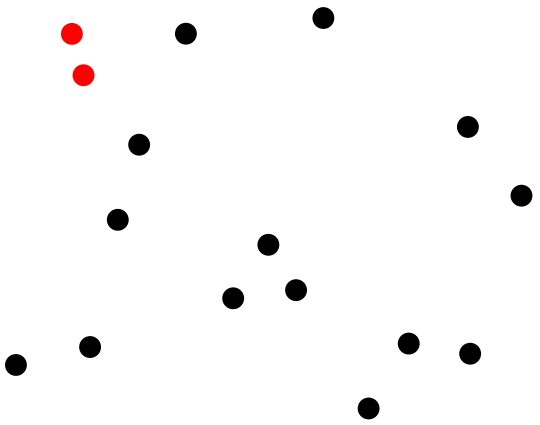
Closest pair of points problem



Closest pair of points shown in red

The **closest pair of points problem** or **closest pair problem** is a problem of computational geometry: given n points in metric space, find a pair of points with the smallest distance between them. The closest pair problem for points in the Euclidean plane[1] was among the first geometric problems that were treated at the origins of the systematic study of the computational complexity of geometric algorithms.

A naive algorithm of finding distances between all pairs of points in a space of dimension d and selecting the minimum requires $O(dn^2)$ time. It turns out that the problem may be solved in $O(n \log n)$ time in a Euclidean space or L^p space of fixed dimension d.[2] In the algebraic decision tree model of computation, the $O(n \log n)$ algorithm is optimal, by a reduction from the element uniqueness problem. In the computational model that assumes that the floor function is computable in constant time the problem can be solved in $O(n \log \log n)$ time.[3] If we allow randomization to be used together with the floor function, the problem can be solved in $O(n)$ time.[4][5]

1 Brute-force algorithm

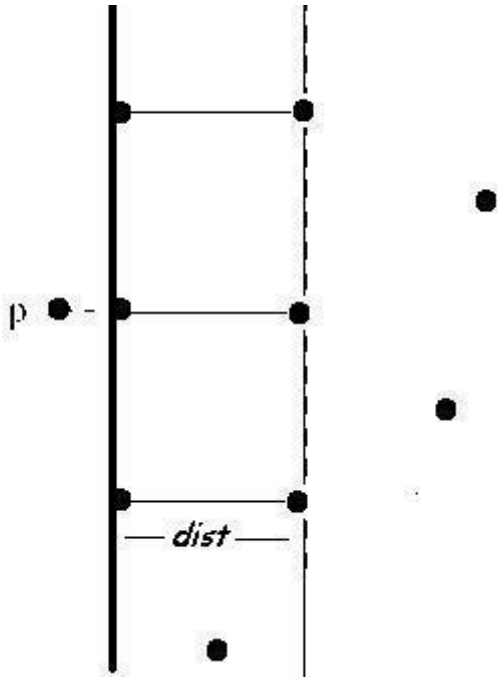
The closest pair of points can be computed in $O(n^2)$ time by performing a brute-force search. To do that, one could compute the distances between all the $n(n-1)/2$ pairs of points, then pick the pair with the smallest distance, as illustrated below.

minDist = infinity **for** $i = 1$ **to** length(P) - 1 **for** $j = i + 1$ **to** length(P) **let** $p = P[i]$, $q = P[j]$ **if** *dist*(p, q) < *minDist*: *minDist* = *dist*(p, q) *closestPair* = (p, q) **return** *closestPair*

2 Planar case

The problem can be solved in $O(n \log n)$ time using the recursive divide and conquer approach, e.g., as follows:[1]

1. Sort points according to their x-coordinates.
2. Split the set of points into two equal-sized subsets by a vertical line $x=x_{\text{mid}}$.
3. Solve the problem recursively in the left and right subsets. This yields the left-side and right-side minimum distances $d\text{L}_{\text{min}}$ and $d\text{R}_{\text{min}}$, respectively.
4. Find the minimal distance $d\text{LR}_{\text{min}}$ among the set of pairs of points in which one point lies on the left of the dividing vertical and the other point lies to the right.
5. The final answer is the minimum among $d\text{L}_{\text{min}}$, $d\text{R}_{\text{min}}$, and $d\text{LR}_{\text{min}}$.



Divide-and-conquer: sparse box observation

It turns out that step 4 may be accomplished in linear time. Again, a naive approach would require the calculation of distances for all left-right pairs, i.e., in quadratic time. The key observation is based on the following sparsity property of the point set. We already know that the closest pair of points is no further apart than dist= min(dL$_{min}$, dR$_{min}$). Therefore, for each point p to the left of the dividing line we have to compare the distances to the points that lie in the rectangle of dimensions (dist, 2 · dist) to the right of the dividing line, as shown in the figure. And what is more, this rectangle can contain at most six points with pairwise distances at least dR$_{min}$. Therefore, it is sufficient to compute at most 6n left-right distances in step 4.[6] The recurrence relation for the number of steps can be written as $T(n) = 2\ T(n/2) + O(n)$, which we can solve using the master theorem to get $O(n \log n)$.

As the closest pair of points define an edge in the Delaunay triangulation, and correspond to two adjacent cells in the Voronoi diagram, the closest pair of points can be determined in linear time when we are given one of these two structures. Computing either the Delaunay triangulation or the Voronoi diagram takes $O(n \log n)$ time. These approaches are not efficient for dimension d>2, while the divide-and-conquer algorithm can be generalized to take $O(n \log n)$ time for any constant value of d.

3 Dynamic closest-pair problem

The dynamic version for the closest-pair problem is stated as follows:

- Given a dynamic set of objects, find algorithms and data structures for efficient recalculation of the closest pair of objects each time the objects are inserted or deleted.

If the bounding box for all points is known in advance and the constant-time floor function is available, then the expected O(n) space data structure was suggested that supports expected-time O(log n) insertions and deletions and constant query time. When modified for the algebraic decision tree model, insertions and deletions would require O(log^2 n) expected time.[7] It is worth noting, though, that the complexity of the dynamic closest pair algorithm cited above is exponential in the dimension d, and therefore such an algorithm becomes less suitable for high-dimensional problems.

4 See also

- GIS
- Nearest neighbor search
- Set cover problem

5 Notes

[1] M. I. Shamos and D. Hoey. "Closest-point problems." In *Proc. 16th Annual IEEE Symposium on Foundations of Computer Science* (FOCS), pp. 151—162, 1975 (DOI 10.1109/SFCS.1975.8)

[2] K. L. Clarkson, "Fast algorithms for the all nearest neighbors problem", FOCS 1983.

[3] S. Fortune and J.E. Hopcroft. "A note on Rabin's nearest-neighbor algorithm." Information Processing Letters, 8(1), pp. 20—23, 1979

[4] S. Khuller and Y. Matias. A simple randomized sieve algorithm for the closest-pair problem. Inf. Comput., 118(1):34—37,1995

[5] Richard Lipton (24 September 2011). "Rabin Flips a Coin".

[6] Cormen, Leiserson, Rivest, and Stein, 2001.

[7] Mordecai Golin, Rajeev Raman, Christian Schwarz, Michiel Smid, "Randomized Data Structures For The Dynamic Closest-Pair Problem", SIAM J. Comput., vo. 26, no. 4, 1998, preliminary version reported at the 4th Annu. ACM-SIAM Symp. on Discrete Algorithms, pp. 301–310 (1993)

6 References

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Pages 957–961 of section 33.4: Finding the closest pair of points.
- Jon Kleinberg; Éva Tardos (2006). *Algorithm Design*. Addison Wesley.
- UCSB Lecture Notes
- rosettacode.org - Closest pair of points implemented in multiple programming languages
- Line sweep algorithm for the closest pair problem

7 Text and image sources, contributors, and licenses

7.1 Text

- **Closest pair of points problem** *Source:* https://en.wikipedia.org/wiki/Closest_pair_of_points_problem?oldid=765885598 *Contributors:* XJaM, Geary, Altenmann, Tea2min, Giftlite, Sam Hocevar, Spoon!, Rrenaud, LOL, GregorB, BD2412, Qwertyus, Vegaswikian, CiaPan, Johndburger, Fulldecent, Gfonsecabr, Salgueiro~enwiki, HenryHRich, Magioladitis, David Eppstein, Thomasda, Davecrosby uk, VolkovBot, Loren.wilton, AgentNN, Addbot, Tide rolls, Luckas-bot, AnomieBOT, Erel Segal, Wachmc, Omnipaedista, Adrignola, Primefac, SaschaWolff, GmeSalazar, AmrinderAroraSW, Miracle173, Mikebolt, Loraof, The Quixotic Potato, Fmadd and Anonymous: 25

7.2 Images

- **File:Closest_pair.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/65/Closest_pair.jpg *License:* Public domain *Contributors:* No machine-readable source provided. Own work assumed (based on copyright claims). *Original artist:* No machine-readable author provided. Mikkalai assumed (based on copyright claims).
- **File:Closest_pair_of_points.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/3/37/Closest_pair_of_points.svg *License:* CC0 *Contributors:* Own work *Original artist:* Qef

7.3 Content license

- Creative Commons Attribution-Share Alike 3.0