On Succinct and Implicit Data Structures for Computational Geometry

Haomin Li,
Department of Computer Science
University of Waterloo
h439li@uwaterloo.ca

Xingyu Liu,
Department of Computer Science
University of Waterloo
x749liu@uwaterloo.ca

Abstract

While much efforts have been made to improve the query efficiency of computational geometric problems, much less work has been made to improve the space efficiency for those problem. Succinct and implicit data structures can be used to significantly reduce the storing cost. In the following, we will introduce these two kinds of data structures and will summarize the paper of Succinct and Implicit Data Structures for Computational Geometry by He [He, 2013]. Then we will discuss some further works of this paper.

1. Introduction

Jacobson Initially proposed the succinct data structure in 1989. Succinct data structure can represent a given dataset using space close to the information-theoretic lower bound. For example, let Z be that lower bound, then a data structure is succinct if the space needed is $(Z + \sqrt{Z})$ bits or $(Z + lgZ)$ bits. Meng He [He, 2013] presented some succinct data structures for geometric query problems including range search and point location. In the paper, we mainly show our study and discovery in the problem of range search.

Implicit data structure stores elements based on relative order, and keeps any structure "implicit" in the relative order of the values. Munro and Suwanda [Munro and Suwanda, 1980] firstly defined an implicit data structure as one "in which structural information is implicit in the way data are stored, rather than explicit in pointers." and they formalized the notion of an implicit data structure while introducing bi-parent heap. One simple example of implicit data structures is heap. Heap can be implemented by an simple array. However, the parent-children (larger or smaller) relationship of each element is preserved in the permutation of the array. We do not need any extra space other than the space to store element in preserving those relationships.

Implicit data structures are very useful in solving problems in computational geometry. We will discuss

how the three fundamental problems of computational geometry: Range search, Point location, and Nearest neighbor can be approached by implicit data structures.

2. Preliminary

Definition 2.1 (Nearest neighbor searching problem)**.** Preprocess a point set so that the point closest to a query point can be found efficiently.

Definition 2.2 (Range search problem)**.** Preprocess a point set so that information regarding points inside a query region can be efficiently retrieved.

Definition 2.3 (Point location problem)**.** Preprocess a subdivison of space into a set of cells so that the cell that contains a query point can be located quickly.

2.1. Succinct data structures

Lemma 2.1. [Jacobson, 1989, Clark and Munro, 1996] A bit vector $B[1..n]$ with v 1s can be represented using $n + o(n)$ bits to support the access to each bit, rank and select in $O(1)$ time.

Lemma 2.2. A bit vector $B[1..kn]$ with v 1s can be represented using $kn + o(n)$ bits to support the access to each bit, rank and select in $O(1)$ time. k is a constant number larger than 1.

Proof. Using lemma 2.1, operation time on $B[1 \dots kn]$ is k times of that on $B[1..n]$, which is $O(1)$. Since k is a constant number, operation time on $B[1..kn]$ is $O(1)$ time. □

Lemma 2.3. A bit vector $B[1..n^2]$ with v 1s can be represented using $n^2 + o(n^2)$ bits to support the access to each bit, rank and select in $O(n)$ time.

Theorem 2.4. A set of n points on an nn grid can be represented using $nlgn + o(nlgn)$ bits to support orthogonal range counting in $O(lgn/lglgn)$ time, and orthogonal range reporting in $O((k + 1)lgn/lglgn)$ time, where k is the number of points reported. This data structure can be constructed in $O(nlgn)$ time.

2.2. Implicit data structures

Definition 2.4 (Vertical ray shooting problem)**.** A set of n disjoint line segments is given, and the query returns the line segment immediately below a given query point.

Definition 2.5 (Decomposable search problem)**.** Decomposable search problem is a searching problem for which one can decompose the searched domain into into groups, computing the solution for each group individually and take the solution that is the best of all.

Theorem 2.5 (Separator theorem)**.** [Chan and Chen, 2008] Given a set P of n points in the plane and a parameter $1 \leq b \leq n$, we can partition P into a subset P_S (the separator) of $O(\sqrt{bn})$ points and $O(b)$ subsets

$P_1, P_2, ...$ (the clusters) each of $O(\frac{n}{b})$ points, so that:
For every two point $p, p' \in P - P_S$, if $Vor(p, P)$ is adjacent to $Vor(p', P)$, then p and p' belong to the same cluster, where $Vor(P)$ is the Voronoi diagram of a point set P and $Vor(p, P)$ is the Voronoi cell of the point $p \in P$ in $Vor(P)$.

3. Succinct data structures: Orthogonal range counting and reporting

For this orthogonal query problem, Meng He [He, 2013] focuses on two-dimensional cases. More specifically, we consider range counting and reporting on an n×n grid with n points. The query value is an axis-aligned query rectangle, and we calculate the points within the query rectangle. We summarize existing succinct data structures using either wavelet tree or generalized wavelet tree. We also raise some question based on these works and made our attempt to find solutions.

3.1. Wavelet tree

First, we need to know some background of bit vector. For a bit vector B[1...n] storing 0s and 1s, we consider three operations: **access**(B,i), **rank**$_\alpha$(B,i) and **select**$_\alpha$(B,i), $\alpha \in \{0,1\}$. **access** returns the i_{th} bit. **rank**$_\alpha$ gives the numbers of bit α in B[1...i]. **select**$_\alpha$ gives the position of i_{th} α. With work of Jacobson [Jacobson, 1989] and Clark & Munro [Clark and Munro, 1996], a succinct bit vector can perform these three operations in $O(1)$ time under the word RAM model using $n + o(n)$ bits. This can be summarized as Lemma 2.1.

Grossi et al [Grossi et al., 2003] designed a wavelet tree to succinctly represent a string S over alphabet $[\sigma]$. For example, we consider a string S, 35841484, over an alphabet of size 8. Here is how we construct a wavelet tree for this string. For the root node r, we represent S but let each number in S that belongs to lower half of $[\sigma]$ be 0, and the rest number that belongs to larger half of $[\sigma]$ be 1. The position of each number remains unchanged. Then the root contains a bit vector 01100010, to which Lemma 2.1 can be applied. Root node has two children. The left one contains the subsequence of S with characters that are represented as 0 bits in root node. Correspondingly, the rest character of S goes to the right child. Then for string S, the alphabet of the two children are respectively [1, 8/2], [8+1, 8]. We continue these processes for each node recursively to get a wavelet tree, which can be seen in figure 1. The space cost of a wavelet tree is $(n + o(n))\lceil lg\sigma \rceil$. For the top-down traversal, access to each level is of constant time, so the overall running time is O(lg σ).

Mäkinen et al. [Mäkinen and Navarro, 2007] then implemented wavelet tree on orthogonal range counting and reporting. The Problem M¨äkinen focuses on is n points on an n*n grid, with a restriction that each point have distinct x-coordinates. In this case, we can have a one-dimension representation of such two-dimension case. For example, we can see the previous example as eight coordinates: (1,3), (2,5), (3,8), (4,4), (5,1), (6,4), (7,8), (8,4). To perform range counting, the query value is a rectangle P [x_1..x_2] × [y_1..y_2]. We

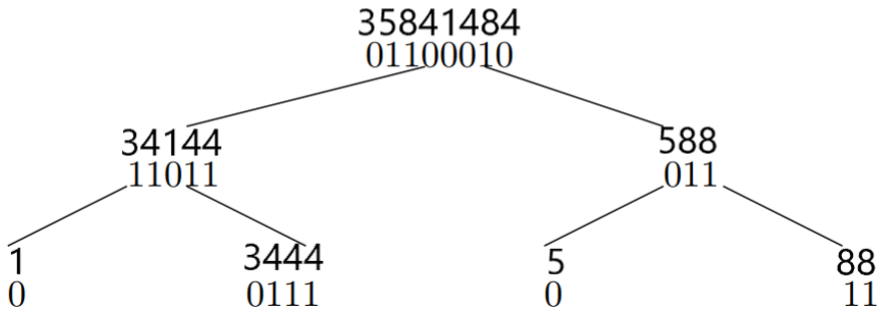

Figure 1. A wavelet tree constructed for the string 35841484 over an alphabet of size 8.

perform a top-down traversal, and cumulatively counting points that are within [y_1..y_2]. For each level, at most two nodes are visited during the traversal. Therefore, the range counting can be performed in $O(lgn)$ time. Similarly, the range reporting can be performed in $O((k+1)lgn)$ time, where the k is the number of points within query rectangle.

3.2. Generalized wavelet tree

Ferragina et al. [Ferragina et al., 2007] proposed a generalized wavelet tree to further improve the performance of rank and select operations on strings. Ferragina focuses on a string of length n over alphabet t, where $t = \lg^{\epsilon} n$, $\epsilon \in (0,1)$. In a generalized wavelet tree, each node has t children. The access and rank can be performed in $O((lg\sigma/lglgn)+1)$ time. Using generalized wavelet tree, Bose et al. [Bose et al., 2009] and Yu et al. [Yu et al., 2011] then designed a succinct data structure for range search problem of n points on n $\times$ $\lg^{\epsilon}$ n grid. Bose et al. [Bose et al., 2009] also eliminated the restriction of the data structure that Mäkinen designed. In latter part of this paper, we will look closer of how Bose et al. [Bose et al., 2009] and Yu et al. [Yu et al., 2011] did elimination. Their work can be summarized as Theorem 2.4.

In conclusion, for range searching problem, Mäkinen et al. [Mäkinen and Navarro, 2007] used wavelet tree to construct succinct data structure for the situation of n points in an n*n grid. Bose et al. [Bose et al., 2009] and Yu et al. [Yu et al., 2011] then improved the performance using generalized wavelet tree. On the bases of their work, we ask a question: can we design a succinct data structure for more than n points in an n$\times$n grid? The number of points can be from slightly larger than n to close to n^2. In the next part, we show our attempt to design succinct data structures for this situation.

3.3. Attempt to design succinct data structure for range search problem of m (m>n) points on an n$\times$n grid

We firstly look at how Bose et al. [Bose et al., 2009] removed the restriction of distinct x-coordinates. The idea of his work is to reassign distinct x-coordinate for each point.For a given point set N that may contain

points of the same x-coordinate, we construct a point set N' from it. Firstly, we sort N in increasing order. Then for the i^{th} point (x_i, y_i) in N, we add point (i,y_i) to N'. Additionally, we need an auxiliary bit vector C = $10^{k1}10^{k2}\cdots10^{kn}$ to record the number of points for each x-coordinate, where k_j means number of points on $x = x_j$. To perform range counting and reporting, we transfer the query rectangle R = [x_1..x_2]×[y_1,y_2] to R' = [x'_1..x'_2]×[y_1,y_2] using bit vector C. This process can be proven of constant time using $Lemma2.1$. Then we only need to perform range counting and reporting on N' with R'. Using merge sort, the time used to construct N' and C takes $O(nlgn + n)$ time. The space and time complexity of N and N' are the same.

Now, we use the above approach on the case of kn points on an n×n grid, where k is a constant number larger than 1. We consider a point set N, where $|N|$= kn. The steps for constructing the new point set N' and bit vector C are similar to the steps above. In this part, we use wavelet tree as the succinct data structure to express N'. The depth of this wavelet tree remains to be $O(lgn)$. The length of C is $(k + 1)n + o(n)$ using Lemma 2.1 for n 1s and kn 0s. The length for each bit vector in the wavelet tree is $kn + o(n)$. Using $Lemma2.2$, access to each node in a wavelet tree of N' takes $O(1)$ times. Based on the work of Mäkinen et al. [Mäkinen and Navarro, 2007], we can easily get that the range counting takes $O(lgn)$ time. Using the method in Bose et al. [Bose et al., 2009] for range reporting, we can get $O((l + 1)klgn)$ time, where l is the output size.

If we further enlarge the size of the given point set to be close to n^2, we can possibly using the method above to design a wavelet tree using $O(n^2lgn)$ space, and supporting range counting in $O(nlgn)$ time, if the Lemma 2.3 can be proven. Thus, we need to conduct further research on work of Jacobson [Jacobson, 1989], and Clark & Munro [Clark and Munro, 1996]. Another emerged problem is that the sorting process of our attempted method dominates the run time as the given point set gets extremely large. We will seek for other algorithm that does not require sorting but still makes data structure succinct, as well as maintaining the run time performance.

4. Implicit data structures

While succinct (semi-implicit) data structures uses the amount of space that is "close" to information-theoretic lower bound ($Z + o(Z)$ bits of space), implicit data structures use $Z + O(1)$ bits of space where Z is the information-theoretical optimal number of bits needed to store some data.

4.1. Range search

In range search, a typical question is that we would like to the number of points inside a specific region. The data structure we would firstly come up with is kd-tree.

Fiat and Munro et al. [Fiat et al., 1991] proposed the first implicit representation of kd-tree for this problem. Given an array A of size n. Each element in A represent the coordinate of a point. Note that dimensions are $0, 1, \ldots, d - 1$.

Algorithm 1: Convert array A to an implicit k-d tree

```
Treat the entire array A as one segment S containing all the array entries ;
Initialize i := 1;
if the segment is not empty or only contains one element then
    j := i mod d;
    Find the point m who is the median in S in terms of dimension j;
    Swap m with the point stored in the middle of S;
    Move the points in S whose coordinates in dimension j are smaller than m to the first half of S;
    Let the first half and the median element be a new segment L;
    Let the second half be a new segment R;
    Recurse on segment L;
    Recurse on segment R;
else
    Return;
end
```

The recursion depth of this algorithm is $O(lgn)$. Eventually, each segment will store exactly one point, and the content of the array A becomes the implicit k-d tree structure.

Arroyuelo et al. [Arroyuelo et al., 2011] designed the first adaptive data structure for two-dimensional orthogonal range search which had the better search performance than the worst case showed by Demaine [Demaine et al., 2000] . It had a implicit version which supports $O(klogn + m)$, where k is the minimum number of monotonic chains that the point set can be decomposed into, m is the number of points returned, and n is the number of points in the point set. The query algorithm of Arroyuelo et al. [Arroyuelo et al., 2011] required the coordinates of the points are stored in such a manner that given two integers i and j, the coordinates of the jth leftmost point in the ith monotonic chain can be retrieved in constant time. Besides, they proposed a way to store all the points as follow:

1. Initialize an array A to store all the points
2. We divide A into k chunks: $A_1, A_2, ..., A_k$. For the i-th chunk in A, A_i has the same size as the number of points in C_i
3. Loop from 1 to k. If C_i has even number of points: we store the coordinate pair of the jth leftmost point of C_i in $A_i[j]$, for $j = 1, 2, ..., l_1$. If C_1 has odd number of points: we store all but the rightmost point in $A_i[1..j-1]$ in the same order

Eventually, we would store all the points in a monotonic chain of an even size in A. Then, we store the rightmost points of odd size in the remaining entries of A sorted by the number of the chain each point is in.

Hence, the required information of the adaptive range search algorithm is encoded by permutating the input array. We do not need any extra space to store those points.

4.2. Point location

Bose et al. [Bose et al., 2012] proposed a succinct data structure to answer point location queries on planar triangulations in $O(lgn)$ time and its related problem - vertical ray shooting problem (definition 2.4). Furthermore, they modified the succinct data structure to let it become an implicit data structure by grouping the vertices of a planar triangulation in pairs and encoding the subregions through permuting those pairs. Each bit of information can be encoded by swapping each of the pairs. In this way, permutation of the data structure can be encoded in indexes. This implicit data structure supports point location query in $O(lg^2n)$ time. The vertical shooting problem can also be supported with this implicit data structure in $O(nlgn)$ time per query.

4.3. Nearest neighbor search

2d-nearest neighbor search problem is well-known be to solved by constructing a Voronoi diagram (figure 2). To search for a query point's nearest neighbor, we can first search for the cell which the query point is in. Then the site corresponding to the cell is the nearest neighbor of the query point. However, the reduction of Voronoi diagram requires $O(n)$ extra space. It cannot be directly used to design an implicit data structure.

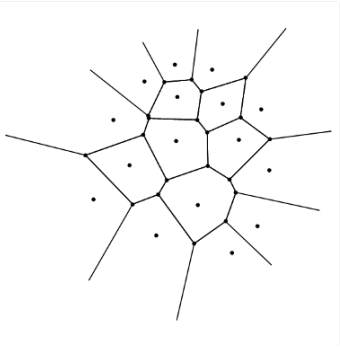

Figure 2. Voronoi diagram

Chan et al. [Chan and Chen, 2008] tried to encode the planar graph - Voronoi diagram. They firstly introduced the slab method to represent the Voronoi diagram: We draw line segments at every vertex of the Voronoi diagram. we store the line segments at each of the $O(n)$ vertical slabs sorted by y-axis. When we query a point q, we perform binary search in x-axis to find the slab containing q. Then we perform binary search in y-axis to find the line segment closely above q. In this method, we need to store each of the O(n) sorted lists in subarrays to support the binary search.

In order to solve this issue without using extra space, Chan el al.[Chan and Chen, 2008] developed a structure that organized points in an array with van Emde Boas layout. With the fact that the nearest neighbor searching problem is a decomposable search problem (Definition 2.5) proved by Bentley and Saxe [Bentley and Saxe, 1980], they applied the Separator theorem (Theorem 2.5) to decompose the Voronoi diagram into several clusters. To be more specific, they permutated the array of input points and let the points in the same cluster be together. And we can use the idea of decomposable search problem which is very similar

to divide and conquer to find the nearest neighbor for the query point. The most difficult part to come up with this idea is representing the a planar graph (Voronoi diagram) by the objects (separator and clusters of points) which can be stored in a simple array. Besides, it is also difficult that Chan et al [Chan and Chen, 2008] also found a permutation to store those objects with all their relationships preserved in an array without extra space.

5. Further work

For succinct data structures, we attempted using method of Bose et al. [14] to design a wavelet tree for range search problem on data set of more than n points on n$\times$n grid. We will continue study that if our method can be applied to generalized wavelet tree for a faster query time. For the problem of near n^2 points on n$\times$n grid, we will continue to prove the Lemma 2.3 and explore other succinct structure for this query problem. Meng He [He, 2013] also introduced succinct data structure for orthogonal query problem of point location. Thus, we will also learn these succinct data structures and raise questions correspondingly.

After researching those implicit data structures in computational geometry, we can summarize the design strategy for those implicit data structures:

1. Represent the geometric objects in the problem by the concrete objects.
2. Design a succinct data structure to solve the problem (permutation + bits model).
3. Encode the contents of the additional memory with the complexity of the operations on the succinct version preserved.

The key step in designing an implicit data structure is encoding information by permutating the given input with all its characteristics preserved. Many other problems in computational geometry have been solved by implicit data structures afterwards. Blunck [Blunck and Vahrenhold, 2010] proposed an implcict data structure to solve problems related to finding maxima among points in two and three dimensions. Asano [Asano, 2012] developed an in-place algroithm to eras a connected component in a binary image. We can also think about if we can build dynamic implicit data structures which allows the implicit data structures to some operations, such as insertion and deletion.

References

[Arroyuelo et al., 2011] Arroyuelo, D., Claude, F., Dorrigiv, R., Durocher, S., He, M., López-Ortiz, A., Ian Munro, J., Nicholson, P. K., Salinger, A., and Skala, M. (2011). Untangled monotonic chains and adaptive range search. *Theor. Comput. Sci.*, 412(32):4200–4211.

[Asano, 2012] Asano, T. (2012). In-place algorithm for erasing a connected component in a binary image. *Theor. Comp. Sys.*, 50(1):111–123.

[Bentley and Saxe, 1980] Bentley, J. L. and Saxe, J. B. (1980). Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301 – 358.

[Blunck and Vahrenhold, 2010] Blunck, H. and Vahrenhold, J. (2010). In-place algorithms for computing (layers of) maxima. *Algorithmica*, 57:1–21.

[Bose et al., 2012] Bose, P., Chen, E. Y., He, M., Maheshwari, A., and Morin, P. (2012). Succinct geometric indexes supporting point location queries. *ACM Trans. Algorithms*, 8(2).

[Bose et al., 2009] Bose, P., He, M., Maheshwari, A., and Morin, P. (2009). Succinct orthogonal range search structures on a grid with applications to text indexing. 5664:98–109.

[Chan and Chen, 2008] Chan, T. M. and Chen, E. Y. (2008). In-place 2-d nearest neighbor search. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, page 904–911, USA. Society for Industrial and Applied Mathematics.

[Clark and Munro, 1996] Clark, D. R. and Munro, J. I. (1996). Efficient suffix trees on secondary storage. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '96, page 383–391, USA. Society for Industrial and Applied Mathematics.

[Demaine et al., 2000] Demaine, E. D., López-Ortiz, A., and Munro, J. I. (2000). Adaptive set intersections, unions, and differences. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, page 743–752, USA. Society for Industrial and Applied Mathematics.

[Ferragina et al., 2007] Ferragina, P., Manzini, G., Mäkinen, V., and Navarro, G. (2007). Compressed representations of sequences and full-text indexes. *ACM Trans. Algorithms*, 3(2):20–es.

[Fiat et al., 1991] Fiat, A., Ian Munro, J., Naor, M., Schäffer, A. A., Schmidt, J. P., and Siegel, A. (1991). An implicit data structure for searching a multikey table in logarithmic time. *Journal of Computer and System Sciences*, 43(3):406 – 424.

[Grossi et al., 2003] Grossi, R., Gupta, A., and Vitter, J. S. (2003). High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, page 841–850, USA. Society for Industrial and Applied Mathematics.

[He, 2013] He, M. (2013). *Succinct and Implicit Data Structures for Computational Geometry*, pages 216–235. Springer Berlin Heidelberg, Berlin, Heidelberg.

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

[Mäkinen and Navarro, 2007] Mäkinen, V. and Navarro, G. (2007). Rank and select revisited and extended. *Theoretical Computer Science*, 387(3):332 – 347. The Burrows-Wheeler Transform.

[Munro and Suwanda, 1980] Munro, J. and Suwanda, H. (1980). Implicit data structures for fast search and update. *Journal of Computer and System Sciences*, 21(2):236 – 250.

[Yu et al., 2011] Yu, C.-C., Hon, W.-K., and Wang, B.-F. (2011). Improved data structures for the orthogonal range successor problem. *Computational Geometry*, 44(3):148 – 159.