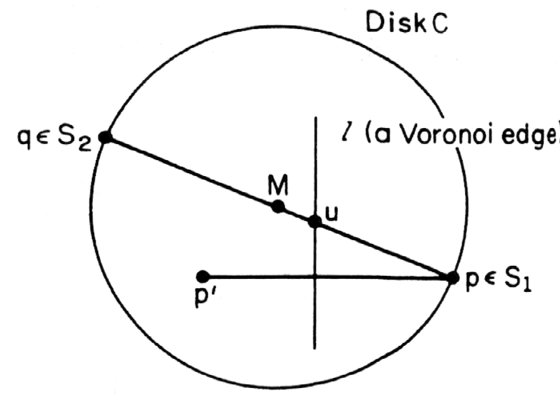
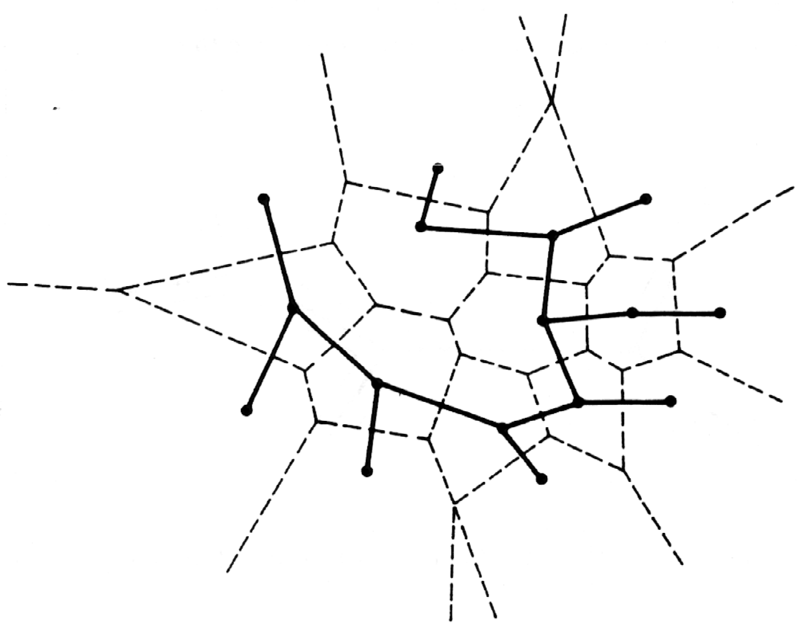
Euclidean Minimum Spanning Trees

- We need to examine only the edges of the Delaunay triangulation.
- We must compute the minimum spanning tree of a planar graph.
- MST in planar graphs can be computed in O(n) time [Cheriton & Tarjan '76]





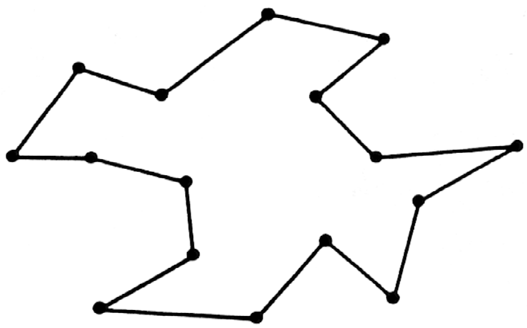
A set of points, its Voronoi diagram, and its EMST.

Theorem 6.1. *An EMST of a set S of N points in the plane can be computed from the Delaunay Triangulation of S in optimal time* $\theta(N)$.

Combining this result with the fact that the Delaunay triangulation is computable in time $\theta(N \log N)$ we have

Corollary 6.1. *An EMST of a set S of N points in the plane can be computed in optimal time* $\theta(N \log N)$.

PROBLEM P.9 (EUCLIDEAN TRAVELING SALESMAN). Find a shortest closed path through N given points in the plane.

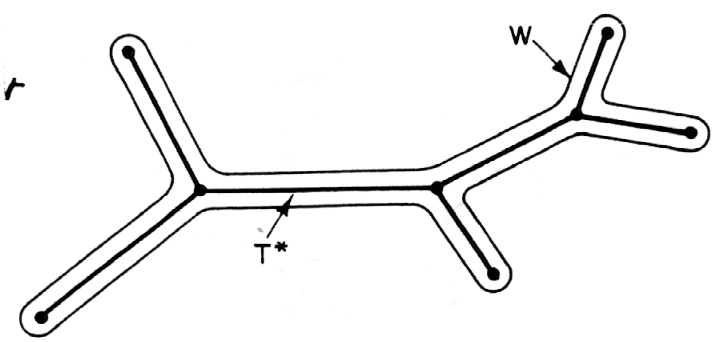


- NP-hard
  - in graphs
  - in L_2
  - in L_1
  - in L_m
  - in L_∞

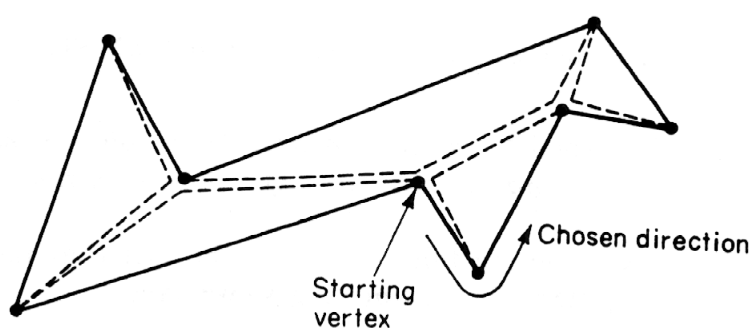
A traveling salesman tour.

Theorem 6.2 [Rosenkrantz–Stearns–Lewis (1974)]. *A minimum spanning tree can be used to obtain an approximate* TSP *tour whose length is less than twice the length of a shortest tour.*

- let T^* be optimal EMST
  Φ be optimal ETS tour
  W be an Euler tour
- $\ell(W) = 2\ell(T^*)$
- $\ell(T^*) \leq \ell(\Phi_{path})$,
  where Φ_{path} is $\Phi \setminus$ edge
- $\ell(W) \leq 2\ell(\Phi_{path}) <$
  $< 2\ell(\Phi)$.



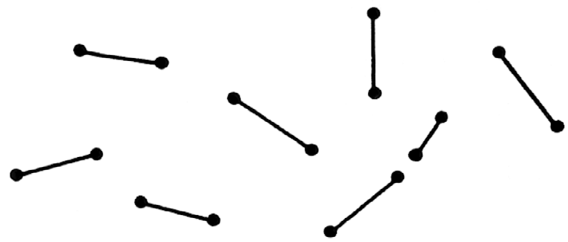
Doubling the EMST results in an Euler tour of all vertices of the se



"Short-cuts" on the Euler tour ensure that each vertex is visited exact once.

PROBLEM P.10 (MINIMUM EUCLIDEAN MATCHING). Given $2N$ points in the plane, join them in pairs by line segments whose total length is a minimum.

- minimum weighted matching in an arbitrary graph can be found in $O(n^3)$ time [Edmonds'65, Gabow'72]



A minimum Euclidean matching.

- Can you get a faster algo?

$\boxed{\text{2-appr. in } O(n \log n) \quad \tfrac{3}{2}\text{-appr. in } O(n^3) \quad \text{Can you do better?}}$ OPEN

Theorem 6.3. *An approximation to the traveling salesman problem whose length is within 3/2 of optimal can be obtained in $O(N^3)$ time if the interpoint distances obey the triangle inequality.* [Christofides'76]

PROOF. The following algorithm achieves the desired result on the given set S:

1. Find a minimum spanning tree T^* of S.
2. Find a minimum Euclidean matching M^* on the set $X \subseteq S$ of vertices of *odd* degree in T^*. (X has always even cardinality in any graph.)
3. The graph $T^* \cup M^*$ is an Eulerian graph, since all of its vertices have even degree. Let Φ_e be an Eulerian circuit of it.
4. Traverse Φ_e edge by edge and bypass each previously visited vertex. Φ_{appr} is the resulting tour.

- $\ell(T^*) < \ell(\Phi)$ (like before)
- $\ell(M^*) \leq \frac{1}{2}\ell(\Phi)$ (take every second edge in Φ)
- $\ell(\Phi_{appr.}) \leq \ell(\Phi_e)$ (from Δ inequality)
- Hence

$$\begin{aligned}
\text{length}(\Phi_{\text{appr}}) &\leq \text{length}(\Phi_e) \\
&= \text{length}(T^*) + \text{length}(M^*) \\
&< \text{length}(\Phi) + \tfrac{1}{2}\,\text{length}(\Phi) \\
&= \tfrac{3}{2}\,\text{length}(\Phi).
\end{aligned}$$

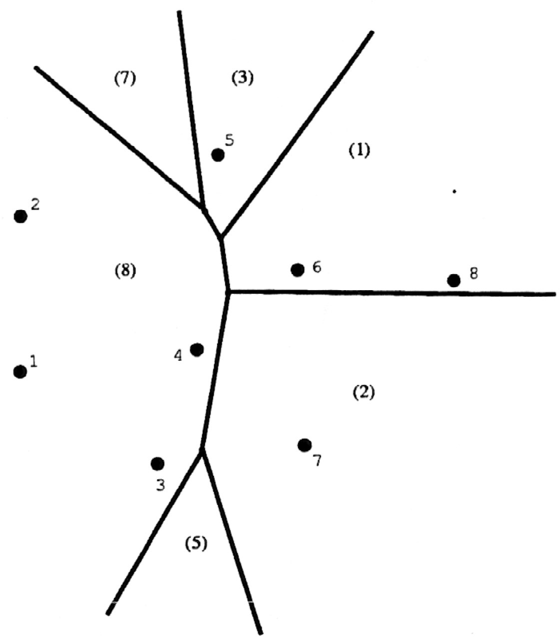
Smallest Circle

- **Given:** n points in the plane
- **Find:** Smallest circle containing all points. $\min_{p_0} \max_i \left((x_i - x_0)^2 + (y_i - y_0\right.$
- There is a unique solution. It goes through three points or has two diameter defining points.
- Trivial algorithm $O(n^4)$.

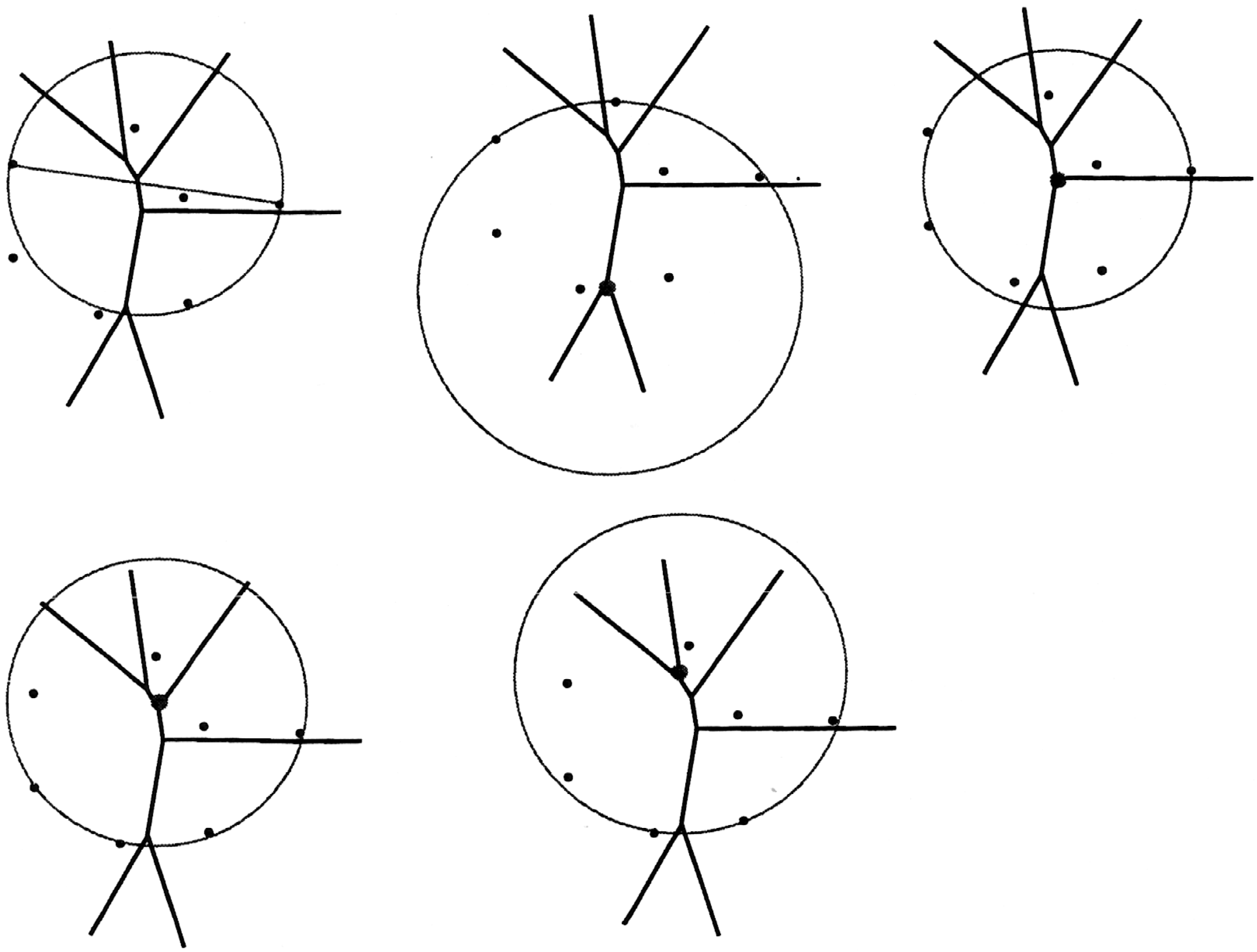
- Determine the diameter of the point set. STOP if the circle with this diameter contains all points. Requires $O(n \log n)$.
- Determine $VD_{n-1}(S)$. Find circle with Voronoi point as origo and through three most remote points. Requires $O(n \log n)$.

- $O(n^4)$ [Rademacher, Toeplitz '57]
- $O(n^2)$ [Elzinga, Hearn '72]
- $O(n \log n)$ [Shamos '78]
- $\Theta(n)$ [Megiddo '83]

Voronoi Diagrams of (n-1)-st Order



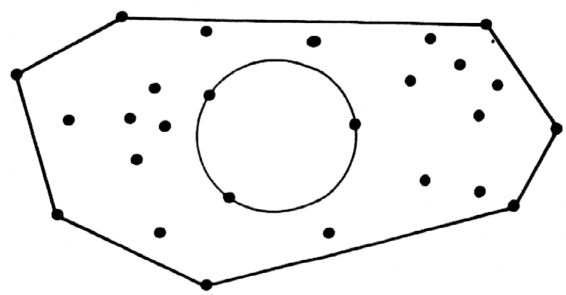
Smallest Circle



Can be solved in $\Theta(n)$ time using Megiddo's prune and search technique for linear programming with 3 variables (PS 297-299).

Largest Empty Circle

- **Given:** n points in the plane
- **Find:** Largest empty circle with center in the convex hull of the point-set.

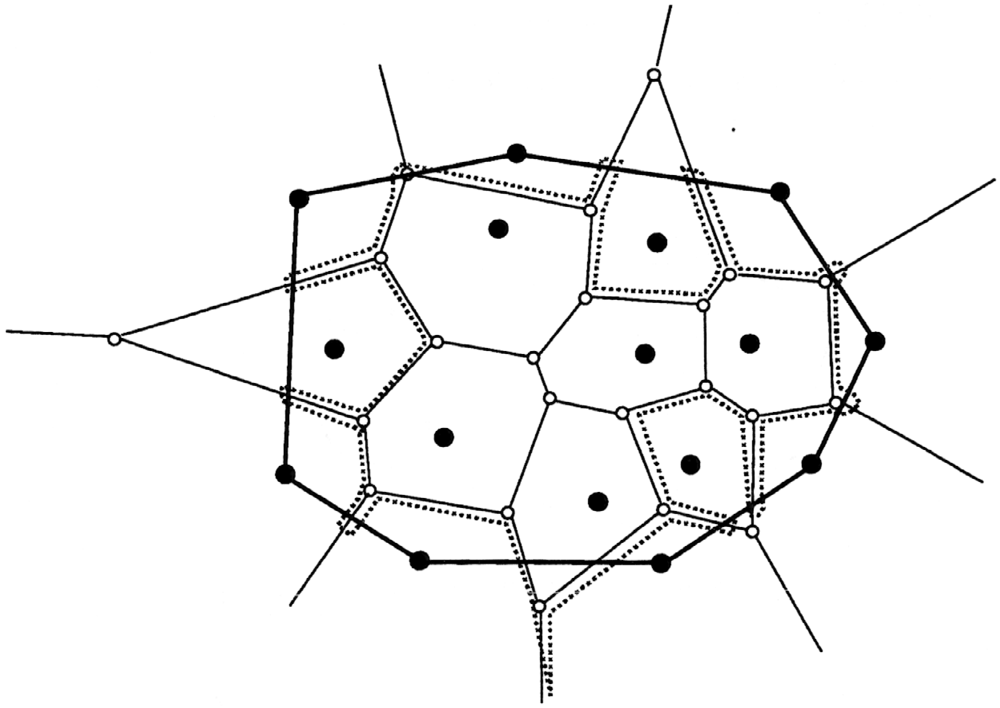


A largest empty circle whose center is internal to the hull.

$$\max_{p_0 \in CH(S)} \min_{i} (x_i - x_0)^2 + (y_i - y_0)^2$$

- Francis, White (1974) in $O(n^3)$ time
- Shamos (1978) in $O(n \log n)$ time

Largest Empty Circle



- The center is either a Voronoi point or occurs at an intersection of a Voronoi edge with the boundary of the convex hull.
- How to find the intersection?
- Voronoi edges intersect at most 2 boundary edges of the convex hull.
- Each side of the convex hull intersects at least one Voronoi edge.

- $f(x,y)$ is the distance of point $p=(x,y)$ from the nearest point in S.
- $f(x,y)$ is a downward-convex function of both x and y within a Voronoi polygon
- Hence, $f(x,y)$ gets its maximum at a vertex of one such polygon

Corollary 6.2. *In the algebraic computation tree model, any algorithm for the* MAXIMUM GAP *problem on a set of N real numbers requires* $\Omega(N \log N)$ *time.*

In a modified computation model, however, Gonzalez (1975) has obtained the most surprising result that the problem can be actually solved in linear time. The modification consists of adding the (nonanalytic) *floor function* "$\lfloor\ \rfloor$" to the usual repertoire. Here is Gonzalez's remarkable algorithm:

```
procedure MAX GAP
  Input:  N real numbers X[1:N] (unsorted)
  Output: MAXGAP, the length of the largest gap between consecutive
          numbers in sorted order.
begin MIN := min X[i];
  MAX := max X[i];
  (*create N − 1 buckets by dividing the interval from MIN to MAX
  with N − 2 equally-spaced points. In each bucket we will retain
  HIGH[i] and LOW[i], the largest and smallest values in bucket i*)
  for i := 1 until N − 1 do
    begin COUNT[i] := 0;
          LOW[i] := HIGH[i] := Λ
    end; (*the buckets are set up*)
  (*hash into buckets*)
  for i := 1 until N − 1 do
    begin BUCKET := 1 + ⌊(N − 1) × (X[i] − MIN)/
          (MAX − MIN)⌋;
          COUNT[BUCKET] := COUNT[BUCKET] + 1;
          LOW[BUCKET] := min (X[i], LOW[BUCKET]);¹¹
          HIGH[BUCKET] := max (X[i], HIGH[BUCKET])¹¹
    end;
  (*Note that N − 2 points have been placed in N − 1 buckets, so by
  the pigeonhole principle some bucket must be empty. This means that
  the largest gap cannot occur between two points in the same bucket.
  Now we make a single pass through the buckets*)
  MAXGAP := 0;
  LEFT := HIGH[1];
  for i := 2 until N − 1 do
    if (COUNT[i] ≠ 0) then
      begin THISGAP := LOW[i]-LEFT;
            MAXGAP := max(THISGAP, MAXGAP);
            LEFT := HIGH[i]
      end
end.
```

This algorithm sheds some light on the computational power of the "floor"

[11] Here, by convention, $\min(x, \Lambda) = \max(x, \Lambda) = x$.