随机线性“筛法”求 *d* 维空间中的最近点对

天津南开中学 李羽修

（本文大部分内容来自参考文献[1]）

求欧几里德空间中的最近点对是一个经典的计算几何问题。在维数 d 确定的前提下，我们可以很容易得到一个 $O(n^2)$的算法——只要枚举C_n^2对点，并在 $O(1)$的时间内算出距离，取其最小值就可以了。几乎所有的算法书中都讲到了一个 $O(n \log n)$的分治算法，这个算法达到了基于比较的求解算法之时间复杂度的下限（类似比较排序和凸包问题）。然而，借助随机化和 Hashing 的思想，我们可以得到一个比较简单的期望复杂度为 $O(n)$的算法。

首先我们来定义一下要解决的问题：对于一个确定的正整数 d，给定一个含 n 个元素有限集合$S \subset \Re^d$，求$\min\left\{\left|P_1P_2\right|\right\}$，其中 $P_1, P_2 \in S$。这里我们只讨论 $d = 2$ 的情况，其他情况的算法可以类似地推广得到。

我们假设两点间的距离可以在常数时间内计算得到；同样我们假设取整的操作也可以在常数时间内计算得到。为了简化问题，我们认为 S 中不存在重复的点。

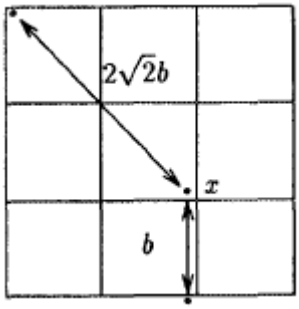
首先，我们设 S 为初始的点集，$\delta(S)$表示 S 中距离最近的一对点的距离。如果点集中的点没有被编号，我们将其编号以便之后的过程使用。这个算法分为两步：首先我们计算$\delta(S)$的一个近似值；然后利用这个近似值计算$\delta(S)$。算法的主要思想是“筛选”——在点集中删除某些不满足我们要求的点。设 S_i 为第 i 次迭代开始之前剩下的点（初始时 $S_1 = S$），我们重复进行过滤的过程直到 S_i 为空。在过滤的过程中，$\left|S_i\right|$ 呈几何级数递减。在这个过程结束的时候，我们可以得到一个不超过$3\delta(S)$的近似值，然后利用该值可以很容易得到$\delta(S)$的精确值。

筛选的过程：在 S_i 中随机选取一个点 x_i。设 $d(x)$表示在当前集合 S_i 中从 x 出发到距离其最近的点的距离。通过枚举 S_i 中的所有点，我们可以计算出 $d(x_i)$。然后从 S_i 中删除所有满足$d(x) \geq d(x_i)$的点 x。在这个过程中，我们还可以删除一部分满足$d(x_i) > d(x) > \dfrac{d(x_i)}{3}$的点 x，但必须在 S_{i+1} 保留所有满足$d(x) \leq \dfrac{d(x_i)}{3}$的点 x。如此迭代直到集合 S_i 为空，设 i^*为最后一次迭代的序号，即$S_{i^*+1} = \varnothing$，$S_i \neq \varnothing$。设x_{i^*}为第 i^*次迭代时，从S_{i^*}中随机选取的点。这样，整个点集的最近距离$\delta(S)$至多是$d(x_{i^*})$，至少是$\dfrac{d(x_{i^*})}{3}$。我们得

到了一个不超过$3\delta(S)$的近似值$d(x_{i^*})$。

剩下的工作便是实现前面的过程，并利用$d(x_{i^*})$计算出$\delta(S)$。利用在其他算法中提到过的“相邻”的概念，这两项工作很容易解决。

在坐标系中建立一个格子边长为 *b* 的网格。定义点 *x* 的相邻空间为包含 *x* 的格子加上 8 个相邻的格子（如右图）。设 *N*(*x*)为点 *x* 的相邻空间中所有点的集合。我们可以得到以下两个结论：



1. 与 *x* 的距离不小于 3*b*（实际是$2\sqrt{2}b$）的点一定不在 *N*(*x*)中。
2. 与 *x* 的距离不大于 *b* 的点一定在 *N*(*x*)中。

注意与 *x* 的距离在 *b* 和$2\sqrt{2}b$之间的点既有可能在 *N*(*x*)中也可能不在 *N*(*x*)中。

筛选过程的实现：对于第 *i* 次迭代，设 x_i 为在 S_i 中随机选取的点。建立边长为$b=\frac{d(x_i)}{3}$的网格。当且仅当一个点 *x* 的相邻空间中除 *x* 以外没有其他点（即$|N(x)|=1$）时，删除点 *x*。利用 Hashing 技术，这一步可以在$O(|S_i|)$期望步数内解决。特别地，我们可以在$O(|S_i|)$的时间内计算出一个完美的 Hash 函数，使得我们可以用$O(|S_i|)$的空间表示$|S_i|$个非空的格子。同样，采用 Universal Hashing 的技术，可以使上面得到的复杂度对于任意输入数据都成立。

引理 1：当筛选过程结束（$S_{i^*+1}=\varnothing$）时，$\frac{d(x_{i^*})}{3}\le\delta(S)\le d(x_{i^*})$。

证明：显然，根据定义，$\delta(S)\le d(x_{i^*})$。根据关于 *N*(*x*)的结论 1，在第 *i* 次迭代时所有满足$d(x)\ge d(x_i)$的点 *x* 都被删除掉了。这样，序列$\{d(x_i)\}$是单调递减的。设(*u*, *v*)为 *S* 中的一个最近点对，j^*为 *u* 被删掉的一次迭代。根据结论 2，满足$d(x)\le\frac{d(x_{j^*})}{3}$的所有点 *x* 都不会从$S_{j^*}$中删除。注意到由于 *u* 和 *v* 都必存在于S_{j^*}中，$d(u)=\delta(S)$，因此$\delta(S)\ge\frac{d(x_{j^*})}{3}\ge\frac{d(x_{i^*})}{3}$。

根据$d(x_{i^*})$计算$\delta(S)$：我们将利用以下两个结论：对于一个边长 *b* 满足$\frac{b}{3}\le\delta(S)\le b$的网格，

3. *S* 中每一个点的相邻空间中包含至多常数个点。
4. 最近点对中的每一个点都在另一个点的相邻空间中。

建立一个边长为 $d(x_{i^*})$（$\delta(S)$的近似值）的网格。对于每一个非空的格子，我们列出格子中所有的点。同样，利用上面提到的 Hashing 技术，这一步可以在线性期望时间内完成。然后，对于每一个点 x，如果 $N(x)$中存在其他点的话，我们计算出 x 到 $N(x)$中其他各点的距离的最小值。根据结论 3，每一个点都可以用暴力法（或者其他更好的方法）在常数时间内处理完毕；结论 4 保证了对于最近点对(u, v)中的一个点 u，另一个点 v 在 $N(u)$中。这样，通过计算这些距离的最小值，必能找到 S 中的最近点对。

下面我们给出这个算法的更多细节。

步骤 0：初始化 $S_1 = S$，$i = 1$。

步骤 1：在 S_i 中随机选取点 x_i，计算 $d(x_i)$即 S_i 中到 x_i 的最小距离。

步骤 2：建立边长为 $b = \frac{d(x_i)}{3}$ 的网格；设 $X_i = \{x_j \mid N(x_j) = \{x_j\}\}$；$S_{i+1} = S_i - X_i$。

步骤 3：如果 $S_{i+1} \neq \varnothing$，把 i 加 1 并回到步骤 1。否则令 $i^* = i$。

步骤 4：建立边长为 $d(x_{i^*})$的网格。对于 S 中每一个点 x，计算 $N(x)$中其他各点（如果存在的话）到 x 的最小距离。对这些距离取最小值即为 $\delta(S)$。

复杂度分析：唯一需要证明的是筛选的过程期望时间复杂度为线性。上面已经提到，第 i 次迭代的时间花费为 $|S_i|$ 的一次函数。我们来证明对于 $i = 1, 2, 3 \ldots$，$|S_i|$ 期望至少以几何级数递减。证明的关键是考查非降序列 $\{d(x): x \in S_i\}$。当随机选取 x_i 时，所有满足 $d(x) \geq d(x_i)$ 的点 x 都将被删除，这样，每次迭代时，平均至少一半的点会被删除，于是期望得到几何级数递减。特别地，我们有：

引理 2：$E\left(\sum_{i=1}^{i^*} |S_i|\right) \leq 2n$。

证明：令 $s_i = |S_i|$。我们首先归纳证明对于 $i \geq 1$，有 $E(s_i) \leq \frac{n}{2^{i-1}}$。根据上面的分析，$E(s_{i+1}) \leq \frac{s_i}{2}$。这样，$E(s_{i+1}) = E(E(s_{i+1})) \leq E\left(\frac{s_i}{2}\right) = \frac{1}{2}E(s_i)$。根据归纳假设，$E(s_{i+1}) \leq \frac{1}{2}\frac{n}{2^{i-1}} = \frac{n}{2^i}$。然后根据数学期望的可数可加性，我们有：

$$E\left(\sum_{i=1}^{i^*} s_i\right) \leq E\left(\sum_{i=1}^{n} s_i\right) = \sum_{i=1}^{n} E(s_i) \leq \sum_{i=1}^{n} \frac{n}{2^i - 1} \leq 2n。$$

于是我们有：

定理 随机化“筛法”能够在 $O(n)$期望时间和 $O(n)$空间内解决最近点对问题。

时间效率对比：我的程序没有使用 Perfect Hashing 和 Universal Hashing 的技术，且由于实现的问题，在测试大数据的时候有些未知 bugs，敬请海涵。以下为暴力法、分治法、筛法的运行时间对比，测试机配

置为 Pentium III 866MHz，128MB RAM，操作系统为 Windows Server 2003，程序使用 Mingw32 的 g++编译，未使用编译器优化开关，所有时间单位为秒，计算实际用时。

数据规模	暴力法	分治法	筛法
10	0.00	0.00	0.00
100	0.00	0.00	0.01
1000	0.24	0.01	0.02
5000	6.13	0.06	0.10
7500	13.76	0.11	0.15
10000	24.60	0.14	0.20
25000	157.17	0.42	0.73
50000		0.94	1.76

从表中可以看出，由于系数比较大，筛法的效率并不比分治法高。我的 Hash 大小为 12347，采用拉链法解决冲突。可以看出 1000、5000、7500、10000 四个数据的时间是呈线性变化的，而当数据规模超过 Hash 空间的时候时间效率明显下降。如果大家有兴趣使用 Perfect Hashing 和 Universal Hashing 技术（参考文献[2]）实现的话，应该会比我的程序效率更高。

【参考文献】

[1] Samir Khuller, Yossi Matias. A Simple Randomized Sieve Algorithm for the Closest-Pair Problem.

[2] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein. Introduction to Algorithms. Second Edition. The MIT Press

【源程序】

Sieve.cpp Brute.cpp Dnc.cpp Gen.cpp Point.h