Minimum Perimeter-Sum Bipartition Implementation

Allen Mi, Haakon Flatval

December 2018

Abstract

The Minimum Perimeter-Sum Bipartition problem is stated as follows: Given set of points in the Euclidean plane, partition it into two subsets such that the sum of perimeters of their convex hulls is as small as possible. In this text, we describe our implementation of the algorithm by Abrahamsen et al [1] and provide analyses on its behaviour given different inputs.

1 Introduction

The Minimum Perimeter-Sum Bipartition problem is the problem of dividing a set of point in the two-dimensional plane into two subsets P_1 and P_2 such that the sum of the perimeters of $\mathrm{CH}(P_1)$ and $\mathrm{CH}(P_2)$ is minimized, where $\mathrm{CH}(P)$ is the convex hull of the set of points P. In 2017, Abrahamsen et al [1] devised an algorithm that solves this problem in time $O(n \log^4 n)$.

We provide an implementation of this algorithm [3] and an overview on how the implementation is structured. Furthermore, we give an analysis of how different input sets affect the performance of the algorithm, and discuss our method of constructing such input sets.

2 Background

3 Implementation

3.1 Hierarchy of Subroutines

The structural hierarchy of the subroutines for this algorithm is outlined in Figure 1.

3.1.1 Partial Convex Hull Scan

For a given angle ϕ and a set of points P, this subroutine computes the upper and lower halves of the convex hull of P, scanning through the nodes like in Graham's

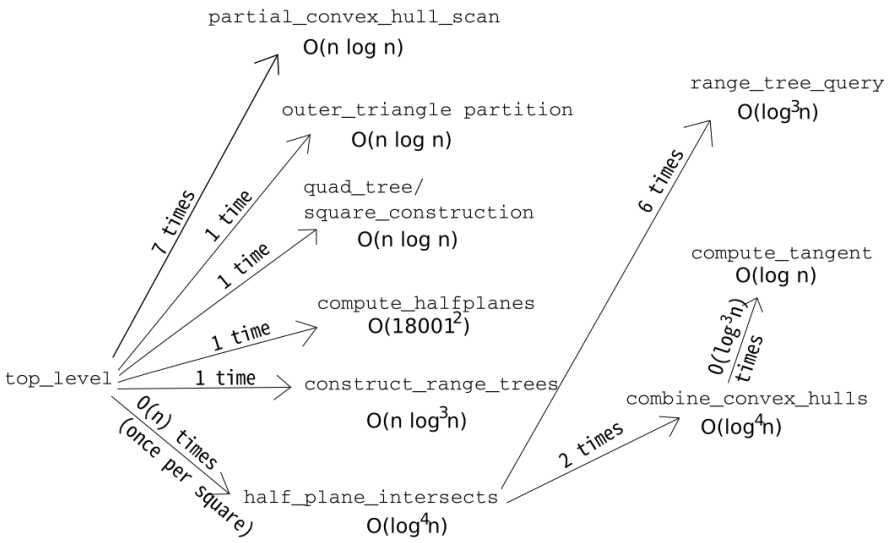

Figure 1: Structural hierarchy of subroutines

scan, but along a line with angle ϕ to the x-axis. This allows us to quickly compute convex hulls of a bipartition of P separated by a line perpendicular to the scan direction. This operation takes $O(n \log n)$ time for a given angle, n being the size of P.

3.1.2 Outer Triangle Partition

For a given set of points P, computes all triangles formed by three successive points on the set's convex hull, and finds and stores the subset of the points contained in each such triangle. This allows us to quickly compute the length of the convex hull of P p, where p is a point on the convex hull of P. This operation takes $O(n \log n)$ time, where $n = |P|$.

3.1.3 Quad Tree Construction

Each node in such a quad tree represents a square in the plane. Each node has four children, representing the regions of the four equal squares that make up their parent's region. The construction of a compressed quad tree over n points takes $O(n \log n)$ time. (Given appropriate computational model)

3.1.4 Range Tree Construction and Search

The range tree we construct here has three levels. The first level sorts the points on the x-axis, the subsequent level sorts the points at each node by the y-axis. The lowest level sort the points by a given direction in the plane. This allows us to easily find points that are restricted to a given rectangular axis-aligned region

in the plane, and that are also inside a specified half-plane. This range tree also compute the convex hull of each node in the lowest level of the range tree. The range tree construction takes $O(n \log^3 n)$, n being the number of points over which we construct the tree

3.1.5 Tangent Computation

For two polygons A and B, computes their tangents using the algorithm provided in [2]. We use this to determine which polygons belongs to the hull in the Convex Hull Combining subroutine. The algorithm uses $O(\log(a+b))$ time, where a and b are the number of points on the boundary of A and B respectively.

3.1.6 Convex Hull Combining

For a given set of convex hulls $\mathcal{C}$, finds the convex hull of all points on all the hulls. We use this to combine the convex hulls found from a query in the range tree described above, including its length, so that we can evaluate the perimeter. The running time for this subroutine is $O(k \log m)$, k being the number of convex polygons, and m the number of points on the hulls in total.

4 Performance

5 Input Design

5.1 Input in Realistic Scenarios

5.2 Adversarial Inputs

5.3 Input Generation via Gradient Descent

6 Conclusion

References

[1] Mikkel Abrahamsen, Mark de Berg, Kevin Buchin, Mehran Mehr, and Ali D. Mehrabi. Minimum Perimeter-Sum Partitions in the Plane, *33rd International Symposium on Computational Geometry* (SoCG 2017)

[2] D. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. *Proc. 2nd Workshop Alg. Data Struct. (WADS),* LNCS 955, pages 183-193, 1995

[3] Allen Mi, Håkon Flatval. *Minimum Perimeter-Sum Bipartition implementation* `https://github.com/AllenGlan/mpsb-implementation`