Fast Computation of Categorical Richness on Raster Data Sets and Related Problems

M. de Berg*
TU Eindhoven, the Netherlands
mdberg@win.tue.nl

C. Tsirogiannis†
MADALGO
Aarhus University, Denmark
constant@madalgo.au.dk

B.T. Wilkinson†‡
MADALGO
Aarhus University, Denmark
btw@cs.au.dk

Abstract

In many scientific fields, it is common to encounter raster data sets consisting of categorical data, such as soil type or land usage of a terrain. A problem that arises in the presence of such data is the following: given a raster $\mathcal{G}$ of n cells storing categorical data, compute for every cell c in $\mathcal{G}$ the number of different categories appearing within a window centered at c. The window can either be a $(2r+1) \times (2r+1)$ square or a disk of radius r for a positive integer parameter r. We call this the *categorical richness problem*, and we present two algorithms for it: one for square windows that runs in $O(n)$ time and one for circular windows that runs in $O((1 + K/r)n)$ time, where K is the number of different categories appearing in $\mathcal{G}$. The algorithms are not only very efficient in theory, but also in practice: our experiments show that our algorithms can handle raster data sets of hundreds of millions of cells.

The categorical richness problem is related to *colored range counting*, where the goal is to preprocess a colored point set such that we can efficiently count the number of colors appearing inside a query range. We present a data structure for colored range counting in $\mathbb{R}^2$ for the case where query ranges are squares. Our structure uses $O(n \operatorname{polylog} n)$ storage and has $O(\operatorname{polylog} n)$ query time, which is significantly better than what is known for arbitrary rectangular ranges.

*MdB is supported by the Netherlands Organization for Scientific Research under grant 024.002.003.

†Work supported by the Danish National Research Foundation grant DNRF84 through the Center for Massive Data Algorithmics (MADALGO).

‡Work supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

1 Introduction

Background and motivation. Terrain data and other geographic information is often stored in the form of a raster data set. The region of interest is partitioned into a *raster* (a grid of square cells) and for each cell in the raster the data pertaining to the corresponding location is stored. Sometimes the data is numerical; in a digital elevation model (DEM), for instance, each cell stores an elevation value. In other applications the data is *categorical*. For a terrain, for example, one may store information about land usage or soil type. Note that numerical data is sometimes interpreted as categorical by considering different ranges of values as different categories (i.e., low elevation, medium elevation, high elevation). Rasters storing categorical data are not only used frequently in GIS, but also in various other fields.

In this paper we are concerned with the following computational problem. Let $\mathcal{G}$ be a raster of n cells storing categorical data and let r be an integer. Without loss of generality, we assume the cells in $\mathcal{G}$ have unit size. We now wish to compute for each cell c in $\mathcal{G}$ its *categorical richness*, that is, the number of different category values appearing in its neighborhood. We call this the *categorical richness problem*. It comes in two flavors, depending on the shape of the window defining the neighborhood of a cell c: in some applications the window is a square of size $(2r+1) \times (2r+1)$ centered at c, in other applications it is a disk of radius r. We refer to the first variant as *square richness* and to the second variant as *disk richness*.

The categorical richness problem appears in many scientific case studies, under different names. In GIS applications, this problem is usually referred to as computing the *patch richness* for every cell in a raster [10, 16]. Standard GIS software such as GRASS [14] and FRAGSTATS [13] provide this functionality, but their implementations are inefficient and cannot handle large data sets.

In ecology, de Albuquerque *et al.* [6] encounter the square richness problem when they want to compute the variability of elevation data on a raster terrain. First, they convert the elevation values of the raster cells into elevation categories, and then they compute for each cell c the number of elevation categories that appear within a square window centered at c. They call this the *topographic heterogeneity* of the cell. The concept also appears in other ecological studies [2, 3].

Related work. A straightforward way to solve the categorical richness problem is to explicitly scan for each cell c in the raster $\mathcal{G}$ its (square of circular) window and record the different category values encountered. Since a window contains $\Theta(r^2)$ cells, this algorithm runs in $O(nr^2)$ time. This is infeasible for large data sets, even for moderate values of r. A more refined algorithm would use that windows of neighboring cells differ in only $\Theta(r)$ cells; this can be exploited to obtain an algorithm with $O(nr)$ time. However, also this approach can become quite slow for large data sets.

An alternative approach, which avoids the dependency on r, is to use data structures for *colored range counting queries* [9] from computational geometry. These data structures store a set of n colored points—the colors represent the different categories—such that the number of colors within a query region can be counted efficiently. After preprocessing the set of cell centers for such queries

one can solve the categorical richness problem by performing a query for each cell c with the window centered at c. Unfortunately, the known data structures for colored range counting are not very efficient: the best known structure with $O(\text{polylog}\, n)$ query time uses $O(n^2 \log^2 n)$ storage [8], which is clearly infeasible in our application. More efficient solutions exist for reporting (rather than counting) all colors in the range: one can obtain $O(\log n + t)$ query time (where t is the number of reported colors) using $O(n \log n)$ storage [15], but this is still too slow for large data sets. For circular ranges, the results are even worse. Note that our application is *offline*, that is, all queries are known in advance. Kaplan *et al.* [11] give an algorithm for the offline problem with n rectangular queries, which runs in $O(n^{1.408})$ time—again too slow for large data sets. They also showed that an $o(n^{1.186})$ solution would imply an improvement of the best running time for the well known Boolean matrix multiplication problem. For the offline version with circular queries, no results are known.

We conclude that general results on (online or offline) colored range searching do not solve the categorical richness problem fast enough. The question is thus: can we exploit the special structure of our problem, where the points form a grid and the ranges are centered at these grid points, to obtain a fast solution for the categorical richness problem?

Our results. We answer the above question affirmatively in Section 2, both for square richness and for disk richness. Our algorithm for square richness runs in $O(n)$ time, independent of the parameter r specifying the window size. Our algorithm for disk richness runs in $O((1 + K/r)n)$ time, where K is the total number of different categories in the data set; in practice, we typically have $K \leqslant r$, in which case the algorithm runs in $O(n)$ time. Our algorithms are not only efficient in theory, but also in practice: in Section 4 we present an experimental evaluation showing they can handle raster data sets of hundreds of millions of cells.

Our second set of results, presented in Section 3, concerns colored range counting queries for general point sets. We show how to preprocess a set P of n points in the plane into a data structure of size $O(n\, \text{polylog}\, n)$ such that a color counting query with a query square (of arbitrary size) can be answered in $O(\text{polylog}\, n)$ time—much better than the best known solution for rectangular queries. We also present a similarly efficient data structure for the 3-dimensional version of the problem where the query ranges are fixed-size cubes. Finally, we investigate the hardness of the offline problem for points in $\mathbb{R}^3$ and ranges that are variable-sized cubes by relating its computational complexity to that of Boolean matrix multiplication.

2 Algorithms for Richness

2.1 Notation and terminology

The raster. We assume for simplicity that the raster for which we want to compute the categorical richness consists of $\sqrt{n} \times \sqrt{n}$ cells, although our approach also works for non-square rasters. Thus we denote the raster by $\mathcal{G}[1..\sqrt{n}, 1..\sqrt{n}]$. We assume without loss of generality that the center of the cell $\mathcal{G}[i, j]$ is the point $(i, j) \in \mathbb{R}^2$; hence, the cells of the raster have unit size. When talking about an

arbitrary cell $c \in \mathcal{G}$, we use p_c to denote the center of p. We define the distance between two cells c and c' to be the (Euclidean) distance between their centers p_c and $p_{c'}$.

Each cell in $\mathcal{G}$ stores a category value. We identify the category values by positive integers, and let $\mathcal{K} := \{1, 2, \dots, K\}$ denote the set of all category values that appear in $\mathcal{G}$. We use $\mathrm{cat}(c)$ to denote the category value of a cell c.

Square richness. Let r be a positive integer and let c be a cell in $\mathcal{G}$. We define $W_{\mathrm{sq}}(c)$, the *square window* of c, to be the square region of size $(2r+1) \times (2r+1)$ centered at p_c. With a slight abuse of notation, we also use $W_{\mathrm{sq}}(c)$ to denote the set of cells whose centers are inside $W_{\mathrm{sq}}(c)$. Thus we sometimes write $c' \in W_{\mathrm{sq}}(c)$ instead of $p_{c'} \in W_{\mathrm{sq}}(c)$. We define $\mathcal{K}_{\mathrm{sq}}(c) \subseteq \mathcal{K}$ to be the set of all category values that occur in the window $W_{\mathrm{sq}}(c)$, that is, $\mathcal{K}_{\mathrm{sq}}(c)$ is the set of all $k \in \mathcal{K}$ such that there is a cell $c' \in W_{\mathrm{sq}}(c)$ with $\mathrm{cat}(c') = k$. Finally, we define $\mathrm{rich}_{\mathrm{sq}}(c)$, the *square richness* of a cell c, as the number of category values in $W_{\mathrm{sq}}(c)$. In other words, $\mathrm{rich}_{\mathrm{sq}}(c) := |\mathcal{K}_{\mathrm{sq}}(c)|$. The square richness problem is now to compute $\mathrm{rich}_{\mathrm{sq}}(c)$ for all cells $c \in \mathcal{G}$.

Disk richness. The disk richness problem is defined in a similar way. We now use a window $W_{\mathrm{d}}(c)$, which is a disk of radius r centered at p_c, define $\mathcal{K}_{\mathrm{d}}(c)$ to be the set of category values occurring in $W_{\mathrm{d}}(c)$, and define $\mathrm{rich}_{\mathrm{d}}(c) := |\mathcal{K}_{\mathrm{d}}(c)|$ to be the disk richness of c.

2.2 An optimal algorithm for square richness

Next we describe an algorithm that solves the square richness problem in $O(n)$ time, which is optimal. The algorithm is based on the following simple observation.

Observation 1. *Let c and c' be two cells in $\mathcal{G}$. Then $c \in W_{\mathrm{sq}}(c')$ if and only if $c' \in W_{\mathrm{sq}}(c)$. Hence, $k \in \mathcal{K}_{\mathrm{sq}}(c)$ if and only if $c \in W_{\mathrm{sq}}(c')$ for at least one cell c' with $\mathrm{cat}(c') = k$.*

This observation implies that $k \in \mathcal{K}_{\mathrm{sq}}(c)$ if and only if c falls inside the union of all windows $W_{\mathrm{sq}}(c')$ such that $\mathrm{cat}(c') = k$. We call the region covered by this union the *influence region* of k, and we denote it by $A(k)$.

The global algorithm. Observation 1 suggests the following algorithm. First, for each category value $k \in \mathcal{K}$, extract all cells c with $\mathrm{cat}(c) = k$ and compute the influence region $A(k)$. Next, scan all cells in $\mathcal{G}$ and calculate for each cell c the number of influence regions containing c. To do this efficiently, we need to refine the algorithm in the following way: instead of processing $\mathcal{G}$ as a whole, we partition it into horizontal *strips*, each consisting of $2r + 1$ consecutive rows from $\mathcal{G}$ (except maybe the top strip, which can have fewer rows). Thus we have a strip $\mathcal{G}[1..\sqrt{n}, 1..2r + 1]$, a strip $\mathcal{G}[1..\sqrt{n}, 2r + 2..4r + 2]$, and so on.

Each strip $\mathcal{S}$ is handled as follows. Define $\mathcal{K}_{\mathcal{S}} \subseteq \mathcal{K}$ to be the set of categories k such that there is a window $W_{\mathrm{sq}}(c)$ intersecting $\mathcal{S}$ and with $\mathrm{cat}(c) = k$. Define $A_{\mathcal{S}}(k)$ to be the part of $A(k)$ that falls within $\mathcal{S}$. We start by computing $A_{\mathcal{S}}(k)$ for all $k \in \mathcal{K}_{\mathcal{S}}$. To this end, we first determine $\mathcal{C}(\mathcal{S}, k)$, the collection of cells $c \in \mathcal{G}$ such that $\mathrm{cat}(c) = k$ and $W_{\mathrm{sq}}(c)$ intersects $\mathcal{S}$. Note that any cell $c \in \mathcal{C}(\mathcal{S}, k)$ is either a cell of the strip $\mathcal{S}$ itself, or a cell of the strips immediately above or below $\mathcal{S}$. Hence, by scanning these (at most) three strips we can

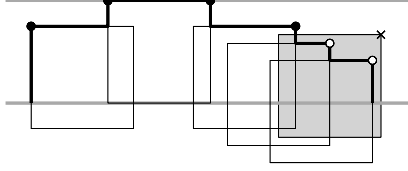

Figure 1: Adding a square to the envelope. The fat polyline indicates the current envelope. When the grey window W is added, the non-solid corners are replaced by the corner of W indicated by a cross.

generate the sets $\mathcal{C}(\mathcal{S}, k)$ in $O(r\sqrt{n})$ time. Note that by scanning the strips column by column, we can make sure the cells in $\mathcal{C}(\mathcal{S}, k)$ are ordered from left to right.

Let $\mathcal{W}(\mathcal{S}, k)$ denote the set of windows corresponding to the cells in $\mathcal{C}(\mathcal{S}, k)$. To compute $A_{\mathcal{S}}(k)$ we run a subroutine *UnionInStrip* on $\mathcal{W}(\mathcal{S}, k)$. After we compute all regions $A_{\mathcal{S}}(k)$, we feed them to a subroutine *RichnessInStrip*. This subroutine computes for every cell c in $\mathcal{S}$ the number of regions $A_{\mathcal{S}}(k)$ containing c, which is equal to the square richness for this cell. The complete algorithm, which we call *SquareRichness*, is summarized as follows.

Algorithm *SquareRichness*($\mathcal{G}$)
1. Partition $\mathcal{G}$ into strips of $2r + 1$ consecutive rows each.
2. **for** every strip $\mathcal{S}$ in $\mathcal{G}$
3. **do** Determine the set $\mathcal{K}_{\mathcal{S}}$.
4. Generate the sets $\mathcal{C}(\mathcal{S}, k)$ for all $k \in \mathcal{K}_{\mathcal{S}}$.
5. **for** all $k \in \mathcal{K}_{\mathcal{S}}$
6. **do** $A_{\mathcal{S}}(k) \leftarrow$ *UnionInStrip*($\mathcal{S}, \mathcal{W}(\mathcal{S}, k)$)
7. Run *RichnessInStrip* on $A_{\mathcal{S}}(k)$ for $k \in \mathcal{K}_{\mathcal{S}}$.

It remains to describe the subroutines *UnionInStrip* and *RichnessInStrip*. For the rest of this section, we use $\mathcal{S}[i, j]$ to denote the cell that appears in the i-th column and j-th row of strip $\mathcal{S}$. Without loss of generality, we assume that the center of this cell has coordinates (i, j).

Subroutine *UnionInStrip*. Consider a category $k \in \mathcal{K}$ and a strip $\mathcal{S}$. Let Union($\mathcal{W}(\mathcal{S}, k)$) denote the union of all windows in $\mathcal{W}(\mathcal{S}, k)$. Our goal is to compute $A_{\mathcal{S}}(k)$, which is equal to Union($\mathcal{W}(\mathcal{S}, k)) \cap \mathcal{S}$. To this end we first partition $\mathcal{C}(\mathcal{S}, k)$ into two sets, $\mathcal{C}(\mathcal{S}, k)^b$ and $\mathcal{C}(\mathcal{S}, k)^t$, which contain the windows intersecting the bottom and top boundary of $\mathcal{S}$, respectively. Note that every window in $\mathcal{W}(\mathcal{S}, m)$ must intersect at least one of the two boundaries. A window that intersects both boundaries is arbitrarily assigned to one of the two sets. We describe how to compute Union($\mathcal{W}(\mathcal{S}, m)^b) \cap \mathcal{S}$; we can compute Union($\mathcal{W}(\mathcal{S}, m)^t) \cap \mathcal{S}$ in a similar way.

The portion of the boundary of Union($\mathcal{W}(\mathcal{S}, m)^b$) above the bottom edge of $\mathcal{S}$ is the upper envelope of $\mathcal{W}(\mathcal{S}, m)^b$. We incrementally build this upper envelope, adding the windows of $\mathcal{W}(\mathcal{S}, m)^b$ in order from left to right. (If we have several windows whose centers have the same x-coordinate, we only need to process the highest one.) Our representation of the upper envelope is simply the sorted list of window corners that appear on the upper envelope, from which it is straightforward to extract a full description of the envelope in linear time. Suppose we are currently inserting window W. As long as the rightmost corner

of the current upper envelope is contained in W, we delete the corner; see Fig. 1. This process will terminate due to one of two possible reasons.

One possibility is that we encounter a corner v which is above W. Then, v must be the top-right corner of a window W'. Since W and W' have the same size, the top side of W' is above the top side of W to the left of v. Hence, no further changes are required to the upper envelope to the left of v. We finish this case by adding the top-right vertex of W to the upper envelope.

Another possibility is that we reach a corner v that is to the left of the left side of W (or there are no remaining corners). Then, we simply add the top-left and top-right corners of W to the upper envelope.

After computing the lower envelope of $\mathcal{W}(\mathcal{S}, m)^t$ in a similar way, we have the boundaries of $\text{Union}(\mathcal{W}(\mathcal{S}, m)^b) \cap \mathcal{S}$ and $\text{Union}(\mathcal{W}(\mathcal{S}, m)^t) \cap \mathcal{S}$ available. Computing $A_{\mathcal{S}}(k)$ can now easily be done by a parallel left-to-right scan of the just computed envelopes. This leads to the following lemma.

Lemma 2. *Given a set $\mathcal{C}(\mathcal{S}, k)$ whose cells are sorted from left to right, we can compute $A_{\mathcal{S}}(k)$ in $O(|\mathcal{C}(\mathcal{S}, k)|)$ time. Hence, we can compute all regions $A_{\mathcal{S}}(k)$ for $k \in \mathcal{K}_{\mathcal{S}}$ in $O(r\sqrt{n})$ time in total.*

Proof. Consider the algorithm described above for constructing the upper envelope of $\mathcal{W}(\mathcal{S}, k)^b$. Inserting the windows of $\mathcal{W}(\mathcal{S}, k)^b$ takes $O(|\mathcal{W}(\mathcal{S}, k)^b|)$ time in total, because each window corner can be inserted into and deleted from the upper envelope at most once. Similarly, we compute the lower envelope of $\mathcal{W}(\mathcal{S}, k)^t$ in $O(|\mathcal{W}(\mathcal{S}, k)^t|)$ time, which sums to $O(|\mathcal{W}(\mathcal{S}, k)|) = O(|\mathcal{C}(\mathcal{S}, k)|)$ time for computing both envelopes. The time to compute $A_{\mathcal{S}}(k)$ by a parallel scan of the two envelopes is linear in the size of those envelopes and the number of crossings between the two envelopes, which is easily seen to be linear in the size of the envelopes. (Indeed, a vertical segment of one envelope intersects at most one horizontal segment of the other envelope.)

To prove the second part of the lemma, we note that (as already described) we can generate the sorted sets $\mathcal{C}(\mathcal{S}, k)$ in $O(r\sqrt{n})$ time. This also implies that $\sum_{k \in \mathcal{K}_{\mathcal{S}}} |\mathcal{C}(\mathcal{S}, k)| = O(r\sqrt{n})$, which finishes the proof. □

Subroutine *RichnessInStrip*. Given a strip $\mathcal{S}$ and the set of influence regions $A_{\mathcal{S}}(k)$ for $k \in \mathcal{K}_{\mathcal{S}}$, we want to compute the square richness $|\mathcal{K}_{\text{sq}}(c)|$ of each cell in $\mathcal{S}$. Recall that $k \in \mathcal{K}_{\text{sq}}(c)$ if and only if $p_c \in A_{\mathcal{S}}(k)$, where p_c is the center of c. We can decide if $p_c \in A_{\mathcal{S}}(k)$ by splitting the set of vertices of $A_{\mathcal{S}}(k)$ into two subsets and counting the number of vertices of each subset that are dominated by p_c, as explained next.

Let $V(k)$ be the set of vertices of $A_{\mathcal{S}}(k)$. We split $V(k)$ into subsets $V_{\text{tl+br}}(k)$ and $V_{\text{bl+tr}}(k)$, as follows. Each vertex $v \in V(k)$ is an endpoint of exactly one vertical edge, which can either be left-bounding (meaning $A_{\mathcal{S}}(k)$ lies locally to the right of the edge) or right-bounding (meaning $A_{\mathcal{S}}(k)$ lies locally to the left). If v is the top vertex of a left-bounding edge or the bottom vertex of a right-bounding edge, we add v to $V_{\text{tl+br}}(k)$; otherwise we add v to $V_{\text{bl+tr}}(k)$.

A point (x_1, y_1) *dominates* another point (x_2, y_2) if and only if $x_1 \geqslant x_2$ and $y_1 \geqslant y_2$. Given a point p and a (multi-)set V of points, let $\text{Dom}(V, p)$ be the subset of V dominated by p. Our algorithm is based on the following lemma.

Lemma 3. *Let c be a cell in $\mathcal{S}$ with center p_c. Define $\mu_k(p_c) := |\operatorname{Dom}(V_{\mathrm{bl+tr}}(k), p_c)| - |\operatorname{Dom}(V_{\mathrm{tl+br}}(k), p_c)|$. If $p_c \in A_{\mathcal{S}}(k)$ then $\mu_k(c) = 1$, otherwise $\mu_k(c) = 0$.*

Proof. Let ρ be the horizontal ray starting at p_c and extending to the left. If $p_c \in A_{\mathcal{S}}(k)$, then ρ intersects one more left-bounding edge than it intersects right-bounding edges; if $p_c \notin A_{\mathcal{S}}(k)$, then ρ intersects as many left-bounding as right-bounding edges. A left-bounding edge intersecting ρ contributes one vertex to $V_{\mathrm{bl+tr}}(k)$ and no vertices to $V_{\mathrm{tl+br}}(k)$, while a right-bounding edge intersecting ρ contributes one vertex to $V_{\mathrm{tl+br}}(k)$ and no vertices to $V_{\mathrm{bl+tr}}(k)$. Hence, the total contribution to $\mu_k(c)$ of the edges intersecting ρ is $+1$ if $p_c \in A_{\mathcal{S}}(k)$ and zero otherwise.

The vertical edges not intersecting ρ have both endpoints dominated by p_c or neither endpoint. Hence, their contribution to $\mu_k(c)$ is zero. (For an edge with neither endpoint being dominated this is trivial; for an edge with both endpoint dominated this follows because the contributions of the two endpoints of a vertical edge cancel each other.) Thus $\mu_k(c) = 1$ if $p_c \in A_{\mathcal{S}}(k)$ and $\mu_k(c) = 0$ otherwise. □

According to Lemma 3 we can decide if $p_c \in A_{\mathcal{S}}(k)$ by counting the number of points in $V_{\mathrm{tl+br}}(k)$ and $V_{\mathrm{bl+tr}}(k)$ dominated by p_c. Instead of doing this separately for each $k \in \mathcal{K}$, we combine these computations. To this end, define $V_{\mathrm{tl+br}}$ and $V_{\mathrm{bl+tr}}$ to be the multisets obtained by merging all sets $V_{\mathrm{tl+br}}(k)$ and $V_{\mathrm{bl+tr}}(k)$, respectively. We have the following lemma, which readily follows from Lemma 3.

Lemma 4. *The number of influence regions containing p_c is equal to $\mu(p_c) := |\operatorname{Dom}(V_{\mathrm{bl+tr}}, p_c)| - |\operatorname{Dom}(V_{\mathrm{tl+br}}, p_c)|$.*

We compute the values $\mu(p_c)$ for all cells $c \in \mathcal{S}$ in a batched manner, as follows. Let $M[1..\sqrt{n}][1..2r+1]$ be a matrix that has the same number of rows and columns as $\mathcal{S}$. Each entry in M is an integer, and initially all entries are set to zero. Note that the points in $V_{\mathrm{bl+tr}}$ and $V_{\mathrm{tl+br}}$ are vertices of influence regions $A_{\mathcal{S}}(k)$, which means that they lie on corners of cells in $\mathcal{S}$. We now go over the points in the multisets $V_{\mathrm{bl+tr}}$ and $V_{\mathrm{tl+br}}$ one by one. When handling a point v we update an entry of M as follows. Let (i,j) be the center of the cell of $\mathcal{S}$ of which v is the bottom-left corner. If $v \in V_{\mathrm{bl+tr}}$ then we increment $M[i,j]$, and if $v \in V_{\mathrm{tl+br}}$ then we decrement $M[i,j]$. (If v is not the bottom-left corner of any cell in $\mathcal{S}$—this can happen when v lies on the top or right boundary of the strip—then we do nothing.)

Note that a center $p_c := (i,j)$ dominates a point $v \in V_{\mathrm{bl+tr}} \cup V_{\mathrm{tl+br}}$ if and only if v is the bottom-left corner of some cell with center (i',j') for $i' \leqslant i$ and $j' \leqslant j$. Hence,

$$\mu((i,j)) = \sum_{i' \leqslant i} \sum_{j' \leqslant j} M[i',j'].$$

This is the so-called *prefix sum* of entry $M[i,j]$. Given a matrix M, the prefix sums of all entries in M can be computed in linear time with a simple algorithm that scans the matrix row by row and uses that $\mu((i,j)) = \mu((i-1,j)) + \sum_{j' \leqslant j} M[i,j']$. This leads to the following lemma.

Lemma 5. *Let $\mathcal{S}$ be a strip that consists of at most $2r+1$ consecutive rows in $\mathcal{G}$. Given the influence regions $A_{\mathcal{S}}(k)$ for all $k \in \mathcal{K}_{\mathcal{S}}$, we can compute the value $\mathrm{rich}_{\mathrm{sq}}(c)$ for every cell $c \in \mathcal{S}$ in $O(r\sqrt{n})$ time in total.*

Putting it together. Since we have $\lceil \sqrt{n}/(2r+1) \rceil$ strips in total, Lemmas 2 and 5 and the fact that $|\mathcal{K}_{\mathcal{S}}| = O(r\sqrt{n})$ imply the following result.

Theorem 6. *Let $\mathcal{K}$ be a set of category values. Let $\mathcal{G}$ be a raster of n cells, each storing a value in $\mathcal{K}$. Algorithm SquareRichness computes the square richness for every cell in $\mathcal{G}$ in $O(n)$ time in total.*

2.3 An algorithm for disk richness

Our algorithm for the disk richness problem is based on a similar observation as we used for square windows.

Observation 7. *Let c and c' be two cells in $\mathcal{G}$. Then $c \in W_{\mathrm{d}}(c')$ if and only if $c' \in W_{\mathrm{d}}(c)$. Hence, $k \in \mathcal{K}_{\mathrm{d}}(c)$ if and only if $c \in W_{\mathrm{d}}(c')$ for at least one cell c' with $\mathrm{cat}(c') = k$.*

Thus we still have that $k \in \mathcal{K}_{\mathrm{d}}(c)$ iff c lies inside the union of all windows $W_{\mathrm{d}}(c')$ with $\mathrm{cat}(c') = k$. However, computing the union is significantly more complicated for disks, and we need to adapt our algorithm in several ways.

The global algorithm. Instead of partitioning the raster $\mathcal{G}$ into strip we now partition it into $O(n/r)$ *tiles*, where each tile consists of (at most) $\lfloor r/\sqrt{2} \rfloor \times \lfloor r/\sqrt{2} \rfloor$ cells. The tiles are processed as follows.

Consider a tile T and a window $W_{\mathrm{d}}(c)$ of a cell $c \in T$. Because the diameter of T—that is, the distance between the centers of the top-left and bottom-right cells in T—is at most the radius of our windows, we know that $W_{\mathrm{d}}(c)$ covers the centers of all cells in T. Based on this observation, we scan all cells in T to determine the subset $\mathcal{K}_{\mathrm{in}}$ of categories associated with the cells $c \in T$. For each cell $c \in T$, we initialize $\mathrm{rich}_{\mathrm{d}}(c)$ to $|\mathcal{K}_{\mathrm{in}}|$. Let $\mathcal{K}_{\mathrm{out}}$ denote the categories in $\mathcal{K} \setminus \mathcal{K}_{\mathrm{in}}$ whose influence regions overlap with T. We need to compute for all cells $c \in T$ the number of influence regions of categories in $\mathcal{K}_{\mathrm{out}}$ that contain c, and add this number to $\mathrm{rich}_{\mathrm{d}}(c)$. This is done as follows.

Let ℓ_{top}, ℓ_{bot}, ℓ_{left}, and ℓ_{right} denote the lines containing the top, bottom, left, and right edge of T, respectively. For a category $k \in \mathcal{K}_{\mathrm{out}}$, let $\mathcal{D}_{\mathrm{out}}(k)$ be the collection of windows $W_{\mathrm{d}}(c)$ that intersect T and have $\mathrm{cat}(c) = k$. Note that $\sum_k |\mathcal{D}_{\mathrm{out}}(k)| = O(r^2)$ and that the sets $\mathcal{D}_{\mathrm{out}}(k)$ can be generated in $O(r^2)$ time. We partition $\mathcal{D}_{\mathrm{out}}(k)$ into four subsets $\mathcal{D}_{\mathrm{top}}(k)$, $\mathcal{D}_{\mathrm{bot}}(k)$, $\mathcal{D}_{\mathrm{left}}(k)$, and $\mathcal{D}_{\mathrm{right}}(k)$. The set $\mathcal{D}_{\mathrm{top}}(k)$ consists of all windows in $\mathcal{D}_{\mathrm{out}}(k)$ whose centers lie above ℓ_{top}, and the set $\mathcal{D}_{\mathrm{bot}}(k)$ consists of all windows whose centers lie below ℓ_{bot}. The set $\mathcal{D}_{\mathrm{left}}(k)$ contains all windows whose centers lie to the left of ℓ_{left}, and which do not belong to $\mathcal{D}_{\mathrm{top}}(k)$ or $\mathcal{D}_{\mathrm{bot}}(k)$. The set $\mathcal{D}_{\mathrm{right}}(k)$ contains the rest of the windows in $\mathcal{D}_{\mathrm{out}}(k)$; note that these windows have their center to the right of ℓ_{right}.

For each of these four sets we will compute the part of their union inside T using a subroutine *EnvelopeInTile*. This part is bounded by a (lower, upper, left, or right) envelope, which will allow us to compute them efficiently. Then we will process the collection of these envelopes using a subroutine *RichnessInTile*. Thus we get the following global algorithm.

Algorithm *DiskRichness*($\mathcal{G}$)
1. Partition $\mathcal{G}$ into tiles of $\lfloor r/\sqrt{2} \rfloor \times \lfloor r/\sqrt{2} \rfloor$ cells each.
2. **for** each tile T in $\mathcal{G}$
3. **do** Determine the set $\mathcal{K}_{\text{in}}$ of all categories stored in T, and set $\text{rich}_{\text{d}}(c) = |\mathcal{K}_{\text{in}}|$ for every cell $c \in T$.
4. Generate the sets $\mathcal{D}_{\text{top}}(k)$, $\mathcal{D}_{\text{bot}}(k)$, $\mathcal{D}_{\text{left}}(k)$, and $\mathcal{D}_{\text{right}}(k)$ for all $k \in \mathcal{K}_{\text{out}}$.
5. **for** every $k \in \mathcal{K}_{\text{out}}$
6. **do** $\mathcal{E}_{\text{top}}(k) \leftarrow$ *EnvelopeInTile*($\mathcal{D}_{\text{top}}(k)$)
7. $\mathcal{E}_{\text{bot}}(k) \leftarrow$ *EnvelopeInTile*($\mathcal{D}_{\text{bot}}(k)$)
8. $\mathcal{E}_{\text{left}}(k) \leftarrow$ *EnvelopeInTile*($\mathcal{D}_{\text{left}}(k)$)
9. $\mathcal{E}_{\text{right}}(k) \leftarrow$ *EnvelopeInTile*($\mathcal{D}_{\text{right}}(k)$)
10. Run *RichnessInTile* on the collection of envelopes computed in Steps 5–9.

It remains to describe the subroutines *EnvelopeInTile* and *RichnessInTile*.

Subroutine *EnvelopeInTile*. Consider a tile T and a collection $\mathcal{D}_{\text{bot}}(k)$ of disks whose centers lie below ℓ_{bot}, the line containing the bottom edge of T. We describe a subroutine *EnvelopeInTile* that computes the part of $\text{Union}(\mathcal{D}_{\text{bot}}(k))$ inside T. The parts of $\text{Union}(\mathcal{D}_{\text{top}}(k))$, $\text{Union}(\mathcal{D}_{\text{left}}(k))$, and $\text{Union}(\mathcal{D}_{\text{right}}(k))$ inside T can be computed similarly.

As in the case of square windows, we may assume that we generated $\mathcal{D}_{\text{bot}}(k)$ such that the disks in $\mathcal{D}_{\text{bot}}(k)$ are sorted from left to right. If we have multiple disks whose centers have the same x-coordinate we only need to keep the highest one. Thus we can also assume that $\mathcal{D}_{\text{bot}}(k)$ (and similarly $\mathcal{D}_{\text{top}}(k)$, $\mathcal{D}_{\text{left}}(k)$, and $\mathcal{D}_{\text{right}}(k)$) have size $O(r)$.

Let ℓ^+_{bot} denote the halfplane above ℓ_{bot}. For each disk $W_{\text{d}}(c) \in \mathcal{D}_{\text{bot}}(k)$, let $\alpha(c) := \partial W_{\text{d}}(c) \cap \ell^+_{\text{bot}}$ denote the part of the boundary of $W_{\text{d}}(c)$ above ℓ_{bot}. Note that $\text{Union}(\mathcal{D}_{\text{bot}}(k))$ in T is bounded from above by (a part of) the upper envelope of the arcs $\alpha(c)$; see Fig. 2. Thus computing $\text{Union}(\mathcal{D}_{\text{bot}}(k))$ in T boils down to computing this envelope. To do this we need the following lemma.

Lemma 8. *Let $\mathcal{D}$ be a set of disks intersecting a horizontal line ℓ whose centers lie below ℓ. Let D be the disk in $\mathcal{D}$ whose center has maximum x-coordinate. If D contributes to the upper envelope of the disks above ℓ, then its contribution is a single arc that is the rightmost arc of the envelope.*

Proof. Let p_1 be the rightmost intersection of the boundary of D with ℓ. Let $\mathcal{E}$ be the upper envelope of all the windows in $\mathcal{D}$ except for D. We consider two cases: whether or not p_1 lies under $\mathcal{E}$.

In the first case, p_1 is not under $\mathcal{E}$. Consider traversing the boundary of D counter-clockwise starting just to the left of p_1. During this traversal, let p_2 be the first intersection point that we encounter between the boundary of D with $\mathcal{E}$ or ℓ. If p_2 lies on ℓ then we are done since D contributes to the upper envelope the arc from p_1 to p_2, and the rest of the boundary of D lies under ℓ. Otherwise,

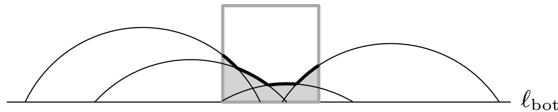

Figure 2: The part of $\text{Union}(\mathcal{D}_{\text{bot}}(k))$ inside T is shown in grey. It is bounded from above by the upper envelope of the arcs $\alpha(c)$.

p_2 lies on the boundary of a window $D' \in \mathcal{D}$. Note that p_2 is one of the two intersection points between the boundaries of D' and D. We now claim that the other intersection point p_2' must lie below ℓ, which implies that D cannot contribute anything other than the arc from p_1 to p_2 to the upper envelope.

Consider the centers q and q' of D and D', respectively. Both intersection points p_2 and p_2' lie on the perpendicular bisector of $\overline{qq'}$. Furthermore they lie in opposite directions from the midpoint of $\overline{qq'}$. Since both q and q' lie under ℓ, so must the midpoint of $\overline{qq'}$. Since p_2 lies over ℓ it must be that p_2', which is in the opposite direction of the midpoint of $\overline{qq'}$, must lie under ℓ.

In the second case, p_1 lies under $\mathcal{E}$. Thus, it is inside another window $D' \in \mathcal{D}$. Let I' and I be the intervals formed by intersecting ℓ with D' and D respectively. We first claim I' contains I. Again, let q' and q be the centers of D' and D, respectively. Notice that the midpoint of I has the same x-coordinate as q, and the same holds for the midpoint of I' and q'. Since q is to the right of q', we have that the midpoint of I is to the right of the midpoint of I'. Thus, more than the right half of I is contained in the right half of I'. Therefore, also the left half of I is contained in I'. Given this configuration, and since the two disks are not identical, if the boundary of D were to intersect with the boundary of D' above ℓ, it would need to do so twice. However, we have already argued that at most one intersection between the two boundaries can occur above ℓ. So, window D does not contribute to the upper envelope of $\mathcal{D}$, and the lemma follows. □

Let $\text{Arcs}(\mathcal{D}_{\text{bot}}(k)) := \{\alpha(c) : W_{\text{d}}(c) \in \mathcal{D}_{\text{bot}}(k)\}$ be the set of arcs of which we want to compute the upper envelope, sorted from left to right. (More precisely, sorted according to the x-coordinate of the centers of the corresponding disks $W_{\text{d}}(c)$.) Our algorithm to compute the upper envelope of $\text{Arcs}(\mathcal{D}_{\text{bot}}(k))$ is similar to the algorithm we used for constructing the upper envelope of a set of squares: we go over the arcs in order, and when adding an arc $\alpha(c)$ we remove the part of the current envelope below $\alpha(c)$ and we add the relevant part of $\alpha(c)$. The details are as follows.

We maintain a list $\mathcal{L}$ that stores the portions of the arcs $\alpha(c)$ appearing on the current envelope, ordered from left to right. To process the next arc $\alpha(c)$, we first check if the right endpoint of $\alpha(c)$ lies to the right of the right endpoint of the last arc in the list. If this is not the case, then $\alpha(c)$ does not contribute to the envelope (by Lemma 8) and we are done with $\alpha(c)$. Otherwise we start walking back along $\mathcal{L}$ as long as we encounter envelope arcs that lie entirely below $\alpha(c)$, and we remove these arcs from $\mathcal{L}$. The walk continues until either (i) we encounter an arc β that is intersected by $\alpha(c)$, or (ii) we encounter an arc that lies fully to the left of $\alpha(c)$, or (iii) the list $\mathcal{L}$ becomes empty.

In case (i) we shrink β by removing the part of β below $\alpha(c)$ and we append the part of $\alpha(c)$ to the right of the intersection point $q := \alpha(c) \cap \beta$ to the list $\mathcal{L}$. By Lemma 8 the arc $\alpha(c)$ does not contribute anything to the left of q, and so we can stop. In case (ii) and (iii) we simply append the entire arc $\alpha(c)$ to $\mathcal{L}$, and we can stop as well.

The running time of the algorithm described above is linear in the number of arcs it processes, since each arc is inserted and deleted at most once. This leads to the following lemma. Let $\mathcal{E}_{\text{bot}}(k)$ denote the envelope that forms the

upper boundary of Union($\mathcal{D}_{\text{bot}}(k)) \cap T$. Define $\mathcal{E}_{\text{top}}(k)$, $\mathcal{E}_{\text{left}}(k)$, and $\mathcal{E}_{\text{right}}(k)$ similarly for $\mathcal{D}_{\text{top}}(k)$, $\mathcal{D}_{\text{left}}(k)$, and $\mathcal{D}_{\text{right}}(k)$.

Lemma 9. *EnvelopeInTile can compute $\mathcal{E}_{\text{bot}}(k)$, $\mathcal{E}_{\text{top}}(k)$, $\mathcal{E}_{\text{left}}(k)$, and $\mathcal{E}_{\text{right}}(k)$, in $O(r)$ time.*

We now present a property of the envelopes that is useful for the rest of our analysis.

Lemma 10. *Each envelope $\mathcal{E}_{\text{bot}}(k)$, $\mathcal{E}_{\text{top}}(k)$, $\mathcal{E}_{\text{left}}(k)$, and $\mathcal{E}_{\text{right}}(k)$ intersects $O(r)$ cells of T.*

Proof. We prove the lemma for the envelope $\mathcal{E}_{\text{bot}}(k)$; the same arguments apply to the other envelopes.

We call a point $q \in \mathcal{E}_{\text{bot}}(k)$ an *extremum* if q is a local maximum or a local minimum on $\mathcal{E}_{\text{bot}}(k)$, and q is not the leftmost or rightmost point of $\mathcal{E}_{\text{bot}}(k)$. Note that a local maximum is the top point of a window; a local minimum is a vertex of $\mathcal{E}_{\text{bot}}(k)$, that is, an intersection between two window boundaries, or between a window boundary and ℓ_{bot}. We split $\mathcal{E}_{\text{bot}}(k)$ into pieces at the extrema. As we traverse such a piece from left to right, the y-coordinate is monotonically increasing or decreasing. This means that each piece has Euclidean length at most $2r$. In particular, the first and last pieces have total length at most $4r$. All other pieces are bounded by two extrema. Consider such a piece, which lies between successive extrema b and b'. Then the length of the envelope between b and b' is at most $|x(b)-x(b')|+|y(b)-y(b')|$. We claim that $|x(b) - x(b')| \geqslant |y(b) - y(b')|$. This implies that the length of $\mathcal{E}_{\text{bot}}(k)$ between the first and last extremum is bounded by $2r$, and thus that the total length of $\mathcal{E}_{\text{bot}}(k)$ is at most $6r$. This bound on the length implies that $\mathcal{E}_{\text{bot}}(k)$ crosses only $O(r)$ cells.

It remains to prove the claim. Assume without loss of generality that b is a local maximum and b' is a local minimum, and let $D \in \mathcal{D}_{\text{bot}}(k)$ be the window whose top is b. Consider the rectangle B whose opposite corners are b and b'. Let q be the intersection between the boundary of D with the bottom edge of B. Then we have $|x(b) - x(q)| \geqslant |y(b) - y(q)| \geqslant |y(b) - y(b')|$, where the first inequality follows from the fact that the center of D lies below ℓ_{bot}. □

Subroutine *RichnessInTile*. Subroutine *RichnessInTile* takes as input the four envelopes $\mathcal{E}_{\text{bot}}(k)$, $\mathcal{E}_{\text{top}}(k)$, $\mathcal{E}_{\text{left}}(k)$, and $\mathcal{E}_{\text{right}}(k)$ for each of the categories $k \in \mathcal{K}_{\text{out}}$. Its task is to determine for each cell $c \in T$ the number of categories $k \in \mathcal{K}_{\text{out}}$ such that c is covered by a window in $\mathcal{D}_{\text{out}}(k)$. The latter is the case if (the center of) c lies below $\mathcal{E}_{\text{bot}}(k)$, above $\mathcal{E}_{\text{top}}(k)$, to the left of $\mathcal{E}_{\text{left}}(k)$, or to the right of $\mathcal{E}_{\text{right}}(k)$. *RichnessInTile* is based on this observation, and consists of two steps.

Step I: Computing entry and exit cells in each column. Consider an envelope $\mathcal{E}$ (which is one of the four envelopes $\mathcal{E}_{\text{bot}}(k)$, $\mathcal{E}_{\text{top}}(k)$, $\mathcal{E}_{\text{left}}(k)$, or $\mathcal{E}_{\text{right}}(k)$, for some k) and a column $T[i, 1..\lfloor r/\sqrt{2} \rfloor]$ of the tile T. Let $A(\mathcal{E})$ denote the area enclosed by $\mathcal{E}$ and the corresponding edge of T. The area $A(\mathcal{E}_{\text{bot}}(k))$, for example is enclosed by $\mathcal{E}_{\text{bot}}(k)$ and the bottom edge of T. As we traverse the column from bottom to top, we may enter and exit the area $A(\mathcal{E})$ one or more times. We call the cells where this happens *entry cells* and *exit cells* (for the given envelope) where the first cell in a column is also considered an entry cell if it lies inside

$A(\mathcal{E})$; see Fig. 3. The envelopes $\mathcal{E}_{\text{bot}}(k)$ and $\mathcal{E}_{\text{top}}(k)$ have at most one entry and exit cell per row, while $\mathcal{E}_{\text{left}}(k)$ or $\mathcal{E}_{\text{right}}(k)$ may have multiple entry and exit cells. Note that entry and exit cells can coincide, if we have a cell whose center is inside $A(\mathcal{E})$ while the centers of both adjacent cells are outside.

In Step I we compute, for each k and each envelope $\mathcal{E}_{\text{bot}}(k)$, $\mathcal{E}_{\text{top}}(k)$, $\mathcal{E}_{\text{left}}(k)$, and $\mathcal{E}_{\text{right}}(k)$, all entry and exit cells. This is done as follows.

First consider $\mathcal{E}_{\text{bot}}(k)$. Here a cell $T[i,j]$ is an exit cell if $\mathcal{E}_{\text{bot}}(k)$ crosses the segment connecting (i,j) and $(i,j+1)$ or if $T[i,j]$ is the top cell in a column and $\mathcal{E}_{\text{bot}}(k)$ passes above (i,j). Thus we can compute all exit cells by tracing $\mathcal{E}_{\text{bot}}(k)$ through T, visiting all cells it crosses. In each column where $\mathcal{E}_{\text{bot}}(k)$ has an exit cell, the bottom cell of the column is an entry cell. By Lemma 10, we can thus compute all entry and exit cells in $O(r)$ time. The entry and exit cells for $\mathcal{E}_{\text{top}}(k)$ can be computed in a similar way.

Now consider $\mathcal{E}_{\text{left}}(k)$; the envelope $\mathcal{E}_{\text{right}}(k)$ is handled similarly. We first compute, in a similar way as above, the *row exit cells* with respect to $\mathcal{E}_{\text{left}}(k)$; these are the cells $T[i,j]$ such that (i,j) lies to the left of $\mathcal{E}_{\text{left}}(k)$ while $(i+1,j)$ lies to the right. We can now decide which cells in a row are column entry and exit cells by looking at the row exit cells in adjacent columns. Indeed, suppose $T[i,j]$ is the row exit cell in the i-th row, and $T[i+1,j']$ is the row exit cell in the $(i+1)$-th column. Then if $j' > j$ then all cells $T[i+1,j'']$ with $j < j'' \leqslant j'$ are entry cells, and if $j' < j$ then all cells $T[i,j'']$ with $j' < j'' \leqslant j$ are exit cells. Because the total number of entry and exit cells is $O(r)$—this follows from Lemma 10 and the fact that for each entry or exit cell, the envelope must either cross that cell or a neighboring cell—we spend $O(r)$ time in total to compute the entry and exit cells for $\mathcal{E}_{\text{left}}(k)$.

Step II: Computing the richness values. With the set of all entry and exit cells available, we can compute the richness of the cells in T column by column. To process a column, we need a counter $count[k]$ for each $k \in \mathcal{K}_{\text{out}}$ and a global counter $count_{\text{g}}$, all initialized to zero. We use $count[k]$ to count how many of the regions $A(\mathcal{E}_{\text{bot}}(k))$, $A(\mathcal{E}_{\text{top}}(k))$, $A(\mathcal{E}_{\text{left}}(k))$, and $A(\mathcal{E}_{\text{right}}(k))$ contain the current cell; thus $0 \leqslant count[k] \leqslant 4$. The global counter $count_{\text{g}}$ counts how many of the counters $count[k]$ are positive for the current cell. Thus $count_{\text{g}}$ indicates the number of categories $k \in \mathcal{K}_{\text{out}}$ such that c is covered by a window in $\mathcal{D}_{\text{out}}(k)$.

Note that a cell can be an entry and/or exit cell for several different categories k. As we traverse a column we handle each encountered cell as follows: for all categories k for which the current cell is an entry cell we increment $count[k]$ and if $count[k]$ was zero we also increment $count_{\text{g}}$. Next we output $count_{\text{g}}$ as

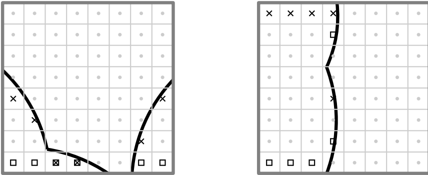

Figure 3: Entry cells are indicated by small squares, exit cells are indicated by crosses. On the left the entry and exit cells for $\mathcal{E}_{\text{bot}}(k)$ are shown, on the right for $\mathcal{E}_{\text{left}}(k)$.

the richness of the current cell. Finally, for all categories k for which the current cell is an entry cell we decrement *count*$[k]$ and if *count*$[k]$ becomes zero we also decrement $count_{\mathrm{g}}$.

As explained above, Step I of the subroutine spends $O(r)$ time for each $k \in \mathcal{K}_{\mathrm{out}}$. Thus the total time for Step I, and also the total number of entry and exit cells generated, is $O(r \cdot |\mathcal{K}_{\mathrm{out}}|) = O(rK)$. Step II spends $O(1)$ time at each cell in T, plus $O(1)$ time for each of the entry and exit points computed in Step I. We get the following lemma.

Lemma 11. *RichnessInTile runs in $O(r^2 + rK)$ time.*

Putting it together. Since we have $O(n/r^2)$ tiles in total, Lemmas 9 and 11 together imply the following result.

Theorem 12. *Let $\mathcal{K}$ be a set of category values. Let $\mathcal{G}$ be a raster of n cells, each storing a value in $\mathcal{K}$. Algorithm DiskRichness computes the disk richness for every cell in $\mathcal{G}$ in $O((1 + K/r)n)$ time in total.*

3 Colored range counting

Let P be a set of n colored points in $\mathbb{R}^d$—note that we do not require the points in P to form a grid—and let $\mathcal{K} := \{1, \ldots, K\}$ denote the set of colors associated to the points in P. In this section we study data structures for *colored range counting*: given a query range R, compute the number of distinct colors associated with the points of P inside R.

3.1 Square ranges in $\mathbb{R}^2$

We start by presenting a data structure for colored range counting in $\mathbb{R}^2$, for the case where the query range is an (arbitrarily-sized) square. Our solution in Section 2.2 hinged on the duality property of Observation 1. In the current setting we can no longer use this duality property, because different query squares may have different sizes. In order to obtain a duality property for variable-size queries, we need to move into three dimensions.

Definitions. We define the *radius* of a square be half of its side length. For a point p, let $W_r(p)$ be the square window of radius r centered at p, and let $\mathcal{K}_r(p) \subseteq \mathcal{K}$ be the set of all colors that occur in the window $W_r(p)$. The *richness* of $W_r(p)$ is $\mathrm{rich}_r(p) := |\mathcal{K}_r(p)|$. For a color $k \in \mathcal{K}$, let $P_k \subseteq P$ be the set of points with color k.

If p is a point on the plane $z = 0$, let $p \uparrow z'$ denote the same point lifted to the plane $z = z'$. This notation extends to sets of points. Let $\mathrm{Pyramid}(p)$ be an infinitely large upside-down 3-dimensional pyramid with its apex at some point p in the $z = 0$ plane. Thus $\mathrm{Pyramid}(p) = \bigcup_{r \in \mathbb{R}^+} (W_r(p) \uparrow r)$.

Main idea. The following lemma is a duality property that is useful for variable-size queries. The lemma is well known, but for completeness we provide a proof here.

Lemma 13. *For any two points p and q we have $p \in W_r(q)$ if and only if $q \uparrow r \in \text{Pyramid}(p)$.*

Proof. The intersection of Pyramid(p) with the plane $z = r$ is an elevated window $W_r(p) \uparrow r$. Thus, $q \uparrow r \in \text{Pyramid}(p)$ if and only if $q \uparrow r \in W_r(p) \uparrow r$. Clearly, $q \uparrow r \in W_r(p) \uparrow r$ if and only if $q \in W_r(p)$. By Observation 1, $q \in W_r(p)$ if and only if $p \in W_r(q)$. □

Lemma 13 implies that $k \in \mathcal{K}_r(q)$ if and only if $q \uparrow r$ lies in the union of all pyramids Pyramid(p) where $p \in P_k$. Similarly to Section 2.2, we therefore define the influence region $A(k)$ as the union of pyramids of the points in P_k.

Our data structure now works as follows. We construct for each color $k \in \mathcal{K}$ the influence region $A(k)$. For a query range with radius r_q and center q, the richness $\text{rich}_{r_q}(q)$ is the number of influence regions containing $q \uparrow r_q$. To compute this number we decompose the influence regions into simpler parts, and count how many of these simpler parts contain point $q \uparrow r_q$.

Complexity of influence regions. To be able to analyze the efficiency of our data structure we need to bound the complexity of the influence regions.

Lemma 14. *For each color $k \in \mathcal{K}$, the complexity of the influence region $A(k)$ is $O(|P_k|)$.*

Proof. An edge of an influence region can either be an edge of one of the pyramids or an intersection of faces of pyramids. We call the former *pyramid edges* and the latter *intersection edges*. There are $4|P_k|$ pyramid edges, because when a pyramid edge enters another pyramid Pyramid(p), it stays inside Pyramid(p). To bound the number of intersection edges, we note that the projection of these edges onto the plane $z = 0$ is equal to the L_∞-Voronoi diagram of the points in P_k. (The relation between Voronoi diagrams in the plane and the lower envelope of certain cones is well known. For the Euclidean distance the cones are circular; for the L_∞-distance the cones are as defined above.) A linear bound on the number of intersection edges thus follows from the fact that L_∞-Voronoi diagrams have linear complexity [1]. □

Decomposition of influence regions. Consider the map $\mathcal{M}_k$ formed by projecting the faces of $A(k)$ to the plane $z = 0$. We know that M_k has $O(|P_k|)$ faces due to Lemma 14. The map is octilinear since L_∞-Voronoi diagrams are octilinear and the projections of the pyramid edges align with four of these eight directions. We partition each face of $\mathcal{M}_k$ into smaller faces with constant complexity by shooting vertical rays up and down from each vertex. The resulting faces are vertical slabs that may be cut at the ends with horizontal or diagonal cuts. Let the new map with these simpler faces be $\mathcal{M}'_k$. It also has $O(|P_k|)$ faces, each with $O(1)$ complexity.

We decompose $A(k)$ into $O(|P_k|)$ pieces $\mathcal{F}_k$, such that each piece is the subset of $A(k)$ that lies above some face of $\mathcal{M}'_k$. We call these pieces *towers*. Thus, $\text{rich}_{r_q}(q)$ is the number of towers (across the influence regions of all colors) that contain point $q \uparrow r_q$. The following lemma describes a data structure that can efficiently compute the number of towers stabbed by an arbitrary query point.

Lemma 15. *There exists a data structure for counting the number of towers stabbed by a query point that uses $O(n \operatorname{polylog} n)$ space and has $O(\operatorname{polylog} n)$ query time.*

Proof. Every input tower is the intersection of at most 5 halfspaces. Each halfspace has one of 12 orientations. We partition the input towers into types based on number and orientation of their defining halfspaces. There are a constant number of types of towers. We handle each type of tower separately and to answer a query we simply add the counts for each type. Fix a type of tower formed by b halfspaces with b different orientations. For each orientation of halfspaces, we create a coordinate axis that is normal to the bounding planes of the halfspaces and that increases towards the interiors of the halfspaces. We transform each tower F into a b-dimensional point f. For each defining halfspace of F, the coordinate of f on the associated coordinate axis is the value on the axis that the halfspace's bounding plane intersects. We also transform q into a b-dimensional point q'. For each coordinate axis, the coordinate value of q' is the projection of q onto the axis. In this way, $q \in F$ if and only if q' dominates f. Computing the number of b-dimensional points dominated by a b-dimensional point is an instance of b-dimensional dominance range counting, which can be solved in $O(n \operatorname{polylog} n)$ space and $O(\operatorname{polylog} n)$ query time for any constant b via range trees [4]. □

We conclude with the following theorem.

Theorem 16. *Let P be a colored point set in $\mathbb{R}^2$. We can store P in a data structure using $O(n \operatorname{polylog} n)$ space such that colored range counting queries with square ranges can be answered in $O(\operatorname{polylog} n)$ time.*

3.2 Cube ranges in $\mathbb{R}^3$

We now consider the 3-dimensional generalization of the problem studied in the previous section. Thus P is a set of n colored points in $\mathbb{R}^3$ and the query ranges are (arbitrarily-sized) cubes. We will show that we should not expect to obtain a data structure of similar efficiency—$O(n \operatorname{polylog} n)$ storage and $O(\operatorname{polylog} n)$ query time—as in the square case. Our negative result holds in an *offline* setting, that is, when all query ranges are known in advance.

We begin with a simple generalization of a technique of Kaplan *et al.* [11] which is able to reduce Boolean matrix multiplication to offline colored range counting problems. The proof of this lemma is similar to the proof given by Kaplan *et al.*.

Lemma 17. *Let $\mathfrak{R}$ be a family of ranges in $\mathbb{R}^d$. Suppose that there exist two point sets P, Q in $\mathbb{R}^d$ and a set of ranges $\mathcal{R} \subseteq \mathfrak{R}$ such that:*

- $|P| = |Q| = \sqrt{n}$ *and* $|\mathcal{R}| = n$,
- *for every* $(p, q) \in P \times Q$, *there is a range* $R \in \mathcal{R}$ *such that* $R \cap (P \cup Q) = \{p, q\}$.

Then we can multiply two Boolean matrices of size $\sqrt{n} \times \sqrt{n}$ by performing n colored range counting queries with ranges from $\mathfrak{R}$ on a colored point set containing at most $2n$ colored points in $\mathbb{R}^d$.

Proof. We are given two $\sqrt{n} \times \sqrt{n}$ Boolean matrices $A = \{a_{i,j}\}$ and $B = \{b_{i,j}\}$. We wish to compute $C = \{c_{i,j}\}$ where $c_{i,j} = \bigvee_k (a_{k,i} \wedge b_{j,k})$.

We assign each row k of A to a distinct point $p_k \in P$. For each true entry $a_{k,i}$, we construct a point of color i at position p_k. We also store a count row_k with the number of true entries in the row. Similarly, we assign each column k of B to a distinct point $q_k \in Q$. For each true entry $b_{j,k}$, we construct a point of color j at position q_k. We also store a count col_k of the number of true entries in the column. We thus construct a set P_A of at most n points for the rows of A and a set P_B of at most n points for the columns of B.

Note that $c_{i,j}$ is true if and only if the range $R \in \mathcal{R}$ that contains only p_i and q_j contains a duplicate color k, because then there is a k with $a_{k,i} = \text{TRUE}$ and $b_{j,k} = \text{TRUE}$. To decide whether or not there is a duplicate color, we count the number of colors in R and compare it to $\text{row}_i + \text{col}_j$. We can thus evaluate all n cells of C by performing n colored range counting queries on the set $P_A \cup P_B$. $\square$

Remark. The proof of Lemma 17 exploits the ability to create point sets with many differently colored points at the same position. (Thus the point set is actually a multiset.) In our setting, where the ranges are cubes in $\mathbb{R}^3$, it is straightforward to perturb the points and ranges so that they do not share any coordinate values.

The reduction in the proof of the lemma above yields an instance of an offline range searching problem, since all queries can be generated in advance. Using this lemma, we obtain the following theorem.

Theorem 18. *Suppose there is a data structure for offline colored range counting in $\mathbb{R}^3$ with cube ranges that has $T_{\text{prep}}(n)$ preprocessing time and $T_{\text{query}}(n)$ query time. Then we can multiply two Boolean matrices of size $\sqrt{n} \times \sqrt{n}$ in time $O(T_{\text{prep}}(2n) + n \cdot T_{\text{query}}(2n))$ time.*

Proof. We intend to invoke Lemma 17 where $\mathfrak{R}$ is the set of all axis-aligned cubes in $\mathbb{R}^3$. All points of the sets P and Q that we construct will lie on the plane $z = x + y$. Let a *cubic section* be any two-dimensional shape formed by the intersection of an axis-aligned cube with the plane $z = x + y$. We choose a new coordinate system so that the plane $z = x + y$ in the old coordinate system becomes the plane $z = 0$ in the new coordinate system and each cubic section has a side parallel to the new x-axis. Then it is sufficient to consider the case where $\mathfrak{R}$ is the set of all cubic sections in $\mathbb{R}^2$.

A cubic section is the intersection of two equilateral triangles with the same centers such that one has a bottom side that is parallel to the x-axis and the other has a top side that is parallel to the x-axis. (These triangles are the intersections of opposing octants with the plane $z = 0$. The intersection of

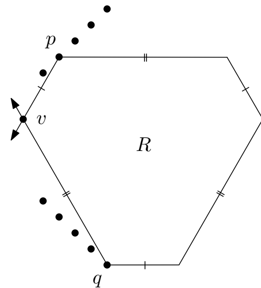

Figure 4: The cubic section R that hits p and q.

the opposing octants is the cube from which the cubic section arises.) A cubic section has either three or six sides. If it has three sides, then it is an equilateral triangle. Otherwise, its opposite sides are parallel and side lengths alternate between two values around the cubic section.

Let P be $\sqrt{n}$ points evenly distributed along the segment from $(0, 1)$ to $(1, 2)$. Let Q be $\sqrt{n}$ points evenly distributed along the segment from $(0, -1)$ to $(1, -2)$. For each pair $(p, q) \in P \times Q$, we construct a cubic section R which contains only points p and q. From p, we shoot a ray at angle $-(2/3)\pi$ with the positive x-axis. From q, we shoot a ray at angle $(2/3)\pi$ with the positive x-axis. Let v be the point of intersection of these two rays. By our construction, v must exist. Let R be the unique cubic section including the two sides $\overline{pv}$ and $\overline{qv}$; see Fig. 4.

Since p and q are vertices of R, then R clearly contains p and q. The other points of P lie on a tangent of vertex p of R and so they do not lie in R, which is convex. The same is true of the other points of Q.

It is straightforward to map our points back to the plane $z = x + y$ and our cubic sections to the cubes that realize them. Therefore, applying Lemma 17 completes the proof of the theorem. $\square$

3.3 Fixed-size cube ranges in $\mathbb{R}^3$

The best known algorithm to multiply $\sqrt{n} \times \sqrt{n}$ Boolean matrices runs in time $O(n^{1.19})$ [12]. Theorem 18 thus implies it is unlikely that we can obtain a data structure with $O(n \operatorname{polylog} n)$ space and $O(\operatorname{polylog} n)$ query time for colored range counting in $\mathbb{R}^3$ when the ranges are arbitrary cubes. Such performance can be achieved, however, in the special case where the query cubes are restricted to be of a fixed size, as we describe next.

Theorem 19. *There exists a data structure for fixed-size cube colored range counting that requires $O(n \operatorname{polylog} n)$ space and $O(\operatorname{polylog} n)$ query time.*

Next we sketch the proof of this theorem. Observe that the duality property stated as Observation 1 for the square case still applies: a point $p \in P$ is in a query cube C of "radius" r if and only if the center of C is in the cube of radius r centered at p. Thus the influence region for color $k \in \mathcal{K}$ is the union of the radius-r cubes centered at the points in P_k. Boissonnat *et al.* [5] show that the union of fixed-sized axis-aligned cubes has linear complexity. It is then possible to generalize Lemmas 3 and 4, which allows us to reduce counting the

number of influence regions stabbed by the center of a query cube to a constant number of 3-dimensional dominance range counting queries on the $O(n)$ vertices of the influence regions. We can perform these dominance range counting queries using range trees [4], which yields the desired bounds.

4 Experimental Evaluation

We implemented algorithm *StripRichness* and algorithm *TileRichness* and we conducted experiments in order to measure their performance in practice. The implementations were developed in C++, and all experiments were run on a workstation with an Intel core i5-2430M CPU. This is a four-core processor with 2.40GHz per core. The main memory of this computer is 7.8 Gigabytes. Our implementations run on a Linux Ubuntu operating system, release 12.04.

We used two raster datasets. The first raster stores categorical data and was extracted from the Harmonized World Soil Database (version 1.2), a raster that maps soil types over the entire planet [7]. The original raster consists of 43,200 × 21,600 cells, and each cell stores an integer in the range $[0, 32000]$ representing a soil type. From this dataset we extracted a smaller raster of 11,000 × 11,000 cells, representing an area that includes regions from Europe, Northern Africa and Western Asia. We refer to the extracted raster as `soils`. The number of distinct category values in `soils` is 7,639. The second dataset we used is a DEM that represents the landscape around mountain Glacier Peak, Washington state. This dataset was acquired from the U.S. Geological Survey (USGS) online server [17] and consists of 10,812 × 10,812 cells. Each cell in this raster stores a floating-point value in the range $[25.46, 3284.32]$. We processed the raster by rounding the cell values to integers, yielding a raster with 3,259 different values. We refer to the resulting raster as `peak`.

In the first experiment, we measured the running time of our implementations with respect to the size of the input raster, while using a fixed window size and a fixed maximum number of categories. In particular, for every integer $m \in [1, 22]$ we extracted from `soils` the raster that starts from the top-left corner of the dataset and consists of $(500m) \times (500m)$ cells. We then fixed the maximum number of distinct categories that appear in each extracted raster; we measured the minimum and the maximum value found in the raster, and we partitioned the interval defined by these two values into K smaller intervals of equal size. Then, each cell c was assigned a category value $i \in \{1, \ldots, K\}$, if the original value of c belonged to the i-th of these intervals. In this experiment, we fixed the value of K to one hundred. We then ran our implementation of *SquareRichness* on each of these rasters with square windows of $(2r+1)\times(2r+1)$ cells, and *DiskRichness* with disks of radius r, using $r = 50$. Similarly, from dataset `peak` we extracted rasters of $(312+500m)\times(312+500m)$ cells for integer $m \in [0, 21]$. We ran our implementations also on these rasters, using the same values for parameters K and r. As reference, we also ran the experiments using two programs (one for square richness and one for disk richness) that compute the richness values in a naive manner in $O(nr^2)$ time. We refer to these two programs as *NaiveSquares* and *NaiveDisks*. The results for this experiment are illustrated in Fig. 5.

We observe that both *SquareRichness* and *DiskRichness* are outstandingly faster than the naive programs. We also see that *DiskRichness* performs worse

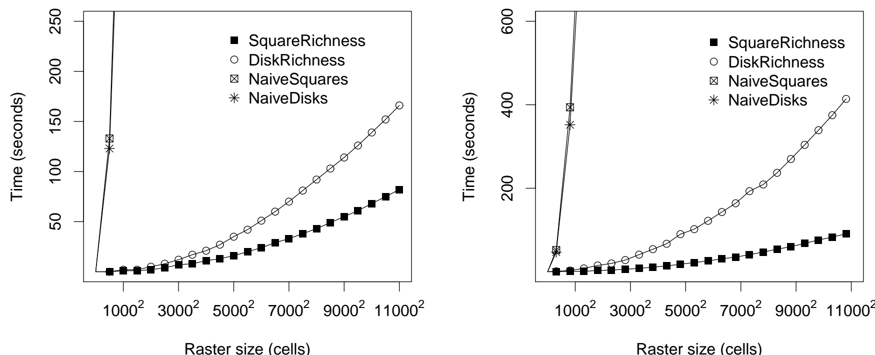

Figure 5: The running times of our implementations on rasters of variable size, using fixed values for parameters $r = 50$ and $K = 100$. Left: the running times for rasters extracted from `soils` dataset. Right: the running times for rasters extracted from `peak` dataset.

on `peak` than on `soils`. This is possibly due to the distribution of category values in `soils`. Recall that *DiskRichness* spends $O(|\mathcal{K}_{\text{out}}|r+r^2)$ time to process a tile of $r/\sqrt{2}\times r/\sqrt{2}$ cells in the raster. Set $\mathcal{K}_{\text{out}}$ consists of the category values whose influence regions overlap with the tile, but no cell in the tile stores any of these values. Therefore, if the average size of $\mathcal{K}_{\text{out}}$ is very small among all tiles in the raster then the running time converges to $O(n)$. Indeed, for $r = 50$ the average size of $\mathcal{K}_{\text{out}}$ among the tiles in the entire `soils` raster is roughly five, while the corresponding number for `peak` is roughly 384.5 .

In the second experiment, we measured the performance of our implementations using windows of variable size, with fixed input size and maximum number of categories. From each dataset we extracted a raster of 5,500 × 5,500 cells and we fixed the maximum number of categories in this raster to $K = 100$. We then ran our implementations on the resulting rasters for $r = 10 + 100 \times m$ for integer $m \in [0, 26]$. Fig. 6 shows the results for this experiment.

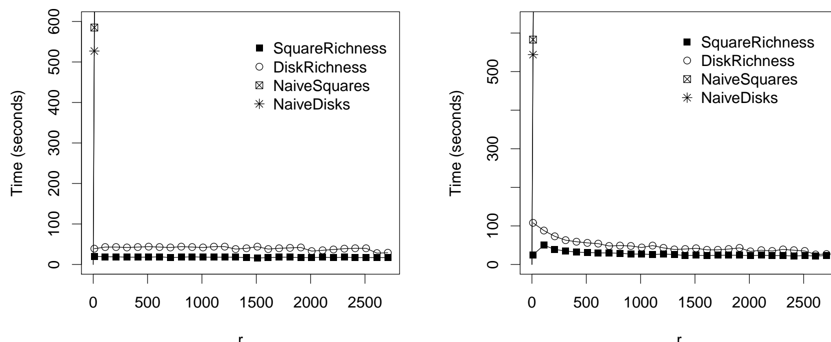

Figure 6: The running times of our two implementations using windows of variable size on a raster of 5,500 × 5,500 cells and with $K = 100$. Left: the running times for a raster extracted from `soils` dataset. Right: the running times for a raster extracted from `peak` dataset.

Again, *SquareRichness* and *DiskRichness* exhibit remarkable performance compared to the naive programs, even for the case where $r = 10$. For larger values of this parameter, each of the naive programs takes several hours to execute. On the other hand, the running times of our algorithms decrease as r becomes larger, especially for *DiskRichness*. This comes to no surprise since the time complexity of *DiskRichness* is $O(n(1 + K/r))$ in theory. The decrease is more evident for the `peak` dataset than for `soils`. This is possibly again due to the fact that in `soils` there is a much smaller number of categories per tile for which the algorithm computes the partial envelopes, leading to performance which depends almost entirely on n even for small values of r.

For the last experiment, we ran our implementations using different values of parameter K, and fixed values for n and r. We ran each of our algorithms on a raster of 5,500 × 5,500 cells extracted from `soils` with fixed $r = 50$ and $K = 100m$ where m ranges from one to twenty-five. We also ran the algorithms on a raster of the same dimensions extracted from `peak`, with $r = 50$ and $K = 100m$ for integer $m \in [1, 32]$ (the number of category values observed in the two rasters is different; this number is 2,569 for the raster taken from `soils` and 3,259 for the raster taken from `peak`). Fig. 7 shows the results for this experiment. This figure does not include any benchmarks for the naive programs as each program took more than an hour to execute on a single input.

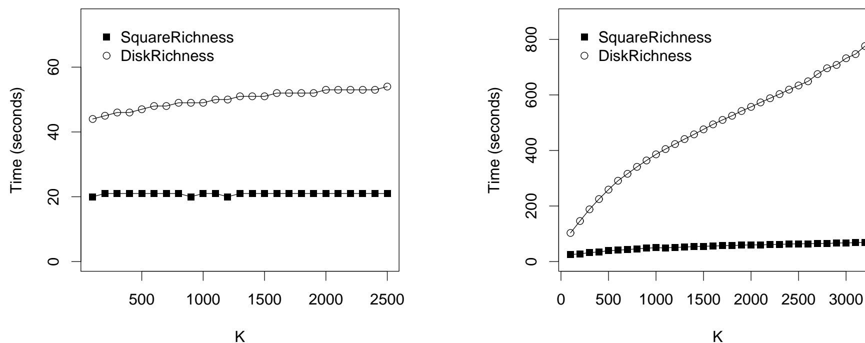

Figure 7: The running times of our implementations for different values of parameter K, on a raster of size $n = 5{,}500 \times 5{,}500$ and with $r = 50$. Left: the running times for a raster extracted from `soils` dataset. Right: the running times for a raster extracted from `peak` dataset.

We see that for the rasters produced from `peak` the performance of *DiskRichness* is affected by K significantly, although this algorithm still has very good performance. We see also a slight increase in the running time of *SquareRichness* for these datasets. For `soils`, the increase in K induces a small increase in the running time of *DiskRichness* and does not really affect *SquareRichness*.

We conclude that our algorithms for categorical richness are practically efficient and behave as expected by theory.

References

[1] F. Aurenhammer. Voronoi diagrams—A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

[2] J. M. Barea-Azcón, B. M. Benito, F. J. Olivares, H. Ruiz, J. Martín, A. L. García, and R. López. Distribution and conservation of the relict interaction between the butterfly Agriades zullichi and its larval foodplant (Androsace vitaliana nevadensis). *Biodiversity and Conservation*, 23(4):927–944, 2014.

[3] B. M. Benito, L. Cayuela, and F. S. Albuquerque. The impact of modelling choices in the predictive performance of richness maps derived from species-distribution models: Guidelines to build better diversity models. *Methods in Ecology and Evolution*, 4(4):327–335, 2013.

[4] J. L. Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[5] J.-D. Boissonnat, M. Sharir, B. Tagansky, and M. Yvinec. Voronoi diagrams in higher dimensions under certain polyhedral distance functions. *Discrete & Computational Geometry*, 19(4):485–519, 1998.

[6] F. S. de Albuquerque, B. Benito, P. Beier, M. J. Assunção-Albuquerque, and L. Cayuela. Supporting underrepresented forests in Mesoamerica. *Natureza & Conservação*, 2015.

[7] FAO, IIASA, ISRIC, and JRC. Harmonized world soil database (version 1.2), 2012.

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

[9] P. Gupta, R. Janardan, and M. Smid. Computational geometry: Generalized intersection searching. In D. P. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2005.

[10] J. J. Hayes and S. M. Robeson. Spatial variability of landscape pattern change following a ponderosa pine wildfire in northeastern New Mexico, USA. *Physical Geography*, 30(5):410–429, 2009.

[11] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Efficient colored orthogonal range counting. *SIAM Journal on Computing*, 38(3):982–1011, 2008.

[12] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, 2014.

[13] K. McGarigal, S. A. Cushman, and E. Ene. *FRAGSTATS v4: Spatial Pattern Analysis Program for Categorical and Continuous Maps*. University of Massachusetts, Amherst, 2012. Available at: http://www.umass.edu/landeco/research/fragstats/ fragstats.html.

[14] M. Neteler, M. H. Bowman, M. Landa, and M. Metz. GRASS GIS: A multi-purpose open source GIS". *Environmental Modelling & Software*, 31:124–130, 2012.

[15] Q. Shi and J. JaJa. Optimal and near-optimal algorithms for generalized intersection reporting on pointer machines. *Information Processing Letters*, 95(3):382–388, 2005.

[16] T. G. Smolinski, M. G. Milanova, and A. E. Hassanien. *Applications of Computational Intelligence in Biology: Current Trends and Open Problems*, volume 122 of *Studies in Computational Intelligence*. Springer-Verlag Berlin Heidelberg, 2008.

[17] U. S. G. Survey. The national map viewer and download platform. `http://nationalmap.gov/viewer.html`. Accessed: 2015-06-28.