Solving geometric optimization problems: a randomized approach

Michiel Smid*

November 14, 2003

1 Introduction

These notes are based on the article *Geometric applications of a randomized optimization technique* by Timothy Chan, which appeared in *Discrete & Computational Geometry*, Volume 22, 1999, pp. 547–567.

We will present a general approach for solving a large class of geometric optimization problems. We start by introducing the general framework.

Let Π be some problem space and let $F : \Pi \longrightarrow \mathbb{R}$ be a function that assigns to each element $S \in \Pi$ a real number $F(S)$ which we call the "solution" of S. (For example, Π could be the set of all finite point sets in $\mathbb{R}^D$, and $F(S)$ could be the minimum distance between any two distinct points of S.) Our goal is to design an efficient algorithm that computes $F(S)$ for any given set $S \in \Pi$. Such an algorithm exists if the following five conditions hold.

Condition 1: There is a constant c such that for any $S \in \Pi$ with $|S| \leq c$, the value of $F(S)$ can be computed in $O(1)$ time.

Condition 2: There is an algorithm A that, when given as input any set $S \in \Pi$ and any $t \in \mathbb{R} \cup \{\infty\}$, decides whether $F(S) < t$. To be more precise, algorithm $A(S, t)$ returns *true* if $F(S) < t$, and *false* if $F(S) \geq t$. We denote the worst-case running time of A by $T_A(n)$, where n is the size of S.

---

[*] School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6. E-mail: `michiel@scs.carleton.ca`.

```
Algorithm $B(S)$
(* $S \in \Pi$; this algorithm computes $F(S)$ *)
if $|S| \leq c$
then compute and return $F(S)$
else compute the sets $S_1, \ldots, S_k \in \Pi$ as in Condition 3;
    $min := \infty$;
    for $i := 1$ to $k$
    do if $A(S_i, min) = true$
        then $min := B(S_i)$
        endif
    endfor;
    return $min$
endif
```

Figure 1: *A slow algorithm for computing $F(S)$.*

Condition 3: There are constants $0 < \alpha < 1$ and $k \in \mathbb{N}$ such that for any $S \in \Pi$, we can compute k sets $S_1, \ldots, S_k \in \Pi$ such that

3.1: $|S_i| \leq \alpha |S|$, for all $1 \leq i \leq k$, and

3.2: $F(S) = \min(F(S_1), \ldots, F(S_k))$.

We assume that these sets $S_1, \ldots, S_k$ can be computed in $T_A(n)$ time, where $n = |S|$.

Condition 4: The function $T_A(n)/n$ is non-decreasing.

Condition 5: $k\alpha \geq 2$.

2 A slow algorithm

It should be clear that Condition 3 implies a recursive algorithm for computing $F(S)$. This algorithm, which we denote by $B(S)$, is given in Figure 1.

The correctness of this algorithm follows from the five conditions. Let us prove an upper bound on the running time of B. Let $T_B(n)$ denote the worst-case running time of B on any input of size n. There is a constant c'

such that $T_B(n) \le c'$ for all $n \le c$. If $n > c$, then

$$\begin{aligned} T_B(n) &\le T_A(n) + \sum_{i=1}^{k} \left(T_A(|S_i|) + T_B(|S_i|) \right) \\ &\le T_A(n) + k \cdot T_A(\alpha n) + k \cdot T_B(\alpha n). \end{aligned}$$

Since

$$k \cdot T_A(\alpha n) = k \cdot \alpha n \cdot \frac{T_A(\alpha n)}{\alpha n} \le k \cdot \alpha n \cdot \frac{T_A(n)}{n} = k\alpha \cdot T_A(n),$$

it follows that

$$T_B(n) \le (1 + k\alpha)\, T_A(n) + k \cdot T_B(\alpha n).$$

Hence we get the following recurrence relation for the function T_B:

$$T_B(n) \le \begin{cases} c' & \text{if } n \le c, \\ (1 + k\alpha)\, T_A(n) + k \cdot T_B(\alpha n) & \text{if } n > c. \end{cases}$$

Unfolding this recurrence relation i times yields

$$T_B(n) \le (1 + k\alpha) \sum_{j=0}^{i-1} k^j \cdot T_A(\alpha^j n) + k^i \cdot T_B(\alpha^i n).$$

We simplify the summation in this inequality, as follows.

$$\begin{aligned} \sum_{j=0}^{i-1} k^j \cdot T_A(\alpha^j n) &= \sum_{j=0}^{i-1} k^j \cdot \alpha^j n \cdot \frac{T_A(\alpha^j n)}{\alpha^j n} \\ &\le \sum_{j=0}^{i-1} k^j \cdot \alpha^j n \cdot \frac{T_A(n)}{n} \\ &= \sum_{j=0}^{i-1} (k\alpha)^j \cdot T_A(n) \\ &= \frac{(k\alpha)^i - 1}{k\alpha - 1} \cdot T_A(n) \\ &\le \frac{(k\alpha)^i}{k\alpha - 1} \cdot T_A(n) \\ &\le 2\, \frac{(k\alpha)^i}{k\alpha} \cdot T_A(n) \\ &= 2(k\alpha)^{i-1} \cdot T_A(n), \end{aligned}$$

where the last inequality follows from the fact that $k\alpha \geq 2$. It follows that

$$T_B(n) \leq 2(1+k\alpha)(k\alpha)^{i-1} \cdot T_A(n) + k^i \cdot T_B(\alpha^i n).$$

For $i = \lceil (\log(n/c))/(\log(1/\alpha)) \rceil$, we have

$$\alpha^i \leq \alpha^{(\log(n/c))/(\log(1/\alpha))} = c/n$$

and

$$k^i \leq k^{(\log(n/c))/(\log(1/\alpha))+1} = k\,(n/c)^{(\log k)/(\log(1/\alpha))}.$$

Therefore,

$$\begin{aligned} T_B(n) \;\; \leq \;\; & 2(1+k\alpha)(n/c)^{(\log k)/(\log(1/\alpha))}(c/(\alpha n)) \cdot T_A(n) \\ & + k\,(n/c)^{(\log k)/(\log(1/\alpha))} \cdot T_B(c). \end{aligned}$$

Since k, α, and c are constants, we have shown that

$$T_B(n) = O\left(n^{(\log k)/(\log(1/\alpha))-1} \cdot T_A(n) + n^{(\log k)/(\log(1/\alpha))}\right).$$

Since $T_A(n) = \Omega(n)$—this follows from Condition 4— we get our final upper bound on the running time of algorithm B:

$$T_B(n) = O\left(n^{(\log k)/(\log(1/\alpha))-1} \cdot T_A(n)\right).$$

2.1 An example: computing the closest pair

Let Π be the set of all finite point sets in the plane. For any set $S \in \Pi$, let

$$F(S) := \min\{d(p,q) : p \in S, q \in S, p \neq q\},$$

i.e., $F(S)$ is the minimum distance between any two distinct points of S. In this case, Condition 1 clearly holds. In order to satisfy Condition 2, we need an algorithm A that decides for any $S \in \Pi$ and any $t \in \mathbb{R}$, whether $F(S) < t$. Here is an algorithm A that does the job. Of course, we may assume that $t > 0$. We construct a grid whose cells have sides of length $t/\sqrt{2}$. If there is a cell that contains at least two points of S, then $F(S) < t$ and we are done. Otherwise, each cell of this grid contains zero or one point of S. If $F(S) < t$, then there are two distinct points p and q in S such that q is in one of the 24 cells in the "neighborhood" of p's cell; see Figure 2.

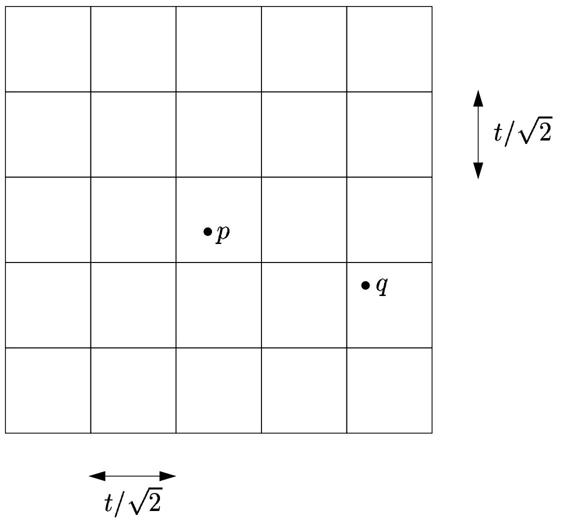

Figure 2: *The neighborhood of p's cell consists of 24 cells.*

Therefore, we do the following. For each $p \in S$, we search for all points q that are in the 24 cells comprising the neighborhood of p's cell and compute $d(p,q)$. (Observe that there are at most 24 such points q.) If in this process, we find two points p and q with $d(p,q) < t$, then we know that $F(S) < t$. Otherwise, we know that $F(S) \geq t$.

How do we implement this algorithm A? Any cell of our grid has the form

$$[it/\sqrt{2}, (i+1)t/\sqrt{2}) \times [jt/\sqrt{2}, (j+1)t/\sqrt{2}),$$

for some integers i and j. We call the pair (i,j) the *label* of this cell. Observe that the label of the cell that contains the point $p = (p_x, p_y)$ is given by $(\lfloor p_x\sqrt{2}/t \rfloor, \lfloor p_y\sqrt{2}/t \rfloor)$. If we store the labels of the non-empty grid cells in a balanced binary search tree, then it is not difficult to see that algorithm A can be implemented such that its running time $T_A(n)$ is $O(n \log n)$.

What about Condition 3? Consider a set $S = \{p_1, \ldots, p_n\}$ of n points in the plane, and assume for simplicity that n is a multiple of three. Define

$$S_1 := \{p_1, \ldots, p_{2n/3}\},$$

$$S_2 := \{p_1, \ldots, p_{n/3}, p_{2n/3+1}, \ldots, p_n\}$$

and

$$S_3 := \{p_{n/3+1}, \ldots, p_n\}.$$

Then $|S_i| = 2n/3$ for $1 \leq i \leq 3$, and

$$F(S) = \min(F(S_1), F(S_2), F(S_3)).$$

Hence, Condition 3 holds with $\alpha = 2/3$ and $k = 3$. Given the set S, the sets S_1, S_2 and S_3 can be computed in $O(n)$ time, which is bounded from above by $T_A(n)$. Finally Condition 4 and 5 clearly hold.

Since all five conditions hold, our algorithm B computes the minimum distance $F(S)$ of any set S of n points in the plane, in time

$$O\left(n^{(\log k)/(\log(1/\alpha))-1} \cdot T_A(n)\right) = O\left(n^{(\log 3)/(\log(3/2))} \log n\right) = O\left(n^{2.71} \log n\right).$$

This is, of course, pretty bad, because there is a much simpler $O(n^2)$-time algorithm.

3 A faster algorithm

How can we improve the running time of our algorithm B? The high running time is caused by the fact that there are k recursive calls in the worst case. This happens if

$$F(S_k) < F(S_{k-1}) < \ldots < F(S_2) < F(S_1).$$

On the other hand, we may be lucky: If

$$F(S_1) < F(S_2) < \ldots < F(S_{k-1}) < F(S_k),$$

then there is only one recursive call.

Here is the trick: in the for-loop of algorithm B, we do not consider the sets $S_1, \ldots, S_k$ in their natural order, but rather in a *random* order. As we will show later, in this way, the expected number of recursive calls will be logarithmic in k. We denote the new algorithm for computing $F(S)$ by $C(S)$. It is given in Figure 3.

We first analyze the expected number of indices i such that algorithm C is run on the set $S_{\sigma(i)}$. Let N be the random variable whose value is equal to the number of such indices. Furthermore, for any i with $1 \leq i \leq k$, let N_i

```
Algorithm $C(S)$
(* $S \in \Pi$; this randomized algorithm computes $F(S)$ *)
if $|S| \leq c$
then compute and return $F(S)$
else compute the sets $S_1, \ldots, S_k \in \Pi$ as in Condition 3;
    let $\sigma$ be a random permutation of $\{1, \ldots, k\}$;
    $min := \infty$;
    for $i := 1$ to $k$
    do if $A(S_{\sigma(i)}, min) = true$
        then $min := C(S_{\sigma(i)})$
        endif
    endfor;
    return $min$
endif
```

Figure 3: *A faster randomized algorithm for computing $F(S)$.*

---

be the random variable whose value is one if C is run on the set $S_{\sigma(i)}$, and zero otherwise. Then $N = \sum_{i=1}^{k} N_i$ and

$$E(N) = \sum_{i=1}^{k} E(N_i) = \sum_{i=1}^{k} \Pr(N_i = 1).$$

Fix an integer i with $1 \leq i \leq k$. We have $N_i = 1$ if and only if $A(S_{\sigma(i)}, min) = true$, which holds if and only if

$$F(S_{\sigma(i)}) < F(S_{\sigma(j)}) \text{ for all } j \text{ with } 1 \leq j < i.$$

Since σ is a random permutation of $\{1, \ldots, k\}$, $\sigma(i)$ is a random element of $\{\sigma(1), \ldots, \sigma(i)\}$. Hence, $F(S_{\sigma(i)})$ is a random element of $\{F(S_{\sigma(1)}), \ldots, F(S_{\sigma(i)})\}$. Therefore, $F(S_{\sigma(i)})$ is less than all other elements of the latter set with probability at most $1/i$. That is, we have proved that

$$E(N) \leq \sum_{i=1}^{k} 1/i < 1 + \int_1^k \frac{dx}{x} = 1 + \ln k.$$

Now we can analyze the expected running time of algorithm C. For any $S \in \Pi$, let $T_C(S)$ be the random variable whose value is equal to the running time of algorithm C on input S. Furthermore, define

$$T(n) := \max\{E(T_C(S)) : S \in \Pi, |S| = n\}.$$

Let S be an arbitrary set of Π having size n with $n > c$. Then

$$T_C(S) \leq T_A(n) + \sum_{i=1}^{k} \left(T_A(|S_{\sigma(i)}|) + N_i \cdot T_C(|S_{\sigma(i)}|)\right).$$

As in Section 2, we get

$$T_C(S) \leq (1 + k\alpha)\, T_A(n) + \sum_{i=1}^{k} N_i \cdot T_C(|S_{\sigma(i)}|).$$

Since the random variables N_i and $T_C(|S_{\sigma(i)}|)$ are independent, we have

$$\begin{aligned} E(T_C(S)) &\leq (1 + k\alpha)\, T_A(n) + \sum_{i=1}^{k} E(N_i) \cdot E(T_C(|S_{\sigma(i)}|)) \\ &\leq (1 + k\alpha)\, T_A(n) + \sum_{i=1}^{k} E(N_i) \cdot T(\alpha n) \\ &\leq (1 + k\alpha)\, T_A(n) + (1 + \ln k)\, T(\alpha n). \end{aligned}$$

Since the set S was arbitrary, we get the following recurrence relation for the expected running time $T(n)$ of algorithm C.

$$T(n) \leq \begin{cases} c' & \text{if } n \leq c, \\ (1 + k\alpha)\, T_A(n) + (1 + \ln k)\, T(\alpha n) & \text{if } n > c. \end{cases}$$

Unfolding this recurrence relation i times yields

$$T(n) \leq (1 + k\alpha) \sum_{j=0}^{i-1} (1 + \ln k)^j\, T_A(\alpha^j n) + (1 + \ln k)^i\, T(\alpha^i n).$$

In order to obtain a good upper bound on $T(n)$, we need one more condition:

Condition 6: $(1 + \ln k)\alpha < 1$.

We have

$$\begin{aligned}
\sum_{j=0}^{i-1}(1+\ln k)^j\, T_A(\alpha^j n) &= \sum_{j=0}^{i-1}(1+\ln k)^j \cdot \alpha^j n \cdot \frac{T_A(\alpha^j n)}{\alpha^j n} \\
&\leq \sum_{j=0}^{i-1}(1+\ln k)^j \cdot \alpha^j n \cdot \frac{T_A(n)}{n} \\
&= \sum_{j=0}^{i-1}(1+\ln k)^j \cdot \alpha^j \cdot T_A(n) \\
&\leq \sum_{j=0}^{\infty}((1+\ln k)\alpha)^j \cdot T_A(n) \\
&= \frac{1}{1-(1+\ln k)\alpha} \cdot T_A(n).
\end{aligned}$$

Hence,

$$T(n) \leq \frac{1+k\alpha}{1-(1+\ln k)\alpha} \cdot T_A(n) + \left(\sqrt{1/\alpha}\right)^i \cdot T(\alpha^i n).$$

For $i = \lceil (\log(n/c))/(\log(1/\alpha)) \rceil$, we have $\alpha^i n \leq c$ and

$$(1/\alpha)^i \leq (1/\alpha)^{(\log(n/c))/(\log(1/\alpha))+1} = n/(\alpha c).$$

Hence we have shown that

$$T(n) \leq \frac{1+k\alpha}{1-(1+\ln k)\alpha} \cdot T_A(n) + \sqrt{\frac{n}{\alpha c}} \cdot T(c).$$

Since k, α, and c are constants, it follows that the expected running time $T(n)$ of algorithm C satisfies

$$T(n) = O(T_A(n) + \sqrt{n}\,) = O(T_A(n)).$$

3.1 Back to our example: computing the closest pair

Let us consider again the example of Section 2.1. Hence, we want to compute the minimum distance in a set S of n points in the plane. In Section 2.1, we used the values $k = 3$ and $\alpha = 2/3$. Unfortunately, for these parameters, Condition 6 does not hold. Instead, we do the following. Let b be a positive integer, whose value will be decided later. Partition S into b subsets

$V_1, \ldots, V_b$, each having size n/b. For $1 \leq i < j \leq b$, define $S_{ij} := V_i \cup V_j$. Then each set S_{ij} has size $2n/b$. Furthermore, we have

$$F(S) = \min\{F(S_{ij}) : 1 \leq i < j \leq n\}.$$

Hence, we get the parameters $k = \binom{b}{2}$ and $\alpha = 2/b$. In order to satisfy Condition 6, we have to choose b such that

$$1 + \ln \binom{b}{2} < b/2.$$

The smallest b for which this holds is $b = 10$.

We have seen in Section 2.1 that the running time $T_A(n)$ of the decision algorithm A is $O(n \log n)$. Therefore, we can compute the minimum distance of any set of n points in the plane in $O(n \log n)$ expected time.

Exercise 1 Prove that the minimum distance of any set of n points in $\mathbb{R}^D$, where the dimension D is a constant, can be computed in $O(n \log n)$ expected time.

3.2 A second example: computing the diameter

In this second example, we show how our general framework can be used to compute the diameter of a planar point set. Hence, we take for Π the set of all finite point sets in the plane. For any set $S \in \Pi$, let

$$F(S) := \max\{d(p,q) : p \in S, q \in S\},$$

i.e., $F(S)$ is the maximum distance between any two points of S. We only have to give an algorithm A for which Condition 2 holds, because all other conditions hold as in the previous example.

Observe that we now have a maximization problem. This means that in our general framework, we have to replace "minimum" by "maximum". Also, algorithm A must decide whether $F(S) > t$ or $F(S) \leq t$.

Let S be a set of n points in the plane, and let t be a real number. We want an algorithm A that decides whether or not the diameter $F(S)$ of S is larger than t. Of course, we may assume that $t > 0$.

For any $p \in S$, let $D(p,t)$ be the disk of radius t centered at p. Then

$$F(S) \leq t \text{ if and only if } S \subseteq \cap_{p \in S} D(p,t).$$

Algorithm A will decide if the condition on the right-hand side holds. This is done in the following way.

Step 1: Compute the intersection of the disks $D(p,t)$, $p \in S$. This intersection can be computed using a divide-and-conquer algorithm, whose merge-step takes $O(n)$ time. Hence, the total time to compute the entire intersection is $O(n \log n)$. Observe that the intersection is a convex "circular-gon" whose edges are circular arcs. We denote this circular-gon by I.

Step 2: For each point p of S, test if p is contained in I. If this is true for all p, then $F(S) \leq t$. Otherwise, we have $F(S) > t$. Testing if a point is contained in I takes $O(\log n)$ time. Hence, the total time for Step 2 is $O(n \log n)$.

Since the total time of algorithm A is $O(n \log n)$, our general framework gives a randomized algorithm that computes the diameter $F(S)$ of S in $O(n \log n)$ expected time.