Data Structures for Retrieval on Integer Grids

Martin David Katz
Dennis J. Volper

November 12, 1984

UCI ICS Technical Report 235

Abstract

A family of data structures is presented for retrieval of the sum of values of points within a half-plane or polygon, given that the points are on integer coordinates in the plane. Fredman has shown that the problem has a lower bound of $\Omega(N^{2/3})$ for intermixed updates and retrievals. Willard has shown an upper bound of $O(N^{2\log_6 4})$ for the case where the points are not restricted to integer coordinates.

We have developed families of related data structures for retrievals of half-planes or polygons. One of the data structures permits intermixed updates and half-plane retrievals in $O(N^{2/3}\log N)$ time, where N is the size of the grid.

We use a technique we call "Rotation" to permit a better match of a portion of the data structure to the particular problem. Rotations appear to be an effective method for trading-off storage redundancy against retrieval time for certain classes of problems.

Submitted to SIAM J. on Computing *as* Data Structures for Retrieval on Square Grids

1 Introduction

Suppose that one needs to determine the total of the values within a region in the plane. For example, one might need to determine the distribution of people in a region for traffic planning, demographic growth studies, or electoral redistricting, or one might represent a graph as a matrix in which a set element is associated with each matrix element; retrieval of the union of set elements in a region is also an application of the problem being studied here. Most of this paper focuses on those applications in which the points at which data resides are at integer coordinates in a plane.

Given an $N \times N$ grid of points on integer coordinates, associate with each point a value which is a member of a commutative semi-group. We give a family of data structures to support updating of the values at points and retrieval of the sum of the values associated with points within half-planes or polygons superimposed on the grid.

This family of data structures provides a trade-off between retrieval time and storage redundancy (which is related to both update time and space). We associate with each structure in this family a pair of numbers denoted $[\rho, t]$ (or $\mathbf{E}[\rho, t]$) representing the worst (or expected) case storage redundancy (ρ) and retrieval time (t).

Following [Fredman 80], our complexity measure is the number of semi-group operations required to perform a task, ignoring the time to index into the data structure. Since the update and retrieval operations are independent of the particular semi-group, one may think of the semi-group as the addition operator over the non-negative integers.

In the same paper, Fredman gives a method for deriving lower bounds for the sum of the storage redundancy and retrieval times for some geometric retrieval problems. The examples in the paper include half-plane and polygon retrievals. Fredman derives a lower bound complexity of $[\rho + t] = M^{4/3}$ for M intermixed operations on data structures with M points. Fredman's result implies a lower bound complexity of $[\rho + t] = N^{2/3}$ on the $N \times N$ grid.

Willard [Willard 82] presents a more general data structure for performing half-plane and polygon retrieval of points in the plane with no restriction on their position. Willard's structure has a retrieval time $= O(N^{2 \log_6 4})$ and preprocessing time $= O(N^4)$ for a universe containing N^2 points. Yao [Yao 83] extends Willard's results to multiple dimensions.

The $N \times N$ grid is analogous to a two dimensional array. Retrievals on the grid can be considered an intermediate problem between the two dimensional orthogonal range queries and the more general geometric retrievals of Willard and Yao. The data structures presented here permit intermixed half-plane retrievals and updates on the $N \times N$ grid with complexity $[\rho + t] = N^{2/3} \log N$.

1.1 General Approach

The new technique introduced in this paper is called a "Rotation." Many data structures (e.g. an array maintenance structure) have an orientation. Our approach is to build many

of these structures with different orientations, each covering the same data. We call each of these oriented structures a rotation.

Each rotation contains a substructure which consists of a structure which solves a query with a given orientation and other structures to extend the solution to other queries with closely related orientations.

1.2 Computational Model

Following Fredman, we formally define a range retrieval problem as a pair (K, Γ), where K is a set referred to as the key space, and Γ is a collection of subsets of K which cover K (Γ is the set of regions). The objective is to define a data structure representing a set $\mathcal{T}$ of records, where each record r has a uniquely identifying key in K (denoted key(r)), and an associated value (denoted value(r)), which is assumed to lie in a commutative semi-group S (a commutative semi-group is a set of elements closed under a commutative and associative operator, $\oplus$). The purpose of the data structure is to efficiently implement of the following tasks:

Retrieve(R): Compute the sum (using $\oplus$) of the values associated with all records in $\mathcal{T}$ whose keys lie in the region R. ($R \in \Gamma$)

Insert(k, x): Add a record $r \in \mathcal{T}$ such that key(r) = k and value(r) = x.

Delete(k): Remove a record r from $\mathcal{T}$ where key(r) = k.

Update(k, x): Change a record $r \in \mathcal{T}$ such that key(r) = k and new-value(r) = old-value(r) $\oplus$ x. ($k \in K$, and $x \in S$)

The data structure $\mathcal{D}$ is an ordered collection of variables. Each variable's *value* field holds a value in S, and each variable's *key* field holds a value in K^* (a representation of the keys which are covered by this variable).

Retrieval time t is defined as the number of operations on values in S which are necessary to perform a Retrieval using $\mathcal{D}$. We will limit the operations on S to one application each of assignment and $\oplus$ per variable of $\mathcal{D}$ for each Retrieval. Therefore, t is also the minimum number of variables in $\mathcal{D}$ which must be accessed to perform a Retrieval. If not otherwise stated, complexities will be assumed to be worst case.

Storage redundancy ρ is defined as the number of variables in $\mathcal{D}$ whose values depend on any single record in $\mathcal{T}$.

For this problem, the records in $\mathcal{T}$ correspond to the points in the integer grid in the plane. $K = \{(i,j) \mid (i,j) \in (\{1...N\}, \{1...N\})\}$. That is, the key of a record is the coordinates of the point. If a record r has no value, value(r) = 0 where 0 is the identity value for the semi-group.

2 The Half-Plane Problem

In the half-plane problem, we define $\Gamma_{hp} = \{(i,j) \mid ai + bj \geq 1\}$ for $(i,j) \in (\{1...N\}, \{1...N\})$ and $(a,b) \neq (0,0)$ and $a,b \in \Re$. That is, each region to be retrieved $g \in \Gamma_{hp}$ is a collection of points in the intersection of a half-plane and the grid.

2.1 The Half-Plane Data Structure

The components of the data structure for the half-plane problem take two geometric forms: rotations and slices. We will nest these structures in an appropriate way to support efficient retrieval.

Our data structure is a collection of πN^{α} rotations evenly spaced over one half circle (π radians). A rotation consists of a disjoint union of slices exactly covering the grid, as shown in figure 1. Each rotation contains $2N^{\beta}$ slices which are up to $\sqrt{2}N$ long and up to $\frac{1}{\sqrt{2}}N^{1-\beta}$ wide.

Each slice is a collection of disjoint rectangles placed side by side which cover the slice (figure 1 shows the rectangles for one slice). There are $2N^{\delta}$ rectangles per slice. Thus, each rectangle is up to $\frac{1}{\sqrt{2}}N^{1-\beta}$ wide and up to $\frac{1}{\sqrt{2}}N^{1-\delta}$ long.

We will develop a description of the legal values for α, β, and δ in the analysis section of this paper. As we shall see, these restrictions will be selected to limit the area of each rectangle to less than 1 while limiting the number of rectangles crossed.

2.2 Performing a Half-Plane Retrieval

The first step in performing a retrieval is selection of the rotation closest to the angle of the half-plane edge. Each slice in this rotation is retrieved in two parts: the interior of the slice and the the frontier of the slice. The interior of the slice is the collection of rectangles in the slice which are entirely within the half-plane; it is retrieved by an array retrieval on the collection of rectangles. The frontier of the slice is the collection of rectangles in the slice which are crossed by the half-plane edge.

In order to efficiently perform an array retrieval, the rectangles within a slice must be arranged in an appropriate data structure (see [Burkhard and Fredman 81], [Fredman 82] for a discussion of the array maintenance problem). We will assume that a complete binary tree is used for maintenance of the array of rectangles in each slice. In this structure, the leaves of the tree are the rectangles, and the value field of each internal node contains the sum of the value fields of its two children. We will call the tree a "slice retrieval tree." The complexity of retrieval for the portion of each slice which is entirely within the half-plane is $[\rho,t]=[O(\log k), O(\log k)]$ where k is the number of rectangles in a slice. In our data structure, $\log k = O(\log N)$, so the complexity of retrieving the interior of a slice is $[\rho,t]=[O(\log N), O(\log N)]$.

We will use the property that if the points in a rectangle are collinear, then an array maintenance structure can also be used to represent the points within a rectangle. The

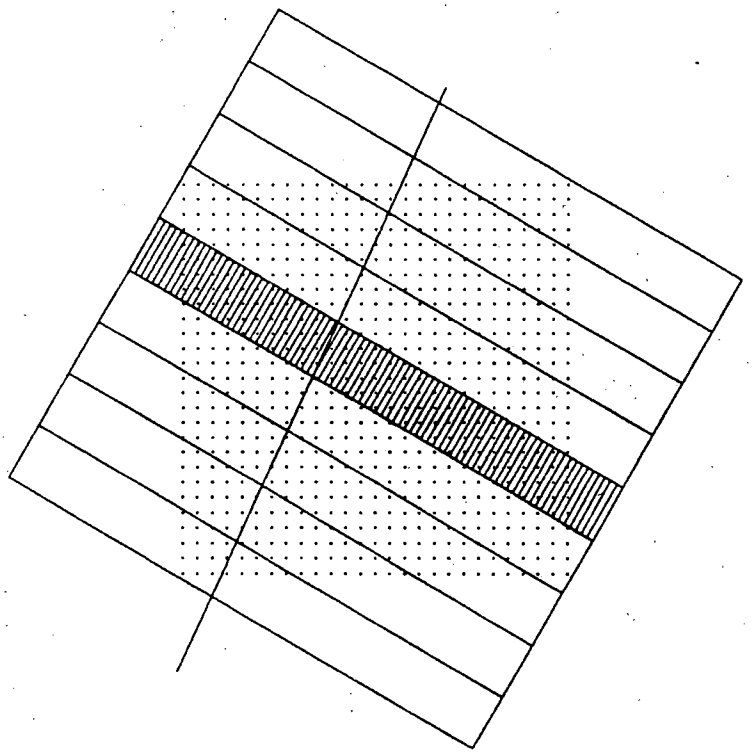


Figure 1: One rotation in the half-plane data structure. For clarity, rectangles are shown for only one slice. The line extending across the structure is a sample half-plane edge whose orientation is covered by the rotation.

rectangles in the frontier of the slice can thus be retrieved by treating the points in each rectangle as an array retrieval. When an extended problem is presented in which the points are not collinear, then the frontier is retrieved by retrieving the points individually.

2.3 Analysis of the Storage Redundancy

The storage redundancy ρ_k of record $\tau_k \in \mathcal{T}$ is the number of variables $d \in \mathcal{D}$ whose values are affected by τ_k. The worst case storage redundancy ρ is $\max\{\rho_k \mid \tau_k \in \mathcal{T}\}$. It is easy to see that if all the dependencies are additive, the number of semi-group operations required to update the value of a record equals the storage redundancy.

Theorem 2.1 *The storage redundancy of the data structure defined above is*

$$O(N^a \log N).$$

Proof: When changing the value of any record (point), we must change the value of all variables which cover that record. Thus, in each rotation structure, we must change the value in all the variables in the slice retrieval tree which are in the path to the root from the leaf associated with the rectangle which covers that record. Likewise, we must change the value of all variables in the rectangle retrieval tree in the path to the root from the variable which exactly covers the record.

The number of rotations is πN^{α}; within each rotation, exactly one slice and one rectangle contains that record. The slice retrieval tree is a complete binary tree with $2N^{\delta}$ leaves and height $O(\log 2N^{\delta})$. The rectangle retrieval tree is a complete binary tree with at most $N^{1-\beta}$ leaves and height $O(\log N^{1-\beta})$. The storage redundancy is thus $O(\pi N^{\alpha}(\log 2N^{\delta} + \log N^{1-\beta}))$ or $O(N^{\alpha} \log N)$. □

2.4 Analysis of the Retrieval Time

Lemma 2.2 *If the area of a rectangle is less than* 1, *then all points in the intersection of the integer grid and the rectangle must be collinear.*

Proof: By contradiction: Suppose that there are three non-collinear points in the rectangle. Three non-collinear points define a triangle. The area of the smallest triangle whose vertices are all integers is $\frac{1}{2}$. The area of the largest triangle which can be inscribed in a rectangle is half the area of the rectangle. This constitutes a contradiction since it was given that area of the rectangle is less than 1. □

Corollary 2.2.1 *If $\beta + \delta \geq 2$, where β and δ are as in the data structure described above, then all points on an integer grid within a rectangle are collinear.*

Proof: A rectangle measures no more than $\frac{1}{\sqrt{2}}N^{1-\beta}$ by $\frac{1}{\sqrt{2}}N^{1-\delta}$. Therefore, the area of the rectangle is $\frac{1}{2}N^{2-(\beta+\delta)}$ which is less than 1 if $\beta + \delta \geq 2$. □

Lemma 2.3 *Given a half-plane and the most closely aligned rotation of the data structure, the maximum number of rectangles crossed by the half-plane edge within any slice of that rotation is $1 + N^{\delta-\alpha-\beta}$.*

Proof: The slope of the half-plane edge relative to the slices is no more than $N^{-\alpha}$ from perpendicular, and the width of the slice is limited to $\frac{1}{\sqrt{2}}N^{1-\beta}$. Therefore, the section of the half-plane edge which intersects the slice can be contained in a rectangle, R, extending $\frac{1}{\sqrt{2}}N^{1-(\alpha+\beta)}$ along the slice. The width of each rectangle in the data structure is $\frac{1}{\sqrt{2}}N^{1-\delta}$. The maximum number of rectangles which can be crossed by R is one more than the width of R divided by the width of a rectangle (round up), which yields the desired number. □

Theorem 2.4 *Given $2-\beta \leq \delta \leq \alpha+\beta$, the worse case time to perform a half-plane retrieval using the data structure described above is $O(N^{\beta} \log N)$.*

Proof: The retrieval algorithm requires at most $2N^{\beta}$ slice retrievals. Each slice retrieval requires a traversal of a slice retrieval tree (performing one semi-group sum and one semi-group assignment per level of the tree) and a rectangle retrieval for each rectangle crossed by the half-plane edge within the slice. Since the slice retrieval tree is a complete binary tree with $2N^{\delta}$ leaves, we can retrieve the section of each slice which is entirely within the half-plane in time $O(\log N)$.

Corollary 2.2.1 shows that when $2 \leq \beta + \delta$ all points in each rectangle are collinear. Thus, the intersection between the half-plane edge and the line defined by the points can be determined and each rectangle retrieval can be performed by traversal of the rectangle retrieval tree. The rectangle retrieval tree is a balanced binary tree with $2N^{(1-\beta)}$ leaves. Therefore, a rectangle retrieval can be performed in time $O(\log N)$. Lemma 2.3 shows that for $\delta \leq \alpha + \beta$ only two rectangle retrievals are required per slice.

The time for a half-plane retrieval is thus
$2N^{\beta} \times [O(\log N) + 2O(\log N)]$ or $O(N^{\beta} \log N)$. □

For $\delta = \alpha + \beta$ the following corollaries hold:

Corollary 2.4.1 *Given $2 \leq \alpha + 2\beta$, the worse case time to perform a half-plane retrieval using the data structure described above is $O(N^{\beta} \log N)$.* □

Corollary 2.4.2 *By theorem 2.1 and corollary 2.4.1 we have shown that the data structure is of complexity $[\rho, t]$=$[O(N^{2-2\beta} \log N), O(N^{\beta} \log N)]$, for $\alpha = 2 - 2\beta$, $0 \leq \beta \leq 1$. Indeed, if we let $\beta = \frac{2}{3}$ then $[\rho, t]$=$[O(N^{2/3} \log N), O(N^{2/3} \log N)]$.* □

2.5 Removing the Integer Coordinates Constraint

The above discussion requires that the points be placed at integer coordinates on a grid. In this section we remove this restriction and obtain an expected complexity for the more general problem.

Assume a universe defined as N^{p} points uniformly distributed over an $N \times N$ region. Lemma 2.2.1 assumes that the points are on an integer grid, so it no longer holds. However, the analysis of the storage redundancy (theorem 2.1) and the time to retrieve the interior of each slice remain the same. Therefore, we wish to analyze the expected time to retrieve the frontier of each slice.

If we examine each point in a rectangle individually, the time to retrieve a rectangle is proportional to the number of points in the rectangle. By construction, the area of a rectangle is at most $O(\frac{1}{2}N^{2-(\beta+\delta)})$. We limit the number of rectangle retrievals per slice to a constant by defining $\delta = \alpha + \beta$ (see Lemma 2.3).

The expected complexity of a rectangle retrieval is the density of points times the area of the rectangle:

$$\frac{N^p}{N \times N} \frac{1}{2} N^{2-(\beta+\delta)} = \frac{1}{2} N^{p-(\alpha+2\beta)}$$

This modifies the result given in theorems 2.1 and 2.4 to yield a complexity of

$$\mathrm{E}[\rho, t] = \mathrm{E}[O(N^{\alpha} \log N^{\alpha+\beta}), O(N^{\beta} \log N^{\alpha+\beta}) + O(N^{p-(\alpha+\beta)})].$$

If we let $\alpha = \beta = (p - \alpha - \beta)$ this yields

$$\mathrm{E}[\rho, t] = \mathrm{E}[O(N^{p/3} \log N), O(N^{p/3} \log N)].$$

Note that this is dependent on the number of points, not N.

When $\alpha = \beta = \frac{2}{3}$ and $p = 2$, then, as expected,

$$\mathrm{E}[\rho, t] = \mathrm{E}[O(N^{2/3} \log N), O(N^{2/3} \log N)].$$

2.6 The Dynamic Data Structure

Thus far we only use Update and Retrieve operations, and assume that all records are present or initialized to 0. This is the simplest case of a dynamic data structure, since the values being retrieved can change, but the structure in which they reside can not. In this section, we will examine two levels at which a data structure can be said to be dynamic:

1. N is known in advance
2. N is unknown.

In both of these versions, we start with an empty grid, and enter records corresponding to points by Inserts. We may also Delete a record or we may Update the value attached to a record. We define modifications to the half-plane data structure which permit the data structure to represent these dynamic grids with only a factor of $O(\log N)$ penalty in complexity.

N Known in Advance

We approach the first form of the problem by abstractly generating a version of the data structure in which all values are initially 0. Portions of the data structure are actually generated only as needed by operations.

The first insertion generates all of the rotations, but only those parts of each rotation necessary to represent the first point (i.e. only one slice and only one rectangle) are filled in. The slices can be stored in a binary tree with no increase in the net complexity (the $O(\log N)$ per rotation storage redundancy of the tree of slices is covered by the $O(\log N)$ storage redundancy of the slice structure). This permits the slices to be created only as needed. The storage of the slices in a tree does not increase the retrieval time since the slice tree can be traversed in linear time (in the number of slices).

Subsequent insertions extend the data structure within each rotation to support the new point. Retrievals are as before except that the trees are not complete.

Each deletion removes a single record from $\mathcal{T}$. Following Fredman, we will assume that semi-group variables can only have values assigned once. Thus, when deleting a record, we discard the variable corresponding to that record, and replace all $O(\log N)$ variables in the retrieval structure from the variable representing the point to the root in the data structures within both the slice and the rectangle. Each deletion thus requires $O(\log N)$ semi-group operations per rotation. Updates may be performed as a deletion followed by an insertion.

The cost is thus $[\rho, t]$=$[O(N^{2-2\beta} \log N), O(N^{\beta} \log N)]$ per operation. Thus, the simplest form of the dynamic structure costs no more per operation than the static problem.

N Not Known in Advance

In the second form of the problem, we are trying to solve the same problem, but we

don't know how large N is. Thus, we initially guess that N is small, and adjust the data structure as updates are made.

We can convert this problem into a decomposable problem by noting that we can assume N to be a power of 2 with only a constant penalty factor. We can then use any of the methods described in [Bentley 79], [Overmars and van Leeuwen 81], or [Saxe and Bentley 79] to convert the static data structure into the dynamic one.

We will take the following approach: Guess $N = 2$. Use the data structure described for the first form of the dynamic case except for insertions of points outside the domain of the data structure. When a point outside the domain of the data structure is inserted, recreate the data structure with a new N whose value is the smallest power of two greater than the distance between the two points furthest apart, and centered on the points which exist.

Since we double the size of the data structure each time, the maximum number of times that the structure must be rebuilt is $\log N$. Thus, the complexity of the data structure is $[\rho, t]$=$[O(N^{2-2\beta}\log^2 N), O(N^{\beta}\log N)]$ per operation.

Not an Integer Grid – Average Case

The modifications to the half-plane data structure in the previous two sections can applied directly to the data structure in section 2.5 which permits retrievals when the points are not on an integer grid. This creates a data structure which has an expected complexity only a factor of $O(\log N^p)$ worse than the static case, even if N and p are not known in advance.

3 Retrieval of Triangular Regions

A problem closely related to half-plane retrieval is triangle retrieval. In triangle retrieval, the same $N \times N$ square grid is defined, but the region to be retrieved consists of the points within a triangle superimposed on the grid. The complexity bounds presented for triangle retrieval by Fredman and Willard are the same for this problem as for half-plane retrieval. We present a family of data structures which permit retrieval time to be traded-off against storage redundancy.

3.1 The Triangle Data Structure

We divide the grid into squares whose sides are parallel to the grid axis. The squares form N^λ rows of N^λ squares each ($0 \leq \lambda \leq 1$). The retrieval algorithm for a square depends on its relationship to the edges of the triangle.

An array retrieval structure is associated with each row of squares to permit the retrieval of a contiguous subset of squares in the row. This structure is used for those squares which are completely contained in the triangle.

Within each square, there are three data structures. The first structure is exactly the same as that used for half-plane retrievals (of course, the β parameter may have a different value). This structure is used when one of more edges of the triangle pass through the square.

The second structure is also collection of rotations. Each rotation corresponds to one of the half-plane rotations. Each of these is a collection of $2N^{2-2\lambda}$ rectangles, called "slivers." Note that the width of each sliver is no more than $\frac{\sqrt{2}}{2}N^{\lambda-1}$ and the area is less than 1. Each sliver contains an array maintenance structure to store the points in the sliver. This structure is used, in addition to the half-plane structure, when there are two edges of the triangle in the square.

The third structure organizes each row of the square into an array retrieval structure. This permits retrieval of the square row by row. Note, the sliver structure could also be used for this, but we have chosen to use row by row retrieval for simplicity.

3.2 Performing a Triangle Retrieval

We superimpose the triangle on the lattice of squares. We classify each square according to its relationship to the edges of the triangle. The following cases are possible:

1. The square is entirely inside the triangle.
2. The square is crossed by exactly one edge of the triangle.
3. The square is crossed by two edges of the triangle. This has the following subcases:
   (a) There are exactly two edges in the square such that the relative slope S of the edges satisfies $0 \leq |S| \leq N^{(1-\lambda)(2\beta-2)}$

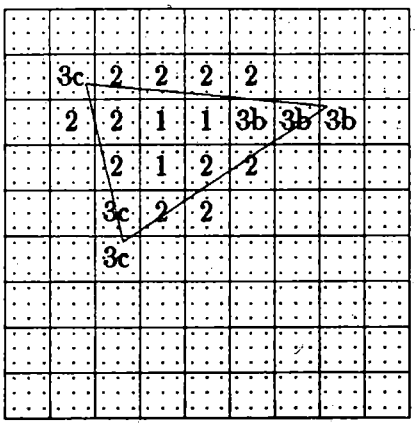


Figure 2: A sample triangle and the corresponding classified squares.

(b) There are exactly two edges in the square such that $N^{(1-\lambda)(2\beta-2)} < |S| < 1$

(c) There are exactly two edges in the square such that $1 \leq |S|$

4. There are three edges in the square.

We determine which rows of squares overlap the triangular region, and which squares in those rows contain portions of the edges of the triangle. The squares which are entirely contained within the triangle (case 1) are retrieved by array retrieval on the rows of squares.

Each square which has only one edge passing through it (case 2) is retrieved by half-plane retrieval.

Each square with two almost parallel edges passing through it (case 3(a)) is retrieved by a modified half-plane retrieval. In this case, an edge is chosen, and the strip is retrieved using the closest rotation. The slices are retrieved as usual (except that the region is bounded on two sides). The rectangles are retrieved as usual except that in each slice a frontier of rectangles must be retrieved along both edges (instead of one). This is called a strip retrieval.

When two edges pass through the square with moderate relative slope (case 3(b)), we perform the retrieval in two steps. The first step is to retrieve as much of the region as possible as a strip retrieval. The second step is to perform a retrieval on each of the slivers which cross the wedge shaped remaining portion. For each sliver, we use the array retrieval structure to retrieve the portion of the sliver which is in the region.

Finally, the remaining cases (3(c) and 4) are retrieved using the structure on the rows of points. This is acceptable because the number of occurrences of each of these cases is small.

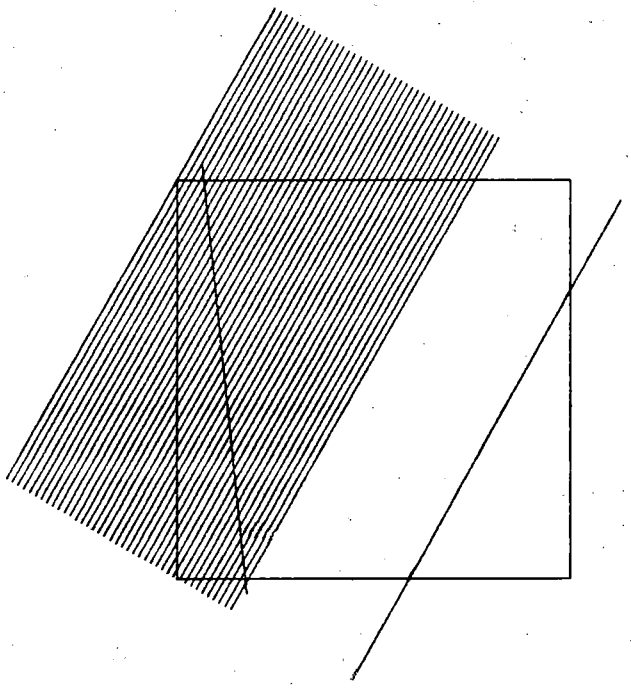


Figure 3: Sample case 3(b) square for triangle problem.

3.3 Analysis of Storage Redundancy

Theorem 3.1 *The storage redundancy ρ of the triangle retrieval data structure is*

$$\rho = O(N^{(1-\lambda)(2-2\beta)} \log N).$$

Proof: Each point is covered by the data structure for rows of squares and the data structures within the squares.

The storage redundancy due to the array retrieval structure on the rows of squares is $\rho = O(\log N^{\lambda})$.

The storage redundancy within each square for the half-plane structure is the same as in half-plane retrievals. By Theorem 2.1, this is $\rho = O(M^{2-2\beta} \log M)$, where M is the size of the half-plane. Since $M = N^{1-\lambda}$, $\rho = O(N^{(1-\lambda)(2-2\beta)} \log N)$.

The sliver structure also has $N^{(1-\lambda)(2-2\beta)}$ rotations. Each point can be in only one sliver. Since each sliver is an array retrieval structure, the storage redundancy due to this structure is $\rho = O(N^{(1-\lambda)(2-2\beta)} \log N)$.

Finally, the storage redundancy for an array retrieval structure on the rows of points in a square is $O(\log N^{1-\lambda})$ □

3.4 Analysis of the Retrieval Time

Theorem 3.2 *The retrieval time of a triangle retrieval is the maximum of*

$$O(N^{2-2\lambda}\log N), \text{ and } O(N^{(1-\lambda)\beta+\lambda}\log N).$$

Proof: We examine each case defined in section 3.2 individually:

1. The number of rows of squares entirely inside the triangle is at most N^{λ}, and each row requires $t = O(\log N)$. The retrieval time for all squares in case 1 is $t = O(N^{\lambda}\log N)$.

2. When the square is crossed by one edge of the triangle, by corollary 2.4.1, the retrieval time of the half-plane structure in each square is $t = O(N^{(1-\lambda)\beta}\log N)$. Because the perimeter of the portion of the triangle within the $N \times N$ grid is $O(N)$, and the size of each square is $N^{1-\lambda}$, there are $O(N^{\lambda})$ squares in this case yielding a total retrieval time of $t = O(N^{(1-\lambda)\beta+\lambda}\log N)$.

3. Because of the length of the perimeter, there can be no more than $O(N^{\lambda})$ squares which are crossed by two edges of the triangle. Each of the subcases is treated as follows:

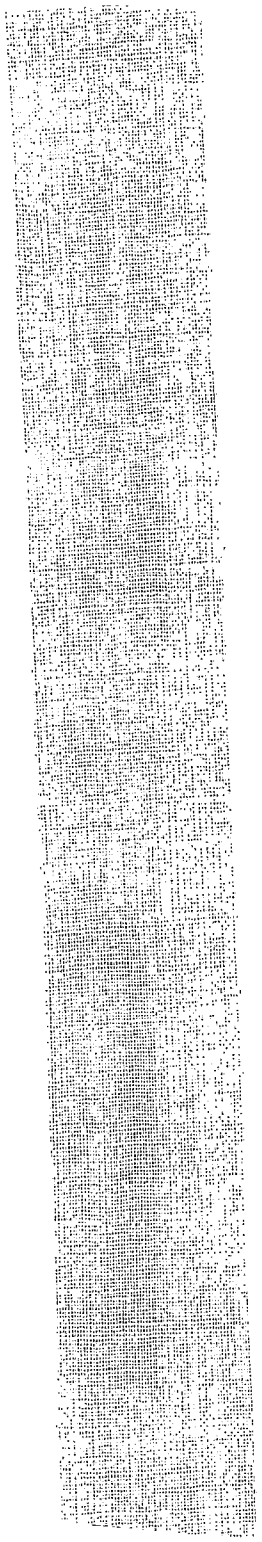
   (a) For this case, the two edges are at a narrow enough angle that the half-plane structure can be used. Thus the retrieval time for all occurrences of this case is $t = O(N^{(1-\lambda)\beta+\lambda}\log N)$.

   (b) There are two edges of the triangle in the square. This implies that the edges are no more than $\sqrt{2}N^{1-\lambda}$ apart. Also, the relative slope of the lines is at least $N^{(1-\lambda)(2\beta-2)}$. From geometry, we can determine that there are no more than $O(N^{(1-\lambda)(2-2\beta)})$ squares in this case per vertex.

   The first step, the strip retrieval, requires time $O(N^{(1-\lambda)\beta}\log N)$ per square (corollary 2.4.1). Therefore, the retrieval time for the first step for all of these retrievals is $t = O(N^{(1-\lambda)(2-\beta)}\log N)$.

   The second part is retrieval of the remainder by slivers. Note that the sum of the widths of the wedge shaped regions retrieved is at most $O(2\sqrt{2}N^{1-\lambda})$ per vertex of the triangle. This is because the total width of the regions is the divergence of the triangle edges. The total divergence of the edges in case 3 is limited to this value. Since each sliver has width $\frac{1}{2}N^{\lambda-1}$ the number of slivers involved is $O(N^{2-2\lambda})$.

   The area of each sliver is less than 1. By Lemma 2.2, all points within a sliver must be collinear. Each sliver has a sliver retrieval tree associated with it, so the retrieve time per sliver is $O(\log N)$. Therefore, the total retrieval time for this case is $t = O(N^{2-2\lambda}\log N)$.



(c) The total number of squares in this case in any triangle retrieval is at most 6. Using the row structure within each of these squares gives a total retrieval time for this case of $t = O(N^{1-\lambda} \log N)$.

4. Only one square in this case can exist. Therefore, using the row structure within each of these squares gives a total retrieval time for this case of $t = O(N^{1-\lambda} \log N)$.

Add the complexities. □

Corollary 3.2.1 *For $\lambda = \frac{2}{3}$ and $\beta = 0$,*

$$[\rho, t] = [O(N^{2/3} \log N), O(N^{2/3} \log N)].$$

□

This is close to the lower bound of $[\rho + t]=[\Omega(N^{2/3})]$ which Fredman shows for the polygon retrieval problem.

Corollary 3.2.2 *On the square grid, the retrieval problem for convex polygons with v vertices can be performed with complexity*

$$\begin{aligned}[\rho, t] = [&O(N^{(1-\lambda)(2-2\beta)} \log N), \\ &O(\max(vN^{2-2\lambda} \log N, \\ &\quad vN^{(1-\lambda)\beta+\lambda} \log N).]\end{aligned}$$

Proof: Any polygon with v vertices and without internal voids can expressed as the disjoint union of $O(v)$ triangles. The triangles can be retrieved as above. Modify the retrieval algorithm such that points on the boundary between triangles are only retrieved once. This is done by careful choice of whether a line is inside or outside the triangle (e.g. each internal edge belongs to the triangle which is furthest clockwise or furthest from the center). To modify the algorithm, before a triangle is to be retrieved, mark each edge of the triangle as either contained in the region or outside the region. When performing the retrieval, include points exactly on the edge only if the edge is marked as inside the triangle. This doesn't entail any changes in the algorithms or data structures as described above. □

4 Retrieval of Circular Disks

Given the $N \times N$ grid defined above. We retrieve the sum of the values associated with those grid points which fall within a circle. We define $\Gamma_{\text{circle}} = \{(i,j) \mid (i-a)^2+(j-b)^2 \leq r^2\}$ for $(i,j) \in (\{1...N\},\{1...N\})$ and $a,b,r \in \Re$. We observe that this is the same as the set of points which fall within the convex hull of the points within the circle. Such a convex hull with its vertices on integer coordinates is a polygon with $O(N^{2/3})$ edges (see appendix). We refer to the polygon as C. Since we are only considering semi-group operations, we ignore the time required to find the convex hull associated with the circle.

4.1 The Circle Data Structure

The data structure is almost identical to that for triangle retrieval, but the structure within the slices is a two dimensional range query structure instead of a one dimensional structure. The structure is based on the orthogonal range query structure [Lueker and Willard 78].

For the circle data structure, we divide the grid into squares whose sides are parallel to the grid axis. The squares form N^{λ} rows of N^{λ} squares each. The retrieval algorithm for a square depends on its relationship to the edges of C.

For retrieving squares entirely within C there is an array retrieval structure associated with each row permitting the retrieval of a contiguous subset of squares in that row.

Within each square, there are two data structures. The first structure is an array retrieval structure associated with each row of points in the square. This permits row by row retrieval of the square.

The second structure is used when one or more edges of C pass through the square. The structure is a modified half-plane retrieval structure. Thus, it is a collection of $N^{(1-\lambda)(2-2\beta)}$ rotations, each containing $N^{\beta(1-\lambda)}$ slices.

In the half-plane data structure, each slice has a slice retrieval tree associated with it. In the standard half-plane structure, each node of the slice retrieval tree has a variable associated with it containing the sum of the values covered by the node. In the half-plane structure modified for circle retrievals, each node in the slice retrieval tree has a two dimensional orthogonal range query structure [Lueker and Willard 78] associated with it instead of just a variable.

The axes of the range query structure are parallel to the grid axes and only points falling within the slice are included in the structure. This permits us to Retrieve those points within the slice which are also contained in a trapezoid with at most one edge not parallel to the grid axes (triangles and rectangles are trivial trapezoids). We will use the version of orthogonal range query structure which has storage redundancy and retrieval time equal to $O(\log^2 N)$.

For some parts of the circle retrieval, we will use the slices for standard half-plane retrievals, and sometimes the interior of the slices will be retrieved using the two dimensional

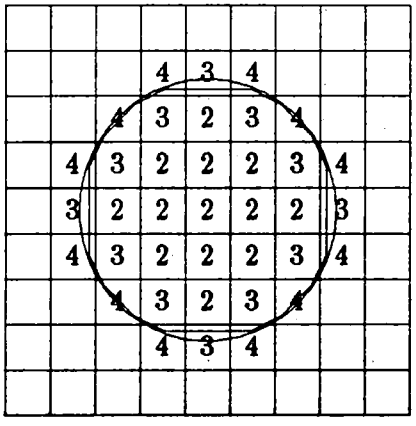


Figure 4: A sample circle and the corresponding classified squares. The data structure for circle retrieval requires that the grid be divided into squares. After the convex hull of the circle is superimposed, the squares are categorized based on their relationship to the segments of the convex hull.

structure. In either case, the rectangles within the slices are treated as in the standard half-plane problem.

4.2 Performing a Circle Retrieval

We have the following cases:

1. The diameter of C is less than $4N^{1-\lambda}$
2. The square is entirely inside C.
3. The square is crossed by exactly one edge of C.
4. The square is crossed by two or more edges of C which are at an obtuse angle.

If C is very small (case 1), then we retrieve each square row by row. That is, we use the array retrieval structure on the rows of points to retrieve the entire square.

Otherwise, we determine which rows of squares overlap the region, and which squares in those rows contain portions of the edges of C. The squares which are entirely contained within the region (case 2) are retrieved by array retrieval on the rows of squares.

The squares which have only one edge passing through them (case 3) are retrieved by standard half-plane retrieval.

If there is more than one edge in the square (case 4), we divide the region into subregions which have at most one edge which is not parallel to the grid axes (each subregion can be retrieved using the modified half-plane structure). The number of subregions is no more than twice the number of edges passing through square.

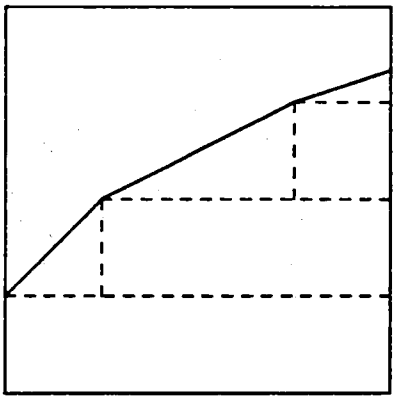


Figure 5: A square from the circle problem, showing the rectangles and triangles which form its subregions.

The subregions are formed by drawing a line from each vertex of C within the square and from each point at which an edge passes through the side of the square horizontally within C until they meet a side of the square or and edge of C. Then, a line is drawn from each vertex of C within the square and from each point at which an edge passes through the side of the square horizontally within C until they meet a side of the square, an edge of C or one of the horizontal lines. This results in a collection of right triangles and rectangles. The number of right triangles is no more than the number of edges, and the number of rectangles is also no more than the number of edges.

The triangular subregions are retrieved by performing a modified slice retrieval for each slice which overlaps the triangle. Each slice is retrieved in two parts: the interior and the frontier. The interior of each slice is retrieved by traversing the rectangle retrieval tree to determine which nodes (representing collections of rectangles — no more than two per level of the tree) are to be retrieved. A node is to be included if it would be included in a half-plane retrieval using the edge of the triangle which is not parallel to the grid axes as the half-plane edge. Each node is retrieved using the two dimensional orthogonal range query structure. Rectangular subregions are retrieved in the same manner as the triangular subregions, using the rotation which has a rotation of 0 radians.

4.3 Analysis of Storage Redundancy

Theorem 4.1 *The storage redundancy ρ of the circle retrieval data structure is*

$$\rho = O(N^{(1-\lambda)(2-2\beta)} \log^3 N).$$

Proof: Each point is covered by the data structure for rows of squares and the data structures within the squares.

The component of the storage redundancy due to the array retrieval structure on the squares is $\rho = O(\log N^{\lambda})$.

The component of the storage redundancy per square for the modified half-plane structure is the number of rotations ($O(N^{(1-\lambda)(2-2\beta)})$) times the redundancy per rotation. The redundancy per rotation is the redundancy of the slice structure, $O(\log N)$ (the height of the tree of rectangles) times the redundancy of the rectangle structure, $O(\log^2 N)$ (the redundancy of the two dimensional orthogonal range query structure). Since each point can be in only one square, slice, and rectangle, the storage redundancy of the modified half-plane structure is $\rho = O(N^{(1-\lambda)(2-2\beta)} \log^3 N)$.

Finally, the storage redundancy for an array retrieval structure on the rows of points in a square is $O(\log N)$

Summing these results yields the hypothesized storage redundancy. □

4.4 Analysis of Retrieval Time

Theorem 4.2 *The retrieval time of a circle retrieval on a square grid is*

$$t = O(\max(N^{(1-\lambda)\beta+2/3} \log^3 N, \\ N^{(1-\lambda)\beta+\lambda} \log N, \\ N^{\lambda} \log N, \\ N^{1-\lambda} \log N)$$

Proof: We discuss each of the cases individually below:

1. In this case, C is small enough, so only a constant number of squares are involved). Retrieve the square by retrieving each row of points in the square. There are at most $O(N^{1-\lambda})$ such rows, requiring $O(\log N)$ time each. Retrieval time is $t = O(N^{1-\lambda} \log N)$.

2. For squares completely within C, retrieve each row of squares by array retrieval. There are at most N^{λ} rows, each taking $O(\log N)$ time. Retrieval time is $t = O(N^{\lambda} \log N)$.

3. If exactly one edge crosses the square, then the square is retrieved by modified half-plane retrieval. There are at most $O(N^{\lambda})$ such squares. Each square is $O(N^{1-\lambda})$ on a side, therefore each square has a retrieval time of $O(N^{(1-\lambda)\beta} \log^3 N)$. The total retrieval time for this case is $t = O(N^{(1-\lambda)\beta+\lambda} \log^3 N)$.

4. The number of occurrences of this case is limited to twice the number of vertices. The number of vertices in the convex hull of the points within a circle of radius R whose vertices are at integer coordinates is $O(R^{2/3})$ (see appendix). Furthermore, the number of vertices within a square N on a side is $O(N^{2/3})$. Therefore, the number of occurrences of this case is $O(N^{2/3})$.

   Each of the subregions (the constructed triangles and rectangles) can be retrieved by modified half-plane retrieval in $t = O(N^{(1-\lambda)\beta} \log^3 N)$. There are no more than two subregions per vertex, and there are $O(N^{2/3})$ vertices to retrieve per circle. Therefore, the total retrieval time for this case is: $t = O(N^{(1-\lambda)\beta+2/3} \log^3 N)$.

The theorem is obtained by adding the above. □

Corollary 4.2.1 *Given $\beta = 0, \lambda = \frac{2}{3}$, the complexity of the data structure is*

$$[\rho, t] = [O(N^{2/3}\log^3 N), O(N^{2/3}\log^3 N)].$$

□

4.5 General Polygon Retrieval

We define the concept of a polygon's concavity factor φ. For a polygon P, $\varphi(P)$ is one plus the number of voids plus the number of discontinuities, plus $\frac{1}{2\pi}$ times the sum of the internal angles of the concave vertices (round up).

Each acute angle cause a turning (external angle minus π) of at least $\frac{\pi}{2}$. The total turning angle for a polygonal boundary (no voids or discontinuities) is 2π plus the sum of the internal angles of the concave vertices. Therefore, by adding the number of vertices and the angles, the total number of acute angles in P can be no more than 4φ.

A boundary with length cN within the $N \times N$ boundaries of the square grid must have a turning angle of at least $c\frac{\pi}{2}$. If P is not a simple polygon (i.e. it has discontinuities or voids) we deal with each boundary independently. Therefore, by adding the perimeters of all boundaries of P, the perimeter of the portion of P within the square grid can be no more than $4N\varphi$.

By drawing appropriate lines, each boundary of P can be divided into convex polygons. If such a convex polygon has more than $O(P^{2/3})$ vertices (P is the perimeter of the convex polygon), then a convex hull for the points within the convex polygon can be substituted for the convex polygon (the convex hull has no more than $O(P^{2/3})$ vertices). Therefore, a piecewise convex hull can be constructed for any polygon such that the number of vertices within the square grid is $O(\varphi N^{2/3})$.

Corollary 4.2.2 *We can retrieve the sum of the values associated with the grid points within any polygon with complexity*

$$\begin{aligned}[\rho, t] = [&O(N^{(1-\lambda)(2-2\beta)}\log^3 N),\\ &O(\varphi \max(N^{(1-\lambda)\beta+\lambda},\\ &\qquad N^{(1-\lambda)\beta+2/3},\\ &\qquad N^{2-2\lambda}\log^3 N)].\end{aligned}$$

Furthermore, if $\beta = 0$ and $\lambda = \frac{2}{3}$ then $[\rho, t]$=$[O(N^{2/3}\log^3 N), O(\varphi N^{2/3}\log^3 N)]$.

Proof: Use the circle data structure, but also include the "sliver" structure from the triangle problem in each square so that acute angles can be retrieved.

Eliminate voids and connect any disconnected regions by connecting one vertex on each boundary to a vertex on another boundary (producing a zero area tunnel). This converts the polygon into a simple polygon without changing the concavity factor.

Now superimpose the polygon on a square lattice as in the circle problem. If there are multiple portions of the region overlapping the square, handle them separately.

If there is an acute angle in the square, subdivide the square as in the circle problem. This will produce at most three regions per acute vertex which can not be handled by the circle problem data structure. These can be handled as in the triangle problem.

All other cases are handled as in the circle problem.

Since the maximum number of acute angles, obtuse angles, the perimeter length, and the number of disconnected strips (when retrieving at the level of strips of squares) are all limited by a value proportional to the concavity factor, the retrieval can be performed in the time given above.

The $O(\varphi N^{(1-\lambda)\beta+\lambda} \log^3 N)$ in the retrieval time is the cost of retrieving the squares which don't contain any vertices. The $O(\varphi N^{(1-\lambda)\beta+2/3} \log^3 N)$ the maximum number of vertices times the cost to retrieve them. The $O(\varphi N^{2-2\lambda}) \log N)$ is the cost of retrieving the acute angles. □

5 Conclusion

We have presented a family of data structures which permit fast half-plane retrieval on a square grid with low storage redundancy. We further showed how one can make use of the ability to trade-off storage redundancy and retrieval time to perform fast retrievals on triangles, circles, and generalized polygons. Our methods permit a data structure to be designed for any of these retrievals which balances the retrieval time and the storage redundancy (and thus update time).

Our complexity measure is to count the number of semi-group operations performed (addition and assignment of values derived from the original square grid). Thus, arithmetic and indexing operations are not included. Assuming that these operations can be performed in time $O(B)$ where B is the number of bits to represent their arguments, this results in only an additional log term to the complexity of the algorithm.

For circles and generalized polygons with more than $N^{2/3}$ vertices (N is the size of the grid), one must determine the convex hull of the grid points inside the region to be retrieved. The time to determine the convex hull was also not considered here.

The conceptual advance in this work is the use of rotated retrieval structures. This work was guided by our knowledge of the lower bound. We were able to use the lower bound to inform us of what types of approaches might work, and what could not work.

Acknowledgement

We wish to thank Larry Larmore for help in developing the proof for Lemma 2.2 and for supplying us with the result convex hull result.

References

[Bentley 79]
Bentley, Jon Louis
Decomposable Searching Problems
Information Processing Letters 8:5 (June 11, 1979) pp 244-251

[Burkhard and Fredman 81]
Burkhard, Walter A. and Fredman, Michael L.
Inherent Complexity Trade-Offs for Range Query Problems
Theoretical Computer Science 16 (1981) 279-290, North Holland Publishing Company

[Fredman 80]
Fredman, Michael L.
The Inherent Complexity of Dynamic Data Structures Which Accommodate Range Queries
proc. 1980 IEEE conf. on Fundamentals of Computer Science pp 191-199

[Fredman 82]
Fredman, Michael L.
The Complexity of Maintaining an Array and Computing its Partial Sums
J. ACM 29:1 (Jan. 1982) pp 250-260

[Lueker and Willard 78]
Lueker, George S. and Willard, Dan E.
A Data Structure for Dynamic Range Queries
proc. 1978 IEEE conf. on Fundamentals of Computer Science

[Overmars and van Leeuwen 81]
Overmars, M.H. and van Leeuwen, J.
Two General Methods for Dynamizing Decomposable Searching Problems
Computing 26 (1981) pp 155-166

[Saxe and Bentley 79]
Saxe, James B. and Bentley, Jon Louis
Transforming Static Data Structures to Dynamic Structures ABRIDGED VERSION
proc. 1978 IEEE conf. on Fundamentals of Computer Science pp 148-168

[Willard 78]
Willard, Dan E.
New Data Structures for Orthogonal Queries
Research Report TR-22-78, Aiken Computation Laboratory, Harvard University

[Willard 82]
Willard, Dan E.
Polygon Retrieval
SIAM Journal on Computing 11:1 (Feb. 1982) pp 149-165

[Yao 83]
Yao, F. Frances
A 3-Space Partition and Its Applications
proc. ACM Symposium on Theory of Computing 1983 pp 258-263

A Limit on Vertices in Convex Polygon

Theorem A.1 *The number of vertices in a convex polygon whose vertices are at integer coordinates is $O(R^{2/3})$ where R is one half of the maximum distance between two vertices.*

Proof: Since the polygon is convex, each of its edges (taken in the clockwise orientation) has a different slope. If each of the edges is treated as a vector and translated to a common origin, each vector must terminate at a different integer point. Since the sum of the translated vectors is $2\pi R$ they can not all be contained in a circle of radius less than $\theta(R^{1/3})$. This implies that the average length of the vectors must be at least $\Omega(R^{1/3})$. The number of vertices is the perimeter of the polygon divided by the average length of the vectors. Therefore, the number of vertices is limited to $O(R^{2/3})$. □