Dynamic Ham-Sandwich Cuts in the Plane*

Timothy G. Abbott† Michael A. Burr‡ Timothy M. Chan§
Erik D. Demaine† Martin L. Demaine† John Hugg¶
Daniel Kane‖ Stefan Langerman** Jelani Nelson†
Eynat Rafalin†† Kathryn Seyboth¶ Vincent Yeung†

Abstract

We design efficient data structures for dynamically maintaining a ham-sandwich cut of two point sets in the plane subject to insertions and deletions of points in either set. A ham-sandwich cut is a line that simultaneously bisects the cardinality of both point sets. For general point sets, our first data structure supports each operation in $O(n^{1/3+\varepsilon})$ amortized time and $O(n^{4/3+\varepsilon})$ space. Our second data structure performs faster when each point set decomposes into a small number k of subsets in convex position: it supports insertions and deletions in $O(\log n)$ time and ham-sandwich queries in $O(k \log^4 n)$ time. In addition, if each point set has convex peeling depth k, then we can maintain the decomposition automatically using $O(k \log n)$ time per insertion and deletion. Alternatively, we can view each convex point set as a convex polygon, and we show how to find a ham-sandwich cut that bisects the total areas or total perimeters of these polygons in $O(k \log^4 n)$ time plus the $O((kb) \operatorname{polylog}(kb))$ time required to approximate the root of a polynomial of degree $O(k)$ up to b bits of precision. We also show how to maintain a partition of the plane by two lines into four regions each containing a quarter of the total point count, area, or perimeter in polylogarithmic time.

1 Introduction

Finding a ham-sandwich cut is a well-studied problem with efficient solutions in many contexts; see, e.g., [Ede87, LMS94, Sto91]. In general, a *ham-sandwich cut* of two subsets S_1 and S_2 of the plane $\mathbb{R}^2$ is a line that simultaneously bisects both sets according to some measure μ. If S_1 and S_2 are discrete sets of points, the measure μ is usually the number of points; see Figure 1(a). If S_1

*Preliminary versions of this paper appeared in the *17th Canadian Conference on Computational Geometry* [ADD+05] and in the *15th Annual Fall Workshop on Computational Geometry* [BHR+05].

†Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar St., Cambridge, MA 02139, USA, {tabbott,edemaine,mdemaine,minilek,vshyeung}@mit.edu.

‡Courant Institute of Mathematical Sciences, New York University, 251 Mercer St., New York, NY 10012, USA, burr@cims.nyu.edu. Partially supported by NSF grant CCF-0431027.

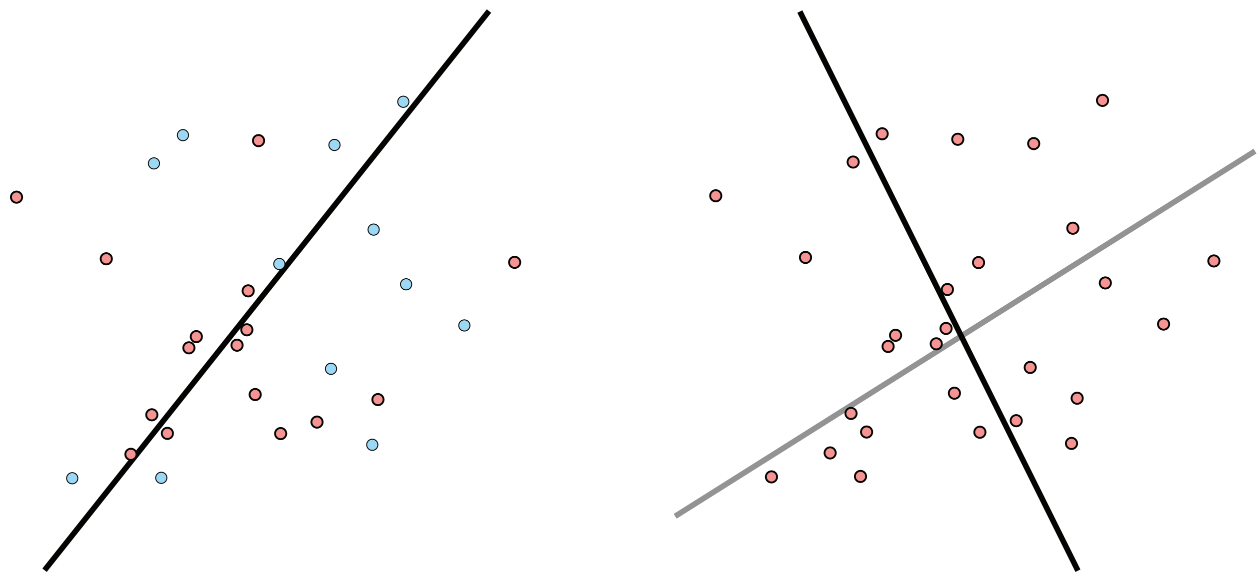
§School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, tmchan@uwaterloo.ca. Supported by NSERC.

¶Department of Computer Science, Tufts University, Medford, MA 02155, USA, {jhugg,kseyboth}@cs.tufts.edu. Partially supported by NSF grant CCF-0431027.

‖Department of Mathematics, Harvard University, 1 Oxford Street, Cambridge, MA 02139, USA, dankane@math.harvard.edu.

**Maître de recherches du F.R.S.-FNRS, Université Libre de Bruxelles, Département d'informatique, ULB CP212, Belgium. Stefan.Langerman@ulb.ac.be.

††Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA, eynat.rafalin@alumni.tufts.edu



(a) Ham-sandwich cut of two point sets.

(b) Two-line partition of a point set.

Figure 1: Examples of the geometric structures we wish to maintain dynamically.

and S_2 are regions, the measure μ could be area, perimeter, or the number of vertices (if S_1 and S_2 are polygonal).

A related problem, introduced by Megiddo [Meg85], is that of finding a two-line partition. A *two-line partition* of a subset S of the plane is a pair of lines that partition the plane into four regions ("quadrants") each containing a quarter of the total measure, $\frac{1}{4}\mu(S)$. Figure 1(b) shows an example for a discrete point set. As detailed in Section 2, the (static) problems of finding a ham-sandwich cut or two-line partition for given sets S_1 and S_2 are well studied, with linear-time solutions for most variations. One connection between this problem and ham-sandwich cuts is that each line in the partition is a ham-sandwich cut with respect to the 2-coloring induced by the other line in the partition.

While the problems of finding ham-sandwich cuts and two-line partitions are all well-understood when the subsets of the plane are given and static, nothing nontrivial is known for the problems of maintaining these structures for dynamically changing subsets of the plane. We initiate this study by giving the first sublinear data structures for maintaining ham-sandwich cuts and two-line partitions of dynamic point sets in the plane. We give two main data structures for this problem: the first considers arbitrary point sets, while the second optimizes for when the point set can be decomposed into a small number of subsets in convex position, in addition to bisecting area and perimeter of convex polygons.

Arbitrary point sets. Our *arbitrary-point-set data structure* maintains two planar point sets P_1 and P_2, of total size n, subject to the following two updates and two queries:

- Insert(p, i): Insert point p into P_i.
- Delete(p, i): Delete point p from P_i.
- Ham-sandwich cut: Find a ham-sandwich cut of P_1 and P_2.

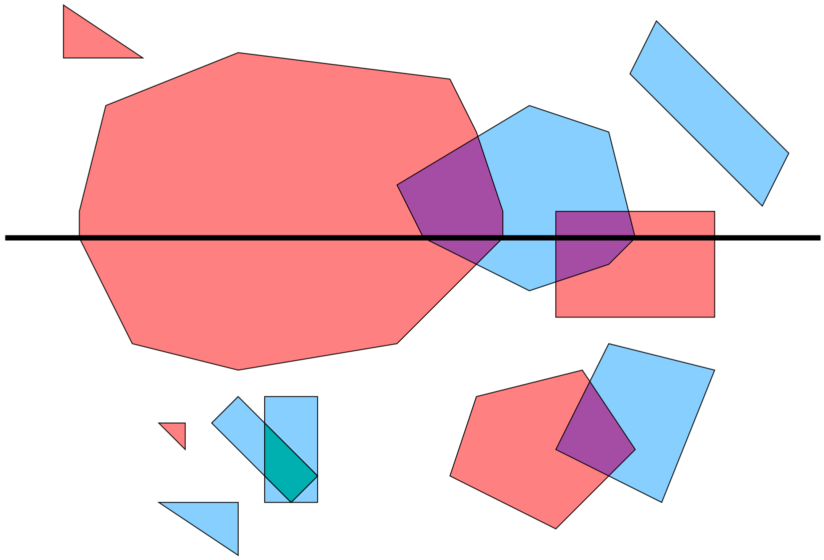


Figure 2: A ham-sandwich cut of several convex polygons of two colors. Darker shading indicates overlap between polygons.

- Two-line partition: Find a two-line partition of $P_1 \cup P_2$.

The data structure provides the following update-query trade-off: for any desired $U(n)$ with $1 \leq U(n) \leq n$, the data structure supports updates in $O^*(U(n))$ worst-case time and supports queries in $O^*(\sqrt{n/U(n)})$ amortized time, using $O^*(n\,U(n))$ space. In particular, if we set the query and update bounds to be equal, we obtain $O^*(n^{1/3})$ time per operation using $O^*(n^{4/3})$ space.

This data structure is simple in its idea but involves some sophisticated techniques. Specifically, it uses a range-counting data structure of Matoušek [Mat92a] and two levels of parametric search. The generality of this data structure is unmatched by our second data structure, which is tuned for special "decomposable" families of point sets (see Figure 2).

Convex pieces. Our *convex-pieces data structure* maintains k planar point sets $P_1, P_2, \ldots, P_k$, each in convex position, and of total size n, subject to the following four updates and two queries:

- Insert(p, i): Insert point p into P_i, provided this insertion maintains the invariant that P_i is in convex position.
- Delete(p, i): Delete point p from P_i.
- Split(i, j, ℓ): Split P_i into two sets P_i and P_j according to sideness with respect to line ℓ, overwriting any previous contents of P_i and P_j.
- Join(i, j): Join two linearly separable sets P_i and P_j, $i \neq j$, into one set P_i, provided this join maintains the invariant that P_i is in convex position, and empty P_j.
- Ham-sandwich cut$(b_1, b_2, \ldots, b_k, \mu)$: Find a ham-sandwich cut of $\bigcup\{P_i \mid b_i = 1\}$ and $\bigcup\{P_i \mid b_i = 2\}$ with respect to measure μ. (In other words, $b_i \in \{1, 2\}$ specifies the color of point set P_i.) The measure μ can specify vertex count, perimeter, or area; the latter two measures treat each P_i as a convex polygon.

- Two-line partition(μ): Find a two-line partition of $P_1 \cup P_2 \cup \cdots \cup P_k$ with respect to measure μ (with the same options as Ham-sandwich cut).

The data structure supports updates in $O(\log n)$ worst-case time, and supports queries in $O(k \log^4 n)$ worst-case time, using $O(n)$ space. When using the perimeter or area measure, the queries additionally require finding the roots of a polynomial of degree $O(k)$, which can be approximated up to b bits of precision in $O(kb \operatorname{polylog}(kb))$ additional time [Pan02, Pan97].[1] If desired, the user can also add or remove an empty point set, incrementing or decrementing the value of k; the time bounds depend on the current value of k. In addition, the user can specify a different measure μ_i for each set P_i, at no additional cost. A natural special case handled by this structure is when there is one convex point set (or equivalently, one convex polygon) of each color, so $k = 2$.

Another case of interest is when $P_1, P_2, \ldots, P_k$ form nested convex point sets. In this case, we obtain the *convex-hull peeling layers* or *onion peeling* [Bar76, Edd82] of the points $P_1 \cup P_2 \cup \cdots P_k$. The convex-pieces data structure can be adapted to handle this case specifically, implementing the interface of the arbitrary-point-set data structure and automatically dividing points into convex layers $P_1, P_2, \ldots, P_k$, using $O(k \log n)$ worst-case time per insertion or deletion. (This version of the data structure does not support split or join.) In this way, the data structure supports arbitrary sets of points, but the running time is fast only when k—the number of convex-hull peeling layers or *peeling depth*—is small. In the worst case, k can be $\Theta(n)$, but in many cases it may be smaller. For example, if the points are drawn uniformly at random from a disk, then $E[k] = \Theta(n^{2/3})$ [Dal04]. In this case, the convex-layers data structure is sublinear, albeit slower than the arbitrary-point-set data structure, but using less space.

More generally, the data structure can support the variation of the convex-layers interface in which $P_1, P_2, \ldots, P_{k'}$ are not constrained to be in convex position. Rather, each P_i can be maintained automatically according to its c_i convex-hull peeling layers. The same time bounds can then be obtained in terms of the total number k of convex sets, i.e., $k = \sum_{i=1}^{k'} c_i$, which is at most k' times the maximum peeling depth of any set. In this way, we remove the restrictions on convexity, but the performance degrades depending on how far the user deviates from convexity.

The convex-pieces data structure is simpler than the arbitrary-point-set data structure, using basic techniques such as balanced search trees and only one application of parametric search. In the simplest form, there are only two changes to the arbitrary-point-set data structure: (1) a different, simpler range-counting data structure, which additionally supports perimeter and area, and (2) additional support for updates that support convex peeling layers. In addition, we show how to avoid one use of parametric search in this case, using an accelerated simultaneous binary search.

2 Background

Early history. The earliest known reference about the existence of ham-sandwich cuts is by Steinhaus and others [S^+38], a Polish paper only recently translated to English [BZ04]. The paper credits Hugo Steinhaus for posing the ham-sandwich problem and credits Stefan Banach for first solving the problem via a reduction to the Borsuk-Ulam theorem (included in the paper). The version it considers is for three-dimensional solids, posed informally as "Can we place a piece of ham under a meat cutter so that meat, bone, and fat are cut in halves?" Stone and Tukey [ST42]

[1] The polynomial root approximation bounds measure the bit complexity of the computation; all other stated bounds are in the real RAM model of computation.

later generalized the result to arbitrary measure spaces. In computational geometry, the discrete case of a set of points is best known; see, e.g., [Ede87].

Existence proof. We give a short proof of the existence of ham-sandwich cuts in two dimensions, as our data structures follow the same basic principle.

First we show that any smooth bounded measure μ has a bisector line of any specified slope m. If we take a line of slope m very far down, then all of the measure μ will be above the line; symmetrically, if we take a line of slope m very high up, then all of the measure μ will be below the line. As we move the line continuously in between, keeping the slope fixed, the measure changes continuously. By the intermediate value theorem, some line in between bisects the measure μ exactly.

Second we prove that any two smooth bounded measures μ_1 and μ_2 have a simultaneous bisector line. Consider the bisector of μ_1 of a specified but varying slope m. If we take the slope m very near $+\infty$, then "above" the line essentially means to the left of the line; while if we take the slope m very near $-\infty$, then "above" the line essentially means to the right of the line. Therefore, whatever α fraction of μ_2 is above the μ_1 bisector in the first case, a $1-\alpha$ fraction of μ_2 will be above the μ_1 bisector in the second case. Varying the slope m between $-\infty$ and $+\infty$, the μ_1 bisector moves continuously. Again by the intermediate value theorem, some μ_1 bisector must also bisect μ_2 as desired.

Point sets. The same arguments apply to point sets in general position, because in this case the measures change by only ± 1 at once, so the intermediate value theorem still applies. To handle general point sets, we need to define the notion of bisection more carefully. Specifically, if ℓ^+ and ℓ^- denote the closed halfplanes on either side of a line ℓ, then ℓ is a bisector of measure μ if $|\mu(\ell^+) - \mu(\ell^-)| \leq \mu(\ell)$. Bose and Langerman [BL04] use this definition and show that it handles weighted points, even with negative weights. For positive weights, the definition is equivalent to a simpler statement: if ℓ^+ and ℓ^- denote the open halfplanes on either side of a line ℓ, then ℓ is a bisector of measure μ if $\mu(\ell^+) \leq \frac{1}{2}\mu(S)$ and $\mu(\ell^-) \leq \frac{1}{2}\mu(S)$. This definition extends easily to two-line partitions as well.

Algorithms. Many algorithms are known for computing ham-sandwich cuts. Lo et al. [LMS94] give an optimal $O(n)$-time algorithm for finding a ham-sandwich cut of two point sets of total size n. Bose and Langerman [BL04] give an $O(n \log n)$-time algorithm when the points have weights (positive or negative). Bose et al. [BDH$^+$04] give an $O(n \log k)$-time algorithm for the case in which the n points and (geodesic) cut are confined within a simple polygon with at most n vertices and k reflex vertices. Stojmenović [Sto91] gives an $O(n)$-time algorithm for finding a ham-sandwich cut that bisects the area of two convex polygons with n vertices total. Díaz and O'Rourke [DO90] give an $O(n b_{\text{in}} b_{\text{out}} \log n)$-time algorithm for the more general case of two simple polygons, where b_{in} and b_{out} denote the desired bit complexity of the input and output, respectively. None of these solutions support sublinear updates as the inputs change.

Two-line partition. Using the existence of ham-sandwich cuts, it is easy to see the existence of a two-line partition of a measure μ [Meg85]. We first bisect the measure μ, and then we find the ham-sandwich cut of the measures on either side of this bisector. The bisector and the ham-sandwich cut serve as a two-line partition. This approach can be used to convert any algorithm for computing ham-sandwich cuts into an algorithm for two-line partitions. However, this transformation does not necessarily apply to data structures, because they may have restrictions on how quickly points can be recolored as red or blue.

3 Arbitrary-Point-Set Data Structure

This section presents a solution to the dynamic ham-sandwich cut problem for two general point sets P_1 and P_2 in the plane.

3.1 Data Structure

The data structure we maintain is (a variation of) a known data structure for simplex range counting, Matoušek's *partition trees* [Mat92a]. In two dimensions, and for any parameter s between n and n^2, we can construct a partition-tree data structure using $O^*(s)$ space and preprocessing that allows us to count the number of points of P_1 (or P_2) inside any given triangle (and in particular any given halfplane) in $O^*(n/\sqrt{s})$ time.

In addition to this basic result, we need two additional features that follow from simple modifications. Similar observations have been made by Agarwal and Matoušek [AM93] as well.

First, the partition-tree data structure can be made dynamic, to support insertions and deletions of points in P_1 (or P_2) in $O^*(s/n)$ amortized time per operation. Although dynamization is explicitly stated only in one case (when $s = n$) in Matoušek's paper [Mat92a], Agarwal and Matoušek [AM95] provide a complete dynamization of another data structure of Matoušek [Mat92b] for halfspace range reporting, and the same techniques carry over to simplex range searching. (In fact, the dynamization is slightly simpler for simplex range searching, because "shallowness" [Mat92b] does not come into play.)

Second, the query algorithm of partition trees can be parallelized efficiently to run in $O(\log n)$ time with $O^*(n/\sqrt{s})$ processors. This parallelization is straightforward, by descending from each level to the next level of the partition tree in parallel. This parallel bound will be essential in our subsequent applications of parametric search.

3.2 Bisector Query

As a first step toward computing a ham-sandwich cut, we consider how to find a bisector of one of the point sets with a specified slope.

Our algorithm uses Megiddo's parametric search technique [Meg83]. For an unknown real value x^*, this technique transforms a parallel algorithm (in the algebraic decision tree model) for deciding whether x^* is at most a given threshold x into a sequential algorithm for computing x^*. If the parallel decision algorithm runs in T_P time on P processors, whose total work is T_1, then the running time of the resulting algorithm is $O((P + T_1 \log P)T_P)$, for any desired value P. Furthermore, for a restricted class of parallel algorithms satisfying a certain "bounded fan-out" property, a refined technique by Cole [Col87] improves the running time to $O((P + T_1)T_P + T_1 \log P)$.

In addition, because our bisection algorithm will be used in another level of parametric search, we need to develop a parallel bisection algorithm. Megiddo [Meg83] describes a parallel version of parametric search for precisely such "second-order" applications. Specifically, the resulting algorithm runs in $O(T_P^2 \log P)$ time on P processors.

Proposition 1 *Given a slope m, we can find a bisector of P_1 (or P_2) of slope m in $O^*(n/\sqrt{s})$ sequential time or in $O(\log^3 n)$ parallel time with $O^*(n/\sqrt{s})$ processors.*

Proof: Let b^* denote the unknown y intercept of the bisector of slope m. First we solve the decision problem: given a value b, test whether $b^* \le b$. To this end, we count the number of points of P_1 below the line with slope m and intercept b. This count corresponds to a halfplane range-counting query and thus partition trees can answer it in $T_1 = O^*(n/\sqrt{s})$ sequential time or $T_P = O(\log n)$ parallel time with $P = O^*(n/\sqrt{s})$ processors. The answer to the decision problem is "yes" ($b^* \le b$) precisely if the count is at least $|P_1|/2$.

Now we apply parametric search to compute b^*. In the sequential version, we obtain a running time of $O([P + T_1 \log P]T_P) = O^*([n/\sqrt{s} + (n/\sqrt{s}) \log(n/\sqrt{s})] \log n) = O^*(n \log^2 n/\sqrt{s}) = O^*(n/\sqrt{s})$. (Cole's refined technique applies here but the logarithmic improvement would disappear in the O^* notation.) In the parallel version, we obtain a running time of $O(T_P^2 \log P) = O(\log^2 n \log(n/\sqrt{s})) = O(\log^3 n)$ on P processors. □

3.3 Ham-Sandwich Cut

Now we are ready to describe the algorithm for finding a ham-sandwich cut. This algorithm uses a second level of parametric search, building on top of the bisector algorithm. However, because the solution is not necessarily unique, the application of parametric search is a little less conventional, so we provide more details about the parametric search.

Theorem 1 *There is a data structure that maintains two point sets of total size n subject to insertion and deletion of points in $O^*(s/n)$ amortized time and subject to queries for a ham-sandwich cut in $O^*(n/\sqrt{s})$ time using $O^*(s)$ space.*

Proof: Let m^* denote the unknown slope of some ham-sandwich cut. We maintain an interval $[m_a, m_b]$, satisfying the invariant that the slope-m_a bisector of P_1 is above the slope-m_a bisector of P_2 but the slope-m_b bisector of P_1 is below the slope-m_b bisector of P_2, or vice versa. By a continuity argument (as in Section 2), we know that there is a solution with a slope in the interval $[m_a, m_b]$. Initially we set $[m_a, m_b] = [-\infty, \infty]$.

We simulate the parallel algorithm from Proposition 1 with T_P parallel steps and P processors, on an unknown slope m^*, first for the point set P_1 and then for P_2. In each parallel step, we need to resolve comparisons of m^* with $O(P)$ values (roots of fixed-degree polynomials). These comparisons can be resolved by performing a binary search over these values (using median finding). In this binary search, when comparing m^* with a value m, we call the sequential algorithm from Proposition 1 with running time T_1 to determine the slope-μ bisectors of P_1 and P_2. We know that at least one of the two subintervals $[\mu_a, \mu]$ and $[\mu, \mu_b]$ still satisfies the invariant, and we modify $[\mu_a, \mu_b]$ to be this subinterval. In the former case, we report that $\mu^* < \mu$; in the latter case, we report that $\mu^* > \mu$. The binary search requires $O(\log P)$ actual comparisons. Thus, all comparisons in each parallel step can be resolved in $O(P + T_1 \log P)$ time. The total time is therefore $O((P + T_1 \log P)T_P)$. In our case, $T_1 = P = O^*(n/\sqrt{s})$ and $T_P = O(\log^3 n)$, yielding the final time bound of $O^*(n/\sqrt{s})$.

At the end of the simulation for both point sets P_1 and P_2, we have identified a point $p_1 \in P_1$ and a point $p_2 \in P_2$ that define the slope-μ bisectors of P_1 and P_2, respectively, for all m inside the final interval $[\mu_a, \mu_b]$. Because a solution exists for some slope inside this interval, we know that some ham-sandwich cut must be defined by both p_1 and p_2, so we are done. □

A similar parametric search appears in an algorithm for ham-sandwich cuts by Cole, Sharir, and Yap [CSY87].

3.4 Two-Line Partition

Because this data structure supports only the point-counting measure μ, it is relatively straightforward to extend the data structure for ham-sandwich cuts to a data structure for two-line partitions of a point set P. Specifically, we maintain the invariant that one of the cuts is a vertical line at the median x coordinate among points in P, and that the points in P are partitioned into P_1 and P_2 according to whether they are left or right of this vertical line. This invariant is easy to maintain: each insertion or deletion on P translates into a constant number of insertions and deletions on P_1 and P_2. Now a second bisector can be obtained simply by computing a ham-sandwich cut with respect to P_1 and P_2, which we have shown how to do. Therefore, in the same time and space bounds, we can maintain two-line partitions of a dynamic point set P.

4 Convex-Pieces Data Structure

The convex-pieces data structure represents each convex polygon P_i by two augmented balanced binary search trees on the polygon edges, one for the upper chain and one for the lower chain, each ordering the edges in counterclockwise order. (In this section, we use the notation P_i to denote both a point set and the induced convex polygon.) The upper and lower chains are defined by their common endpoints of minimum and maximum x coordinate. We use a balanced binary search tree that supports insertion, deletion, search, split, and concatenate in $O(\log n)$ time per operation, such as red-black trees [CLRS01, ch. 13], and for simplicity we view the data as being stored in the leaves.

With each edge (p, q) of a convex polygon P_i, we store three measures: (1) the signed area of the trapezoid defined by p, q, and the projections of p and q onto the x axis; (2) the length of the line segment from p to q; and (3) the number 1. In (1), *signed area* measures the area of the portion of the trapezoid above the x axis minus the area of the portion below the x axis, for edges on the upper hull, and the negation of this difference for edges on the lower hull (i.e., edges pointing right), following [IL00, CCAU98]. Each node x of a binary search tree, which represents a subchain of P_i corresponding to the descendant leaves, maintains three subtree sums, one for each measure. From this information we can compute the measure of any subchain of a convex polygon P_i, in $O(\log n)$ time, by adding the sums from the corresponding $O(\log n)$ subtrees.

4.1 Updates

First we describe how to maintain this data structure subject to insertions, deletions, splits, and joins according to the second interface described in Section 1, where the user specifies which convex set P_i should be updated. All operations can be supported in $O(\log n)$ worst-case time.

The simplest operation to implement is Delete(p, i): we delete the point p from the one or two trees containing it, in $O(\log n)$ time. Two trees contain p if p happens to be an endpoint of the upper and lower chains; in this case, we must also add the new extreme point (either leftmost or rightmost) to the other chain. These changes require updating the measures of $O(1)$ edges, and the subtree sums can be propagated in $O(\log n)$ time. During rebalancing, we can maintain subtree sums by adding $O(1)$ time to the cost of a rotation; thus this information can be maintained with a constant-factor overhead.

Next consider inserting a new point p in a convex polygon P_i, with the property that the resulting vertex set $P_i \cup \{p\}$ remains in convex position. With a sidedness test between p and the line connecting the two endpoints of the upper and lower chains, we can determine whether p should be added to the upper or lower chain. Then we simply insert p into the binary search tree

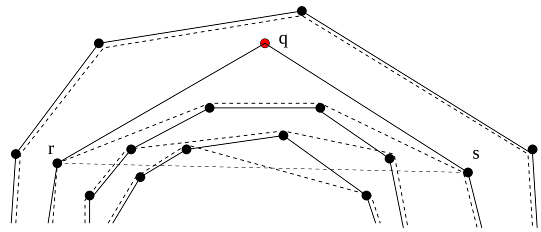


Figure 3: Insertion and deletion of point q and its affect on the convex-hull peeling-layers. The layers with point q are drawn as solids lines while the layers without q are drawn as dashed lines. Point q has depth 2, and its insertion or deletion affects only layers of depth ≥ 2.

representing that chain, preserving the sorted order of the points by x coordinates. If p turns out to be a new extreme (minimum or maximum) x coordinate, then we insert p as a new endpoint into the other chain as well, and remove the old endpoint from the chain that we first inserted p into.

Join(i, j) can be viewed as a generalization of Insert: instead of adding one point to a convex polygon P_i, we now add an entire convex chain P_j to the polygon P_i. To find the edge in P_i to be deleted, we find where any point of P_j would be inserted, as above; similarly, we also find the edge in P_j that would be deleted if any point of P_i would be inserted. Now we are left with two open chains which can be glued into one closed convex polygon, using the $O(\log n)$-time split and concatenate operations provided by the binary search trees representing the upper and lower chains.

The inverse operation Split(i, j, ℓ) is similar. In $O(\log n)$ time, we can find the two edges of the convex polygon P_i intersected by the line ℓ [O'R98]. Then we can partition the upper and lower chains appropriately using the $O(\log n)$ split and concatenate operations provided by the binary search trees.

4.2 Convex-Hull Peeling Layers

Next we describe how to extend our data structure to allow each point set P_i to be in nonconvex position, by automatically maintaining a decomposition of P_i into convex-hull peeling layers. Specifically, we let $P_{i,1}, P_{i,2}, \ldots, P_{i,c_i}$ denote the convex peeling layers of P_i, where $P_{i,1}$ is outermost. The update operations Insert and Delete refer to the overall set P_i, and the data structure automatically maintains the convex peeling layers. The running time of the operations will increase somewhat, to $O(k \log n)$ worst-case time per operation where $k = \sum_{i=1}^{k'} c_i$ is the total number of convex layers. The key property we use is that, when inserting or deleting points, the convex-hull peeling layers change by moving entire intervals between adjacent layers; see Figure 3. Both insertions and deletions affect the layer containing the input point and possibly all more deeply nested layers, but affect none of the shallower layers.

To insert a point p into P_i, we first find the two adjacent layers $P_{i,j-1}$ and $P_{i,j}$ such that p is interior to the polygon $P_{i,j-1}$ but not interior to the polygon $P_{i,j}$. These layers are easy to find in $O(\log c_i \log n)$ time by binary searching on j, and at each step j of the binary search, spending $O(\log n)$ time to decide whether p is interior to $P_{i,j}$. Now we enter a general recursion in which we wish to insert a convex chain of points $p_1, p_2, \ldots, p_r$ (initially, consisting of just a single point p) into layer $P_{i,j}$. We also have as an invariant of the recursion that this convex chain either consists of a single point or it used to belong to the next outer layer $P_{i,j-1}$. Thus we know that the chain's tangents extending edges $p_1 p_2$ and $p_{r-1} p_r$ do not intersect $P_{i,j}$. Hence the two bridges (common tangents) between the to-be-inserted convex chain and $P_{i,j}$ pass through p_1 and p_r, respectively.

We can find each of these bridges in $O(\log n)$ using a binary search, at each stage performing a sidedness test between the chain endpoint and the line extending an edge of $P_{i,j}$. If we find that these tangents to $P_{i,j}$ actually intersect the to-be-inserted chain $p_1, p_2, \ldots, p_r$ (in addition to passing through p_1 or p_r), then the convex polygon $p_1, p_2, \ldots, p_r$ actually contains $P_{i,j}$. In this case, we define this polygon as a new layer $P'_{i,j}$ and increment the layer number j of all layers nested within, including the old $P_{i,j}$. Otherwise, we have actual bridges and, using the $O(\log n)$-time split and concatenate operations, we can cut out the portion of $P_{i,j}$ strictly between the two bridge endpoints of $P_{i,j}$, and splice in the to-be-inserted chain $p_1, p_2, \ldots, p_r$ (itself represented by a balanced binary search tree). Then, if the cut-out chain has at least one point, we recursively insert it into the next layer, $P_{i,j+1}$. In the base case, the layer $P_{i,j+1}$ is empty, in which case we trivially add the convex chain to the layer. The total time spent by the recursion to update $P_{i,j}, P_{i,j+1}, \ldots, P_{i,c_i}$ is $O(c_i \log n)$.

Deleting a point p from P_i reduces to insertion. We find the layer $P_{i,j}$ to which p belongs in $O(\log c_i \log n)$ time. Then we delete p and its incident edges from this layer, leaving an open chain, and insert this chain into the next deeper layer $P_{i,j+1}$ as above. Finally, we renumber the layer numbers to use $P_{i,j}$.

This concludes the description of how to maintain convex-hull peeling layers. This maintenance affects updates but not queries. For the purposes of uniformly describing the queries, we assume henceforth that the P_i's are convex sets; in the convex-hull-peeling data structure, this notation in fact refers to the $P_{i,j}$ layers.

4.3 Basic Queries

In preparation for ham-sandwich cuts, we describe two basic queries that form necessary subroutines: range counting (or more accurately, range measurement) and bisection.

Proposition 2 *Given an oriented line ℓ, a desired subset $\mathcal{P}$ of $\{P_1, P_2, \ldots, P_k\}$, and a measure μ of vertex count, perimeter, or area, we can compute the measure of the portion of $\bigcup \mathcal{P}$ left of ℓ in $O(k \log n)$ sequential time or $O(\log n)$ parallel time on k processors.*

Proof: We can consider each convex polygon $P_i \in \mathcal{P}$ separately and add up the computed measures. In $O(\log n)$ time, we can find the two edges e_1, e_2 of the convex polygon P_i intersected by the line ℓ [O'R98], as well as the points of intersection. Here we label e_1 and e_2 in the order they are intersected by the oriented line ℓ. Then we compute the sum of the measures of the interval of edges clockwise from e_1 to e_2, including both e_1 and e_2, in $O(\log n)$ using the appropriate subtree sums in the binary search tree. For vertex count, we subtract 1 from this sum to count the number of vertices strictly between e_1 and e_2. For perimeter, we subtract off the length of the portions of e_1 and e_2 on the right of oriented line ℓ. For area, we follow the ideas of [IL00, CCAU98]. We subtract off the area of the trapezoid defined by the two points of intersection between ℓ and P_i and their two projections onto the x axis. We also subtract off the area under e_1 and e_2 right of ℓ. The result is the desired area of the portion of P_i left of ℓ.

In all cases, we spend $O(\log n)$ time per convex polygon P_i, for a total of $O(k \log n)$ time. With k processors, we can process each convex polygon in parallel, and then sum the answers in $O(\log k) = O(\log n)$ time. □

Langerman [Lan03] proves that, in the worst case, any data structure, even static, supporting range measurement queries as in Proposition 2 requires $\Omega(k)$ time per query in the case of perimeter and area measures. While this lower bound does not extend to the problems of bisection and ham-sandwich cuts, it limits the running times we can expect from any data structure based on range measurement as a foundation.

Proposition 3 *Given a slope m, a desired subset $\mathcal{P}$ of $\{P_1, P_2, \ldots, P_k\}$, and a measure μ of vertex count, perimeter, or area, we can find the edges of each $P_i \in \mathcal{P}$ intersected by a bisector of $\bigcup \mathcal{P}$ of slope m in $O(k \log^2 n)$ sequential time or $O(\log^2 n)$ parallel time on k processors.*

Proof: The algorithm is a global binary search over all vertices of polygons P_i in $\mathcal{P}$, with a total of $O(\log n)$ rounds. In general, we suppose we have a range $[b_1, b_2]$ of (y) intercepts, such that there is a bisector of slope m with intercept in the range. Initially, $[b_1, b_2] = [-\infty, +\infty]$.

The main challenge in a step of the binary search is to find a good "candidate intercept" b in the range $[b_1, b_2]$. Let R denote the strip of lines of slope m and with intercept in the range $[b_1, b_2]$. For each convex polygon P_i, we compute a median point q_i in $P_i \cap R$ with respect to the intercept, i.e., a point q_i such that the line of slope m passing through q_i roughly bisects the points of P_i contained in the strip R. Such a median point can be computed in $O(\log n)$ time using an algorithm for computing the median of the union of two sorted arrays [CLRS01, Ex. 9.3-8, p. 193]; here, the two arrays correspond to subchains of the upper and lower chains of P_i. We also define the *weight* w_i of the median point q_i to be the number of points in $P_i \cap R$, which again can be computed in $O(\log n)$ time. Now we compute a *weighted median* q_j of $q_1, q_2, \ldots, q_k$, i.e., a point q_j such that the total weight of points q_i with intercept smaller than q_j's intercept, and similarly the total weight of points q_i with intercept larger than q_j's intercept, are both at most half the total weight. Such a weighted median can be computed in $O(k)$ time [CLRS01, Prob. 9-2, p. 194], but for our purposes it suffices to just sort the q_j's by intercept and scan the array until at most half the weight is on either side, using $O(k \log k)$ time.

Now we apply Proposition 2 to compute the measure of $\bigcup \mathcal{P}$ left of the line with slope m and intercept b, using $O(k \log n)$ time. If the measure happens to be half of the total measure $\mu(\bigcup \mathcal{P})$ (which we can compute once at the beginning), then we have the desired bisector. Otherwise, the measure left of the line is either larger or smaller than half the total measure. If it is larger, we can narrow our intercept interval to $[b, b_2]$; if it is smaller, we can narrow our intercept interval to $[b_1, b]$. In either case, we eliminate roughly half of the points from the polygons P_i whose median point q_i has intercept either smaller or larger than q_j, including P_j and q_j itself. Together, these q_i's constitute at least half of the weight, so we eliminate at least roughly a quarter of the points from $\bigcup \mathcal{S} \cap R$. Therefore, the total running time of the binary search is $O(k \log^2 n)$.

Using k processors, we can compute the median point q_i and its weight w_i for each polygon P_i in parallel. We can compute the weighted median in $O(\log k)$ time by sorting by intercept, computing prefix sums on the weights, and then binary searching for the weighted median. This cost is dominated by the $O(\log n)$ cost to compute each q_i. Thus the total parallel running time is $O(\log^2 n)$. □

Our sequential algorithm in Proposition 3 is similar to an algorithm for selection among multiple sorted arrays [FJ82], except that we pay an extra logarithmic factor for using trees instead of arrays. We believe that this logarithmic overhead can be removed using weight-balanced trees and a careful implementation of [FJ82], but have not verified the details. A somewhat weaker result could also be obtained simply by applying parametric search, as with Proposition 1. The time bounds would then be a factor of $O(\log k)$ worse: $O(k \log k \log^2 n)$ sequential and $O(\log k \log^2 n)$ parallel on k processors.

4.4 Ham-Sandwich Cut

Now we turn to one of the main queries of interest, ham-sandwich cuts.

Theorem 2 *There is a data structure that maintains k convex point sets with n points total subject to insertion and deletion of vertices in $O(\log n)$ worst-case time and subject to queries for a ham-sandwich cut in $O(k \log^4 n)$ worst-case time plus, in the case of area or perimeter measures, $O((kb)\log(kb))$ time to approximate the roots of a polynomial of degree $O(k)$ up to b bits of precision.*

Proof: We use parametric search as in Theorem 1, using the bisector subroutine from Proposition 3. Thus, $T_1 = O(k \log^2 n)$, $T_P = O(\log^2 n)$, and $P = k$. Therefore, the running time is $O((P + T_1 \log P)T_P) = O((k + k \log^2 n \log k) \log^2 n) = O(k \log k \log^4 n)$. Cole's refined technique [Col87] applies here because our parallel algorithm satisfies the bounded fan-out property: each comparison influences only a constant number of comparisons at the next parallel step. With this technique, the running time improves to $O((P + T_1)T_P + T_1 \log P) = O((k + k \log^2 n) \log^2 n + k \log^2 n \log k) = O(k \log^4 n)$.

For the area and perimeter measures, we need some additional care. Proposition 3 determines the edges of the polygons intersected by the bisector of a given slope. Langerman [Lan03] shows that the perimeter or area of the k polygons on one side of the bisecting line can be written as a ratio of two polynomials, where the numerator is of degree 2 in the intercept and degree $O(k)$ in the slope, and where the denominator depends only on the slope and is of degree $O(k)$. Thus, during the parametric search, given the current guess of the slope, we can compute the intercept using the quadratic formula. At the end of the algorithm, though, we need to solve for the slope as well, which requires solving two polynomials of degree $O(k)$, which is equivalent to one polynomial of degree $O(k)$. Here we use polynomial root-finding algorithms [Pan02, Pan97] which compute b bits of precision in $O((kb)\log(kb))$ time, measured as bit computations. □

4.5 Two-Line Partition

Recall that the two-line partition of a set S is a pair of lines dividing the plane into quadrants each containing equal measure $\frac{1}{4}\mu(S)$. In the convex-layers data structure, S is defined to be $P_1 \cup P_2 \cup \cdots P_k$. We show that the same ham-sandwich data structure can be used to find two-line partitions as well, in $O(k \log^4 n)$ time.

To find a two-line partition of S, we first find an arbitrary bisecting line ℓ of S, in $O(k \log^2 n)$ time by Proposition 3. This line ℓ defines a 2-coloring of the points in S. We form this 2-coloring by splitting each set P_i according to the line ℓ—Split$(i, k+i, \ell)$ for each i—which costs $O(k \log n)$ time.[2] Then we make a ham-sandwich query with the 2-coloring $b_1, b_2, \ldots, b_{2k} = \underbrace{1, 1, \ldots, 1}_{k}, \underbrace{2, 2, \ldots, 2}_{k}$ defined by the side of the split, which costs $O(k \log^4 n)$ time. The ham-sandwich cut, together with the line ℓ, define a two-line partition. We can restore the original sets either by calling Join$(i, k+i)$ for each i, or by undoing the (logged) changes made by the split. The total time required is $O(k \log^4 n)$, dominated by the ham-sandwich cut.

[2]We can perform the split even in the case of convex-peeling layers, simply by cutting each layer individually; we do not need to compute the ramifications of the splits on the convex peeling because we will later undo the changes.

5 Conclusion

Our results give one of the first dynamic data structures for maintaining ham-sandwich cuts in sublinear time per update. Ham-sandwich cuts can be generalized in many directions, as described in Section 2, and it would be interesting to consider dynamic data structures for these generalizations. Can we support weighted points, or bisecting the area of nonconvex polygons, or geodesic cuts within a polygon? What about point sets in higher (fixed) dimensions?

Acknowledgments

This work began at an open-problem session organized as part of the MIT Advanced Data Structures class (6.897) in Spring 2005. The authors thank the other participants of that session—Brian Dean, Nick Harvey, Pramook Khungurn, Michael Lieberman, Mihai Pătraşcu, and Yoyo Zhou—for helpful discussions and a stimulating environment. We also thank the anonymous referees for helpful comments.

References

[ADD+05] Timothy Abbott, Erik D. Demaine, Martin L. Demaine, Daniel Kane, Stefan Langerman, Jelani Nelson, and Vincent Yeung. Dynamic ham-sandwich cuts of convex polygons in the plane. In *Proceedings of the 17th Canadian Conference on Computational Geometry*, pages 61–64, Windsor, Canada, August 2005.

[AM93] Pankaj K. Agarwal and Jiří Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993.

[AM95] P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995.

[Bar76] V. Barnett. The ordering of multivariate data. *Journal of the Royal Statistical Society, Series A*, 139(3):318–355, 1976. With a discussion by R. L. Plackett, K. V. Mardia, R. M. Loynes, A. Huitson, G. M. Paddle, T. Lewis, G. A. Barnard, A. M. Walker, F. Downton, P. J. Green, Maurice Kendall, A. Robinson, Allan Seheult, and D. H. Young.

[BDH+04] Prosenjit Bose, Erik D. Demaine, Ferran Hurtado, John Iacono, Stefan Langerman, and Pat Morin. Geodesic ham-sandwich cuts. In *Proceedings of the 20th Annual ACM Symposium on Computational Geometry*, pages 1–9, Brooklyn, New York, June 2004.

[BHR+05] Michael A. Burr, John Hugg, Eynat Rafalin, Kathryn Seyboth, and Diane L. Souvaine. Dynamic ham-sandwich cuts for two point sets with bounded convex-hull-peeling depth. Technical Report TR-2005-7, Department of Computer Science, Tufts University, November 2005. Presented at the *15th Annual Fall Workshop on Computational Geometry and Visualization*, Philadelphia, PA, November 2005.

[BL04] Prosenjit Bose and Stefan Langerman. Weighted ham-sandwich cuts. In *Revised Papers from the Japan Conference on Discrete and Computational Geometry*, volume 3742 of *Lecture Notes in Computer Science*, pages 48–53, Tokyo, Japan, October 2004.

[BZ04] W. A. Beyer and Andrew Zardecki. The early history of the ham sandwich theorem. *Amer. Math. Monthly*, 111(1):58–61, 2004.

[CCAU98] J. Czyzowicz, F. Contreras-Alcalá, and J. Urrutia. On measuring areas of polygons. In *Proceedings of the 10th Canadian Conference on Computational Geometry*, 1998.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[Col87] Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, January 1987.

[CSY87] Richard Cole, Micha Sharir, and Chee-K. Yap. On k-hulls and related problems. *SIAM J. Comput.*, 16(1):61–77, 1987.

[Dal04] Ketan Dalal. Counting the onion. *Random Structures & Algorithms*, 24(2):155–165, 2004.

[DO90] Matthew Díaz and Joseph O'Rourke. Ham-sandwich sectioning of polygons. In *Proceedings of the 2nd Canadian Conference on Computational Geometry*, pages 282–286, 1990.

[Edd82] William F. Eddy. Convex hull peeling. In H. Caussinus and P. Ettinger, editors, *COMPSTAT 1982: Proceedings in Computational Statistics, Part 1*, pages 42–47, Vienna, 1982. Physica-Verlag.

[Ede87] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *Monographs in Theoretical Computer Science*. Springer, 1987.

[FJ82] Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in $X+Y$ and matrices with sorted columns. *Journal of Computer and System Sciences*, 24(2):197–208, April 1982.

[IL00] John Iacono and Stefan Langerman. Volume queries in polyhedra. In *Revised Papers from the Japan Conference on Discrete and Computational Geometry*, volume 2098 of *Lecture Notes in Computer Science*, pages 156–159, Tokyo, Japan, November 2000.

[Lan03] Stefan Langerman. The complexity of halfspace area queries. *Discrete & Computational Geometry*, 30(4):639–648, 2003.

[LMS94] Chi-Yuan Lo, J. Matoušek, and W. Steiger. Algorithms for ham-sandwich cuts. *Discrete Comput. Geom.*, 11(4):433–452, 1994.

[Mat92a] Jiří Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8(3):315–334, 1992.

[Mat92b] Jiří Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, 2(3):169–186, 1992.

[Meg83] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the Association for Computing Machinery*, 30(4):852–865, 1983.

[Meg85] Nimrod Megiddo. Partitioning with two lines in the plane. *J. Algorithms*, 6(3):430–433, 1985.

[O'R98] Joseph O'Rourke. Stabbing a convex polygon (section 7.9.1). In *Computational Geometry in C*, pages 271–272. Cambridge University Press, second edition, 1998.

[Pan97] Victor Y. Pan. Solving a polynomial equation: some history and recent progress. *SIAM Review*, 39(2):187–220, 1997.

[Pan02] Victor Y. Pan. Univariate polynomials: nearly optimal algorithms for numerical factorization and root-finding. *Journal of Symbolic Computation*, 33(5):701–733, 2002. Computer algebra (London, ON, 2001).

[S+38] Hugo Steinhaus et al. A note on the ham sandwich theorem. *Mathesis Polska*, 9:26–28, 1938.

[ST42] A. H. Stone and J. W. Tukey. Generalized "sandwich" theorems. *Duke Mathematical Journal*, 9:356–359, 1942.

[Sto91] Ivan Stojmenović. Bisections and ham-sandwich cuts of convex polygons and polyhedra. *Information Processing Letters*, 38(1):15–21, 1991.