COT 6936: Topics in Algorithms

Giri Narasimhan

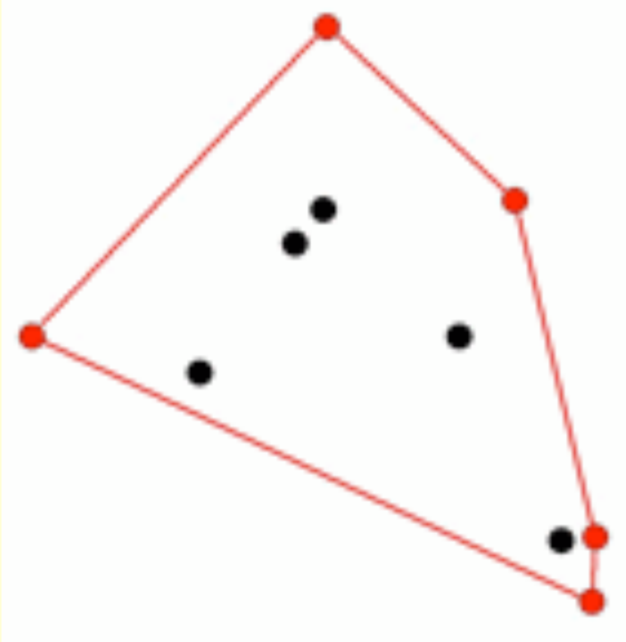
ECS 254A / EC 2443; Phone: x3748

giri@cs.fiu.edu

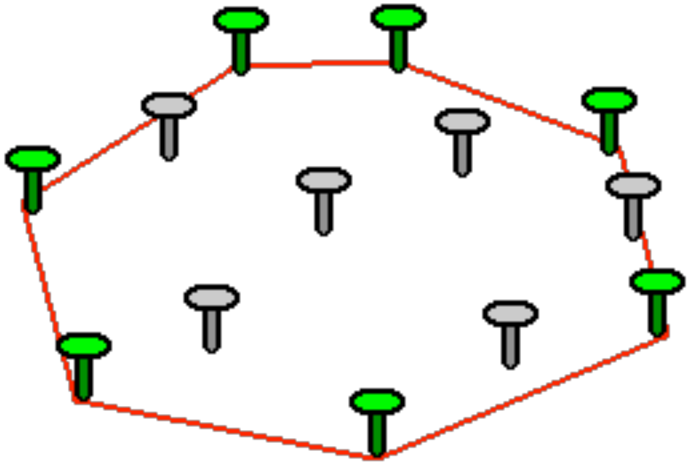
http://www.cs.fiu.edu/~giri/teach/COT6936_S12.html

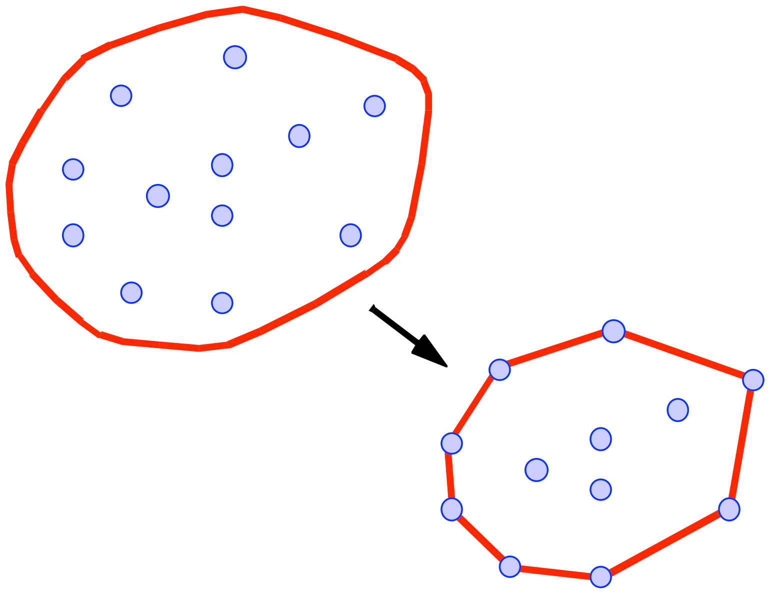
Convex Polygons

- Convex region: A region in space is called convex if line joining any two points in the region is completely contained in the region.
- Convex hull of a set of points, S, is the smallest convex region containing S.



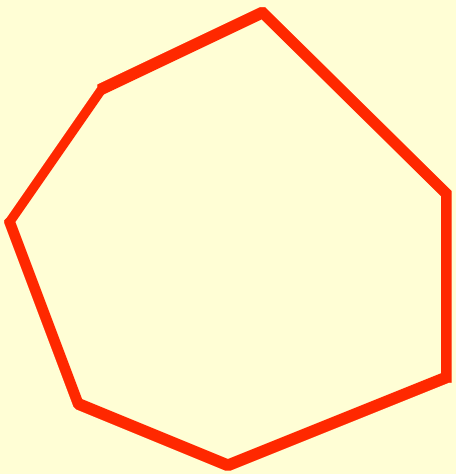
Rubber Band Analogy

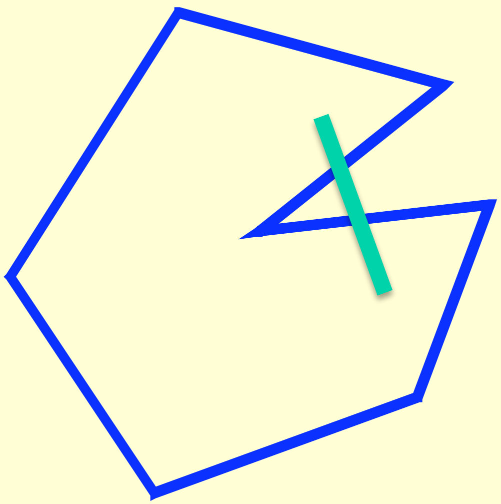




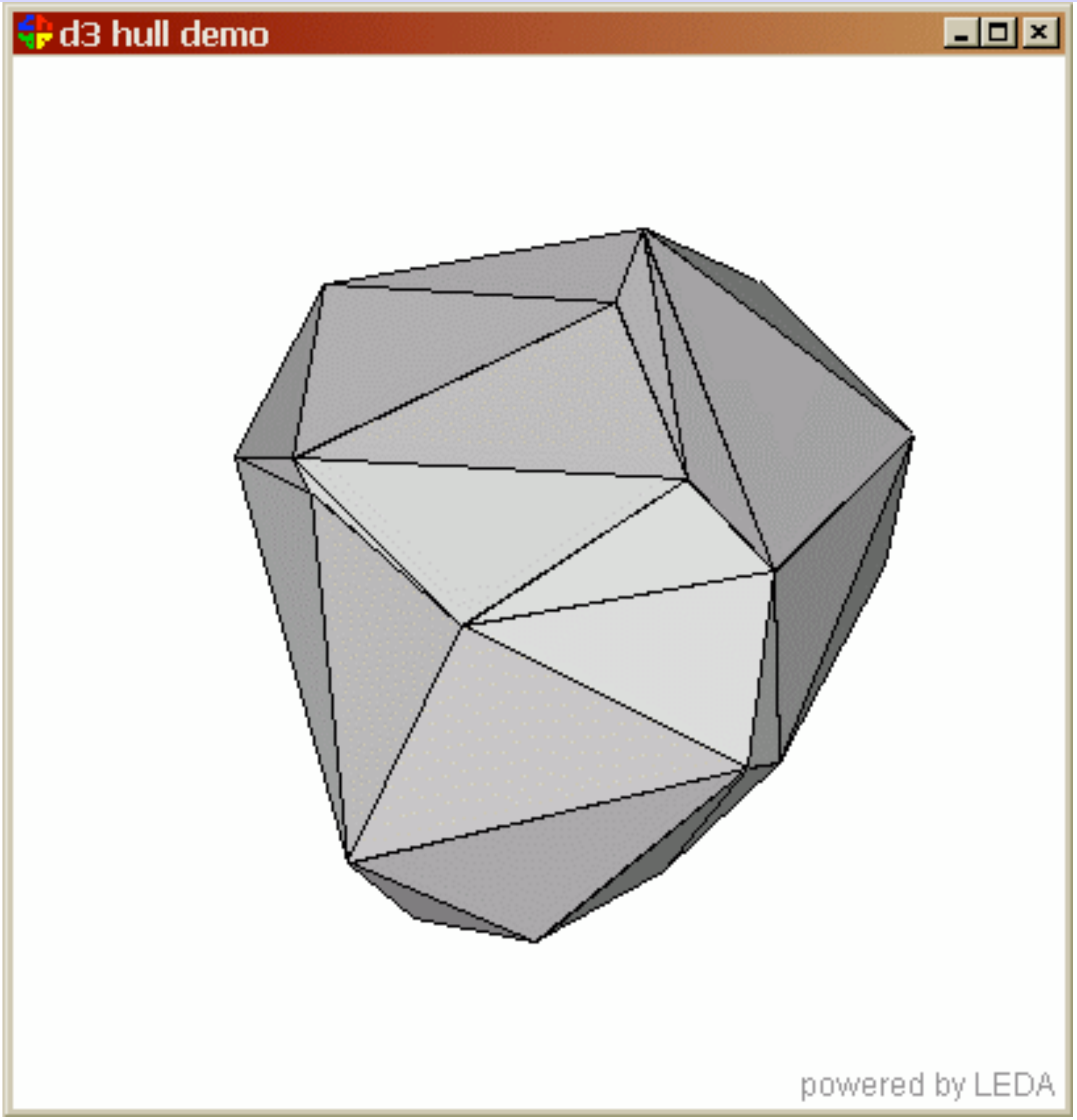
Non-convex polygons

- Convex vs Non-convex





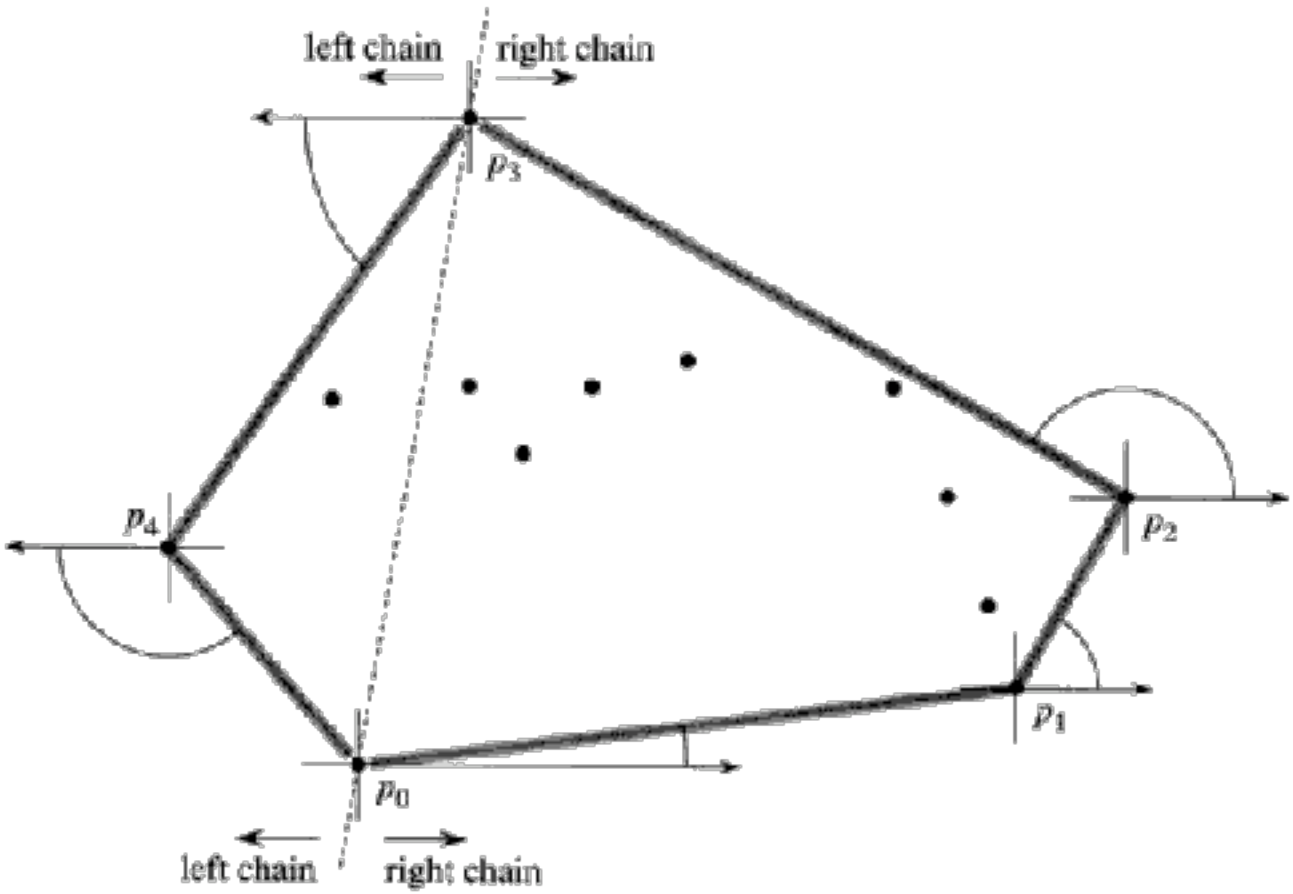
3D convex hulls



Convex Hull: Graham Scan applet

- http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/ConvexHull/GrahamScan/grahamScan.htm
  - Main cost: sorting
    - O(n log n)

Package Wrapping: Jarvis March



Package Wrapping: Jarvis March

- Time complexity
  - (Cost of iteration) X (# iterations)
- Each iteration: $O(n)$
- Number of iterations = $O(n)$
- Cost = $O(nh)$
  - h = # of points on convex hull

Complexity of Convex Hull

- Graham Scan: $O(n \log n)$
- Jarvis March: $O(nh)$ [output sensitive]
- Lower Bound = $\Omega(n \log h)$

Chan's Algorithm

- Combines the benefits of both algorithms
- Partition points into n/m groups of size m
- Use Graham scan on each one
  - $O((m \log m)(n/m)) = O(n \log m)$
- Merge the n/m convex hulls using a Jarvis march algorithm by treating each group as a "big point"
  - Tangent between a point and a convex polygon with m points can be computed in $O(\log m)$ time
  - $O((n/m)(\log m)(h)) = O((n/m)h \log m)$

Chan's Algorithm

- Time Complexity = $O(n \log m + (n/m) h \log m)$
- If m = h, then time = $O(n \log h)$
- How to guess h?
  - Linear Search
    - Time complexity = $O(nh \log h)$
  - Binary Search
    - Time complexity = $O(n \log^2 h)$
  - Doubling Search (m = 1, 2, 4, 8, …)
    - Time Complexity = $O(n \log^2 h)$
  - ???

Chan's Algorithm: More tricks

- What if m = h^2?
  - Then O(n log m) = O(n log h)
- So try: m = 2, 4, 16, 256, …
  - Analysis

$$\sum_{t=1}^{\lg \lg h} n2^t = n \sum_{t=1}^{\lg \lg h} 2^t \leq n2^{1+\lg \lg h} = 2n \lg h = O(n \log h),$$

Closest Pair Problem

- **Input**: Set of points S in the plane
- **Output**: The closest pair of points in S
- **Naïve Solution**: $O(n^2)$ time
- **Divide-&-Conquer**:
  - $T(n) = 2T(n/2) + M(n)$
  - $M(n)$ = time to merge solutions to the two subproblems
  - Only need to merge 2 strips on 2 sides of vertical split
  - Naïve Solutions: $M(n) = O(n^2)$
  - Sort the points by y-coordinate: $M(n) = O(n\log n)$
  - Global sorting at the start: $M(n) = O(n)$
- **Lower Bound**: $O(n\log n)$ time
- **Randomized Algorithm**: $O(n)$ time [Rabin]

Post Office Problem

- **Preprocess**: Given set S of points in the plane representing post offices.
- **Input**: Query point p.
- **Output**: Report the closest post office to p.

1-d Post Office Problem

- **Preprocessing**: Build balanced BST on S.
  - O(nlogn)
  - Alternatively, build a sorted array on S.
- **Query Algorithm**: Given a value p, identify the smallest value larger than p and the largest value smaller than p and among the two pick the one that is closest to p.
  - O(log n)

2-d L_∞ Post Office Problem

- $L_p = ((|a_x-b_x|)^p + (|a_y-b_y|)^p)^{1/p}$
- L_2 = Euclidean distance
- $L_\infty = \max\{|a_x-b_x|, |a_y-b_y|\}$
- **Preprocessing**: Build Range Tree on S.
  - O(nlogn)
- **Query Algorithm**: Given a value p, identify the closest point to the right of p and the closest point to the left of p and among the two pick the one that is closest to p.
  - O(log n)

2-D Range Tree

- Build the X-Tree, a balanced binary search tree on set S using the x-coordinates of the points.
- For each node in the X-Tree, build a Y-Tree, a balanced binary search tree on the set of points in the subtree of that node using the y-coordinates of the points.
- **Application**: Output all points with x-coordinates in range [A,B] and y-coordinates in range [C,D].
- **Application**: Post office problem

Definitions

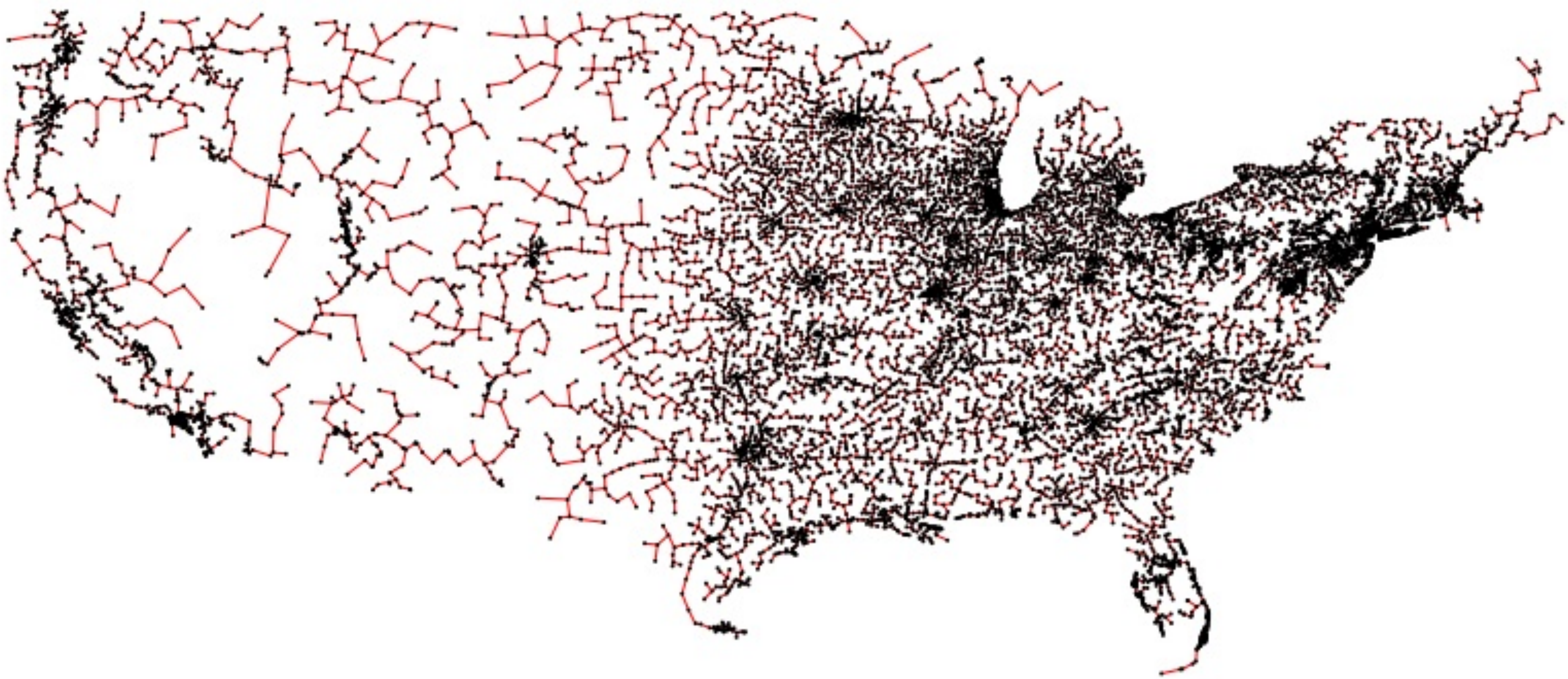
- A **Geometric Network** N has vertices S that correspond to points in $\Re^d$ and edges E whose weights equal the distance between the endpoints.

Examples:

Good Network Design

- Small size
- Small weight
- Small degree
- Small diameter
- Highly connected, highly fault-tolerant
- Planar, low genus
- Small load factor
- SMALL DILATION

MST on 13,509 cities of US



Definitions

- **Dilation** or **Stretch Factor** ($t(N)$) of a network N is the maximum amount by which the distance between some pair of vertices in the network is increased.

$$t(N) = \max_{a,b \in N} \left\{ \frac{d_N(a,b)}{|ab|} \right\}$$

- t-**Spanner** is a network with dilation at most t.

t-Spanner Networks: Examples

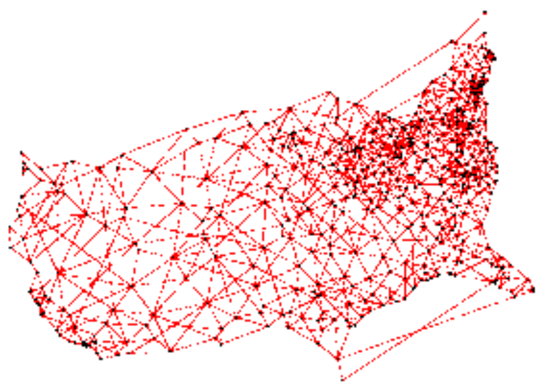
t = 10

t = 5

t = 3

t = 2

t = 1.5



t = 1.25

Application of Geometric Spanners

- Network Design – Transportation, Communication
- Distributed Algorithms – Synchronizers
- Graphics – Model Simplification
- Pattern Recognition – Approx. Nearest Neighbors
- Robotics – Approximate Shortest Path Problems
- Approximation Algorithm design [Rao and Smith]

Design of t-Spanners

- **Theta graphs**

[Clarkson 87, Keil 88, Althofer et al. 93]

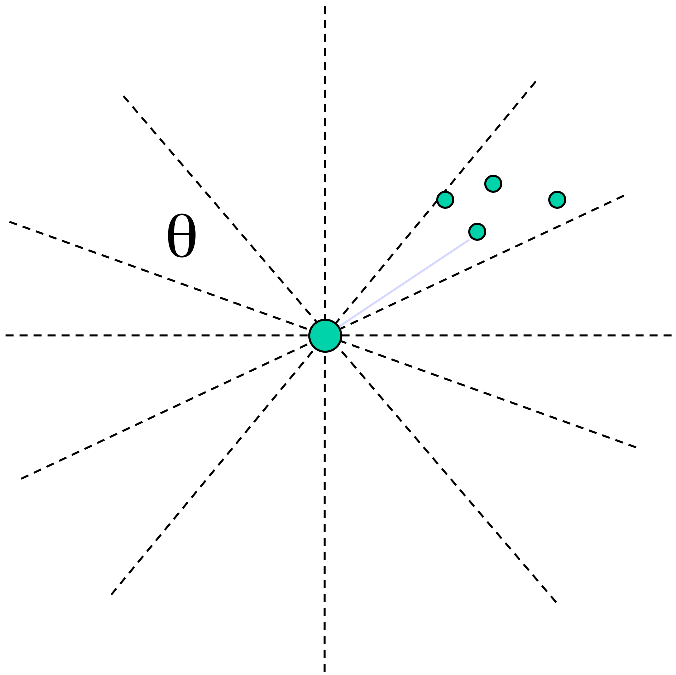
- **Greedy algorithms**

[Bern 89, Althofer et al. 93]

- **Well-separated pair decomposition**

[Callahan & Kosaraju 95]

Theta Graphs



$t = 1/(\cos\theta - \sin\theta)$

Algorithm GREEDY(G=(V, E),t)

Sort E by non-decreasing weights

Initialize G′(V,E′) to be empty

for each edge $e = (u, v) \in E$ **do**

    if $(d_{G'}(u, v) > t * wt(e))$ **then**

        Add edge e to E′

output *G′*

Well-Separated Pair Decomposition

Definition: [Callahan and Kosaraju, 95]

Given a set, S, of n points in R^d, and $s > 0$, a WSPD is sequence of pairs of subsets of S,

$$\{A_1, B_1\}, \ldots, \{A_m, B_m\}, \text{ s.t.}$$

1. Every pair of vertices $\{p, q\}$ is in exactly one pair of the decomposition.
2. A_i and B_i are well-separated for each $i = 1, \ldots, m$
3. $m = O(n)$
4. The decomposition can be computed in $O(n \log n)$ time.

t-Spanner Construction Using WSPD

[Arya, Das, Mount, Salowe, Smid, 95]

1. Compute a WSPD with $s = (4t + 4)/(t-1)$
2. For each well-separated pair (A_i, B_i)

   add an arbitrary edge between A_i and B_i.
3. Pruning Step: Remove unnecessary edges.

Analysis

- Stretch factor = t
- Max degree = O(1)
- Total weight = O(1) wt(MST)

Theorem

Given a set S of n sites in R^d, and a real number $t > 1$, there exists an efficient algorithm to construct a network G such that:

- $t(G) \le t$,
- $wt(G) = O(1) \cdot wt(MST)$, and
- maximum degree of G is $O(1)$

[Gudmundsson, Levcopoulos, Narasimhan 00]

Comparison of Spanner Construction Methods

- **Theta Graphs:** O(nlogn) time, O(n) space
  [Arya, Das, Mount, Salowe, Smid 95]
- **WSPD Spanners:** O(nlogn) time, O(n) space
  [Callahan & Kosaraju 95]
- **Greedy Algorithms:** Low weight guarantees
  O(nlogn) time, O(n) space, O(1) wt(MST) weight
  [Das, Heffernan, Narasimhan, Salowe 93, 94, 95, Gudmundsson, Levcopoulos, Narasimhan '00]

Algorithm NewGREEDY(G=(V, E),t)

Sort E by non-decreasing weights

Initialize $G'(V,E')$ to be empty

for each edge $e = (u, v) \in E$ **do**

  if $(d_{G'}(u, v) > t(1+\varepsilon) * wt(e))$ **then**

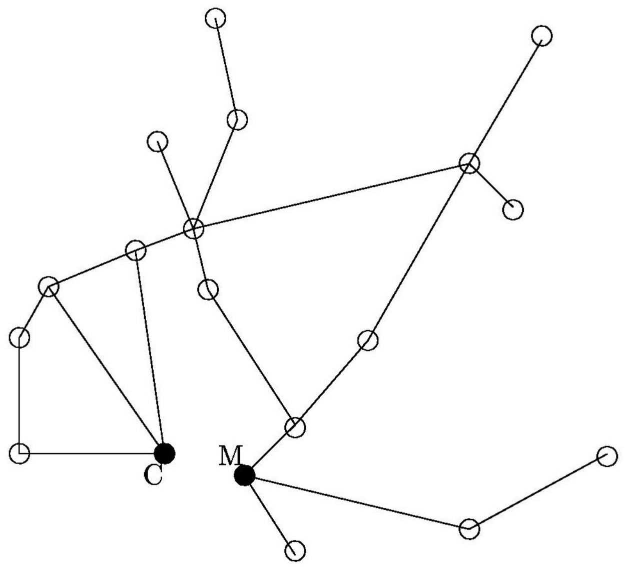
    Add edge e to E'

output G'

Computing Stretch Factors

<u>Input</u>: A geometric graph N on a set S of n sites

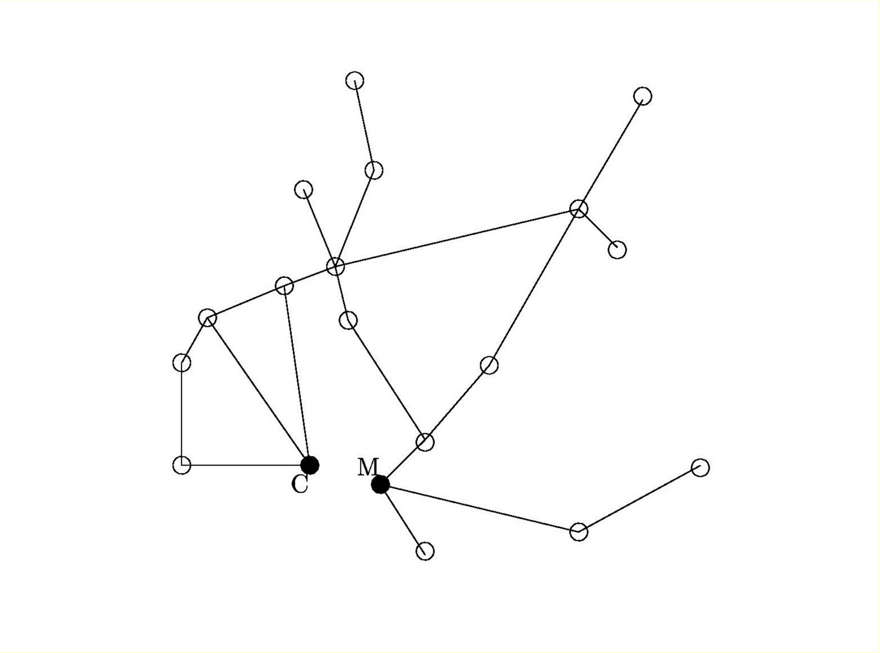
<u>Output</u>: Compute the stretch factor of N.



Approximate Stretch Factors

Input: A geometric graph N on a set S of n sites

Output: Compute (approx) stretch factor of N.



Reduction to O(n) shortest path queries.
[Narasimhan, Smid '01]

ε-APPROXIMATION ALGORITHM

Step 1: Using separation constant **$s = 4(2+\varepsilon)/\varepsilon$**

Compute a WSPD: $(A_1, B_1), \ldots, (A_m, B_m)$

Step 2: For every well-separated pair (A_i, B_i) pick an **arbitrary** pair of vertices (a_i, b_i) such that

$a_i \in A_i$, $b_i \in B_i$.

Step 3: Return

$$\max_i \{d_N(a_i,b_i)/|a_i b_i|\}$$

[Narasimhan & Smid '00]

[Trivial Exact Algorithm using APSP]

Approximate Stretch Factors

- PATH NETWORKS
  - O(nlogn)
- CYCLE NETWORKS
  - O(nlogn)
- TREE NETWORK
  - O(nlogn)
- PLANAR NETWORKS
  - O(nlogn)
- ARBITRARY NETWORKS
  - O(m + nlogn) [(1+ε)-approx]

GEOMETRIC ANALYSIS

Input: Set S of n sites; Set E of edges joining sites;
Property P Satisfied by E

Output: wt(E) ≤ ??

- Theta Graph Property [Clarkson, Keil]
- Diamond Property [Das]
- Gap Property [Das, Narasimhan]
- Leapfrog Property [Das, Narasimhan]
- Isolation Property [Das, Narasimhan]

Spanner Networks with other Properties

- Fault-Tolerance [Narasimhan, Smid]
- Small Degree
  [Soares, Salowe, Das, Heffernan, Arya et al.]
- Small Diameter [Arya et al.]
- Bottleneck Spanners [Narasimhan, Smid]
- Steiner Spanners – "Banyans" [Rao, Smith]
- Tree Spanners & Planar Spanners [Arikati et al.]
- Probabilistic Embeddings [Bartal]

Experiments with Spanners

- WSPD-based spanners followed by (approximate) greedy algorithm performs well.
    [Narasimhan & Zachariasen '00]

Problem

Preprocess a geometric spanner network so that approximate shortest path lengths between two query vertices can be reported efficiently (using subquadratic space).

Applications

- Shortest path queries in polygonal domains with obstacles.
- Approximate closest pair.
- Computing approximate stretch factors of geometric graphs.