Independent Range Sampling, Revisited*

Peyman Afshani[1] and Zhewei Wei[†2]

1 **MADALGO[‡], Department of Computer Science, Aarhus University, Aarhus, Denmark**
`peyman@cs.au.dk`

2 **School of Information, Renmin University of China, Beijing, China**
`zhewei@ruc.edu.cn`

Abstract

In the independent range sampling (IRS) problem, given an input set P of n points in $\mathbb{R}^d$, the task is to build a data structure, such that given a range R and an integer $t \geq 1$, it returns t points that are uniformly and independently drawn from $P \cap R$. The samples must satisfy *inter-query independence*, that is, the samples returned by every query must be independent of the samples returned by all the previous queries. This problem was first tackled by Hu *et al.* [15], who proposed optimal structures for one-dimensional dynamic IRS problem in internal memory and one-dimensional static IRS problem in external memory.

In this paper, we study two natural extensions of the independent range sampling problem. In the first extension, we consider the static IRS problem in two and three dimensions in internal memory. We obtain data structures with optimal space-query tradeoffs for 3D halfspace, 3D dominance, and 2D three-sided queries. The second extension considers weighted IRS problem. Each point is associated with a real-valued weight, and given a query range R, a sample is drawn independently such that each point in $P \cap R$ is selected with probability proportional to its weight. Walker's alias method is a classic solution to this problem when no query range is specified. We obtain optimal data structure for one dimensional weighted range sampling problem, thereby extending the alias method to allow range queries.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems—Geometrical problems and computations.

Keywords and phrases data structures, range searching, range sampling, random sampling

Digital Object Identifier 10.4230/LIPIcs.ESA.2017.<1>

1 Introduction

Range searching is a fundamental problems in computational geometry.. The input is a set P of n data points in d-dimensional real space, $\mathbb{R}^d$ (possibly weighted). The goal is to preprocess the points into a data structure, s.t., given a query range R, the points in $P \cap R$ can be counted or reported efficiently. Range searching has been studied extensively and we refer the reader to the survey by Agarwal and Erickson [4] for a broad overview of the area.

Sampling is one of the most natural operations to deal with large data, making efficient and robust sampling vital in many applications. Here, we consider the range sampling problem, where the goal is to design a data structure to support efficient methods to sample

* This work was partly supported by the Partially supported by the National Natural Science Foundation of China (NSFC. 61502503).
† Corresponding author.
‡ A center of Danish National Research Foundation.

from data in the query range. These queries do not fit in the traditional range searching frameworks (such as, the semi-group range searching framework). However, the ability to generate random samples for a given range is useful in many database applications, such as online aggregation [14], interactive queries [5] and query optimization [9]. Within the context of database systems, the importance of sampling queries were identified early on. Olken and Roten's survey from 1995 [20] presents various possible sampling strategies as well as attempts to solve them (see also [19]). However, for spatial queries, i.e., range queries, most of the existing solutions have shortcomings. In one category of solutions, the idea is to use R-trees or Quadtrees where the performance of the data structures depend on input parameters such as "density" and "coverage" [19]. Thus, the worst-case performance of such solutions could be very bad. In another category of solutions, one can select a random sample of the points, preprocess and store them in a data structure (see [15] for more details) but this does not guarantee independence between future and past queries (i.e., asking the same query twice will return the same set of samples).

Hu *et al.* [15] studied the *independent range sampling* problem for the first time using the worst-case analysis. In this variant, it is required that the results of every query must be independent from those returned by the previous queries. Thus, issuing the same query multiple times will fetch different samples, which is desirable in many data analytic applications [11, 27, 17], For example, in *interactive spatial exploration and analytics* [11, 27], the user specifies a query range on the map, and the goal is to continuously generating samples from that range for analytic purpose. The query process is interactive since the user can terminate the query whenever s/he finds the precision of the analysis is acceptable. Independence among the sampling results of all queries is crucial in interactive spatial exploration and analytics, since the user may issue queries with similar query ranges and expect to get independent estimations. Hu *et al.* [15] studied the problem in one dimension for unweighted points, and proposed a data structure that consumes $O(n)$ space, can be updated in $O(\log n)$ time and can answer queries in $O(\log n + t)$ time, where t is the number of samples. In this paper, we study the problem in two and three dimensions, and obtain optimal data structures for some important categories of queries: three dimensional halfspaces and by extension, three-dimensional dominance queries, and two-dimensional three-sided queries. We also propose optimal data structure for one-dimensional *weighted* range sampling problem, in which the sampling probability is defined by the weights of the points.

We focus on the space-query time trade-off for *static* data structures that solve the independent range sampling problem. We focus on the *with-replacement* sampling, in which each sample is independent selected from the query range. We defer the discussion on *without-replacement* sampling to the appendix. For the unweighted case, the input is a set $P = \{p_1, \cdots, p_n\}$ of n points in R^d where U is the domain size, and a range space $\mathcal{R}$. Given a range $R \in \mathcal{R}$ and an integer $t \geq 1$, the query returns a *sequence* of t points, where each element of the sequence is a random point of $P \cap R$ that is sampled uniformly and independently (i.e., with probability $\frac{1}{|P \cap R|}$). We impose the constraint that the sampling result must be independent from those returned by the previous queries.

For the weighted case, each input point p_i is associated with a real-valued weight w_i. The query returns a sequence of t points, where each element of the sequence is a point $p_i \in P \cap R$ sampled independently and with probability $w_k / \sum_{p_j \in P \cap R} w_j$. Note that if range R is omitted in each query, such sampling oracle can be implemented with a classic data structure called *Walker's alias method*, which uses linear space and returns a weighted sample in constant time. Alias method has been successfully adapted in many data mining algorithms [16, 6], so it is also of theoretical interest to see if it can be extended to support range queries. We assume the existence of an oracle that can generate random real numbers or integers in $O(1)$ time. We assume *real RAM model of computation*: a machine that is equipped with w-bit integer registers, for $w = \Omega(\log n)$, as well as real-valued registers that can store any real number. Arithmetic operations take constant time but storing the contents of a real-valued register into an integer register (via the "floor" function) is only allowed when the result has at most w bits [1].

Our results. We obtain an optimal data structure for three dimensional halfspace ranges for the unweighted independent range sampling problem. Given a query halfspace h, it can extract t independent uniform random samples from h in $O(\log n + t)$ expected time. The structure uses $O(n)$ space. This also implies optimal data structures for two-sided and three-dimensional dominance queries. For weighted range sampling problem, we obtain an optimal data structure for one dimensional point sets in the real RAM model. More precisely, we assume the coordinate of each point can fit in a word of $\Theta(\log U)$ bits, and the real value weights are stored in real registers. The reason we make this assumption is to assure that the space used to store the weights cannot be charged to the space used to store the points. The query is given as an interval $[a, b]$, where a and b are indices. The goal is to extract t independent samples from the indices in $[a, b]$, such that each index $i \in [a, b]$ is selected independently with probability proportional to its weight w_i. Our solution uses $O(n)$ space and answers a query in $O(\text{Pred}(U, w, n) + t)$ time, where $\text{Pred}(U, w, n)$ is the query time of a predecessor search data structure that uses $O(n)$ space on an input of size n from the universe $[U]$ and on a machine with w-bit integers [22].

1.1 Related Work

In the database community, the problem has a long history and it dates back to the 80's when it was introduced as the *random sampling queries* problem. For a database and a given query (range, relational operator, etc.), the goal is to return a random sample set in the query result rather than the entire query result itself. Olken and Rotem considered the problem of independently returning random samples from a query range on B-trees [20], and obtained a structure that returns a sample with $O(\log_B n)$ cost. Olken and Rotem also studied the range sampling problem in high dimensional space using R-tree based structures [21]. We refer the readers to see an excellent survey in [20].

This problem has regained attention recently, due to the "big query" phenomenon where a query result may contain a huge number of elements, and thus it is infeasible to list them all. As mentioned, Hu *et al.* [15] studied the range sampling problem for one dimensional points, with insertions and deletions. They proposed a dynamic RAM structure of $O(n)$ space that answers a range sampling query in $O(\log n + t)$ expected time, and supports an update in $O(\log n)$ time. The static *unweighted* range sampling problem is trivial for one dimensional point sets, since given a query with range $R = [a, b]$ and parameter t, one can perform two predecessor queries to identify the boundaries of the points in R, and one range counting query with constant cost to obtain $P \cap R$, the number of points in R. Then, we can simply sample from $P \cap R$ by generating t random integers between 1 and $|P \cap R|$ and accessing the corresponding t points.

Walker's Alias Method. In the weighted sampling problem, the input is a set of non-

[1] We actually don't need a "floor" instruction since we can simulate it using binary search in $O(w)$ time. As this is used during the preprocessing phase, the query cost can still be kept constant.

negative real numbers $w_1, \ldots, w_n$, and the goal is to build a data structure, such that a query extracts an index i with probability $p_i = w_i / \sum_{j=1}^{n} w_j$. The indices returned by different queries should be independent. The classic solution to this problem is Walkers' alias method [26], which uses $O(1)$ query time and $O(n)$ preprocessing time. See Appendix 5.1 for short description of this method.

Another problem very related to halfspace range sampling is the halfspace range reporting problem where the goal is to simply report all the points in the query halfspace. To see this relationship, observe that extracting t random samples from a query range that contains only t points should extract a faction of the points in the range with constant probability. Halfspace range reporting has been extensively studied for over 30 years, and various results were obtained on this problem. In 2D, the problem was optimally solved in 1985 by Chazelle, Guibas, and Lee [10] but in contrast, the first optimal solution in 3D was obtained relatively recently in 2009 by Afshani and Chan [2], where they showed that one can report the set of points in a query halfspace in $O(\log n + t)$ time using $O(n)$ space where t is the output size. The 3D solution is based on powerful tools created by Matoušek [24, 18] which have been a vital part of all the previous attempts to solve halfspace range reporting problems in three and higher dimensions [7, 24, 23]. Note that the fact that we can match the performance of the best reporting data structures for the queries considered is very desirable.

2 Unweighted Range Sampling in Three Dimensions

Let $p_1, \cdots, p_n$ be a set of three-dimensional points. The main result of this section is an optimal data structure that given a query halfspace h, it can extract t independent uniform random samples from h in $O(\log n + t)$ expected time.

We will use most of the known tools in range reporting: shallow cutting, shallow partition, and partition theorems together with new ideas that take advantage of the structure of the range sampling queries. The rough summary of our approach is as follows: we first build a "core" data structure to sample from query halfspaces that contain many points; later by using shallow cuttings we can extend this to all query halfspaces. To build the core data structure, we create a hierarchy of shallow cuttings and then for each cell in the resulting cuttings, we build data structures that can sample a point uniformly randomly from inside the cell. This part is the main technical contribution since without additional ideas, this approach is not going to give us a linear-space solution[2]. To use only linear space, we build one "global" data structure which is an array that stores the point set in some order, and then for each cell in a shallow cutting, we store a data structure of *sublinear* size that can be used to generate one "random" entry point per sample, into the global array; the final sample point is obtained through this random entry point in *constant time*. A careful analysis shows that the space complexity of the data structure is indeed linear and that each sample is picked with the correct probability.

2.1 Preliminaries and Definitions

We present the dual of a hyperplane h (resp. point p) with $\overline{h}$ (resp. $\overline{p}$): a point p that is below a hyperplane h is mapped to a hyperplane $\overline{p}$ that passes below the point $\overline{h}$. An xy-monotone

[2] An expert reader can verify that this approach can easily give us a solution that uses $O(n \log n)$ space, if we spend $O(n)$ space per each shallow cutting level. By using another classical idea, that is, building the shallow cuttings every $\log \log n$ levels and bootstrapping using simplex range searching data structures, this can be reduced to $O(n \log \log n)$. However, this approach seems hopeless to get to $O(n)$ space.

function in $\mathbb{R}^3$ is a surface, s.t., any line parallel to the z-axis intersects the surface exactly once. Given an xy-monotone surface $\mathcal{F}$ in 3D and a point $q = (q_x, q_y, q_z) \in \mathbb{R}^3$, we say q is *above* $\mathcal{F}$ iff the "downward" ray $(q_x, q_y) \times [q_z, -\infty)$ intersects $\mathcal{F}$. The *below* relationship is defined analogously. Let P be a set of points in 3D and let H be a set of n hyperplanes dual to points in P. A *k-shallow cutting* $\mathcal{F}$ for H is an xy-monotone surface that is a piece-wise linear function composed of $O(n/k)$ vertices, edges, and triangles, s.t., there are at least k and at most $O(k)$ hyperplanes passing below every point of $\mathcal{F}$. The *conflict list* of a point p on $\mathcal{F}$ is defined as the set of all the hyperplanes in H that pass below p and it is denoted by H_p. The conflict list of a triangle $\tau \in \Delta(\mathcal{F})$ is the set of hyperplanes that pass below τ and is denoted by H_τ. The set of points dual to H_τ is denoted by P_τ. See Figure 1.

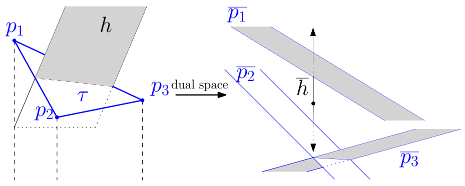

Figure 1 (left) For a k-shallow cutting $\mathcal{F}$, a triangle $\tau \in \Delta(\mathcal{F})$ is shown. For every point on τ, there are at least k and at most $O(k)$ hyperplanes passing below it (not shown here). H_τ is the set of hyperplanes h that are below at least one of the vertices of τ. (right) In dual space, $\overline{h}$ is a point that is below at least one of the hyperplanes corresponding to the vertices of τ.

▶ **Theorem 1** (Shallow Cutting Theorem). *[24] For any given set H of n hyperplanes in 3D and an integer $1 \le k < n/2$, k-shallow cuttings exist. Furthermore, for $k_i = 2^i$, $0 \le i < \log n$, k_i-shallow cuttings $\mathcal{F}_i$, together with the conflict lists of all their vertices, can be constructed in $O(n \log n)$ total time.*

▶ **Lemma 2.** *Given a shallow cutting $\mathcal{F}_i$ and its set $\Delta(\mathcal{F}_i)$ of $O(n/k_i)$ triangles, we can build a data structure of size $O(n/k_i)$ s.t., given a point $p \in \mathbb{R}^3$, we can decide if p is below $\mathcal{F}_i$ or not. In the first case, the triangle $\tau \in \Delta(\mathcal{F}_i)$ that lies directly above p can be found in $O(\log n)$ time.*

Proof. Simply project all the triangles onto the xy-plane. Since $\mathcal{F}_i$ is xy-monotone, we obtain a decomposition of the plane into $O(n/k_i)$ triangles. Build a point location data structure on the planar decomposition [12]. Given the query point p, project it onto the xy-plane, find the triangle τ whose projection contains the projection of p, and decide if p is below τ or not. ◀

▶ Theorem 1 (Partition Theorem). [18] Given a set P of n points in 3D and an integer $0 < r \le n/2$, there exists a partition of P into r subsets $P_1, \cdots, P_r$, each of size $\Theta(n/r)$, s.t., each subset P_i is enclosed by a tetrahedron T_i, s.t., any hyperplane crosses $O(r^{2/3})$ tetrahedra.

▶ Theorem 2 (Shallow Partition Theorem). [24] Given a set P of n points in 3D and an integer $0 < r \le n/2$, there exists a partition of P into r subsets $P_1, \cdots, P_r$, each of size $\Theta(n/r)$, s.t., each subset P_i is enclosed by a tetrahedron T_i, s.t., any halfspace that has at most n/r points of P crosses $O(\log r)$ tetrahedra.

We also need the following known optimal data structures for halfspace range reporting (Theorem 3) and approximate halfspace range counting (Theorem 4).

▶ **Theorem 3.** *[2] Given a set P of n points in $\mathbb{R}^3$, one can build a data structure of linear size s.t., given a query halfspace h, it can list the points in $P \cap h$ in $O(\log n + |P \cap h|)$ time.*

► **Theorem 4.** *[3, 1] Given a set P of n points in $\mathbb{R}^3$, and a constant $\varepsilon > 0$, one can build a data structure of linear size s.t., given a halfspace h, in $O(\log n)$ time, one can produce an integer $\tilde{k}$ s.t., $\tilde{k}/(1+\varepsilon) \leq |h \cap P| \leq \tilde{k}$.*

2.2 The Overall Data Structure

We now return to our original problem. Our input is a set P of n points in $\mathbb{R}^3$. Let H be the set of hyperplanes dual to P. Define $k_i = 2^i, 0 \leq i < \log n$. We say an integer m is *large* if it is greater than $2^{C(\log\log n)^2}$, for a global constant C to be set later. The following lemma will be proved in the next subsection.

► **Lemma 5.** *Given a set P of n points, we can build a structure of linear size to answer the following queries. Given any query halfspace h in which $|P \cap h|$ is large, we can extract t independent random samples from $P \cap h$ in $O(\log n + t)$ time.*

Furthermore, the query can be carried over in an "online" fashion. After the initial search time of $O(\log n)$, the data structure can fetch each subsequent sample in $O(1)$ expected time, until interrupted by the user.

By standard techniques, this gives us our main theorem. See Appendix 5.2 for the proof.

► **Theorem 6.** *Given a set P of n points in $\mathbb{R}^3$, we can build a data structure of size $O(n)$ s.t., given a halfspace h and a parameter t, we can extract t samples from the subset $P \cap h$ in $O(\log n + t)$ expected time.*

Furthermore, the query can be carried over in an "online" fashion, without knowledge of t: After the initial search time of $O(\log n)$, the data structure can fetch each subsequent sample in $O(1)$ expected time, until interrupted by the user.

The above theorem easily extends to sampling from dominance queries as well. Given two points p and q in d-dimensional space, q *dominates* p if every coordinate of q is greater than that of p. In dominance reporting, the goal is to preprocess a set of n points s.t., given a query point q all the points dominated by q can be reported efficiently. As observed by Chan *et al.* [8], three-dimensional dominance queries can be solved using halfspace queries. It is also known that a dominance query can solve two-dimensional a three-sided query, that is, a query region $[a, b] \times (-\infty, c]$ given by three values a, b, and c.

2.3 Proof of Lemma 5

As previously mentioned, this is the heart of the problem and this is where we significantly deviate from the previous techniques (even though we use similar building blocks): To obtain optimal halfspace range reporting, Afshani and Chan [2] rely heavily on the fact that if a halfspace h contains too many points, then the data structure is allowed to spend a lot of time on the query, since we will spend a lot of time producing the output; in other words, the search time can be charged to the output size. In our case, we might be interested only in a small subset of points in h and thus the search cost cannot be charged to the output size.

Our idea is to build two main components: a global array and a number of local structures. The global array will store each point once in an array of size n, in some carefully selected order. The array compactly stores a number of "canonical sets" of total size $O(n \log n)$. We use shallow cuttings to build the local structures. The important point is that the local structures in total will have *sublinear* size and their utility is to find entry points into the global structure: given a query, using the local structures we locate a subarray of the global array and then uniformly sample from the subarray. We present the technical details below.

Using Shallow Cutting Theorem, we build a k_i-shallow cutting $\mathcal{F}_i$ (as well as its set of triangles $\Delta(\mathcal{F}_i)$), for each large k_i where $k_i = 2^i, 0 \le i < \log n$, We have $|\Delta(\mathcal{F}_i)| = O(n/k_i)$ by the Shallow Cutting theorem. For a triangle $\tau \in \Delta(\mathcal{F}_i)$, the conflict lists H_τ and P_τ are defined as before. Observe that for each triangle $\tau \in \Delta(\mathcal{F}_i)$, with vertices p_1, p_2 and p_3, H_τ is the union of H_{p_1}, H_{p_2}, and H_{p_3} (with duplicates removed). In this subsection, h will refer to the query halfspace in the primal space. In dual space we will denote $\overline{h}$ with q. Thus, our objective is to either sample a random point of P below h, or a random member of H_q (a random hyperplane of H that passes below q).

The Global Structure. Using Shallow Partition Theorem, we build a partition tree T_{global} as follows. The root of T_{global} represents P. Consider a node of T_{global} that represents a subset $S \subseteq P$. We use Shallow Partition Theorem with parameter $r = |S|^\varepsilon$ to obtain subsets $S_1, \cdots, S_r$, for a small enough constant $\varepsilon > 0$. If a subset S_i contains at most b points, for a parameter b to be defined later, we call it a *base subset*. Unlike the approach in [2], we only recurse on subsets S_i that are not base subsets. Thus, the leaves of T_{global} are base subsets. For each base subset B, we build a secondary data structure that is another partition tree T_B: The root of T_B represents B. At a node of T_B that represents a subset $S \subseteq B$, we use Partition Theorem (*not* the shallow version) with parameter $r = |S|^\varepsilon$ to obtain subsets $S_1, \cdots, S_r$. We recurse on each subset S_i until we reach subproblems of constant size. We store an in-order traversal of the leaves of T_B, in an array A_B; the size of A_B is exactly equal to $|B|$ and for every internal node $v \in T_B$ the points in the subtree of v are mapped to a contiguous interval of array A_B. We build our global array A by concatenating all the arrays A_B over all base subsets B. Finally, we build a data structure for approximate range counting queries (Theorem 4).

The Local Structure for τ. Consider a triangle $\tau \in \Delta(\mathcal{F}_i)$ and let p_1, p_2, and p_3 be the vertices of τ. Remember that H_τ was defined as the union of conflict lists of p_1, p_2, and p_3 after duplicate removal (Figure 1). We will store a local structure for τ that consumes $o(|H_\tau|)$ space ($O(|H_\tau|/\log^{O(1)} n)$ to be more precise). p_1, p_2 and p_3 correspond to three different hyperplanes, $\overline{p_1}, \overline{p_2}$ and $\overline{p_3}$ in the primal space; a hyperplane $h \in H_\tau$ corresponds to a point $\overline{h} \in P$ that is below one of the hyperplanes $\overline{p_1}, \overline{p_2}$ or $\overline{p_3}$. Let P_τ be the set of such points (in other words, P_τ is the set of points dual to hyperplanes in H_τ). Let $B_1, \cdots, B_m$ be the base subsets that are intersected by or are below at least one hyperplane $\overline{p_i}, 1 \le i \le 3$. We will store a data structure of size $O(mb^{3/4})$ at triangle τ: For each base subset B_i, we consider the partition tree T_{B_i}. For every node $v \in T_{B_i}$, the subset of points in the subtree of v defines a canonical subset of B_i. By the properties of partition trees (see e.g., [18]), we can write $P_\tau \cap B_i$ as the union of $O(|B_i|^{2/3} \log |B_i|^{O(1)}) = O(|B_i|^{3/4}) = O(b^{3/4})$ canonical subsets of B_i. However, as each subtree of T_{B_i} maps to a contiguous interval of A_{B_i}, it follows that we can represent $P_\tau \cap B_i$ as the union of $O(|B_i|^{3/4})$ intervals from A_{B_i}. We collect all these intervals, over all base subsets $B_1, \cdots, B_m$. Let $I_1, \cdots, I_M$ be the set of all such intervals. Observe that we have $|I_1| + |I_2| + \cdots + |I_M| = |P_\tau|$ since the intervals partition P_τ. Also, $M = O(mb^{3/4})$. We store the numbers $|I_1|, |I_2|, \cdots, |I_M|$ in a data structure $T_{\text{sample}}(\tau)$ for weighted sampling, using the Alias method; the data structure consumes $O(M) = O(mb^{3/4})$ space and in $O(1)$ time can produce a pointer to an interval I_j with probability $\frac{|I_j|}{|I_1|+|I_2|+\cdots+|I_M|} = \frac{|I_j|}{|P_\tau|}$.

Answering Queries. Using approximate halfspace range counting data structure, we can produce an integer $\tilde{k}$ s.t., $\tilde{k}/2 \le k \le \tilde{k}$. Let i be the smallest index s.t., $\tilde{k} \le k_i$. We can find $\tilde{k}$ and i in $O(\log n)$ time, by Theorem 4. Clearly, q is below k_i-shallow cutting $\mathcal{F}_i$. Let $\tau \in \Delta(\mathcal{F}_i)$ be the triangle that lies above the query point q. τ can be found in $O(\log n)$ time using a point location query. We claim it is sufficient to be able to sample from H_τ: to sample a hyperplane that passes below q, we repeat extracting independent uniform samples from the set H_τ until we find one that passes below q. Since H_q is a subset of H_τ, this guarantees independent uniform sampling. On the other hand, since $|H_q| \geq \tilde{k}/2$ we get that $|H_\tau| = O(k)$, and thus on average we need to extract $O(t)$ random samples from H_τ to produce t random samples from H_q[3]. Note that after initial $O(\log n)$ time to find τ, we spend $O(1)$ expected time per sample and thus we can continue without knowledge of t.

Sampling from H_τ. Consider the intervals $I_1, \cdots, I_M$ stored for the triangle τ; by construction, the points stored in these intervals form a partition of P_τ. Using structure $T_{\text{sample}}(\tau)$, in $O(1)$ time, we can select an interval I_j with probability $\frac{|I_j|}{|P_\tau|}$. Next, we generate a random integer ℓ between 1 and $|I_j|$ and output the ℓ-th point in the interval I_j. Clearly, the probability of outputting an element of P_τ is exactly equal to $\frac{1}{|P_\tau|}$ and query time is $O(1)$ per sample.

Space Analysis. This is the main part of the proof. First, observe that the global structure clearly consumes linear space since points in every base subset B are stored only once in the array A_B. Thus it remains to bound the space usage of the local structures. Consider a triangle $\tau \in \Delta(\mathcal{F}_i)$ with its three vertices p_1, p_2, and p_3. Let H_{p_1} be the set of hyperplanes of H below p_1, or equivalently, let $P_{\overline{p_1}}$ be the subset of points of P below the hyperplane $\overline{p_1}$. Let $k = |H_{p_1}|$. Let $f(n,k)$ be the maximum number of base subsets of T_{global} intersected by any hyperplane h that lies above k point of P. Remember that we have used the Shallow Partition theorem with parameter $r = n^\varepsilon$. If $n \leq b$, then we are already at a base subset so $f(b,k) = 1$. Otherwise, depending on the value of k, we might intersect either all the r subsets or only $O(\log r)$ subsets. So we get the following recurrence, which is a generalization of the one found in [2] (we must note that the recurrence in [2] bounds the query time where here we are only bounding the crossing number):

$$f(n,k) \leq \begin{cases} 1 & \text{if } n \leq b \\ \sum_{i=1}^{O(\log n)} f(cn^{1-\varepsilon}, k_i) & \text{if } k \leq n^{1-\varepsilon} = \frac{n}{r} \\ \sum_{i=1}^{n^\varepsilon} f(cn^{1-\varepsilon}, k_i) & \text{if } k > n^{1-\varepsilon} = \frac{n}{r} \end{cases} \quad \text{where } k = \sum_i k_i.$$

We solve the recurrence by guessing that it solves to the "correct" bound, that is, it solves to $f(n,k) = 2^{c(\log\log n)^2} + kg(n)/b^{1-3\varepsilon}$, where $g(n)$ is a monotonously increasing function that is always upper bounded by a constant and c is a constant. This is similar to the analysis done in [2], so we postpone the details to Appendix 5.3.

The analysis shows that the total number of the base sets intersected by three hyperplanes $\overline{p_1}, \overline{p_2}$, or $\overline{p_3}$ is at most $3f(n, O(k_i))$. Thus, the value m in the local structure of τ is bounded by $3f(n, O(k_i))$. The local structure of τ consumes $O(mb^{3/4})$ space. Remember that we are aiming to build the data structure for halfspace containing large number of points. Thus, $k_i = \Omega(2^{C(\log\log n)^2})$. We set the constant C in the definition of a large integer to $2c$, which means $k_i = \Omega(2^{2c(\log\log n)^2})$. We also set $b = \log^{C'} n$ for a large enough constant C'. We

[3] This is the only part that breaks down for the weighted range sampling problem. This is also the only hurdle that makes the query time "expected". All the other parts of the data structure work with a worst-case query time.

plug the values in $f(n,k)$, and thus space used by the local structure of τ is bounded by

$$O(mb^{\frac{3}{4}}) = O\left(f(n,O(k_i))b^{\frac{3}{4}}\right) = O\left(\left(2^{c(\log\log n)^2} + \frac{k_i}{b^{1-3\varepsilon}}\right)b^{\frac{3}{4}}\right)$$
$$= O\left((\log n)^{\frac{3C'}{4}}2^{c(\log\log n)^2} + \frac{k_i}{(\log n)^{(\frac{3C'}{4}-3\varepsilon)}}\right) = O\left(\frac{k_i}{\log^2 n}\right)$$

if we set C' large enough and set $\varepsilon < 1/4$. The number of triangles τ in $\Delta(\mathcal{F}_i)$ is $O(n/k_i)$ and thus the total amount of space used by the triangles is $O(n/\log^2 n)$ and over all the indices this sums up to $o(n)$. This proves all the local data structures consume sublinear space and concludes the proof of Lemma 5.

3 Weighted Range Sampling in One Dimensions

In this section, we address the one-dimensional range sampling problem. Let U be an integer that denotes the universe size of the coordinates. We assume the word size $w = \Theta(\log U)$, such that the coordinate of each point can fit in a word. The input is a set $P = \{p_1, \cdots, p_n\}$ of n points on grid $[u]$, and each point p_i is associated with a non-negative real-valued weight w_i. Given an interval $R = [a,b]$ and an integer $t \geq 1$, the query returns a sequence of t points, where each element of the sequence is random point $p_i \in P \cap R$ that is sampled independently and with probability $w_k / \sum_{p_j \in P\cap R} w_j$.

Note that the coordinate of a point can be stored in a word of $w = \Theta(\log U)$ bits, but a weight is a real number and cannot be stored in an *integer* word. We say the space usage of a data structure is $S(n)$ if it uses at most $S(n)$ words and at most $S(n)$ real registers.

Weighted v.s. Uniform IRS. We first offer some intuition to show that weighted independent range sampling is a non-trivial problem, even in one-dimension. Consider one-dimensional uniform independent range sampling problem. There is a simple solution: we store the points of P in ascending order using an array A. Given a query range $[a,b]$ and an integer t, we perform predecessor/successor search to identify the subsequence in A that consists of the elements covered by q. Then, we can simply sample from the subsequence by generating t random ranks and accessing t points. The total query cost is $O(\text{Pred}(U,w,n)+t)$ where $\text{Pred}(U,w,n)$ is the query time of a predecessor search data structure that uses $O(n)$ space on an input of size n from the universe $[U]$ and on a machine with w-bit integers [22]. For the weighted IRS problem, the above approach does not work. The main issue is that sampling from the identified subsequence requires an alias structure designed specifically to that subsequence. Since there are $\Omega(n^2)$ difference subsequences, one needs $\Omega(n^2)$ space to make this approach work.

Notations. We begin by defining some notations. Given a set S, we use $W(S)$ to denote its weight. With a slight abuse of notation, we also use $W(S)$ to denote the set S. Given two integers $1 \leq a \leq b \leq u$, $[a,b]$ is the range from a to b. With a slight abuse of notation, we will also use $[a,b]$ to denote the points in range $[a,b]$, and $W(a,b) = \sum_{p_k\in[a,b]} w_k$ to denote the total weights in $[a,b]$. We use $P_{pre}(a)$ to denote the predecessor of a in P, and $P_{suc}(a)$ to denote the successor of a in P. Given a point $p_i \in [a,b]$, we use $W_{pre}(p_i,a,b) = \sum_{p_j\in[a,b],j<i} w_j$ to denote the prefix sum of point p_i in $[a,b]$, and $W_{suc}(p_i,a,b) = \sum_{p_j\in[a,b],j>i} w_j$ to denote the suffix sum of point p_i in $[a,b]$, respectively.

Let $\mathcal{T}$ denote a balanced binary tree on the n points, with height $h = \log n$. Given an internal node u, we use $W(u)$ to denote the total weight of the subtree rooted by u. Fixing an internal node u and a leaf v in u's subtree, let $\mathcal{P}(u,v)$ be the set of nodes on

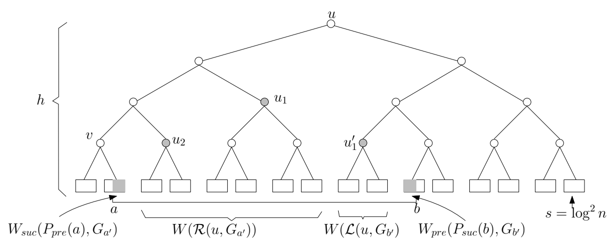
u
h
u_1
v
u_2
u'_1
a
b
$s = \log^2 n$
$W_{suc}(P_{pre}(a), G_{a'})$
$W(\mathcal{R}(u, G_{a'}))$
$W(\mathcal{L}(u, G_{b'})$
$W_{pre}(P_{suc}(b), G_{b'})$

Figure 2 A schematic illustration of the fat points and partial sums.

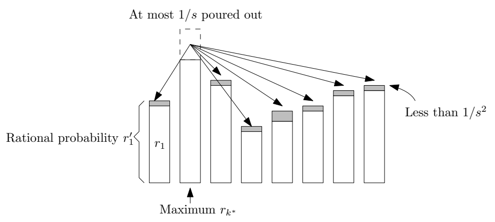

Figure 3 A schematic illustration of the rounding process

the path from u to v, excluding node u. We define the *left canonical set* of $\mathcal{P}(u, v)$ to be $\mathcal{L}(u, v) = \{w \in \mathcal{P}(u, v) \mid w \text{ is a left child}\} \cup \{v\}$, and similarly the *right canonical set* to be $\mathcal{R}(u, v) = \{w \in \mathcal{P}(u, v) \mid w \text{ is a right child}\} \cup \{v\}$. It is easy to see that the point set in range $[a, b]$ is made up by the subtrees rooted at the nodes in $\mathcal{R}(u, P_{pre}(a)) \cup \mathcal{L}(u, P_{suc}(b))$. Here we define $P_{pre}(a) = P_{suc}(a)$ if a is in S. See figure 4.

A baseline structure. We will use the following baseline structure, which uses $O(n \log^2 n)$ space draws sample with constant cost. The proof of Lemma 7 is deferred to the Appendix.

▶ **Lemma 7.** *For the one-dimensional weighted IRS problem, there is a structure of $O(n \log^2 n)$ space that can answer a weighted sampling query in $O(Pred(U, w, n) + t)$ time.*

3.1 A structure with linear space and $O(\log^* n)$ query cost.

In this subsection, we improve the space of our structure to linear by sacrificing the per-sample query cost.

Structure. We group the points into $m = n/s$ fat points, $G_1, \cdots, G_m$, where each fat point G_i includes $s = \log^2 n$ consecutive points. The weight of G_i is defined to be the summation of weights in G_i. Then we build the baseline structure, denoted by $\mathcal{T}$, on the fat points. Since the number of fat points is $n/s = n/\log^2 n$, the space usage is reduced to $O(n)$. Inside each fat point G_i, we bootstrap a baseline structure, denoted by $\mathcal{T}(G_i)$, for all s points contained in G_i. This takes $O(s \log^2 s) = O(\log^2 n \log^2 \log n)$ space for each fat point, and $O(n \log^2 \log n)$ space for all $n/\log n$ fat points. For each point $p_k \in G_i$, we also store $W_{pre}(p_k, G_i)$ and $W_{suc}(p_k, G_i)$, the prefix and suffix sums of point p_k in G_i, respectively. Finally, we store n global prefix sums, $W_{pre}(p_i, P)$, for $i = 1, \ldots, n$. It is easy to see the total space usage is $O(n \log^2 \log n)$.

Answering Queries. Given a query range $[a, b]$, we first find $P_{pre}(a)$, the predecessor of a and $P_{suc}(b)$, the successor of b in P, in $\text{Pred}(U, w, n)$ time. Then we locate the fat points $G_{a'}$ and $G_{b'}$ that contains $P_{pre}(a)$ and $P_{suc}(b)$, respectively. Figure 2 illustrates that $W(a, b)$ can be decomposed into the summation of partial weights in fat leaves $G_{a'}$ and $G_{b'}$, and weights of subtrees in canonical sets $\mathcal{R}(u, G_{a'})$ and $\mathcal{L}(u, G_{b'})$. More precisely, we have

$$W(a, b) = W_{suc}(P_{pre}(a), G_{a'}) + W_{pre}(P_{suc}(b), G_{b'}) + W(\mathcal{R}(u, G_{a'})) + W(\mathcal{L}(u, G_{b'})).$$

We retrieve these four weights and sample one of the weights. If $W(\mathcal{R}(u, G_{a'}))$ or $W(\mathcal{L}(u, G_{b'}))$ is selected, we sample a fat leaf G_i from $G_{a'}, \ldots, G_{b'}$ using baseline solution $\mathcal{T}$, and then sample a point p_k from G_i using the alias structure $\mathcal{A}(G_i)$. Otherwise, assume that the partial sum $W_{suc}(P_{pre}(a), G_{a'})$ is selected. We simply query the baseline structure in $\mathcal{T}(G_{a'})$ with range $[a, \infty)$ to retrieve a sample as the query result.

Analysis. To see that above sampling procedure gives the correct probability distribution, note that a point p_k in fat point $G_{a'}$ is selected if and only if the partial sum $W_{suc}(P_{pre}(a), G_{a'})$ is sampled from $W(a,b)$, and p_k is sampled from $W_{suc}(P_{pre}(a), G_{a'})$. Thus the probability is

$$\frac{w_k}{W_{suc}(P_{pre}(a), G_{a'})} \cdot \frac{W_{suc}(P_{pre}(a), G_{a'})}{W(a,b)} = \frac{w_k}{W(a,b)}.$$

On the other hand, consider a point p_k in fat point G_i that lies completely in (a,b). Without loss of generality, we assume G_i is in left canonical set $\mathcal{R}(u, G_{a'})$ of the baseline structure $\mathcal{T}$. Observe that p_k is selected if and only if the following events happen: 1. $W(\mathcal{R}(u, G_{a'}))$ is selected from $W(a,b)$; 2. $W(G_i)$ is selected from alias structure $\mathcal{A}(\mathcal{R}(u, G_{a'}))$; 3. p_k is selected from alias structure $\mathcal{A}(G_i)$. Thus the probability for p_k being picked can be computed as

$$\frac{W(\mathcal{R}(u, G_{a'}))}{W(a,b)} \cdot \frac{W(G_i)}{W(\mathcal{R}(u, G_{a'}))} \cdot \frac{w_k}{W(G_i)} = \frac{w_k}{W(a,b)}.$$

Bootstrap. Now that we have a structure that uses $O(n \log^2 \log n)$ space and answers weighted IRS queries in $O(\text{Pred}(U, w, n) + t)$ time, we can bootstrap this structure to reduce the space usage. More precisely, we note that the extra $\log^2 \log n$ factor comes from the baseline structure in each fat point. Thus, we can group the points in a fat point into secondary fat point of size $\log^2 \log n$ and build the baseline structure in the secondary fat point to reduce the space usage to $O(n \log^2 \log \log n)$. Repeat the bootstrap process $\log^* n$ times and we will have a structure with $O(n)$ space and $O(\log^* n)$ per-sample query time. The number of predecessor queries need to be performed is $O(\log^* n)$. However, for dataset with size $O(\log \log n)$, a predecessor query can be answered in constant time, which implies that the time for performing predecessor queries is still bounded by $O(\text{Pred} U, w, n)$.

▶ **Lemma 8.** *There is a structure of $O(n)$ space that can answer a one-dimensional weighted IRS query in $O(Pred(U, w, n) + t \log^* n)$ time.*

3.2 A structure with linear space and constant query cost.

In this subsection, we show how to obtain constant query cost by using *RAM tricks* to pack multiple integers into a single word.

Packing weights. We first apply the fat point technique twice to reduce the size of a fat point to $s = \log^2 \log n$. Note that if there is a linear size structure for s points with constant per-sample query time, we can apply it to each fat point, and achieve a linear size structure and constant per-sample query time for arbitrary number of weights.

Consider point sequence $p_1, \ldots, p_s$ with weights $w_1, \cdots, w_s$, where $s = \log^2 \log n$. If we maintain an alias structure for point sequence $\{w_i, \ldots, w_j\}$, for any pair $1 \le i \le j \le s$, then we can answer weighted IRS queries with constant time per sample. The problem is that there are $O(s^2)$ such pairs, so it requires $\Omega(s^3)$ space to store these structures.

To reduce the space cost, we round the probabilities to rational numbers with precision up to $1/s^2$, and pack multiple rational numbers into a single word. While constructing $O(s^2)$ alias structures for real weights is costly, constructing $O(s^2)$ alias structures for rational weights can be made space-efficient.

More precisely, given index pair $1 \le i \le j \le s$, let $r_k = w_k / W(p_i, p_j)$, $k = i, \ldots, j$, be the probability that p_k is sampled from $\{p_i, \ldots, p_j\}$, and let r_{k^*} denote the maximum probability in $\{r_i, \ldots, r_j\}$. We *conceptually* perform the following probability transfers: for every index $\ell \in [i,j]$, $\ell \neq k^*$, we define *rational probability* $r'_\ell = \lceil r_\ell s^2 \rceil / s^2$, and *deviation*

$\alpha_\ell = r'_\ell - r_\ell < 1/s^2$. We then pour α_ℓ probability mass from r_{k^*} to r_ℓ, to form the rational probability r'_ℓ, for $\ell \neq k^*$. Note that the probability mass left for k^* is

$$r'_{k^*} = r_{k^*} - \sum_{j \neq i_1} \alpha_j > r_{k^*} - s \cdot \frac{1}{s^2} \geq r_{k^*} - \frac{1}{s} > 0.$$

See figure 3. Then we build an alias structure $\mathcal{A}'(i,j)$ for $r'_i, \ldots, r'_j$. Here $\mathcal{A}'$ indicates that we build the alias structure on the rational probabilities rather than on the original probabilities. The key insight for this probability transfer is that we can store each rational probability r'_i with $2\log s$ bits, thus the alias structure $\mathcal{A}'(a,b)$ can be represented by $O(s \log s)$ bits. Over all possible pairs (i,j), this sums up to $O(s^3 \log s) = O(\log^6 \log n \log\log\log n) = o(\log n)$ bits. Thus, we only need one word to store all rational alias structures. We also record the index k^* for each pair (i,j), which takes $s^2 \cdot \log s = o(\log n)$ bits and fits in a word. Finally, we maintain all s prefix sums $W(1,p_i)$, $i = 1\ldots, s$ and this requires $O(s)$ *real-valued* storage. It is easy to see that the structure takes $O(s)$ space.

Answering queries. We focus on query ranges of form $[p_i, p_j]$, $1 \leq i \leq j \leq s$. Recall that s is the size of the secondary fat leaves. Note that each query visits at most two fat leaves of size s, so if we can generate a sample in constant time from ranges of form $[p_i, p_j]$, $1 \leq i \leq j \leq s$, we can answer weighted IRS queries on n points in $O(\text{Pred}(U, w, n) + t)$ time.

Given such a query $[p_i, p_j]$, we first compute $W(p_i, p_j)$ by subtracting two prefix sums $W(p_1, p_j) - W(p_1, p_{i-1})$. Then we retrieve w_{k^*}, the maximum weight in $[p_i, p_j]$, and compute probability $r_{k^*} = w_{k^*}/W(a,b)$. We then sample an point p_k from the rational alias structure $\mathcal{A}'(a,b)$. If $k = k^*$, we return p_{k^*} as a sample. Otherwise, we compute $r_k = w_k/W(a,b)$ and roll a random dice z uniformly chosen in $[0, r'_k]$. If $z \leq r_i$, we return p_k as the sample, otherwise, we return p_{k^*} as the sample.

Analysis. Since $W(p_i, p_j)$ and k^* can be supplied in constant time, the total query cost is constant. To verify the probability distribution, first consider a point $p_k \in [p_i, p_j]$, $k \neq k^*$. Observe that p_k is sampled if and only if its rational probability r'_k is sampled from $\mathcal{A}'(a,b)$, and the random dice z from $[0, r'_k]$ is smaller than r_k. The probability is $r'_i \cdot \frac{r_i}{r'_i} = r_i$. On the other hand, this also implies that k^* is returned with probability $1 - \sum_{p_k \in [p_i, p_j], k \neq k^*} r_k = r_{k^*}$. Thus the sampling probability distribution is correct, and we obtain the following theorem.

▶ **Theorem 9.** *Given a set $P = \{p_1, \cdots, p_n\}$ of n points on grid $[U]$, such that each point p_i is associated with an non-negative real-valued weight w_i, we can build a data structure of size $O(n)$, such that given a interval $[a,b]$ and a parameter t, we can extract t weighted random samples from the subset $P \cap [a,b]$ in $O(Pred(U, w, n) + t)$ time.*

4 Conclusions

In this paper we considered the range sampling queries where the goal is to store a given set of points in a data structure such that given a geometric range, a query returns a random sample of the points contained in the query range. We optimally solved some of the important cases of the problem: 3D halfspace queries for unweighted points, 3D dominance queries and 2D three-sided queries, and 1D two-sided (interval) queries for weighted points.

There are still a number of interesting open problems left to consider. For example, we have not investigated weighted 2D orthogonal queries at all. Also, while we solve three-sided and two-sided queries for the unweighted case, 2d four-sided queries for the unweighted case is still unsolved. Another direction is to consider weighted 3D halfspace queries.

References

1 Peyman Afshani and Timothy M. Chan. On approximate range counting and depth. *Discrete and Computational Geometry*, 42:3–21, 2009.

2 Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 180–186, 2009.

3 Peyman Afshani, Chris Hamilton, and Norbert Zeh. A general approach for cache-oblivious range reporting and approximate range counting. *Computational Geometry: Theory and Applications*, 43:700–712, 2010. preliminary version at SoCG'09.

4 Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. *Advances in Discrete and Computational Geometry*, pages 1–56, 1999.

5 Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

6 Arnab Bhadury, Jianfei Chen, Jun Zhu, and Shixia Liu. Scaling up dynamic topic models. In *Proceedings of International World Wide Web Conferences (WWW)*, pages 381–390. International World Wide Web Conferences Steering Committee, 2016.

7 Timothy M. Chan. Random sampling, halfspace range reporting, and construction of (<= k)-levels in three dimensions. *SIAM Journal of Computing*, 30(2):561–575, 2000.

8 Timothy M Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the ram, revisited. *arXiv preprint arXiv:1103.5510*, 2011.

9 Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Random sampling for histogram construction: How much is enough? In *ACM SIGMOD Record*, pages 436–447. ACM, 1998.

10 Bernard Chazelle, Leonidas J. Guibas, and D. T. Lee. The power of geometric duality. *BIT Numerical Mathematics*, 25(1):76–90, 1985.

11 Robert Christensen, Lu Wang, Feifei Li, Ke Yi, Jun Tang, and Natalee Villa. Storm: Spatio-temporal online reasoning and management of large spatio-temporal data. In *Proceedings of ACM Management of Data (SIGMOD)*, pages 1111–1116. ACM, 2015.

12 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3 edition, 2008.

13 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal of Computing*, 13(2):338–355, 1984.

14 Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. Online aggregation. *ACM SIGMOD Record*, 26(2):171–182, 1997.

15 Xiaocheng Hu, Miao Qiao, and Yufei Tao. Independent range sampling. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 246–255. ACM, 2014.

16 Aaron Q Li, Amr Ahmed, Sujith Ravi, and Alexander J Smola. Reducing the sampling complexity of topic models. In *Proceedings of ACM Knowledge Discovery and Data Mining (SIGKDD)*, pages 891–900. ACM, 2014.

17 Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. Wander join: Online aggregation via random walks. In *Proceedings of the 2016 International Conference on Management of Data*, pages 615–629. ACM, 2016.

18 Jiří Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8(3):315–334, 1992.

19 Frank Olken. *Random sampling from databases*. PhD thesis, University of California at Berkeley, 1993.

20 Frank Olken and Doron Rotem. Random sampling from databases: a survey. *Statistics and Computing*, 5(1):25–42, 1995.

21 Frank Olken and Doron Rotem. Sampling from spatial databases. *Statistics and Computing*, 5(1):43–57, 1995.

22 Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.

23 Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Symposium on Computational Geometry (SoCG)*, pages 390–399, 1999.

24 Jiří Matoušek. Reporting points in halfspaces. *Computational Geometry, Theory and Applications*, 2(3):169–186, 1992.

25 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 75–84. IEEE, 1975.

26 Alastair J. Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, 1974.

27 Lu Wang, Robert Christensen, Feifei Li, and Ke Yi. Spatial online sampling and aggregation. *Proceedings of the VLDB Endowment*, 9(3), 2015.

5 Appendix

5.1 Walker's Alias Method.

In the weighted sampling problem, the input is a set of non-negative real numbers $w_1, \ldots, w_n$, and the goal is to build a data structure, such that a query extracts an index i with probability $p_i = w_i / \sum_{j=1}^{n} w_j$. The indices returned by different queries should be independent. As discused, the classic solution to this problem is Walkers' alias method [26], which uses $O(1)$ query time and $O(n)$ preprocessing time and it works as follows.

Consider n vases, where vase i contains some (probability) mass r_i. Initially, r_i is equal to p_i. During the algorithm, we iteratively modify the values v_i's in the following way. In each iteration, we pick a vase i with $r_i > 1/n$ and another vase j with $r_j < 1/n$ of mass, and pour $1/n - r_j$ probability mass of vase i to vase j. This makes the probability mass in vase j equal to $1/n$. If no such two vases exist, then all the vases contain $1/n$ of mass. After at most n such iterations, each vase contains $1/n$ mass that comes from at most two vases. It is easy to see that the preprocessing time is $O(n)$. To draw a sample, we first uniformly sample a vase $k \in \{1, \ldots, n\}$. Assume that the vase k contains a probability mass p from p_i and a probability mass $1/n - p$ from p_j. We then draw a random number $z \in [0, 1/n]$, and return i as the final sample if $z < p$, and j as the final sample otherwise. It is easy to see that drawing a sample takes $O(1)$ time.

5.2 Proof of Theorem 6

We begin by handling some easy cases for the query. We build the following three data structures: one, using Theorem 3 a structure for for halfspace range reporting queries on P, two, using Theorem 4 with $\varepsilon = 0.5$ another structure for approximate halfspace range counting queries in P, and three, an instance of the data structure of Lemma 5 on P.

We use a bootstrapping idea, that was employed before [7, 2, 23]. Using the approximate halfspace range counting data structure, in $O(\log n)$ time we find a value $\tilde{k}$ s.t., $\tilde{k}/2 \le k \le \tilde{k}$. If $\tilde{k} \le \log n$ then using the halfspace range reporting data structure (Theorem 3) we find all the k points contained in h and extract t random samples from them in online fashion; the running time of this step is $O(\log n + k + t) = O(\log n + t)$. Furthermore, if $\tilde{k}/2$ is large, then it follows that k is large and thus we can answer the query using Lemma 5.

So assume $\tilde{k}/2$ is not large. Let $r = 2^{C(\log\log n)^2}$. To handle this case, we build a r-shallow cutting $\mathcal{F}$ on H, and its set of triangles, $\Delta(\mathcal{F})$. For every triangle $\tau \in \Delta(\mathcal{F})$, we build a "local" copy of the data structure of Lemma 5 on the set P_τ. Each local copy consumes $O(r)$ space as the data structure of Lemma 5 uses linear space, and there are $O(n/r)$ triangles in $\Delta(\mathcal{F})$, by the shallow cutting theorem. We also store a point location data structure corresponding to the xy-projection of the triangles in $\Delta(\mathcal{F})$. Thus, this step in total requires $O(n)$ space. Consider a query halfspace h that contains fewer than r points. In dual space, $\overline{h}$, is a point that lies below $\mathcal{F}$ and thus is below a triangle $\tau \in \Delta(\mathcal{F})$. The triangle τ is found once by a point location query in $O(\log n)$ time. This implies $P \cap h$ is a subset of P_τ. If h contains more than $2^{C(\log\log|P_\tau|)^2}$ points, then we can answer the query using the local copy of Lemma 5 built on P_τ. Otherwise, we must have fewer than $2^{C(\log\log|P_\tau|)^2}$ points in h. Since $|P_\tau| = O(r)$, it follows that h contains $o(\log n)$ points and thus can be answered using the first easy cases considered.

5.3 Solving the Recurrence in Section 2.3

In this section we deal with the following recursion.

$$f(n,k) \le \begin{cases} 1 & \text{if } n \le b \\ \sum_{i=1}^{O(\log n)} f(cn^{1-\varepsilon}, k_i) & \text{if } k \le n^{1-\varepsilon} = \frac{n}{r} \\ \sum_{i=1}^{n^\varepsilon} f(cn^{1-\varepsilon}, k_i) & \text{if } k > n^{1-\varepsilon} = \frac{n}{r} \end{cases} \quad \text{where } k = \sum_i k_i.$$

Similar to the analysis done in [2], we solve the recurrence by guessing that it solves to the "correct" bound, that is, it solves to $f(n,k) = 2^{c(\log\log n)^2} + kg(n)/b^{1-3\varepsilon}$, where $g(n)$ is a monotonously increasing function that is always upper bounded by a constant and c is a constant.

The claim clearly holds for the base case when $n \le b$. Now, assume $k \le n^{1-\varepsilon}$. By the induction hypothesis, we have

$$f(n,k) \le \sum_{i=1}^{O(\log n)} f(cn^{1-\varepsilon}, k_i) \le \sum_{i=1}^{O(\log n)} \left(2^{c\left(\log\log cn^{1-\varepsilon}\right)^2} + \frac{g(cn^{1-\varepsilon})k_i}{b^{1-3\varepsilon}} \right)$$
$$\le O\left(\log n 2^{c\left(\log\log cn^{1-\varepsilon}\right)^2}\right) + \frac{g(n)k}{b^{1-3\varepsilon}}$$

where the last inequality follows since $g(n)$ is assumed to be monotonously increasing and that $\sum_{i=1} k_i = k$. So we just need to verify that $\log n 2^{c\left(\log\log cn^{1-\varepsilon}\right)^2} = o(2^{c(\log\log n)^2})$ which follows as long as $c \ge 4/\varepsilon$.

Now consider the last case when $k > n^{1-\varepsilon}$:

$$f(n,k) \quad \le \quad \sum_{i=1}^{n^\varepsilon} 2^{c(\log\log n)^2} + \frac{k_i g(cn^{1-\varepsilon})}{b^{1-3\varepsilon}} \le n^\varepsilon 2^{c(\log\log n)^2} + \frac{kg(cn^{1-\varepsilon})}{b^{1-3\varepsilon}} \le n^{2\varepsilon} + \frac{kg(cn^{1-\varepsilon})}{b^{1-3\varepsilon}}.$$

Thus, we must have $n^{2\varepsilon} + kg(cn^{1-\varepsilon})/b^{1-3\varepsilon} \le kg(n)/b^{1-3\varepsilon}$ which since $k > n^{1-\varepsilon}$, gives the recursion

$$g(n) \ge b^{1-3\varepsilon} n^{2\varepsilon}/n^{1-\varepsilon} + g(cn^{1-\varepsilon}) = (b/n)^{1-3\varepsilon} + g(cn^{1-\varepsilon}).$$

However, since $n \ge b$, if we set $\varepsilon < 1/3$, this recursion clearly upper bounds $g(n)$ by a constant.

5.4 Without-replacement sampling.

Another sampling strategy is the *without-replacement* sampling. For the unweighted case, given a range $R \in \mathcal{R}$ and an integer $t \ge 1$, the query returns a subset $S \subset R \cap P$ with $|S| = t$ that is selected uniformly at random from all the subsets of $P \cap R$ that have size t. For the weighted case, the query yields a *set* of t random points where after sampling a point, the point is removed before continuing with the next sample, leading to no repetitions. More precisely, let S_k denote the set of the first k samples, and define $S_0 = \emptyset$. The $(k+1)$-th sample s_{k+1} is selected from $(P \cap R) \setminus S_k$, such that each point $p_i \in (P \cap R) \setminus S_k$ is selected with probability $w_k / \sum_{p_j \in (P\cap R)\setminus S_k} w_j$. By selecting $k = t$ points, we obtain a without-replacement sample set $S = S_t$.

Note that for uniform sampling, with-replacement sampling can be used to generate without-replacement samples: if the range contains less than $2t$ points then we can just find them all and then sample, otherwise, we generate $\Theta(t)$ with-replacement samples and reject all repetitions. For the weighted sampling, however, the rejection algorithm does not work since the weights may be unbalanced, and with-replacement sampling will keep sampling

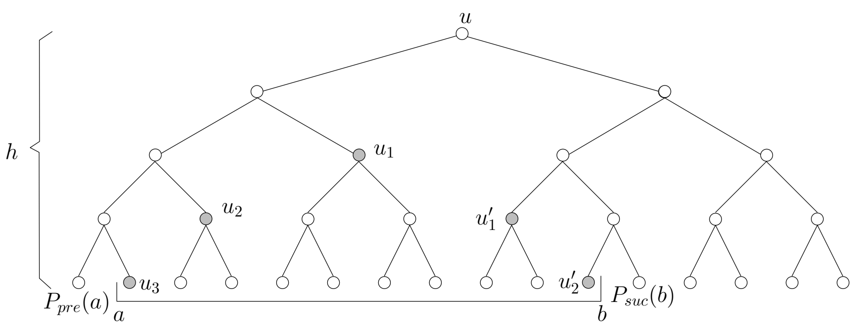

Figure 4 An illustration of canonical sets and predecessors/successors. The predecessor of a is $P_{pre}(a)$, and the successor of b is $P_{pre}(b)$. The lowest common ancestor of $P_{pre}(a)$ and $P_{pre}(b)$ is u. The right canonical set $\mathcal{R}(u, P_{pre}(a)) = \{u_1, u_2, u_3\}$ and left canonical set $\mathcal{R}(u, P_{suc}(b)) = \{u'_1, u'_2\}$ make up the point set in $[a, b]$.

the points with large weights which leads to too many repetitions. In this paper, we only consider with-replacement sampling for weighted points. Note that obtaining a linear-size data structure that can return "without-replacement" samples in $O(t)$ time is probably a very difficult problem, since this is still open even when no query range is specified.

5.5 Proof of Lemma 7

Proof. We build three indexing structures: a balanced binary tree $\mathcal{T}$ on the n point, a van Emde Boas tree [25] that can return the predecessor $P_{pre}(a)$ and successor $P_{suc}(a)$ of any coordinate a in $O(\log\log U)$ time, and a lowest common ancestor (LCA) structure [13] that returns LCA of any two leaves of $\mathcal{T}$ in constant time. Consider a root-to-leaf path $\mathcal{P}(r, v)$, where r is the root of $\mathcal{T}$ and v is a leaf. For each internal node u on $\mathcal{P}(r, v)$, we build an alias structure $\mathcal{A}(\mathcal{P}(u, v))$, for the weights of nodes in the right canonical set $\mathcal{R}(u, v)$. This structure allows us to sample a subtree from $\mathcal{R}(u, v)$ according to their weights. We also store $W(\mathcal{R}(u, v))$, the total weights in $\mathcal{R}(u, v)$. Similarly, we store $\mathcal{A}(\mathcal{L}(u, v))$ and $W(\mathcal{L}(u, a))$ for the left canonical set $\mathcal{L}(u, v)$. Note that there are at most $h = O(\log n)$ different internal nodes on $\mathcal{P}(r, v)$, and each internal node stores an alias structure with $O(h) = O(\log n)$ weights. Thus, the space usage for each root-to-leaf path is $O(\log^2 n)$, which sums up to $O(n \log^2 n)$ for n leaves. Finally, for each internal node u, we build an alias structure $\mathcal{A}(u)$ with the weights in the subtree of u. Observe that each weight participates in at most h such structures, so the space usage for the $\mathcal{A}(u)$ structures is $O(n \log n)$. Therefore, the total space usage is $O(n \log^2 n)$.

Answering queries. Given a query range $[a, b]$, we first find $P_{pre}(a)$, the predecessor of a and $P_{suc}(b)$, the successor of b in P, in $\log\log U$ time. Note that $P_{pre}(a)$ and $P_{suc}(b)$ define the boundaries of the points in $[a, b]$. Then we find u, the lowest common ancestor of $P_{pre}(a)$ and $P_{suc}(b)$, in constant time. We sample a weight from $W(\mathcal{R}(u, P_{pre}(a))$ and $W(\mathcal{L}(u, P_{suc}(b)))$ to determine if the final sample is from the left or right canonical set. Suppose the right canonical set $\mathcal{R}(u, P_{pre}(a))$ is selected; Then we sample from alias structure $\mathcal{A}(\mathcal{R}(u, P_{pre}(a)))$ to retrieve an internal node $v \in \mathcal{R}(u, P_{pre}(a))$, and sample an point from $\mathcal{A}(v)$ as the query result. For query cost, note that it takes $O(\log\log U)$ time to find the successor $P_{pre}(a)$ and predecessor $P_{suc}(b)$, and constant time to find the lowest common ancestor u takes constant time. Thus,the search time is $O(\log\log U)$. More over, it takes constant time for sampling from alias structures $\mathcal{A}(\mathcal{R}(u, a))$ and $\mathcal{A}(v)$, it follows that the per-sample query cost is constant.

Analysis To see that this returns a point with correct probability distribution, we consider the probability that an point $p_k \in [a, b]$ is sampled. Without loss of generality, we assume p_k is in the right canonical set $\mathcal{R}(u, P_{pre}(a))$, and $v \in \mathcal{R}(u, P_{pre}(a))$ is the internal node that contains p_k. The probability that the right canonical set $\mathcal{R}(u, P_{pre}(a))$ is selected is $W(\mathcal{R}(u, P_{pre}(a)))/W(a, b)$. The probability that v is sampled in $\mathcal{A}(\mathcal{R}(u, P_{pre}(a)))$ is $W(v)/W(\mathcal{R}(u, P_{pre}(a)))$. Finally, the probability that p_k is sampled in $\mathcal{A}(v)$ is $w_k/W(v)$. Therefore, the probability that k is sampled is

$$\frac{W(\mathcal{R}(u, P_{pre}(a)))}{W(a, b)} \cdot \frac{W(v)}{W(\mathcal{R}(u, P_{pre}(a)))} \cdot \frac{w_k}{W(v)} = \frac{w_k}{W(a, b)}.$$

◀