathsf{V}(t') \subseteq \mathsf{V}(t)$ and $\mathsf{E}(t') \subseteq \mathsf{E}(t)$. A* rooted subtree *$t'$ of a tree t is a subtree of t such that $\mathsf{root}(t) = \mathsf{root}(t')$. We name it* r-subtree *for short. The weight function for a subtree t' of a tree t is always assumed to be the restriction of the weight function for t on the nodes in t'.*

For technical reasons, we also sometimes talk of the *empty* subtree that contains no node.



(a) A tree (b) A binary tree

Fig. 1. Two example trees (usage of the square node will be introduced in Section 5.2)

Example 1. Figure 1 presents two example trees. The nodes $\{n_0, n_2, n_5\}$, along with the edges connecting them, form an r-subtree of the tree in Figure 1(a). Likewise, the nodes $\{n_2, n_4, n_5\}$ and the appropriate edges form a subtree of that tree (but not an r-subtree). The tree of Figure 1(b) is a binary tree (ignore the different shapes of the nodes for now). □

We now present our notion of data quality, by defining the completeness of an r-subtree, based on the weight function of the original tree:

Definition 2. *(Weight of a tree) For a node $n \in \mathsf{V}(t)$ of a tree t, we define inductively $\mathsf{weight}(n) \coloneqq \mathsf{w}(n) + \sum_{(n,n') \in \mathsf{E}(t)} \mathsf{weight}(n')$. With slight abuse of notation, we note $\mathsf{weight}(t) \coloneqq \mathsf{weight}(\mathsf{root}(t))$ as the* weight *of t.*

Definition 3. *(Completeness of an r-subtree) Let t be a tree and t' be an r-subtree of t. The* completeness *of t' with respect to t is $c_t(t') \coloneqq \frac{\mathsf{weight}(t')}{\mathsf{weight}(t)}$. It is obvious that $c_t(t') \in [0, 1]$.*

We study a framework for data markets where the data consumer can buy an incomplete document from the data provider while paying a discounted price. The formal presentation of this framework consists of three parts:

1. An XML document t.
2. A pricing function φ_t for t whose input is the desired completeness for an r-subtree of the XML document, and whose value is the price of this r-subtree. Hence, given a proposed price pr_0 by a data consumer, the completeness of the returned r-subtree is decided by $\varphi_t^{-1}(pr_0)$.
3. An algorithm to sample an r-subtree of the XML document uniformly at random among those of a given completeness.

We study the question of the sampling algorithm more in detail in subsequent sections. For now, we focus on the pricing function, starting with a formal definition:

Definition 4. *(Pricing function) The* pricing function *for a tree t is a function $\varphi_t : [0,1] \to \mathbb{Q}^+$. Its input is the completeness of an r-subtree t' and it returns the price of t', as a non-negative rational.*

A healthy data market should impose some restrictions on φ_t, such as:

Non-decreasing. The more complete an r-subtree is, the more expensive it should be, i.e., $c_1 \geqslant c_2 \Rightarrow \varphi_t(c_1) \geqslant \varphi_t(c_2)$.

Arbitrage-free. Buying an r-subtree of completeness $c_1 + c_2$ should not be more expensive than buying two subtrees with respective completeness c_1 and c_2, i.e., $\varphi_t(c_1) + \varphi_t(c_2) \geqslant \varphi_t(c_1 + c_2)$. In other words, φ_t should be subadditive. This property is useful when considering repeated requests, studied in Section 7.

Minimum and maximum bound. We should have $\varphi_t(0) = pr_{\min}$ and $\varphi_t(1) = pr_t$, where $pr_{\min}$ is the minimum cost that a data consumer has to pay using the data market and pr_t is the price of the whole tree t. Note that by the non-decreasing character of φ_t, $pr_t \geqslant pr_{\min} \geqslant 0$.

All these properties can be satisfied, for instance, by functions of the form $\varphi_t(c) \coloneqq (pr_t - pr_{\min})c^p + pr_{\min}$ where $p \leqslant 1$; however, if $p > 1$, the arbitrage-free property is violated.

Given a proposed price pr_0 by a data consumer, $\varphi_t^{-1}(pr_0)$ is the set of possible corresponding completeness values. Note that φ_t^{-1} is a relation and may not be a function; φ_t^{-1} is a function if different completeness values correspond to different prices. Once a completeness value $c \in \varphi_t^{-1}(pr_0)$ is chosen, the weight of the returned r-subtree is fixed as $c \times \mathsf{weight}(t)$.

Therefore, in the rest of the paper, we consider the problem of uniform sampling an r-subtree with prescribed weight (instead of with prescribed completeness). Let us now define the problem that should be solved by our sampling algorithm:

Definition 5. *(Sampling problem) The problem of* sampling an r-subtree*, given a tree t and a weight k, is to sample an r-subtree t' of t, such that $\mathsf{weight}(t') = k$, uniformly at random, if one exists, or to fail if no such r-subtree exists.*

4 Tractability

Having defined our sampling problem, we now turn to the question of designing an algorithm to solve it, and of studying its complexity.

4.1 Intractability of the Sampling Problem

We start by showing that, sadly, this problem is NP-hard in the general formulation that we gave.

Proposition 1. *Given a tree t and a weight k, it is* NP*-complete to decide whether there exists an r-subtree of weight k, and* NP*-hard to sample such an r-subtree uniformly at random.*

Proof. Deciding whether there exists an r-subtree of weight k is in NP, since, given an r-subtree, it takes polynomial time to check whether this r-subtree is of weight k by summing up the weights of all the nodes.

We now show that the problem is NP-hard, by describing a PTIME reduction from the NP-hard subset-sum problem. This is the problem of determining, given a set S of integers and a target value v (written in binary), whether there exists a subset $S' \subseteq S$ which sums to v. Any set S can be encoded in polynomial time to a tree t such that $\mathsf{w}(n) = 0$ except if n is a leaf, and the leaves correspond to the elements of S. Now, clearly there is an r-subtree of weight v in t iff there is a subset of S with sum v. This completes the reduction and shows that the problem of deciding the existence of an r-subtree of weight k is NP-complete.

Now there is a PTIME-reduction from the decision problem to the sampling problem, as an algorithm for sampling can be used to decide whether there exists an r-subtree of the desired weight (the algorithm returns one such) or if none exists (the algorithm fails). Therefore, the sampling problem is NP-hard. □

Even though the general sampling problem is intractable, we devise in Section 5 a *pseudo-polynomial time* algorithm to solve it, which runs in polynomial time in the *value* of k (but is exponential in the size of k).

4.2 Tractable Cases

We now define restricted variants of the sampling problem where the weight function is assumed to be of a certain form. In Section 6, we show that sampling for these variants can be performed in PTIME.

Unweighted Sampling. In this setting, we take the weight function $\mathsf{w}(n) = 1$ for all $n \in \mathsf{V}(t)$. Hence, the weight of a tree t is actually the number of nodes in t, i.e., its size, which we write $\mathsf{size}(t)$.

In this case, the hardness result of Proposition 1 does not apply anymore. However, sampling an r-subtree with prescribed size uniformly at random is still not obvious to do, as the following example shows:

Example 2. Consider the problem of sampling an r-subtree t' of size 3 from the tree t in Figure 1(a). We can enumerate all such r-subtrees: $\{n_0, n_1, n_2\}$, $\{n_0, n_1, n_3\}$, $\{n_0, n_2, n_3\}$, $\{n_0, n_2, n_4\}$ and $\{n_0, n_2, n_5\}$, and choose one of them at random with probability $\frac{1}{5}$. However, as the number of r-subtrees may be exponential in the size of the document in general, we cannot hope to perform this approach in PTIME.

Observe that it is not easy to build a random r-subtree node by node: it is clear that node n_0 must be included, but then observe that we cannot decide to include n_1, n_2, or n_3 uniformly at random. Indeed, if we do this, our distribution on the r-subtrees will be skewed, as n_1 (or n_3) occurs in $\frac{2}{5}$ of the outcomes whereas n_2 occurs in $\frac{4}{5}$ of them. Intuitively, this is because there are more ways to choose the next nodes when n_2 is added, than when n_1 or n_3 are added. □

0/1-weights Sampling. In this problem variant, we require that $\mathsf{w}(n) \in \{0, 1\}$ for all $n \in \mathsf{V}(t)$, i.e., the weight is a binary value. This variant generalizes the unweighted sampling case, but allows the data provider to give a weight of zero to some nodes that she is willing to give away for free. This can be useful, e.g., for nodes that are only structural and do not contain any useful information.

5 Algorithms for General Sampling Problem

In this section, we present a pseudo-polynomial algorithm for the general sampling problem, namely the problem of sampling an r-subtree of weight k from an XML document, uniformly at random.

We first describe the algorithm for the case of *binary* trees, in Section 5.1. Next, we adapt the algorithm in Section 5.2 to show how to apply it to arbitrary trees.

5.1 Sampling for Binary Trees

In this section, we provide an algorithm which proves the following theorem:

Theorem 1. *The sampling problem for binary trees can be solved in time $O(nk^2)$, where n is the number of nodes in the tree and k is the desired weight value.*

Our general algorithm to solve this problem is given as Algorithm 1. The algorithm has two phases, which we study separately in what follows. For simplicity, whenever we discuss binary trees in this section, we will add special NULL children to every node of the tree (except NULL nodes themselves), so that all nodes, including leaf nodes (but excluding NULL nodes), have exactly two children (which may be NULL). This will simplify the presentation of the algorithms. Of course $\mathsf{weight}(\mathsf{NULL}) = 0$.

Algorithm 1: Algorithm for the sampling problem on binary trees

Input: a binary tree t and an integer $k \geqslant 0$
Result: an r-subtree t' of t of $\mathsf{weight}(t') = k$ uniformly at random
// Phase 1: count the number of subtrees
1 $D \leftarrow \mathsf{SubtreeCounting}(t)$;
// Phase 2: sample a random subtree
2 **if** $k \leqslant \mathsf{weight}(t) \wedge D_{\mathsf{root}(t)}[k] \neq 0$ **then**
3 | **return** $\mathsf{UniformSampling}(\mathsf{root}(t), D, k)$;
4 **else**
5 | **fail**;

First phase: Subtree Counting (Algorithm 2). We start by computing a matrix D such that, for every node n_i of the input tree t and any value $0 \leqslant x \leqslant \mathsf{weight}(t)$, $D_i[x]$ is the number of subtrees of weight x rooted at node n_i. We do so with Algorithm 2 which we now explain in detail.

There is only one subtree rooted at the special NULL node, namely the empty subtree, with weight 0, which provides the base case of the algorithm (line 3). Otherwise, we compute D_i for a node n_i from the values D_l and D_r of D for its children n_l and n_r (which may be NULL); those values have been computed before because nodes are considered bottom-up.

Intuitively, any r-subtree of weight $x > 0$ rooted at n_i is obtained by retaining n_i, and choosing two r-subtrees t_l and t_r, respectively rooted at n_l and n_r (the children of n_i), such that $\mathsf{weight}(t_l) + \mathsf{weight}(t_r) = x - \mathsf{w}(n_i)$ (which accounts for the weight of the additional node n_i). The number of such choices is computed by the *convolution* of D_l and D_r in line 7, defined as:

$$\text{For } 0 \leqslant p \leqslant \mathsf{weight}(t), \quad (D_l * D_r)[p] := \sum_{m=0}^{p} D_l[m] \times D_r[p-m].$$

We explain Algorithm 2 by considering the cases of $\mathsf{w}(n_i) = 0$ (line 8 to line 11) and of $\mathsf{w}(n_i) \neq 0$ (line 13 to line 15), respectively.

If $\mathsf{w}(n_i) = 0$, the number of r-subtrees of weight $x > 0$ rooted at n_i is the number of pairs of r-subtrees t_l and t_r, respectively rooted at n_l and n_r (the children of n_i), such that $\mathsf{weight}(t_l) + \mathsf{weight}(t_r) = x$ (because node n_i does not contribute weight). That is to say, $D_i[x] = (D_l * D_r)[x]$ for $x > 0$ (line 11). By contrast, for $x = 0$, an r-subtree of weight 0 rooted at n_i can be obtained either in the same way, or by keeping the empty subtree. Therefore $D_i[0] = 1 + (D_l * D_r)[0]$ (line 9).

If $\mathsf{w}(n_i) \neq 0$, there is only one r-subtree of weight 0 at n_i, namely the empty tree. That is to say, $D_i[0] = 1$ as shown in line 13. The number of r-subtrees of weight $x \geqslant \mathsf{w}(n_i)$ rooted at n_i is the number of pairs of r-subtrees t_l and t_r rooted at n_l and n_r (the children of n_i) respectively such that $\mathsf{weight}(t_l) + \mathsf{weight}(t_r) = x - \mathsf{w}(n_i)$, which implies that $D_i[x] = (D_l * D_r)[x - \mathsf{w}(n_i)]$ (line 15). For $x \in [1, \mathsf{w}(n_i) - 1]$, it is impossible to get an r-subtree of weight x at n_i, so $D_i[x]$ remains at 0.

Algorithm 2: SubtreeCounting(t)

Input: a binary tree t
Result: a matrix D such that $D_i[x]$ is the number of r-subtrees of weight x rooted at n_i for all n_i and x

```
1  for x ∈ [0, weight(t)], n_i ∈ V(t) ⊔ {NULL} do
2  |  D_i[x] ← 0;
3  D_NULL[0] ← 1;
   // We browse all nodes in topological order, from leaves to the root
4  foreach non-NULL node n_i accessed bottom-up do
5  |  n_l ← first child of n_i;
6  |  n_r ← second child of n_i;
7  |  T ← D_l * D_r;
8  |  if w(n_i) = 0 then
9  |  |  D_i[0] ← 1 + T[0];
10 |  |  for x ∈ [1, weight(n_i)] do
11 |  |  |  D_i[x] ← T[x];
12 |  else
13 |  |  D_i[0] ← 1;
14 |  |  for x ∈ [w(n_i), weight(n_i)] do
15 |  |  |  D_i[x] ← T[x − w(n_i)];
16 return D;
```

Example 3. Let t be the tree presented in Figure 1(b) (again, ignore the different shapes of nodes for now). Assume $\mathsf{w}(n_0) = \mathsf{w}(n_1) = \mathsf{w}(n_2) = 0$, $\mathsf{w}(n_3) = \mathsf{w}(n_4) = 1$, $\mathsf{w}(n_5) = \mathsf{w}(n_6) = 2$. Starting from the leaf nodes, we compute $D_4 = D_3 = (1, 1)$ and $D_5 = (1, 0, 1)$, with $T = D_{\mathsf{NULL}} * D_{\mathsf{NULL}} = (1)$, by line 13 to line 15. We compute $D_1 = (2)$ with line 8 to line 11.

Now, when computing D_2, we first convolve D_4 and D_5 to get the numbers of pairs of r-subtrees of different weights at $\{n_4, n_5\}$, i.e., $D_4 * D_5 = (1, 1, 1, 1)$, so that $D_2 = (2, 1, 1, 1)$ (applying line 8 to line 11). When computing D_6, we first compute $D_2 * D_3 = (2, 3, 2, 2, 1)$, so that $D_6 = (1, 0, 2, 3, 2, 2, 1)$ (applying line 13 to line 15). Finally, $D_0 = (3, 0, 4, 5, 4, 4, 3)$. □

We now state the correctness and running time of this algorithm.

Lemma 1. *Algorithm 2 terminates in $O(nk^2)$ time (where n is the number of nodes in the given tree, and k is the desired weight) and returns D such that, for every n_i and x, $D_i[x]$ is the number of r-subtrees of weight x rooted at node n_i.*

Proof. Let us first prove the running time. All arrays under consideration have size at most W (where $W = \mathsf{weight}(t)$), so computing the convolution of two such arrays is in time $O(W^2)$. The number of convolutions to compute overall is $O(n)$, because each array D_i occurs in exactly one convolution. The overall running time is thus $O(nW^2)$. The running time can be optimized because, in fact, the arrays in D do not need to have size W, but can be bounded to size k

(subtrees larger than k are never relevant). Therefore the complexity is in fact $O(nk^2)$.

Let us now show correctness. We proceed by induction on the node n_i to prove the claim for every x. The base case is the NULL node, whose correctness is straightforward. Let us prove the induction step. Let n_i be a node, and assume by induction that $D_l[x']$ is correct for every x' and every child n_l of n_i. Let us fix x and show that $D_i[x]$ is correct.

Let us separate the proof in two cases: $\mathsf{w}(n_i) \neq 0$ and $\mathsf{w}(n_i) = 0$.

We first prove the case where $\mathsf{w}(n_i) \neq 0$. To select an r-subtree at n_i, (1) if $x = 0$, there is exactly one possibility (the empty subtree); (2) if $x \in [1, \mathsf{w}(n_i))$, there is no such possibility; (3) if $x \geqslant \mathsf{w}(n_i)$, the number of possibilities is the number of ways to select a pair of r-subtrees at the children of n_i so that their weights sum to $x - \mathsf{w}(n_i)$. This is the role of line 13 to line 15.

Now, to enumerate the ways of choosing r-subtrees at children of n_i whose weight sum to $x - \mathsf{w}(n_i)$, we can first decide the weight of the selected r-subtree for each child: the ways to assign such weights form a partition of the possible outcomes, so the number of outcomes is the sum, over all such assignments of r-subtree weights to children, of the number of outcomes for this assignment. For a fixed assignment, the subtrees rooted at each children are chosen separately, so the number of outcomes for a fixed assignment is the product of the number of outcomes for the given weight for each child, which by induction hypothesis is correctly reflected by the corresponding $D_l[x']$. Hence, for a given x, (1) when $x = 0$, $D_i[x] = 1$ (line 13); (2) when $x \in [1, \mathsf{w}(n_i))$, $D_i[x] = 0$ (from how D_i is initialized); (3) when $x \geqslant \mathsf{w}(n_i)$, $D_i[x]$ is $(D_l * D_r)[x - \mathsf{w}(n_i)]$ by line 15, which sums, over all possible subtree weights assignments, the number of choices for this subtree weight assignment.

We next prove the case where $\mathsf{w}(n_i) = 0$. To select an r-subtree at n_i, (1) if $x = 0$, the number of possibilities is the number of ways to select a pair of r-subtrees at the children of n_i so that their weights sum to 0 plus one (this extra possibility is the empty subtree); (2) if $x > 0$, the number of possibilities is the number of ways to select a pair of r-subtrees at the children of n_i so that their weights sum to x, because node n_i contributes no weight. Similar to the previous case, the number of ways to select a pair of r-subtrees at the children of n_i so that their weights sum to x is $(D_l * D_r)[x]$. Therefore, for a given x, (1) when $x = 0$, $D_i[x] = 1 + (D_l * D_r)[0]$ (line 9); (2) when $x \neq 0$, $D_i[x] = (D_l * D_r)[x]$ (line 11).

Hence, by induction, we have shown the desired claim. □

Second phase: Uniform Sampling (Algorithm 3). In the second phase of Algorithm 1, we sample an r-subtree from t in a recursive top-down manner, based on the matrix D computed by Algorithm 2. Our algorithm to perform this uniform sampling is Algorithm 3. The basic idea is that to sample an r-subtree rooted at a node n_i, we decide on the weight of the subtrees rooted at each

Algorithm 3: UniformSampling(n_i, D, x)

Input: a node n_i (or NULL), the precomputed D, and a weight value x
Result: an r-subtree of weight x at node n_i

1 **if** $x = 0 \wedge \mathsf{w}(n_i) \neq 0$ **then**
2 **return** $\emptyset$;
3 $p_0 \leftarrow \mathrm{rand}([0,1])$; // p_0 is generated from $[0,1]$ randomly
4 **if** $x = 0 \wedge \mathsf{w}(n_i) = 0 \wedge p_0 \leqslant \frac{1}{D_i[0]}$ **then**
5 **return** $\emptyset$;
6 $n_l \leftarrow$ first child of n_i;
7 $n_r \leftarrow$ second child of n_i;
8 **for** $0 \leqslant s_l, s_r \leqslant x$ **s.t.** $s_l + s_r = x - \mathsf{w}(n_i)$ **do**
9 $p(s_l, s_r) \leftarrow D_l[s_l] \times D_r[s_r]$;
10 Sample an (s_l, s_r) with probability $p(s_l, s_r)$ normalized by $\sum_{s_l, s_r} p(s_l, s_r)$;
11 $L \leftarrow \mathsf{UniformSampling}(n_l, D, s_l)$;
12 $R \leftarrow \mathsf{UniformSampling}(n_r, D, s_r)$;
13 **return** the tree rooted at n_i with child subtrees L and R;

child node, biased by the number of outcomes as counted in D, and then sample r-subtrees of the desired weights recursively.

Let us now explain Algorithm 3 in detail.

If $x = 0$ and $\mathsf{w}(n_i) \neq 0$, we must return the empty tree (line 1 to line 2), since the empty tree is the only choice in this case. If $x = 0$ and $\mathsf{w}(n_i) = 0$, there are $D_i[0]$ r-subtrees of weight 0 at node n_i, of which one is the empty tree while the others are non-empty retaining n_i. Therefore, to ensure the unform distribution of the samples, we return the empty tree with probability $\frac{1}{D_i[0]}$ (line 4 to line 5), and return a non-empty tree of weight 0 retaining n_i with probability $1 - \frac{1}{D_i[0]}$ (the rest of the algorithm).

Except for the above two cases that return the empty subtree, we return n_i and subtrees t_l and t_r rooted at the children n_l and n_r of n_i. We first decide on the weight s_l and s_r of t_l and t_r (line 8 to line 10), biasing by the number of outcomes for each weight combination, and then recursively sample a subtree of the prescribed weight (line 11 to line 12), uniformly at random, and return it.

To be a suitable choice, the weight pair (s_l, s_r) must be such that $s_l + s_r = x - \mathsf{w}(n_i)$ (which accounts for node n_i). Intuitively, to perform a uniform sampling, we now observe that the choice of the weight pair (s_l, s_r) partitions the set of outcomes. Hence, the probability that we select one weight pair should be proportional to the number of possible outcomes for this pair, namely, the number of r-subtrees t_l and t_r such that $\mathsf{weight}(t_l) = s_l$ and $\mathsf{weight}(t_r) = s_r$. We compute this from D_l and D_r (line 9) by observing that the number of pairs (t_l, t_r) is the product of the number of choices for t_l and for t_r, as every combination of choices is possible.

Example 4. Follow Example 3. Assume we want to sample an r-subtree t' of $\mathsf{weight}(t') = 3$ uniformly. Let D be the result of Algorithm 2.

We first call $\mathsf{UniformSampling}(n_0, D, 3)$. We have to return n_0 (as $\mathsf{w}(n_0) = 0$). Now n_0 has two children, n_1 and n_6. The only possible weight pair is $(0, 3)$, with probability $p(0, 3) = 1$. We now call recursively $\mathsf{UniformSampling}(n_1, D, 0)$ and $\mathsf{UniformSampling}(n_6, D, 3)$.

When calling$\mathsf{UniformSampling}(n_1, D, 0)$, since $\mathsf{w}(n_1) = 0$, we return $\emptyset$ with probability $\frac{1}{D_1[0]} = \frac{1}{2}$ and return n_1 with probability $1 - \frac{1}{D_1[0]} = \frac{1}{2}$. Assume n_1 is returned.

We proceed to $\mathsf{UniformSampling}(n_6, D, 3)$. We have to return n_6 (note that $\mathsf{w}(n_6) = 2$). Now n_6 has two children, n_2 and n_3. The possible weight pairs for this call are $(1, 0)$ and $(0, 1)$, with respective (unnormalized) probabilities $p(1, 0) = D_2[1] \times D_3[0] = 1 \times 1 = 1$ and $p(0, 1) = D_2[0] \times D_3[1] = 2 \times 1 = 2$. The normalized probabilities are $p(1, 0) = \frac{1}{3}$ and $p(0, 1) = \frac{2}{3}$. Assume that we choose $(0, 1)$ with probability $\frac{2}{3}$. We now call recursively $\mathsf{UniformSampling}(n_2, D, 0)$ and $\mathsf{UniformSampling}(n_3, D, 1)$.

When calling $\mathsf{UniformSampling}(n_2, D, 0)$ (note $\mathsf{w}(n_2) = 0$), we return $\emptyset$ with probability $\frac{1}{D_2[0]} = \frac{1}{2}$ and return n_2 with probability $1 - \frac{1}{D_2[0]} = \frac{1}{2}$. Assume n_2 is returned.

We finish with $\mathsf{UniformSampling}(n_3, D, 1)$. Node n_3 is selected.

Hence, the end result is the r-subtree whose nodes are $\{n_0, n_1, n_6, n_2, n_3\}$ (and whose edges can clearly be reconstituted in PTIME from t). This r-subtree is selected with the probability $\frac{1}{2} \times \frac{2}{3} \times \frac{1}{2} = \frac{1}{6}$. Indeed, recall that we know there are 6 r-subtrees of t of weight 3, according to $D_0[3] = 6$. □

We now show the tractability and correctness of Algorithm 3, concluding the proof of Theorem 1.

Lemma 2. *For any tree t, node $n_i \in \mathsf{V}(t)$ and integer $0 \leqslant x \leqslant \mathsf{weight}(n_i)$, given D computed by Algorithm 2, $\mathsf{UniformSampling}(n_i, D, x)$ terminates in $O(nk)$ time (where n is the number of nodes in the given tree and k is the desired weight) and returns an r-subtree of weight x rooted at n_i, uniformly at random (i.e., solves the sampling problem for binary trees).*

Proof. Let us first prove the complexity claim. On every node n_i of the binary tree t, the number of possibilities to consider is at most k, and for each possibility the number of operations performed is constant (assuming that drawing a number uniformly at random can be performed in constant time). The overall running time is $O(nk)$.

We now show correctness by induction on n_i. The base case is $n_i = \mathsf{NULL}$, in which case we must have $x = 0$ and we correctly return $\emptyset$. Let us now assume that n_i is not NULL. If $x = 0$ and $\mathsf{w}(n_i) \neq 0$, the only possibility is the empty subtree and we correctly return $\emptyset$. If $x = 0$ and $\mathsf{w}(n_i) = 0$, there are $D_i[0]$ r-subtrees of weight 0 at node n_i, of which one is the empty subtree while the others are non-empty and retain n_i. To ensure that the distribution is uniform, we return the empty tree with probability $\frac{1}{D_i[0]}$ and return a non-empty tree of weight 0

retaining n_i with probability $1 - \frac{1}{D_i[0]}$. Otherwise, we need to return an r-subtree retaining n_i. As in the proof of Lemma 1, the set of possible outcomes of the sampling process is partitioned by the possible assignments, and only the valid ones correspond to a non-empty set of outcomes. Hence, we can first choose a weight pair, *weighted by the proportion of outcomes which are outcomes for this pair*, and then choose an outcome for this pair. Now, observe that, by Lemma 1, D correctly represents the number of outcomes for each child of n_i, so that our computation of p (which mimics that of Algorithm 2) correctly represents the proportion of outcomes for each weight pair. We then choose an assignment according to p, and then observe that choosing an outcome for this assignment amounts to choosing an outcome for each child of n_i whose weight is given by the assignment. By induction hypothesis, this is precisely what the recursive calls to $\mathsf{UniformSampling}(n_i, D, x)$ perform. This concludes the proof. □

5.2 Sampling for Unranked Trees

In this section, we show that the algorithm of the previous section can be adapted so that it works on arbitrary unranked trees, not just binary trees.

We first observe that the straightforward generalization of Algorithm 1 to trees of arbitrary arity, where assignments and convolutions are performed for all children, is still correct. However, its running time would no longer be polynomial in n and k, as there would be a potentially exponential number of weight assignments to consider.

Fortunately, there is still hope to avoid considering all weights assignments over all the children, because *convolution is associative*. Informally, assuming we have three children $\{n_1, n_2, n_3\}$, we do the following: we treat $\{n_1\}$ as a group and $\{n_2, n_3\}$ as the second group, then enumerate weight pairs over $\{n_1\}$ and $\{n_2, n_3\}$; once a weight pair, in which a positive integer is assigned to $\{n_2, n_3\}$, is selected, we can treat $\{n_2\}$ and $\{n_3\}$ as new groups and enumerate weight pairs over $\{n_2\}$ and $\{n_3\}$. In other words, this strategy can be implemented by transforming the original tree to a binary tree.

We now present an encoding process transforming unranked trees to *encoded trees*, which are binary trees whose nodes are either regular nodes or *dummy* nodes. Intuitively, the encoding operation replaces sequences of more than two children by a hierarchy of dummy nodes representing those children; replacing dummy nodes by the sequence of their children yields back the original tree. The encoding is illustrated in Figure 1, where the tree in Figure 1(b) is the encoded tree of the one in Figure 1(a) (dummy nodes are represented as squares). We require that the weight of every dummy node is 0, i.e., $\mathsf{w}(n_i) = 0$ where n_i is a dummy node. (However, dummy nodes are not exactly equivalent to regular nodes with weight 0, as we must not consider that we have the possibility of either keeping them or not keeping them.)

We formally present an algorithm (Algorithm 4) for this encoding process. Algorithm 4 performs a constant number of operations on every considered node plus a constant number of operations on every child of the considered node. Hence, the overall number of operations performed for every node (both when

Algorithm 4: Transforming to binary tree

Data: a tree t with nodes $n_0, \ldots, n_{k-1}$
Result: the encoded tree t'

```
1  m ← k and t' ← t;
2  for each node n_i of t do
3      if |children(n_i)| > 2 then
4          create dummy nodes n_m, n_{m+1}, ..., n_{m+|children(n_i)|-3} in V(t');
5          for 0 ⩽ j < |children(n_i)| − 3 do
6              create an edge from n_{m+j} to n_{m+j+1} in t';
7          create an edge from n_i to n_m in t';
8          disconnect the 2nd, 3rd, ..., children of n_i from n_i in t';
9          for 0 ⩽ j ⩽ |children(n_i)| − 3 do
10             create an edge in t' from n_{m+j} to the (j + 2)^th child of n_i in t;
11         create an edge in t' from n_{m+|children(n_i)|-3} to the last child of n_i in t;
12         m ← m + |children(n_i)| − 3 + 1;
```

examining it and when examining its unique parent) is constant, so it completes in linear time. For a tree t with n nodes, the number of nodes in its encoded tree t' is no more than $n + n - 3$ (the worst case being achieved by a tree with a root node and $n - 1$ children). Therefore the size of the encoded tree is linear in the size of the original tree. Note that the weights of created dummy nodes are set to 0.

Based on this encoding process, we now state our result:

Theorem 2. *The sampling problem can be solved in $O(nk^2)$ (where n is the number of nodes in the given tree and k is the desired weight), for arbitrary unranked trees.*

Proof. It can be shown that, up to the question of keeping or deleting the dummy nodes with no regular descendants (we call them *bottommost*), there is a bijection between r-subtrees in the original tree and r-subtrees in the encoded tree. Hence, we can solve the sampling problem by choosing an r-subtree in the encoded tree with a set of regular nodes of weight k, uniformly at random, and imposing the choice of keeping bottommost dummy nodes.

We do this by adapting Algorithm 2 and Algorithm 3 to run correctly on encoded trees, that is, managing dummy nodes correctly, by imposing that they are always retained.

In Algorithm 2, we have to define the computation of D_i for a dummy node n_i as $D_i \leftarrow D_l * D_r$ (as it must always be kept, and does not increase the weight of the r-subtree).

In Algorithm 3, some operations have to be distinguished between regular nodes and dummy nodes. In line 1 and line 4, we additionally require in the if clauses that n_i is either NULL or a regular node: indeed, for dummy nodes, even if $x = 0$ we cannot return $\emptyset$, because we must keep them.

The correctness and running time of the modified algorithms can be proved by straightforward adaptations of Lemma 1 and Lemma 2. □

6 Tractable Uniform Sampling

As presented in the previous section, the complexity of Algorithm 1 (and its variant in Theorem 2) is $O(nk^2)$, where n is the number of nodes in the given tree and k is the desired weight. It is thus a *pseudo-polynomial time* algorithm to solve the general sampling problem, which is polynomial in the *value* of k, but is still exponential in the *size* of k.

We now observe that for the tractable cases in Section 4.2, Algorithm 1 (and its variant in Theorem 2) runs in time polynomial in the size of the input tree; more specifically, in $O(n^3)$. Indeed, for unweighted sampling (where $\mathsf{w}(n_i) = 1$ for every n_i) and 0/1-weights sampling (where $\mathsf{w}(n_i) = \{0, 1\}$ for every n_i), the desired weight k is bounded by the size of the tree, since $k \leqslant \mathsf{weight}(t) \leqslant n$. Therefore the complexity of Algorithm 1 (and its variant in Theorem 2) is then $O(n^3)$, that is, cubic in the size of the input tree. Hence, we have the following claim:

Theorem 3. *The unweighted sampling and 0/1-weights sampling can be solved in $O(n^3)$ time, where n is the number of nodes in the given tree.*

7 Repeated Requests

In this section, we consider the more general problem where the data consumer requests a *completion* of a certain price to data that they have already bought. The motivation is that, after having bought incomplete data, the user may realize that they need additional data, in which case they would like to obtain more incomplete data that is not redundant with what they already have.

A first way to formalize the problem is as follows, where data is priced according to a known subtree (provided by the data consumer) by considering that known nodes are free (but that they may or may not be returned again).

Definition 6. *The problem of sampling an r-subtree of weight k in a tree t* conditionally *to an r-subtree t' is to sample an r-subtree t'' of t uniformly at random, such that* $\mathsf{weight}(t'') - \sum_{n \in (\mathsf{V}(t') \cap \mathsf{V}(t''))} \mathsf{w}(n) = k$.

An alternative is to consider that we want to sample an *extension* of a fixed size to the whole subtree, so that all known nodes are always part of the output:

Definition 7. *The problem of sampling an r-subtree of weight k in a tree t that* extends *an r-subtree t' is to sample an r-subtree t'' of t uniformly at random, such that (1) t' is an r-subtree of t''; (2)* $\mathsf{weight}(t'') - \mathsf{weight}(t') = k$.

Note that those two formulations are not the same: the first one does not require the known part of the document to be returned, while the second one does. While it may be argued that the resulting outcomes are essentially equivalent (as they only differ on parts of the data that are already known to the data consumer), it is important to observe that they define different distributions: though both problems require the sampling to be uniform among their set of outcomes, the additional possible outcomes of the first definition means that the underlying distribution is not the same.

As the uniform sampling problem for r-subtrees can be reduced to either problem by setting t' to be the empty subtree, the NP-hardness of those two problems follows from Proposition 1. However, we can show that, in the unweighted case, those problems are tractable, because they reduce to the 0/1-weights sampling problem which is tractable by Theorem 3:

Proposition 2. *The problem of sampling an r-subtree of weight k in a tree t conditionally to an r-subtree t' can be solved in $O(n^3)$ time if t is unweighted. The same holds for the problem of sampling an r-subtree that extends another r-subtree.*

Proof. For the problem of Definition 6, set the weight of the nodes of t' in t to be zero (the intuition is that all the known nodes are free). The problem can then be solved by applying Theorem 2.

For the problem of Definition 7, set the weight of the nodes of t' in t to be zero but we have to ensure that the nodes in t' are always returned. To do so, we adapt Theorem 2 by handling the nodes in t' in the same way as handling dummy nodes in the previous section. □

8 Sampling Extension: Sampling on Weight and Height

In this section, we consider a more complicated sampling scenario: sampling an r-subtree of weight k and height h uniformly at random. We present a pseudo-polynomial time algorithm for this sampling problem. To start with, we define the height of a tree.

Definition 8. *(Height of a tree) For a node $n \in \mathsf{V}(t)$, we define inductively $\mathsf{height}(n) \coloneqq 1 + \max_{(n,n')\in\mathsf{E}(t)} \mathsf{height}(n')$, with $\mathsf{height}(n) = 1$ if n is a leaf of t. With slight abuse of notation, we note $\mathsf{height}(t) = \mathsf{height}(\mathsf{root}(t))$ the* height *of t.*

We first revisit the problem of pricing a tree depending on its weight and height. Then, in Section 8.2, we first describe the algorithm for the case of *binary* trees. Next, we adapt the algorithm in Section 8.3 to show how to apply it to arbitrary trees.

8.1 Pricing Function

Height, as well, as weight, can be used as a measure of data completeness: two trees of same weight could be priced differently if they have different height. We

Algorithm 5: Algorithm for the sampling problem on binary trees

Input: a binary tree t and two integers $k \geqslant 0$ and $h \geqslant 0$
Result: an r-subtree t' of t of $\mathsf{weight}(t') = k$ and $\mathsf{height}(t') = h$ uniformly at random

```
// Phase 1: count the number of subtrees
```
$D \leftarrow \mathsf{SubtreeCounting}(t)$;
```
// Phase 2: sample a random subtree
```
if $k \leqslant \mathsf{weight}(t) \wedge h \leqslant \mathsf{height}(t) \wedge D_{\mathsf{root}(t)}[k,h] \neq 0$ **then**
  return $\mathsf{UniformSampling}(\mathsf{root}(t), D, k, h)$;
else
  fail;

are thus in a setting where a pricing function $\varphi_t(k,h)$ for the tree t should take into account both the weight k and the height h of an r-subtree. A healthy data market should impose at least the following conditions on φ_t:

Non-decreasing for weight. $k_1 \geqslant k_2 \Rightarrow \varphi_t(k_1,h) \geqslant \varphi_t(k_2,h)$.
Non-decreasing for height. $h_1 \geqslant h_2 \Rightarrow \varphi_t(k,h_1) \geqslant \varphi_t(k,h_2)$.
Arbitrage-free for weight. $\varphi_t(k_1,h) + \varphi_t(k_2,h) \geqslant \varphi_t(k_1+k_2,h)$.
Arbitrage-free for both. $\varphi_t(k_1,h_1) + \varphi_t(k_2,h_2) \geqslant \varphi_t(k_1+k_2,h_1+h_2)$.

An example pricing function is $\varphi_t(k,h) \coloneqq \alpha h + \beta\frac{k}{k+1}$, where $\alpha \geqslant 1 \geqslant \beta \geqslant 0$.

Similarly as in Section 3, once such a pricing function is fixed, the problem becomes to sample a tree of prescribed weight and height uniformly at random.

8.2 Sampling for Binary Trees

In this section, we provide an algorithm which proves the following theorem for binary trees:

Theorem 4. *The sampling problem for binary trees is solvable in time $O(nk^2h^2)$, where n is the number of nodes in the tree, k is the desired weight value, and h is the desired height.*

The sampling algorithm is adapted from the one in Section 5.1, and presented as Algorithm 5. The detailed adaptation of the two phases in the sampling algorithm is discussed in the following.

Modifications in Phase 1. As in Algorithm 2 (where $D_i[x]$ stores the number of r-subtrees of weight x rooted at node n_i), we need to record not only the weight but also the height of such r-subtrees. More precisely, we use $D_i[x,y]$ to denote the number of r-subtrees of weight x and height y rooted at node n_i, where $x \in [0, \mathsf{weight}(t)]$ and $y \in [0, \mathsf{height}(t)]$. We present Algorithm 6, to compute $D_i[x,y]$ for each node n_i in the given tree t.

Algorithm 6: SubtreeCounting(t)

Input: a binary tree t
Result: a matrix D such that $D_i[x, y]$ is the number of r-subtrees of weight x and height y rooted at n_i for all n_i, x and y

```
1  for x ∈ [0, weight(t)], y ∈ [0, height(t)], n_i ∈ V(t) ⊔ {NULL} do
2  |  D_i[x, y] ← 0;
3  D_NULL[0, 0] ← 1;
   // We browse all nodes in topological order, from leaves to the root
4  foreach non-NULL node n_i accessed bottom-up do
5  |  D_i[0, 0] = 1;
6  |  n_l ← first child of n_i;
7  |  n_r ← second child of n_i;
8  |  T ← D_l * D_r;
9  |  for x ∈ [w(n_i), weight(n_i)] do
10 |  |  for y ∈ [1, height(n_i)] do
11 |  |  |  D_i[x, y] ← T[x − w(n_i), y − 1];
12 return D;
```

As the base case, for NULL nodes, $\mathsf{weight}(\mathsf{NULL}) = 0$ and $\mathsf{height}(\mathsf{NULL}) = 0$. Hence $D_{\mathsf{NULL}}[0, 0] = 1$ (as shown in line 3). For other cases (i.e., $x > 0$ or $y > 0$), $D_{\mathsf{NULL}} = 0$.

For a non-NULL node, if the height is 0, then either the weight is also 0 and only the empty tree is possible, or the weight is greater than 0 and there is no possibility. Otherwise, intuitively, an r-subtree of weight $x \geqslant 0$ and height $y > 0$ rooted at node n_i is obtained by retaining n_i and choosing two r-subtrees t_l and t_r, respectively rooted at n_l and n_r (the children of n_i), such that $\mathsf{weight}(t_l) + \mathsf{weight}(t_r) = x - \mathsf{w}(n_i)$ and $\max\{\mathsf{height}(t_l), \mathsf{height}(t_r)\} = y - 1$ (which accounts for the weight and height of the additional node n_i). Similar to Algorithm 2, the number of such choices is computed in line 8 as the convolution of D_l and D_r in a certain sense, defined as follows, for $0 \leqslant p \leqslant \mathsf{weight}(t)$ and $0 \leqslant q \leqslant \mathsf{height}(t)$:

$$(D_l * D_r)[p, q] := \sum_{\substack{0 \leqslant h_1, h_2 \leqslant q \\ h_1 = q \text{ or } h_2 = q}} \sum_{m=0}^{p} D_l[m, h_1] \times D_r[p - m, h_2]$$

$(D_l * D_r)[p, q]$ represents the number of pairs of r-subtrees t_l and t_r such that $\mathsf{weight}(n_l) + \mathsf{weight}(n_r) = p$ and $\max\{\mathsf{height}(t_l), \mathsf{height}(t_r)\} = q$. In other words, there are three mutually exclusive ways to meet the requirement on the heights:

1. $\mathsf{height}(t_l) < q$ and $\mathsf{height}(t_r) = q$;
2. $\mathsf{height}(t_l) = q$ and $\mathsf{height}(t_r) < q$;
3. $\mathsf{height}(t_l) = q$ and $\mathsf{height}(t_r) = q$.

All in all, as shown in line 9 to line 11, the number of r-subtrees of weight x and height y, namely, $D_i[x, y]$, is the number of pairs of r-subtrees t_l and t_r

Algorithm 7: UniformSampling(n_i, D, x, y)

Input: a node n_i (or NULL), the precomputed D, a weight value x and a height value y

Result: an r-subtree of weight x and height y at node n_i if one exists

```
1  if y = 0 then
2      return ∅;
3  n_l ← first child of n_i;
4  n_r ← second child of n_i;
5  for 0 ⩽ s_l, s_r ⩽ x and 0 ⩽ o_l, o_r ⩽ y s.t. s_l + s_r = x − w(n_i) and
       max{o_l, o_r} = y − 1 do
6      p([s_l, o_l], [s_r, o_r]) ← D_l[s_l, o_l] × D_r[s_r, o_r];
7  Sample an ([s_l, o_l], [s_r, o_r]) with probability p([s_l, o_l], [s_r, o_r]) normalized by
       ∑_{([s_l,o_l],[s_r,o_r])} p([s_l, o_l], [s_r, o_r]);
8  L ← UniformSampling(n_l, D, s_l, o_l);
9  R ← UniformSampling(n_r, D, s_r, o_r);
10 return the tree rooted at n_i with child subtrees L and R;
```

rooted at n_l and n_r respectively such that $\mathsf{weight}(n_l) + \mathsf{weight}(n_r) = x - \mathsf{w}(n_i)$ and $\max\{\mathsf{height}(t_l), \mathsf{height}(t_r)\} = y - 1$ (which is $T[x - \mathsf{w}(n_i), y - 1]$ in line 11). Note that when $\mathsf{w}(n_i) \neq 0$ there exists no such r-subtrees at node n_i of weight x (where $x \in [0, \mathsf{w}(n_i) - 1]$) and height $y \neq 0$, so $D_i[x, y]$ remains 0 in this case.

The time complexity to compute D is $O(nk^2h^2)$. To sample an r-subtree of weight k and height h, we need to record the number of r-subtrees of weight up to k and height up to h rooted at every node. Therefore each array in D is a $k \times h$ array. Computing the convolution sum of such two arrays takes $O(k^2h^2)$ time, since computing each value in the convolution sum takes $O(kh)$ time. The number of convolution sums to compute overall is $O(n)$, because each array D_i occurs in exactly one convolution sum. The overall running time is $O(nk^2h^2)$.

Modifications in Phase 2. Similarly to Algorithm 3, we present Algorithm 7 to sample an r-subtree of weight x and height y at node n_i uniformly at random, given the computed matrix D in the previous section. If $y = 0$, the only possible r-subtree is the empty tree, therefore that is the output (line 1 to line 2). Note that the result will be incorrect if $x > 0$, but in this case Algorithm 5 would not have called Algorithm 7 because there is no such subtree to sample.

Except for this special case, we return n_i and subtrees t_l and t_r rooted at the children n_l and n_r of n_i. We first decide on the weight (respectively, height) s_l (respectively, o_l) and s_r (respectively, o_r) of t_l and t_r (line 5 to line 7) before sampling recursively a subtree of the prescribed weight and the prescribed height (line 8 to line 9), uniformly at random, and returning it.

The possible weight and height pairs $([s_l, o_l], [s_r, o_r])$ must satisfy the following conditions to be possible choices for the weights and the heights of the subtrees t_l and t_r:

1. $0 \leqslant s_l, s_r \leqslant x$ and $0 \leqslant o_l, o_r \leqslant y$;
2. $s_l + s_r = x - \mathsf{w}(n_i)$ and $\max\{o_l, o_r\} = y - 1$ (which accounts for node n_i).

Intuitively, to perform a uniform sampling, we now observe that the choice of the weight and height pair $([s_l, o_l], [s_r, o_r])$ partitions the set of outcomes. Hence, the probability that we select one weight and height pair should be proportional to the number of possible outcomes for this pair, namely, the number of r-subtrees t_l and t_r such that $\mathsf{weight}(t_l) = s_l$, $\mathsf{weight}(t_r) = s_r$ and $\mathsf{height}(t_l) = o_l$, $\mathsf{height}(t_r) = o_r$. We compute this from D_l and D_r (line 6) by observing that the number of pairs (t_l, t_r) is the product of the number of choices for t_l and for t_r, as every combination of choices is possible.

The uniform sampling phase takes $O(nkh)$ time. On every node n_i of the binary tree t, the number of possibilities to consider is $O(kh)$ because every node has exactly two children, and for each possibility the number of operations performed is constant (assuming that drawing a number uniformly at random is constant-time). The overall running time is $O(nkh)$.

8.3 Sampling for Unranked Trees

In this section, we show that the algorithm of the previous section can be adapted so that it works on arbitrary unranked trees, not just binary trees. Similarly to Section 5.2, we transform an unranked tree to a binary tree whose nodes are either regular nodes or dummy nodes. A dummy node is a virtual node gathering a sequence of more than two nodes. Therefore a dummy node does not contribute any weight nor height to r-subtrees. After transforming an arbitrary unranked tree to the corresponding binary tree using Algorithm 4, we apply Algorithm 5 to solve the sampling problem, while making sure the dummy nodes are managed correctly. We explain how to adapt Algorithm 6 and Algorithm 7 to handle the dummy nodes.

In Algorithm 6, we have to define the computation of D_i for a dummy node n_i as $D_i \leftarrow D_l * D_r$ (as it must always be kept, and does not increase the weight nor the height of the r-subtree).

In Algorithm 7, some operations have to be distinguished between regular nodes and dummy nodes. In line 1 we add one more condition in the if clause: n_i is either NULL or a regular node (for dummy nodes, even if $x = 0$ and $y = 0$ we cannot return $\emptyset$ as we must keep dummy nodes). In line 5, if n_i is a dummy node, the condition for possible weight and height pairs is: $s_l + s_r = x$ and $\max\{o_l, o_r\} = y$, because a dummy node does not contribute any weight nor height.

These adaptations do not affect the complexity of the sampling algorithm, therefore the algorithm for sampling unranked trees on both weight and height is also $O(nk^2h^2)$.

8.4 Tractable Cases

As presented in the previous section, the complexity of Algorithm 5 (and its variant in Section 8.3) is $O(nk^2h^2)$, where n is the number of nodes in the given

tree, k is the desired weight and h is the desired height. As $h \leqslant n$, the time complexity of Algorithm 5 is $O(n^3k^2)$. It is a pseudo-polynomial time algorithm to solve the sampling problem on both weight and height, which is polynomial in the *value* of k, but is still exponential in the *size* of k.

The tractable cases in Section 4.2 are still tractable when we sample on both weight and height. To solve such tractable cases, Algorithm 5 (and its variant in Section 8.3) runs in polynomial-time to the size of the tree, more specifically, $O(n^5)$, because $k \leqslant n$ when $\mathsf{w}(n_i) = 1$ or $\mathsf{w}(n_i) \in \{0, 1\}$. Hence:

Theorem 5. *The unweighted sampling and 0/1-weights sampling on both weight and height can be solved in $O(n^5)$ time, where n is the number of nodes in the given tree.*

9 Conclusion

We proposed a framework for a data market in which data quality can be traded for a discount. We studied the case of XML documents with completeness as the quality dimension. Namely, a data provider offers an XML document, and sets both the price and weights of nodes of the document. The data consumer proposes a price but may get only a sample if the proposed price is lower than that of the entire document. A sample is a rooted subtree of prescribed weight, as determined by the proposed price, sampled uniformly at random.

We proved that if nodes in the XML document have arbitrary non-negative weights, the sampling problem is intractable. We devise a pseudo-polynomial time algorithm to solve this general sampling problem, and proved the time complexity and correctness of the algorithm. We identified tractable cases, namely the unweighted sampling problem and 0/1-weights sampling problem, for which the pseudo-polynomial time algorithm actually runs in polynomial time. We also considered repeated requests and provided PTIME solutions to the unweighted cases.

As a more complicated sampling scenario, we studied the problem of uniform sampling an r-subtree of prescribed weight and height. We devised a pseudo-polynomial time sampling algorithm, and showed that it still runs in polynomial time in the tractable cases.

The more general issue that we are currently investigating is that of sampling rooted subtrees uniformly at random under more expressive conditions than size restrictions or 0/1-weights (with or without height). In particular, we intend to identify the tractability boundary to describe the class of tree statistics for which it is possible to sample r-subtrees in PTIME under a uniform distribution.

Acknowledgment. This work is supported by the French Ministry of Foreign Affairs under the STIC-Asia program, CCIPX project.

References

1. S. Cohen, B. Kimelfeld, and Y. Sagiv. Running tree automata on probabilistic XML. In *PODS*, 2009.

2. M. R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najork. On near-uniform URL sampling. *Computer Networks*, 33(1-6), 2000.
3. C. Hübler, H.-P. Kriegel, K. Borgwardt, and Z. Ghahramani. Metropolis algorithms for representative subgraph sampling. In *ICDM*, 2008.
4. P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Query-based data pricing. In *PODS*, 2012.
5. P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. QueryMarket demonstration: Pricing for online data markets. *PVLDB*, 5(12), 2012.
6. P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Toward practical query pricing with QueryMarket. In *SIGMOD*, 2013.
7. J. Leskovec and C. Faloutsos. Sampling from large graphs. In *SIGKDD*, 2006.
8. C. Li, D. Y. Li, G. Miklau, and D. Suciu. A theory of pricing private data. In *ICDT*, 2013.
9. C. Li and G. Miklau. Pricing aggregate queries in a data marketplace. In *WebDB*, 2012.
10. B.-R. Lin and D. Kifer. On arbitrage-free pricing for general data queries. *PVLDB*, 7(9), 2014.
11. X. Lu and S. Bressan. Sampling connected induced subgraphs uniformly at random. In *SSDBM*, 2012.
12. C. Luo, Z. Jiang, W.-C. Hou, F. Yu, and Q. Zhu. A sampling approach for XML query selectivity estimation. In *EDBT*, 2009.
13. A. S. Maiya and T. Y. Berger-Wolf. Sampling community structure. In *WWW*, 2010.
14. A. Muschalle, F. Stahl, A. Loser, and G. Vossen. Pricing approaches for data markets. In *BIRTE*, 2012.
15. L. Pipino, Y. W. Lee, and R. Y. Wang. Data quality assessment. *Commun. ACM*, 45(4), 2002.
16. B. F. Ribeiro and D. F. Towsley. Estimating and sampling graphs with multidimensional random walks. In *Internet Measurement Conference*, 2010.
17. R. Tang, A. Amarilli, P. Senellart, and S. Bressan. Get a sample for a discount: Sampling-based XML data pricing. In *DEXA*, 2014.
18. R. Tang, D. Shao, S. Bressan, and P. Valduriez. What you pay for is what you get. In *DEXA (2)*, 2013.
19. R. Y. Wang and D. M. Strong. Beyond accuracy: What data quality means to data consumers. *J. of Management Information Systems*, 12(4), 1996.
20. W. Wang, H. Jiang, H. Lu, and J. X. Yu. Containment join size estimation: Models and methods. In *SIGMOD*, 2003.