Sublinear Geometric Algorithms

Bernard Chazelle, Ding Liu
Princeton University

Avner Magen
University of Toronto

Huge input and sublinear time

Large geometric datasets require algorithms that examine only a small fraction of the input.

Traditionally, the cure is preprocessing.

Huge input and sublinear time

Large geometric datasets require algorithms that examine only a small fraction of the input.

Traditionally, the cure is preprocessing.

Examples. what can be done with preprocessing?

- Nearest Neighbour Search. find point in A closest to q. Can get poly$(\log n)$.
- Do two polytopes intersect? Can get $O(\log^2 n)$ time.
- Is a query point inside an n-vertex polytope? $O(\log n)$ time.
- point location in a planar subdivision. $O(\log n)$.

But – Preprocessing is unrealistic for massive datasets, even when it takes linear time.

Other ways to achieve sublinear time?

- Dynamically maintain a solution. Update the Euclidean Minimum Spanning tree when adding a point in $O(\sqrt{n}\log n)$ [Eppstein '95].
- Use specialized data-structures. Approximate the Euclidean Minimum Spanning tree in $O(\sqrt{n})$ [CEFMNRS '03].

Can we do without all these?

Outline

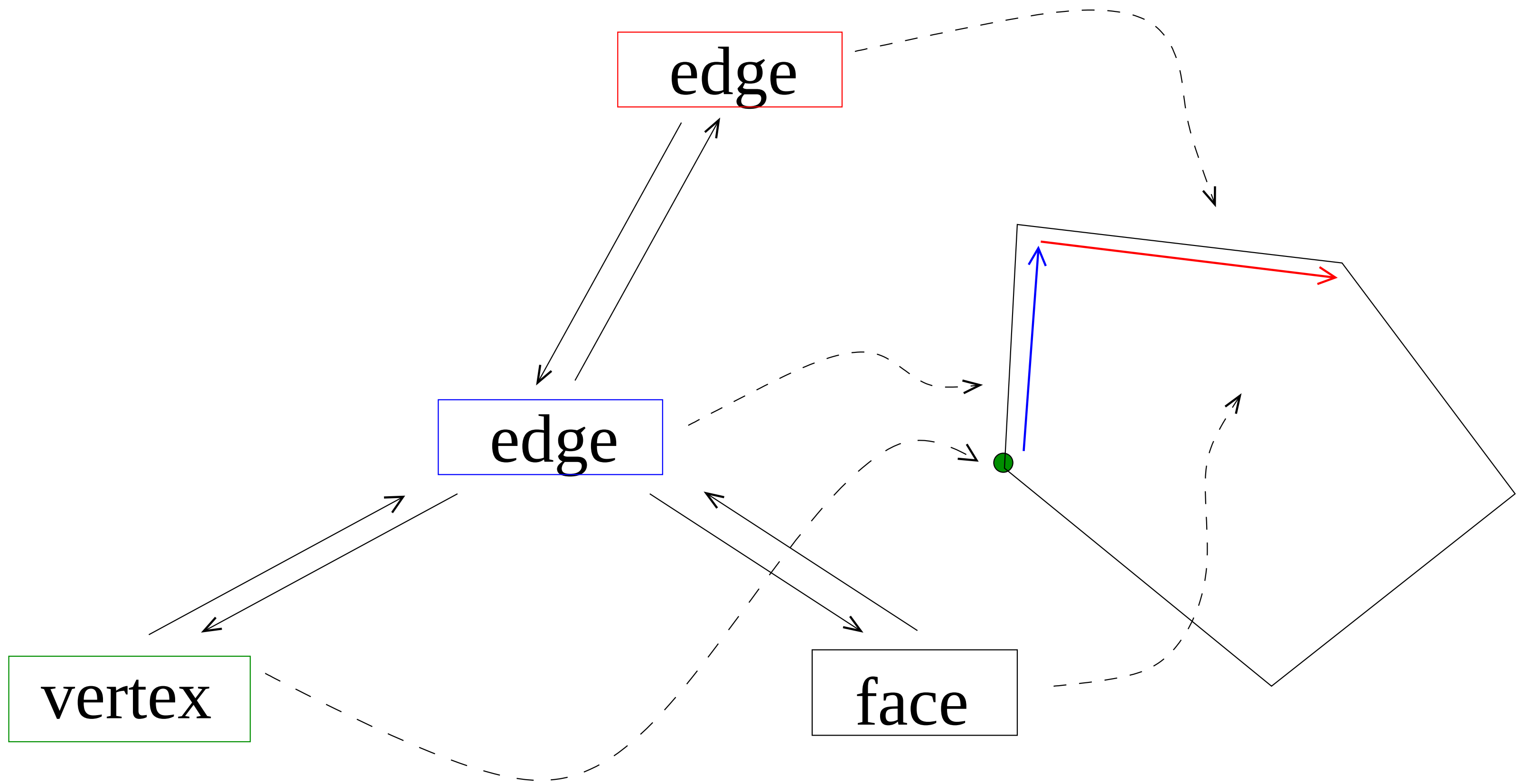
- Present the standard input model.
- The necessary use of randomization and the field of *property-testing*.
- Previous work.
- The problems we deal with and our Las-Vegas algorithms running in sublinear time $O(\sqrt{n})$.
  1. detecting intersection of convex polytopes in 3D.
  2. ray-shooting, nearest neighbour from a point to a polytope.
  3. point location in Voronoi Diagram and Delaunay triangulation.
  4. approximate the volume of polytopes.
  5. approximate the shortest path on polytopes.
- Open questions.

The input model

- Input in standard representation with no extra assumptions.
- Planar subdivision and 3-D polytopes are given in classical edge-based structure.
- The main theme : there is table holding the edges, and some relations to the neighbouring objects.

DCEL : Doubly Connected Edge List

Collections of records for edges, vertices and faces. various operations. can sample a random edge in constant time.



Randomization in sublinear algorithms and property-testing

If not reading the whole inputs, cannot do much deterministically.

Property-testing: Sublinear time algorithms to check combinatorial or geometric properties of an object.

- An object which does not satisfy the property has a *distance* measuring how far it is from having the property.
- Object has the property? say YES. *far* from having the property? say NO.
- Example: Check whether a set of points is in convex position. it is far from having this property if a large number of points are in the interior of the convex hull [CS '01]. SAY how does it differ?

Previous work

[DMZ '96], [MSZ '98], a deterministic algorithm for point location in 2-D and 3-D Delaunay triangulation of n points takes an average time $O(n^{1/3})$ in 2-D and $O(n^{1/4})$ in 3-D.

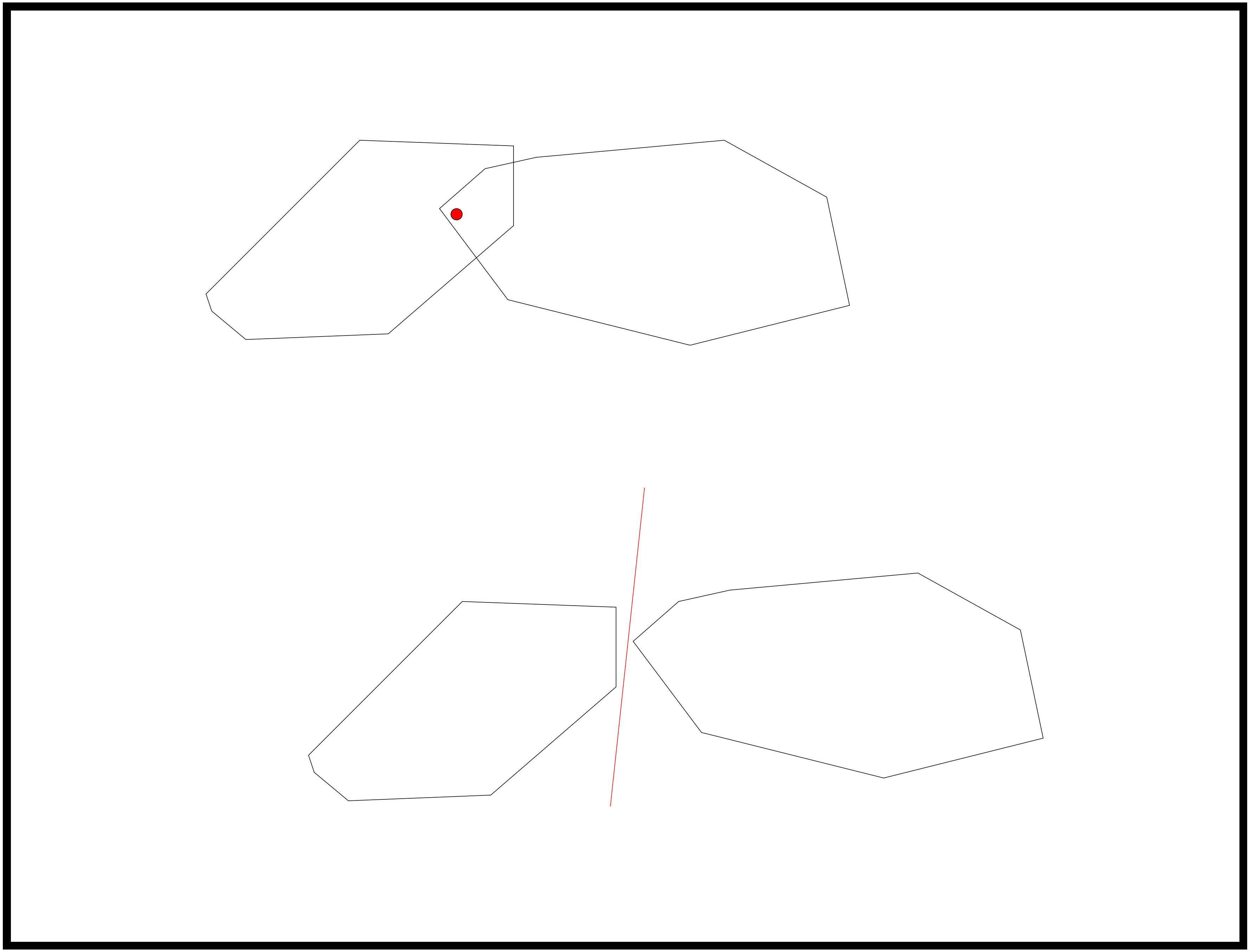
No preprocessing. No assumptions on the input model are provided.

Our Results

Sublinear algorithms for several classical problems. n is input size.

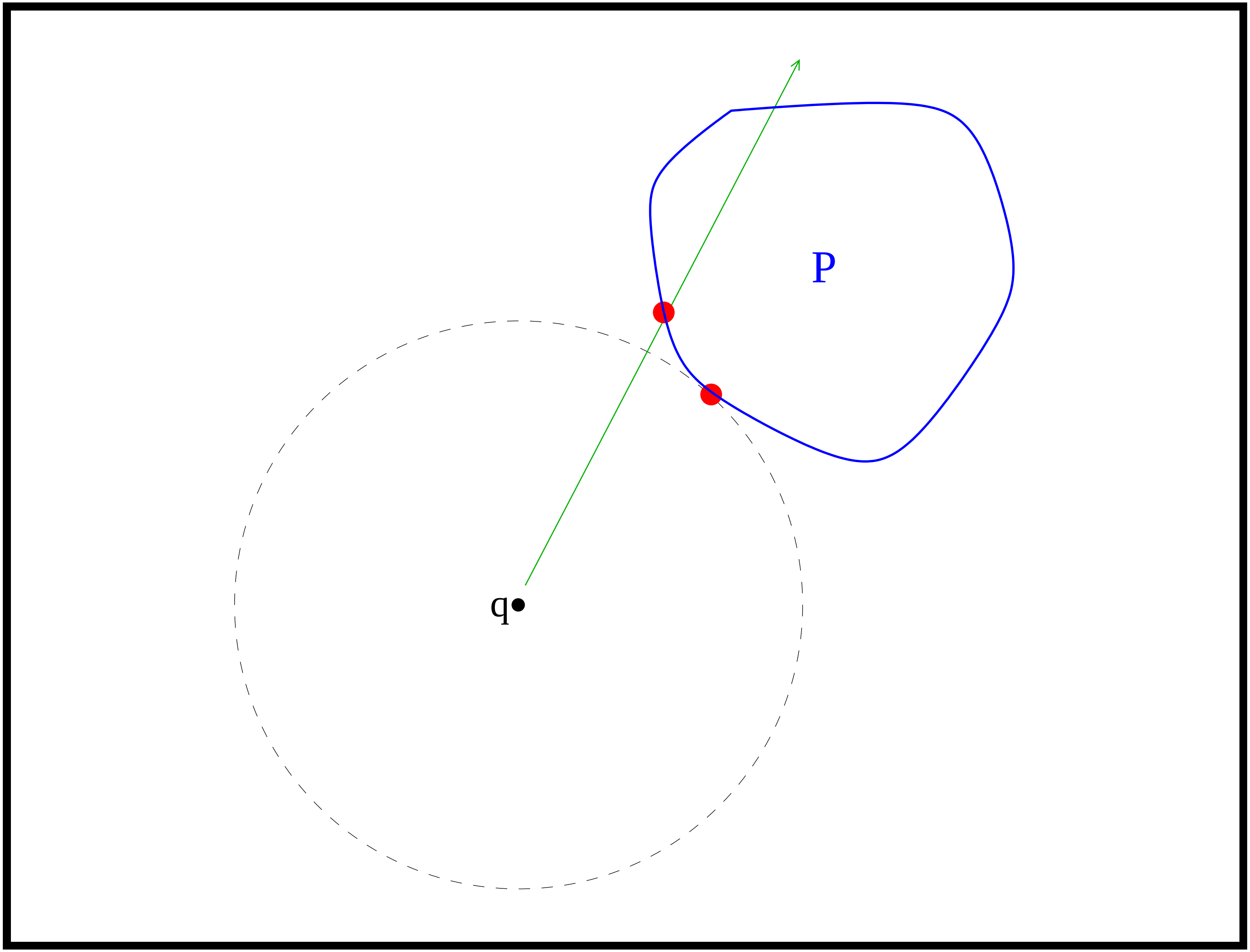
Algorithms are Las Vegas: never err, and we measure expected time.

- checking intersections of two convex polygons (polytopes) in 2-D (3-D); Optimal $O(\sqrt{n})$ time.



Our results

- Checking intersections of two convex polygons (polytopes) in 2-D (3-D); Optimal $O(\sqrt{n})$ time.
- Ray-shooting in polytopes; Nearest neighbour in polytopes; point location in 2-D Delaunay triangulation and Voronoi diagrams. Optimal $O(\sqrt{n})$ time.


P
q

Our results

- Checking intersections of two convex polygons (polytopes) in 2-D (3-D); Optimal $O(\sqrt{n})$ time.
- Ray-shooting in polytopes; Point location in 2-D Delaunay triangulation and Voronoi diagrams; optimal O($\sqrt{n}$) time.
- $(1+\varepsilon)$-approximate the volume of a polytope, in $O(\varepsilon^{-1}\sqrt{n})$ time.

Our results

- Checking intersections of two convex polygons (polytopes) in 2-D (3-D); Optimal $O(\sqrt{n})$ time.
- Ray-shooting in polytopes; Point location in 2-D Delaunay triangulation and Voronoi diagrams; Optimal O($\sqrt{n}$) time.
- $(1+\varepsilon)$-approximate the volume of a 3-D polytope, in $O(\varepsilon^{-1}\sqrt{n})$ time.
- $(1+\varepsilon)$-approximate the shortest path between two points on the surface of a polytope, in $O(\varepsilon^{-5/4}\sqrt{n})$.

Our results

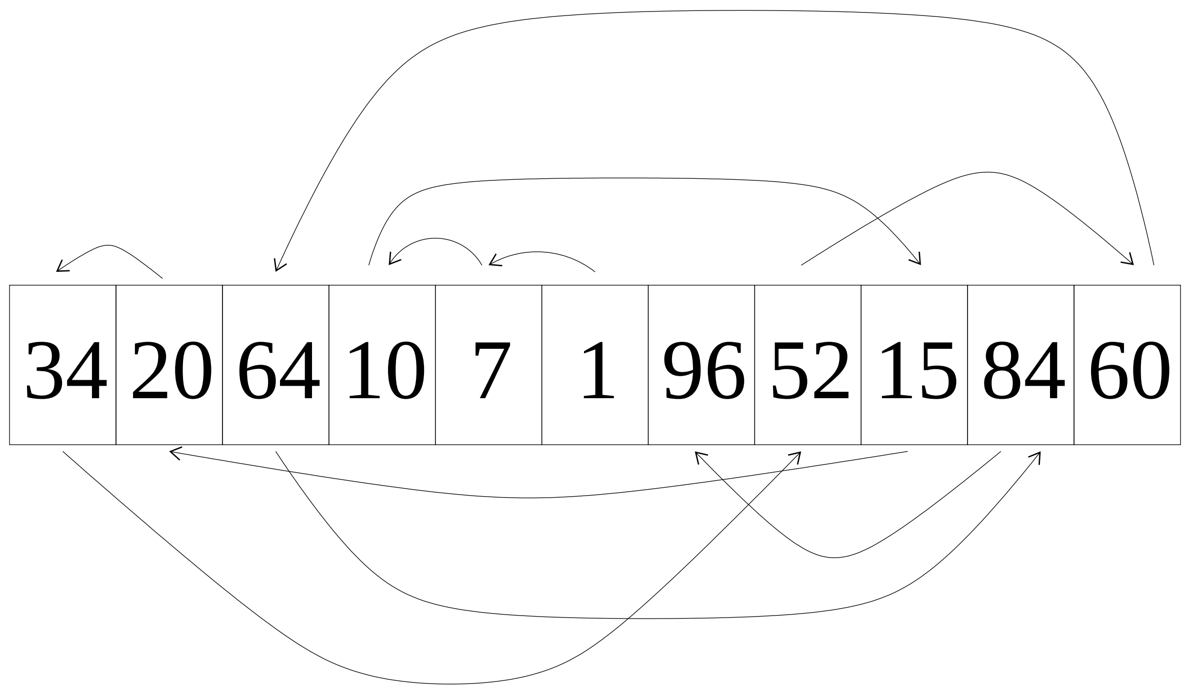
- Checking intersections of two convex polygons (polytopes) in 2-D (3-D); Optimal $O(\sqrt{n})$ time.
- Ray-shooting in polytopes; Point location in 2-D Delaunay triangulation and Voronoi diagrams; Optimal O($\sqrt{n}$) time.
- $(1+\varepsilon)$-approximate the volume of a 3-D polytope, in $O(\varepsilon^{-1}\sqrt{n})$ time.
- $(1+\varepsilon)$-approximate the shortest path between two points on the surface of a polytope, in $O(\varepsilon^{-5/4}\sqrt{n})$.
- Supply a new and most efficient construction of a *wrapper* : a polytope containing the original polytope with small number of vertices and which approximates shortest path on the original polytope.

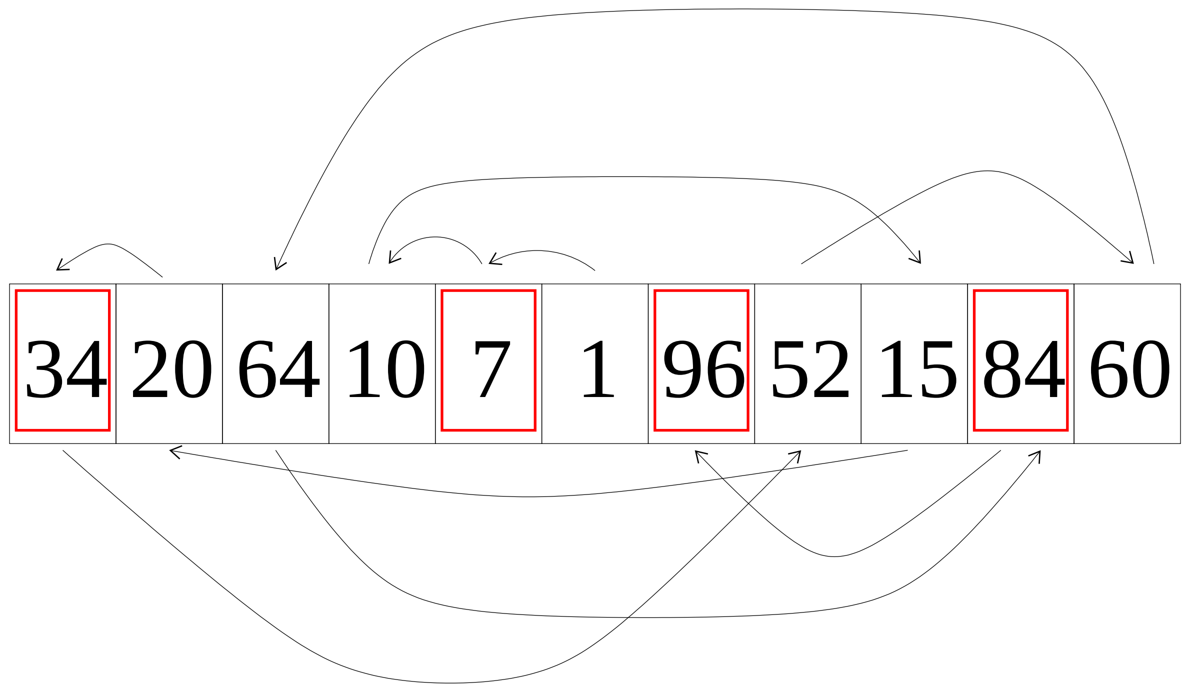
A warmup: successor searching

Input : n keys in an array. A linked list leading from an element to its successor. A key q. Output : Smallest number in the list bigger than q.

Of course if the table itself is sorted, can do in $O(\log n)$

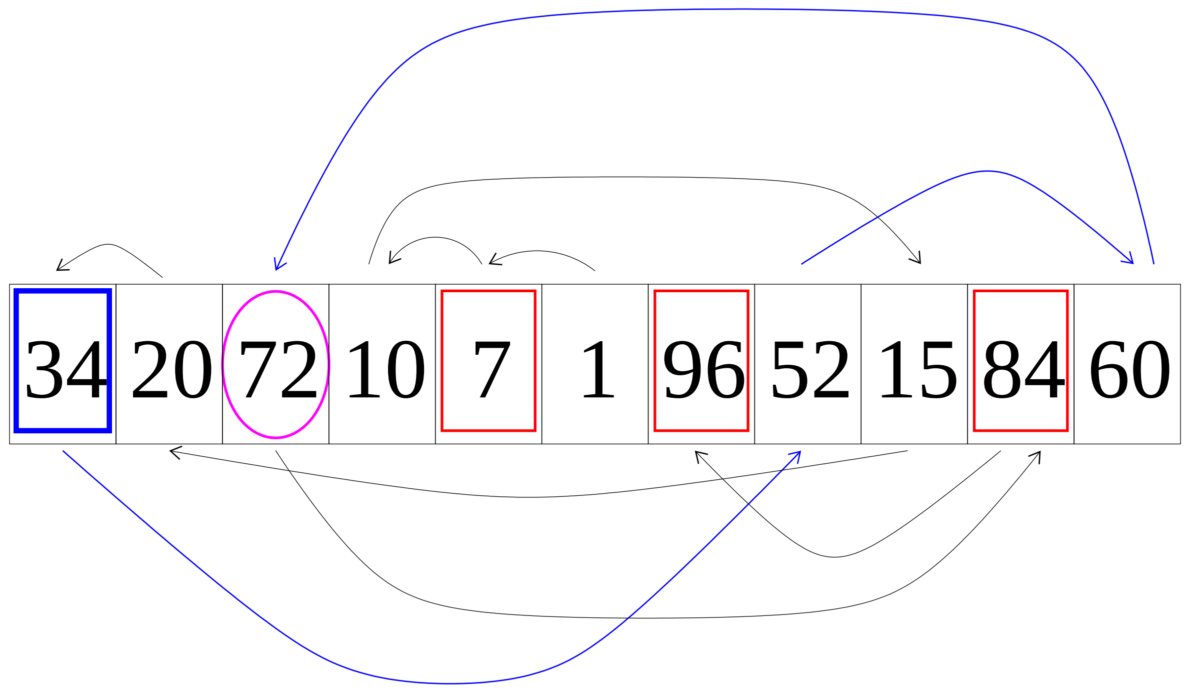
34	20	60	10	7	1	96	52	15	84	64


34 20 64 10 7 1 96 52 15 84 60



q=70

Choose r elements at random.



q=70

Find the predecessor among those; traverse the original list via the links.

Analysis and optimality

Expected time is $r + n/r$. Optimize to get $O(\sqrt{n})$ time.

This alg' is optimal. As always, to give a lower bound for a randomize algorithm, we use Yao's minimax principle: provide a distribution over inputs and lower bound the expected time of a deterministic algorithm.

Lower bound

Input: the numbers $1, \ldots, n$ are ordered (in the table) by a random permutation, and we need to find the number n.

The two operations we have are

(T) pick an arbitrary element from the table;

(L) go to the previous/next from the current element;

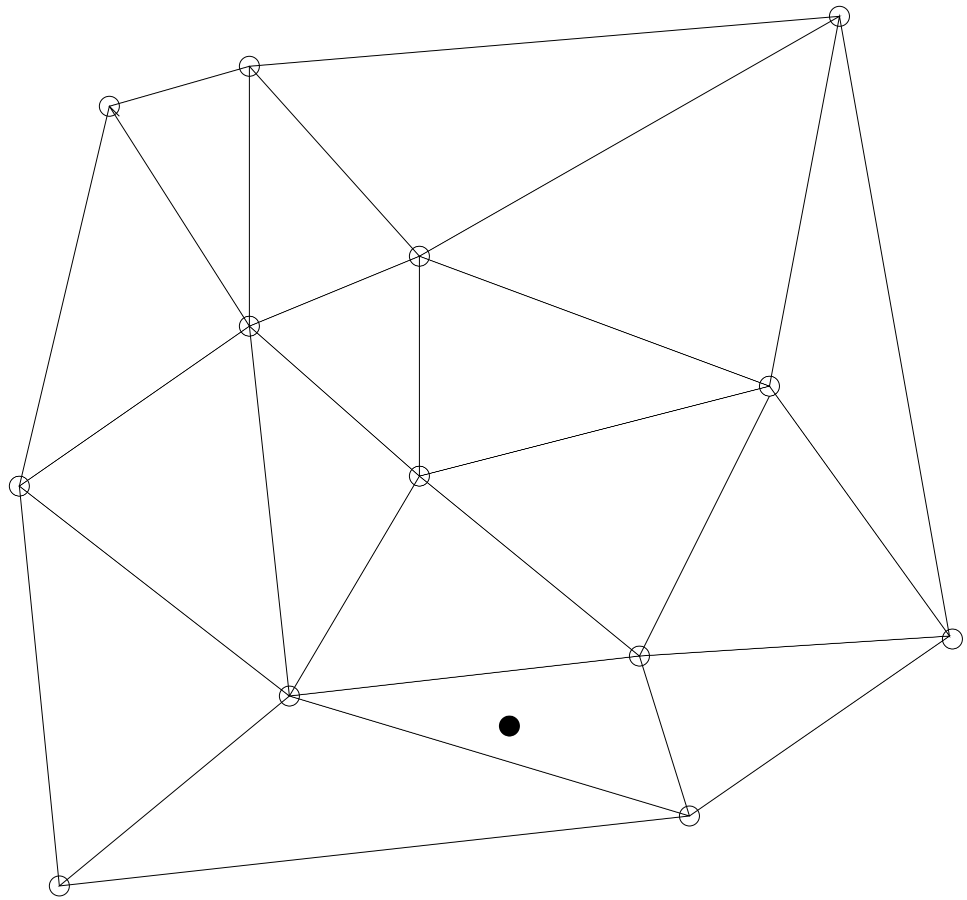
Enough to show that if $O(\sqrt{n})$ T-operations are performed, then with const' prob' we do not hit any of the last $\sqrt{n}$ elements.

This holds since an element that is obtained by a T-operation is random among all unseen elements. Now Birthday paradox.

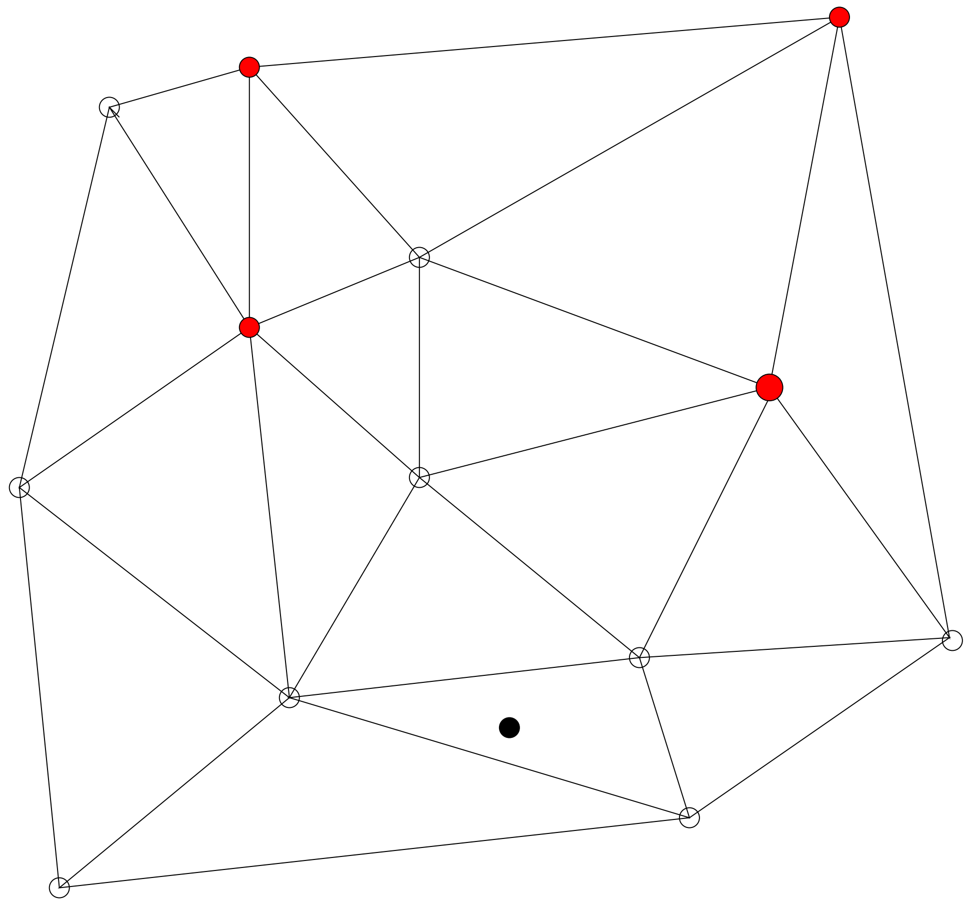
DMZ algorithm

<u>Problem</u> : Given a Delaunay Triangulation of a set of n points A in the plane, and a point q, find a triangle containing q.

DMZ algorithm

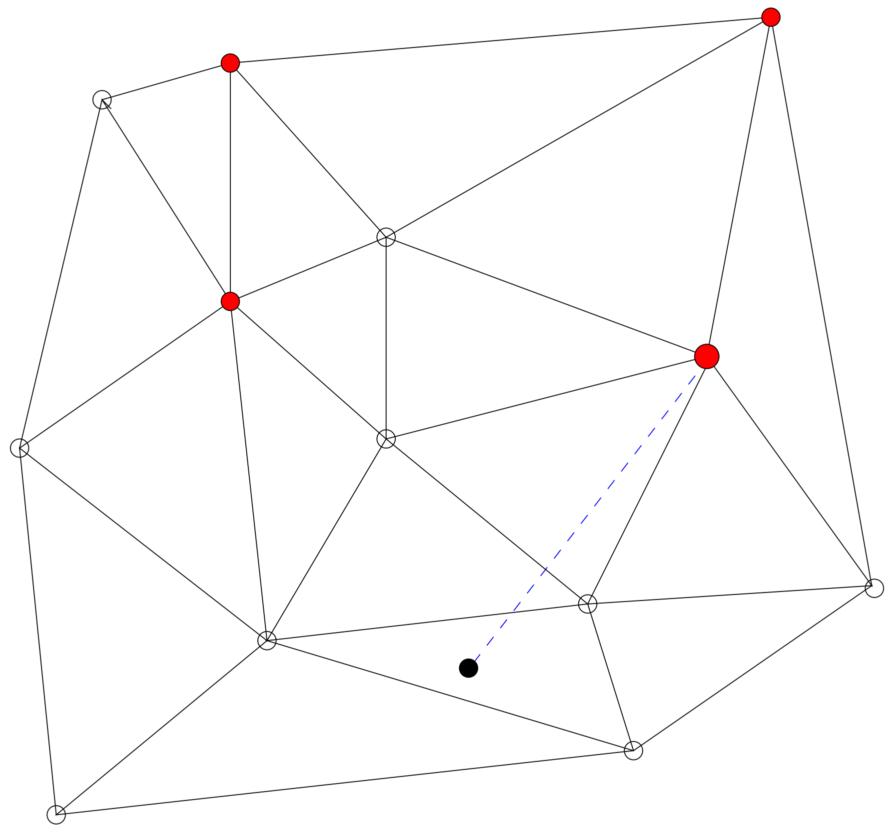


DMZ algorithm



Sample r vertices; find the closest to q.

DMZ algorithm



Traverse all triangles crossing the line segment from closest point to query. there are $O(\sqrt{n/r})$ in average.
total average time $= r + \sqrt{n/r}$. optimize to get $n^{1/3}$ time.

The schema so far

- Small sample. use table-operations.
- Exhaustive naive algorithm to point to the interesting part of the sample.
- Pin down to a restricted region and use linked-list-operations to finish.

Intersecting polytopes

Polytopes P, Q with n vertices each (and so $O(n)$ edges and faces);
Output : A point in the intersection or a separating hyperplane.

If preprocessed then can do in $O(\log^2 n)$ time [CD '87].
But without preprocessing?

We show $O(\sqrt{n})$; asymptotically optimal;

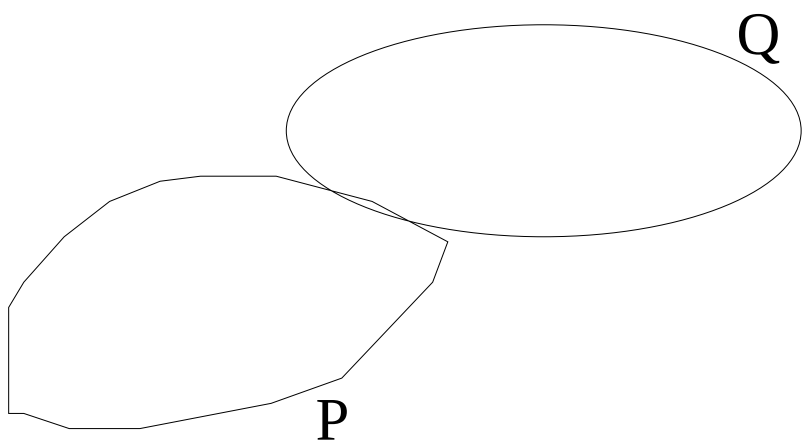
First step. There is a way to get a linear time algorithm

- Can solve the intersection-problem of two polytopes in $\mathbf{R^d}$ in <u>linear</u> time when d is constant.

- How? Write a *linear-program* for the problem. Look for a hyperplane that separates them. Need $d+1$ variables and $2n$ constraints

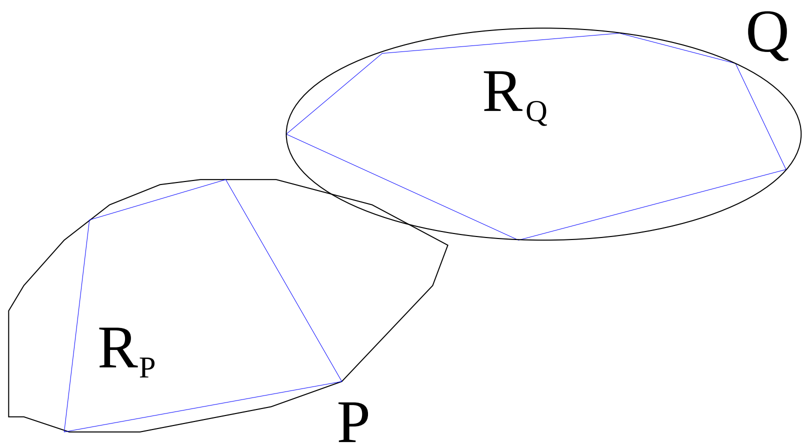
- Can solve LP of constant dimension in $O(n)$ time [Megiddo ,Dyer, Frieze, Kalai, Matoušek]

Algorithm - the case of 2D polygons



1. Sample $\sqrt{n}$ edges from each of the polygons: Consider the polytopes spanned by the sample. Do not compute the polytopes.

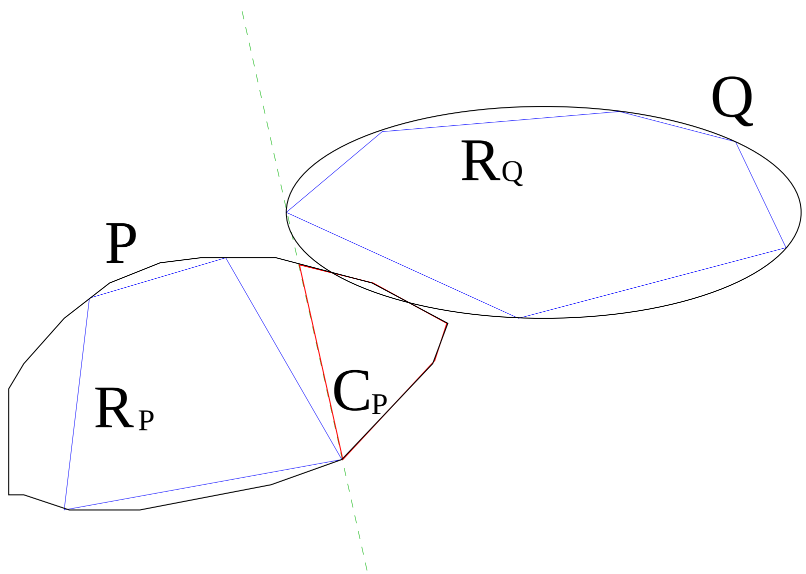
Algorithm - 2D



1. Sample r edges from each of the polygons: Consider the polytopes spanned by the sample. do not compute the polytopes.

If they intersect (check with LP) we are done.

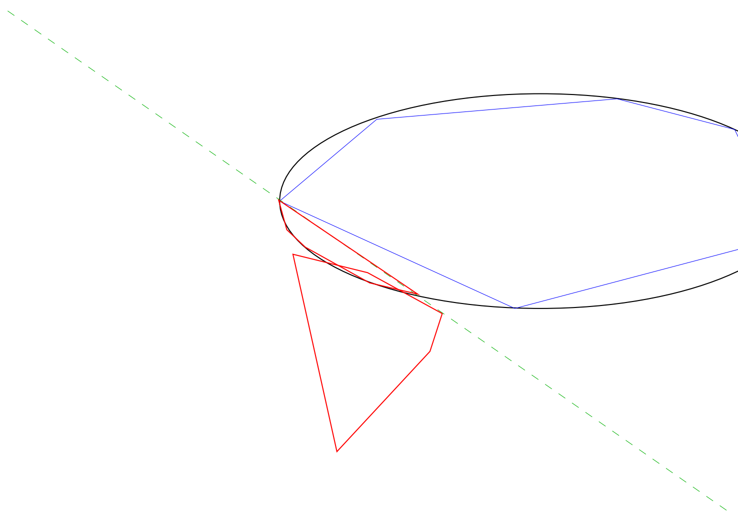
Algorithm - 2D



2. Else, take a plane L separating and tangent to both.

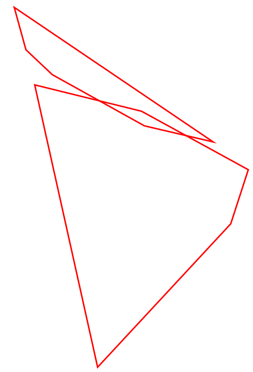
only need to check if the "leftovers" intersect the original polytopes.

Algorithm - 2D



apply LP again to get intersection between C_p and R_q or separating hyperplane. Get another leftover polytope of Q.

Algorithm - 2D



check linearly (LP) the intersection of the two leftovers.

Analysis - 2D

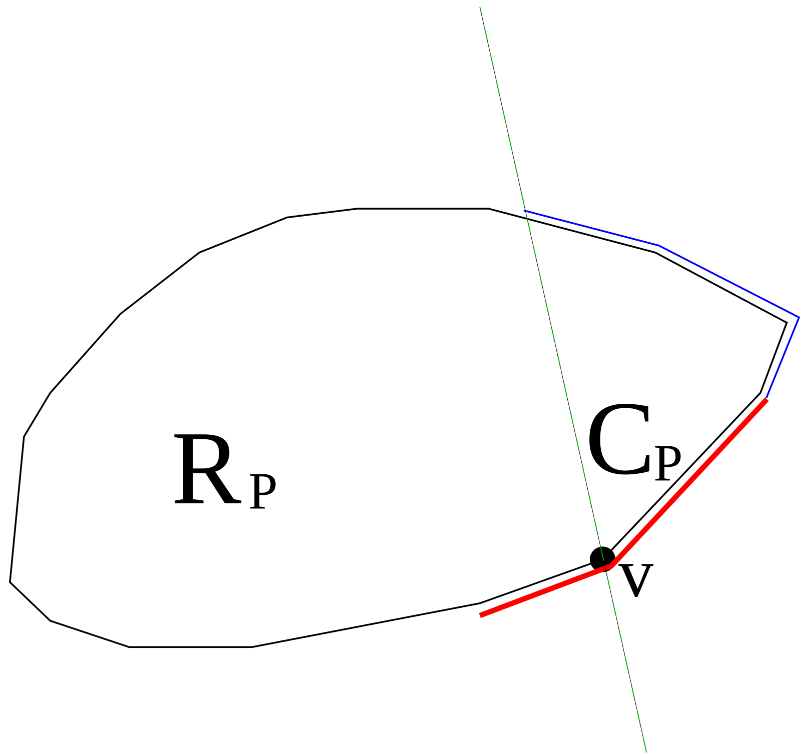
construct leftover lemma: C_p computable in $O(|C_p|)$.

small leftover lemma: $E|C_p| = O(n/r)$.

Conclusion : total time is $O\left(\frac{n}{r} + r\right)$.

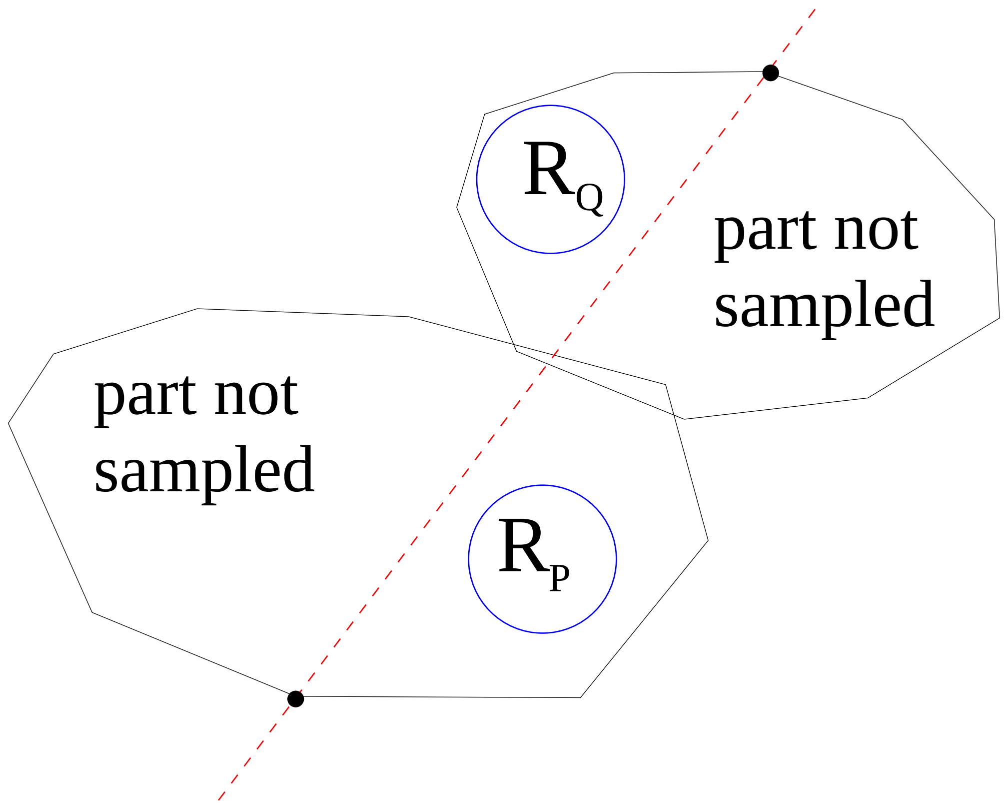
optimizing gives $O(\sqrt{n})$.

construct-leftover



Look at the two neighbours of v. if one of them crosses the separating line, follow it up to the crossing back. if none it C_p is empty.

Analysis - small leftover lemma

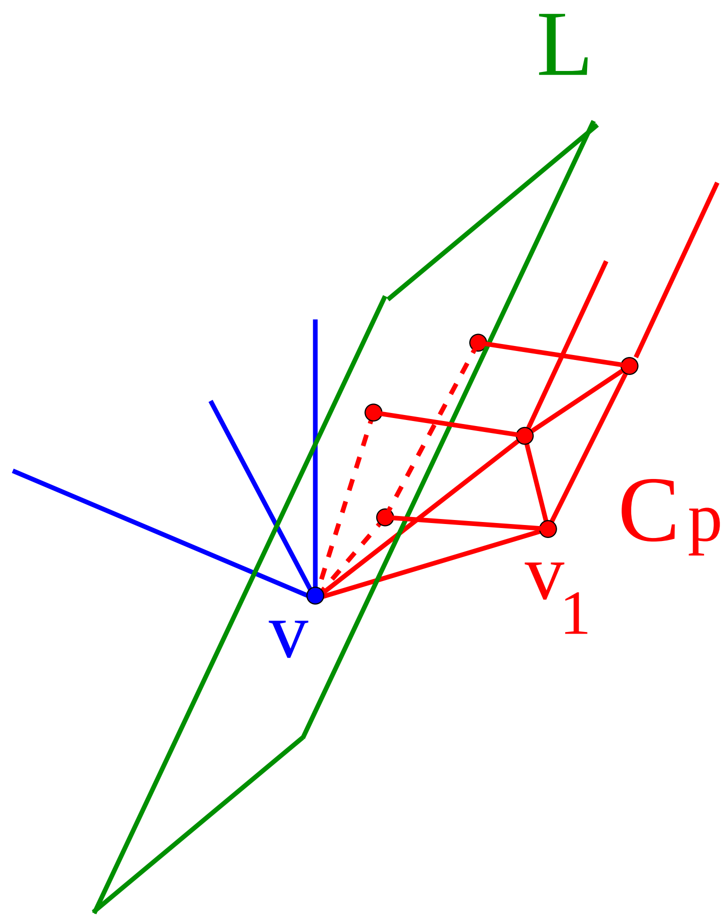


- If the size $>> n/r$ very unlikely to be missed when $|\text{sample}| = r$.
- This gives $E|C_p| = O(n/r)\log n$. A more careful and elaborated analysis gives $E|C_p| = O(n/r)$.

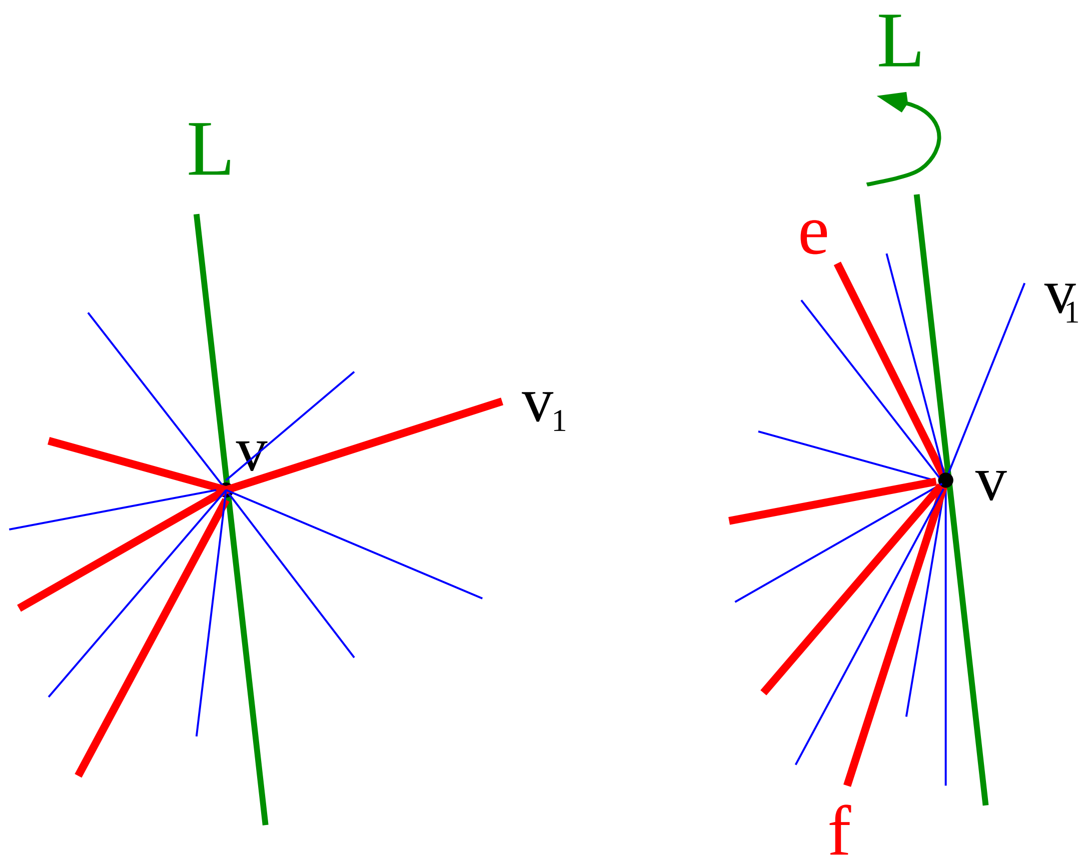
Extending to 3D

All is the same (including analysis of "small-leftover" lemma) except the "construct-leftover" procedure.

3D construct-leftover lemma: C_p is computable in $O(|C_p| + \sqrt{n})$.



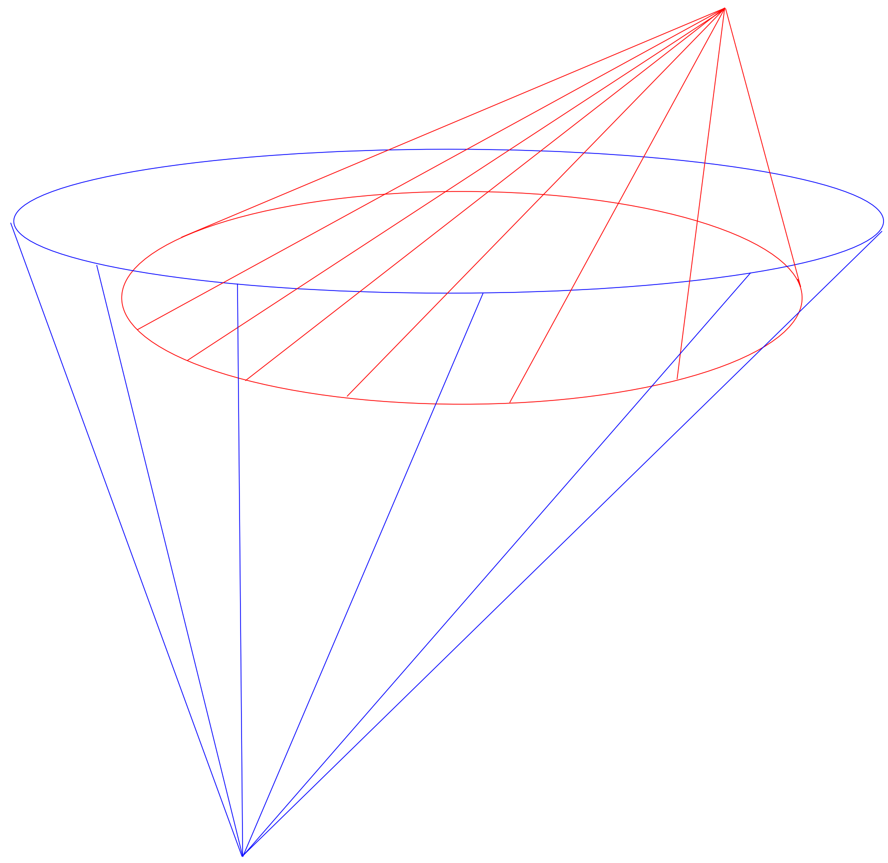
Once there is an edge from p crossing to the "other side" of the hyperplane, can continue by a DFS along the edge structure (convexity).



To find such an edge need to **sample again**; if no crossing edge in the sample, there are two extreme between which we should scan.

Dont go over the edge

It is essential to sample edges rather than, say, vertices. Here, if we sample vertices, the small-leftover lemma clearly fails.



Other problems

Using the same sampling techniques can perform

1. Ray shooting from a point to a polytope;
2. Nearest neighbour from a point on a polytope;
3. Point location in 2-D Voronoi diagrams;
4. Point location in 2-D Delaunay triangulation;
5. And others; all optimal $O(\sqrt{n})$.

Approximating volume of convex 3D polytopes

The result: An algorithm to approximate the volume in $O(\sqrt{n}/\varepsilon)$ time.

Idea: Dudley's construction $+$ the nearest-neighbour sublinear alg $+$ additional use of our sublinear primitives.

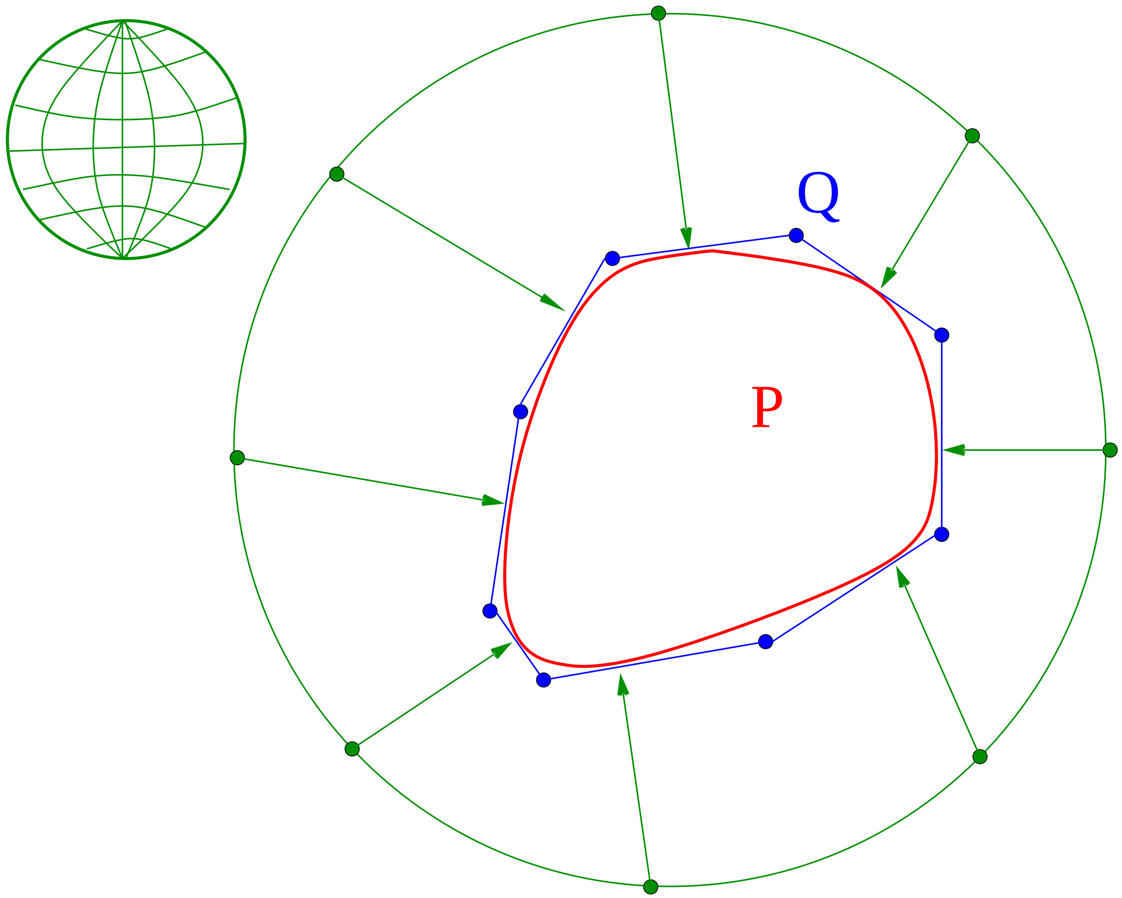
Observe : exact volume computation is easily done in $O(n)$ using triangulation.

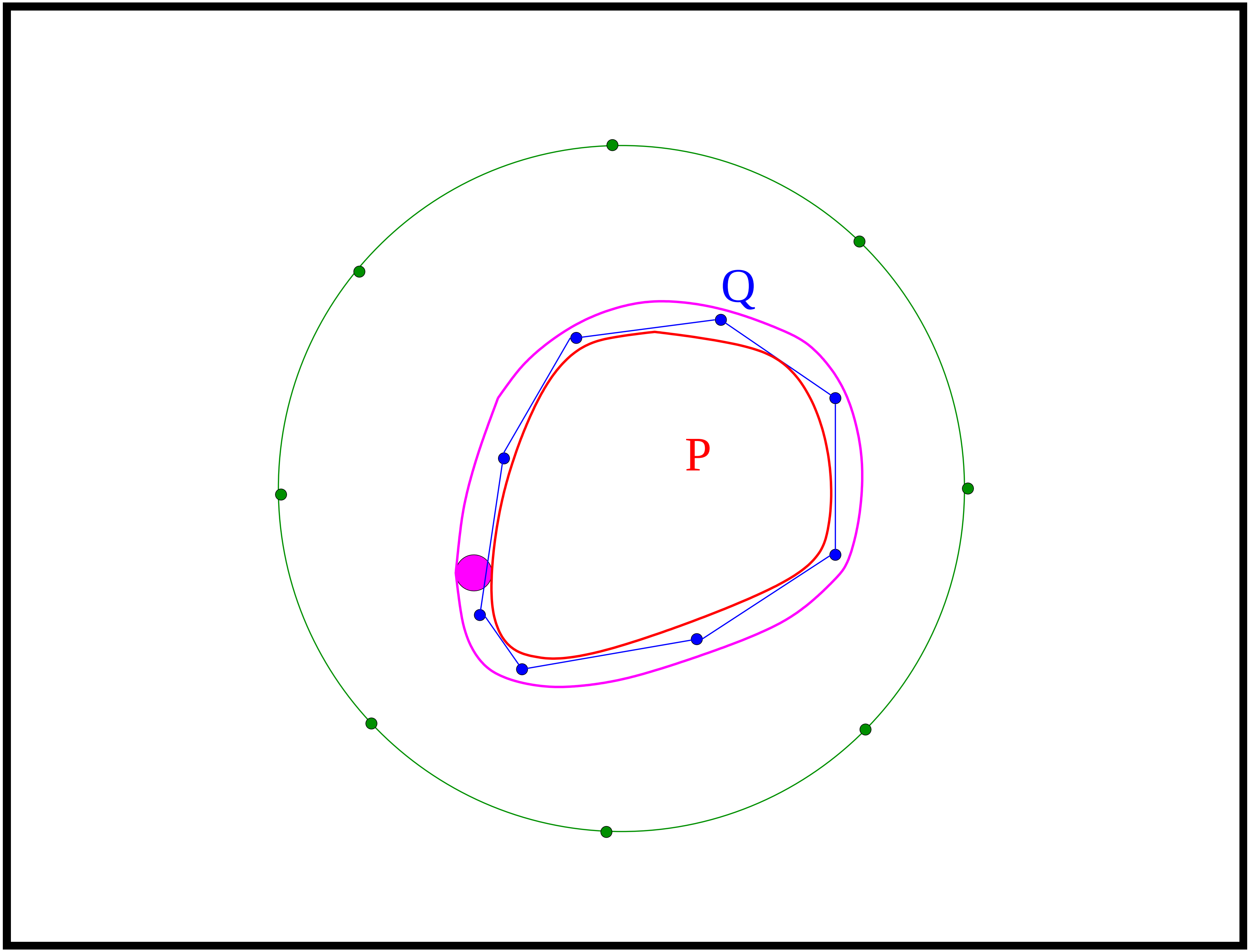
Given a polytope P with n vertices, we want to approximate it by Q with much fewer vertices.

Dudley's construction

[Dudley '74] let P be contained in the unit ball; there is a polytope Q with $O(1/\varepsilon)$ vertices, such that: (1) $P \subseteq Q$
(2) Hausdorff distance between P, $Q \leq \varepsilon$.

Q formed by:
(1) sphere of radius 2;
(2) longitude/latitude grid of resolution $\sqrt{\varepsilon} \times \sqrt{\varepsilon}$;
(3) project $(\textbf{nearest-neighbors})$ the grid onto P.


Q
P

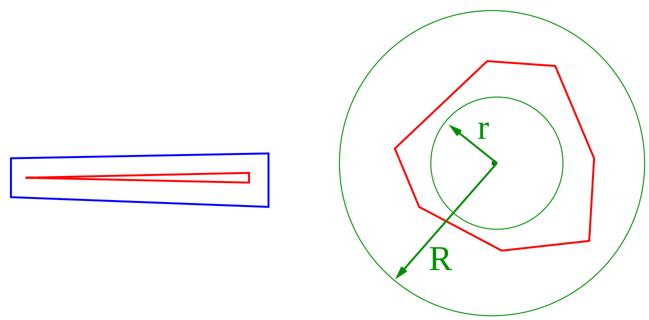

Q
P

The Algorithm

- Compute Q using $1/\varepsilon$ nearest-neighbour operations. time $= O(\sqrt{n}/\varepsilon)$.
- Compute exact volume of Q. time $= O(1/\varepsilon)$.

From small Hausdorff distance to volume approximation

Q does not necessarily well approximates volume :



P should be fat! Goal : Affine transform P to P'.

From small Hausdorff distance to volume approximation

How to transform?

- Find a constant approximating volume inside P with constant number of vertices. constant many applications of nearest neighbour and ray-shooting.
- Compute the largest ellipsoid enclosed (Löwner-John ellipsoid) in constant time.
- Use it to rescale P to get P' which is fat.

Shortest paths on a convex polytope

Given s, t on the surface of a polytope P. Compute the shortest path from s to t on its surface.

Well studied in computational geometry. <u>motivations</u>: robotics. geographic info systems. computer assisted surgery. many more.

Exact shortest path

previous work:

- [Sharir and Schorr '86] $O(n^3 \log n)$
- [Mitchell, Mount, Papadimitriou '87] $O(n^2 \log n)$
- [Chen and Han '90] $O(n^2)$
- [Kapoor '99] $O(n \log^2 n)$

Approximate shortest paths

Previous work

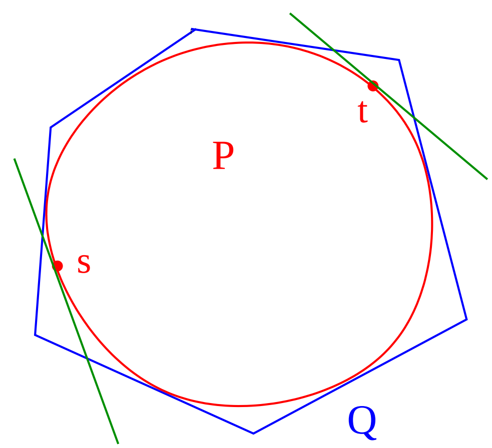
- [Hershberger and Suri '95]:
  2-approximation, $O(n)$ time;
- [Agarwal, Har-Peled, Sharir, Varadarajan '97]:
  $(1+\varepsilon)$-approximation, $O(n+\varepsilon^{-3})$ time;
  $O(n\log(1/\varepsilon)+\varepsilon^{-3})$ time if output the path;
- [Agarwal, Har-Peled, Karia '00]:
  $(1+\varepsilon)$-approximation, $O(n/\sqrt{\varepsilon}+\varepsilon^{-4})$ time;
  practical, implemented;

Our result: a $O(\varepsilon^{-5/4}\sqrt{n})$ algorithm.

Note : We approximate the length or supply a path outside of the interior of P but not necessarily on its surface. This is necessary.

A problem of independent interest

An approximating polytope Q ("ε-wrapper") is wanted with the property: For an s, t far enough on the surface of P, their shortest path Q^+ approximates to within relative error ε the shortest path on P. How many vertices must Q have?

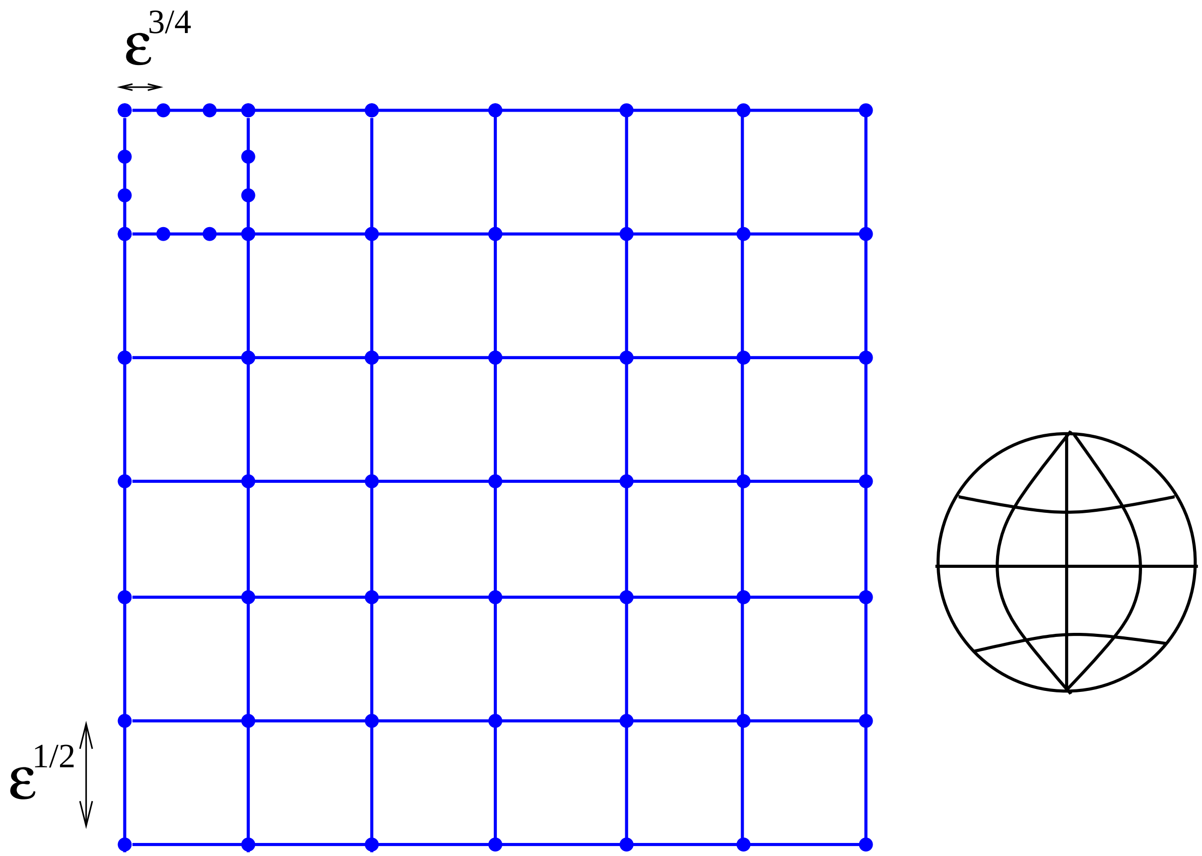


Theorem: There exists such Q with $O(\varepsilon^{-5/4})$ vertices.
Improves on $O(\varepsilon^{-3/2})$ in [Agarwal, et al. '97].

The Algorithm

- Truncate polytope so that s and t are far enough compared to its radius (use the sublinear primitives).
- Construct Q. Project a certain grid on the sphere onto P (not a simple grid this time). There are $\varepsilon^{-5/4}$ points in the grid, so time $= O(\varepsilon^{-5/4}\sqrt{n})$.
- Use an exact algorithm to find a shortest path on Q^{+}.
  time $= O(\varepsilon^{5/4} \log 1/\varepsilon)$.

Our construction



Open Questions

- What makes a problem solvable in sublinear time?
- General planar subdivision. Conjecture - no sublinear alg. Techniques for showing linear lower bounds?
- Can we do something for nonconvex bodies?
- More efficient “wrapper” construction?

Other things I study

- Embeddings of metric spaces; the relevance to approximation algorithms.
- Convex and mathematical programming. cutting-plane methods.
- Concrete complexity. Example : How can we define *dynamic-programming*, and how can we bound its strength?
- Bioinformatics. Particularly combinatorial problems that arise naturally in the analysis of genomic sequences and also the geometry of *edit-distance*.