Algorithms. Lecture Notes 13

Problem: Closest Points

Given: a set of n points in the plane, specified by their Cartesian coordinates (x_i, y_i).

Goal: Find a pair of points with minimum distance.

Motivations:
Some approaches to hierarchical clustering of data take the two closest data points and combine them to a cluster by replacing these two points by their midpoint, and this step is repeated until one cluster remains.

Divide-and-Conquer in Geometry: Closest Points

Fast geometric calculations are needed in computer graphics, computer-aided design, robotics, planning (transport optimization, facility location), chemistry (modelling molecules and their dynamics), for extracting information from geographic databases, etc. The amount of data can be huge (e.g., elements of a picture), such that efficient algorithms make a difference.

Divide-and-conquer is suitable for various geometric problems, because instances can be divided in a natural way. (However, the conquer phase is usually less trivial). To give at least an impression, we discuss another geometric problem example: finding a pair of closest points among n given points in the plane.

An obvious algorithm would compute all pairwise distances and determine the minimum in $O(n^2)$ time. Instead, we aim at a divide-and-conquer algorithm satisfying the recurrence $T(n) = 2T(n/2)+O(n)$, hence with time complexity $T(n) = O(n \log n)$.

It is natural to divide the set by a straight line. To make the calculation details simple, we first sort the points by their x-coordinates, and then halve the set by a vertical separator line. More formally, we take the median z of all x-values and put all points with coordinate $x < z$ and $x > z$, respectively, in the two sets. Recall that sorting takes $O(n \log n)$ time, which does not destroy the desired time bound. Wouldn't it be enough to compute the median in $O(n)$ time, without sorting? Yes, it is enough for the first step, but we will recursively split the point set further, on the lower recursion levels. Sorting the points once in the beginning is simpler and cheaper (in terms of the hidden constant factors) than median computations on every recursion level.

Then, of course, we compute the closest pairs in both subsets recursively. Let d be the minimum of the two minimum distances. The tricky part is to combine the partial solutions. The global solution could be the best of the two closest pairs from the two subsets, but there could also exist a pair of points with distance smaller than d, having one point in each subset. The candidates for such pairs of points are in a stripe of breadth d on both sides of the separating line. Moreover, each point has only constantly many partners (at distance smaller than d) on the other side, hence $O(n)$ such pairs of close points must be considered. These pairs can be identified in $O(n)$ time, if all points are already sorted by their y-coordinates as well. With careful implementation, all steps in the conquer phase run in $O(n)$ time as desired.

Problem: Clustering with Maximum Spacing

A **clustering** of a set of (data) points is simply a partitioning into disjoint subsets of points, called **clusters**. Some distance function is defined between the points. The distance of two point sets A and B is the minimum distance of two points $a \in A$ and $b \in B$. The **spacing** of a clustering is the minimum distance of two clusters (or equivalently, the minimum distance of any two points from different clusters).

Given: a set of n points in some geometric space, and an integer $k < n$. The pairwise distances of points are known, or they can be easily computed from their coordinates.

Goal: Construct a clustering with k clusters and maximum spacing.

Motivations:

Clustering in general has many applications in data reduction, pattern recognition, classification, data mining, and related fields. Coordinates of points are often numerical features of objects. Every cluster shall consist of "similar" objects, whereas objects in different clusters shall be "dissimilar". However, we have to make these intuitive notions precise. There exist myriads of meaningful quality measures for clusterings, and each one gives rise to an algorithmic problem: to find a clustering that optimizes this quality measure.

Many clustering problems can be formulated as graph problems, where the data objects are nodes. For instance, Graph Coloring can be seen as a clustering problem: The desired number k of clusters is given, and every cluster must fulfill some "internal" criterion, namely, not to contain any pair of dissimilar nodes. Spacing is an "external" quality measure. It demands that any two clusters be far away from each other, while nothing is explicitly said about the inner structure of clusters.

Clustering with Maximum Spacing via MST

Kruskal's MST algorithm has a nice application and interpretation in the field of clustering problems. Suppose that the nodes of our graph are data points, and the edge costs are the distances. (The graph is complete, that is, all possible edges exist.) A clustering with maximum spacing (i.e., maximized minimum distance between any two clusters) can be found as follows: Do $n - k$ steps of Kruskal's algorithm and take the node sets of the so obtained k trees $T_1, \dots, T_k$ as clusters.

We prove that the obtained spacing d is in fact optimal: Consider any partitioning into k clusters $U_1, \dots, U_k$. There must exist two nodes p, q in some T_r that belong to different clusters there, say $p \in U_s$, $q \in U_t$. Due to the rule of Kruskal's algorithm, all edges on the path in T_r from p to q have cost at most d. But one of these edges joins two different "U clusters", hence the spacing of the other clustering can never exceed d.