Succinct and Implicit Data Structures for Computational Geometry

Meng He

Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 4R2, Canada,
mhe@cs.dal.ca

Abstract. Many classic data structures have been proposed to support geometric queries, such as range search, point location and nearest neighbor search. For a two-dimensional geometric data set consisting of n elements, these structures typically require $O(n)$, close to $O(n)$ or $O(n \lg n)$ words of space; while they support efficient queries, their storage costs are often much larger than the space required to encode the given data. As modern applications often process very large geometric data sets, it is often not practical to construct and store these data structures.
This article surveys research that addresses this issue by designing space-efficient geometric data structures. In particular, two different but closely related lines of research will be considered: *succinct geometric data structures* and *implicit geometric data structures*. The space usage of succinct geometric data structures is equal to the information-theoretic minimum space required to encode the given geometric data set plus a lower order term, and these structures also answer queries efficiently. Implicit geometric data structures are encoded as permutations of elements in the data sets, and only zero or $O(1)$ words of extra space is required to support queries. The succinct and implicit data structures surveyed in this article support several fundamental geometric queries and their variants.

1 Introduction

Many applications such as spatial databases, computer graphics and geographic information systems store and process geometric data sets that typically consist of point coordinates. In other applications such as relational databases and data mining applications, the given data are essentially sets of records whose fields are values of different properties, and thus can be modeled as geometric data in multidimensional space. Thus the study of geometric data structures which can potentially be used to preprocess these data sets so that various queries can be performed quickly is critical to the design of a large number of efficient software systems.

Researchers have studied many different geometric queries. Among them, the following three geometric query problems are perhaps the most fundamental:

- *Range search*: Preprocess a point set, so that information regarding points inside a query region, e.g., the number of points in this region, can be efficiently computed;

– *Point location*: Preprocess a subdivision of space into a set of cells, so that the cell that contains a query point can be located quickly;
– *Nearest neighbor*: Preprocess a point set, so that the point closest to a query point can be found efficiently.

Extensive work has been done to design data structures that provide fast support for these queries. Take planar point location for example. A number of different data structures have been designed to support point location in 2D, achieving $O(\lg n)$[1] optimal query time using linear space, including the classic data structures proposed by several different groups of researchers in the 70's and 80's [53, 52, 32, 28, 68]. Such work has yielded a large number of solutions to geometric query problems, and invented many data structure design techniques.

Most geometric data structures designed to manipulate data sets in two-dimensional space use linear, almost linear, or $O(n \lg n)$ words of space. Asymptotically, $O(n)$-space structures occupy less space than other solutions, but the constants hidden in the asymptotic space bounds are however usually large. As modern applications often process large data sets measured in terabytes or even petabytes, it has become undesirable or even infeasible to construct these data structures. Even for smaller data sets of several gigabytes, the storage cost of these structures often makes it impossible to load them into internal memory of computer systems, and performance is sacrificed if they have to be stored in external memory which is orders of magnitude slower. Thus, during the past decade, researchers have been developing data structure techniques that can be used to reduce storage cost drastically.

Succinct data structures have been used to design more space-efficient solutions to geometric query problems. Initially proposed by Jacobson [50] to represent combinatorial objects such as bit vectors, trees and graphs, the research in succinct data structures aims at designing data structures that can represent the given data using space close to the information-theoretic lower bound, and support efficient navigational operations in them. Numerous results have been achieved to represent combinatorial objects and text indexes succinctly [61, 66, 10, 64, 35, 67, 62], which save a lot of space compared with standard data structures.

The study of *implicit data structures* is another line of research that is focused on improving space efficiency of data structures, and it has also been applied to computational geometry. The term implicit means that structural information is encoded in the relative ordering of data elements, rather than explicit pointers. Thus an implicit data structure is stored as a permutation of its data elements, and with zero or a constant number of words of additional space, it can support operations. The standard binary heap is one of the earliest and most well-known data structures that are implicit. Later, researchers have designed implicit data structures for a number of problems such as partial match on a set of multi-key records [59, 40, 6] and dynamic predecessor search [60, 39].

The aim of this article is to survey research work that has been conducted to design succinct geometric data structures and implicit geometric data structures,

[1] $\lg n$ denotes $\log_2 n$.

in order to help researchers understand and follow research work in this area. The main ideas of the most relevant work are summarized, and main results are presented in theorems. To help readers evaluate these contributions, some background information of each geometric query discussed in this article is also given, though a thorough review of all the related work which is not necessarily focused on space efficiency issues is out of the scope of this survey. In the rest of this article, we devote one section to each of the three fundamental geometric queries listed in this section, to discuss succinct and implicit data structures designed for these queries and their variants. We present some concluding remarks and give some open problems in the last section.

2 Range Search

In this section, we survey succinct and implicit data structures designed for range search. As there are many variants of range search queries and they may be studied under different models, we further divide this section into subsections. In each subsection, before we describe succinct or implicit solutions, we define each range query and give some background information.

2.1 Implicit Data Structures for Orthogonal Range Reporting

Orthogonal range reporting is a fundamental range search problem. In this problem, we preprocess a set, N, consisting of n points in d-dimensional space, so that given a d-dimensional axis-aligned query box P, the points in $N \cap P$ can be reported efficiently. Thus in the two-dimensional case, P is an orthogonal query rectangle. We follow the convention and let k denote the number of points to be reported.

This problem has been studied extensively. The classic kd-tree proposed by Bentley [11] is a linear-space data structure that can answer orthogonal range reporting queries in $O(n^{1-1/d} + k)$ time. Thus in the two-dimensional case, the query time is $O(\sqrt{n} + k)$. By allowing penalties for each point to be reported, different query time can be achieved; in two-dimensional space, the linear-space data structure of Chazelle [25] can answer queries in $O(\lg n + k \lg^{\epsilon} n)$ time, for any positive constant ϵ. However, to achieve the optimal $O(\lg n + k)$ query time, more space is required. The data structure of Chazelle [25] uses $O(n \lg^{\epsilon} n)$ words of space to answer queries in optimal time, for any positive constant ϵ. For more recent results that aim at improving query time in higher dimensions using more space under the RAM model, see the work of Chan *et al.* [22]. For similar work under the pointer machine model, we refer to results presented by Afshani *et al.* [1, 2].

The first implicit data structure for range reporting is an implicit representation of kd-tree proposed by Munro in 1979 [59]. This is the only implicit or succinct geometric structure that was not designed within the past decade. Originally, this structure was introduced as a structure storing multi-key data records

to support partial matching, but its description can be easily rewritten to support geometric queries as follows. To construct this data structure, let $C[0..n-1]$ be the array in which each element stores the coordinates of a point in N, and the construction algorithm reorders the elements of C to encode the kd-tree implicitly. Assume that dimensions are numbered $0, 1, \ldots, d-1$. The construction algorithm consists of $\lceil \lg n \rceil$ stages. Initially, before stage 0, we treat the entire array C as one segment containing all the array entries; in each stage, we essentially divide each segment of C produced in the previous stage into two halves and reorder elements accordingly. More precisely, in stage i, for each segment S produced in the previous stage, we perform the following three steps:

1. Look for the point whose coordinate in dimension $(j = i \bmod d)$ is the median among the coordinates of all the points in S in this dimension, and swap it with the point stored in the middle of S. Let m be this median value.
2. Move the points in S whose coordinates in dimension j are smaller than m to the first half of S, and those with large coordinates to the second half.
3. Let the first half and the median element be one new segment, and the second half the other new segment.

At the end of the last stage, each segment will store exactly one point, and the content of array C becomes the implicit structure.

Readers who are familiar with kd-trees can now see that after this construction algorithm, the array C encodes a kd-tree implicitly. If we number the levels of a kd-tree starting from the root level as levels $0, 1, \ldots$, then a segment produced in the ith stage in this algorithm corresponds to a node at the ith level of the kd-tree. Furthermore, the region represented by each node can be inferred during a top-down traversal by checking the median values. Thus we can modify algorithms over kd-trees to support orthogonal range reporting: Start from the segment that contains all the points. Each time we investigate a segment produced in stage i, we check if the region corresponding to this segment is entirely contained in the query box. If it is, then report all the points stored in this segment. Otherwise, perform the algorithm recursively on the two child segments produced from this segment. The analysis on the original kd-tree can also be used to show the following theorem:

Theorem 1 ([59]). *Given a set of n points in d-dimensional space, there exists an implicit data structure that can answer orthogonal range reporting queries over this point set in $O(n^{1-1/d}+k)$ time, where k is the number of points reported. This data structure can be constructed in $O(n \lg n)$ time.*

Arroyuelo *et al.* [8] designed adaptive data structures for two-dimensional orthogonal range reporting, including an implicit version. Their data structures are adaptive in the sense that they take advantage of the inherent sortedness of given data to improve search efficiency. In their work, they say that a set of points form a monotonic chain if, when listed from left to right, the y-coordinates of these points are either monotonically increasing or monotonically decreasing; the sortedness of a given point set is then measured in terms of the minimum

number, m, of monotonic chains that the points in this set can be decomposed into. Their linear-space data structure can support two-dimensional orthogonal range counting in $O(m + \lg n + k)$ time. For any point set, m is bounded by $O(\sqrt{n})$ (this is an obvious consequence of the Erdős-Szekeres theorem), so the query performance matches that of the kd-tree in the worst case, and can be significantly better if m is small. Though it is NP-hard to compute m, there is an $O(n^3)$-time approximation algorithm that can achieve constant approximation ratio [38].

We briefly summarize the main idea of constructing the implicit version of this data structure. Observe that, if we decompose the point set into m monotonic chains and store each chain in a separate array, then we can perform binary search m times to answer a query. If we concatenate these arrays into a single array storing all point coordinates, we need encode the starting position of each sub-array, which requires $O(m \lg n) = O(\sqrt{n} \lg n)$ bits. This can be encoded using a standard technique: in each sub-array, we group points into pairs. If we swap the points in a pair, then we use this to encode a 1 bit; otherwise, a 0 bit is encoded. If we know whether the chain this pair is in is monotonically increasing or decreasing, then we can decode this bit by comparing this pair of coordinates. Extra care has to be taken to address the case in which chains contain odd numbers of points, so that the encoded information can be decoded correctly. Note that the query performance of this implicit structure is slightly worse than their linear-space structure; the techniques used in the latter to speed up query requires the storing of duplicate copies of some points which is not allowed in the design of implicit data structures. We summarize the space and time cost of their implicit data structure in the following theorem:

Theorem 2 ([8]). *Given a set of n points in two-dimensional space, there exists an implicit data structure that can answer orthogonal range reporting queries over this point set in $O(m \lg n + k)$ time, where k is the number of points reported, and $m = O(\sqrt{n})$ is the minimum number of monotonic chains that this set can be decomposed into. This data structure can be constructed in $O(n^3)$ time.*

2.2 Succinct Data Structures for Orthogonal Range Counting and Reporting

Another fundamental range query is *orthogonal range counting*. Here we focus on the two-dimensional case, as this is what the succinct data structures that we survey in this section are designed for. In the two-dimensional range counting problem, we are to preprocess a set, N, of n points in the plane, so that given an axis-aligned query rectangle, P, the number of points in $N \cap P$ can be computed efficiently. Among linear-space data structures for this problem, the structure of Chazelle [25] supports range counting in $O(\lg n)$ time. When point coordinates are integers, JáJá *et al.* [51] designed a linear-space structure that answers queries in $O(\lg n / \lg \lg n)$ time. This matches the lower bound on query time proved by Pătraşcu [63] under the cell probe model for data structures occupying $O(n \lg^{O(1)} n)$ words of space.

In a special case, the point sets are in rank space, i.e., they are on an $n \times n$ grid. The general orthogonal range counting and reporting problems can be reduced to this using a well-known technique [42], and Pătraşcu's lower bound mentioned in the previous paragraph also applies to this special case. Some range search structures [42, 25, 5] for the more general case were achieved by first considering rank space. Indeed, this reduction allowed Chazelle [25] to use operations under RAM to achieve the first linear-space solution that supports range counting in $O(\lg n)$ time, which is more space-efficient than the classic range tree [12] which uses $O(n \lg n)$ space to provide the same query support. In addition, the study of this problem is crucial to the design of several space-efficient text indexing structures [54, 14].

To describe succinct data structures designed for this problem, some background knowledge is required. In particular, there is a key structure which is also used in most other succinct data structures: bit vectors. Given a bit vector $B[1..n]$ storing 0 and 1 bits, the following three operations are considered:

- $\texttt{access}(B, i)$ which returns $B[i]$;
- $\texttt{rank}_\alpha(B, i)$ which returns the number of times the bit α occurs in $B[1..i]$, for $\alpha \in \{0, 1\}$;
- $\texttt{select}_\alpha(B, i)$ which returns the position of the ith occurrence of α in B, for $\alpha \in \{0, 1\}$.

Jacobson [50] considered this problem under the bit probe model; later Clark and Munro [27] showed how to represent a bit vector succinctly using $n + o(n)$ bits to support these three operations in $O(1)$ time under the word RAM model with word size of $O(\lg n)$ bits (this is the model that almost all succinct data structures assume; unless otherwise specified, succinct data structure results surveyed in the rest of this article assume this model). We refer to the work of Pătraşcu [64] for the most recent result on this extensively studied fundamental problem.

The definitions of `rank` and `select` operations on bit vectors can be generalized to a string $S[1..n]$ over alphabet $[\sigma]$[2], by letting α take any value in $[\sigma]$. The first succinct data structure designed for this problem is the wavelet tree of Grossi *et al.* [44]. In a wavelet tree representing string S, each node, v, represents a range of alphabet symbols $[a..b]$. Let S_v be the subsequence of S (not necessarily a contiguous subsequence) consisting of all the characters of S in $[a..b]$, and let $n_v = |S_v|$. Then a bit vector B_v of length n_v is constructed for the node v, in which $B_v[i] = 0$ iff $S_v[i] \in [a..\lfloor (a+b)/2 \rfloor]$. Thus the 0 bits correspond to the characters in the smaller half of the range of alphabet symbols represented by v, and 1 bits correspond to the greater half. Node v has two children v_1 and v_2, corresponding to the two subsequences of S_v that consist of all the characters of S_v in $[a..\lfloor (a+b)/2 \rfloor]$ and $[\lfloor (a+b)/2 \rfloor + 1..b]$, respectively. In other words, v_1 and v_2 correspond to the 0 and 1 bits stored in B_v, respectively. Bit vectors for these children and their descendants are defined in a recursive manner. To construct a wavelet tree for S, the root represents the range $[1..\sigma]$, and nodes at

[2] $[\sigma]$ denotes the set $\{1, 2, \ldots, \sigma\}$.

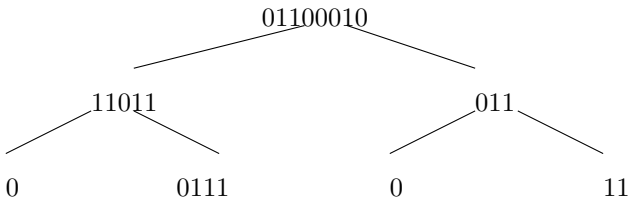

Fig. 1. A wavelet tree constructed for the string 35841484 over an alphabet of size 8. This is also a wavelet tree constructed for the following point set on an 8 by 8 grid: $\{1,3\},\{2,5\},\{3,8\},\{4,4\},\{5,1\},\{6,4\},\{7,8\},\{8,4\}$.

each successive level are created recursively as mentioned previously. Each leaf represents a range of size 2. Thus the wavelet tree has $\lceil \lg \sigma \rceil$ levels. The wavelet tree is not stored explicitly. Instead, for each level, we visit the nodes from left to right, and concatenate the bit vectors created for these nodes. The concatenated bit vector is represented using a bit vector structure supporting `rank` and `select`. Thus the wavelet tree is stored as $\lceil \lg \sigma \rceil$ bit vectors of the same length n, which occupy $(n + o(n))\lceil \lg \sigma \rceil$ bits in total. See Figure 1 for an example.

No additional information is required to navigate in the wavelet tree; during a top-down traversal, the `rank` operation over each length-n bit vector is sufficient to identify the starting and ending positions of the bit vector B_v which corresponds to a node v. By taking advantage of this, one can design algorithms that support `access`, `rank` and `select` operations over S. These algorithms perform a constant number of rank/select operations over the bit vector constructed for each level of the tree, and hence their running time is $O(\lg \sigma)$.

Mäkinen *et al.* [54] showed that a wavelet tree can be used to support orthogonal range counting and reporting for n points on an $n \times n$ grid, when the points have distinct x-coordinates (this restriction can be removed through a reduction [14]). To construct a wavelet tree for such a point set, we conceptually treat it as a string S of length n over alphabet $[n]$, in which $S[i]$ stores the y-coordinate of the point whose x-coordinate is i. Thus Figure 1 can also be considered as a wavelet tree constructed for 8 points on an 8 by 8 grid, and this point set is given in the caption of the figure. To show how to support range search, take range counting for example. Suppose that the given query rectangle P is $[x_1..x_2] \times [y_1..y_2]$. To count the number of points in P, we perform a top-down traversal of the wavelet tree, and use a variable c (initially 0) to record the number of points that we have identified to be inside P during this traversal. At each node v, if the range $[a, b]$ represented by v is a subset of $[y_1..y_2]$, then the entries of S_v that correspond to points inside P form a contiguous substring of S_v. The starting and ending positions of this substring in S_v can be computed by performing rank queries over bit vectors, during the top-down traversal. With these positions, we can increased the value of c by the size of this substring. If range $[a, b]$ intersects $[y_1..y_2]$ but is not its subset, then we visit the children of v whose ranges intersect $[y_1..y_2]$ and perform the above process recursively. This algorithms visits at most two nodes and performs a constant number of

rank/select operations over bit vectors at each level of the wavelet tree, and thus range counting can be supported in $O(\lg n)$ time. Range reporting can be supported in a similar manner, in $O((k+1)\lg n)$ time.

There is a similarity between wavelet trees and Chazelle [25]'s data structure for orthogonal range search in rank space. During the construction of Chazelle's structure for n points on an $n \times n$ grid, a set of $\lceil \lg n \rceil$ conceptual bit vectors of length n is also defined. Unlike bit vectors in wavelet trees, these vectors encode the process of performing mergesort on y-coordinates when points are pre-sorted by x-coordinate. Bits in these conceptual vectors are then organized in a set of non-succinct data structures to facilitate queries; these non-succinct structures could be replaced by succinct bit vectors designed after Chazelle's work to reduce space cost. Similar algorithms can be performed on wavelet trees and Chazelle's structure to support range search. The exact content of these bit vectors are however not the same. The underlaying tree structures are also different in these two structures: The bit vectors corresponding to the nodes at the same level of a wavelet tree may be of different lengths, while this is not the case in Chazelle's structure due to the nature of mergesort. The difference in layout allows wavelet trees to directly encode strings succinctly to support rank/select when the string length is much larger than the alphabet size.

To speed up rank/select operations on strings, Ferragina *et al.* [37] designed a data structure called generalized wavelet tree. The main difference between this structure and the original wavelet tree is that each node, v, in the generalized wavelet tree has $t = \lg^{\epsilon} n$ children for a positive constant ϵ less than 1. Instead of constructing a bit vector for v, a string over alphabet $[t]$ is constructed, and each character in $[t]$ corresponds to a subrange represented by a child of v. They then designed a succinct representation of strings over an alphabet of size t that can support `access`, `rank` and `select` in constant time. Thus a generalized wavelet tree has $O((\lg \sigma / \lg \lg n) + 1)$ levels, and operations at each level can be supported in constant time. This can be used to support `access`, `rank` and `select` in $O((\lg \sigma / \lg \lg n) + 1)$ time. Note that the idea of increasing the fanout of a tree structure to speed up query time by an $O(\lg \lg n)$ factor is a standard strategy under the word RAM model, and was also used by JáJá *et al.* [51] for range search. New auxiliary structures, however, had to be designed to guarantee that the resulting data structure is still succinct.

To further use a generalized wavelet tree to provide better support for range search, we need a succinct representation of n points on a narrow grid, i.e., an $n \times O(\lg^{\epsilon} n)$ grid, that supports range counting and reporting in constant time. The simultaneous work of two groups of researchers, Bose *et al.* [14] and Yu *et al.* [70] designed such data structures. These data structures are essentially equivalent and they provide the same support for queries. The following theorem summarizes the support for range search provided by a generalized wavelet tree:

Theorem 3 ([14, 70]). *A set of n points on an $n \times n$ grid can be represented using $n \lg n + o(n \lg n)$ bits to support orthogonal range counting in $O(\lg n / \lg \lg n)$ time, and orthogonal range reporting in $O((k+1) \lg n / \lg \lg n)$ time, where k is*

the number of points reported. This data structure can be constructed in $O(n \lg n)$ time.

Bose *et al.* [14] further used this to improve previous results on designing succinct data structures for integer sequences and text indexes. One text index they designed is an improvement over previous results on *position-restricted text search*, which looks for the occurrences of a given query string within a given substring of the text.

2.3 Other Range Queries

2D 3-sided orthogonal range reporting. In the *2D 3-sided orthogonal range reporting* problem, we are given a set, P, of n points in the plane, and the query is to report the points in 3-sided query ranges of the form $[x_1, x_2] \times (-\infty, y_2]$. The classic priority search tree of McCreight [58] can answer such a query in $O(\lg n + k)$ time, where k is the number of points to be reported. Each node in a priority search tree stores one point from P; the entire tree can be viewed as a min-heap over the y-coordinates of points in P, and a binary search tree over x-coordinates. To facilitate the navigation in the tree by x-coordinate, each node also stores the median value of the x-coordinates of the points stored in its descendants, as this value is used to distribute points to subtrees rooted at the left or right child of this node. Brönnimann *et al.* [15] designed a variant of priority search trees to avoid storing this extra median x-coordinate in each node, and this variant can be made implicit. To lay out points from P in an array, they store the point, p_0, with the smallest y-coordinate among points in P at the head of the array, and the point, p_1, with the median x-coordinate among points in $P \setminus \{p_0\}$ in the next entry. Then they divided all the points in $P \setminus \{p_0, p_1\}$ into two halves by this median x-coordinate, and recurse in these two subarrays. The query algorithm for the original priority search tree can be easily adapted to this implicit variant to answer a 3-sided orthogonal range reporting query in $O(\lg n + k)$ time.

If we would like to use this implicit priority search tree to support range reporting for both query ranges of the form $[x_1, x_2] \times (-\infty, y_2]$ and ranges of the form $[x_1, x_2] \times [y_1, \infty)$, then two trees have to be constructed. To avoid the duplication of point coordinates, De *et al.* [29] designed a min-max priority search tree which stores one copy of point coordinates to answer queries of both forms. They also showed that their structure can be made implicit, answering queries in $O(\lg n + k)$ time.

Simplex, halfspace and ball range reporting. In the *simplex range reporting* problem, the query region is a simplex, while in *halfspace range reporting*, it is a halfspace. For these two problems, Brönnimann *et al.* [15] showed how to lay out, in an array of coordinates, the variants of partition trees proposed by Matousek [55, 56]. In d-dimensional space, one variant of their implicit partition tree can answer simplex range reporting in $O(n^{1-1/d+\epsilon} + k)$ time, where k is the number of points to be reported and ϵ is a positive constant that can be

arbitrarily small. The query performance is very close to $O(n^{1-1/d}+k)$ which is believed to be the optimal query time using linear-space structures [57, 20]. They designed another implicit partition tree that can support halfspace range reporting in $O(n^{1-1/\lfloor d/2\rfloor+\epsilon}+k)$ time. For discussions of better trade-offs for halfspace range reporting in different cases using linear-space structures, see [20]. Both implicit data structure can be constructed in $O(n \lg n)$ expected time.

In the unpublished full version of [15], Brönnimann *et al.* further showed how to speed up the support for simplex range reporting at the cost of increased preprocessing time. More precisely, the n^{ϵ} factor in the query time can be replaced by polylog n, and the preprocessing time is increased to $O(n \text{ polylog } n)$. They also stated that that their implicit data structures can support *simplex range counting*, i.e., counting the number of points in the query simplex, without the $O(k)$ additive term (this applies to both trade-offs).

The d-dimensional *ball range reporting* problem, in which the query region is a ball, can be reduced to $(d+1)$-dimensional halfspace range reporting by the standard lifting transformation that maps points in $\mathbb{R}^d$ to points on the unit paraboloid in $\mathbb{R}^{d+1}$. Thus it follows from [15] that there is an implicit data structure supporting ball range reporting in $\mathbb{R}^d$ in $O(n^{1-1/\lfloor (d+1)/2\rfloor} \text{ polylog } n + k)$ time.

3 Point Location

In the *planar point location query* problem, we preprocess a planar subdivision with n vertices, so that the face containing a given query point can be located quickly. In a special case of this problem, the planar subdivision is a planar triangulation, i.e., a planar subdivision in which each face is a triangle. When the data set is static, the general problem can be reduced to this special case by triangulating all the faces of the planar subdivision. As mentioned in Section 1, a number of linear-space classic solutions were proposed to support point location in the optimal $O(\lg n)$ time based on different techniques [53, 52, 32, 28, 68].

Succinct data structures that represent planar triangulations and planar maps [17, 18, 9] using $O(n)$ bits support queries regarding connectivity information such as adjacency test, but they cannot be directly combined with point location structures without using additional space of $O(n)$ words or $O(n \lg n)$ bits. Thus, to design space-efficient solutions to point location, Bose *et al.* [13] proposed to design data structures called *succinct geometric indexes*. These data structures occupy $o(n)$ bits, excluding the space needed for the input array, which stores the coordinates of the n vertices of the subdivision. The n vertices may be permuted in the array. Hence $o(n)$ bits of space is the only extra storage required to support queries, in addition to the storage cost required of the given data.

They first designed a succinct geometric index that supports point location in planar triangulations. To construct this index, they use graph separators twice to decompose the given planar triangulation T. More precisely, they first, in the top-level partition, apply the t-separator theorem [3] on the dual graph of T, choosing $t = \lg^3 f/f$, where f is the number of faces of T. By doing so, they partition

T into a separator consisting of $O(n/\lg^{3/2} n)$ faces and $O(n/\lg^{3/2} n)$ subgraphs called *regions*; each region consists of $O(\lg^3 n)$ vertices and corresponds to a connected component of the dual graph after removing the separator. They further, in the bottom-level partition, apply the separator theorem on each region to create subregions consisting of $O(\lg n)$ vertices each. The reason why they perform two levels of partition is that they intend to create one point location structure for the top-level partition (any linear-space solution supporting query in logarithmic time will be sufficient) and a point location structure for each region to answer queries. The structure for the top-level partition is constructed by first triangulating the graph consisting of the outer face and the separator for the top-level partition, and then building a structure to answer point location. This structure will either report that the query point is in a separator face and thus terminate, or locate the region containing the query point. Since the size of this graph is $O(n/\lg^{3/2} n)$, $O(n/\lg^{1/2} n) = o(n)$ bits would be sufficient. Then, they construct a similar point location structure for each region to tell whether the given query point is in a separator face of this region, or in a particular subregion. They hope that, since there are $O(\lg^3 n)$ vertices in each region, $O(\lg\lg n)$ bits would be sufficient to identify each vertex and to encode each pointer in these point location structures. This guarantees that all these point location structures occupy $o(n)$ bits in total. Finally, if the point is in a subregion, they check each face of the subregion to compute the result. If this idea works, then the query time would be $O(\lg n)$.

The main challenge for this to work is that after applying the separator theorem, each vertex could appear in multiple regions and/or subregions, so that we can not simply assign an $O(\lg\lg n)$-bit identifier for each vertex when constructing the point location structure of a region. To overcome this difficulty, they use the following strategy to assign identifiers at three different levels for each vertex. First, for each subregion that a vertex is in, a *subregion-label* of $O(\lg\lg n)$ bits is assigned by applying the approach of Denny and Sohler [30] that permutes the vertex set to encode the graph structure of a planar triangulation. For each subregion, the rank of a vertex in the permuted sequence for this subregion becomes its subregion-label for its occurrence in this subregion. Next, for each region, a *region-label* is assigned to each of its vertices as follows: Visit the subregions in this region in an arbitrary order, and for each subregion visited, visit its vertices by subregion-label. During this traversal, they incrementally assign region-labels (starting from 1) for each vertex in the order in which it is first visited; thus, even if a vertex appears in multiple subregions, it has a distinct region-label. Finally, each vertex is assigned a distinct *graph-label* over the entire triangulation. Graph-labels are constructed from region-labels in a way similar to the way in which region-labels are constructed from subregion-labels. Point coordinates are then stored in an array, indexed by graph-label. Given a subregion (or region) and a subregion-label (or region-label) of a vertex, the graph-label of this vertex can be computed in constant time using succinct sparse bit vectors [66] and other data structures; readers with background in succinct data structures can attempt to design an $o(n)$-bit structure achieving

this on their own. With this, a vertex in the point location structure constructed for a region can be identified using its $O(\lg \lg n)$-bit region-label, to guarantee that the succinct index constructed occupies $o(n)$ bits only. Graph-labels are also used as point coordinates in the point location structure constructed for the top-level partition, so that no point coordinates are duplicated.

To further construct a succinct index for a general planar subdivision, they partition each large face into smaller faces and assign identifiers at three different levels to each face. Their main result can then be summarized in the following theorem:

Theorem 4 ([13]). *Given a planar subdivision of n vertices, there exists an $o(n)$-bit succinct geometric index that supports point location in $O(\lg n)$ time. This index can be constructed in $O(n)$ time.*

Three variants of the succinct geometric index for planar triangulations were also designed, to match the query efficiency of data structures with improved query time under various assumptions. The first index supports point location using $\lg n + 2\sqrt{\lg n} + O(\lg^{1/4} n)$ point-line comparisons, which matches the query efficiency of the linear-space structure of Seidel and Adamy [69]. The second addresses the case in which the query distribution is known. In this case, let p_i denote the probability of a query point residing in face p_i, and the entropy of the distribution is $H = \sum_{i=1}^{f}(p_i \lg \frac{1}{p_i})$, where f is the number of faces. They designed a succinct index supporting point location in $O(H+1)$ expected time, which matches the query time of the linear-space structure of Iacono [49]. The third variant assumes that the point coordinates are integers bounded by $U \leq 2^w$, where w is the number of bits in a word, and it supports queries in $O(\min\{\lg n/\lg \lg n, \sqrt{\lg U/\lg \lg U}\})$ time. This matches the query efficiency of the linear-space structure of Chan and Pătraşcu [23][3]. These three succinct geometric indexes can be constructed in linear time.

The succinct geometric index can be further used to design implicit data structures for point location. The main idea is to adopt the standard approach of encoding one bit of information by swapping one pair of points, in order to encode the $o(n)$-bit geometric index in the permuted sequence of point coordinates. This requires several modifications to the succinct index, including labeling schemes. The support for queries becomes slower, as $O(\lg n)$ time is required to decode one word of information. The implicit structure is summarized in the following theorem:

Theorem 5 ([13]). *Given a planar subdivision of n vertices, there exists an implicit data structure that supports point location in $O(\lg^2 n)$ time. This data structure can be constructed in $O(n)$ time.*

[3] The query time of this variant of succinct geometric index was stated as $O(\min\{\lg n/\lg \lg n, \sqrt{\lg U}\})$ in [13], and this was because a preliminary version of the structure in [23] was used to prove the query time. It is trivial to apply the main result of [23] to achieve the query time stated here.

Bose *et al.* [13] also showed how to design succinct geometric indexes and implicit data structures for a related problem called *vertical ray shooting*. In this problem, a set of n disjoint line segments is given, and the query returns the line segment immediately below a given query point. The succinct index and implicit data structure support this query in $O(\lg n)$ and $O(\lg^2 n)$ time, respectively. They can be constructed in $O(n \lg n)$ time.

He *et al.* [48] considered the problem of maintaining a dynamic planar subdivision to support point location. The update operations they consider include

- Inserting a new vertex v by replacing the edge between two existing vertices u_1 and u_2 with two new edges (v, u_1) and (v, u_2);
- Deleting a node of degree 2 by replacing its two incident edges with a single edge connecting its two neighbors if they were not adjacent;
- Inserting an edge between two existing vertices across a face whose boundary contains these two vertices, preserving planarity;
- Deleting an edge between two vertices of degrees greater than 2.

To design a succinct geometric index for this problem, they designed a succinct version of the P-tree proposed by Aleksandrov and Djidjev [4] which maintains the partition of a planar subdivision with constant face size under the same update operations; these operations can be used to transform any connected planar subdivision to any other connected planar subdivision. He *et al.* then applied two-level partitioning on the given subdivision using a succinct P-tree. Combined with linear-space data structures for dynamic point location [26, 7], they designed succinct geometric indexes to match the query times of previous results, though the update times are slightly slower:

Theorem 6 ([48]). *Let G be a planar subdivision of n vertices in which faces are of constant size and vertices have coordinates that can be encoded in $M = \Theta(\lg n)$ bits. Under the word RAM model with $\Theta(\lg n)$-bit word size, there exists a data structure that can represent G in $nM + o(n)$ bits to supports, for any positive constant ϵ,*

- *point location in $O(\lg n)$ time and updates in $O(\lg^{3+\epsilon} n)$ amortized time[4];*
- *point location in $O(\lg^2 n)$ time and updates in $O(\lg^{2+\epsilon} n)$ worst-case time.*

4 Nearest Neighbor Search

Given a set, N, of n points in the plane, the two-dimensional nearest neighbor query returns the point that is closest (in terms of Euclidean distance) to a given query point. It is well-known that this problem can be reduced to planar point location: Construct the Voronoi diagram. Then the answer is the point whose Voronoi cell contains the query point. This however cannot be used directly to design implicit structures for nearest neighbor search, as the reduction requires $O(n)$ extra space.

[4] This tradeoff is from the unpublished full version of He *et al.* [48].

To design an implicit data structure for two-dimensional nearest neighbor search, Brönnimann *et al.* [15] applied a separator theorem (such as the t-separator theorem [3][5]) on T to partition the Voronoi diagram into a separator of $O(n/\lg n)$ cells and clusters of $O(\lg^2 n)$ cells each. Thus a point location structure of $O(n)$ bits can be constructed to tell which separator cell or cluster contains the query point. By choosing an appropriate parameter for the separator, this point location structure can be represented using at most $n/2$ bits, which can be encoded using the standard techniques of encoding bits by swapping pairs of points. This yields an implicit data structure supporting nearest neighbor search in $O(\lg^2 n)$ time.

Brönnimann *et al.* [15] showed that a similar approach can be used to construct implicit data structures for a set of n halfspaces in 3-dimensional space in $O(n \lg n)$ time, which answer *ray shooting* and *linear programming* queries in the intersection of halfspaces in $O(\lg^2 n)$ time. These two queries are defined as follows: A ray shooting query determines the first halfspace hit by a query ray. In a linear programming query, linear constraints are represented as halfspaces, and the query returns a point that minimizes the value of a query linear function while satisfying all these constraints. Note that the linear-space structure of Dobkin and Kirkpatrick [31] can answer both queries in $O(\lg n)$ time in $\mathbb{R}^3$.

To further improve the query efficiency for nearest neighbor search, Chan and Chen [21] designed a recursive structure. They organize points in an array recursively in a generalization of the van Emde Boas layout [34]: Apply the separator theorem to partition the Voronoi diagram into a separator consisting of $O(\sqrt{bn})$ cells and $O(b)$ clusters each consisting of $O(n/b)$ cells; the parameter b is to be determined by the recursive formula for the query time. According to this decomposition, they reorder the array of points, so that points corresponding to the separator are stored in the first segment of the array, and points corresponding to each cluster are stored in a subsequent segment. They then apply this strategy recursively to the separator as well as each cluster, reordering points in a separator or cluster in each recursion.

Their query algorithm over this structure is also recursive: First determine the cluster containing the cell whose corresponding point is nearest to the given query point among all the points whose cells are not in the separator. Then, perform this algorithm recursively in both this cluster and the separator, to find two points that are candidates of the nearest neighbor. Between these two points, the one that is closer to the query point is the answer. The challenging part is how to locate this cluster, and they proved geometric properties between the Voronoi diagrams of a point set and a subset of it, and designed additional recursive structures. As with the van Emde Boas tree, the query time of their structure is also determined by a recursive function. With the choice of parameter $b = n^{1/3}$, the main recurrences in the critical cases are of the form $Q(n) = 2Q(n^{2/3}) + O(\lg^c n)$, where c is a constant number depending on the particular

[5] They actually applied the separator theorem of Frederickson [41] which requires $O(n \lg n)$ time. However, the t-separator theorem would work as well, and the advantage is that a t-separator can be computed in $O(n)$ time.

case. Thus the query time can be shown to be $O(\lg^{\log_{3/2} 2} n \lg\lg n) = O(\lg^{1.71} n)$, and their main result can be summarized in the following theorem:

Theorem 7 ([21]). *Given a set of n points in two-dimensional space, there exists an implicit data structure that supports nearest neighbor search in $O(\lg^{1.71} n)$ time. This data structure can be constructed in $O(n \lg n)$ time.*

Chan [19] considered the approximate nearest neighbor search problem in constant-dimensional space. Here the word "approximate" means that for any fixed positive constant ϵ, the distance between the query point q and the point returned as the answer is guaranteed to be within a factor of $1 + \epsilon$ from the minimum distance to q. When point coordinates are integers, they designed a simple strategy of laying out coordinates in an array based on shifting and sorting. Surprisingly, it can be proved that this guarantees an approximation ratio of $1 + \epsilon$. As random choices are made by the preprocessing algorithm, its query time is expected.

Theorem 8 ([19]). *Given a set of n points with integer coordinates in constant-dimensional space, there exists an implicit data structure that supports approximate nearest neighbor search in $O(\lg n)$ expected time. This data structure can be constructed in $O(n \lg n)$ time.*

We finally mention that when polylogarithmic query time is not required, there is an implicit data structure for points in $\mathbb{R}^d$ supporting nearest neighbor search in $O(n^{1-1/\lfloor (d+1)/2 \rfloor} \text{ polylog } n)$ time. This again uses the implicit halfspace range reporting structure [15] summarized in Section 2.3, via lifting transformation. For ray shooting and linear programming queries in intersections of halfspaces in $\mathbb{R}^d$ where $d \geq 4$, Brönnimann *et al.* [15] designed an implicit structure that can answer these queries in $O(n^{1-1/\lfloor d/2 \rfloor} \text{ polylog } n)$ time, which is also based on their structure for halfspace range search.

5 Conclusion

In this article, we have surveyed previous results on designing succinct and implicit data structures for geometric query problems. Research in these directions developed new algorithmic approaches for computational geometry, succinct data structures and implicit data structures. As more and more applications process large geometric data structures, we also expect that such research will have great impact in the engineering of modern software systems.

There has been some recent development in the design of solutions to geometric query problems that make use of succinct data structures. Unlike the work surveyed in this article that focus on designing succinct solutions, the key strategy is to use succinct data structures to either achieve improvement upon previous results in terms of running time, or to reduce space usage by non-constant factors. Note that some of Chazelle [25]'s structures already used tricks under word RAM which happen to be useful for succinct data structures as well, though these tricks do not include techniques particularly developed

later for succinct data structures. Among the works that focus on using succinct structures to improve running time, the work of He and Munro [45] on dynamic two-dimensional orthogonal range counting problem is perhaps the most relevant to this survey. In this problem, in addition to supporting queries, the insertion/deletion of a point into/from the given point set is also considered. The structure of He and Munro occupies $O(n)$ words of space, answers queries in $O((\lg n/\lg\lg n)^2)$ time, and performs updates in $O((\lg n/\lg\lg n)^2)$ amortized time. This is currently the most efficient linear-space solution to this problem. Succinct data structures are also extensively used in the design of data structures occupying linear or near-linear space for dynamic range median [46], range majority [33], path queries on weighted trees [47], orthogonal range maxima [36] and adaptive and approximate range counting [24]. The use of succinct data structure techniques is crucial to achieving these results.

We end our article by giving several important open problems in the design of succinct and implicit geometric data structures:

- Can implicit partitions trees be further improved to match the performance of Chan's optimal partition trees [20]?
- Can the implicit data structures that achieve polylogarithmic query times, including the structures for 2D point location and nearest neighbor search, be further improved? The $O(\lg^{1.71} n)$ and $O(\lg^2 n)$ query times are slower than logarithmic query times of linear-space data structures for the same problems.
- The implicit geometric data structures that we have surveyed are all designed for static data sets, and the only dynamic succinct geometric structure is the structure of He *et al.* [48] for point location. Brönnimann *et al.* [15] considered semi-dynamization of their implicit geometric data structures which only allows the insertion of points without supporting deletions. Thus designing fully dynamic versions of most of the succinct and implicit structures surveyed here remains open. Many dynamic succinct and implicit data structures have been already designed for bit vectors [65], strings [43, 62], trees [67], graphs [16] and partial search [60, 39], and thus we expect that progress can be made regarding this open problem.
- Even though a number of data structures have been presented here, there are many other geometric query problems that do not have succinct or implicit solutions. In fact, for each geometric query problem that has a linear-space solution, we can ask the following questions: Can we construct a succinct or implicit solution to this query problem? If the answer is negative, how to give a related lower bound proof?

References

1. Afshani, P., Arge, L., Larsen, K.D.: Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In: Proceedings of the 26th Annual ACM Symposium on Computational Geometry. pp. 240–246 (2010)

2. Afshani, P., Arge, L., Larsen, K.G.: Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In: Proceedings of the 28th Annual ACM Symposium on Computational Geometry. pp. 323–332 (2012)
3. Aleksandrov, L., Djidjev, H.: Linear algorithms for partitioning embedded graphs of bounded genus. SIAM Journal on Discrete Mathematics 9(1), 129–150 (1996)
4. Aleksandrov, L., Djidjev, H.: A dynamic algorithm for maintaining graph partitions. In: 7th Scandinavian Workshop on Algorithm Theory. pp. 71–82 (2000)
5. Alstrup, S., Brodal, G.S., Rauhe, T.: New data structures for orthogonal range searching. In: Proceedings of the 41st IEEE Symposium on Foundations of Computer Science. pp. 198–207 (2000)
6. Alt, H., Mehlhorn, K., Munro, J.I.: Partial match retrieval in implicit data structures. Information Processing Letters 19(2), 61–65 (1984)
7. Arge, L., Brodal, G.S., Georgiadis, L.: Improved dynamic planar point location. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science. pp. 305–314 (2006)
8. Arroyuelo, D., Claude, F., Dorrigiv, R., Durocher, S., He, M., López-Ortiz, A., Munro, J.I., Nicholson, P.K., Salinger, A., Skala, M.: Untangled monotonic chains and adaptive range search. Theoretical Computer Science 412(32), 4200–4211 (2011)
9. Barbay, J., Castelli Aleardi, L., He, M., Munro, J.I.: Succinct representation of labeled graphs. Algorithmica 62(1-2), 224–257 (2012)
10. Barbay, J., He, M., Munro, J.I., Satti, S.R.: Succinct indexes for strings, binary relations and multilabeled trees. ACM Transactions on Algorithms 7(4), 52:1–52:27 (2011)
11. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM 18(9), 509–517 (1975)
12. Bentley, J.L.: Decomposable searching problems. Information Processing Letters 8(5), 244–251 (1979)
13. Bose, P., Chen, E.Y., He, M., Maheshwari, A., Morin, P.: Succinct geometric indexes supporting point location queries. ACM Transactions on Algorithms 8(2), 10:1–10:26 (2012)
14. Bose, P., He, M., Maheshwari, A., Morin, P.: Succinct orthogonal range search structures on a grid with applications to text indexing. In: Proceedings of the 11th International Symposium on Algorithms and Data Structures. pp. 98–109 (2009)
15. Brönnimann, H., Chan, T.M., Chen, E.Y.: Towards in-place geometric algorithms and data structures. In: Symposium on Computational Geometry. pp. 239–246 (2004)
16. Castelli Aleardi, L., Devillers, O., Schaeffer, G.: Dynamic updates of succinct triangulations. In: Proceedings of the 17th Canadian Conference on Computational Geometry. pp. 134–137 (2005)
17. Castelli Aleardi, L., Devillers, O., Schaeffer, G.: Succinct representation of triangulations with a boundary. In: Proceedings of the 9th Workshop on Algorithms and Data Structures. pp. 134–145 (2005)
18. Castelli Aleardi, L., Devillers, O., Schaeffer, G.: Succinct representations of planar maps. Theoretical Computer Science 408(2-3), 174–187 (2008)
19. Chan, T.M.: A minimalist's implementation of an approximate nearest neighbor algorithm in fixed dimensions (2006), unpublished manuscript
20. Chan, T.M.: Optimal partition trees. Discrete & Computational Geometry 47(4), 661–690 (2012)

21. Chan, T.M., Chen, E.Y.: In-place 2-d nearest neighbor search. In: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 904–911 (2008)
22. Chan, T.M., Larsen, K.G., Patrascu, M.: Orthogonal range searching on the RAM, revisited. In: Proceedings of the 27th ACM Symposium on Computational Geometry. pp. 1–10 (2011)
23. Chan, T.M., Pătraşcu, M.: Transdichotomous results in computational geometry, I: Point location in sublogarithmic time. SIAM Journal on Computing 39(2), 703–729 (2009)
24. Chan, T.M., Wilkinson, B.T.: Adaptive and approximate orthogonal range counting. In: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 241–251 (2013)
25. Chazelle, B.: A functional approach to data structures and its use in multidimensional searching. SIAM Journal on Computing 17(3), 427–462 (1988)
26. Cheng, S.W., Janardan, R.: New results on dynamic planar point location. SIAM Journal on Computing 21(5), 972–999 (1992)
27. Clark, D.R., Munro, J.I.: Efficient suffix trees on secondary storage. In: Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 383–391 (1996)
28. Cole, R.: Searching and storing similar lists. Journal of Algorithms 7(2), 202–220 (1986)
29. De, M., Maheshwari, A., Nandy, S.C., Smid, M.H.M.: An in-place min-max priority search tree. Computational Geometry: Theory and Applications 46(3), 310–327 (2013)
30. Denny, M., Sohler, C.: Encoding a triangulation as a permutation of its point set. In: Proceedings of the 9th Canadian Conference on Computational Geometry (1997)
31. Dobkin, D.P., Kirkpatrick, D.G.: Determining the separation of preprocessed polyhedra - a unified approach. In: Proceedings of the 17th International Colloquium on Automata, Languages and Programming. pp. 400–413 (1990)
32. Edelsbrunner, H., Guibas, L.J., Stolfi, J.: Optimal point location in a monotone subdivision. SIAM Journal on Computing 15(2), 317–340 (1986)
33. Elmasry, A., He, M., Munro, J.I., Nicholson, P.K.: Dynamic range majority data structures. In: Proceedings of the 22nd International Symposium on Algorithms and Computation. pp. 150–159 (2011)
34. Emde Boas, P.: Preserving order in a forest in less than logarithmic time. In: Proceedings of the 16th Annual Symposium on Foundations of Computer Science. pp. 75–84 (1975)
35. Farzan, A., Munro, J.I.: Succinct representations of arbitrary graphs. In: Proceedings of the 16th Annual European Symposium on Algorithms. pp. 393–404 (2008)
36. Farzan, A., Munro, J.I., Raman, R.: Succinct indices for range queries with applications to orthogonal range maxima. In: Proceedings of the 39th International Colloquium on Automata, Languages and Programming. pp. 327–338 (2012)
37. Ferragina, P., Manzini, G., Mäkinen, V., Navarro, G.: Compressed representations of sequences and full-text indexes. ACM Transactions on Algorithms 3(2), 20:1–20:24 (2007)
38. Fomin, F.V., Kratsch, D., Novelli, J.C.: Approximating minimum cocolorings. Information Processing Letters 84(5), 285–290 (2002)
39. Franceschini, G., Grossi, R.: Optimal worst-case operations for implicit cache-oblivious search trees. In: Proceedings of the 8th International Workshop on Algorithms and Data Structures. pp. 114–126 (2003)

40. Frederickson, G.N.: Implicit data structures for the dictionary problem. Journal of the ACM 30(1), 80–94 (1983)
41. Frederickson, G.N.: Fast algorithms for shortest paths in planar graphs, with applications. SIAM Journal on Computing 16(6), 1004–1022 (1987)
42. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proceedings of the 16th Annual ACM Symposium on Theory of Computing. pp. 135–143 (1984)
43. González, R., Navarro, G.: Rank/select on dynamic compressed sequences and applications. Theoretical Computer Science 410(43), 4414–4422 (2009)
44. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 841–850 (2003)
45. He, M., Munro, J.I.: Space efficient data structures for dynamic orthogonal range counting. In: Proceedings of the 12th International Symposium on Algorithms and Data Structures. pp. 500–511 (2011)
46. He, M., Munro, J.I., Nicholson, P.K.: Dynamic range selection in linear space. In: Proceedings of the 22nd International Symposium on Algorithms and Computation. pp. 160–169 (2011)
47. He, M., Munro, J.I., Zhou, G.: Succinct data structures for path queries. In: Proceedings of the 20th Annual European Symposium on Algorithms. pp. 575–586 (2012)
48. He, M., Nicholson, P.K., Zeh, N.: A space-efficient framework for dynamic point location. In: Proceedings of the 23rd International Symposium on Algorithms and Computation. pp. 548–557 (2012)
49. Iacono, J.: Expected asymptotically optimal planar point location. Computational Geometry 29(1), 19–22 (2004)
50. Jacobson, G.: Space-efficient static trees and graphs. In: Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science. pp. 549–554 (1989)
51. JáJá, J., Mortensen, C.W., Shi, Q.: Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In: Proceedings of the 15th International Symposium on Algorithms and Computation. pp. 558–568 (2004)
52. Kirkpatrick, D.G.: Optimal search in planar subdivisions. SIAM Journal on Computing 12(1), 28–35 (1983)
53. Lipton, R.J., Tarjan, R.E.: Application of a planar separator theorem. In: Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science. pp. 162–170 (1977)
54. Mäkinen, V., Navarro, G.: Rank and select revisited and extended. Theoretical Computer Science 387(3), 332–347 (2007)
55. Matousek, J.: Efficient partition trees. Discrete & Computational Geometry 8, 315–334 (1992)
56. Matousek, J.: Reporting points in halfspaces. Computational Geometry: Theory and Applications 2, 169–186 (1992)
57. Matousek, J.: Range searching with efficient hiearchical cutting. Discrete & Computational Geometry 10, 157–182 (1993)
58. McCreight, E.M.: Priority search trees. SIAM Journal on Computing 14(2), 257–276 (1985)
59. Munro, J.I.: A multikey search problem. In: Proceedings of the 17th Allerton Conference on Communication, Control and Computing. pp. 241–244 (1979)
60. Munro, J.I.: An implicit data structure supporting insertion, deletion, and search in $O(\log^2 n)$ time. Journal of Computer and System Sciences 33(1), 66–74 (1986)

61. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM Journal on Computing 31(3), 762–776 (2001)
62. Navarro, G., Nekrich, Y.: Optimal dynamic sequence representations. In: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 865–876 (2013)
63. Pătraşcu, M.: Lower bounds for 2-dimensional range counting. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing. pp. 40–46 (2007)
64. Pătraşcu, M.: Succincter. In: Proceedings of 49th IEEE Annual Symposium on Foundations of Computer Science. pp. 305–313 (2008)
65. Raman, R., Raman, V., Rao, S.S.: Succinct dynamic data structures. In: Proceedings of the 7th International Workshop on Algorithms and Data Structures. pp. 426–437 (2001)
66. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Transactions on Algorithms 3(4), 43 (2007)
67. Sadakane, K., Navarro, G.: Fully-functional succinct trees. In: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 134–149 (2010)
68. Sarnak, N., Tarjan, R.E.: Planar point location using persistent search trees. Communications of the ACM 29(7), 669–679 (1986)
69. Seidel, R., Adamy, U.: On the exact worst case query complexity of planar point location. Journal of Algorithms 37(1), 189–217 (2000)
70. Yu, C.C., Hon, W.K., Wang, B.F.: Improved data structures for the orthogonal range successor problem. Computational Geometry: Theory and Applications 44(3), 148–159 (2011)