Range Counting with Distinct Constraints

J. Ian Munro* Yakov Nekrich† Sharma V. Thankachan‡

Abstract

In this paper we consider a special case of orthogonal point counting queries, called queries with distinct constraints. A d-dimensional orthogonal query range $Q = [b_1, b_2] \times [b_3, b_4] \times \ldots \times [b_{2d-1}, b_{2d}]$ is a range with r distinct constraints if there are r distinct values among b_1, b_2, $\ldots$, b_{2d}. We describe a data structure that supports orthogonal range counting queries with r distinct constraints. We show that the space and query time complexity of such queries depend only on the number of distinct constraints r even if r is much smaller than d. An application of queries with r distinct constraints to persistent range counting is also considered.

1 Introduction

In the orthogonal range counting problem we keep a set S of d-dimensional points in a data structure; for any orthogonal query range $Q = [b_1, b_2] \times [b_3, b_4] \times \ldots \times [b_{2d-1}, b_{2d}]$ we must be able to compute the number of points from S that are inside Q. Henceforth $[s, e]$ denotes a closed interval that contains all real values x satisfying $s \le x \le e$; $(-\infty, a]$ (or $[b, +\infty)$) denotes a half-open interval that contains all real values x satisfying $x \le a$ (resp. $x \ge b$). Two-dimensional orthogonal range counting queries can be supported in $O(\log n/ \log\log n)$ using an $O(n)$-space data structure [7]. For $d > 2$, the query time and space usage grow by a factor $O(\log n/ \log\log n)$ with every further dimension; thus d-dimensional orthogonal range counting queries can be answered in $O((\log n/ \log\log n)^{d-1})$ time by a data structure that needs $O(n(\log n/ \log\log n)^{d-2})$ space [7]. In this paper we consider a special case of orthogonal range counting queries that can be supported in less time and using less space. For a query range $Q = [b_1, b_2] \times [b_3, b_4] \times \ldots \times [b_{2d-1}, b_{2d}]$ let r denote the number of distinct values b_i in the multiset $\{ b_1 \ldots, b_{2d} \}$ such that $b_i \ne \pm\infty$. We will say that r is the number of *distinct constraints* of a query Q. A query Q such that $r < d$ will be called a distinct-constraint query. We show that distinct-constraint queries can be answered faster and using less space than the general orthogonal range counting queries.

We describe our data structure in Sections 2 and 3. Potential applications are discussed in Section 4 and Section 5. In Section 4 we describe a data structure that supports persistent range counting queries. In Section 5 we describe data structures for some special cases of the orthogonal color counting problem; our solution for the color counting problem has the same complexity as the best previously known data structure.

2 Stabbing Counting Queries

As a warm-up we describe a folklore data structure for counting one-dimensional intervals that are stabbed by a query point.

Lemma 1 *Suppose that there exists an $s(n)$-space data structure that counts the number of points in a one-dimensional range $(-\infty, a]$ in time $q(n)$. Then there exists a $2s(n)$-space data structure that counts the number of intervals that are stabbed by a query point q in time $2q(n)$.*

Proof: Let S_s be the set that contains the starting points of all intervals and let S_e be the set that contains the endpoints of all intervals. An interval $[s, e]$ is stabbed by a query point q if and only if $s \le q.x_1 \le e$.

Let $c_q = |\{ [s, e] \in S \,|\, s \le q \le e \}|$, $c^+ = |\{ s \in S_s \,|\, s \le q \}|$ and $c^- = |\{ e \in S_e \,|\, e < q \}|$. That is, c_q is the answer to a query q, c^+ is the number of intervals with starting point at most q, and c^- is the number of intervals with endpoint before q. If the endpoint of an interval is smaller than q, then its starting point is also smaller than q. If the starting point of an interval s is smaller than q, then either its endpoint is smaller than q or q stabs s. Hence $c_q = c^+ - c^-$. We can thus compute c_q by answering two range counting queries on sets of one-dimensional points. □

3 Counting with Distinct Constraints

In this section we generalize the result of Section 2 to $d > 1$ dimensions. We consider queries that ask for the number of points in a set $\{ p \in S \,|\, p.x_1 \gtrless a_1, p.x_2 \gtrless a_2, \ldots, p.x_d \gtrless a_d \}$ where $\gtrless$ denotes either "greater than or equal" or "smaller than or equal". We show that the complexity of such queries depends only on the number

*Cheriton School of CS, University of Waterloo, Waterloo, Canada. imunro@uwaterloo.ca

†Cheriton School of CS, University of Waterloo, Waterloo, Canada. ynekrich@uwaterloo.ca

‡School of CSE, Georgia Institute of Technology, Atlanta, USA. sharma.thankachan@gatech.edu

of distinct values in the sequence $a_1, \ldots, a_d$ and does not depend on d itself.

Lemma 2 *Suppose that there exists a $(d+1)$-dimensional $s(n)$-space data structure that counts the number of points in a range $(-\infty, a] \times Q_d$, where Q_d is an arbitrary d-dimensional range and a is an arbitrary real value, in time $q(n)$. Then there exists a $(d+2)$-dimensional data structure that uses space $3s(n)$ and counts the number of points in a range $(-\infty, a] \times [a, +\infty) \times Q_d$ in time $3q(n)$.*

Proof: Let $Q_m = (-\infty, a] \times [a, +\infty) \times Q_d$. We define $Q^+ = (-\infty, a] \times (-\infty, +\infty) \times Q_d$, $Q^- = (-\infty, a] \times (-\infty, a] \times Q_d$, and $Q^a = (-\infty, a] \times [a, a] \times Q_d$. Then $Q_m = (Q^+ \setminus Q^-) \cup Q^a$. We keep two $(d+1)$-dimensional sets. The set S^+ contains a point $plus(p) = (p.x_1, p.x_3, \ldots, p.x_{d+2})$ for every point $p = (p.x_1, p.x_2, p.x_3, \ldots, p.x_{d+2})$ in S. Whereas the set S^- contains a point $\max(p) = (p.x'_1, p.x_3, \ldots, p.x_{d+2})$ for every $p \in S$, where the new coordinate x'_1 is defined as $p.x'_1 = \max(p.x_1, p.x_2)$. We also keep a set S^v that contains the point $plus(p)$ for all $p \in S$, such that $p.x_2 = v$. We keep a set S^v for every value v, such that $p.x_2 = v$ for at least one $p \in S$. All auxiliary sets are kept in data structures that support $(d+1)$-dimensional range counting queries. A point $p \in S$ is in Q^+ if and only if $plus(p)$ is in $(-\infty, a] \times Q_d$. A point $p \in S$ is in Q^- if and only if $p.x'_1 = \max(p.x_1, p.x_2) \leq a$ and $(p.x_3, \ldots, p.x_{d+2}) \in Q_d$. Hence p is in Q^- if and only if $\max(p)$ is in $(-\infty, a] \times Q_d$. Finally $p \in S$ is in Q^a if and only if $p \in S^a$ and $plus(p)$ is in $(-\infty, a] \times Q_d$. Hence the numbers of points in Q^+, Q^-, and Q^a can be found by answering range counting queries on S^+, S^- and S^a respectively. Thus a query $(-\infty, a] \times [a, +\infty) \times Q_d$ is answered by answering three $(d+1)$-dimensional counting queries. □

The following Theorem is a direct corollary of Lemma 2 for the case when a $(d+2r)$-dimensional query contains at most $d+r$ distinct constraints.

Theorem 3 *Suppose that there exists a $(d+r)$-dimensional $s(n)$-space data structure that counts the number of points in a range $(-\infty, a_1] \times (-\infty, a_2] \times \ldots \times (-\infty, a_r] \times Q_d$, where Q_d is an arbitrary d-dimensional range and $a_1, \ldots, a_r$ are arbitrary real values, in time $q(n)$. Then there exists a $(d+2r)$-dimensional data structure that uses space $3^r s(n)$ and counts the number of points in a range $(-\infty, a_1] \times [a_1, +\infty) \times (-\infty, a_2] \times [a_2, +\infty) \times \ldots \times (-\infty, a_r] \times [a_r, +\infty) \times Q_d$ in time $3^r q(n)$.*

Proof: Lemma 2 is applied r times. □

We can also generalize our result for the case when the same constraint value occurs more than twice.

Lemma 4 *Suppose that there exists a $(d+1)$-dimensional $s(n)$-space data structure that counts the number of points in a range $(-\infty, a] \times Q_d$, where Q_d is an arbitrary d-dimensional range and a is an arbitrary real value, in time $q(n)$. Then there exists a $(d+d_1+d_2)$-dimensional data structure that uses space $3s(n)$ and counts the number of points in any range $Q = (-\infty, a_1] \times \ldots \times (-\infty, a_{d_1}] \times [a_{d_1+1}, +\infty) \times \ldots \times [a_{d_1+d_2}, +\infty) \times Q_d$, where $a_1 = a_2 = \ldots = a_{d_1+d_2} = a$, in time $3q(n)$.*

Proof: Let S be a set of $(d+d_1+d_2)$-dimensional points. We replace the first d_1 coordinates of each point by their maximum and the following d_2 coordinates by their minimum. The resulting set S_{new} contains a $(d+2)$-dimensional point $p_{new} = (\mu_1, \mu_2, p.x_{d_1+d_2+1}, \ldots, p.x_{d_1+d_2+d})$ for every point $p \in S$ where $\mu_1 = \max(p.x_1, \ldots, p.x_{d_1})$ and $\mu_2 = \min(p.x_{d_1+1}, \ldots, p.x_{d_1+d_2})$. Clearly, p_{new} is in $Q_{new} = (-\infty, a] \times [a, +\infty) \times Q_d$ if and only if the corresponding point p is in Q. By Lemma 2 we can count the number of points in $S_{new} \cap Q_{new}$ in time $3q(n)$ using $3s(n)$ space. □

Theorem 5 *The problem of answering d-dimensional range counting queries with r distinct constraints has the same asymptotic space and query time complexity as the general r-dimensional range counting, when r is constant.*

Proof: For each point $p = (p.x_1, \ldots, p.x_d)$ in S we create a $2d$-dimensional point $\overline{p} = (x_1, x_1, x_2, x_2, \ldots, x_d, x_d)$. That is, $\overline{p}$ contains two copies of each p's coordinate. Let $\overline{S}$ be the set of such points $\overline{p}$. A query $Q = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ on S is equivalent to a $2d$-dimensional query $[a_1, +\infty) \times (-\infty, b_1] \times [a_2, +\infty) \times (-\infty, b_2] \times \ldots \times [a_d, +\infty) \times (-\infty, b_d]$ on $\overline{S}$. We re-order the coordinates of points in $\overline{S}$ so that half-open intervals with the same constraint value are grouped together and intervals $(-\infty, a]$ precede intervals $[a, +\infty)$ for the same value a. The transformed query is of the form $Q' = (-\infty, a_1] \times (-\infty, a_2] \times \ldots \times (-\infty, a_{2d}]$ and only $Q' = Q_1 \times Q_2 \times \ldots \times Q_r$ where each Q_i is a query range with one distinct constraint: for $1 \leq i \leq r$, $Q_i = (-\infty, a_1] \times \ldots \times (-\infty, a_{f_i}] \times [a_{f_i+1}, +\infty) \times \ldots \times [a_{g_i}, +\infty)$ where $a_j = v_i$ for some v and for all j such that $f_i \leq j \leq g_i$. Lemma 4 is applied r times to query range Q'. In this way the query is reduced to 3^r r-dimensional queries. The total space usage of our data structure is $3^r s(n)$, where $s(n)$ is the space needed by a data structure for r-dimensional counting queries.

□

4 Persistent Counting

Now we turn to applications of our approach. Consider a dynamic set of points S. A data structure on S is called partially persistent if every update (insertion or deletion of a point) creates a new version and queries on any version of the data structure are supported. A partially persistent range counting query (Q, t_q) asks for the number of points $p \in Q \cap S$ that were stored in D at time t. A data structure is called offline partially persistent if the sequence of updates is known in advance (that is, all updates of S are known when the data structure is constructed). We refer to the seminal paper of Driscoll *et al.* [5] and to a survey of Kaplan [9] for an extensive description of persistence.

In this section we describe a general method of designing persistent data structures for counting problems. Let Q_d denote an arbitrary d-dimensional range. Our approach enables us to transform any data structure that answers $(d+1)$-dimensional queries of the form $Q_d \times (-\infty, a]$ into a partially persistent data structure that counts the number of points in Q_d and supports both insertions and deletions. The same method can be also applied to other geometric objects (segments, rectangles etc.) We show that d-dimensional offline partially persistent range counting is equivalent to $(d+1)$-dimensional static orthogonal range counting. For instance, one-dimensional partially persistent counting queries can be answered in $O(\log n/\log\log n)$ time using an $O(n)$ space data structure. We remark that a straightforward application of techniques for obtaining partially persistent data structures from dynamic data structures [5] does not lead to a linear space data structure for one-dimensional persistent range counting: the data structure of Driscoll et al [5] can be used to turn a balanced tree into a persistent data structure. In order to support one-dimensional counting queries, we have to keep information about the number of leaves stored below every tree node. Every insertion or deletion changes this information for $O(\log n)$ nodes. Hence a straightforward algorithm for making a data structure persistent would result in an $O(n \log n)$-space data structure.

Lemma 6 *Suppose that there exists an $s(n)$-space data structure that counts the number of points in a range $Q_d \times (-\infty, a]$, where Q_d is an arbitrary d-dimensional range, in time $q(n)$. Then there exists an offline partially persistent data structure that uses space $3s(n)$ and counts the number of points in a range Q_d in time $3q(n)$.*

Proof: We associate a *lifetime* interval $[t_s(p), t_e(p)]$ with each point p, where $t_s(p)$ and $t_e(p)$ denote the times when p was inserted into S and deleted from S. We associate a point $temp(p) = (t_s(p), t_e(p), p.x_1, \ldots, p.x_d)$ to each $p \in S$. Let $S_{temp} = \{\, temp(p) \mid p \in S\,\}$. Given a query (Q_d, t), we must count points p such that $p \in Q_d$ and $t_s(p) \le t \le t_e(p)$. Counting all points that are in Q_d and are stored in a data structure at time t is equivalent to answering $(d+2)$-dimensional query $(-\infty, t] \times [t, +\infty) \times Q_d$ on S_{temp}. Such queries have at most $d+1$ constraint. By Lemma 2, such queries can be answered in time $3q(n)$ using $3s(n)$ space. □

5 Color Counting

Colored or categorical orthogonal range searching is an important variant of the range searching problem. The set of points S of size n, such that each point is assigned a color, is pre-processed and stored in a data structure. For any rectangular query range Q, we must be able to find some information about colors of points in $S \cap Q$. In the case of color counting queries, we want to compute the number of distinct point colors in $S \cap Q$. In the case of color reporting queries, we want to enumerate all distinct point colors in $S \cap Q$. In this section we describe a data structure for color counting. Our solution, based on counting with distinct constraints, matches the best previously known bounds. Thus we show that distinct-constraint counting provides an alternative solution for this problem.

Color searching problems arise naturally in many database applications when the input data objects are distributed into categories. We may want to enumerate (or count the number of) categories of objects whose attribute values are in a certain range. For instance, suppose that a geographic database contains data about locations. Given a query area, we may be interested in listing (or counting) types of soil in that area. Other applications of this problem include document retrieval [12, 13], computational geometry [10], and VLSI layout [6].

Color range searching problem were studied extensively during the last two decades, see e.g., [8, 6, 3, 4, 2, 12, 10, 14, 15, 11]. In spite of significant efforts, space-efficient data structures (i.e., using $n \log^{O(1)} n$ space) are known only for color reporting in $d \le 3$ dimensions. Space-efficient color counting in $d \ge 2$ dimensions is possible only in some special cases. Thus counting or reporting distinct point colors appears to be significantly harder than counting or reporting all points in an orthogonal range.

Gupta et al. [6] describe a data structure for one-dimensional color counting queries that uses $O(n \log n)$ space and supports queries in $O(\log n)$ time. This result is obtained by reducing one-dimensional color counting to two-dimensional point counting. Using the reduction from [6] and a linear size data structure for point-counting, described by JaJa et al. [7], we can obtain an $O(n)$-space data structure that answers one-dimensional color counting queries in $O(\log n/\log\log n)$ time. Space-efficient data structures for some special cases of two-dimensional queries are also known. A two-dimensional dominance query is a product of two half-open intervals, e.g., $(-\infty, b] \times (-\infty, h]$. A three-sided query range is a product of a closed interval and a half-open interval, e.g., $[a, b] \times (-\infty, h]$. In [6] the authors describe an $O(n \log n)$-space data structure that supports dominance color counting in $O(\log n)$ time and an $O(n \log^2 n)$-space structure that supports three-sided color counting in $O(\log^2 n)$ time; they also describe an $O(n^2 \log^2 n)$ data structure that answers general queries in 2-D in $O(\log^2 n)$ time. Kaplan et al [10] describe a general method that reduces the problem of counting colors in d-dimensional dominance range to counting d-dimensional rectangles that are stabbed by a point q. The set of rectangles used in [10] consists of $O(n)$ rectangles for $d = 2$ or $d = 3$. Kaplan et al [10] describe an $O(n \log n)$-space data structure that answers two-dimensional dominance color counting queries in $O(\log n)$ time and an $O(n \log^2 n)$ space data structure that answers three-dimensional dominance color counting queries in $O(\log^2 n)$ time.

However if we combine the best currently known data structures for rectangle stabbing counting with the reduction from [10], then both query time and space usage can be reduced. There is a data structure that answers two-dimensional dominance color counting queries in $O(\log n / \log \log n)$ time and uses space $O(n)$. There is also a data structure that answers three-dimensional dominance color counting queries in $O((\log n / \log \log n)^2)$ time and uses space $O(n(\log n / \log \log n)$.

Below we provide an alternative solution for color dominance in two and three dimensions. Although our data structures have the same complexity as previous best solutions, we believe that our alternative solution is also of interest.

Dominance Color Counting in 2-D. In the two-dimensional dominance query (aka 2-sided two-dimensional query), the query range Q is a product of two half-open intervals. We will consider queries $(-\infty, a] \times (-\infty, b]$. A two-dimensional point q dominates a point p if both coordinates of q are not smaller than p, $q.x_1 \geq p.x_1$ and $q.x_2 \geq p.x_2$. The skyline M of a set S consists of all points in S that do not dominate any other point in S. If we arrange the points on a skyline M in increasing order of their first coordinates, then the second coordinates of points in M will form a decreasing sequence. For a point $p \in M$, let $next(p) = p'.x_1$ where p' is the right neighbor of p on M.

Let the set S_c contain all points of color c in S. Let M_c denote the skyline of S_c. The set S_1 contains a three-dimensional point $p_1 = (p.x_1, next(p), p.x_2)$ for each $p \in M_c$ and for all colors c. It was shown in [6] that $Q = (-\infty, a] \times (-\infty, b]$ contains a point of color

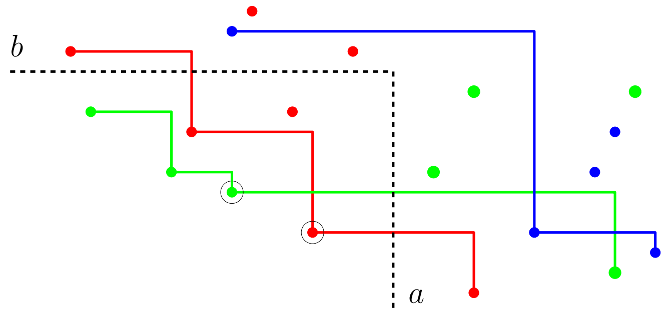

Figure 1: Answering a two-dimensional dominance color counting query on a set of red, blue, and green points. Skyline points are connected by straight lines. For a query $Q = (-\infty, a] \times (-\infty, b]$, we count the number of circled points. Exactly one point for each color that occurs in $(-\infty, a] \times (-\infty, b]$ is considered.

c if and only if there is exactly one point $p \in M_c$ such that $p.x_1 \leq a$, $next(p) \geq a$, and $p.x_2 \leq b$. See Fig. 1 for an example.

Thus we can count the number of colors in Q by answering a query $(-\infty, a] \times [a, +\infty) \times (-\infty, b]$ with 2 distinct constraints on S_1.

Theorem 7 *Two-dimensional dominance color counting has the same space and query time complexity as two-dimensional point counting.*

Optimal data structures in the RAM and external memory models follow immediately. We plug the data structures from [7] and [1] into Theorem 7.

Corollary 1 *There exists an $O(n)$-space data structure that answers two-dimensional dominance color counting queries in optimal $O(\log n / \log \log n)$ time.*

Corollary 2 *There exists an external-memory data structure that uses $O(n)$ words of space and answers two-dimensional dominance color counting queries in $O(\log_B n)$ I/Os.*

Insertion-Only Dominance Color Counting in 2-D. Let D_1 denote the data structure that supports two-dimensional dominance queries. The data structure D_1 can also support insertions. Suppose that a new point p_{new} of color c is inserted. If p_1 dominates some $p \in M_c$, then we do not have to change M_c and no updates of D_1 are necessary. Otherwise, we insert p_{new} into M_c. In this case we also may have to remove a number of other points from M_c. Data structure D_1 is updated accordingly. An insertion of a single point into M_c can lead to a large number of updates. But Gupta et al. [6] have shown that n insertions into an initially empty data structure require $O(n)$ updates of skylines M_c. Hence D_1 is also updated $O(n)$ times. The key observation is that each point in inserted and removed from some M_c at most once: if p is removed from M_c, it will not be re-inserted into M_c in the future. We refer to [6] for details.

Dominance Counting in 3-D. Following the approach of [6], we can transform a 2-D dominance query into a 3-D dominance using a persistent version of the two-dimensional data structure described above. While in [6] this technique was applied to range reporting, we use it to obtain a range counting data structure. We sort points of a three-dimensional set S in increasing order by their z-coordinates. These points are then inserted in the same order into a partially persistent variant of the data structure D_1 which we will denote by D_2. We use the approach outlined in Section 4 for adding persistence. Each point in D_2 is associated with two additional coordinates. For every point $p \in D_1$ that was inserted at time t_s and removed at time t_e, D_2 contains a point $p = (p.x, next(p), p.y, t_s, t_e)$. In order to answer a query $(-\infty, a] \times (-\infty, b] \times (-\infty, h]$,we find the version t_h that corresponds to the largest z-coordinate that does not exceed h. Then we count the number of colors in a two-dimensional range $(-\infty, a] \times (-\infty, b]$ at time t_h. That is, we answer a counting query $(-\infty, a] \times [a, +\infty) \times (-\infty, b]$ at time t_h. As shown in Section 4, this is equivalent to answering a query $(-\infty, a] \times [a, +\infty) \times (-\infty, b] \times (-\infty, t_h] \times [t_h, +\infty)$. Although this is a five-dimensional query, it has three distinct constraints. Hence, it has the same complexity as three-dimensional point counting.

Theorem 8 *Three-dimensional dominance color counting has the same space and query time complexity as three-dimensional point counting.*

Again we plug the data structures from [7] and [1] into Theorem 8.

Corollary 3 *There exists an* $O(n(\log n/\log\log n))$*-space data structure that answers three-dimensional dominance color counting queries in* $O((\log n/\log\log n)^2)$ *time.*

Corollary 4 *There exists an external-memory data structure that uses* $O(n \log_B n)$ *words of space and answers three-dimensional dominance color counting queries in* $O((\log_B n)^2)$ *I/Os.*

References

[1] Pankaj K. Agarwal, Lars Arge, Sathish Govindarajan, Jun Yang, and Ke Yi. Efficient external memory structures for range-aggregate queries. *Comput. Geom.*, 46(3):358–370, 2013.

[2] Pankaj K. Agarwal, Sathish Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In *Proc. 10th Annual European Symposium on Algorithms (ESA 2002)*, pages 17–28, 2002.

[3] Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP 95)*, pages 464–474, 1995.

[4] Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New results on intersection query problems. *Computer Journal*, 40(1):22–29, 1997.

[5] James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[6] Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.

[7] Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. 15th International Symposium on Algorithms and Computation (ISAAC 2004)*, pages 558–568, 2004.

[8] Ravi Janardan and Mario A. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry and Applications*, 3(1):39–69, 1993.

[9] Haim Kaplan. Persistent data structures. In *Handbook on Data Structures and Applications, D. Mehta and S. Sahni (Editors)*, pages 241–246. CRC Press 2001, 2005.

[10] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38(3):982–1011, 2008.

[11] Marek Karpinski and Yakov Nekrich. Searching for frequent colors in rectangles. In *Proc. 20th Annual Canadian Conference on Computational Geometry (CCCG)*, 2008.

[12] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002.

[13] Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4):52, 2013.

[14] Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9, 2014.

[15] Matthew Skala. Array range queries. In *Space-Efficient Data Structures, Streams, and Algorithms*, pages 333–350, 2013.