Dynamic Ham-Sandwich Cuts for Two Overlapping Point Sets

M. A. Burr* J. Hugg* E. Rafalin* K. Seyboth* D. L. Souvaine*

Abstract

We provide an efficient data structure for dynamically maintaining a ham-sandwich cut of two *overlapping* point sets in convex position in the plane. The ham-sandwich cut of S_1 and S_2 is a line that simultaneously bisects the area, perimeter or vertex count of both point sets. Our algorithm supports insertion and deletion of vertices in $O(\log n)$ time, and area, perimeter and vertex-count queries in $O(\log^3 n)$ time, where n is the total number of points of $S_1 \cup S_2$.

An extension of the algorithm for sets with a bounded number c of convex hull peeling layers enables area and perimeter queries, using $O(c \log n)$ time for insertions and deletions and $O(\log^3 n)$ query time.

Our algorithm improves previous results [15, 1]. The *static* method of Stojmenović [15] and the dynamic algorithm of Abbott *et al.* [1] maintain a ham-sandwich cut of two *disjoint convex polygons* in the plane. Our dynamic algorithm removes the restrictions based on the separation of the points. It also solves an open problem from 1991 about finding the area and perimeter ham-sandwich cuts for overlapping convex point sets in the static setting [15].

1 Introduction

The *ham-sandwich cut* of two subsets S_1 and S_2 in the plane is a line that simultaneously bisects both sets according to some measure (for example the vertex count, area, or perimeter). The static problem of finding a ham-sandwich cut of a point set is well studied, with a linear time solution [9, 12, 11]. Other variations exist [15, 3].

The ham-sandwich cut problem is closely related to the problem of finding a *two-line partition* of a subset S of the plane [13]: a pair of lines that partition the plane into four regions each containing a quarter of the total measure. A line in the two-line partition "is a ham-sandwich cut with respect to the two coloring induced by the other line" [1].

Data depth is a statistical analysis method that assigns a numeric value to a point corresponding to its centrality relative to a data set. The *half-space depth* [10, 16] (in the literature sometimes called *location depth* or *Tukey depth*) of a point x relative to a set of points $\mathcal{S} = \{X_1, ..., X_n\}$ is the minimum number of points of $\mathcal{S}$ lying in any closed half-space determined by a line through x. The two-line partition can approximate the point of maximal half-space depth (the *Tukey median*): the intersection point of a two-line partition of the data set is a point of half-space depth $n/4$ and will approximate the Tukey median by a factor of 2.

Much work has been done on dynamically maintaining geometrical structures, allowing sequences of insertions and deletions of objects and updating the structures appropriately (for example [14, 7]). Study of dynamic update of depth functions, however, is relatively new. Recent

*Department of Computer Science, Tufts University, Medford, MA 02155. {mburr, jhugg, erafalin, kseyboth, dls}@cs.tufts.edu. Work was supported by the National Science Foundation under grant CCF-0431027

results include the problem of dynamic updating of half-space depth contours [4] and the problem of maintaining the ham-sandwich cut for two dynamic sets of points, where the points in each of the two sets lie in convex position and the two convex hulls are disjoint [1]. The latter result approximates the Tukey median in a dynamic setting, based on the method described above.

We provide an efficient data structure for dynamically maintaining a ham-sandwich cut of two, possibly overlapping, point sets in the plane. An extension of the algorithm for sets with a *bounded number c of convex hull peeling layers* allows efficient area and perimeter queries.

Our algorithm handles a broader class of cases compared to previous results [15, 1]. The *static* method of Stojmenović [15] finds a ham-sandwich cut that bisects the area of two *disjoint* convex polygons in linear time. The dynamic algorithm [1] of Abbott *et al.* maintains a ham-sandwich cut of two *disjoint* convex polygons in the plane in $O(\log^3 n)$ query time and $O(\log n)$ update time. Our dynamic algorithm removes the restrictions based on the separation of the points. It also solves an open problem from 1991 about finding the area and perimeter ham-sandwich cuts for overlapping convex point sets in the static setting [15].

Section 2 describes our main algorithm, which finds the ham-sandwich cut that bisects the number of vertices for two overlapping point sets in convex position. Section 3 describes a modification of the algorithm to find a ham-sandwich cut that bisects the area or perimeter of the convex hull of two sets with a bounded number of convex hull peeling layers.

2 A Ham-Sandwich Cut that Bisects the Number of Vertices for Two Overlapping Point-Sets In Convex Position

In this problem we are given two, possibly overlapping, planar point sets S_1, S_2, each in convex position. The total number of points $|S_1 \cup S_2|$ is n and points can be dynamically added to or deleted from the sets. The goal is to find a ham-sandwich cut that bisects the number of points in each point set.

Our basic data structure is a concatenable queue, a binary search tree which enables efficient searching, splitting and concatenation. We use four concatenable queues: one for each of the upper and lower hulls for each point set. The leaves of the trees contain the vertices of the hulls in a left-to-right order, corresponding to their order along the convex-hull, and are linked by a chain. Each inner node stores the number of vertices in its subtree. Insertions are enabled using a $O(\log n)$ search to detect the location of the inserted point in the tree, and a $O(\log n)$ update. Deletions are performed similarly, using a $O(\log n)$ node deletion and update step.

Propositions 1 and 2 and Lemma 1 are based on work described in [1]. Theorem 1 contains our main contribution, applying the method to overlapping point sets.

Proposition 1. Given a line l, we can find which edges of the convex hull of $S_i, i \in \{1, 2\}$ it intersects in $O(\log n)$ [1].

Proof. We describe an algorithm that finds edges in the upper hull of S_i, if they exist. The algorithm works similarly for the lower hull.

We wish to find two vertices v, w along the hull that are on opposite sides of l. Once such vertices are found, binary searching the path from v to w will locate two adjacent vertices on opposite sides of l.

Consider the differences between the slope of l and the slopes of edges formed by adjacent vertices along the chain. These differences are monotonic along the chain. By binary searching the

slopes locate the vertex a whose two adjacent edges have positive and negative slopes relative to l. This vertex is the furthest vertex of the upper hull above l or of the lower hull below l. We compare a to the two endpoints of the chain of the upper convex hull b_1, b_2. Either a is on the opposite side of l relative to b_1 or b_2 or l does not cross the chain. □

Proposition 2. Given a line l we can find the number of points of $S_i, i \in \{1, 2\}$ above l in $O(\log n)$ [1].

Proof. Using the method in Proposition 1 we find the edges that cross l. The number of points of S_i above l is the sum of the number of points of the upper chain above l and number of points of the lower chain above l. Since the points form a contiguous chain, their sum can be read from the data structure in $O(\log n)$ time, by first locating the vertices that are directly above l in the tree and then reading the sum of the vertices in the section of the tree between the two vertices (using the internal value of the least common ancestor of the two vertices in the tree). □

Lemma 1. Given a point p outside the convex hull of S_i we can find the bisector of $S_i, i \in \{1, 2\}$ through p in $O(\log^2 n)$ time [1].

Proof. Define $F(v)$ to be the number of points of S_i above the line $\overline{pv}$. Then as we advance in counter-clockwise order along the hull $F(v)$ is unimodal. $F(v)$ can be computed using the method in Proposition 2. By Fibonacci searching [5] $F(v)$ we can find the edge of S_i that the bisector of S_i through p crosses in $O(\log^2 n)$ time. □

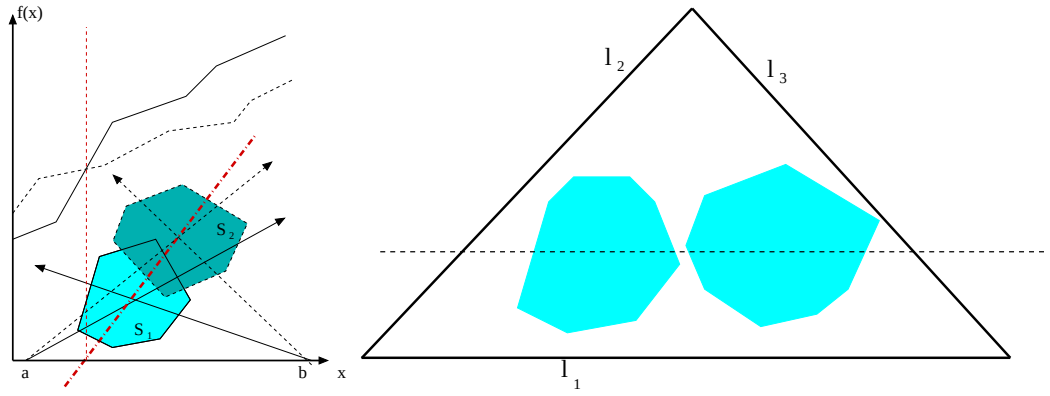
Theorem 1. The ham-sandwich cut of S_1, S_2 can be found in $O(\log^3 n)$ time.

Proof. Consider a *designated line* l, a line outside the convex-hull of the union of both point sets S_1, S_2. For clarity, assume that l is the x-axis and is below both point sets. For any real number x, let $f_i(x)$ be the slope of the line that both bisects S_i and passes through the point $p = (x, 0)$ on l. Each function $f_i, i \in \{1, 2\}$ is monotonic and continuous. Define $f(x) = f_1(x) - f_2(x)$. Assume h is a ham-sandwich cut of the two point-sets that intersects the designated line l in point $p_h = (x_h, 0)$. Then $f_1(x_h) = f_2(x_h)$ (see Figure 1(a)). While two separated point sets have only one possible ham-sandwich cut [15], overlapping point sets can have more than one (for example two identical point sets) so the set of ham-sandwich cuts may intersect a designated line any number of times, including 0.

In some cases there may exist a designated line l that does not intersect any ham-sandwich cut (see Figure 1(b)). We guarantee that the algorithm finds at least one intersection point of the ham-sandwich cut with a designated line by considering a set of 3 designated lines l_1, l_2, l_3 that form a triangle enclosing the point sets. Any two of the lines capture every possible slope of the ham-sandwich cut. Each line l_i is processed separately to find a ham-sandwich cut that intersects it (if one exists).

Given a vertex of S_1 the algorithm uses the method in Lemma 1 to find the line h through that vertex bisecting S_1 and intersecting the designated line l at point $p = (x, 0)$. The method used in Lemma 1 is reapplied to find the line m passing through p and bisecting S_2 to compute $f(x)$.

Consider a segment $[a, b]$ on l intersected by a ham-sandwich cut of S_1, S_2. Then $f(a)$ and $f(b)$ change signs. By Fibonacci search on $f()$ along l we can limit the region of l that intersects the ham-sandwich cut, until the region represents an edge or two adjacent vertices on S_1 or S_2. To bound the running time of our algorithm, use the vertices of S_1, S_2 to constrain the points of l that are visited during the search. The algorithm starts by considering the segment $[a, b]$ defined by



(a) **(b)**

Figure 1: (a) Two convex regions representing two point sets S_1, S_2 and their bisectors, represented using solid and dashed lines respectively. The slope of S_1's bisector is lower than that of S_2's bisector for the bisectors intersecting point a, but is higher for the bisectors intersecting point b. The two monotonically increasing lines represent functions $f_1()$ and $f_2()$, showing the slopes of the bisectors as a function of their intersection point with l_1. In this instance, the two lines intersect in one point, which corresponds to the ham-sandwich cut of the two sets. (b) Two convex polygons representing two point sets S_1, S_2. The ham-sandwich cut of the point set does not intersect l_1 but does intersect lines l_2 and l_3.

the extreme bisectors through vertices of the two point sets and performs Fibonacci search on $f()$ (using the method in Lemma 1) to refine $[a, b]$ until the region cannot be further minimized.

The order of the vertices along the convex hull of $S_i (i \in 1, 2)$ corresponds to the order of the respective intersection points on l of the bisectors of S_i passing through these vertices. However, the order of the vertices of the combined set $S_1 \cup S_2$ does not correspond to the order of the respective intersection points. Therefore, instead of performing Fibonacci search on the combined set of points $S_1 \cup S_2$, our algorithm searches one set S_1 in $O(\log^3 n)$ time to find the smallest region $[a, b]$ on l where $f()$ changes signs; locates the corresponding vertices of S_2 whose bisectors cross l immediately to the left and to the right of $[a, b]$ (Lemma 1); and then refines $[a, b]$, if possible, by searching S_2. The total time complexity is therefore $O(\log^3 n) + O(\log^2 n) + O(\log^3 n) = O(\log^3 n)$. □

3 Ham Sandwich Cut of Area and Perimeter for Sets with a Bounded Number of Convex Hull Peeling Layers

We describe a modification of the algorithm that extends area and perimeter queries to a larger family of data sets, those with a bounded number of convex hull peeling layers.

The convex hull peeling layers [8, 2] are defined as follows: the points on the outer convex hull of $\mathcal{S}$ are designated layer one and the points on the kth layer are the points on the convex hull of the set $\mathcal{S}$ after the points on layers 1 to $k-1$ were removed. The maximal convex-hull peeling depth in the plane can range from 1 (all the points are on convex hull) to $n/3$ (each convex hull contains 3 points). The convex-hull peeling layers can be computed in the plane in optimal $\Theta(n \log n)$ time using Chazelle's deletions-only dynamic convex-hull algorithm [6].

In this problem we are given two, possibly overlapping, planar point sets S_1, S_2, where points can be dynamically added to or deleted from the sets. The total number of points $|S_1 \cup S_2|$ is n

and the number of convex-hull peeling layers for each set is bounded by a constant c. Our goal is to find a ham-sandwich cut that bisects the area or perimeter of the convex hull of each point set.

Theorem 2. Given S_1, S_2, two point sets with a bounded number c of convex hull peeling layers, with $|S_1 \cup S_2| = n$, there exists a dynamic algorithm that maintains the area and perimeter ham-sandwich cuts in $O(\log^3 n)$ query time and $O(c \log n)$ update time.

Section 3.1 describes how to perform the $O(\log^3 n)$ time area and perimeter queries. Section 3.2 describes the $O(c \log n)$ time insertions and deletions.

3.1 Area and Perimeter Queries

We view each convex-hull peeling layer of the sets S_1, S_2 as a separate polygon. Let P_i^j be the j^{th} convex-hull peeling layer (represented as a convex polygon) of point set S_i (where $i \in \{1, 2\}$ and P_i^0 is the outermost layer which is the convex hull of the set). We use the same data structure as for the vertex count queries, a concatenable queue, but need a total of $O(4c)$ trees: for every convex-hull peeling layer, one for the upper hull and one for the lower hull. Section 3.2 explains how this data structure supports dynamic operations.

Our data structure stores the vertices of a single layer P_i^j in sorted left-to-right order along the upper or lower convex hull of the layer $v_1, \ldots v_{N(i,j)}$, where $N(i, j)$ represents the total number of vertices along the hull. Each vertex v_k (for $1 < k \leq N(i, j)$) in the data structure for layer P_i^j stores (1) the signed area of the trapezoid defined by a section of the x-axis and edge (v_k, v_{k-1}) (2) the length of (v_k, v_{k-1}). Each inner node stores the sum of the measures of leaves in its subtree. This allows the measure associated with each continuous chain to be computed in $O(\log n)$ time.

The computation of the area and perimeter applies only to the vertices in the outermost convex-hull layer so a query requires a search only on the outermost convex-hull peeling layer. An insertion or deletion of a point changes the measure of at most two edges in a convex-hull, which propagates in the data structure in $O(\log n)$.

Propositions 1 and 2, Lemma 1 and Theorem 1 all extend easily for area and perimeter queries. One main difference exists: for vertex count queries, as described in Section 2, the ham-sandwich cut is a family of lines delimited by two pairs of edges of every convex set. However, for area and perimeter queries once these edges are detected further computation, to find the roots of a polynomial, is needed in order to compute the parameters of the ham-sandwich cut. Abbott *et al.* [1] describe how these computations can be done.

3.2 Insertion and Deletion

We achieve the required time complexity by utilizing the fact that when points are inserted or deleted the convex hull peeling layers change by transition of entire sections of one layer to the adjacent layer. Our algorithm needs to maintain a correct representation of the convex hull peeling layers of every data set.

Insertions Consider a point q that is inserted into a data set S and assume that q is inserted in the region between layers P^k and P^{k+1}, of depth k and $k+1$ (by the assumption of general position no point can be inserted on the boundary of a layer). The point's insertion does not change the convex-hull peeling layers of depth $\leq k$ and will only affect the convex hull peeling layers of depth $> k$ (see Figure 2).

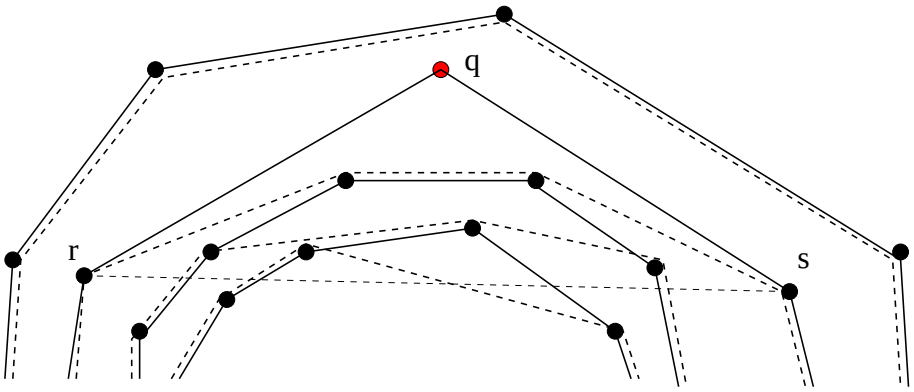


Figure 2: Insertion and deletion of point q and its affect on the convex-hull peeling-layers. The layers with point q are drawn as solids lines while the layers without q are drawn as dashed lines. Point q has depth 2, and its insertion or deletion affects only layers of depth ≥ 2. when q is deleted from the second peeling layer, points from several inner layers are outside the modified second layer formed by q's removal (with line $\overline{rs}$). However, only points from the topmost layer (the third layer) will be merged into the second layer to form an updated set.

Our algorithm updates the layers incrementally, upper and lower hulls separately. In every step of the algorithms a single point or a convex chain $Q = \{q_1, \ldots q_l\}$ outside layer k' is inserted into the layer. In the first step of the algorithm $k' = k + 1$ and $Q = \{q\}$. During the update step of the upper hull of layer k', $P^{k'}$, the layer is maintained as a concatenable queue where the points are ordered left-to-right along the layer $p_1, \ldots p_m$. The chain Q is maintained as a concatenable queue in a left-to-right order.

Since points $q_1, ..q_l$ are in convex position outside $P^{k'}$ they will belong to the new layer $\overline{P^{k'}}$. The algorithm uses binary search to find the two tangents lines from Q to $P^{k'}$, $\overline{q_1 p_i}, \overline{q_m p_j}$. The points that are no longer on the layer form one continuous chain $p_i, p_{i+1}, \ldots p_{j-1}, p_j$. Therefore to construct the updated k'th upper hull we split the concatenable queue representing $P_{k'}$ in points p_i, p_j, merge the two outer parts with the inserted points to get $\overline{P^{k'}} = \{p_1, \ldots p_{i-1}, p_i, q_1, \ldots q_l, p_j, p_{j+1} \ldots p_m\}$. Points $\{p_{i+1}, p_{i+2}, \ldots p_{j-1}\}$ are now points in convex position maintained as a concatenable queue that are outside the next layer, $P^{k'+1}$.

Deletions Consider a point q that is removed from the data set. Assume that q's depth is k. Then q's deletion will have no affect on layers of depth $< k$ and will change layers with depth k and higher (see Figure 2).

Our algorithm updates the layers incrementally, upper and lower hulls separately. During the update step of the upper hull of layer k', $P^{k'}$, it is maintained as a concatenable queue where the points are ordered left-to-right along the layer $p_1, \ldots p_m$. In addition points $Q = \{p_i, \ldots q_j\}$ are a chain of the layer $P^{k'}$ that is about to be removed. In the first step of the algorithm $k' = k$ and $p_i = p_j = q$.

When points Q are removed line $\overline{p_{i-1} p_{j+1}}$ is on the boundary of the convex hull of the set $P^{k'} \setminus Q$. However, it may not be on the boundary of the set $\cup_{l \geq k'} P^l \setminus Q$ as points from inner layers can be on the outer side of the line (see Figure 2). Points from several layers may be on the outer side of the line but only points from the next inner layer $P^{k'+1}$ may be on the new convex hull and

therefore should be removed from layer $k'+1$ and added to layer k'. These points form a single chain $R = \{r_l, \ldots r_o\}$. The outer points of the chain r_l, r_o can be found using binary search on the chain representation of $P^{k'+1}$ that are above the line $\overline{p_{i-1}p_{j+1}}$ where lines $\overline{r_l p_i}$ and $\overline{r_o p_j}$ are tangents to layer $P^{k'+1}$.

To construct the updated k'th upper hull we split the concatenable queue representing $P_{k'}$ in points p_i, p_j, and split the concatenable queue representation of $P^{k'+1}$ in points r_l, r_o. The new k'th level is formed by merging the two outer parts of $P^{k'}$ with R and the process repeats for the next layer, $P^{k'+1}$.

Acknowledgement The authors thank Jelani Nelson and Erik Demaine for finding an error in an earlier version of this paper.

References

[1] 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 (CCCG'05)*, pages 61–64, 2005.

[2] V. Barnett. The ordering of multivariate data. *J. Roy. Statist. Soc. Ser. A*, 139(3):318–355, 1976.

[3] S. Bespamyatnikh, D. Kirkpatrick, and J. Snoeyink. Generalizing ham sandwich cuts to equitable subdivisions. *Discrete Comput. Geom.*, 24(4):605–622, 2000. ACM Symposium on Computational Geometry (Miami, FL, 1999).

[4] M. Burr, E. Rafalin, and D. L. Souvaine. Dynamic update of half-space depth contours. In *14th Annual Fall Workshop on Computational Geometry*, 2004.

[5] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *J. ACM*, 34(1):1–27, 1987.

[6] Bernard Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, 31(4):509–517, 1985.

[7] Y.J. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *PRoc. of the IEEE*, 80(9):1412–1434, 1992.

[8] W. Eddy. Convex hull peeling. In H. Caussinus, editor, *COMPSTAT*, pages 42–47. Physica-Verlag, Wien, 1982.

[9] H. Edelsbrunner. *Algorithms in combinatorial geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1987.

[10] J. Hodges. A bivariate sign test. *The Annals of Mathematical Statistics*, 26:523–527, 1955.

[11] Chi-Yuan Lo. *Ham-Sandwich Cuts and Related Problems*. Ph.D. thesis, Dept. Comput. Sci., Rutgers University., New Brunswick, NJ, 1992.

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

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

[14] Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. System Sci.*, 23(2):166–204, 1981.

[15] Ivan Stojmenović. Bisections and ham-sandwich cuts of convex polygons and polyhedra. *Inform. Process. Lett.*, 38(1):15–21, 1991.

[16] J. W. Tukey. Mathematics and the picturing of data. In *Proc. of the Int. Cong. of Math. (Vancouver, B. C., 1974), Vol. 2*, pages 523–531. Canad. Math. Congress, Montreal, Que., 1975.