Dynamic Ham-Sandwich Cuts of Convex Polygons in the Plane

Timothy Abbott* Erik D. Demaine* Martin L. Demaine* Daniel Kane* Stefan Langerman† Jelani Nelson* Vincent Yeung*

Abstract

We provide an efficient data structure for dynamically maintaining a ham-sandwich cut of two non-overlapping convex polygons in the plane. Given two non-overlapping convex polygons P_1 and P_2 in the plane, the ham-sandwich cut of P_1 and P_2 is a line that simultaneously bisects the area (or perimeter or vertex count) of both polygons. We provide a data structure that supports queries for the ham-sandwich cut in $O(\log^3 n)$ worst-case time and insertions and deletions of vertices of the P_i in $O(\log n)$ worst-case time. We also show how this data structure can be used to maintain a partition of the plane by two lines into four regions each containing a quarter of the total polygon area (or perimeter or vertex count). In particular, if we use the vertex-count measure, the intersection of these two lines gives a point of Tukey depth $n/4$, which serves as an approximate Tukey median.

1 Introduction

Finding a ham-sandwich cut is a well-studied problem with linear-time solutions in many contexts; see, e.g., [3, 7, 9]. 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 m. If S_1 and S_2 are discrete sets of points, the measure m is usually the number of points; if S_1 and S_2 are regions, measure m can be area, perimeter, or the number of vertices (if S_1 and S_2 are polygonal).

A related problem, introduced by Megiddo [8], 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, $\frac{1}{4}m(S)$. 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. The 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.

A two-line partition also serves as an approximate Tukey median or centerpoint; see, e.g., [5]. A *Tukey median* of a subset S of the plane is a point of maximum Tukey depth. The *Tukey depth* of a point is the minimum measure of the intersection of S with a half-plane whose bounding line passes through the point. The depth of the Tukey median is always between $n/3$ and $n/2$; any point of depth at least $n/3$ is called a *centerpoint*. The intersection point of the two-line partition provides a point of Tukey depth $n/4$, which approximates the maximum possible depth within a factor of 2 and approximates the centerpoint guarantee within a factor of $\frac{4}{3}$.

While the problems of finding ham-sandwich cuts, two-line partitions, and Tukey medians and centerpoints 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 solving the dynamic versions of ham-sandwich cuts, two-line partitions, and hence approximate Tukey medians, in the case of two disjoint convex polygons P_1 and P_2. (For two-line partitioning and Tukey medians, the subset S is the union $P_1 \cup P_2$.) Our main result is a data structure that supports insertion and deletion of a vertex in either P_1 or P_2 in $O(\log n)$ worst-case time, supports query for a ham-sandwich cut, a two-line partition, or an approximate Tukey median in $O(\log^3 n)$ worst-case time, where n denotes the total number of vertices among P_1 and P_2. (For the two-line partition we must also solve a polynomial of constant degree > 4.)

Data structures for dynamic convex polygons have not been considered before to our knowledge. We consider the following formulation of updates. The data structure must maintain two planar point sets subject to insertions and deletions of points in either of the sets, so long as these updates preserve the invariants that the points in each set lie in convex position and that the two convex hulls P_1 and P_2 are disjoint.

2 Background

The existence of a ham-sandwich cut is a well-known result; see, e.g., [3]. We give a short proof in the context of two disjoint convex polygons P_1 and P_2 because our data structure uses the same principle. Consider any line ℓ separating P_1 and P_2 into separate half-planes. Without loss of generality, assume that ℓ is the y-axis

*MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {tabbott,edemaine,mdemaine,dankane,minilek,vshyeung}@mit.edu.

†Chercheur qualifié du FNRS, Université Libre de Bruxelles, Département d'informatique, ULB CP212, Belgium. Stefan.Langerman@ulb.ac.be.

and that P_1 is on the left. For any real number x, let $f_i(x)$ be the slope of the line that passes through the point $(0, x)$ on ℓ and bisects the measure of P_i. Define $f(x) = f_1(x) - f_2(x)$. Because f is continuous, and because $\lim_{x\to\infty} f(x) = +\infty$ and $\lim_{x\to-\infty} f(x) = -\infty$, it follows by the intermediate value theorem that there is an x such that $f(x) = 0$, which implies that the bisectors of P_1 and P_2 through x are in fact the same line, which provides a ham-sandwich cut.

Lo et al. [7] give an optimal $O(n)$-time algorithm for finding a ham-sandwich cut of two static point sets of total size n. Stojmenović [9] gives an $O(n)$-time algorithm for finding a ham-sandwich cut that bisects the area of two static convex polygons with n vertices total. However, his solution does not support fast updates upon insertion or deletion of vertices. To our knowledge, our solution for ham-sandwich cuts is the first that handles dynamic convex polygons in $o(n)$ time per update.

Using the existence of ham-sandwich cuts, it is easy to prove the existence of two-line partitions [8]. Consider any line ℓ_1 that bisects the measure of $S = P_1 \cup P_2$. Now consider partitioning the measure into the portion S_1 to the left of ℓ_1 and the portion S_2 to the right of ℓ_1. A ham-sandwich cut of (S_1, S_2) gives us a line ℓ_2 that simultaneously bisects the measure of S_1 and S_2, each of which bisected the entire measure S, and thus (ℓ_1, ℓ_2) is a two-line partition. Based on this proof, Megiddo [8] gives an $O(n)$-time algorithm for the two-line partition problem for two static sets of points in the plane. (This result preceded the $O(n)$-time algorithm for general ham-sandwich cuts.)

Unfortunately, our data structure for two-line partitioning cannot proceed so simply. The partition of $S = P_1 \cup P_2$ into the polygons P_1 and P_2 may not bisect the measure $m(S)$, so this partition may not induce a suitable line ℓ_1. Thus such an approach would need a more general ham-sandwich data structure than the case of two convex polygons. Nonetheless we show that, if we chose ℓ_1 to be a ham-sandwich cut of (P_1, P_2), then it is possible to find a suitable line ℓ_2 using similar techniques to ham-sandwich cutting, without more than constant-factor overhead.

3 The Data Structure

Our data structure represents each convex polygon P_i by two augmented balanced binary search trees, one for the upper chain and one for the lower chain, each ordering the edges in counterclockwise order. The upper and lower chains are defined by their common endpoints of the leftmost and rightmost vertices. With each edge (v_1, v_2) of either P_i, we store three measures: (1) the signed area of the trapezoid defined by v_1, v_2, and the projection of v_1 and v_2 down onto the x-axis; (2) the length of (v_1, v_2); and (3) the number 1. Each node x of a binary search tree maintains three subtree sums, one for each measure. From this information we can compute the measure of any subchain in $O(\log n)$ time.

First we discuss how to maintain this structure under insertion and deletion in $O(\log n)$ time per operation. Both insertion and deletion require updating the measures of at most two edges, and the subtree sums can be propagated in $O(\log n)$ time. For vertex deletion, we just delete the vertex from the tree containing it 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.

Consider adding a new vertex v to a polygon P_i, such that the resulting set of vertices of P_i are still in convex position. By a sidedness test between v and the line connecting the two endpoints of the two chains, we can determine whether v should be added to the upper or lower chain. By symmetry, assume it is the upper chain. We want to find the unique edge of the upper chain whose line extension is below v; the rest are above v if v is to be in convex position with the rest. Now pick the median edge e, and let ℓ be its line extension. If v is above ℓ, we add v in the upper chain between e's endpoints and are done. Otherwise, v is below ℓ. Now ℓ partitions the exterior of the upper chain into two pieces, the left and the right. Because v is below ℓ, we can test whether v is in the left or right piece by testing sidedness against a vertical line through any point along e. Then we can recurse in the appropriate side, and find the proper edge in $O(\log n)$ time. Therefore, insertion can be done in $O(\log n)$ time.

Next we show how to perform some basic queries using this data structure, which will be necessary for the queries of interest.

Proposition 1 *Given a line ℓ, we can find which edges of P_i it intersects in $O(\log n)$ time.*

Proof. We describe how to find the edge of the upper chain of P_i that ℓ intersects, as the problem of finding the edge in the lower chain is symmetric.

First we find two vertices of the upper chain on opposite sides of ℓ. If the two endpoints of the chain are on opposite sides, we use those. Otherwise, for each edge of the upper chain, consider the absolute difference between its slope and ℓ's slope. Over the edges in counterclockwise order, this function is unimodal with a unique local minimum (corresponding roughly to the tangent of P_i parallel to ℓ). We can therefore find this minimum in $O(\log n)$ time via Fibonacci search [6]. If any vertex of P_i is on the opposite side of ℓ compared to the two endpoints of the chain, then one of the endpoints of this edge must be. Thus we obtain two vertices v and w of the chain on opposite sides of ℓ, if two such vertices exist. (Otherwise, ℓ does not intersect P_i at all, and we are done.)

Now we can binary search to find an edge that straddles ℓ, by considering the median vertex m between v and w and recursing on either (m, v) or (m, w), whichever pair straddles ℓ, until we find that (v, w) is a single edge. Once we have found such a straddling edge (v, w), we output it and recurse on the two subchains resulting from removing that edge. If there is a second edge that intersects ℓ, one of these recursions will find it. Only one recursion of this type is necessary, so the total running time is $O(\log n)$. □

Proposition 2 *Given a line ℓ, we can find the measure of the portion of P_i above ℓ in $O(\log n)$ time.*

Proof. By the previous proposition, we can find the two edges e_1, e_2 that ℓ intersects in $O(\log n)$ time, as well as the points of intersection. We then compute the sum of the measures of the interval of edges from e_1 and e_2. For perimeter and vertex count, we are done. For area, we follow the ideas of [2]. We subtract 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 the area under e_1 and e_2 below ℓ. The result is the desired area of P_i above ℓ. □

Proposition 3 *Given a point p, we can find the bisector of P_i through p in $O(\log^2 n)$ time.*

Proof. Observe that, as a function of the slope of lines ℓ through p, the measure of P_i to the left of ℓ is monotonic. Thus, we can binary search for the right edge of P_i intersected by the bisector containing p. We do this by binary searching through the vertices of P_i, where for each vertex v we consider the measure to the left of the directed line $\overline{vp}$ to tell us which way to go in the binary search. We will eventually be down to a range of two adjacent vertices, which must be the desired edge intersected by the bisector. Similarly, we can find the left edge intersected by the bisector through p. Now we can find the actual bisecting line in constant time by solving a quartic polynomial. Each of the two binary searches use $O(\log n)$ iterations. During each iteration, we compute the measure of P_i cut by a line, which takes $O(\log n)$ time by the previous proposition. The total running time is therefore $O(\log^2 n)$. □

Now we turn to the main queries of interest: ham-sandwich cuts and two-line partitions.

3.1 Ham-Sandwich Cut

Theorem 4 *There is a data structure that maintains two convex polygons with n vertices total subject to insertion and deletion of vertices (subject to the vertices remaining in convex position) in $O(\log n)$ worst-case time and queries for a ham-sandwich cut in $O(\log^3 n)$ worst-case time.*

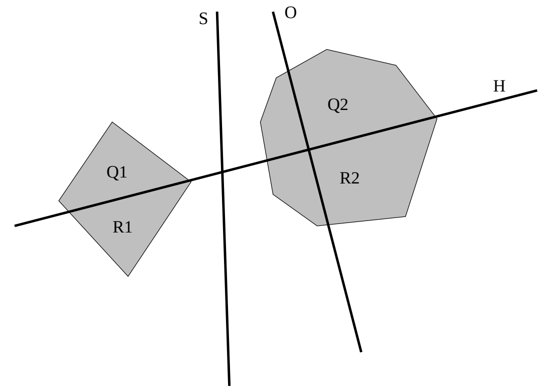

Figure 1: An example of a two-line partition.

We first find a line S that separates the plane such that P_1 is completely on one side and P_2 on the other. We can do this in $O(\log n)$ time by a method described in [4]. We conceptually rotate the plane so that S is vertical and so that P_1 is to its left. (This notion of "vertical", used for the rest of the section, is distinct from the fixed notion of vertical used to define the upper and lower chains.)

Now, we binary search for the right edge of P_1 intersected by the ham-sandwich cut. Consider all the lines that bisect the measure of P_1 and intersect the edge (v_1, v_2). Their intersections with S form a contiguous segment, starting with the intersection from the P_1 bisector that passes through v_1 up to the intersection from the bisector that passes through v_2. So, for an edge (v_1, v_2), we find the bisectors of P_1 through v_1, v_2, and let p_1 and p_2 be the intersections of those bisectors with S. For each p_i, we then find the bisector of P_2 through p_i. We can then tell which way to go in our binary search by comparing the slopes of the bisectors of P_1 with the slopes of the bisectors of P_2.

Once we have found the right edge of P_1 intersected by the ham-sandwich cut, we can repeat a similar procedure to find the left edge of P_1 then the two edges of P_2. Stojmenović [9] shows that, once we have found the four edges intersected by the ham-sandwich cut, the actual ham-sandwich cut can be found in $O(1)$ time by solving for the roots of a quartic polynomial. Finding each edge requires a binary search, each iteration of which performs two bisector queries. By Proposition 3, we can find the ham-sandwich cut in $O(\log^3 n)$ time.

3.2 Two-Line Partition

Theorem 5 *The data structure of Theorem 4 can also be used to support two-line partition queries in $O(\log^3 n)$ worst-case time.*

We now show how to find a second line that, combined with the ham-sandwich cut, forms a two-line partition.

Let H be the ham-sandwich cut that we have just shown how to find. We can now view our plane as having four convex polygons, two on each side of H. Call them Q_1, Q_2 and R_1, R_2, where the Q_i are above H, and Q_1 (resp. R_1) is closer to $-\infty$ along H than Q_2 (resp. R_2). So, $P_i = Q_i \cup R_i$. We can view H as being similar to S when we were looking for a ham-sandwich cut, in that it separates two regions whose measures we would like to simultaneously bisect. Figure 1 shows a two-line partition with the polygons and lines labeled as above.

There are three cases for the second line, which we call O, that completes the two-line partition: (1) it intersects neither Q_1 nor Q_2, (2) it intersects Q_1, and (3) it intersects Q_2 but not Q_2.

For case 1 to occur, O must also intersect neither R_1 nor R_2 in order for the measures in all quadrants to be equal. Symbolically, $m(R_1) = m(R_2) = m(Q_1) = m(Q_2)$, but recall that $m(P_1) = m(Q_1) + m(R_1)$ and $m(P_2) = m(Q_2) + m(R_2)$. So, this case occurs if and only if the measures of P_1 and P_2 are equal, which we can check for and return the second line to be S if so.

Now we check whether case 2 holds. The algorithm is similar to the algorithm from Section 3.1 for finding H. We assume that O does intersect Q_1, and we binary search over the edges of Q_1 to find the edge that O passes through. For each edge (a, b), we find the bisector of the half-plane containing the Q_i that passes through a. We do the same for b. We cannot use the approach in Proposition 3 as stated since we now have two polygons, but the idea is the same. The only difference is that now at each stage of the binary search, we also have to search for the edge intersected on Q_2. This only gives an additive $O(\log n)$ in the time, so we can still find a bisector in $O(\log^2 n)$. We note that the slope of bisectors is monotonic as a function of where the bisector intersects H, and the slope is monotonic in the other direction for bisectors of the half-plane containing the R_i. So, after finding the bisectors passing through a and b, we can find bisectors of the half-plane containing the R_i and compare slopes to tell us which way to go in the binary search.

If we do eventually find an edge that works, we are indeed in case 2. We then check whether O intersects the edge of Q_1 on H. If so, we know that O intersects R_1 and does not intersect Q_2. We can then binary search over edges of R_2 to see if O intersects R_2. If O does not intersect the edge of Q_1 on H, then O must intersect Q_1 twice, not intersect R_1, and intersect P_2 (the original polygon that Q_2 and R_2 came from) exactly twice. We can find these intersections again using binary search. Once we know the set of edges of our polygons that O intersects, we can find the slope in time linear in the number of desired output bits, by computing a root of a constant-degree polynomial within the range of slopes that intersect that set of edges.

If we failed to be in cases 1 and 2, then we must be in case 3. This case is identical to case 2 with Q_1 and Q_2 swapped, so we may solve it exactly the same way.

4 Conclusion

These results can be seen as the first dynamic data structures for maintaining geometric robust statistics. It would be interesting to continue this line of pursuit, and find data structures for maintaining geometric robust statistics, and ham-sandwich cuts, under more interesting distributions than two disjoint convex polygons. The most natural distribution to consider is any set of points, which seems a more difficult problem. Another direction for extension is to consider convex polyhedra in higher (fixed) dimension. We expect to be able to generalize our structure for ham-sandwich cuts to handle two sets separated by a line, each of which is a union of a constant number of convex polygons, using the ideas of our two-line partition structure.

Other interesting directions for future work include dynamically maintaining a centerpoint (point of Tukey depth at least $n/3$) or even a Tukey median (point of maximum Tukey depth). The best results for the static versions of these problems are $O(n)$ time [5] and $O(n \log n)$ expected time [1], respectively.

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.

References

[1] T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. *SODA* 2004: 430-436.

[2] J. Czyzowicz, F. Contreras-Alcalá, and J. Urrutia. On measuring areas of polygons. *CCCG*. 1998.

[3] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag. 1987.

[4] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *J. Algorithms* 6(2): 213-224. 1985.

[5] S. Jadhav and A. Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Discrete and Computational Geometry* 12: 291-312. 1994.

[6] J. Kiefer. Sequential minimax search for a maximum. *Proc. Amer. Math. Soc.*, 4:502–506, 1953.

[7] Chi-Yuan Lo, Jiří Matoušek, and William L. Steiger. Algorithms for ham-sandwich cuts. *Discrete and Computational Geometry* 11: 433-452. 1994.

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

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