Accurate and efficient algorithms for proximity problems

Peter Schorn, Institut für Theoretische Informatik, ETH Zürich, CH-8092 Zürich, Switzerland

Extended Abstract

1. Introduction

Geometric computation with imprecise primitives has become an important research topic, since floating point arithmetic is the predominant arithmetic supported by hardware in todays computers. Implementations which naively assume that floating point arithmetic is the same as real arithmetic will in the best case lead to a program with unknown error bounds or in the worst case to unexpected failure. Although there has been progress in this area [GY 86, H 89, M 88, M 89, SSG 89] there are still not many geometric algorithms known which are practical, efficient and above all accurate if implemented on the basis of ubiquitous floating point arithmetic.

This paper presents and analyzes an O(n log n) algorithm for the closest pair problem. Implemented in a floating point arithmetic fulfilling some set of reasonable axioms, it computes the distance of the closest pair with the same relative error bound with which we can compute the distance function, e.g. any L_p distance function. Later we modify this algorithm to get an equally accurate method for the all nearest neighbors problem which is no longer optimal in the theoretical sense but still faster in practice than optimal algorithms for point sets with a size of up to $4*10^3$ points.

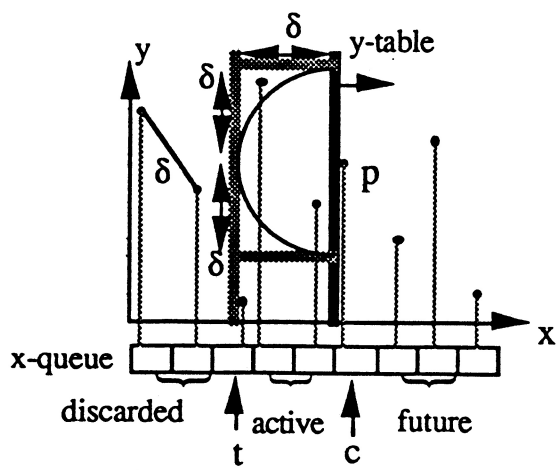
In the following we briefly review a closest pair algorithm and elaborate on its numerical analysis. Then we present an algorithm for the all nearest neighbors problem for which essentially the same numerical analysis holds before we finish stating some open problems.

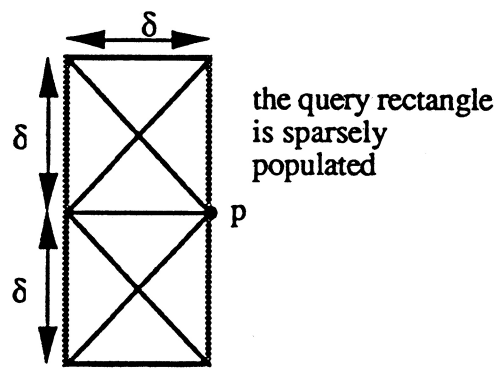
2. The closest pair problem

We solve the following problem: given a set S of n points in the plane find two points with smallest distance. Since we are going to use floating point arithmetic we cannot hope for identifying a correct closest pair, but we will strive for computing the distance of a correct closest pair with the same error as if we had taken the minimum of all n*(n-1) / 2 pairwise distances.

2.1 Review: a plane sweep approach to the closest pair problem

Basically we use the algorithm presented in [HNS 88] which we review only briefly since the main point is proving accuracy. The configuration of points is swept from left to right stopping at each data point. As an invariant we keep the distance δ of the closest pair of points to the left of the sweep line. In a structure called y-table we keep points sorted by their y-coordinates which are possible candidates for forming a closest pair with a new data point. Processing a data point p consists of removing all points from the y-structure which differ in their x-coordinate by more then δ from p, inserting the new data point p into the y-table and checking all its neighbors in the y-table that differ by at most δ in their y-coordinate from p.





Since the number of points to be examined for each data point is limited by eight we obtain an algorithm which is optimal in the algebraic decision tree model of computation.

The proof of its accuracy proceeds in two steps: first we show that an implementation using floating point arithmetic is consistent with the invariants and second we bound the error using a simple lemma from numerical analysis.

2.2 Consistent interpretation

Careful analysis of the underlying geometric primitives and invariants produces the following algorithm scheme CP:

```
S := sortByX(S); c:=3; t:=1; δ := d(S[1], S[2]); Y := {S[1], S[2]};
while c ≤ n do
   (* Invariant: δ = δ(c) ∧ Y = Y(c) ∧ t = t(c) where        *)
   (* δ(c) = min( d(S[i], S[j]): 1 ≤ i < j < c )             *)
   (* Y(c) = { S[i]: 1 ≤ i < c ∧ dx(S[c-1], S[i]) < δ(c-1) } *)
   (* t(c) = min( i: S[i] ∈ Y(c) )                           *)
   transition; c := c + 1;
end;
   (* post condition: δ = min( d(S[i], S[j]): 1 ≤ i < j ≤ n ) *)

transition ≡
   p := S[c];
   while dx(p, S[t]) ≥ δ do      (* normally: p_x - S[t]_x ≥ δ *)
      deleteY(S[t]); t := t + 1; (* remove points left of the δ-slice *)
   end;
   insertY(p);
   q := p;
   repeat                        (* examine upper neighbors *)
      q := succY(q); if d(p, q) < δ then δ := d(p, q) end;
   until dy_r(p, q) > δ;         (* normally: q_y - p_y > δ *)
   q := p;
   repeat                        (* examine lower neighbors *)
      q := predY(q); if d(p, q) < δ then δ := d(p, q) end;
   until dy_l(p, q) > δ;         (* normally: p_y - q_y > δ *)
```

The invariant responsible for the post condition is preserved if the following axioms hold for the primitives d, dx, dy_r and dy_l assuming the relational operators are exact:

$d: \mathbf{R}^2 \times \mathbf{R}^2 \to \mathbf{R}$	(normally: $(\lvert p_x - q_x\rvert^k + \lvert p_y - q_y\rvert^k)^{1/k}$
$d(p, q) = d(q, p)$	(symmetry)
$dx: \mathbf{R}^2 \times \mathbf{R}^2 \to \mathbf{R}$	(normally: $p_x - q_x$)
$p_x \geq p'_x \Rightarrow dx(p, q) \geq dx(p', q)$	(monotonicity in the first argument)
$q_x \leq q'_x \Rightarrow dx(p, q) \geq dx(p, q')$	(inverse monotonicity in the second argument)
$p_x \geq q_x \Rightarrow d(p, q) \geq dx(p, q)$	(lower bound with respect to the distance)
$dy_l: \mathbf{R}^2 \times \mathbf{R}^2 \to \mathbf{R}$	(normally: $p_y - q_y$)
$q_y \leq q'_y \Rightarrow dy_l(p, q) \geq dy_l(p, q')$	(inverse monotonicity in the second argument)
$p_y \geq q_y \Rightarrow d(p, q) \geq dy_l(p, q)$	(lower bound with respect to the distance)
$dy_r: \mathbf{R}^2 \times \mathbf{R}^2 \to \mathbf{R}$	(normally: $q_y - p_y$)
$q_y \geq q'_y \Rightarrow dy_r(p, q) \geq dy_r(p, q')$	(monotonicity in the second argument)
$p_y \leq q_y \Rightarrow d(p, q) \geq dy_l(p, q)$	(lower bound with respect to the distance)

Note that d requires only symmetry to achieve a correct result, although the sparsity inducing property

of d makes the algorithm efficient. We call any interpretation of these primitives *consistent* if the aforementioned axioms hold. In the following we denote floating point subtraction by $-^*$ and the floating point implementation of the distance function d by d^*.

Lemma 1: The interpretation of CP with $d(p, q) = d^*(p, q)$, $dx(p, q) = p_x -^* q_x$, $dy_l(p, q) = p_y -^* q_y$ and $dy_r(p, q) = q_y -^* p_y$ is consistent.

Proof: The monotonicity laws for floating point subtraction hold for any reasonable implementation, since we only require that $a -^* b$ does not increase if b is increased. The lower bound property of dx, dy_r and dy_l can be obtained by using $d^{**}(p, q)=\max(dx(p, q), dy_r(p, q), dy_l(p, q), d^*(p, q))$ instead of $d^*(p, q)$.

Trivial interpretations, e.g. setting all functions identical to zero, show that consistency is not enough to guarantee an accurate result. We must analyze the resulting post condition numerically.

2.3 Numerical analysis

Lemma 2: The minimum of n imprecise quantities with a common relative error bound eps can be viewed as an ε-perturbation of the minimum of the precise quantities:

$$\min\left(a_i^*: 1 \le i \le n\right) = \min(a_i: 1 \le i \le n)(1+\varepsilon) \text{ where } a_i^* = a_i(1+\varepsilon_i) \text{ with } \varepsilon_i < \text{eps}, \varepsilon < \text{eps}.$$

Proof: Follows easily by induction over n.

Lemma 3: CP given the interpretation of lemma 1 computes the distance of the closest pair with the same relative error bound as the distance function d^*.

Proof: Lemma 1 shows that the floating point interpretation is consistent which in turn gives us the validity of the post condition. Lemma 2 proves that the minimum over the computed distances can be viewed as a minimum over the correct distances with a perturbation of the same order of magnitude with which we compute distances.

Note that we can easily implement $d^*(p, q)$ such that $d^*(p, q) = d(p, q)\ (1 + k\ \varepsilon)$, $\varepsilon < \text{eps}$, $k \le 8$ where eps is the machine precision ($\text{eps} = 1/2\ b^{1-p}$, b = base of floating point arithmetic, p = number of digits in the mantissa). Note also that implementing the line '**while** $p_x - s[t]_x \ge \delta$ **do**' as '**while** $p_x - \delta \ge s[t]_x$ **do**' leads to a highly increased error bound if the distances and coordinates differ greatly in magnitude.

3. Cooperative sweep for the all nearest neighbors problem

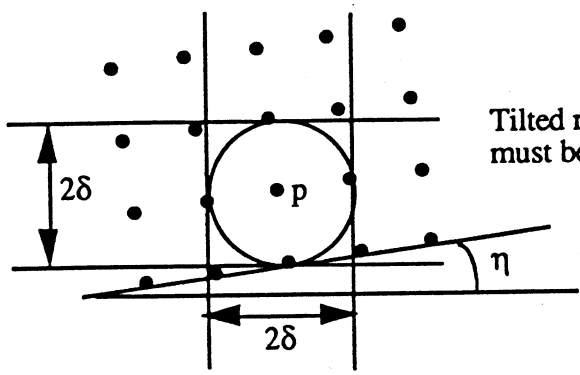
The all nearest neighbors problem asks for a nearest neighbor for each given point. Experiments have shown that the following very simple algorithm is surprisingly efficient:

```
Input:  a set S of n points in the plane
Output: for each point the distance to a nearest neighbor

X := sortByX(S); Y := sortByY(S); (* X, Y: array [1..n] of point *)
FOR EACH p IN S DO
   left := find(p, X);  down := find(p, Y); (* by binary search *)
   right := left;       up := down; δ := ∞;
   REPEAT
      left := left-1; right := right+1; down := down-1; up := up+1;
   UNTIL  (halfSliceDone(X, left) ∧ halfSliceDone(X, right)) ∨
          (halfSliceDone(Y, down) ∧ halfSliceDone(Y, up));
   δ(p) := δ;
END;
```

```
FUNCTION halfSliceDone(Z, dir): boolean;
   halfSliceDone := (dir ≤ 0) ∨ (dir > n) ∨ (|Z[dir]_Z - p_Z| ≥ δ);
   IF ¬halfSliceDone ∧ (d(Z[dir],p) < δ) THEN δ := d(Z[dir],p); END;
END;
```

Its main idea is to sweep in two orthogonal directions at the same time, stopping either sweep as soon as the answer is known, thus avoiding the disastrous case where all points are on a vertical line. We have found the nearest neighbor for a point p when we have examined all the points in either the horizontal or vertical 2δ-slice centered around p where δ denotes p's distance to its nearest neighbor. This method can also be employed to solve the post-office problem or the online version of the all nearest neighbors problem. Furthermore it enjoys the same error analysis as the closest pair algorithm and is therefore the most accurate algorithm known based on floating point arithmetic. It is also faster than an exact implementation of an optimal plane sweep algorithm given in [HNS 90] for configurations consisting of up to a few thousand points. Probabilistic analysis and the following figure have led us to the conjecture that its running time is $\Theta(n^{1.5})$ where the constant is favorably low.



Tilted regular grid: for each point $\Theta(\sqrt{n})$ points must be examined.

4. Conclusion and open problems

We have presented an optimal algorithm for the closest pair problem using floating point arithmetic which achieves the same relative error bound for computing the distance of the closest pair as the exhaustive method that takes the minimum over all n*(n-1)/2 distances. The same analysis can be extended to a very simple, practical but not optimal algorithm. It would be interesting to determine the actual complexity of this useful algorithm, i.e. the worst case configuration. In the case of the closest pair algorithm we ask whether there are any other interesting interpretations besides the L_p metrics that have a O(n log n) running time.

References

[GY 86] D. Greene, F. Yao: Finite Resolution Computational Geometry, Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 143 - 152.

[H 89] C. Hoffmann: The Problems of Accuracy and Robustness in Geometric Computation, Computer, Vol. 22, No. 3, March 1989, pp. 31-42.

[HNS 88] K. Hinrichs, J. Nievergelt, P. Schorn: Plane-Sweep Solves the Closest Pair Problem Elegantly, Information Processing Letters 26 (11 Jan. 1988), pp. 255 - 261.

[HNS 90] K. Hinrichs, J. Nievergelt, P. Schorn: An all-round sweep algorithm for 2-dimensional nearest-neighbor problems, submitted.

[M 88] V. Milenkovic: Verifiable Implementations of Geometric Algorithms using Finite Precision Arithmetic, CMU report CMU-CS-88-168, Carnegie Mellon, 1988.

[M 89] V. Milenkovic: Double Precision Geometry: a General Technique for Calculating Line and Segment Intersections using Rounded Arithmetic, 30th Annual Symp. on Found. of Computer Science, 1989.

[SSG 89] D. Salesin, J. Stolfi, L. Guibas: Epsilon Geometry: Building Robust Algorithms from Imprecise Calculations, Proc. of the Fifth Annual Symposium on Computational Geometry, 1989, pp. 208 - 217.