Succinct Geometric Indexes Supporting Point Location Queries

Prosenjit Bose * Eric Y. Chen † Meng He * Anil Maheshwari * Pat Morin *

Abstract

We propose to design data structures called succinct geometric indexes of negligible space (more precisely, $o(n)$ bits) that support geometric queries in optimal time, by taking advantage of the n points in the data set permuted and stored elsewhere as a sequence. Our first and main result is a succinct geometric index that can answer point location queries, a fundamental problem in computational geometry, on planar triangulations in $O(\lg n)$ time[1]. We also design three variants of this index. The first supports point location using $\lg n + 2\sqrt{\lg n} + O(\lg^{1/4} n)$ point-line comparisons. The second supports point location in $o(\lg n)$ time when the coordinates are integers bounded by U. The last variant can answer point location queries in $O(H + 1)$ expected time, where H is the entropy of the query distribution. These results match the query efficiency of previous point location structures that occupy $O(n)$ words or $O(n \lg n)$ bits, while saving drastic amounts of space. We generalize our succinct geometric index to planar subdivisions, and design indexes for other types of queries. Finally, we apply our techniques to design the first implicit data structures that support point location in $O(\lg^2 n)$ time.

1 Introduction

The problem of efficiently storing and retrieving geometric data sets that typically consist of collections of data points and regions is fundamental in computational geometry. Researchers have designed many data structures to support various types of queries, such as point location [14, 16, 27, 31], nearest neighbour [25], range searching [1] and ray shooting [23]. Among these queries, planar point location is fundamental and has been studied extensively. Given a planar subdivision, the problem is to construct a data structure so that the face of the subdivision containing a query point can be located quickly. In the 1980s, various researchers [14, 16, 27, 31] designed data structures of $O(n)$ words, where n is the number of vertices of the planar subdivision, to support point location in $O(\lg n)$ time, which is asymptotically optimal.

Researchers have also considered improving the query efficiency of point location structures under various assumptions. Several researches [20, 32] considered the exact number of steps (i.e. point-line comparisons) required to answer point location queries. Seidel and Adamy [32] showed that there is an $O(n)$-word structure that can answer point location in $\lg n + 2\sqrt{\lg n} + O(\lg^{1/4} n)$ steps, and it can be constructed in $O(n \lg n)$ time. Researchers later considered the case where the query distribution is known. If the probability of the i^{th} face of the planar subdivision containing the query point is p_i, the lower bound of the expected time of answering a query under the binary decision tree model is the entropy $H = \sum_{i=1}^{f}(p_i \log_2 \frac{1}{p_i})$, where f is the number of faces. When the planar subdivision is a planar triangulation, data structures of $O(n)$ words can be constructed to answer point location queries in $O(H + 1)$ expected time [24] or even using $H + O(\sqrt{H} + 1)$ expected comparisons per query [3]. Recently, Chan [10] and Pătraşcu [29] considered the case where the coordinates of the points are integers bounded by $U \leq 2^w$, and proposed a linear-space structure that answers point location queries in $O(\min\{\lg n / \lg \lg n, \sqrt{\lg U}\})$ time.

As we have already seen, much work has been done to improve the query efficiency of point location. However, much less effort has been made to further reduce the storage cost. As a result of the rapid growth of geometric data sets available in GIS, spatial databases and graphics, many modern applications process geometric data measured in gigabytes or even terabytes. Although the above point location structures require linear space, the constants hidden in the asymptotic space bounds are usually large, so that they often occupy space many times the size of the data. When the data is huge, it is often impossible or at least undesirable to construct and store these data structures. Most data structures supporting other geometric queries are facing the same problem.

Some attempts, however, have been made to improve the space efficiency. Goodrich *et al.* [20] showed that given a planar triangulation, a structure of sublinear space can be constructed to answer point loca-

*School of Computer Science, Carleton University, Canada, {jit, mhe, anil, morin}@cg.scs.carleton.ca

†Cheriton School of Computer Science, University of Waterloo, Canada, y28chen@uwaterloo.ca

[1]We use $\lg n$ to denote $\lceil \log_2 n \rceil$.

tion queries in $O(\lg n)$ time. However, their approach assumed that the connectivity information (i.e. adjacencies) of the planar triangulation is given and stored elsewhere. This information can easily occupy much more space than that required to store the point coordinates, and can make the total space of the point location structure to be $O(n)$ words. By applying the idea of implicit data structures [28], researchers [7, 11] have designed some implicit geometric data structures. These data structures store a permuted sequence of the point set, so that with zero or $O(1)$ extra space, geometric queries can be answered efficiently. The recent result by Chan and Chen [11] was an implicit structure that answers 2D nearest neighbour query in $O(\lg^{1.71} n)$ time. This approach saves a lot of space, but there are still limitations. First, the above query time is not asymptotically optimal. Second, it is not known how to support point location in planar triangulations using implicit data structures.

Have researchers tried all the major known techniques to design space-efficient geometric data structures? The answer is no. There has been another line of research on data structures called *succinct data structures*. Succinct data structures were first proposed by Jacobson [26] to encode bit vectors, (unlabeled) trees and planar graphs in space close to the information-theoretic lower bound, while supporting efficient navigational operations. For example, Jacobson showed how to represent a tree on n nodes using $2n + o(n)$ bits, so that the parent and the children of a node can be efficiently located. The obvious approach uses $3n$ words, which is about 96 times as much as the space required for the succinct representation. This technique was successfully applied to various other abstract data types, such as dictionaries [30], strings [5, 6, 21], binary relations [5, 6] and labeled trees [5, 6, 17, 19]. It has also been applied to data structures related to computational geometry. There are succinct representations of planar triangulations and planar graphs [4, 8, 9, 12] that use $O(n)$ bits, and support queries such as testing the adjacency between two vertices in constant time. However, they only encode the connectivity information, so they are succinct graph data structures rather than succinct geometric data structures. It is not known how to combine them with point location structures without using $O(n)$ extra words or $O(n \lg n)$ bits.

1.1 Our Results In this paper, we design succinct geometric data structures. Given a geometric data set, our goal is to store the coordinates of the points as a permuted sequence, and design an auxiliary data structure called *succinct geometric index* that occupies negligible space (more precisely, $o(n)$ bits) to support various geometric queries in optimal time[2].

Our first and main result is a succinct geometric index of $o(n)$ bits that supports point location in $O(\lg n)$ time on planar triangulations (Section 3). The preprocessing time is $O(n)$. The data structure here has three levels, each of which is formed by choosing a separator made of triangles, which divides the remaining triangles into smaller regions. Two levels of this structure reduce the regions to a sufficiently small size so that known results on encoding triangulations as a permutation of their vertices can be applied. However, to use this to design a space-efficient point location structure, a data structure is required that translates indices of vertices from the lower levels to the higher levels. We construct such a structure using succinct data structures for representing bit vectors that support rank and select operations. Based on this, we design three variants of this index:

- A succinct geometric index that supports point location using $\lg n + 2\sqrt{\lg n} + O(\lg^{1/4} n)$ point-line comparisons, which matches the result of Seidel and Adamy [32] while occupying $o(n)$ bits. The preprocessing takes $O(n)$ time, which is an improvement upon the $O(n \lg n)$ preprocessing time of the latter structure.
- A succinct geometric index that supports point location in $o(\lg n)$ time when the coordinates are integers bounded by U.
- A succinct geometric index that answers point location queries in $O(H + 1)$ expected time, where H is the entropy of the query distribution.

All these results match the query efficiency of previous point location structures that occupy $O(n)$ words or $O(n \lg n)$ bits, while saving drastic amounts of space.

We then generalize our approach to planar subdivisions (we assume the subdivision is within a polygon boundary), and design an $o(n)$-bit index that supports point location in planar subdivisions in $O(\lg n)$ time (Section 4). Finally, we apply our techniques to related problems and design (Section 5):

- A succinct geometric index that can test whether a query point is inside a simple given polygon in $O(\lg n)$ time.
- A succinct geometric index that supports vertical ray shooting in $O(\lg n)$ time.
- The first implicit data structures that support point location in $O(\lg^2 n)$ time.

[2] When constructing succinct geometric indexes, we assume the word RAM model of $\Theta(\lg n)$-bit word size. However, the values of point coordinates can be unbounded, as long as each coordinate can be retrieved in constant time.

2 Preliminaries

Bit Vectors. A key structure for many succinct data structures and for our research is a bit vector $B[1..n]$ that supports `rank` and `select` operations. For $\alpha \in \{0,1\}$, the operator $\texttt{rank}_B(\alpha, x)$ returns the number of occurrences of α in $B[1..x]$, and the operator $\texttt{select}_B(\alpha, r)$ returns the position of the r^{th} occurrence of α in B. The subscript B is omitted when it is clear from the context. Lemma 2.1 addresses the problem of succinct representations of bit vectors, in which part (a) is from Jacobson [26] and Clark and Munro [13], and part (b) is from Raman *et al.* [30].

LEMMA 2.1. *A bit vector $B[1..n]$ with v 1s can be represented using either: (a) $n + o(n)$ bits, or (b) $\lg \binom{n}{v} + O(n \lg\lg n / \lg n)$ bits, to support the access to each bit,* `rank` *and* `select` *in $O(1)$ time.*

Graph Separators. The variant of graph separators we use is called *t-separators*. Let $G = \{V, E\}$ be a planar graph of n vertices, whose vertices have non-negative weights. A *t-separator* $(0 < t < 1)$ of G is a subset of V whose removal from G leaves no connected component of total weight more than $tw(G)$, where $w(G)$ is the sum of the weights of the vertices of G. Aleksandrov and Djidjev [2] proved:

LEMMA 2.2. ([2]) *Consider a planar graph G with n vertices, whose vertices have nonnegative weights. For any t such that $0 < t < 1$, there is a t-separator consisting of $O(\sqrt{n/t})$ vertices that can be computed in $O(n)$ time.*

Encoding a Planar Triangulation by Permuting its Vertex Set. Denny and Sohler [15] showed how to encode the connectivity information of a planar triangulation by permuting its vertex set:

LEMMA 2.3. ([15]) *Given a planar triangulation of n vertices, where $n > 1090$, there is an algorithm that can encode it as a permutation of its point set in $O(n)$ time, such that it can be decoded from this permutation in $O(n)$ time.*

3 Point Location in Planar Triangulations

In this section, we show how to design succinct geometric indexes to support point location queries in a planar triangulation G of n vertices, m edges and f internal faces. We define a planar triangulation to be a planar subdivision in which each face (including the outer face) is a triangle. For simplicity, we use the term planar triangulation to refer to both the triangulation itself (with coordinates), and the embedded abstract planar graph underlying it. We start with our scheme of partitioning the triangulation. We then show how to label its vertices (i.e. how to permute its point set). We finally design data structures and algorithms to support point location queries.

3.1 Partitioning a Planar Triangulation by Removing Faces

We present an approach to partition a planar triangulation by removing a set of internal faces. We first define the following terms. In a planar triangulation, two faces are *adjacent* if they have a common edge. A *face path* is a sequence of the faces such that each two consecutive faces in this sequence are adjacent. An *adjacent face component* is a set of internal faces of the graph in which there exists a face path between any two faces in this set, and no face in this set is adjacent to an internal face not in this set. We define the *size* of an adjacent face component to be the number of its internal faces. With these we can define the notion of graph separators consisting of faces:

DEFINITION 3.1. *Consider a planar triangulation with f internal faces. A* **t-face separator** *$(0 < t < 1)$ of G is a set of its internal faces of size $O(\sqrt{f/t})$ whose removal from G leaves no adjacent face component of more than tf faces.*

By applying Lemma 2.2 to the dual graph of the planar triangulation, we have:

LEMMA 3.1. *Consider a planar triangulation G of f internal faces. For any t such that $0 < t < 1$, there is a t-face separator consisting of $O(\sqrt{f/t})$ faces that can be computed in $O(f)$ time.*

We define the *boundary* of an adjacent face component to be a set of edges in which each edge is shared by an internal face of this component and a face of the t-face separator. Thus the boundary of an adjacent face component consists of one or more simple cycles: one simple cycle which is the outer face of the adjacent face component, and at most one simple cycle corresponding to each adjacent face component inside it. The simple cycle corresponding to the outer face does not share an edge with any cycle inside; otherwise, the cycle inside is simply part of the outer face. A useful observation is that any two such simple cycles do not have a common edge, because otherwise, there are two faces of G sharing an edge that are in two different adjacent face components, which contradicts Definition 3.1. This means that to bound the number of adjacent face components, we need only bound the number of edges of the separator. Therefore:

LEMMA 3.2. *Consider a planar triangulation G with f internal faces and a t-face separator S constructed using Lemma 3.1. The number of adjacent face components of $G \setminus S$ is $O(\sqrt{f/t})$.*

A *boundary vertex* of an adjacent face component is a vertex on the boundary of the component, and an *internal vertex* is a vertex inside it. A vertex *belongs to* an adjacent face component iff it is either a boundary vertex or an internal vertex of this component. The *duplication degree* of a vertex is the number of adjacent face components it belongs to. Thus an internal vertex has duplication degree 1. To bound the duplication degrees of boundary vertices, the following lemma is crucial (we omit the proof because of space constraints):

LEMMA 3.3. *Consider a planar triangulation G with f internal faces and a t-face separator S constructed using Lemma 3.1. The sum of the duplication degrees of all its boundary vertices is $O(\sqrt{f/t})$.*

3.2 The Two-Level Partitioning Scheme We perform two levels of partitioning on the input graph G, motivated by the two-level separator decomposition used in the fast planar shortest path algorithm of Fredrickson [18]. However, we apply the t-face separators defined in Section 3.1, with different parameters, based on which we construct new data structures for point location. Recall that G has n vertices and f internal faces. Thus $f = 2n - 5$. We first use Lemma 3.1 to partition G. We choose $t = (\lg^a f)/f$, where a is a positive constant parameter that we will fix later. Then the t-face separator, S, has $O(\sqrt{f/t}) = O(f/\lg^{a/2} f)$ faces and thus $O(n/\lg^{a/2} n)$ vertices. We call each adjacent face component of $G \setminus S$ a *region*. By Lemma 3.2, there are $r = O(\sqrt{f/t}) = O(f/\lg^{a/2} f)$ regions. Each region has at most $tf = \lg^a f$ internal faces, and thus $O(\lg^a n)$ vertices. We use R_i to denote the i^{th} region of G (the relative order of regions does not matter). We call this the *top-level partition* of G.

We perform another level of partitioning. For each region R_i, we triangulate the graph that consists of R_i and the triangular outer face of G, and we denote the resulting planar triangulation by R'_i. Thus R'_i has $O(\lg^a n)$ vertices. We partition R'_i into smaller "regions" called *subregions* so that each subregion has $O(\lg^b n)$ vertices, where b is a positive constant parameter smaller than a that we will fix later. We use $R_{i,j}$ to denote the j^{th} region of R_i. To do this, let n_i be the number of vertices in R_i. If $n_i > \lg^b n$ (otherwise, the entire region is also a subregion and the separator has size 0), we choose $t_i = (\lg^b n)/n_i$ and use Lemma 3.1 to construct a t_i-face separator, S_i, for R'_i. Then S_i has $O(n_i/\lg^{b/2} n)$ vertices. The sum of the numbers of vertices in all the S_i's is $\sum_{i=1}^{r} O(n_i/\lg^{b/2} n) = O(n/\lg^{b/2} n)$. By Lemma 3.2, there are $O(n_i/\lg^{b/2} n)$ subregions in R_i. Therefore, the number of subregions in G is $O(n/\lg^{b/2} n)$. This is the *bottom-level partition*.

3.3 The Labeling of the Vertices We now design a labeling scheme for the vertices based on the two-level partition in Section 3.2. This labeling scheme assigns a distinct number from the set $[n]$ to each vertex x of the graph[3]. We call this number the *graph-label* of x. For each region R_i, this labeling scheme also assigns a distinct number from the set $[n_i]$ to each vertex x in R_i, where n_i is the number of vertices in this region. We call this number the *region-label* of x. For each subregion $R_{i,j}$, a unique number from the set $[n_{i,j}]$ is assigned to each vertex in $R_{i,j}$, where $n_{i,j}$ is the number of vertices in this subregion. We call this number the *subregion-label* of x. Observe that, although each vertex x has one and only one graph-label, it may have zero, one or several region-labels or subregion-labels. This is because each vertex may belong to more than one region or subregion, or only belong to the separators.

We assign the labels from bottom up. We first assign the subregion-labels. Given subregion $R_{i,j}$, we use Lemma 2.3 to permute its vertices (we have to surround $R_{i,j}$ using a triangle and triangulate the resulting graph to use this lemma, but as the vertices of the added triangle are always the last three vertices when permuted, this does not matter). If a vertex x in $R_{i,j}$ is the k^{th} vertex in this permutation, then the subregion-label of x in $R_{i,j}$ is k.

To assign a region-label to a vertex x that belongs to region R_i, there are two cases. First, we consider the case where x belongs to one or more subregions in R_i. Let h_i be the number of vertices in R_i that belong to at least one subregion. We assign a distinct number from $[h_i]$ to each such vertex as its region-label in the following way: We visit each subregion $R_{i,j}$, for $i = 1, 2, \cdots, u_i$, where u_i is the number of subregions in R_i. When we visit $R_{i,j}$, we check all its vertices sorted by their subregion-labels in increasing order. We output a vertex of $R_{i,j}$ iff we have not output this vertex before (i.e. it does not belong to subregions $R_{i,1}, R_{i,2}, \cdots, R_{i,j-1}$). This way we output each vertex that belongs to at least one subregion in R_i exactly once. We assign the number k to the k^{th} vertex we output, and this number is its region-label in R_i. Second, we consider the case where x does not belong to any subregion in R_i. There are $n_i - h_i$ such vertices. We assign a distinct number from $\{h_i+1, h_i+2, \cdots, n_i\}$ to each of them in an arbitrary order, and the numbers assigned are their region-labels. The same approach is used to assign graph-labels to the vertices. More precisely, we permute the h vertices that have region-labels in the order we first visit them when we check all the vertices by region. The k^{th} vertex in such a

[3]We use $[i]$ to denote the set $\{1, 2, ..., i\}$.

permutation has graph-label k. We assign a distinct graph-label from $\{h+1, h+2, \cdots, n\}$ to the rest of the vertices of the graph in an arbitrary order. To perform conversions between different labels of the same vertex, we have:

LEMMA 3.4. *There is a data structure of $o(n)$ bits such that given a vertex x as a subregion-label k in subregion $R_{i,j}$, the region-label of x in R_i can be computed in $O(1)$ time. Similarly, there is a data structure of $o(n)$ bits such that given a vertex x as a region-label k in region R_i, the graph-label of x can be computed in $O(1)$ time if $a > 2$.*

Proof. To prove the first claim of this lemma, recall that we use u_i to denote the number of subregions in R_i, and h_i to denote the number of vertices in R_i that have subregion-labels. We denote the number of regions of G by r. We denote the number of vertices of subregions $R_{i,1}, R_{i,2}, \cdots, R_{i,u_i}$ by $n_{i,1}, n_{i,2}, \cdots, n_{i,u_i}$, respectively, where u_i is the number of subregions in R_i. Let n'_i be $\sum_{j=1}^{u_i} n_{i,j}$. As no internal vertex of a subregion occurs in another subregion, we need only consider the boundary vertices of the subregions to bound n'_i. Then by Lemma 3.3, we have $n'_i = h_i + O(\sqrt{n_i/t_i}) = h_i + O(n_i/\lg^{b/2} n) \leq n_i + O(n_i/\lg^{b/2} n)$. By the same lemma, we also have $\sum_{i=1}^{r} n_i \leq n + O(n/\lg^{a/2} n)$.

We consider a conceptual array, A_i, of length n'_i for each region R_i defined as follows. For each subregion $R_{i,j}$, we construct a conceptual array $A_{i,j}$ in which $A_{i,j}[k]$ stores the region-label of the vertex in $R_{i,j}$ whose subregion-label is k. Then $A_i = A_{i,1}A_{i,2}\cdots A_{i,u_i}$. Clearly A_i has the answers to our queries, but we do not store it explicitly. Instead, we construct for each region R_i:

- Bit vector $B_i[1..n'_i]$ to store $n_{i,1}, n_{i,2}, \cdots, n_{i,u_i}$ in unary, i.e. $B_i = 10^{n_{i,1}-1}10^{n_{i,2}-1}\cdots 10^{n_{i,u_i}-1}$;[4]
- Bit vector $C_i[1..n'_i]$ in which $C_i[k] = 1$ iff the first occurrence of the region-label $A_i[k]$ in A is at position k (let q_i denote the number of 0s in C_i);
- Array $D_i[1..q_i]$ in which $D_i[k]$ stores the region-label of the vertex that corresponds to the k^{th} 0 in C_i, i.e. $D_i[k] = A_i[\texttt{select}_{C_i}(0, k)]$.

To analyze the space cost of the above data structures constructed for the entire graph G, we first show how to store all the B_is. The first step is to concatenate all the B_is and store them as a single sequence B, and use part (b) of Lemma 2.1 to represent B. Let y be the length of B and z be the number of 1s in B. Then B can be stored in $\lg\binom{y}{z} + O(y \lg\lg y/\lg y)$ bits. As $n'_i \leq n_i + O(n_i/\lg^{b/2} n)$ and $\sum_{i=1}^{r} n_i \leq n + O(n/\lg^{a/2} n)$, we have $y = \sum_{i=1}^{r} n'_i \leq n + O(n/\lg^{a/2} n) + O(n/\lg^{b/2} n)$. By Lemma 3.2, we have $u_i = O(n_i/\lg^{b/2} n)$, so $z = \sum_{i=1}^{r} u_i = O(n/\lg^{b/2} n) + o(n/\lg^{b/2} n)$. As each subregion has $O(\lg^b n)$ vertices, we also have $z = \Omega(n/\lg^b n)$. By applying the inequality $\lg\binom{u}{v} \leq v \lg\frac{eu}{v} + O(1)$ [22, Section 4.6.4], we have $\lg\binom{y}{z} = O(n \lg\lg n/\lg^{b/2} n)$. Thus space cost of B is $O(n \lg\lg n/\lg^{b/2} n) + O(n \lg\lg n/\lg n) = o(n)$ bits. The rank/select operations on B can be performed in constant time, so in order to support the same operations on each B_i in constant time, it suffices to locate the starting position of any B_i in B in constant time. This can be done by using another bit vector, X, of length y to mark the starting positions of all the B_is in B. Thus X is of length at most $n + o(n)$ and it has $r = O(n/\lg^{a/2} n)$ 1s, which can be stored in $o(n)$ bits using part (b) of Lemma 2.1. Similarly, all the C_is and D_is can be stored in $O(n \lg\lg n/\lg^{b/2} n) + o(n)$ bits (each entry of D_i requires $O(\lg\lg n)$ bits).

We now show how to compute, given a vertex x as a subregion-label k in subregion $R_{i,j}$, the region-label of x in R_i. We first locate the position, l, in A_i that corresponds to the occurrence of vertex x in subregion $R_{i,j}$. As the vertex with subregion-label 1 in $R_{i,j}$ corresponds to position $\texttt{select}_{B_i}(1, j)$ in A_i, we have $l = \texttt{select}_{B_i}(1, j) + k - 1$. We then retrieve $C_i[l]$. If it is 1, then the first occurrence of the region-label of x in A is at position l, so $\texttt{rank}_{C_i}(1, l)$ is the result. Otherwise, the result is stored in $D_i[\texttt{rank}_{C_i}(0, l)]$. The above operations clearly take constant time.

The second claim of the lemma can be proved similarly, and the space required is $O(n/\lg^{a/2-1} n) + o(n)$ bits, which is $o(n)$ bits when $a > 2$. □

3.4 Answering Point Location Queries

THEOREM 3.1. *Given a planar triangulation G of n vertices, there is a succinct geometric index of $o(n)$ bits that supports point location in $O(\lg n)$ time. This index can be constructed in $O(n)$ time.*

Proof. We perform the two-level partitioning of G as in Section 3.2, and use the approach in Section 3.3 to assign labels to the vertices of G, but we do not store these labels. Instead, we sort the vertices by their graph-labels in increasing order, and store their coordinates as a sequence.

The succinct geometric index consists of three sets of data structures. This first set of data structures are the data structures constructed in Lemma 3.4 that supports conversions between subregion-labels, region-labels and graph-labels. They occupy $O(n/\lg^{a/2-1} n) + o(n)$ bits.

[4] We use 0^l to denote a bit sequence of l 0s.

The second set of structures are for the top-level partition. We consider the graph S' constructed by triangulating the graph consisting of the separator S and the outer face of G. S' is a planar triangulation of $O(n/\lg^a n)$ vertices, so we can use the approach of Kirkpatrick [27] (any structure that uses $O(n \lg n)$ bits for an n-vertex planar triangulation to answer point location in $O(\lg n)$ time can be used here) to construct a data structure P of $O(n/\lg^a n)$ words (i.e. $O(n/\lg^{a-1} n)$ bits) to support point location in S'. Note that when we construct P, we simply use the graph-label of any vertex to refer to its coordinates, so that we do not store any coordinate in P. For each face of S', we store an integer. If this face is in region R_i, we store i. We store 0 if it is a face of S. As $\lg n$ bits are sufficient to encode each integer, these values occupy $O(n/\lg^{a-1} n)$ bits.

The third set of structures are constructed over the regions of G. Given a region R_i, recall that we construct a planar triangulation R'_i to perform the bottom-level partition. Consider the graph S'_i constructed by triangulating the graph consisting of the separator S_i and the outer face of R'_i. Then S'_i is a planar triangulation of $O(r_i/\lg^{b/2} n)$ vertices. As $r_i = O(\lg^a n)$, a pointer that refers to a vertex of S'_i can be stored in $O(\lg\lg n)$ bits. To refer to the coordinates of any vertex of S'_i, we use its region-label so that we can use Lemma 3.4 to compute its graph-label in $O(1)$ time, and $O(\lg\lg n)$ bits are sufficient to store a region-label. Thus we can use the approach of Kirkpatrick [27] to construct a data structure P_i of $O(r_i \lg\lg n/\lg^{b/2} n)$ bits, to support point location in S'_i in $O(\lg\lg n)$ time. We also store a number for each face of S'_i, and this number is j if this face is in subregion $R_{i,j}$, and 0 if it is a face in separator S_i. As there are $O(r_i/\lg^{b/2} n) = O(\lg^{a-b/2} n)$ subregions in R'_i, each number occupies $O(\lg\lg n)$ bits. Thus these numbers can be stored in $O(r_i \lg\lg n/\lg^{b/2} n)$ bits. These data structures for all the regions use $\sum_{i=1}^{r} O(r_i \lg\lg n/\lg^{b/2} n) = O(n \lg\lg n/\lg^{b/2} n) = o(n)$ bits. Therefore, the succinct geometric index occupies $O(n/\lg^{a/2-1} n) + o(n)$ bits.

We now show how to answer point location queries using this index. Given a query point x, we first locate the face of S' that contains x using P in $O(\lg n)$ time. We retrieve the integer, i, assigned to the face of S' that x is in. If i is 0, then this face is in S, and we return its three vertices as the result. If it is not, then x is inside region R_i. We then use P_i to perform a point location query on the graph S'_i in $O(\lg\lg n)$ time using x as the query point. We retrieve the integer, j, assigned to the face of S'_i that x is in. If j is 0, then this face is in S_i, and we return its three vertices as the result (we need convert the region-labels of these three vertices to their graph-labels when we return them). If j is not, then x is inside region $R_{i,j}$. Using the bit vector B_i constructed in the proof of Lemma 3.4, we can compute the number of vertices of $R_{i,j}$ in constant time. Recall that $r_{i,j}$ denotes this number. By Lemma 3.4, we can compute the graph-label of any of these vertices in constant time, and thus retrieve its coordinates in $O(1)$ time. As we number these vertices using Lemma 2.3, we can use Lemma 2.3 to construct the graph $R'_{i,j}$ in $O(r_{i,j})$ time, and then check each of its faces to find the answer. This takes $O(r_{i,j}) = O(\lg^b n)$ time. Therefore, the entire process takes $O(\lg n + \lg^b n)$ time.

We have so far designed a succinct geometric index of $O(n/\lg^{a/2-1} n) + o(n)$ bits to support point location in $O(\lg n + \lg^b n)$ time. Choosing $a = 3$ and $b = 1$ yields an index of $o(n)$ bits that supports point location in $O(\lg n)$ time. It is easy to prove that this index can be built in $O(n)$ time. □

3.5 Three Variants of the Index We now design three variants of this succinct geometric index to address the query efficiency with different assumptions. We first consider the exact number of point-line comparisons:

COROLLARY 3.1. *Given a planar triangulation G of n vertices, there is a succinct geometric index of $o(n)$ bits that supports point location using at most $\lg n + 2\sqrt{\lg n} + O(\lg^{1/4} n)$ steps. This index can be constructed in $O(n)$ time.*

Proof. We use the same approach as that for Theorem 3.1. When constructing the data structures for the top-level partition, we use the approach by Seidel and Adamy [32] to construct the point location structure P. Thus, point location in S' can be computed in at most $\lg n + 2\sqrt{\lg n} + O(\lg^{1/4} n)$ steps. For the bottom-level partition, we choose $b = 1/4$. As the point location in S'_i and $R'_{i,j}$ can be supported in $O(\lg\lg n)$ and $O(\lg^b n) = O(\lg^{1/4} n)$ steps, respectively, the corollary follows. □

Corollary 3.1 not only uses negligible space to match the best result [32] in terms of the exact number of point-line comparisons, but also reduces the preprocessing time from $O(n \lg n)$ [32] to $O(n)$. If all the coordinates are integers bounded by $U \leq 2^w$, we have:

COROLLARY 3.2. *Assume that all the point coordinates in the plane are integers bounded by $U \leq 2^w$. Given a planar triangulation G of n vertices, there is a succinct geometric index of $o(n)$ bits that supports point location in $O(\min\{\lg n/\lg\lg n, \sqrt{\lg U}\} + \lg^\epsilon n)$ time, for any constant $\epsilon > 0$. This index can be constructed in $O(n)$ time.*

Proof. We use the approach of Theorem 3.1, but we choose $b = \epsilon$. We use the approach by Chan [10] and Pătraşcu [29] to construct P in $O(n/\lg^3 n)$ time, so that point location in S' can be computed in $O(\min\{\lg n/\lg\lg n, \sqrt{\lg U}\})$ time. As point location in S'_i and $R'_{i,j}$ is supported in $O(\lg\lg n)$ and $O(\lg^b n) = O(\lg^\epsilon n)$ time, respectively, the corollary follows. □

When query distribution is known, we have:

COROLLARY 3.3. *Given a planar triangulation G of n vertices, there is a succinct geometric index of $o(n)$ bits that supports point location in $O(H+1)$ expected time, where H is the entropy of the query distribution. This index can be constructed in $O(n)$ time.*

Proof. If the probability of a face or a set of faces containing a query point is p, we say that this face or this set of faces has probability p. In this proof, we define the t-face separator to be the set of faces of G whose removal partitions G into adjacent face components each of which has probability at most t. We consider graph G^*, which is the dual graph of G excluding the vertex corresponding to the outer face of G and its incident edges, and assign the probability of each face of G as the weight to its corresponding vertex in G^*. By Lemma 2.2, the following lemma is immediate:

LEMMA 3.5. *Consider a planar triangulation G of f internal faces, where the weight of a face is its probability. For any t such that $0 < t < 1$, there is a t-face separator consisting of $O(\sqrt{f/t})$ faces that can be computed in $O(n)$ time.*

Observe that Lemmas 3.2 and 3.3 also apply to t-face separators for graphs whose faces are associated with probabilities. This is because we prove these two lemmas by bounding the number of edges in the separator, which has nothing to do with probabilities.

We choose $t = \lg^3 f/f$ to apply Lemma 3.5 to G. Let S'' be the t-face separator. Then S'' has $O(n/\lg^{3/2} n)$ vertices. We call each adjacent face component of $G \setminus S$ a *super region*. Thus the number of super regions is $O(n/\lg^{3/2} n)$, and the sum of the duplication degrees of the boundary vertices of all the super regions is also $O(n/\lg^{3/2} n)$. Note that we can no longer prove that each super region has $o(n)$ vertices; this is because a super region can have a large number of faces with very low probabilities. Thus we further perform a two-level partition on each super region as in the proof of Theorem 3.1. Therefore, we actually perform a three-level partition on G, and we call them first-level, second-level and third-level partitions from top down. It is straightforward to extend the techniques in Section 3.2 to this case to compute the permuted sequence of the vertices, and to perform conversions between the labels assigned to the same vertices at different levels of the partition.

For the first level partition, we construct a triangulated graph G'' by triangulating the graph consisting of S'' and the outer face of G. To assign a probability to each face of G'', initially we let the probability of each face of S'' to be the same as its probability in G, and let the probability of any other internal face to be $1/n$. However, the sum of all the probabilities of the faces of G'' can be larger than 1, though it is at most 2. We thus reduce the probability of each face of G'' by a constant ratio, so that the sum becomes 1. It is clear that the above process reduces the probability of each face by at most half. Therefore, the probability of each face of S'' in G'' is at least half of that in G, and the probability of each internal face of G'' that is not in S'' is at least $1/(2n)$. We construct a point location structure, P'', for G'' using the approach of Iacono [24], or any linear-space structure that answers point location in $O(\lg(1/p))$ time, if the query point is contained in a face of probability p. P'' occupies $O(n/\lg^{1/2} n)$ bits. We also store additional information for each face of G'' to indicate whether it is a face in S'' and, if not, which super region it is in. For the second-level and third-level structures, we construct data structures similar to those constructed in Theorem 3.1. The algorithm to answer point location queries is similar, except that we now perform operations at three levels of partition.

To analyze the query time, it is sufficient to show that, if the face, z, of G that contains the query point x has probability p, the query can be answered in deterministic time $O(\min\{\lg n, \lg(1/p)\})$. There are two cases. First, z is a face in S''. In this case, we need only use P'' to retrieve the result. By Iacono's result [24], the time required is $O(\min\{\lg n, \lg(1/p')\})$, where p' is the probability of z in G''. By the analysis in the above paragraph, $p' > p/2$. Thus the claim is true in this case. Second, z is not a face in S''. In this case, the query is answered in $O(\lg n)$ time. Thus it suffices to prove that $O(\min\{\lg n, \lg(1/p)\}) = O(\lg n)$. Recall that each super region has probability at most $t = \lg^3 n/n$. As z is part of a super region, we have $p \le \lg^3 n/n$. Thus $\lg(1/p) \ge \lg n - 3\lg\lg n$, and the claim follows.

It is straightforward to show that the space cost is $o(n)$ bits and that the preprocessing time is $O(n)$. □

4 Point Location in Planar Subdivisions

We now generalize the techniques of Section 3 to general planar subdivisions. We adopt the assumption that a planar subdivision G is inside a bounding simple polygon (i.e. it does not have any infinite faces), and

each face is also a simple polygon. Let n, m and f denote the numbers of vertices, edges and internal faces of G, respectively. We omit all the proofs in this section because of space constraints.

4.1 Partitioning a Planar Subdivision by Removing Faces The definitions of adjacent face component and t-face separator in Section 3.1 can be directly applied to planar subdivisions. Observe that the correctness of Lemma 3.1 does not require the fact that each face of a planar triangulation is a triangle. Thus this lemma also applies to planar subdivisions. We also define the notion of boundary, boundary vertex, internal vertex and duplication degree on planar subdivisions as in Section 3.1. Same as the case of planar triangulations, we can bound the number of adjacent face components and the sum of the duplication degrees of boundary vertices after removing a t-face separator, S, from a planar subdivision G. The only difference is when we count the number of edges in S, we can no longer use the fact that each face has three edges. Instead, we use the maximum number, k, of vertices on an internal face of the planar subdivision. We immediately have that the number of adjacent face components of $G \setminus S$ is $O(k\sqrt{f/t})$, and that the sum of the duplication degrees of all its boundary vertices is also $O(k\sqrt{f/t})$.

4.2 The Two-Level Partitioning Scheme We use the techniques of Section 4.1 to partition G, but we cannot use it directly, as f can be as small as 1 for any n. Instead, we divide the faces with sufficiently many vertices into smaller faces whose sizes are bounded by non-constant parameters. It may seem odd not to simply divide the faces into triangles, but it is crucial to choose a non-constant parameter for our solution.

LEMMA 4.1. *Consider a simple polygon P of n vertices. Given an integer l where $l < n$, there is an $O(n)$-time algorithm that can, by adding edges between the vertices of P that only intersect at the vertices of P, divide the interior of P into a planar subdivision such that each internal face has at least l vertices (with the exception of at most one internal face) and at most $3l$ vertices.*

With the above lemma, we can now present our partitioning scheme. We choose $l = (\lg^2 n)/3$ and use Lemma 4.1 to divide each internal face of G that has more than $\lg^2 n$ vertices into smaller faces. We denote the resulting graph by G'. Thus any internal face of G' has at most $\lg^2 n$ vertices. We call a face of G' that is a face of G an *original face*, and a face of G' that is part of a larger face of G a *modified face*. Then the total number of modified faces of G' is $O(f/l) = O(n/\lg^2 n)$. Let f' be the number of internal faces of G'. Then we have $4(2n-5)/\lg^2 n \le f' \le 2n-5$.

For the top-level partition, we choose $t = (\lg^8 f')/f'$. Then the t-face separator, S, of G' has $O(f'/\lg^4 f') = O(n/\lg^4 n)$ faces. As each face of G' has at most $\lg^2 n$ vertices, S has $O(n/\lg^2 n)$ vertices. We call each adjacent face component of $G' \setminus S$ a *region*. Then there are $r = O(n/\lg^2 n)$ regions. Each region has at most $\lg^8 f' = O(\lg^8 n)$ faces, and thus $O(\lg^{10} n)$ vertices. The sum of the duplication degrees of the boundary vertices of all the regions is $O(n/\lg^2 n)$, so $\sum_{i=1}^{r} n_i = n + o(n)$.

Consider a region R_i. Let f_i and n_i be the number of faces and vertices of R_i, respectively. Then $f_i = O(\lg^8 n)$ and $n_i = O(\lg^{10} n)$. We choose $l = (\lg^{1/4} n)/3$ to apply Lemma 4.1 to divide each internal face of R_i that has more than $\lg^{1/4} n$ vertices into smaller faces. We denote the resulting graph by R_i'. Thus any internal face of R_i' has at most $\lg^{1/4} n$ vertices. We call a face of R_i' that is a face of R_i an *original region face*, and a face of R_i' that is part of a larger face of R_i a *modified region face*. Same as the analysis for G', we have that the number of modified region faces in R_i is $O(n_i/\lg^{1/4} n)$, so the total number of modified region faces in all the regions of G' is $O(n/\lg^{1/4} n)$. Let f_i' be the number of internal faces of R_i'. We also have $4(2n_i-5)/\lg^{1/4} n \le f_i' \le 2n_i - 5$.

We perform bottom-level partition on each region R_i. Let $t_i = (\lg^{3/4} n)/f_i'$. We compute a t_i-face separator, S_i, for R_i'. Then S_i has $O(f_i'/\lg^{3/8} n)$ faces, and thus $O(f_i'/\lg^{1/8} n)$ vertices. We call each adjacent face component of $R_i' \setminus S_i$ a *subregion* of R_i' (or R_i), and we denote the j^{th} subregion of R_i' by $R_{i,j}$. Then in Region R_i, there are at most $O(\lg^{1/4} n \times \sqrt{f_i'/t_i}) = O(f_i'/\lg^{1/8} n)$ subregions. As $f_i' \le 2n_i - 5$, the total number of subregions of all the regions of G_i is $O(n/\lg^{1/8} n)$. The number of faces of each subregion is at most $t_i f_i' = \lg^{3/4} n$, so each subregion has at most $\lg n$ vertices. The sum of the duplication degrees of the boundary vertices of the subregions in R_i is $O(n_i/\lg^{1/8} n)$, so the sum of the duplication degrees of all the boundary vertices of the subregions in G is $O(n/\lg^{1/8} n)$.

4.3 Vertex Labels and Face Labels The same scheme of Section 3.3 is used to assign subregion-labels, region-labels and graph-labels to the vertices of G. The only difference is that to permute the vertices of a subregion, we generalize Lemma 2.3 to planar subdivisions. The techniques of Lemma 3.4 can also be used here. The analysis of the number of regions/subregions and the sum of duplication degrees of boundary vertices in Section 4.2 guarantees that the

space needed is still $o(n)$ bits. Thus we can use $o(n)$ bits to support the $O(1)$-time conversion from a subregion-label (or region-label) in a subregion (or region), to the corresponding region-label (or graph-label).

We also need to design a labeling scheme for the faces. We do not have to do this for planar triangulations, because in that case, each face has three vertices, and it is sufficient to locate these three vertices to return the face. However, we cannot do so for general planar subdivisions, because a face may have a large number of vertices, and it may take too much time to return them.

For each face of G, we assign a distinct number called *graph-id* from the set $[f]$. This is the identifier we return when answering point location queries. For each face in a region R_i, we also assign a distinct number called *region-id* from the set $[f_i]$. Note that a face of the region R_i is not necessarily a face of G (i.e. it can be a modified face). For each face in a subregion $R_{i,j}$, we assign a distinct number called *subregion-id* from the set $[f_{i,j}]$, where $f_{i,j}$ is the number of faces in $R_{i,j}$. Again a face of $R_{i,j}$ is not necessarily a face of R_i. We number the faces from bottom up. For each subregion $R_{i,j}$, we list its faces in a canonical order (such as BFS order) of the corresponding vertices of its dual graph. The k^{th} face listed is assigned k as its subregion-id in $R_{i,j}$. The same technique used for vertices can be used to assign region-ids and graph-ids to the faces. We have the following lemma to perform conversions between different identifiers of the same face:

LEMMA 4.2. *There is a data structure of $o(n)$ bits such that given a face x with subregion-id k in subregion $R_{i,j}$, the region-label of the face of R_i that contains x can be computed in $O(1)$ time. Similarly, there is a data structure of $o(n)$ bits such that given a face x with region-id k in region R_i, the graph-id of the face of G that contains x can be computed in $O(1)$ time.*

4.4 Answering Point Location Queries To support point location, we perform two-level partition on G as in Section 4.2, and assign labels and identifiers to vertices and faces as in Section 4.3. Same as the proof of Theorem 3.1, we build point location structures at different levels to answer queries. We have the following theorem:

THEOREM 4.1. *Given a planar subdivision G of n vertices, there is a succinct geometric index of $o(n)$ bits that supports point location in $O(\lg n)$ time. This index can be constructed in $O(n)$ time.*

We can use this theorem to solve the *membership query* problem: given a simple polygon and a query point, we want to test whether the polygon contains the query point.

COROLLARY 4.1. *Given a simple polygon of n vertices, there is a succinct geometric index of $o(n)$ bits that supports membership query in $O(\lg n)$ time. This index can be constructed in $O(n)$ time.*

5 Applications

Vertical Ray Shooting. Given a set of disjoint line segments, the *vertical ray shooting query* is to return the line segment immediately above (or below) a given query point. The following theorem presents a succinct geometric index for this problem:

THEOREM 5.1. *Given a set of disjoint line segments in the plane, there is a succinct geometric index of $o(n)$ bits that supports vertical ray shooting in $O(\lg n)$ time. This index can be constructed in $O(n \lg n)$ time.*

Implicit Geometric Data Structures. If we group the vertices of a planar triangulation in pairs and permute these pairs to encode subregions, we can encode one bit of information by swapping each pair. This can encode our index in Theorem 3.1 implicitly. Thus:

THEOREM 5.2. *Given a planar triangulation of n vertices, there is a permutation of the vertex coordinates array that can support point location in $O(\lg^2 n)$ with $O(1)$ words of working space.*

Using the same technique, we can design implicit data structures to answer point location queries in an arbitrary planar subdivision and to support vertical ray shooting in $O(\lg^2 n)$ time.

References

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 23 of *Contemporary Mathematics*, pages 1–56. 1999.

[2] L. Aleksandrov and H. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM Journal on Discrete Mathematics*, 9(1):129–150, 1996.

[3] S. Arya, T. Malamatos, D. M. Mount, and K. C. Wong. Optimal expected-case planar point location. *SIAM Journal on Computing*, 37(2):584–610, 2007.

[4] J. Barbay, L. Castelli Aleardi, M. He, and J. I. Munro. Succinct representation of labeled graphs. In *Proceedings of the 18th International Symposium on Algorithms and Computation*, pages 316–328. Springer-Verlag LNCS 4835, 2007.

[5] J. Barbay, A. Golynski, J. I. Munro, and S. S. Rao. Adaptive searching in succinctly encoded binary re-

lations and tree-structured documents. *Theoretical Computer Science*, 387(3):284–297, 2007.

[6] J. Barbay, M. He, J. I. Munro, and S. S. Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 680–689, 2007.

[7] H. Brönnimann, T. M. Chan, and E. Y. Chen. Towards in-place geometric algorithms and data structures. In *Proceedings of the 20th ACM Annual Symposium on Computational Geometry*, pages 239–246, 2004.

[8] L. Castelli Aleardi, O. Devillers, and G. Schaeffer. Succinct representation of triangulations with a boundary. In *Proceedings of the 9th Workshop on Algorithms and Data Structures*, volume 3608 of *LNCS*, pages 134–145. Springer, 2005.

[9] L. Castelli Aleardi, O. Devillers, and G. Schaeffer. Optimal succinct representations of planar maps. In *Proceedings of the 22nd Annual ACM Symposium on Computational Geometry*, pages 309–318, 2006.

[10] T. M. Chan. Point location in o(log n) time, voronoi diagrams in o(n log n) time, and other transdichotomous results in computational geometry. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 333–344, 2006.

[11] T. M. Chan and E. Y. Chen. In-place 2-d nearest neighbor search. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 904–911, 2008.

[12] Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. Orderly spanning trees with applications. *SIAM Journal on Computing*, 34(4):924–945, 2005.

[13] D. R. Clark and J. I. Munro. Efficient suffix trees on secondary storage. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 383–391, 1996.

[14] R. Cole. Searching and storing similar lists. *Journal of Algorithms*, 7(2):202–220, 1986.

[15] M. Denny and C. Sohler. Encoding a triangulation as a permutation of its point set. In *Proceedings of the 9th Canadian Conference on Computational Geometry*, 1997.

[16] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

[17] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 184–196, 2005.

[18] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987.

[19] R. F. Geary, R. Raman, and V. Raman. Succinct ordinal trees with level-ancestor queries. *ACM Transactions on Algorithms*, 2(4):510–534, 2006.

[20] M. T. Goodrich, M. W. Orletsky, and K. Ramaiyer. Methods for achieving fast query times in point location data structures. In *Proceedings of the 8th Annual ACM-SIAM symposium on Discrete algorithms*, pages 757–766, 1997.

[21] R. Grossi, A. Gupta, and J. S. Vitter. High-order entropy-compressed text indexes. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 841–850, 2003.

[22] M. He. *Succinct Indexes*. PhD thesis, University of Waterloo, December 2007.

[23] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995.

[24] J. Iacono. Expected asymptotically optimal planar point location. *Computational Geometry*, 29(1):19–22, 2004.

[25] P. Indyk. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, chapter 39*. CRC Press, 2004. 2rd edition.

[26] G. Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 549–554, 1989.

[27] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[28] J. I. Munro. An implicit data structure supporting insertion, deletion, and search in $O(\log^2 n)$ time. *Journal of Computer and System Sciences*, 33:66–74, 1986.

[29] M. Pătraşcu. Planar point location in sublogarithmic time. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 325–332, 2006.

[30] R. Raman, V. Raman, and S. R. Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms*, 3(4):43, 2007.

[31] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.

[32] R. Seidel and U. Adamy. On the exact worst case query complexity of planar point location. *Journal of Algorithms*, 37(1):189–217, 2000.