# Dynamic Range Majority Data Structures*

Amr Elmasry[1], Meng He[2], J. Ian Munro[3], and Patrick K. Nicholson[3]

[1] Department of Computer Science, University of Copenhagen, Denmark
[2] Faculty of Computer Science, Dalhousie University, Canada
[3] David R. Cheriton School of Computer Science, University of Waterloo, Canada,
elmasry@diku.dk, mhe@cs.dal.ca, {imunro, p3nichol}@uwaterloo.ca

**Abstract.** Given a set $P$ of $n$ coloured points on the real line, we study the problem of answering range $\alpha$-majority (or "heavy hitter") queries on $P$. More specifically, for a query range $Q$, we want to return each colour that is assigned to more than an $\alpha$-fraction of the points contained in $Q$. We present a new data structure for answering range $\alpha$-majority queries on a dynamic set of points, where $\alpha \in (0, 1)$. Our data structure uses $O(n)$ space, supports queries in $O((\lg n)/\alpha)$ time, and updates in $O((\lg n)/\alpha)$ amortized time. If the coordinates of the points are integers, then the query time can be improved to $O(\lg n/(\alpha \lg \lg n))$. For constant values of $\alpha$, this improved query time matches an existing lower bound, for any data structure with polylogarithmic update time. We also generalize our data structure to handle sets of points in $d$-dimensions, for $d \geq 2$, as well as dynamic arrays, in which each entry is a colour.

## 1 Introduction

Many problems in computational geometry deal with point sets that have information encoded as colours assigned to the points. In this paper, we design dynamic data structures for the *range $\alpha$-majority problem*, in which we want to report colours that appear *frequently* within an axis-aligned query rectangle. This problem is useful in database applications in which we would like to know typical attributes of the data points in a query range [14, 15]. For the one-dimensional case, where the points represent time stamps, this problem has data mining applications for network traffic logs, similar to those of coloured range counting [10].

Formally, we are given a set, $P$, of $n$ points, where each point $p \in P$ is assigned a colour $c$ from a set, $C$, of colours. Let $\mathrm{col}(p) = c$ denote the colour of $p$. We are also given a fixed parameter $\alpha \in (0, 1)$, that defines the threshold for determining whether a colour is to be considered frequent. Our goal is to design a *dynamic range $\alpha$-majority data structure* that can perform the following operations:

- Query($Q$): We are given an axis-aligned hyper-rectangle $Q$ as a query. Let $P(Q)$ be the set $\{p \mid p \in Q, p \in P\}$, and $P(Q, c)$ be the set $\{p \mid p \in P(Q), \mathrm{col}(p) = c\}$. The answer to the query $Q$ is the set of colours $M$

* This work was supported by NSERC and the Canada Research Chairs Program.

such that for each colour $c \in M$, $|P(Q,c)| > \alpha|P(Q)|$, and for all $c \notin M$, $|P(Q,c)| \leq \alpha|P(Q)|$. We refer to a colour $c \in M$ as an *$\alpha$-majority* for $Q$, and the query $Q$ as an *$\alpha$-majority query*. When $\alpha = \frac{1}{2}$, the problem is to identify the majority colour in $Q$, if such a colour exists.
- INSERT$(p, c)$: Insert a point $p$ with colour $c$ into $P$.
- DELETE$(p)$: Remove the point $p$ from $P$.

### 1.1 Previous Work

*Static and Dynamic $\alpha$-Majority:* In all of the following results, the threshold $\alpha \in (0, 1)$ is fixed at construction time, rather than given for each query.

Karpinski and Nekrich [14] studied the problem of answering range $\alpha$-majority queries, which they call *coloured $\alpha$-domination* queries. In the static case, they gave an $O(n/\alpha)$ space data structure that supports one-dimensional queries in $O((\lg n \lg\lg n)/\alpha)$ time, and an $O((n \lg\lg n)/\alpha)$ space data structure that supports queries in $O((\lg n)/\alpha)$ time. In the dynamic case, they gave an $O(n/\alpha)$ space data structure for one-dimensional queries which supports queries and insertions in $O((\lg^2 n)/\alpha)$ time, and deletions in $O((\lg^2 n)/\alpha)$ amortized time. They also gave an alternative $O((n \lg n)/\alpha)$ space data structure which supports queries and insertions in $O((\lg n)/\alpha)$ time, and deletions in $O((\lg n)/\alpha)$ amortized time. For points in $d$-dimensions, they gave a static $O((n \log^{d-1} n)/\alpha)$ space data structure that supports queries in $O((\log^d n)/\alpha)$ time, as well as a dynamic $O((n \log^{d-1} n)/\alpha)$ space data structure that supports queries and insertions in $O((\log^{d+1} n)/\alpha)$ time, and deletions in $O((\log^{d+1} n)/\alpha)$ amortized time.

Recently, Durocher et al. [8] described a static $O(n(\lg(1/\alpha) + 1))$ space data structure that answers range $\alpha$-majority queries in an array in $O(1/\alpha)$ time.

Lai et al. [15] studied the dynamic problem, using the term *heavy-hitters* instead of $\alpha$-majorities. They presented a dynamic data structure based on sketching, which provides an approximate solution with probabilistic guarantees for constant values of $\alpha$. For one-dimension their data structure uses $O(hn)$ space and supports queries and updates in $O(h \log n)$ time, where the parameter $h = O(\frac{\lg m}{\varepsilon} \lg(\frac{\lg m}{\alpha\delta}))$ depends on the threshold $\alpha$, the approximation factor $\varepsilon$, the total number of colours $m$, and the probability of failure $\delta$. They also note that the space can be reduced to $O(n)$, at the cost of increasing the query time to $O(h \lg n + h^2)$. Thus, for constant values of $\varepsilon$, $\delta$, and $\alpha$, their data structure uses $O(n)$ space and has $O((\lg n \lg\lg n)^2)$ query and update time in the worst case when $\lg m = \Omega(\lg n)$. Another approximate data structure based on sketching was proposed by Wei and Yi [17]. Their data structure uses linear space, answers queries in $O(\log n + 1/\varepsilon)$ time, and may return false positives with relative frequency between $\alpha - \varepsilon$ and $\alpha$. The cost of updates is $O(\mu \log n \log(1/\varepsilon))$, where $\mu$ is the cost of updating the sketches.

Finally, a cell-probe lower bound of Husfeldt and Rauhe [12, Prop. 11] implies that $t_q = \Omega(\lg n/(\lg(t_u b \lg n)))$, for any dynamic range $\alpha$-majority data structure with query time $t_q$ and update time $t_u$, when $\alpha$ is a constant and our machine has cell size $b$.

*Other Related Work:* Several other results exist for finding $\alpha$-majorities in the streaming model [16, 6, 13]. De Berg and Haverkort [5] studied the problem of reporting $\tau$-*significant* colours. For this problem, the goal is to output all colours $c$ such that at least a $\tau$-fraction of *all* of the points with colour $c$ lie in the query rectangle.

There are other data structure problems that also deal with coloured points. In *coloured range reporting problems*, we are interested in reporting the set of distinct colours assigned to the points contained in an axis-aligned rectangle. Similarly, in *the coloured range counting problem* we are interested in returning the number of such distinct colours. Gupta et al. [11], Bozanis et al. [4], and, more recently, Gagie and Kärkkäinen [10] studied these problems and presented several interesting results.

### 1.2 Our Results

In this paper we present new data structures for the dynamic range $\alpha$-majority problem in the word-RAM model with word size $\Omega(\lg n)$, where $n$ is the number of points in the set $P$, and $\alpha \in (0, 1)$. Our results are summarized and compared to the previous best results in Table 1. The *input* column indicates the type of data we are considering. We use *points* to denote a set of points on a line with real-valued coordinates that we can compare in constant time, *integers* to denote a set of points on a line with word sized integer coordinates, and *array* to denote that the input is to be considered a dynamic array, where the positions of the points are in rank space.

| Source | Input | Space | Query | Insert | Delete |
|---|---|---|---|---|---|
| [14, Thm. 3] | points | $O(\frac{n}{\alpha})$ | $O(\frac{\lg^2 n}{\alpha})$ | $O(\frac{\lg^2 n}{\alpha})$ | $O(\frac{\lg^2 n}{\alpha})$* |
| [14, Thm. 3] | points | $O(\frac{n \lg n}{\alpha})$ | $O(\frac{\lg n}{\alpha})$ | $O(\frac{\lg n}{\alpha})$ | $O(\frac{\lg n}{\alpha})$* |
| New | points | $O(n)$ | $O(\frac{\lg n}{\alpha})$ | $O(\frac{\lg n}{\alpha})$* | $O(\frac{\lg n}{\alpha})$* |
| New | integers | $O(n)$ | $O(\frac{\lg n}{\alpha \lg \lg n})$ | $O(\frac{\lg n}{\alpha})$* | $O(\frac{\lg n}{\alpha})$* |
| New | array | $O(n)$ | $O(\frac{\lg n}{\alpha \lg \lg n})$ | $O(\frac{\lg^3 n}{\alpha \lg \lg n})$* | $O(\frac{\lg n}{\alpha})$* |

**Table 1.** Comparison of the results in this paper to the previous best results. For the entries marked with "*" the running times are amortized.

Our results improve upon previous results in several ways. Most noticeably, all of our data structures require linear space. In order to provide fast query and update times for our linear space structures, we prove several interesting properties of $\alpha$-majority colours. We note that the lower bound from Section 1.1 implies that, for constant values of $\alpha$, an $O(\lg n / \lg \lg n)$ query time for integer point sets is optimal for any data structure with polylogarithmic update time, when the word size is $b = \Theta(\lg n)$. Our data structure for points on a line with integer coordinates achieves this optimal query time. Our data structures can

also be generalized to handle $d$-dimensional points, improving upon previous results [14].

*Road Map* In Section 2 we present a dynamic range $\alpha$-majority data structure for points in one dimension. In Section 3 we show how to speed up the query time of our data structure in the case where the points have integer coordinates. Finally, in Section 4 we generalize our one dimensional data structures to higher dimensions. We defer the details of our dynamic array data structure to the full version of this paper.

*Assumptions About Colours* In the following sections, we assume that we can compare colours in constant time. In order to support a dynamic set of colours, we employ the techniques described by Gupta et al. [11]. These techniques allow us to maintain a mapping from the set of colours to integers in the range $[1, 2n]$, where $n$ is the number of points *currently* in our data structure. This allows us to index into an array using a colour in constant time.

For the dynamic problems discussed, this mapping is maintained using a method similar to global rebuilding to ensure that the integer identifiers of the colours do not grow too large [11, Section 2.3]. When a coloured point is inserted, we must first determine whether we have already assigned an integer to that colour. By storing the set of known colours in a balanced binary search tree, this can be checked in $O(\lg n_c)$ time, where $n_c$ is the number of distinct colours currently assigned to points in our data structure. Therefore, from this point on, we assume that we are dealing with integers in the range $[1, 2n]$ when we discuss colours.

## 2 Dynamic Data Structures in One Dimension

In one-dimension we can interchange the idea of points and $x$-coordinates in $P$, since they are equivalent. Depending on the context we may use either term. Our data structure is a modification of the approach of Karpinski and Nekrich [14], and we prove several interesting combinatorial properties in order to provide more efficient support for queries and updates.

We create a weight-balanced B-tree [2] $T$ with branching parameter 8 and leaf parameter 1 such that each leaf represents an $x$-coordinate in $P$. From left to right the leaves are sorted in ascending order of the $x$-coordinate that they represent. Let $T_u$ be the subtree rooted at node $u$. Each internal node $u$ in the tree represents a range $R_u = [x_s, x_e]$, where $x_s$ and $x_e$ are the $x$-coordinates represented by the leftmost and rightmost leaves in $T_u$, respectively. We number the levels of the tree $T$ $0, 1, ..., \lg n$ from top to bottom. If a node is $h$ levels above the leaf level, we say that this node is of *height* $h$. By the properties of weight-balanced B-trees, the range represented by an internal node of height $h$ contains at least $8^h/2$ (except the root) and at most $2 \times 8^h$ points, and the degree of each internal node is at least 2 and at most 32.

### 2.1 Supporting Queries

Given a query $Q' = [x'_a, x'_b]$, we perform a top-down traversal on $T$ to map $Q'$ to the range $Q = [x_a, x_b]$, where $x_a$ and $x_b$ are the points in $P$ with $x$-coordinates that are the successor and the predecessor of $x'_a$ and $x'_b$ in $P$, respectively. We call the query range $Q$ *general* if $Q$ is not represented by a single node of $T$. We first define the notion of *representing* a general query range by a set of nodes:

**Definition 1.** *Given a general query range $Q = [x_a, x_b]$, $T$ partitions $Q$ into a set of blocks, in which each block is represented by an node of $T$, and the range represented by the parent of each such node is not entirely contained in $Q$. The set, $I$, of all the nodes representing these blocks is the set of nodes of $T$* representing *$Q$.*

For each node $v \in T$, we keep a list, $C(v)$, of $k$ *candidate* colours, i.e., the $k$ most frequent colours in the range $R_v$ represented by $v$ (breaking ties arbitrarily); we will fix $k$ later. Let $L = \cup_{v \in I} C(v)$. For each colour $c$, we keep a separate range counting data structure, $F_c$, containing all of the points of colour $c$ in $P$, and also a range counting data structure, $F$, containing all of the points in $P$. Let $m$ be the total number of points in $[x_a, x_b]$, which can be answered by $F$. For each $c \in L$, we query $F_c$ with the range $[x_a, x_b]$ letting $f$ be the result. If $f > \alpha m$, then we report $f$.

It is clear that $I$ contains at most $O(\lg n)$ nodes. Furthermore, if a colour $c$ is an $\alpha$-majority for $Q$, then it must be an $\alpha$-majority for at least one of the ranges in $I$ [14, Observation 1]. If we set $k = \lceil 1/\alpha \rceil$ and store $\lceil 1/\alpha \rceil$ colours in each internal node as candidate colours, then, by the procedure just described, we will perform a range counting query on $O((\lg n)/\alpha)$ colours. If we use balanced search trees for our range counting data structures, then this takes $O((\lg^2 n)/\alpha)$ time overall. However, in the sequel we show how to improve this query time by exploiting the fact that the blocks in $I$ that are closer to the root of $T$ contain more points in the ranges that they represent.

We now prove useful properties of a general query range $Q$ and the set, $I$, of nodes representing it in Lemmas 1, 2, 3, and 4. In these lemmas, $m$ denotes the number of points in $Q$, and $i_1, i_2, ...$ denote the distinct values of the heights of the nodes in $I$, where $i_1 > i_2 > ... \geq 0$. We first give an upper bound on the number of points contained in the ranges represented by the nodes of $I$ of a given height:

**Lemma 1.** *The total number of points in the ranges represented by all the nodes in $I$ of height $i_j$ is less than $m \times \min(1, 8^{1-j} \times 31)$.*

*Proof.* Since $Q$ is general and it contains at least one node of height $i_1$, $m$ is greater than the minimum number of points that can be contained in a node of height $i_1$, which is $8^{i_1}/2$. The nodes of $I$ whose height is $i_j$ must have at least one sibling that is not in $I$, and the number of points contained in the interval represented by this sibling is at least $8^{i_j}/2$. Therefore, the number, $m_j$, of points in the ranges represented by the nodes of $I$ at level $i_j$ is at most $2 \times 8^{i_j+1} - 8^{i_j}/2 = 8^{i_j} \times (31/2)$. Thus, $m_j/m < 8^{i_j - i_1} \times 31 < 8^{1-j} \times 31$. □

We now use the above lemma to bound the number of points whose colours are not among the candidate colours stored in the corresponding nodes in $I$.

**Lemma 2.** *Given a node $v \in I$ of height $i_j$ and a colour $c$, let $n_v^{(c)}$ denote the number of points with colour $c$ in the range $R_v$, if $c$ is not among the first $k_j = k/2^{j-1}$ most frequent candidate colours in the candidacy list of $v$, and $n_v^{(c)} = 0$ otherwise. Then $\sum_{v \in I} n_v^{(c)} < \frac{5.59m}{k+1}$.*

*Proof.* If $c$ is not among the first $k_j$ candidate colours stored in $v$, then the number of points with colour $c$ in $R_v$ is at most $1/(k_j + 1)$ times the number of points in $R_v$. Thus,

$$\begin{aligned}\sum_{v \in I} n_v^{(c)} &< \sum_{j=1}^{2} \frac{1}{k_j + 1} \times m + \sum_{j \geq 3} \frac{1}{k_j + 1} \times m \times 31 \times 8^{1-j} \\ &< \frac{m}{k+1} \times (1 + 2 + 31 \times (\frac{2^2}{8^2} + \frac{2^3}{8^3} + \dots)) \\ &< \frac{5.59m}{k+1}\end{aligned}$$

□

We next consider the nodes in $I$ that are closer to the leaf levels. Let $I_t$ denote the nodes in $I$ that are at one of the top $t = \lceil \frac{\lg(\frac{1}{\alpha})}{3} + 2.05 \rceil$ (not necessarily consecutive) levels of the nodes in $I$. We prove the following property:

**Lemma 3.** *The number of points contained in the ranges represented by the nodes in $I \setminus I_t$ is less than $\alpha m/2$.*

*Proof.* By Lemma 1, the number of points contained in the ranges represented by the nodes in $I \setminus I_t$ is less than:

$$\begin{aligned}m \times 31 \times \sum_{j \geq t+1} 8^{1-j} &< m \times 31 \times (\frac{1}{8^t} + \frac{1}{8^{t+1}} + ...) \\ &< m \times 31 \times \frac{8}{7} \times \frac{1}{8^t}\end{aligned}$$

Since $t \geq \frac{\lg(\frac{1}{\alpha})}{3} + 2.05$, the above value is less than $\alpha m/2$. □

With the above lemmas, we can now choose an appropriate value for $k$ to achieve the following property that is critical to achieve improved query time:

**Lemma 4.** *When $k = \lceil \frac{11.18}{\alpha} \rceil - 1$, any $\alpha$-majority, $c$, of $Q$ is among the union of the first $\lceil k/2^{j-1} \rceil$ candidates stored in each node of height $i_j$ representing a range in $I_t$.*

*Proof.* The total number of points with colour $c$ in the ranges represented by the nodes in $I \setminus I_t$ is less than $\alpha m/2$ by Lemma 3. By Lemma 2 and our choice for the value of $k$, less than $\alpha m/2$ points in the ranges represented by the nodes in $I_t$ for which $c$ is not a candidate can have colour $c$. The lemma thus follows. □

For each node $v \in T$, we keep a semi-ordered list of the $k$ candidate colours in the range $R_v$ represented by $v$. The order on the colours for any candidacy list is maintained such that the most frequent $k/2^{j-1}$ colours come first, for all $j = 2, 3, \ldots$, arbitrarily ordered within their positions. Note that such a semi-ordering can be obtained in linear time by repeated median findings. Thus, by setting $k = \lceil 11.18/\alpha \rceil - 1$, Lemma 4 implies that the colours that we have checked are the only possible $\alpha$-majority colours for the query. Furthermore, Lemma 3 implies that we need only check the nodes on the top $O(\lg(1/\alpha))$ levels in $I$. Let $I_t$ denote the set of nodes in these levels. We present the following lemma:

**Lemma 5.** *The data structures described in this section occupy $O(n)$ words, and can be used to answer a range $\alpha$-majority query in $O((\lg n)/\alpha)$ time with the help of an additional array of size $2n$.*

*Proof.* To support $\alpha$-majority queries, we only consider the nontrivial case in which the query range $Q$ is general. By Lemma 4, the $\alpha$-majorities can be found by examining the first $\lceil k/2^{j-1} \rceil$ candidate colours stored in each node representing a range in $I_t$. Thus, there are at most $O(\lceil \frac{1}{\alpha} \rceil + \lceil \frac{1}{2\alpha} \rceil + \lceil \frac{1}{4\alpha} \rceil + \ldots + \lceil \frac{1}{2^{t-1}\alpha} \rceil) = O(\frac{1}{\alpha})$ relevant colours to check. Let $L_t$ denote the set of these colours. For each $c \in L_t$ we query our range counting data structures $F_c$ and $F$ in $O(\lg n)$ time to determine whether $c$ is an $\alpha$-majority. Thus, the overall query time is $O((\lg n)/\alpha)$.

There are $O(n)$ nodes in the weight-balanced B-tree. Therefore, we would expect the space to be $O(n/\alpha)$ words, since each node stores $O(1/\alpha)$ colours. We use a pruning technique on the lower levels of the tree in order to reduce the space to $O(n)$ words overall. If a node $u$ contains at most $1/\alpha$ points, then we need not store $C(u)$, since every colour in $T_u$ is an $\alpha$-majority for $R_u$. Instead, during a query, we can do a linear scan of the leaves of $T_u$ in order to determine the unique colours. To make this efficient, we require an array $D$ of size $2n$ to count the frequencies of the colours in $R_u$. As mentioned in Section 1.2, we can map a colour into an index of the array $D$, which allows us to increment a frequency counter in $O(1)$ time. Thus, we can extract the unique colours in $R_u$ in $O(|T_u|) = O(\frac{1}{\alpha})$ time. The number of nodes whose subtrees have more than $1/\alpha$ leaves is $O(n\alpha)$. Thus, we store $O(k) = O(1/\alpha)$ words in $O(n\alpha)$ nodes, and the total space used by our B-tree $T$ is $O(n)$ words. The only other data structures we make use of are the array $D$ and the range counting data structures $F$ and $F_c$ for each $c \in C$, and together these occupy $O(n)$ words. □

### 2.2 Supporting Updates

We now establish how much time is required to maintain the lists $C(v)$ in node $v$ under insertions and deletions. We begin by observing that it is not possible to lazily maintain the list of the top $k = \lceil \frac{11.18}{\alpha} \rceil - 1$ most frequent colours in

each range: many of these colours could have low frequencies, and the list $C(v)$ would have to be rebuilt after very few insertions or deletions. To circumvent this problem, we relax our requirements on what is stored in $C(v)$, only guaranteeing that *all* of the $\beta$-majorities of range $R_v$ must be present in $C(v)$, where $\beta = \lceil \frac{11.18}{\alpha} \rceil^{-1}$. With this alteration, we can still make use of the lemmas from the previous section, since they depend only on the fact that there are no colours $c \notin C(v)$ with frequency greater than $\beta|T_v|$. The issue now is how to maintain the $\beta$-majorities of $R_v$ during insertions and deletions of colours.

Karpinski and Nekrich noted that, if we store the $(\beta/2)$-majorities for each node $v$ in $T$, then it is only after $|T_v|\beta/2$ deletions that we must rebuild $C(v)$ [14]. For the case of insertions and deletions, their data structure performs a range counting query at each node $v$ along the path from the root of $T$ to the leaf representing the inserted or deleted colour $c$. This counting query is used to determine if the colour $c$ should be added to, or removed from, the list $C(v)$.

In contrast, our strategy is to be lazy during insertions and deletions, waiting as long as possible before recomputing $C(v)$, and to avoid performing range counting queries for each node in the update path. We provide a tighter analysis (to constant factors) of how many insertions and deletions can occur before the list $C(v)$ must be rebuilt. One caveat is that the results in this section only apply when $\beta \in (0, \frac{1}{2}]$. However, since $\alpha < 1$, our choice of $\beta$ satisfies this condition. We present the following lemma, deferring its proof until a later version of this paper:

**Lemma 6.** *Suppose the list $C(v)$ for node $v$ contains the $\lceil \frac{1-\beta+\sqrt{1-\beta}}{\beta} \rceil$ most frequent colours in the range $R_v$, breaking ties arbitrarily. For $\beta \in (0, \frac{1}{2}]$, this value is upper bounded by $\lceil \frac{2}{\beta} \rceil$. Let $\ell$ be the number of points contained in $R_v$. Only after $\lceil \frac{\beta\ell}{\sqrt{1-\beta}(1+\sqrt{1-\beta})} \rceil \geq \lceil \frac{\beta\ell}{2} \rceil$ insertions or deletions into $T_v$ can a colour $c \notin C(v)$ become a $\beta$-majority for the range spanned by node $v$.*

By Lemma 6, our lazy updating scheme only requires each list $C(v)$ to have size $\lceil \frac{1-\beta+\sqrt{1-\beta}}{\beta} \rceil = O(1/\alpha)$. This leads to the following theorem:

**Theorem 1.** *Given a set $P$ of $n$ points in one dimension and a fixed $\alpha \in (0, 1)$, there is an $O(n)$ space data structure that supports range $\alpha$-majority queries on $P$ in $O((\lg n)/\alpha)$ time, and insertions and deletions in $O((\lg n)/\alpha)$ amortized time.*

*Proof.* Query time follows from Lemma 5. In order to get the desired space, we combine Lemmas 5 and 6, implying that each list $C(v)$ containing $O(1/\alpha)$ colours. This allows us to use the same pruning technique described in Lemma 5 in order to reduce the space to $O(n)$.

When an update occurs, we follow the path from the root of $T$ to the updated node $u$. Suppose, without loss of generality, the update is an insertion of a point of colour $c$. For each vertex $v$ in the path, if $v$ contains a list $C(v)$ and $c$ is in $C(v)$, we increment the count of colour $c$, which takes $O(1/\alpha)$ time. We also increment the counter for $v$ that keeps track of the number of updates into $T_v$

that have occurred since $C(v)$ was rebuilt. Thus, modifying the counters along the path requires $O((\lg n)/\alpha)$ time in the worst case.

We now look at the costs of maintaining the lists $C(v)$. The list $C(v)$ can be rebuilt in $O(|T_v|)$ time, using the array $D$. Note that $D$ can be maintained under updates using the same scheme described in Section 1.2. First, we use $D$ to compute the frequency of all the colours in $R_v$ in $O(|T_v|)$ time. Since there are at most $O(|T_v|)$ colours, we can select the top $k$ most frequent colours in $O(|T_v|)$ time, where $k$ is the value from Lemma 6. We can then enforce the necessary semi-ordering on this list in $O(k) = O(\frac{1}{\alpha})$ time. Thus, each leaf in $T_v$ pays $O(1)$ cost every $O(|T_v|\alpha)$ insertions, or $O(1/\alpha)$ cost for $O(|T_v|)$ insertions. Since each update may cause $O(\lg n)$ lists to be rebuilt, this increases the cost to $O((\lg n)/\alpha)$ amortized time per update.

We make use of standard local rebuilding techniques to keep the tree balanced, rebuilding the lists in nodes that are merged or split during an update. Since a node $v$ will only be merged or split after $O(|T_v|)$ updates by the properties of weight-balanced B-trees, these local rebuilding operations require $O(\lg n)$ amortized time. Thus, updates require $O((\lg n)/\alpha)$ amortized time overall, and are dominated by the costs of maintaining the lists $C(v)$ in each node $v$. □

## 3 Speedup for Integer Coordinates

We now describe how to improve the query time of the data structure from Theorem 1 from $O((\lg n)/\alpha)$ to $O(\lg n/(\alpha \lg \lg n))$ for the case in which the $x$-coordinates of the points in $P$ are word-sized integers.

To accomplish this goal, we require an improved one dimensional range counting data structure, which we get by combining two existing data structures. The fusion tree of Fredman and Willard [9] is an $O(n)$ space data structure that supports predecessor and successor queries, insertion, and deletion in $O(\lg n/\lg \lg n)$ time. The list indexing data structure of Dietz [7] uses $O(n)$ space and supports rank queries, insertion, and deletion in $O(\lg n/\lg \lg n)$ time. In Andersson et al. [1], it was observed that these data structures could be combined to support dynamic one-dimensional range counting queries in $O(\lg n/\lg \lg n)$ time per operation. We refer to this data structure as an *augmented fusion tree*.

In order to achieve $O(\lg n/(\alpha \lg \lg n))$ query time, we implement all of our range counting data structures as augmented fusion trees: i.e., the data structures $F$, and $F_c$ for each $c \in C$. Immediately we get that we can perform a query in $O(\lg n/(\alpha \lg \lg n) + \lg n)$ time: $O(\lg n/(\alpha \lg \lg n))$ time for the range counting queries, and $O(\lg n)$ time to find the nodes in $I_t$. To remove the additive $O(\lg n)$ term, we modify our weight balanced B-tree to support dynamic lowest common ancestor queries. We defer the details to a later version of this paper, and present the following theorem:

**Theorem 2.** *Given a set $P$ of $n$ points in one dimension with integer coordinates and a fixed $\alpha \in (0, 1)$, there is an $O(n)$ space data structure that supports range $\alpha$-majority queries on $P$ in $O(\lg n/(\alpha \lg \lg n))$ time, and both insertions and deletions into $P$ in $O((\lg n)/\alpha))$ amortized time.*

## 4 Dynamic Data Structures in Higher Dimensions

As an immediate application of Theorems 1 and 2, we can generalize our data structure to higher dimensions in the same manner as Karpinski and Nekrich [14], who used standard range tree techniques [3]. We present the following theorem:

**Theorem 3.** *Given a set $P$ of $n$ points in $d$ dimensions ($d \geq 1$) and a fixed $\alpha \in (0, 1)$, there is an $O(n \lg^{d-1} n)$ space data structure that supports range $\alpha$-majority queries on $P$ in $O((\lg^d n)/\alpha)$ time, and insertions and deletions into $P$ in $O((\lg^d n)/\alpha)$ amortized time. If the points have integer coordinates the query can be improved to $O((\lg^d n)/(\alpha \lg \lg n))$ time, without changing the update time.*

## References


1. Andersson, A., Miltersen, P., Thorup, M.: Fusion trees can be implemented with $AC^0$ instructions only. Theoretical Computer Science 215(1-2), 337–344 (1999)
2. Arge, L., Vitter, J.S.: Optimal external memory interval management. SIAM J. Comput. 32(6), 1488–1508 (2003)
3. Bentley, J.: Multidimensional divide-and-conquer. Communications of the ACM 23(4), 214–229 (1980)
4. Bozanis, P., Kitsios, N., Makris, C., Tsakalidis, A.: New upper bounds for generalized intersection searching problems. In: Proc. ICALP. pp. 464–474 (1995)
5. De Berg, M., Haverkort, H.: Significant-presence range queries in categorical data. Algorithms and Data Structures pp. 462–473 (2003)
6. Demaine, E., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Proc. ESA. pp. 11–20. Springer (2002)
7. Dietz, P.: Optimal algorithms for list indexing and subset rank. Algorithms and Data Structures pp. 39–46 (1989)
8. Durocher, S., He, M., Munro, J.I., Nicholson, P.K., Skala, M.: Range majority in constant time and linear space. In: Proc. ICALP. vol. 6755, pp. 244–255 (2011)
9. Fredman, M., Willard, D.: Surpassing the information theoretic bound with fusion trees. Journal of Computer and System Sciences 47(3), 424–436 (1993)
10. Gagie, T., Kärkkäinen, J.: Counting colours in compressed strings. In: Proc. CPM. pp. 197–207 (2011)
11. Gupta, P., Janardan, R., Smid, M.: Further Results on Generalized Intersection Searching Problems: Counting, Reporting, and Dynamization. Journal of Algorithms 19(2), 282–317 (1995)
12. Husfeldt, T., Rauhe, T.: New lower bound techniques for dynamic partial sums and related problems. SIAM Journal on Computing 32(3), 736–753 (2003)
13. Karp, R., Shenker, S., Papadimitriou, C.: A simple algorithm for finding frequent elements in streams and bags. ACM TODS 28(1), 51–55 (2003)
14. Karpinski, M., Nekrich, Y.: Searching for frequent colors in rectangles. In: Proc. CCCG. pp. 11–14 (2008), http://cccg.ca/proceedings/2008/paper02.pdf
15. Lai, Y., Poon, C., Shi, B.: Approximate colored range and point enclosure queries. Journal of Discrete Algorithms 6(3), 420–432 (2008)
16. Misra, J., Gries, D.: Finding repeated elements. Science of Computer Programming 2(2), 143–152 (1982)
17. Wei, Z., Yi, K.: Beyond simple aggregates: indexing for summary queries. In: Proc. PODS. pp. 117–128. ACM (2011)