Colored Range Sampling Draft*

Matin Pirouz
California State University, Fresno
Fresno, California
mpirouz@csufresno.edu

Nathaniel Madrigal
California State University, Fresno
Clovis, California
nthnlmdrgl@mail.fresnostate.edu

Alexander Madrigal
California State University, Fresno
Clovis, California
lxndrmdrgl@mail.fresnostate.edu

***Abstract*—Range searching has been an important problem relating to querying databases. Reporting all points in a database which lie in a range is a common problem, but with the increased size of databases in the big-data era, these types of queries have become unsatisfactory as too many points may be found. It is thus important to find a representative sample of points lying in the query range instead. When points are placed in certain categories (have different colors), we may also want to report samples proportionally to the weights of colors. In this paper, we present data structures to answer orthogonal range sampling queries on colored datasets. We study current research on the colored range reporting and counting problems, and how they can be translated to give satisfactory colored range sampling solutions. In two-dimensional space, we present a data structure which can sample colored points which intersect an orthogonal range proportionally to the non-negative weight of colors in near-linear space and near-polylogarithmic time. We evaluate our algorithms with real and simulated datasets.**

I. Introduction

Range searching is an important problem in the field of computational geometry for its uses in databases [2]. Objects in a database can be represented by points with the number of dimensions correlating with different fields of information. When querying, *orthogonal range searching* allows you to find all points within an axis-aligned box. When reporting points from a range query, we want our query to be *output-sensitive* to the number of reported points. This means that the time needed to query is proportional to the number of reported points allowing smaller queries to be faster than if there were more points to report. With modern datasets, this solution does not suffice as new large datasets have a significantly large amount of points to be reported. Instead, we want to find a sample of the points which lie in the query.

In a web searching example, let's say we search for phones within a specific price range. We would like to select a sample of phones to show on the front page. The question becomes, which sample of phones do we select? One consideration is that the phones selected should be representative of all the phones within the price range. Representative samples can be found with independent range sampling [6], [12], [13]. Independent sampling also allows the samples to have greater fairness in respect to queries. Ensuring that queries are independent of sampling is important when multiple queries are performed on similar ranges.

If we also assign different categories to different phones such as brand, we also want our returned sample to be fair to the different categories. For example, in consideration of different sponsorships you may want to increase or decrease the probability of selecting certain brands in our sample. It is thus necessary to give each category a specified weight and perform weighted random sampling on the categories. In scenarios where categories are distributed unequally across a dataset and different categories are underrepresented, weights can be used to increase representation in sampling.

Colored range searching is a variation or range sampling where data is assigned with different colors to represent being placed into separate categories. This problem is also known as *generalized range searching* or *categorical range searching*. Many papers have studied this generalization on range searching [3]–[5], [7], [8], [14], [15]. Two common goals of colored range searching are to report all distinct colors or count all distinct colors which intersect a range called *colored range reporting* and *colored range counting* respectively [3], [4]. Different ranges studied can be axis-aligned rectangles, half-spaces, or other geometric shapes such as circles. There are also static or dynamic variations of this problem.

In this paper our focus is on a modification of the colored range searching problem. Our goal is to preprocess a data structure in such a way that finds an independent random sample of points which lie in an axis-aligned box efficiently. The colors of the sampled points should be found proportionally to the weights of the colors. We call this problem *colored orthogonal range sampling*.

A. Related Work

1) Colored orthogonal range reporting: In [3], [4], solutions for colored orthogonal range reporting were found for spaces in R^1, R^2, and R^3. In R^1, colored range searching is solved by reducing the problem into the standard range searching problem in R^2. If we have a set of points of the same color $p_1, p_2, \ldots, p_n$ such that $p_1 < p_2 < \cdots < p_n$, they can be mapped to the points $(p_1, -\infty), (p_2, p_1), \cdots, (p_n, p_{n-1})$. If we have a query range $[a, b]$, searching on the all of the transformed points in R^2 with the range $[a, b] \times (-\infty, a]$, it is observed that if a point of color c is in $[a, b]$ there must be a unique point of color c is in $[a, b] \times (-\infty, a]$. Using this transformation, where k is the number of distinct colors in range, a data structure with $O(n)$ size can be constructed to report all colors in a query interval in $O(logn + k)$ time and support insertions and deletions in $O(logn$ time.

In R^3, a solution by [8] improves the query time by using a randomized Las Vegas data structure with $O(nlogn)$ space which can report all k distinct colors in a query dominance range in $O(loglogU + kloglogn)$ expected time. The orthogonal reporting problem is found by the authors using a halfspace range searching algorithm in the same paper. They also found a solution in R^3 with a randomized Las Vegas data structure using $O(nlog^{d-1+\varepsilon}n)$ space which can report all k distinct colors in a query range in $O(k(\frac{logn}{loglogn})^{d-3}loglogn)$ expected time. These data structures have query times proportional to number of distinct colors k meaning that these data structures have faster times when k is smaller relative to previous solutions.

2) Colored orthogonal range counting: In [3], [4], solutions for colored orthogonal range counting in the static version of the problem were found for spaces in R^1 and R^2. In R^1, the same transformation technique was used as colored range reporting in R^1 to construct a data structure which can can count k number of distinct colors in a query interval in $O(logn + k)$ time with $O(nlogn)$ space.

In R^2, algorithms which transform the standard orthogonal range into semi-unbounded queries as well as one with a time-space tradeoff which can improve space requirements to near-linear at the expense of query time were found by [5]. A transformation is given where orthogonal counting queries in R^d can be transformed into a semi-unbounded counting query in R^{2d}. Since semi-unbounded range queries can count all colors with $O(n^{\lfloor d/2 \rfloor}log^{d-1}n)$ space and $O(log^{d-1}n)$ query time, then there also is an algorithm which can solve colored orthogonal range counting queries with $O(n^dlog^{2d-1}n)$ space and $O(log^{2d-1}n)$ query time. For the time-space tradeoff, the orthogonal query with range $[a, b] \times [c, d]$ is decomposed into two three-sided queries of $[a, b] \times [c, \infty)$ and $[a, b] \times (-\infty, d]$ where there may be overlapping colors counted by the two queries. These three-sided queries in turn can be solved with semi-unbounded queries in R^3. The number of intersecting colors between the two queries must be computed and removed from the color count. Here, time-space tradeoff is applied by computing this intersection either in preprocessing or during querying. By choosing to preprocess the intersection of canonical points with a count (the number of leaves in the subtree) less than the tradeoff parameter $1 \leq X \leq n$ we are then able to create a data structure with $O((\frac{n}{X})^2log^6n + nlog^4n)$ storage and can count the number of colors in $O(Xlog^2n)$ query time. With this solution, choosing a tradeoff parameter $X \geq \sqrt{n}$ provides a data structure with near linear space.

3) Colored halfspace range searching: In [7], solutions for colored halfspace range searching (allowing both reporting and counting) were found for spaces in R^2, R^3, and R^d where $d \geq 4$.

In R^3, faster query times were found by [8] using a randomized Las Vegas data structure which has $O(nlogn$ space and can report all k distinct colors in a query halfspace in $O(klogn)$ expected time.

TABLE I
PREVIOUS RESULTS FOR COLORED RANGE SEARCHING

Ref.	Problem	space	query time
Gupta et al. (2018) [3]	1-D Range reporting	$O(n)$	$O(logn + i)$
	2-D Range reporting	$O(nlogn)$	$O(log^2n + ilogn)$
	3-D Range reporting	$O(nlog^4n)$	$O(log^2n + i)$
Gupta et al. (2018) [3]	1-D Range counting	$O(nlogn)$	$O(logn + i)$
	2-D Range counting	$O(n^2log^2n)$	$O(log^2n + i)$
Gupta et al. (1996) [7]	2-D Halfspace range searching	$O(nlogn)$	$O(log^2n + i)$
	3-D Halfspace range searching	$O(nlog^2n)$	$O(n^{1/2+\varepsilon} + i)$
	3-D Halfspace range searching	$O(nlogn)$	$O(n^{2/3+\varepsilon})$
	Halfspace range searching $d \geq 4$	$O(n^{\lfloor d/2 \rfloor} \div (logn)^{\lfloor d/2 \rfloor - 1 - \varepsilon})$	$O(logn + ilog^2n)$
Kaplan et al. (2008) [5]	Semi-unbounded range counting	$O(n^{\lfloor d/2 \rfloor} \times log^{d-1}n)$	$O(log^{d-1}n)$
	Orthogonal range counting	$O(n^dlog^{2d-1}n)$	$O(log^{2d-1}n)$
	2-D Orthogonal range counting	$O((\frac{n}{X})^2log^6n + nlog^4n)$	$O(Xlog^2n)$
Chan et al. (2020) [8]	3-D Halfspace range reporting	$O(nlogn)$	$O(ilogn)$ expected
	Orthogonal range reporting $d > 3$	$O(nlog^{d-1+\varepsilon}n)$	$O(i(\frac{logn}{loglogn})^{d-3} \times loglogn)$ expected

Note: Here $i \geq 0$ denotes the number of distinct colors in the query range, $d \geq 0$ denotes the dimension of space R^d, and X denotes the tradeoff parameter where $1 \leq X \leq n$.

B. Our Contribution

Let P be a set of n points in R^d where each point has a color from C given colors. For each $c \in C$ all points with color c will have the same non-negative weight w given. Our goal is to preprocess P into some data structure that, given a query axis-aligned box $Q \subseteq R^d$, samples s points from $P_q := Q \cap P$. The samples from P_q should be found randomly with probabilities proportional to the weights. In other words, for each point $p \in P_q$, where $w(p)$ notates the weight of the p, p is sampled with probability $w(p)/\sum_{p' \in P_q} w(p')$. This is the definition of the colored orthogonal range sampling problem.

In this paper we find solutions for this problem in R^1 and R^2 in the static case. Our solutions are the combinations of known colored range searching algorithms and independent range sampling techniques. We describe the method for independent sampling utilized and how it can be integrated into colored range searching solutions to provide similar space and time requirements for sampling. We also conduct experiments on various datasets to prove the correctness of our algorithms.

II. Methods

A. Weighted Random Sampling on Orthogonal Search Trees

Before we explore the colored range sampling problem, first we will find how to generate a weighted random sample (WRS) in the standard range sampling problem. In [1], an algorithm for solving WRS over a population of n weighted items in one pass is given. For every item v, this algorithm generates a random number $u = random(0, 1)$ and calculates a key $k = u^{1/w(v)}$. A WRS of size s can then be found by selecting s items from the population with the greatest keys. This algorithm thus finds a WRS in time linear to population size n.

With this algorithm in mind, a naive solution to solve range sampling can be found. First, receive the full result of a range reporting query and then sample from this result by searching for the s greatest keys. However, this solution goes against our motivation for range sampling by having to first compute all points within a query range. We should look for a solution that can generate a WRS of points in time significantly less than the number of points which lie in a query.

Using the properties of how certain orthogonal search trees have canonical nodes, there exists a method to find a s weighted random samples over k number of points inside a query range after the canonical nodes of a query range are computed in an additional $O(s(logn + logk))$ time. Canonical nodes are nodes in a tree whose subtree rooted at that node only contains leaves inside the query range.

To demonstrate the additional time required for WRS, let us examine an example of range sampling in one dimension. To solve range searching in one dimension, you can build a balanced binary search tree on P sorted on the x-coordinates [2]. Leaves on this tree store the points $p \in P$, and internal nodes store splitting values to direct a search. Assign the weight of a parent node to be the sum of the weights of its children. Given a query range Q represented by the interval $[x : x']$, you can find all subtrees which intersect Q by finding the leaf nodes where the search to x and x' end starting from the split node – the node where the paths to x and x' split. When finding the path to x, for every instance where you traverse to the current node's left child, the right child's subtree only contains leaves with points which intersect Q. Nodes where subtrees with leaves only in Q are rooted at are canonical nodes. On the path to x', whenever you step to the current node's right child, the left child is a canonical node. Canonical nodes may also be the leaves where the path to x and x' end if these points intersect Q. Since we are working with a balanced binary search tree, the paths to x and x' are length $O(logn)$. It follows that the amount of canonical nodes found from this query is $O(logn)$ since a maximum of one canonical node is found on each step towards x or x'.

Going back to WRS, we can use the $logn$ canonical nodes we found to generate a single sample. First, use the algorithm in [1] on the weights of the $logn$ canonical nodes. Find the canonical node with the greatest key and conduct the following steps to select a leaf node to sample. While the current node is not a leaf, find which child has the greatest key and traverse that child. The leaf node at the end of this traversal is your WRS. Repeat this process for s samples.

In summary, a range searching query finding $O(logn)$ canonical nodes representing a query range Q can find s weighted random samples in $O(s(logn + logk))$ time where n is the number of points in the dataset and k is the number of points which intersect Q. It is important to keep in mind that this algorithm finds s samples $with$ replacement. In other words, different samples in the same range sampling query may return the same point multiple times.

B. Dependent Weighted Random Sampling

We will provide a simple solution for the finding weighted random samples intersecting an orthogonal range in all dimensions if we do not require independent sampling using the weighted sampling algorithm given by [1]. This technique constructs the randomness of the sampling queries into the preprocessing instead of querying. As a result, the result of a random sample is dependent on the query range provided to the data structure. The following data structure is a modification of a kd-tree.

Construct a kd-tree on a set of weighted points P in R^d. On the leaf nodes of the kd-tree, store a preprocessed calculation for a key using the weight of their points. Each node will also have a $maxNode$ pointer to the leaf with the greatest key inside of the subtree rooted at this node. The $maxNode$ of a leaf node should point to itself. The key of a parent node should be the maximum key of either the left or right child node. Similarly, the $maxNode$ of a parent node should be the same $maxNode$ of the child with the greatest key. When the final data structure is complete, every node in the kd-tree points to some leaf which represents the point with the greatest key for that path along the kd-tree. This kd-tree is constructed in asymptotically similar time to a standard kd-tree, meaning that the preprocessing time of this data structure is $O(nlogn)$ time and requires $O(n)$ storage.

A WRS of s points can be found from this data structure in $O(n^{1-1/d} + slogn)$. First, find all the canonical nodes representing a query range. From here find the canonical node with the largest key and sample the point it's $maxNode$ points to. Repeat this for every one of s samples. Since there $logn$ number of canonical nodes and no additional traversal is needed to find samples, the additional time required to find a sample set after computing canonical nodes is $O(slogn)$. With this implementation, every query range with the same canonical nodes will sample the same points, namely the points where each canonical nodes' $maxNode$ points to.

Put together, there exists a data structure which can answer weighted orthogonal range sampling in R^d by preprocessing randomness, requiring $O(nlogn)$ preprocessing time and $O(n)$ storage. Queries with s samples can be found in $O(n^{1-1/d} + slogn)$ time. While this data structure is strong in needing only near-linear space and sublinear query time, it has a massive downfall in not returning independent samples. In practice, it is not uncommon to want to return multiple different samples

of points in similar ranges. This data structure will return a non-diverse set of samples over the same query range. Since our goal of range sampling was to find a representative set of samples of P_q, a more desirable solution is necessary. All range sampling data structures from this point onwards will find samples from P_q independently of the given query range Q.

It is also important to note that this data structure does find a correct solution for colored range sampling as this solution does not guarantee the correct proportion of sampled colors. Constructed on a set of colored points where the weights of points are the weights of its color, the algorithm only guarantees that a sample is found proportionally to the weight of that point and all points found in the query. In the scenario where points of certain colors have non-uniform distribution inside of a query, the probabilities of a point of a color being sampled are increased when there are more points with that color. By contrast, if there is a uniform distribution of colored points in a query (let's say one unique point of a color), this data structure would provide the correct solution. Since it is not realistic in datasets to have evenly distributed categories or colors, we would like to be able to find a uniform distribution of colored points during the querying process to sample from. This observation reveals that being able to separate unique colored points from all points which lie in a query range is necessary to use this weighted sampling algorithm.

C. Reducing to Standard Range Sampling in R^1

As demonstrated in the previous section, being able to find samples of colored points independently of Q is an important consideration for any data structures answering colored orthogonal range sampling. One method is to use algorithms which solve the standard range sampling problem with independent sampling. We must, however, be able to reduce the colored range sampling problem to a standard problem without colors. In this section we will be overviewing the solution found in [3], [4] to reduce the colored range counting and reporting problems in R^1 to a standard range searching problem in R^2 and how this technique can be used to solve the colored range sampling problem in one dimension.

As described in detail previously, [3], [4] describes a transformation from one dimension to two which finds a unique point for every color within a query range. With this transformation, it is possible to be able to report all colors found within a query interval with standard range searching in two dimensions. It is then also possible to sample colors proportionally to the weights of colors. We then are able to trivially sample a point from sampled color. Using this transformation, we will now describe a data structure that solves orthogonal colored range sampling taking $O(n)$ storage, $O(nlogn)$ preprocessing time, and $O(\sqrt{n} + s(logk + logn))$ query time where s is the number of samples and k is the number of colors within the query range.

For every point of a certain color c in P, sort all of these points in ascending order. Transform the points into two dimensions as described in [3], [4] and then construct a kd tree on all transformed colored points. When building the kd tree, the leaf nodes store the weight of the colored point and parent nodes store the sum of weight for both of its children nodes. When performing a query on kd tree with the transformed query range in two dimensions, it is known that there will be at most one point for each color inside the query. This means that across all canonical nodes of the query, at the leaves lie all colors which lie within the original one dimensional range interval. After computing all canonical nodes, to find s samples we then perform the weighted random sampling technique used previously. We find the canonical node with the greatest key $k = u^{1/w(p)}$ where $u = random(0, 1)$ and $w(p)$ is the non-negative weight of the canonical node. After finding a canonical node, traverse down the subtree. Compute whether the left or right child has a greater key and traverse that child repeating this step until a leaf is reached. By finding the color of this leaf node, we have randomly sampled a color in proportion to the weight of colors in the query range. Repeat the above steps to find multiple independent sample colors. Now we must use these sampled colors to sample a point of that color inside the query range. To accomplish this, in the preprocessing of the data structure also build a binary search tree for every unique color on requiring an additional $O(n)$ space. Searching on the bst of the color sampled previously, perform a range search using the query range interval and sample a point uniformly at random. Uniform random sampling can be achieved by assigning each leaf node to a same constant weight, assigning an internal node's weight to be the sum of the weight of the left and right child, and then performing the same weighted sampling technique used previously. We now have a solution for colored orthogonal range sampling for one dimensional points.

D. Reducing to Standard Orthogonal Range Sampling in R^2 using Semi-unbounded Queries

To answer colored range sampling in two dimensions, a new reduction must be used to find the colors present in a query range. In this section we will be using the solution to colored range counting by [5] on both semi-unbounded queries and two dimensional orthogonal queries and how this can be extended to answer colored range sampling. This also shows an example where unlike the solution in R^1, colored range sampling poses unique challenges not trivially answered with colored range counting algorithms.

Algorithm 1 Algorithm for $Build2DTree$

$Input$. A set of P colored points on a plane
$Output$. The root of a 2-dimensional colored range sampling tree

1: **if** P contains only one point p **then**
2: Create a leaf node v storing this point and assign $w(p)$ as the weight of v
3: **else**
4: Split P into two subsets; one subset P_{left} contains y-coordinates less than or equal to y_{mid}, the median y-coordinate of points in P, and the other subset P_{right} contains points with y-coordinate larger than y_{mid}
5: $v_{left} \leftarrow Build2DTree(P_{left})$
6: $v_{right} \leftarrow Build2DTree(P_{right})$
(* $Build3SidedTree$ creates trees in such a way that they can report all unique colors from points within a query range $[a, \infty) \times [b, \infty) \times [c, \infty)$ *)
7: $aux_{left} \leftarrow Build3SidedTree(P_{left})$
8: $aux_{right} \leftarrow Build3SidedTree(P_{right})$
(* $BuildMatrix$ will align all *heavy* internal nodes of aux_{left} (with a weight greater than a tradeoff parameter X) against all heavy internal nodes of aux_{left} in a matrix. Each cell in the matrix stores the sum of weights of all intersecting colors between the two internal nodes *)
9: $m \leftarrow BuildMatrix(aux_{left}, aux_{right})$
10: Create a node v storing p_{mid}–the point with median x-coordinate–make v_{left} (resp. v_{right}) the left (resp. right) child of v, make aux_{left} (resp. aux_{right}) the auxiliary left (resp. right) structure of v, make m the assigned matrix of v, and make the sum of $w(v_{left})$ and $w(v_{right})$ the weight of v
11: **return** v

As described previously, [5] describes a data structure which can count the number of colors in a semi-unbounded query. This algorithm reduces the problem into a standard range searching by creating disjoint boxes from points for every color in such a way that a query will have at most one box from every color inside the query range. These disjoint boxes are found by computing the *skyline* points of a set of points of a specific color and performing a plane sweep algorithm to find all *maximal empty orthants* (see [5] for more details). There is also on-going research on finding the skyline points which could improve the construction time of creating disjoint boxes for every color [9]. In the end, we have a search tree where a query finds a set of canonical nodes with at most one point of every unique color at the leaves of all the canonical nodes. It is clear that we can use the weighted random sampling method on these canonical nodes to independently sample a color from a query and building a kd tree for every unique color allows you to trivially sample a point inside the query range from a sampled color.

To answer colored range sampling with orthogonal ranges in two dimensions, [5] describes a method to use a multi-level search tree with the data structure answering semi-unbounded queries at the lowest level. A two dimensional query range is transformed into two three-sided queries which can be answered by semi-unbounded queries in R^3. There may be duplicate colors between the two queries, so the intersection of colors must be calculated and accommodated for in random sampling. In the range counting problem, this step is easier as the number of colors found intersecting the two queries can be subtracted from the sum of colors found in the two queries. We can extend this algorithm for range sampling by instead finding which colors are found in both queries and sampling with adjusted weights of canonical subsets. This also allows us to use the time-space tradeoff described in [5] which uses a selected tradeoff parameter $1 \leq X \leq n$ to preprocess the intersection of canonical nodes with a count greater than X (heavy canonical nodes) and finds the other intersections during querying. Take every heavy node from the two semi-unbounded search trees and compute the intersecting weight in a matrix. The intersection between two nodes is found by brute-force comparing every leaf from one subtree rooted at the node to all leaves found in the subtree of the other node.

During querying, lookup the intersection between heavy canonical nodes in the matrix to find the intersecting weight. For non-heavy canonical nodes, find the intersecting weight using the brute-force method during the query time. Choose one of the semi-unbounded search trees and temporarily subtract the weights of canonical nodes by the intersecting weights. To sample, perform the weighted random sampling method on the combined list of adjusted canonical nodes from one of the semi-unbounded search trees and the original canonical nodes from the other semi-unbounded search tree. As we only find the intersecting weights of canonical nodes instead of which specific colors are duplicated, only choosing a canonical node to sample a color from is proportional and the sampled color which results from the weighted random traversal down a canonical node is a close approximation to the weights of the respective colors. This is because we only temporarily removed the weight from the canonical node and did not adjust the weights of every node in the subtree removing the weights of specific intersecting colors. If we instead found all intersecting colors instead of weights, we could eliminate this approximation but drastically increase the storage size by the number of colors which can be an order of n. Nonetheless, each sampled color can be used to sample a colored point within the query range by preprocessing a kd tree for every set of points of a unique color.

Algorithm 2 Algorithm for $Query2DTree$

$Input$. A 2D Colored Range Sampling Tree T and a query range $Q = [a, b] \times [c, d]$

$Output$. A list of s sampled color points from P which intersect Q and are randomly selected proportionally to the weights of all colors in P

(* $FindSplitNode$ finds the node of lowest depth which is a parent to the leaf node containing y-coordinate c and leaf node containing y-coordinate d *)

$sp \leftarrow FindSplitNode([c, d])$

$Q_{left} \leftarrow [a, \infty) \times [-b, \infty) \times [c, \infty)$

$Q_{right} \leftarrow [a, \infty) \times [-b, \infty) \times [-d, \infty)$

(* $ReportCanonicalNodes$ will find the list of canonical nodes from a 3 sided tree and query range *)

$CN_{left} \leftarrow ReportCanonicalNodes(aux_{left}(sp), Q_{left})$

$CN_{right} \leftarrow ReportCanonicalNodes(aux_{right}(sp), Q_{right})$

For every canonical node p in CN_{left} and CN_{right}, let the search weight of p be the weight of p

for $cn_{left} \in CN_{left}$ **do**

  for $cn_{right} \in CN_{right}$ **do**

    if $w(cn_{left}) \geq X$ *and* $w(cn_{right}) \geq X$ **then**

      $sw(cn_{right}) \leftarrow sw(cn_{right}) - LookUpMatrix(m(sp), cn_{left}, cn_{right})$

      (* sw notates the search weight of a point and m notates the assigned matrix*)

    else

      $sw(cn_{right}) \leftarrow sw(cn_{right}) - FindIntersection(cn_{left}, cn_{right})$

For every canonical node p in CN_{left} and CN_{right}, compute the key $k = u^{1/sw(p)}$ where $u = random(0, 1)$ and let v be the canonical node with the greatest key

From v traverse down its children until v is a leaf node. At each step, let v be the left or right child with the greatest computed key k.

From the color c of leaf node v, perform uniform random range sampling on a search tree built of only points in P with the color c. Query s samples of colored points and store in list S.

return S

Selecting a tradeoff parameter $X \geq \sqrt{n}$, there exists a data structure which can solve colored orthogonal sampling in two dimensions with near-linear space and near-polylogarithmic time.

III. Experiments

A. Datasets

We utilized real-world, semi-synthetic, and synthetic datasets to run our experiments. The real-world dataset used was the UCI Machine Learning Repository Adult population census dataset [11]. The semi-synthetic population dataset used in [10] and was generated through their PopSim system.

Points were determined through the features of the datasets. Numerical features were used as the dimensions of the points. The colors of points were determined by the union of select categorical features. In the PopSim dataset, 5 colors were separated by race. Similarly in the Adult dataset products of 5 races and sex generated a total of 10 distinct colors (Ex. Black-Female or Asian-Pacific-Islander-Male).

TABLE II
Dataset Features

Datasets	Synthetic	PopSim	Adult
Size	100000	4110806	48842
Numerical Features	1	2	6
Categorical Features	1	1	8
Colors	5	5	10

B. Evaluation

For this experiment, we sought to evaluate the performance and fairness of weighted random range sampling of colored points. Performance was measured by the average runtime of reporting samples. Every sample will be found with a unique randomly generated query range as input to the colored range sampling algorithms. Fairness is evaluated by determining the frequency of reported colors in comparison to the actual frequency of colors in the dataset. For our algorithm to be fair, the proportion of samples from a particular color must match the proportion of the weight of that color. For the synthetic dataset, an exponential function generated the non-negative weights. For the PopSim and Adult Dataset, weights for a specific color were calculated to be inverse to the percentage of that color in the dataset. Where f is the frequency of a color c in a dataset and $0 \leq f \leq 1$, the weight of c was calculated with $w(c) = 1 - f$. This formula simulates increasing the representation of underrepresented categories when being sampled.

IV. Results

Fig. 1 shows the distribution of colors in the real-world Adult dataset while Fig. 2, and Fig. 3 are the results of the Dependent Range Sampling algorithm tested on the Adult dataset. As expected of an independent weighted random sampling algorithm without considerations to varying frequencies of colors, there is a weak correlation between the frequency of colors in samples and the weights of colors as seen in Fig. 3. Instead, the frequency of color samples more closely follows the original distribution of colors in the Adult dataset shown in Fig. 1.

Fig. 4 shows the uniform distribution of colors in the synthetic dataset set while Fig. 5 and Fig. 6 show the results for the 1D Colored Range Sampling algorithm on the synthetic dataset. Disregarding the distribution of Fig. 4, Fig. 5 and Fig. 6 demonstrate how the color samples from the 1D Range Sampling algorithm correlate strongly with the weight of colors. A near perfect perfect overlap between the weight proportion and sampling color frequency is found in Fig. 6.

Fig. 7 shows the distribution of colors in the semi-synthetic Popsim dataset while Fig. 8 and Fig. 9 are the results of the 2D Colored Semi-Unbounded Range Sampling algorithm on the PopSim dataset. The algorithm generated color samples with

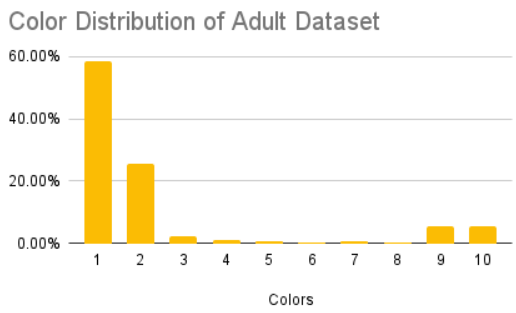

Fig. 1. Adult Dataset

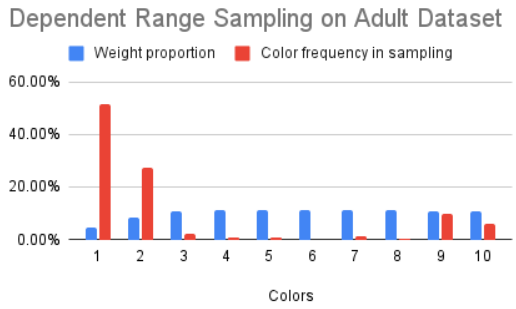

Fig. 2. 6D Dependent Range Sampling

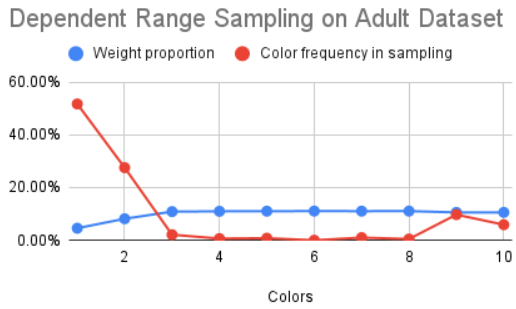

Fig. 3. 6D Dependent Range Sampling

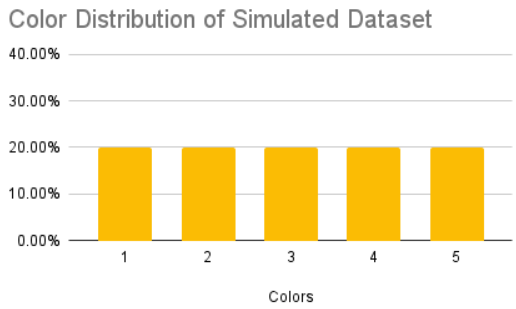

Fig. 4. Simulated dataset

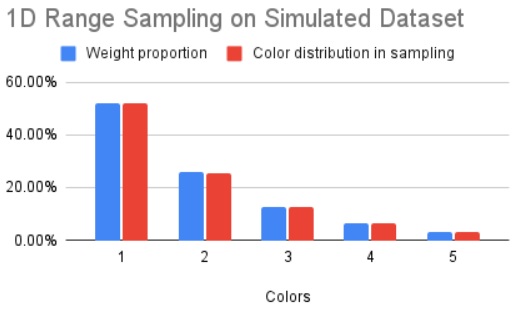

Fig. 5. 1D Range Sampling

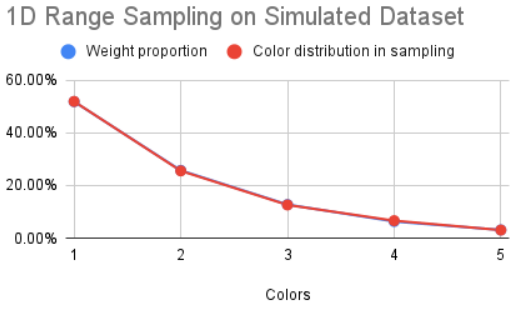

Fig. 6. 1D Range Sampling

frequencies strongly correlated to the proportions of weights as clearly shown in Fig. 8 and Fig. 9.

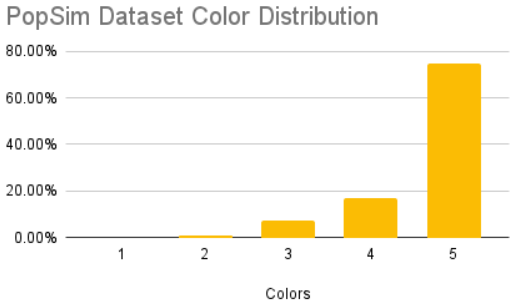

Fig. 7. PopSim Dataset

Fig. 10 and Fig. 11 show the results of the 2D Colored Orthogonal Range Sampling algorithm on the PopSim dataset. See Fig. 7 to see the color distribution of the dataset. Similar to the semi-unbounded range sampling algorithm, a strong relationship is found between the frequencies of colors in the sampling and the proportions of weights.

Either of the real-world or semi-synthetic dataset results illustrate an increase in fairness of sampled colors as a result of weight colors. Frequency of sampled colors was not observed to be near uniform as expected, and the fairness of samples chosen were not predictable. A ongoing factor in the distribution of samples reported is the distributions of colors and points in the dataset.

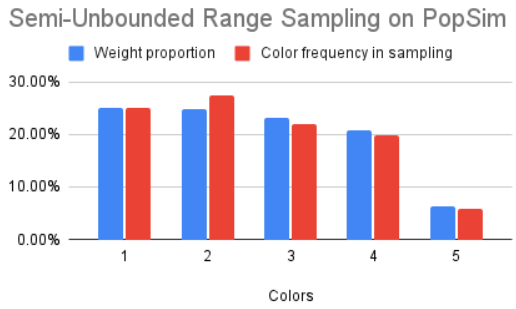

Fig. 8. 2D Semi-Unbounded Range Sampling

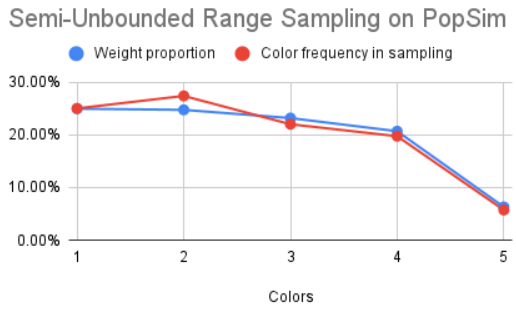

Fig. 9. 2D Semi-Unbounded Range Sampling

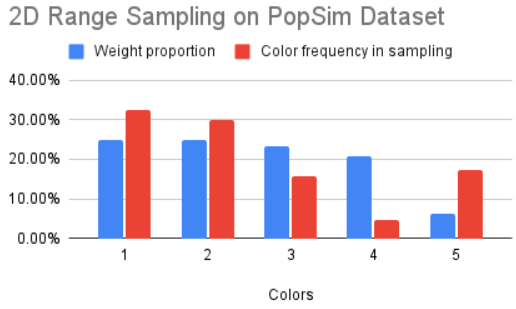

Fig. 10. 2D Orthogonal Range Sampling

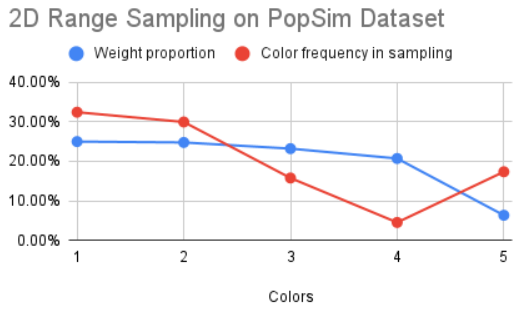

Fig. 11. 2D Orthogonal Range Sampling

Table III presents the query time of the algorithms tested. The algorithms were implemented in the Python coding language and ran on an Intel i7 CPU.

TABLE III
TABLE TYPE STYLES

Dataset	Avg Query Time	Samples
Dependent	10.021279ms	10000
1D	0.01406372ms	10000
Semi-Unbounded	0.08749976ms	10000
2D	0.0969685ms	1000

Note: The average query time for the 2D algorithm was found on a randomly selected subset of the PopSim dataset of 1000 points instead of the entire dataset.

V. CONCLUSION

We have shown two solutions for colored orthogonal range sampling for spaces in R^1 and R^2 which have near-linear space and near-polylogarithmic time as well as established the connections between solutions in range counting and sampling. Expanded research in range counting may find reductions from the colored problem to standard range searching that may allow for better space or query time requirements for weighted range sampling. As shown by the decomposition of a two dimensional orthogonal range sampling into sampling from multiple semi-unbounded search trees, more research in semi-unbounded cases may find new solutions in higher dimensional space. The same goes for the potential of data structures in lower dimensions for multi-level data structures for higher dimensions. Other future research involves finding to what degree does our solution in two dimensions approximate sampling proportionally to the weights of colors as well as searching for newer reduction methods that don't need approximations. More research for different solutions of weighted random sampling which closely approximate the weights of different categories by a parameter may also lead to better space or time requirements.

APPENDIX

TABLE IV
NOTATIONS

P	Set of points in dataset
Q	Query range
P_q	Set of points in dataset intersecting query range
C	Set of colors in dataset
$w(p)$	Weight of a point p
n	Number of points in dataset
d	Number of dimensions in dataset
k	Number of colors in dataset intersecting query range
s	Number of samples in query
X	Time-space tradeoff parameter

REFERENCES

[1] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," Information Processing Letters, vol. 97, no. 5, pp. 181–185, Mar. 2006, doi: https://doi.org/10.1016/j.ipl.2005.11.003.

[2] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, Computational Geometry. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. doi: https://doi.org/10.1007/978-3-540-77974-2.

[3] P. Gupta, Ravi Janardan, Saladi Rahul, and M. Smid, "Computational Geometry: Generalized (or Colored) Intersection Searching," pp. 1043–1058, Mar. 2018, doi: https://doi.org/10.1201/9781315119335-67.

[4] P. Gupta, R. Janardan, and M. Smid, "Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization," Journal of Algorithms, vol. 19, no. 2, pp. 282–317, Sep. 1995, doi: https://doi.org/10.1006/jagm.1995.1038.

[5] H. Kaplan, N. Rubin, M. Sharir, and Elad Verbin, "Counting colors in boxes," pp. 785–794, Jan. 2007, doi: https://doi.org/10.5555/1283383.1283467.

[6] Y. Tao, "Algorithmic Techniques for Independent Query Sampling," Association for Computing Machinery, pp. 129–138, Jun. 2022, doi: https://doi.org/10.1145/3517804.3526068.

[7] P. Gupta, R. Janardan, and M. Smid, "Algorithms for generalized halfspace range searching and other intersection searching problems," Computational Geometry, vol. 6, no. 1, pp. 1–19, Apr. 1996, doi: https://doi.org/10.1016/0925-7721(95)00012-7.

[8] T. M. Chan, Q. He, and Y. Nekrich, "Further Results on Colored Range Searching," arXiv (Cornell University), Jan. 2020, doi: https://doi.org/10.48550/arxiv.2003.11604.

[9] Saladi Rahul and Ravi Janardan, "Algorithms for range-skyline queries," Nov. 2012, doi: https://doi.org/10.1145/2424321.2424406.

[10] K. D. Nguyen, N. Shahbazi, and A. Asudeh, "PopSim: An Individual-level Population Simulator for Equitable Allocation of City Resources," arXiv.org, Apr. 25, 2023. https://arxiv.org/abs/2305.02204 (accessed Apr. 18, 2024).

[11] "UCI Machine Learning Repository," archive.ics.uci.edu. https://archive.ics.uci.edu/dataset/2/adult (accessed Nov. 16, 2023).

[12] X. Hu, M. Qiao, and Y. Tao, "Independent range sampling," Jun. 2014, doi: https://doi.org/10.1145/2594538.2594545.

[13] Peyman Afshani and J. M. Phillips, "Independent Range Sampling, Revisited Again," arXiv (Cornell University), p. 13, Mar. 2019, doi: https://doi.org/10.4230/lipics.socg.2019.4.

[14] Y. Nekrich, "Efficient range searching for categorical and plain data," ACM Transactions on Database Systems, vol. 39, no. 9, pp 1-21, Jan. 2014, doi: https://doi-org.hmlproxy.lib.csufresno.edu/10.1145/2543924.

[15] T. M. Chan, Y. Nekrich, "Better data structures for colored orthogonal range reporting," Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp 627-636, Jan. 2020, doi: https://doi.org/10.1137/1.9781611975994.38.