Untangled Monotonic Chains and Adaptive Range Search*

Diego Arroyuelo[1**], Francisco Claude[2], Reza Dorrigiv[2], Stephane Durocher[3], Meng He[2], Alejandro López-Ortiz[2], J. Ian Munro[2], Patrick K. Nicholson[2], Alejandro Salinger[2], and Matthew Skala[2,4]

[1] Yahoo! Research Latin America, Chile, darroyue@dcc.uchile.cl
[2] Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, {fclaude,rdorrigiv,sdurocher,mhe,alopez-o,imunro, p3nichol,ajsalinger,mskala}@cs.uwaterloo.ca
[3] Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada, durocher@cs.umanitoba.ca
[4] Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, mskala@ansuz.sooke.bc.ca

Abstract. We present the first adaptive data structure for two-dimensional orthogonal range search. Our data structure is adaptive in the sense that it gives improved search performance for data with more inherent sortedness. Given n points on the plane, the linear-space data structure can answer range queries in $O(\log n + k + m)$ time, where m is the number of points in the output and k is the minimum number of monotonic chains into which the point set can be decomposed, which is $O(\sqrt{n})$ in the worst case. Our result matches the worst-case performance of other optimal-time linear-space data structures, or surpasses them when $k = o(\sqrt{n})$. Our data structure can also be made implicit, requiring no extra space beyond that of the data points themselves, in which case the query time becomes $O(k \log n + m)$. We present a novel algorithm of independent interest to decompose a point set into a minimum number of untangled, same-direction monotonic chains in $O(kn + n \log n)$ time.

1 Introduction

Applications in geographic information systems, among others, require structures that can store and retrieve spatial data efficiently in both space and time. In this work we describe a data structure and algorithm for two-dimensional orthogonal range search, which is a commonly-encountered spatial data retrieval problem. Our data structure is *adaptive*, giving improved query performance for

* Funding for this research was made possible by NSERC Discovery Grants, the Canada Research Chairs Program, and the NSERC Strategic Grant on Optimal Data Structures for Organization and Retrieval of Spatial Data.

** Much of this work took place while the first author was a visitor at the University of Waterloo.

data with more inherent sortedness; and can be *implicit*, requiring no added storage space beyond that of the data points themselves.

The problem of *two-dimensional orthogonal range search* can be defined as follows: let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane, and let $r = [x_1, x_2] \times [y_1, y_2]$ be a *query range.* The orthogonal range search problem asks for all points $p_i \in P$ such that $x_1 \leq x(p_i) \leq x_2$ and $y_1 \leq y(p_i) \leq y_2$, where $x(p_i)$ and $y(p_i)$ denote the x and y coordinate values of point p_i, respectively. An orthogonal range search data structure preprocesses the set P in order to efficiently answer arbitrary range queries; a natural goal is to balance the conflicting objectives of minimizing both the space required by the data structure and the time required to answer queries.

Our basic data structure's worst-case query time is $O(k \log n + m)$, where n is the number of points in the point set; m the number of points in the output; and k the minimum number of monotonic chains into which the point set can be decomposed, which is $O(\sqrt{n})$ in the worst case. Applying fractional cascading [4] reduces the query time to $O(\log n + k + m)$ at the cost of implicitness.

We require that the monotonic chains should be untangled. That is, when successive vertices are connected by line segments, the chains should not intersect each other. This requirement does not increase the minimal number of chains. We present a novel algorithm for finding a minimal set of untangled chains (all monotonic in the same direction) in $O(kn + n \log n)$ time; this untangling algorithm is of independent interest.

2 Previous Work

Any set of n points can be split into some number k of chains in which the y coordinate is *monotonically* increasing or decreasing as the x coordinate increases. When all chains must be ascending (or all descending), the problem of finding a minimal chain decomposition is well-studied. With worst-case data the minimal number of chains may be $\Theta(n)$, even given a choice of ascending or descending chains. Supowit gives an algorithm for it with worst-case running time $\Theta(n \log n)$ [12], which is optimal [3]. If chains of both types are allowed, then minimizing the number of chains is NP-hard [5]. However, an algorithm of Fomin, Kratsch, and Novelli acheives a constant-factor approximation of the minimal number of chains in $O(n^3)$ time [6]. An algorithm of Yang, Chen, Lu, and Zheng generates a decomposition into at most $\lfloor \sqrt{2n + 1/4} - 1/2 \rfloor$ chains of both types (which is the minimal number for worst-case data) in $O(n^{3/2})$ time [14]. They do not prove any guaranteed approximation factor when the minimal number of chains is smaller than $O(\sqrt{n})$, but comment that in practical experiments their algorithm often achieves very close to the constant-factor approximation value.

The two-dimensional orthogonal range search problem has received considerable attention, and several efficient data structures exist. For instance, R-trees [7] are a multidimensional extension of B-trees. An R-tree is a height-balanced tree, where each tree node represents a region of the underlying space. Thus, the data structure divides the space with hierarchically nested (and possibly overlapping) minimum bounding rectangles. The search algorithm descends the tree, recursing into every subtree whose bounding rectangle overlaps the query. In the worst case a search could be forced to examine the entire tree in $O(n)$ time, even when the query rectangle is empty. However, R-trees are simple to implement, use linear space, tend to perform much better in practice than the theoretical worst case, and are popular as a result.

Range trees [9] support multidimensional range queries by generalizing balanced binary search trees to multiple dimensions. The data points are indexed along one dimension in a standard balanced binary search tree. At each node v of that tree, we collect all the descendants of v and store a new balanced binary search tree storing all those points indexed along the second dimension. A rectangle query descends the first tree to do a one-dimensional range search in $O(\log n)$ time, then searches along the other dimension for an overall time of $O(\log^2 n+m)$. More advanced techniques, like fractional cascading [4], allow the two-dimensional search time to be reduced to $O(\log n + m)$; and the technique can also be extended to higher dimensions at some cost in search time.

Alternative solutions exist that require linear space like R-trees but improve on the worst-case search time. Kanth shows that $O(\sqrt{n} + m)$ worst-case search time is optimal for non-replicating (or linear-space) data structures [8]. Bentley achieves it with kd-trees [2], which recursively divide a k-dimensional space with hyperplanes. Munro describes an *implicit* kd-tree, with optimal search time and no storage used beyond that of the points themselves [10]. Arge describes priority R-trees, or PR-trees [1], also with $O(\sqrt{n} + m)$ worst-case search time. In a recent result, Nekrich [11] presents a data structure that uses linear space with search time $O(\log n+m \log^{\epsilon} n)$, trading suboptimal performance in m for better performance in n. See Table 1 for a comparison of methods.

Table 1. Summary of orthogonal range query results; n is the number of points in the database, m is the number of points returned, and k is the number of chains.

Data structure	Worst-case search time	Space
R-trees [7]	$O(n)$	$O(n)$
kd-trees [2, 10]	$O(\sqrt{n} + m)$	implicit
PR-trees [1]	$O(\sqrt{n} + m)$	$O(n)$
Range trees [9]	$O(\log n + m)$	$O(n \log n)$
Nekrich [11]	$O(\log n + m \log^{\epsilon} n)$	$O(n)$
This paper	$O(\log n + k + m)$	$O(n)$
This paper	$O(k \log n + m)$	implicit

To summarize, R-trees are practical, but do not provide worst-case guarantees at search time, and range trees have an impractical $O(n \log n)$ space requirement. There are alternative solutions requiring linear space and providing better search time. However, none of these can profit from “easy” data. Here we present an *adaptive* data structure. When the data can be decomposed into a small number of monotonic chains, our search performance improves. If the number of chains $k = o(\sqrt{n})$, we surpass the performance of optimal-time linear-space data structures [1, 2, 8, 10].

3 Finding Untangled Chains

In the next section we describe an adaptive algorithm and data structure for two-dimensional orthogonal range search on data decomposed into a union of monotonic chains. The data structure performs better when there are fewer chains. Furthermore, we can search more efficiently by assuming that the chains are untangled: successive data points can be connected with line segments with no segments intersecting. That raises the question of how to find an optimal untangled chain decomposition, which we resolve in this section. Due to lack of space we omit proofs in this section, as they are mainly based on exhaustive case analysis.

Although our data structure asks for an optimal decomposition into chains with both ascending and descending monotonic chains allowed, it actually functions by splitting the points into the two directions as a preprocessing step and then considering the two directions separately; chains are only required to be untangled with respect to other chains of the same type. The untangling problem of interest to us, then, is how to decompose a set of points into a minimal number of untangled chains all in one direction (without loss of generality, descending). Also assume that points in the input set are in general position.

It is easy to see that removing a single tangle between two chains does not change the number of chains, so the minimum number of untangled chains is the same as the minimum number of possibly-tangled chains.

However, finding tangles to remove requires search, and each untangling move could introduce many new tangles, resulting in an expensive untangling procedure. Van Leeuwen and Schoone show that such a process must terminate after $O(n^3)$ moves [13]. They describe an $O(n^2)$ exhaustive search to find each tangle, for an overall time of $O(n^5)$. We describe an algorithm for finding a minimal number of chains in $O(n \log n + kn)$ time where k is the number of chains.

3.1 Untangling Monotonic Chains

Given two points $p_i, p_j \in P$, we say that the edge or line segment (p_i, p_j) is *valid* if $x(p_i) \leq x(p_j)$ and $y(p_j) \leq y(p_i)$. We also say that points p_i and p_j are *compatible* if (p_i, p_j) or (p_j, p_i) is valid. A *chain* is a sequence of edges $C = \{(p_1, p_2), (p_2, p_3), \ldots, (p_{n-1}, p_n)\}$ where each one is valid. We will often refer to a point $p \in C$ for some chain C, which means that p is an endpoint of some edge in C. A *sub-chain* S of C is a contiguous subset of the edges $\{(p_k, p_{k+1}), \ldots, (p_{k+\ell-1}, p_{k+\ell})\}$, where $k + \ell \leq n$. We call ℓ the length of S.

Supowit [12] proposed an algorithm, Algorithm 1, for decomposing points into a minimal number of possibly-intersecting same-direction monotonic chains.

Algorithm 1 Minimum number of descending chains

$S \leftarrow \emptyset$
for $i = 1 \dots n$ **do**
  let $S' = \{A \in S, \text{miny}(A) \geq y(p_i)\}$
  if $S' \neq \emptyset$ **then**
    let $A_0 = \text{argmin}_A\{\text{miny}(A), A \in S'\}$
    append p_i to A_0
  else
    add p_i as a chain to S
return S

Let A be a chain and $\text{miny}(A) = \min\{y|(x, y) \in A\}$. Let $P = \{p_1, p_2, \dots, p_n\}$ be the data points sorted by increasing x-coordinate.

If an edge in one chain intersects an edge in another chain, we call the intersection a *tangle* and the chains *tangled* with each other. Let $\mathcal{L}(P)$ be the set of all valid edges and $\mathcal{L}^*(P)$ be the set of edges created by running Algorithm 1 on P. Then for any edge (p_i, p_j), define $H^+(p_i, p_j)$ to be the open half-plane bounded by the line through p_i and p_j and containing the point $(x(p_i) + 1, y(p_i) + 1)$, and $H^-(p_i, p_j)$ symmetrically. Now we can show that all tangles in the output of Algorithm 1 are of a special kind.

Definition 1. *Suppose we have two chains C_2 and C_1 with edges $(q_1, q_2) \in C_2$ and $(p_1, p_2), \dots, (p_{\ell-1}, p_\ell) \in C_1$ such that $p_1 \in H^-(q_1, q_2)$, $p_\ell \in H^-(q_1, q_2)$, and $p_i \in H^+(q_1, q_2)$ for all $1 < i < \ell$. We call such a tangle a "valid"-tangle (abbreviated as v-tangle). Fig. 1 shows examples. We call (q_1, q_2) the upper part of the v-tangle, and $(p_1, p_2), \dots, (p_{\ell-1}, p_\ell)$ the lower part.*

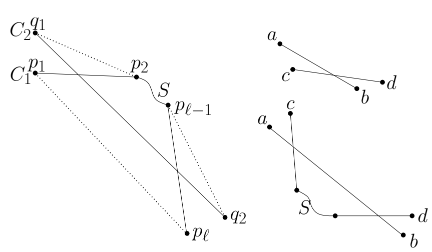

Fig. 1. (Left) Valid tangles (v-tangles) generated by Algorithm 1. (Right) Two examples of tangles that cannot be generated by Algorithm 1.

Lemma 1. *All tangles created by Algorithm 1 are v-tangles.*

Since only v-tangles are possible in the output of Algorithm 1, there is an intuitive ordering on the set of chains. Suppose we run Algorithm 1 on P and it generates k chains. We can create a set of k points $Q = \{q_1, \ldots, q_k\}$ such that $x(q_i) < x(q_{i+1})$, no two points in Q are compatible with each other, but every point in Q is compatible with every point in P, Then, if we execute Algorithm 1 again on $P \cup Q$, each q_i will be added to a single chain C_i, and we can order the chains based on these points. We will assume we have such a set at the beginning of the chains and another at the end in order to avoid special boundary cases. Thus, given two chains C_i and C_j, we can refer to C_j as the upper chain if $j > i$. The uppermost chain is C_k.

With this ordering in mind, we now discuss how to untangle a v-tangle. The following lemma illustrates that untangling a v-tangle does not create new tangles involving upper chains.

Remark 1. Given a v-tangle, as shown in Fig. 1, we can untangle it by using the dotted lines as edges. This is just moving S to be part of C_2. As we just explained, it does not matter how the points change and move around chains, chain C_i is the one that would contain q_i.

Lemma 2. *Consider two tangled chains C_i and C_j as in Fig. 2. By removing a v-tangle, where C_j is above C_i, we cannot generate new tangles involving chains above C_j.*

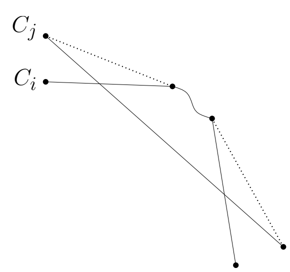

Fig. 2. Illustration of the case considered in Lemma 2.

Consider Algorithm 2. Each iteration of the outer `for` loop ensures that chain C_i is not tangled with any chains below, $C_1, \ldots, C_{i-1}$.

Algorithm 2 Untangled-Chains(P)

1: Run Algorithm 1 on P to get chains $C_1, \ldots, C_k$ where C_k is the uppermost chain.
2: **for** $i = k$ down to 1 **do**
3:     **for** $j = i - 1$ down to 1 **do**
4:         Find and untangle all v-tangles between C_i and C_j

To find the tangles we just traverse both chains in order of increasing x-coordinates of their points, so the process take time proportional to the sum of the lengths of the chains. Our method of untangling also has the following useful invariant properties.

Lemma 3. *Consider the set of points R in chains* $C_1, \ldots, C_{i-1}$ *after untangling* $C_i, \ldots, C_k$. *If we run Algorithm 1 with input R, the resulting set of chains is exactly* $C_1, \ldots, C_{i-1}$.

Lemma 4. *After we have untangled* C_i *with chains* $C_{i-1}, \ldots, C_1$, *no subsequent untangling operations occurring among chains* $C_1, \ldots, C_{i-1}$ *can cause a new tangle to form with* C_i.

The previous results allow for the possibility that during the untangling of C_i, we could (temporarily) create non-v-tangles involving C_i. In fact, such tangles are possible if the order in which the untangling is done is arbitrary, which can be seen in Fig. 3. However, since Algorithm 2 untangles the chains in descending order, this situation cannot occur.

What remains to be shown is that in the process of untangling the upper chain C_i from the chains $C_1, \ldots, C_{i-1}$, when untangling a v-tangle, any other v-tangles involving C_i either disappear or remain being a v-tangle.

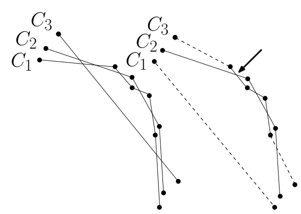

Fig. 3. Untangling chains in an arbitrary order may cause tangles which are not v-tangles. For example, untangling C_1 and C_3 results in such a situation. The arrow points to a new tangle that is not a v-tangle.

Lemma 5. *Suppose a v-tangle between* C_i *and* C_j *is removed by Algorithm 2, where* C_j *is the upper chain. Any tangles between* C_j *and* C_ℓ *where* $\ell < j$ *may have been altered. However, the remaining tangles are still v-tangles.*

Now we state our main theorem about the untangling process:

Theorem 1. *A set of n points in the plane can be decomposed into a minimal set of chains without tangles in* $O(n \log n + kn)$ *time, where k is the number of chains.*

4 Adaptive Orthogonal Range Search

First, observe that if the data points form a single monotonic chain, then the answer to any query must be a contiguous interval of the ordered list of points, and we can find it with a binary search. We can store such a data set in $O(n)$ space and answer queries in $O(\log n + m)$ time, where n is the number of data points and m is the number of points returned by the query.

Now assume that as a preprocessing step the data points have been decomposed into a minimal number k of monotonic chains. A truly optimal decomposition would require solving an NP-hard problem, but we can come within a constant factor in $O(n^3)$ time with the algorithm of Fomin, Kratsch, and Novelli, and that is good enough to preserve the asymptotic search time of our data structure [6]. The $O(n^{3/2})$ partitioning algorithm of Yang, Chen, Lu, and Zheng offers no guarantee of a minimal decomposition, but appears to come close in practice and may be preferable in real applications [14]. In either case, once we have a decomposition of the data points into chains, we separate the ascending and descending chains, and treat the two directions separately, building a data structure for each and running every query on both.

The two-direction minimization algorithms are used only to decide for each point whether it will go into the ascending or descending structure. Having made that decision, we run the algorithm of the previous section to find a minimal set of untangled chains for each direction; doing so cannot increase the number of chains further.

Without loss of generality, we describe the data structure for descending chains here. The ascending case is symmetric. Let $\{C_1, C_2, \ldots, C_k\}$ be the set of untangled descending chains, and let ℓ_i be the length of C_i. Let $r = [x_1, x_2] \times [y_1, y_2]$ be the query range.

We first find the set of chains that intersect r. If we store the chains ordered from left to right as described in the previous section, we can find the first chain to pass above the point (x_1, y_1) and the last chain to pass below the point (x_2, y_2), and know that all chains intersecting the query range must be between those two chains in the ordering. Evaluating whether a point is above or below a chain can be accomplished by a simple binary search over the points in the chain in $O(\log n)$ time, so with two binary searches over the chains we can find the start and end of the range of chains that might intersect r, in $O(\log k \log n)$ time. Let $k' \leq k$ be the number of chains in that subset.

For each of the k' chains that might intersect r, we can do two more binary searches to find the start and end of the interval of data points within the chain, that are actually included in the query range. Note that because of the monotonicity of the chains, this must be a contiguous interval. The time to do these searches is $O(\log \ell_i)$ for each of the k' chains, and since $\sum \ell_i = n$, the time for this step is $O(k' \log(n/k'))$.

The number of points m returned by the query also places a lower bound on the running time simply because we must spend time writing them out. Adding up the times gives the following lemma:

Lemma 6. *Given a set of n points which can be decomposed into k monotonic chains, we can in $O(n^3)$ time construct a linear-space data structure answering two-dimensional orthogonal range search queries in* $O(\log k \log n + k' \log(n/k') +$

m) time, where m is the number of points returned and $k' \leq k$ depends on the query.

Observe that the above solution involves performing binary searches for the same keys in separate ordered lists. Thus, we can use the technique of fractional cascading [4] to speed up the query time and achieve the following result:

Theorem 2. *Given a set of n points which can be decomposed into k monotonic chains, we can in $O(n^3)$ time construct a linear-space data structure answering two-dimensional orthogonal range search queries in either $O(\log n + k + m)$ time or $O(\log k \log n + k' + m)$ time, where m is the number of points returned and $k' \leq k$ depends on the query.*

Proof. To check whether the query rectangle $[x_1, x_2] \times [y_1, y_2]$ intersects a given chain C_i, it is sufficient to perform binary searches on the list of x-coordinates (or y-coordinates) of the points on C_i using x_1 and x_2 (or y_1 and y_2) as search keys. This also finds which edge, if any, of C_i intersects each edge of the query rectangle. Therefore, we can report the points on C_i that are located in the query range in $O(\log n + k_i)$ time, where k_i is the number of such points.

Then to answer orthogonal range search queries using our data structure, we can perform two binary searches on the list of x-coordinates of the points on each chain, and two binary searches on the list of y-coordinates for each chain. Thus, we can store the sorted lists of x-coordinates and y-coordinates corresponding to the monotonically increasing chains separately, and use the technique of fractional cascading [4] to speed up the query time without increasing the asymptotic space cost of our data structure. We augment the data structure for the monotonically decreasing chains using the same approach. This yields a data structure of linear space that supports orthogonal range search in $O(\log n + k + m)$ time.

The other result in the theorem can be achieved by locating the start and the end of the range of chains that might intersect the query rectangle, and then using fractional cascading to compute the answer starting from the uppermost chain in this range. $\square$

The $O(n^3)$ preprocessing time may be improved to $O(n^{3/2})$ (matching the untangling step) in practical cases when the partitioning algorithm of Yang, Chen, Lu, and Zheng gives acceptable results [14]. We can also make the data structure of Lemma 6 implicit:

Corollary 1. *A set of n points in the plane can be arranged as an array of n coordinate pairs so that any orthogonal range query over this point set can be answered in $O(\log k \log n + k' \log(n/k') + m)$ time with $O(1)$ working space.*

5 Conclusions

We have presented a new data structure for two-dimensional orthogonal range search that is adaptive to the minimum number of monotonic chains that the input points can be partitioned into. For data which is considered easy in this sense, our data structure outperforms existing alternatives, either in query time or space requirements. Furthermore, we show that our structure can be made implicit, requiring only constant space in addition to the space required to encode the input points.

As a contribution of independent interest, we show how to partition a set of two-dimensional points into a minimal number of untangled monotonic chains. This decomposition is a key element of our data structure, and could also be useful in other geometric applications.

References

1. Arge, L., de Berg, M., Haverkort, H.J., Yi, K.: The priority R-tree: A practically efficient and worst-case optimal R-tree. ACM Transactions on Algorithms **4**(1) (2008)
2. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18**(9) (September 1975) 509–517
3. Bloniarz, P.A., Ravi, S.S.: An $\Omega(n \log n)$ lower bound for decomposing a set of points into chains. Information Processing Letters **31**(6) (1989) 319–322
4. Chazelle, B., Guibas, L.J.: Fractional cascading: I. a data structuring technique. Algorithmica **1**(2) (1986) 133–162
5. Di Stefano, G., Krause, S., Lübbecke, M.E., Zimmermann, U.T.: On minimum k-modal partitions of permutations. Journal of Discrete Algorithms **6**(3) (2008) 381–392
6. Fomin, F.V., Kratsch, D., Novelli, J.C.: Approximating minimum cocolorings. Information Processing Letters **84**(5) (December 2002) 285–290
7. Guttman, A.: *R*-trees: a dynamic index structure for spatial searching. SIGMOD Record (ACM Special Interest Group on Management of Data) **14**(2) (1984) 47–57
8. Kanth, K.V.R., Singh, A.: Optimal dynamic range searching in non-replicating index structures. In: ICDT'99. Volume 1540 of LNCS., Springer (1999) 257–276
9. Lueker, G.S.: A data structure for orthogonal range queries. In: 19th Annual Symposium on Foundations of Computer Science (FOCS '78), Long Beach, Ca., USA, IEEE Computer Society Press (October 1978) 28–34
10. Munro, J.I.: A multikey search problem. In: Proceedings of the 17th Allerton Conference on Communication, Control and Computing, University of Illinois (1979) 241–244
11. Nekrich, Y.: Orthogonal range searching in linear and almost-linear space. Computational Geometry **42**(4) (2009) 342–351
12. Supowit, K.J.: Decomposing a set of points into chains, with applications to permutation and circle graphs. Information Processing Letters **21**(5) (1985) 249–252
13. van Leeuwen, J., Schoone, A.A.: Untangling a traveling salesman tour in the plane. In: Proceedings of the 7th Conference on Graphtheoretic Concepts in Computer Science (WG 81), München, Germany, Hanser Verlag (1981) 87–98
14. Yang, B., Chen, J., Lu, E., Zheng, S.Q.: A comparative study of efficient algorithms for partitioning a sequence into monotone subsequences. In: TAMC 2007. Volume 4484 of LNCS., Springer (2007) 46–57