11. Convex Hull and Closest Pair

CPSC 535 ~ Spring 2019

Kevin A. Wortman

CALIFORNIA STATE UNIVERSITY FULLERTON

November 18, 2019

Convex Hulls

convex hull problem
input: set of $n \geq 3$ points Q
output: $CH(Q)$, the subset of Q that is the set of vertices on the convex hull of Q

Convex hull $\equiv$ boundary of convex polygon enclosing all of Q

Applications

- object intersection in raytracing, video games, GUIs
- drawing implicit regions in GIS
- finding farthest points
- component of other algorithms

Approaches to Convex Hulls

Like the sorting problem, many algorithm patterns work for convex hulls, and there is a rich literature of competitive algorithms.

- Greedy pattern: line-sweep, update hull as we go
- Divide-and-conquer: divide Q in half, compute convex hulls for each half, merge two convex hulls into one
- Iterative improvement: start with a superset of $CH(Q)$; refine by repeatedly eliminating a constant fraction of the points until only $CH(Q)$ remains

Baseline Algorithm

Observe

- any two input points define a line ℓ
- when those points are both in $CH(Q)$, remaining $n - 2$ points are all on the same side of ℓ *(geometric property)*
- $\Longrightarrow$ for each pair of input points p, q, see whether all other points are on the same side of ℓ
- if so include p, q in $CH(Q)$

Baseline Pseudocode

```
function NAIVE-CONVEX-HULL(Q)
    H = ∅
    for distinct points p, q ∈ Q do
        form line ℓ intersecting p and q
        k = # points above ℓ
        if k = (n − 2) or k = 0 then
            H = H ∪ {p, q}
        end if
    end for
    return H
end function
```

Analysis: $\Theta(n^3)$ time

Graham Scan Idea

- greedy pattern, reduction-to-sorting
- Heuristic: when touring the hull in counter-clockwise order, we **only make left turns**
- right turn = exiting a concavity; middle point not in hull
- ∴ sweep counter-clockwise, keep points that participate in left turns, drop points in the middle of right turns
- alternative kind of line sweep: rotating the line (not left-to-right)

Graham Scan Greedy Heuristic

- $p_1, \ldots, p_m = Q$ sorted into counter-clockwise order, eliminating ties
- stack S of points; contains hull of points visited *already*
- base case: push first 3 points onto S
  - for any three points p, q, r forming a non-degenerate triangle, $CH(\{p, q, r\}) = \{p, q, r\}$
- inductive case:
  - examine next input point p_i, top of stack t, next-lowest stack point r
  - if $\angle rtp_i$ is not a left turn $\implies$ t not on hull
- Note: need stack data structure w/ accessor to top **two** elements

Graham Scan Pseudocode

function GRAHAM-SCAN(Q) ▷ guaranteed $|Q| \geq 3$
    p_0 = lowest point in Q (break ties by choosing leftmost point)
    $p_1 \ldots p_m$ = sort $Q - \{p_0\}$ into counter-clockwise order, by polar angle with p_0; break ties by keeping only the point farthest from p_0
    $S =$ new stack
    $S.PUSH(p0)$
    $S.PUSH(p1)$
    $S.PUSH(p2)$
    for i from 3 through m **do**
        while $\angle p_i, S.TOP, S.BELOWTOP$ is non-left turn **do**
            $S.POP()$
        end while
        $S.PUSH(p_i)$
    end for
    return S
end function

Graham Scan Analysis

- find p_0: $\Theta(n)$
- sort: $\Theta(n \log n)$
- eliminate tied points: $\Theta(n)$
- each stack operation is $\Theta(1)$
- **for** loop repeats $m < n$ times
- turn angle test, stack operations are $\Theta(1)$
- $\Rightarrow \Theta(n \log n)$ time
- dominating term is sort, organizing data structure is arrayed stack $\Rightarrow$ good constant factors

Jarvis March

Alternative greedy heuristic: moving around the hull counter-clockwise, each step from one vertex to the next is *the input point whose angle is shallowest* ("gift wrapping")

Jarvis march

1. Find the lowest and highest points in Q.
2. (right chain) Starting from the lowest point, and until we reach the highest point:
   2.1 Linear search Q for the next point, minimizing the angle between the two points.
   2.2 Add the first point to $CH(Q)$ and move to the second point.
3. (left chain) Starting from the highest point, repeat this process until we reach the lowest point.
4. Return $CH(Q)$

Jarvis March Analysis

Preprocessing to find highest/lowest: $\Theta(n)$

Each iteration of the left/right-chain loops identifies one hull point $\implies$ in total they iterate h times, where $h \equiv$ number of points on the hull.

linear search inside the loops takes $\Theta(n)$ time.

$\therefore \Theta(nh)$ total time.

Faster than Graham scan's $\Theta(n \log n)$ when $h \in o(\log n)$.

Optimal output-sensitive: Chan's algorithm, $\Theta(n \log h)$.

Summary of Convex Hull Algorithms

Algorithm	**Time**	**Main Idea**
Graham Scan	$\Theta(n \log n)$	sort, skip right turns
Jarvis March	$\Theta(nh)$	gift-wrapping
Chan's algorithm	$\Theta(n \log h)$	divide w/ Graham, merge w/ Jarvis

Closest Pair Problem

closest pair problem
input: set of $n \geq 2$ points Q
output: two points $p, q \in Q$ minimizing $d(p, q)$

$d(p, q)$ is standard Euclidean distance
$d((x_p, y_p), (x_q, y_q)) \equiv \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$

Applications

- find two objects at greatest risk of collision
- determine numerical precision needed for points
- match predicted user preference to products
- match players for fair contest

Baseline Algorithm

```
function CLOSEST-PAIR-NAIVE(Q)                ▷ guaranteed |Q| ≥ 2
    p = q = NIL
    δ = ∞
    for distinct a, b ∈ Q do
        δ_ab = d(a, b)
        if δ_ab < δ then
            p = a, q = b, δ = δ_pq
        end if
    end for
    return p, q
end function
```

Analysis: $\Theta(n^2)$

Divide-and-Conquer First Draft

- base case: $n \leq 3$, use baseline algorithm
- else draw vertical line ℓ dividing Q into halves L, R
- recursively find closest pairs p_L, q_L and p_R, q_R
- solution is one of
  - (from the left) p_L, q_L
  - (from the right) p_R, q_R
  - (straddling the boundary) some $p \in L$ and $q \in R$ even closer than $d(p_L, q_L)$ and $d(p_R, q_R)$
- naïve search for straddling case is $\Theta(n^2) \implies$ need to be more clever to speed up
- clever = use geometry

Narrowing Search at Boundary

- **Claim**: only need to check $O(n)$ pairs of straddling points, not $\Theta(n^2)$
- let $\delta = \min(d(p_L, q_L), d(p_R, q_R))$ = distance between closest pair entirely in L or entirely in R
- suppose $\exists p_S$ left of ℓ, q_S right of ℓ, with p_S, q_S closer than δ
- such p_S, q_S must reside in a $2\delta \times \delta$ rectangle centered on ℓ
- *packing argument:* since non-straddling point pairs are separated by $\geq \delta$, there are at most 8 non-straddling points in this rectangle (4 per corner of each square)
- $\therefore$ for each point p within δ of ℓ, test p against the 7 points nearest p in y-direction
- $\leq n$ points within δ of ℓ so $\leq 7n$ pairs of points $\in O(n)$

Divide-and-Conquer Second Draft

```
function CLOSEST-PAIR-DC(Q)
    if n ≤ 3 then
        return CLOSEST-PAIR-NAIVE(Q)
    else
        X = sort Q by x-coordinate
        Y = sort Q by y-coordinate
        ℓ = vertical line through median x-coordinate
        L = {p ∈ Q : p left of ℓ}, R = Q − L
        p_L, q_L = CLOSEST-PAIR-DC(L)
        p_R, q_R = CLOSEST-PAIR-DC(R)
        p, q = closer of p_L, q_L versus p_R, q_R; δ = d(p, q)
        for a ∈ Q and within δ of ℓ do
            for 7 points b preceding a in Y do
                if d(a, b) < δ then
                    p = a, q = b, δ = d(a, b)
                end if
            end for
        end for
        return p, q
```

Second Draft Analysis

- base case is $\Theta(1)$
- each sort is $\Theta(n \log n)$
- compute ℓ is $\Theta(1)$ (given sorted X)
- form L, R is $\Theta(n)$
- straddling **for** loop is $\Theta(7n) = \Theta(n)$
- $T(n) = 2T(n/2) + n \log n$
- by master theorem, $\Theta(n^2 \log n)$
- **bottleneck** is sorting X, Y; can do this once before recursion

Third Draft – Outer Algorithm

```
function CLOSEST-PAIR(Q)
    X = sort Q by x-coordinate
    Y = sort Q by y-coordinate
    Return CLOSEST-PAIR-HELPER(X, X, Y)
end function
```

Third Draft – Recursive Helper

```
function CLOSEST-PAIR-HELPER(P, X, Y)
    if n ≤ 3 then
        return CLOSEST-PAIR-NAIVE(P)
    else
        x_m = median x-coordinate in P
        ℓ = vertical line through x_m
        L = {p ∈ P : p left of ℓ}, R = P − L
        p_L, q_L = CLOSEST-PAIR-HELPER(L, X, Y)
        p_R, q_R = CLOSEST-PAIR-HELPER(R, X, Y)
        p, q = closer of p_L, q_L versus p_R, q_R; δ = d(p, q)
        for a ∈ P and within δ of ℓ do
            for 7 points b preceding a in Y do
                if d(a, b) < δ then
                    p = a, q = b, δ = d(a, b)
                end if
            end for
        end for
        return p, q
    end if
```

Third Draft Analysis

- helper:
  - find median x is $\Theta(n)$
  - (use general median-finding algorithm; or count $k = |P \cap X|$ then iterate past $k/2$ elements of X)
  - compute ℓ is $\Theta(1)$ (given median)
  - form L, R is $\Theta(n)$
  - straddling **for** loop is $\Theta(7n) = \Theta(n)$
  - $T(n) = 2T(n/2) + n \in \Theta(n \log n)$ by master theorem
- outer algorithm:
  - each sort is $\Theta(n \log n)$
  - helper is $\Theta(n \log n)$
- total $\Theta(n \log n)$

Closest Pair Summary

Divide-and-conquer algorithm takes $\Theta(n \log n)$ time.

Depends on

- geometric packing argument: checking only $7n$ pairs of straddling points suffices
- sort in $\Theta(n \log n)$
- median in $\Theta(n)$
- master theorem