空间最近点对的计算机算法研究

胡金初　张晓红

（上海师范大学数理信息学院　上海 200234）

摘　要　最近点对问题是空中交通控制系统中的一个重要问题，并且在许多领域都有应用，也是计算几何学研究的基本问题之一。利用分治法解决该问题的线性和平面情况，算法可以在 $O(n*\log n)$ 时间内完成。本文在此基础上，进一步实现空间最接近点的算法，并对算法的复杂性进行分析。

关键词　算法复杂性，最近点对，空间，分治法

The Computer Algorithms of Finding Shortest Distance Pair of Points in Space

HU Jin-Chu　ZHANG Xiao-Hong

(Shanghai Normal University, Shanghai 200234)

Abstract　Finding Shortest Distance pair of points in space is the important problem of the air traffic control system. There are a lot of applications with the problem and it is also a basic one of the computing geometry study. By methods of divide and conquer, the problem has been solved from the points of linearity and plane, it can be accomplished within the $O(n*\log n)$ time. Under the base of unidimentional and two-dimensional algorithm, this paper solves the Shortest Distance pair of points in space problem and analyzes the complexity of the algorithms.

Keywords　Algorithms complexity, Shortest distance pair of points, Space, Divide and conquer

1　问题的描述

在空间飞机和航天器的运行，在海水中潜艇的行动都与三维最接近点对问题有关。这也是计算几何学研究的基本问题之一。该问题的形式化描述为：在给定的集合 S 中由 n 个点组成，需要找出其中的一对点，使得由 n 个点组成的所有点对中，该点对间的距离最小。我们用 $d(p,q)$ 表示 p 和 q 两点间的距离，即问题需要求解：$\min\{d(p,q)\mid p\in S,\ q\in S\}$。最接近点对有可能不是 1 对，为了讨论方便，我们只找出其中的 1 对作为该问题的解。根据这 n 个点在空间的分布情况，可以分成线性、平面和空间三种情况来讨论。

在线性情况下，S 中的 n 个点可以用 x 数轴上的 n 个实数 $x_1,x_2,\cdots,x_n$ 来表示。最接近的点对即为这 n 个实数中距离相差最小的 2 个实数。即成为求 $\min\{|x_p-x_q|\}$ 的问题。解决这一问题可以先确定 n 个点坐标的中位数 m，以 m 为界将集合 S 划分为 2 个子集合 S_1 和 S_2。坐标值小于和等于 m 的点在集合 S_1 中，大于 m 的点在集合 S_2 中。递归地分别在 S_1 和 S_2 上找出最接近的点对 $\{p_1,p_2\}$ 和 $\{q_1,q_2\}$，并设 $d=\min\{|p_1-p_2|,|q_1-q_2|\}$。$S$ 中的最接近点对间的距离或者是 d，或者是某对 $\{p_3,q_3\}$ 间的距离，其中 p_3 在 S_1 中并且 $p_3\in(m-d,m)$，q_3 在 S_2 中并且 $q_3\in(m,m+d)$。显而易见，这时 p_3 一定是 S_1 中的最大点，q_3 是 S_2 中的最小点。只要判断它们两点的距离就可以了。从而我们用线性时间就可以将 S_1 的解和 S_2 的解合并成为 S 的解。即该算法的分割和合并步骤总共耗时 $O(n)$。求解算法耗费的计算时间 $T(n)$ 满足的递归方程：

$$\begin{cases}T(n)=O(1) & n<4\\ T(n)=2T(n/2)+O(n) & n\geqslant 4\end{cases}$$

通过解递归方程，可得到 $T(n)=O(n\log n)$。

如果集合 S 中的点分布在平面中，每个点都具有 x 和 y 两个坐标。解决这一平面问题，可以选取一条垂直线 $L|_{x=m}$ 来作为分割直线，其中 m 为集合 S 中各点 x 坐标的中位数。将集合 S 分割为 S_1 和 S_2 两个半平面区域中的子集，递归地分别在 S_1 和 S_2 上找出具有最小距离的点对 $d_1(p_1,p_2)$ 和 $d_2(q_1-q_2)$，并设 $d=\min\{d_1,d_2\}$。用 P_1 和 P_2 分别表示在直线 L 左边和右边与其距离在 d 范围的点构成的两个垂直长条平面区域，$P_1:\{p\mid_{|m-x(p)|\leqslant d\wedge p\in P_1}\}$；$P_2:\{q\mid_{|x(q)-m|\leqslant d\wedge q\in P_2}\}$。分析可以知道，$S$ 中的最近点对的距离或者是 d，或者是某个 $\{p,q\}$ 点对的距离，其中 $p\in P_1$，$q\in P_2$。如果 $\{p,q\}$ 是 S 中的最近点对，则必有 $\text{distance}(p,q)<d$。对于 P_1 中的任意一点 p，满足这个条件的 P_2 中的点 q 一定落在一个 $d\times 2d$ 的矩形 R 中；可以证明在矩形 R 中最多只有 6 个 S 中的点作为 P_1 中点 p 的最近点对的候选者。因此很容易在 $O(n)$ 时间内进行处理。在进行分治处理之前需要对集合 S 中点的坐标采用预排序技术，预先将 n 个点依次按照坐标排序。分治算法中只对排好序的序列做一次线性扫描，即可完成合并。因此，平面最近点对算法耗费的计算时间 $T(n)$ 也满足递归方程，可以在 $O(n*\log n)$ 时间内完成。

在上述算法的基础上，进一步推广到空间最近点对问题的求解，处理的方法还是采用分治的思想，将 n 个点的集合 S 分成 2 个子集 S_1 和 S_2（$S_1\cup S_2=S$），每个集合中约含有 $n/2$ 个点。然后在每个子集中递归的求其最近的点对，最后合并得出 S 中的最近点对。

求解空间的最接近点对问题，需要引入一个平面对 n 个点进行划分，分为两个点数大致相同的子空间区域，并递归求解子区域中的最接近点对，然后合并解得空间集合 S 中的最近点对的问题。

2　空间最近点对问题的算法

2.1　解决问题的算法

集合 S 中的点如果分布在空间中，情况会复杂一点。这时集合中的每一个点都有 x、y、z 三个坐标值。为了将集合 S 中的空间点线性分割为大小大致相等的 2 个子集 S_1 和 S_2，需要引入一个垂直平面 $P|y=m$ 来作为分割平面，其中 m 为集合 S 中各点的 y 坐标值的中位数。从而将集合 S 中的点分割为两个子空间 S_1 和 S_2 中的点，其中 $S_1=\{p\mid_{y(p)\leqslant m\wedge p\in S}\}$，$S_2=\{q\mid_{y(q)>m\wedge q\in S}\}$。从而使得 S_1 和 S_2 分别位于平面 P 的左侧和右侧，且 $S=S_1\cup S_2$。

类似与线性和平面情况的求解算法，递归地在 S_1 和 S_2 上解最近点对问题。设求解得到 S_1 和 S_2 中的最近点对 $\{p_1,$

$p_2\}$和$\{q_1,q_2\}$，对应的最小距离分别为d_1和d_2，并设$d=\min\{d_1,d_2\}$。如果集合S中的最近点对的距离小于d，则这两个点p_3、q_3必定分别属于子集S_1和S_2。即S中的最近点对的距离或者是d，或者是某个$\{p_3,q_3\}$点对的距离，其中$p_3\in S_1$，$q_3\in S_2$，如图1所示。

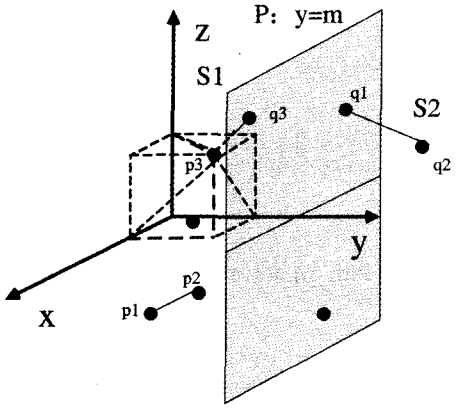

图1　空间最接近点对分布情况

如果集合S中的最近点对是由$\{p_1,q_1\}$或者是$\{p_2,q_2\}$组成，那么问题的求解就成为一个递归问题，很容易在$O(n*\log n)$时间内完成递归方程的求解。

但是如果集合S中的最近点对是由$\{p_3,q_3\}$组成，我们需要分析这时点对的处理的时间复杂性。通过分析可以知道这点对有$\text{distance}(p_3,q_3)<d$，而且点$p_3$和$q_3$距平面$P|y=m$的距离均小于$d$。为了说明$(p_3,q_3)$的求解，我们引入$C_1$和$C_2$分别表示垂直平面$P$左边和右边的宽度为$d$的两个垂直长条空间区域，$C_1$：$\{p\ |_{|m-y(p)|\leqslant d\wedge p\in C1}\}$；$C_2$：$\{q\ |_{|y(q)-m|\leqslant d\wedge q\in C2}\}$，则有$p_3\in C_1$，$q_3\in C_2$。这时我们所关心的是，$C_1$中的某一点与$C_2$中有条件作为最接近点对候选的点有多少个。在最坏情况下，C_1和C_2中分别包含了原S中1/2的点，表面上看一共需要考察$n^2/4$对，工作量很大。但是经过分析，在C_1和C_2中的点具有稀疏性质，这就使我们不必一一检查所有$n^2/4$对候选者。考虑C_1中任意一点p，它若与C_2中的点q构成最接近点对的候选者，则必须满足$\text{distance}(p,q)<d$。那么具备这个条件的C_2中的点一定落在一个$d\times 2d\times 2d$的长方体单元C中，如图2所示。

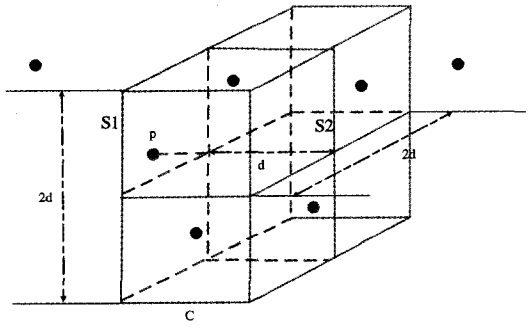

图2　包含点q的$d\times 2d\times 2d$的长方体C

由前面变量d的定义可知：在C_2中任何2个点的距离都不小于d。由此可以推出长方体C中最多只能有24个集合S中的点。我们进一步可以将长方体C的长和宽分别为$2d$的边3等分和4等分，而将它的高为d的边2等分，由此导出24个大小相等的$(d/2)\times(d/2)\times(2d/3)$的小长方体。

如果长方体单元C中有多于24个集合S中的点，则由鸽舍原理易知至少有一个$(d/2)\times(d/2)\times(2d/3)$的小长方体中有2个$S$中的点。设$u,v$是这样的2个点，它们位于同一个小长方体中。那么这2点之间的距离满足下面的关系式：$[\text{distance}(u,v)]^2\leqslant(d/2)^2+(d/2)^2+(2d/3)^2=17d^2/18<d^2$。即$\text{distance}(u,v)<d$，这就与前面$d$的定义相矛盾。这就证明长方体$C$中最多只有24个满足考察条件的点。

由于长方体C的稀疏性质，对于C_1中任一点p，C_2中最多只有24个点有条件与它构成最接近点对的候选。因此，在分治法的合并步骤中，我们最多也只需要考察$24\times n/2=12n$对点，而不需要考察$n^2/4$对点。显然计算的复杂性大大下降。

为了进一步确定C_1中每个点p最多考察C_2中的哪24个点，从而转化成可以操作的程序，我们可以将点p和C_2中所有S_2的点投影到平面$P|y=m$上。由于能与p点一起构成最接近点对候选者的S_2中点一定在长方体C中，因此它们在平面P上的投影点与点p在P上投影点的距离小于d。由上面的分析可知，这种投影点最多只有24个。因此，若将C_1和C_2中所有S的点依次按其x坐标和z坐标排好序，则对C_1中任意点p而言，对已经按照x坐标和z坐标排好序的点列作一次线性扫描，就可以找出所有最接近点对的候选者。设点$q(x,y,z)$为C_2中可以与C_1中的一点$p(x_0,y_0,z_0)$构成候选点对的排好序的24个点中的一点，则它的坐标值满足$x\in(x0-d,x0+d)$，$z\in(z0-d,z0+d)$，即投影点在以p的投影点为中心的$2d\times 2d$的正方形区域中的点是我们要考察的候选点。由于采用这个算法，对于需要考察的点是和n线性相关的，这就意味着我们可以在$O(n)$时间内完成分治法的合并步骤。点的分布情况如图3所示。

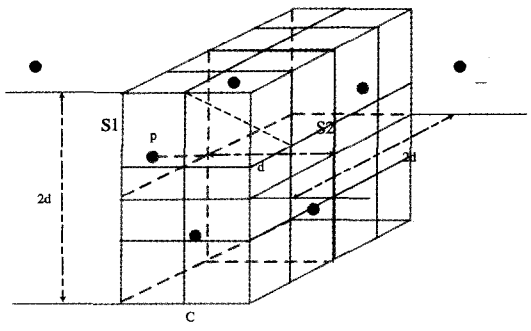

图3　长方体C中点分布的稀疏性

2.2　算法的描述和伪代码表示

通过上面的分析，下面给出用分治法求空间最接近点对的算法spair，伪代码如下：

```
double spair(S)
{n=|S|;/*|S|表示S中点的个数*/
  if(n<2) return ∞;
    1. m=S中各点y坐标的中位数;
       利用平面P:y=m划分构造子空间S1和S2;
       /* S1={p|y(p)≤m∧p∈S},S2={q|y(q)>m∧q∈S}*/
    2. d1=spair(S1);
       d2=spair(S2);/*递归调用*/
    3. dm=min(d1,d2);
    4. 设C1是S1中距垂直分割面P的距离在dm之内的所有点组成的集合;
       C2是S2中距垂直分割面的距离在dm之内的所有点组成的集合;
       将C1和C2中点依其x坐标值和z坐标值排序;
       设X1、X2是C1、C2依据x坐标值排好序的点集;Z1、Z2是X1、X2再依据z坐标值排好序的点集;
    5. 依次扫描Z1以及对于Z1中每个点考察Z2中相应的最多24个点,并且完成合并操作;
       当Z1中的扫描指针沿着某一个方向移动时,Z2中的扫描指针可在2dm×2dm的方形区间内移动;
       设dl是按这种扫描方式找到的点对间的最小距离;
    6. d=min(dm,dl);
       return d;
       }
```

另外需要说明的是，由于空间的对称性，算法中的分割平面也可以是由各点 x 坐标值的中位数决定的，与 YOZ 面平行的平面作为分割面；或者取各点 z 坐标值的中位数决定的，与 XOY 面平行的平面作为分割面。其余坐标做相应的改动。

2.3 算法的复杂性分析

下面需要分析一下算法 spair 的计算复杂性。对于确定空间 n 个点的点集 S，算法耗时为 $T(n)$。算法的第 1 步完成查找各点 y 坐标值的中位数并进行划分，我们已经知道中位数的查找算法可以在线性时间 $O(n)$ 内完成；第 2 步是递归调用算法，通过解递归方程，算法耗时为 $O(n*\log n)$；第 3 步的计算和第 6 步的计算都是常数时间；在前面 2.2 节中已经分析了第 5 步的工作是线性相关的，所以算法耗时为 $O(n)$。若在每次执行第 4 步时都需要进行排序，则在最坏情况下第 4 步都需要用 $O(n\log n)$时间。这显然并不是最优的，这就包含了许多重复的排序操作。因此，在程序中采取预排序技术，即在使用分治法之前，预先将 S 中 n 个点依次按其 x 坐标值和 z 坐标值排好序。在执行分治法的第 4 步时，只要对已经排好序的点列作一次线性扫描，即可抽取出我们所需要的排好序的 P_1 和 P_2 中的点集 Z_1 和 Z_2。然后，在第 5 步中再对 Z_1 作一次线性扫描，即可求得 d_l。因此，第 4 步和第 5 步的两遍扫描合在一起只要用 $O(n)$时间。这样一来，经过预排序处理后的算法 spair 所需的计算时间 $T(n)$也满足递归方程：

$$\begin{cases} O(1) & n<4 \\ T(n)=2T(n/2)+O(n) & n\geqslant 4 \end{cases}$$

通过解递归方程，算法耗时为 $T(n)=O(n*\log n)$，再加上开始点集 S 的预排序所需的计算时间为 $O(n*\log n)$。因此，整个算法所需的计算时间为 $O(n*\log n)$。

结束语 有关空间最接近点对的问题，有很强的应用背景，许多问题的研究和它有关，大到宇宙空间中的运行星球，小到物质内部分子和原子的运动，都在三维空间中运动。研究各个点的位置以及最接近点对算法，具有实际意义。本文采用分治法实现了三维空间中最接近点对算法。是从线性和平面的最接近点对问题，进一步推广到三维空间问题的一般求解，具有典型的特征。同时也论证了算法可以在 $O(n*\log n)$时间内完成。在渐近的意义下，此算法已是最优的了。这一算法可以直接应用于复杂的空间交通控制系统中。

参 考 文 献

1 王晓东．算法设计与分析．北京：清华大学出版社，2002
2 Knuth D E. The Art of Computer Programming. Addison-Wesley, 2002
3 Baase S. Computer Algorithms Introduction to Design and Analysis(3rd edition). Pearson Education, 2000

(上接第 215 页)

8,10 和 11。只满足支持度和兴趣度的规则有 1,4,8,10 和 11。只满足兴趣度和复杂度的规则有 1,7,8,10 和 11。类似地，用户还可以用阈值的析取形式表示约束条件得到相应的概念格，在概念格上抽取感兴趣的规则。

从格中选择出满足条件的规则，用户可以根据规则，分别找出规则所对应的质心，按相应的比例进行物品的搭配进货。例如规则 1{可乐}→{牙膏，香皂}满足用户的阈值条件，规则 1 的支持集为$\{I_2, I_5, I_8, I_9, I_{13}\}$，所对应的质心为(可乐，牙膏，香皂)=(5.2,3.6,2.4)。因为规则的可信度为 100%，所以，可以根据大约按可乐：牙膏：香皂=1：9/13：6/13 的比例搭配进货。当多个规则中有商品交叉时，进货比例按叠加原则处理。

叠加原则：给定规则：$X_1\rightarrow Y_1$ 和：$X_2\rightarrow Y_2$，其支持集分别为 $I_1^*, I_2^*\subseteq I$。设 $X_1\cup Y_1=\{x_1,\cdots,x_k, x_{k+1},\cdots,x_n\}$，$X_2\cup Y_2=\{y_1,\cdots,y_k,y_{k+1},\cdots,y_m\}$，$x_i=y_i(1\leqslant i\leqslant k)$。若这两规则的质心分别为：$(a_1,\cdots,a_k, a_{k+1},\cdots,a_n)$ 和 $(b_1,\cdots,b_k, b_{k+1},\cdots,b_m)$，则规则质心叠加和为：$(|I_1^*|^2a_1+|I_2^*|^2b_1,\cdots, |I_1^*|^2a_k+|I_2^*|^2b_k, |I_1^*|^2a_{k+1},\cdots,|I_1^*|^2a_n, |I_2^*|^2b_{k+1},\cdots,|I_2^*|^2b_m)$。

3.3 约束规则格抽取算法

现在，把上述的思路表示成算法的形式。

算法 2 约束规则格抽取算法

输入：交易数据样本集和阈值组合
输出：满足阈值条件的关联规则概念格
过程：
步骤 1：把交易数据样本集转化为形式上下文(G_1,M_1,I_1)的表格表示。//* G_1 是交易事件集，M_1 是商品集合，表中交叉处表示交易事件中对应商品的数量 * //
步骤 2：调用算法 1 抽取(G_1,M_1,I_1)中的关联规则集 R；
步骤 3：计算关联规则 $R\in R$ 的支持度、兴趣度、复杂度和平衡度；
步骤 4：生成规则集 R 所对应的形式上下文(G_2,M_2,I_2)；//* $G_2=R$，$M_2=$｛支持度，兴趣度，复杂度，平衡度｝，表中交叉处表示规则对应属性的取值 * //
步骤 5：根据用户的阈值约束将(G_2,M_2,I_2)修改为(G_2,M_2,I_3) // * 将不符合用户阈值条件的属性改为空(Null) * //
步骤 6：调用算法 3 生成约束的规则格

算法 3 概念生成算法[7]

输入：形式上下文(G, M, I)，其中 $G=\{g_1,\cdots,g_N\}:=\{1,2,\cdots,N\}$ 和 $M=\{m_1,\cdots,m_k\}$
输出：概念序列
步骤 1：$A:=\Phi''$，$\Xi:=\{\}$// * Ξ 表示概念集 * //
步骤 2：FOR $i=N$ to 1 DO
IF $A<_i((A\cap\{1,2,\cdots,i-1\})\cup\{i\})''$
THEN$A\oplus i=((A\cap\{1,2,\cdots,i-1\})\cup\{i\})''$
$\Xi\leftarrow\Xi\cup\{(A,A')\}$
$A\leftarrow A\oplus i$
ELSE $i:\leftarrow i-1$
步骤 3：对 A 执行步骤 2 的循环
步骤 4：输出 Ξ

总结 本文主要讨论了如何用形式概念分析理论进行超市交易数据分析和挖掘。在关联规则柔性选取的基础上，求出规则的质心，然后根据叠加原则进行商品搭配进货决策。柔性体现在用户可以自行定义阈值选取感兴趣的规则，来指导决策规划。

参 考 文 献

1 Lei Yuxia, Wang Yan, Cao Baoxiang, Yu Jiguo. Concept Interconnection Based on Many-Valued Context Analysis. In: Z.-H. Zhou, H. Li, Q. Yang, eds. PAKDD'07, LNAI 4426, 2007. 623～630
2 Stumme G, Taouil R, Bastide Y, et al. Intelligent Structuring and Reducing of Association Rules with Formal Concept Analysis. In: F. Baader, G. Brewka, T. Eiter, eds. KI 2001, LNAI 2174, 2001. 335～350
3 周欣，沙朝锋．兴趣度—关联规则的又一个阈值．计算机研究与发展，2000，37(5)：627～633
4 谢志鹏，刘宗田．概念格与关联规则发现．计算机研究与发展，2000，37(12)：1415～31421
5 周皓峰，朱扬勇，施伯乐．一个基于兴趣度的关联规则采掘算法．计算机研究与发展，2002，39(4)：450～457
6 胡吉明，鲜学丰．挖掘关联规则中 Apriori 算法的研究与改进．计算机技术与发展，2006，16(4)：99～101，104
7 Bernhard G, Wille R. Formal Concept Analysis: Mathematical Foundations. Springer, 1999
8 Zhang Ji-fu, Zhang Su-lan, Hu Li-hua. Constrained concept lattice and its construction method(In Chinese)[J]. CAAI Transactions on Intelligent Systems, 2006, 1 (2): 31～38