最简单的计算几何入门基础简介选讲

APIO2021

清华大学 何昊天 kiana810@126.com

研究计算几何问题的方法思路

- 对问题几何性质的剖析
- 对算法和数据结构的应用

主要内容

- 半平面交
- 凸包

半平面交问题

- 设 $H = \{h_1, h_2, \cdots, h_n\}$ 为一族双变量线性约束，也即每个 h_i 形如 $a_i x + b_i y \leq c_i$，其中 x, y 为约束变量，a_i, b_i, c_i 为常数且 a_i, b_i 不同时为 0
- 现在我们想求出满足所有约束的变量取值范围，从几何上我们可以将每个约束看作以直线 $a_i x + b_i y = c_i$ 为界的半平面，则所求范围就是这些半平面的交
- 在这一节中我们将从几何的角度回顾半平面交问题，然后通过几何观点来考察与之非常相似的线性规划问题，最后将后者用到的一些方法再应用到另一个几何问题中

凸集

定义

设 S 是平面上的一个点集，若 $\forall p, q \in S$ 满足线段 $\overline{pq}$ 完全包含在 S 中，则称 S 是一个凸集.

引理

两个凸集的交仍是凸集.

- 显然半平面是凸集，由上述引理即得半平面交的结果一定是一个凸集，事实上我们熟知的半平面交算法充分利用到了凸性，但首先我们来看一个不使用凸性的算法

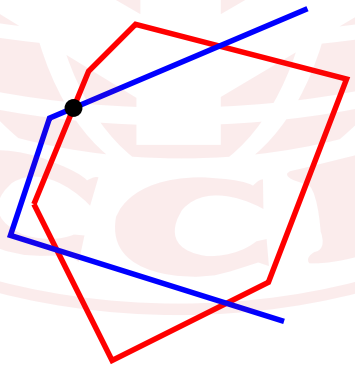
半平面交分治算法

Algorithm 1 半平面交分治算法

```
1:  function IntersectHalfPlanes(H = {h_1, h_2, ··· , h_n})
2:      if n=1 then
3:          return h_1
4:      end if
5:      H_1 ← {h_1, h_2, ··· , h_⌈n/2⌉}
6:      H_2 ← {h_⌈n/2⌉+1, h_⌈n/2⌉+2, ··· , h_n}
7:      C_1 ←IntersectHalfPlanes(H_1)
8:      C_2 ←IntersectHalfPlanes(H_2)
9:      return C_1 ∩ C_2
10: end function
```

凸集求交的转化

- 在进行分析之前我们需要对伪代码的最后一步 $C_1 \cap C_2$ 做一点说明，我们需要求解两个凸集的交，虽然这些凸集的边界除了线段外还可能是直线或射线，但稍作改动不难将这个问题转化为求一族线段交点的问题（注意到在这里转化时我们就放弃了凸性）



关于线段求交的进一步内容 *

- 使用扫描线可以在 $O(n\log n + k\log n)$ 的时间复杂度内计算出一族线段所有交点，这里 n 是线段的数量，k 是交点的总数，对于半平面交问题有 $k = O(n)$，所以这一步的时间复杂度可以写为 $O(n\log n)$
- 或许我们对这个传统的做法不太满意，但很快就能发现这个问题的下界是 $\Omega(n\log n + k)$ 的，达到下界的算法也是已知的[1]，不过这对我们没有什么帮助
- 值得一提的是，如果我们只是想计算交点数量而非具体的坐标，目前存在 $O(n^{\frac{4}{3}}(\log n)^{\frac{1}{3}})$ 的算法[2]，此处不再赘述

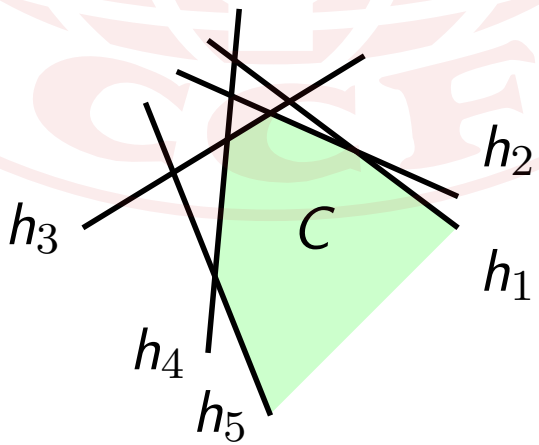
[1] https://www2.cs.sfu.ca/~binay/813.2011/Balaban.pdf
[2] https://www.cs.princeton.edu/~chazelle/pubs/CuttingHyperplanes.pdf

分治算法分析

- 根据之前的分析，可以得到分治算法关于时间的递推关系 $T(n) = 2T(\frac{n}{2}) + O(n \log n)$，解得 $T(n) = O(n \log^2 n)$
- 这个复杂度和我们常用的半平面交算法复杂度之间还有差距，原因也已多次提及：转化的过程中我们放弃了凸性
- 既然如此，我们就把目光放到凸性上，只需在 $O(n)$ 的时间复杂度内计算出平面上两个由半平面交成的凸集的交集，就可以将分治算法改进为 $O(n \log n)$

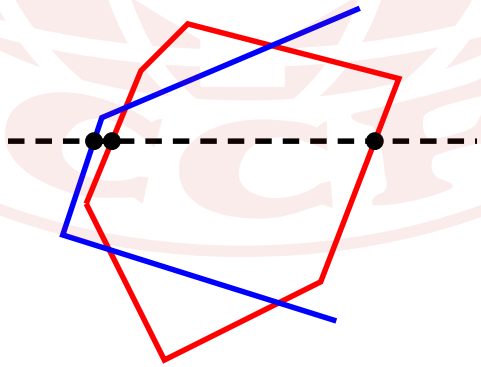
由半平面交成的凸集

- 我们专门指出只希望处理“由半平面交成的凸集”，除了为了避免其它奇怪形状的凸集（例如圆和椭圆等等）引起歧义外，也是想为这类凸集做出更准确的描述
- 例如下面这个例子，构成凸集 C 的半平面分成了左右两组，可以用两个有序列表 $L(C) = \{h_2, h_1\}$ 和 $R(C) = \{h_3, h_3, h_5\}$ 来表示它，对于水平的直线代表的半平面，我们可以规定朝下的属于 $L(C)$、朝上的属于 $R(C)$，规定的方式不影响我们接下来的操作



凸集求交的算法

- 仍然采用扫描线来求解两个凸集的交，不难发现任意时刻扫描线至多与凸集的边界有四个交点，因此不需要维护一个复杂的数据结构，只需记录相交的边界情况即可
- 每次扫描到一条新的边时，可以 $O(1)$ 找出可能在两个凸集的交上的边，这里需要一个较繁琐的分类讨论，但总之整个过程可以在 $O(n)$ 的时间复杂度内完成



分治算法总结

- 有了 $O(n)$ 对两个大小为 n 的凸集求交的算法，我们可以将分治算法关于时间的递推式改为 $T(n) = 2T(\frac{n}{2}) + O(n)$，解得 $T(n) = O(n \log n)$，这个复杂度足以使我们满意
- 在改进分治算法的过程中我们利用到了半平面的凸性，这是有一点几何趣味的做法，事实上半平面交的另一个做法（也就是算法竞赛中常用的做法）更加有趣：它几乎和平面凸包的求解算法是一样的，关于半平面交和凸包的关系我们会在稍后做进一步的讨论

线性规划问题

- 一个线性规划问题由一族约束条件和一个目标函数构成：

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,d}x_d \le b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,d}x_d \le b_2 \\ \cdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,d}x_d \le b_n \end{cases}$$

$$f(x_1, x_2, \cdots, x_d) = c_1x_1 + c_2x_2 + \cdots + c_dx_d$$

- 我们需要在所有变量满足约束条件的前提下，求解目标函数的最大值
- 类比半平面交，相同的是每个约束条件可以看作 $\mathbb{R}^d$ 中的一个半空间，但不同点在于半平面交求出的是所有满足约束条件的可行解，而线性规划问题需要找到这些可行解中使得目标函数取到最值的那一个

线性规划问题求解

- 对于一般的线性规划问题，我们有单纯形算法等熟知的方法可以处理，对于其中的一些特例例如网络流，我们有相应的算法可以对付
- 为了更靠近半平面交，这里我们只考虑一类特殊的线性规划问题：所有的约束条件都是双变量线性约束
- 在维数较低时，事实上传统的线性规划方法未必有很好的效率，接下来介绍的从计算几何中发展起来的方法往往会更加行之有效

解的几何特性

- 回到双变量的情形，目标函数 $f(x_1, x_2) = c_1x_1 + c_2x_2$ 可以看作向量 (c_1, c_2) 和 (x_1, x_2) 的内积，这说明在可行解区域内沿着 (c_1, c_2) 方向目标函数的值会不断增大，那么会分成四种不同情形：
  - 可行区域为空，此时原线性规划问题无解
  - 沿着 (c_1, c_2) 方向有唯一角点，此时原线性规划问题的解唯一，且这个解落在可行区域的顶点上
  - 沿着 (c_1, c_2) 方向到达一条垂直于该方向的可行区域的边，此时在这条边上都取到原线性规划问题的最优解，我们规定字典序较小的解更优，那么这种情况也会得到唯一解
  - 沿着 (c_1, c_2) 方向是无解的，此时目标函数的值理论上可以取到无穷大
- 上述第四种情况是比较棘手的，我们需要预先做一点处理

有界线性规划问题

- 不妨只考虑有界的情形，形式上只需加入两个新约束条件：

$$h_1' = \begin{cases} x_1 \le M & c_1 \ge 0 \\ -x_1 \le M & c_1 < 0 \end{cases} \qquad h_2' = \begin{cases} x_2 \le M & c_2 \ge 0 \\ -x_2 \le M & c_2 < 0 \end{cases}$$

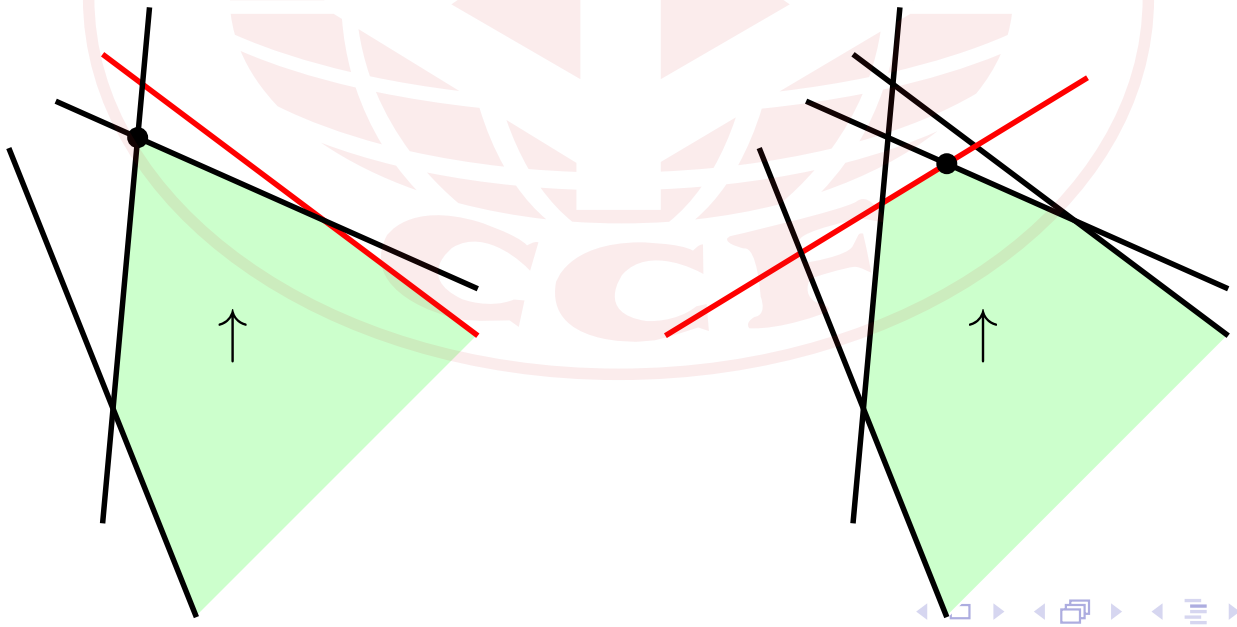
- 这里 M 是一个足够大的常数，使得其不致于影响最优解，那么加上这两个约束条件之后该线性规划问题如果有解就一定是有唯一解的有界问题了
- 接下来我们考虑增量法，即对于每一步我们都维护当前的唯一最优解，再逐步考虑加入下一个约束条件时解的变化情况

加入约束条件的变化

定理

设 $C_i = h'_1 \cap h'_2 \cap h_1 \cap \cdots \cap h_i$，$p_i$ 表示目标函数在 C_i 中的（唯一）最优点，则：

- 若 $p_i \in h_{i+1}$，则 $p_{i+1} = p_i$
- 若 $p_i \notin h_{i+1}$，则要么 $C_{i+1} = \varnothing$，要么 $p_{i+1} \in \partial h_{i+1}$



一维子问题的求解

- 有了这个定理，我们每次进行增量时要么能够判定无解，要么得到一个新的线性规划问题：在给定直线上找到满足之前所有约束的目标函数最大值点
- 这是一个一维的线性规划问题，显然最优解一定在边界上取到，在第 i 步我们可以花 $O(i)$ 的时间去检查这个问题是否有解，如果有还可以求得这个唯一的最优解，那么现在我们可以给出一个完整的算法了

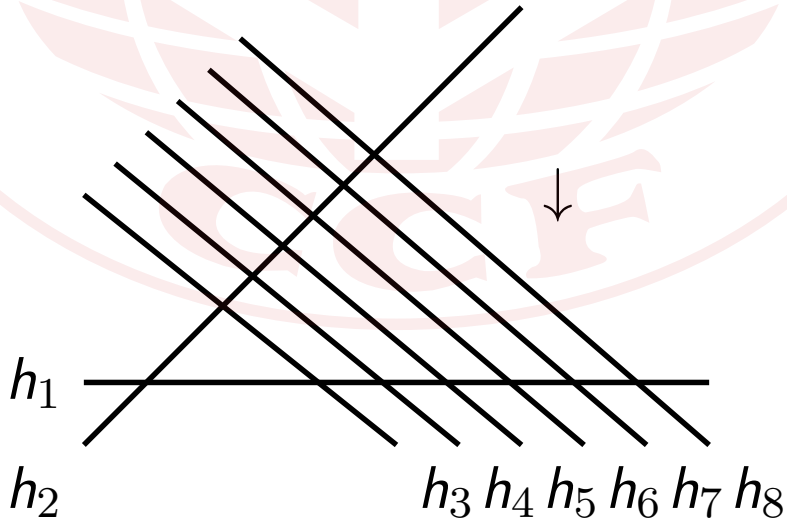
双变量有界线性规划增量算法

Algorithm 2 双变量有界线性规划增量算法

$p_0 \leftarrow C_0$ 的角点（这是唯一的）
for $i \leftarrow 1, 2, \cdots, n$ **do**
    if $p_{i-1} \in h_i$ **then**
        $p_i \leftarrow p_{i-1}$
    else
        $p_i \leftarrow 1\mathrm{DSolveLP}(h_i, C_{i-1}, p_{i-1})$
        if p_i 不存在 **then**
            报告原问题无解
        end if
    end if
end for

增量算法分析

- 由于每次求解一维问题的时间复杂度是 $O(i)$ 的，所以增量算法的总时间复杂度为 $\sum_{i=1}^{n} O(i) = O(n^2)$，事实上它也会在如下图这种情况中达到复杂度上界
- 如果希望得到更好的复杂度，我们看起来需要预先确定半平面的一个加入顺序，使得最优点发生变动的次数尽可能少，这是不能完全做到的，但我们可以做出一个有趣的尝试



随机增量法

- 我们无法决定最优的半平面加入顺序，但最坏情况看起来又是以一种特殊顺序加入才会形成的，这自然启发我们考虑以随机顺序加入半平面
- 设随机变量 $X_i = [p_{i-1} \notin h_i]$，则在随机情况下增量法的时间复杂度为 $\sum_{i=1}^{n} O(i)X_i$，由期望的线性性，随机增量法期望时间复杂度为 $E[\sum_{i=1}^{n} O(i)X_i] = \sum_{i=1}^{n} O(i)E[X_i]$
- 显然 $E[X_i]$ 恰好就是 $p_{i-1} \notin h_i$ 的概率，下面我们对其做进一步的分析

随机增量法分析

- 倒着考虑算法过程：当 p_i 已经计算完成时，如果去掉约束 h_i，最优点是否会发生变化？
- 如果发生变化，这就意味着 $p_{i-1} \notin h_i$，由于确定一个角点至少需要两个半平面，而 h_i 的选取是随机的，因此其确定 p_i 的概率至多只有 $\frac{2}{i}$，也即 $E[X_i] \leq \frac{2}{i}$
- 那么对于随机增量法，我们可以给出期望时间复杂度上界为 $\sum_{i=1}^{n} O(i)\frac{2}{i} = O(n)$，这个复杂度已经足够优秀

无界线性规划问题

- 回到分析双变量线性约束解的几何特性这里，当时我们提到对于某些线性规划问题目标函数的值理论上可能可以取到无穷大，为了规避这种情况我们引入了两个新约束条件，在有界的前提下得到了复杂度令人满意的解法
- 但这个处理有许多问题，首先 M 的取值非常模糊，看起来只能具体问题具体分析，其次在这种情况下我们也许会希望找到目标函数具体是如何增大的，具体而言，希望得到一条射线，使得沿着射线目标函数的值确实会无限增大

无界线性规划问题的判定

定理

一个线性规划问题是无界的，当且仅当存在方向 $\vec{d} = (d_1, d_2)$，使得 $c_1 d_1 + c_2 d_2 > 0$，且 $\forall h \in H$ 有 $\vec{d} \cdot \vec{\eta_h} \geq 0$，且线性规划问题 $H' = \{h \in H \mid \vec{d} \cdot \vec{\eta_h} = 0\}$ 总是可行的，这里 $\vec{\eta_h}$ 表示半平面 h 朝着可行区域方向的法向量

- 有了这个定理，我们可以构造出一个切实可行的判定算法来判断一个线性规划问题是不是无界的，若是，我们还可以尝试构造出这个方向 $\vec{d}$ 来，它具体地描述出了这个无界性

无界线性规划问题判定转化

- 现在看起来有点奇怪，我们在双变量线性规划问题中引入了新的双变量线性规划问题，但其实方向选取的一般性可以解决这个问题：总是可以旋转坐标轴使得 $c_1 = 0$，这样可以将方向 $\vec{d} = (d_1, d_2)$ 规范化为 $\vec{d} = (d_x, 1)$ 的形式
- 设 $\vec{\eta_h} = (\eta_{h,x}, \eta_{h,y})$，则 $\vec{d} \cdot \vec{\eta_h} \geq 0$ 可以写作 $d_x \eta_{h,x} \geq -\eta_{h,y}$，这就变成了一个一维线性规划问题，记之为 $\bar{H}$
- 在 $\bar{H}$ 可行的情况下，对一个可行解 $\vec{d}^*$ 我们还需要验证 $H' = \{h \in H \mid d_x^* \eta_{h,x} = -\eta_{h,y}\}$ 是可行的，注意到 H' 中的半平面边界一定是平行的，所以这也可以转化为一个一维线性规划问题 $\bar{H'}$

无界线性规划问题判定算法

- 我们整理一下现在有的结果：
  - 为了找到一个方向 $\vec{d}$，我们需要求解一维线性规划问题 $\bar{H}$
  - 在 $\bar{H}$ 有解 $\vec{d}^*$ 的情况下，我们需要判定其对应的一维线性规划问题 $\bar{H}'$ 可行
  - 若 $\bar{H}'$ 可行，则 H 是无界的，沿着 $\bar{H}$ 的解方向 $\vec{d}^*$ 目标函数会无限增大
  - 若 $\bar{H}'$ 不可行，事实上可以推知 H' 不可行，则原问题 H 是不可行的
  - 若 $\bar{H}$ 无解，说明 H 是一个有界线性规划问题
- 看起来在最后一种情况下我们会回到有界线性规划问题的求解上，但 M 的取值问题仍然没有解决，不过没有关系，在判定 $\bar{H}$ 无解时一定找到了两个互相矛盾的法向量，它们对应的半平面就可以否决 H 的无界性，所以直接用它们充当 h'_1 和 h'_2，即可改造出无需 M 的有界线性规划问题算法

高维线性规划问题

- 设 $G = \{g_1, g_2, \cdots, g_n\}$ 为一族 d 变量线性约束，也即每个 g_i 对应 $\mathbb{R}^d$ 中的一个半空间
- 考虑沿用增量法，不难得到相同的结论：每次加入一个新的半空间时，（唯一）最优点要么不变，要么一定出现在新半空间的边界上
- 与二维的情形不同，例如在三维时新半空间的边界与之前的可行区域的交集是一个多边形区域，我们可以简单利用投影将其转化为一个标准的二维线性规划问题，对于更高维的情形也可以类似递归计算
- 对于无界性的判别其实也与二维的情形类似，只不过在 d 维时 $\bar{H}$ 变成了 $d-1$ 维，我们也可以递归进行计算

高维情形的随机增量法

- 事实上我们能得到的结论看起来非常强

定理

对任意固定的维数 d，采用随机增量法都能在期望 $O(n)$ 的时间复杂度内求解一个 d 维线性规划问题

- 可惜在接下来的分析中我们将会看到，这个算法有一个依赖于维数 d 的常数因子，且这个因子随着维数 d 的增大增长得非常快
- 但这并不代表随机增量法没有用武之地，在维数比较低的情形下它比求解线性规划的通用算法通常表现得更为优秀

高维随机增量法分析

- 设随机增量法解决 d 维线性规划问题的常数因子为 T_d
- 首先进行无界性的判别，注意到我们最终需要求出可以替代 M 的 d 个半空间，这并不能够一次获得（虽然在二维的情形下确实是一次获得的），最坏情况下需要做 $O(d)$ 次，每次需要解常数个 $d-1$ 维线性规划问题，这部分耗时 $T_{d-1}O(nd)$
- 在增量法第 i 步可能需要递归求解 $i-1$ 个 $d-1$ 维线性规划问题，这部分期望耗时 $\sum_{i=1}^{n}(O(id)+(i-1)T_{d-1})E[X_i]$，易知这里的 $E[X_i]$ 应为 $\frac{d}{i-d}$
- 将总耗时用 $T_dO(n)$ 控制，可得 $T_d=O(T_{d-1}d)$，可以取一个正常数 c 使得 $T_d=O(c^d\cdot d!)$，可见在维数较高时这个方法难以承受

高维情形的半空间交 *

- 实际上已经可以发现，双变量线性规划问题是比半平面交更“简单”的，因为前者只需要求出可行区域内的一个点（即使有目标函数最优化的附加限制），而后者需要求出整个可行区域
- 讨论完半平面交和凸包的关系后，我们会得到 $O(n \log n)$ 的算法求解三维情形下的半空间交，但要进一步推广到高维则有本质困难，因为 n 个 d 维半空间交成的凸多胞体的大小是 $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ 的，可以看出二维和三维确实有特殊性（此时 $\lfloor \frac{d}{2} \rfloor = 1$），所以我们才能得到复杂度较为满意的算法

最小圆覆盖

- 考虑如下问题：对平面上给定的 n 个点 $p_1, p_2, \cdots, p_n$，试求出一个半径最小的圆，使得这些点都被该圆包含
- 这是计算几何中的一个经典问题，处理时用到了随机增量法，下面我们对该问题进行一个回顾

最小圆覆盖的性质

引理

设 $P_i = \{p_1, p_2, \cdots, p_i\}$，$D_i$ 表示点集 P_i 对应的半径最小的覆盖圆，则：

- 若 $p_{i+1} \in D_i$，则 $D_{i+1} = D_i$
- 若 $p_{i+1} \notin D_i$，则 p_{i+1} 一定在 D_{i+1} 的边界上

- 可以看出，这个引理和线性规划的增量算法中用到的定理几乎一模一样，实际上具有类似性质的问题都可以考虑采用增量法，并通过随机打乱增量顺序来规避可能的最坏情况，从而得到一个优秀的复杂度

最小圆覆盖算法

Algorithm 3 最小圆覆盖算法

```
RandomShuffule(p_1, p_2, ..., p_n)
D_2 ← p_1, p_2 确定的最小圆（这是唯一的）
for i ← 3, 4, ..., n do
    if p_i ∈ D_{i-1} then
        D_i ← D_{i-1}
    else
        D_i ←MinimumDisk1(p_1, p_2, ..., p_{i-1}; p_i)
    end if
end for
```

最小圆覆盖算法

Algorithm 4 最小圆覆盖算法 MinimumDisk1 函数

```
1:  function MinimumDisk1(p_1, p_2, ··· , p_n; q)
2:      E_1 ← p_1, q 确定的最小圆（这是唯一的）
3:      for j ← 2, 3, ··· , n do
4:          if p_j ∈ E_{j−1} then
5:              E_j ← E_{j−1}
6:          else
7:              E_j ←MinimumDisk2(p_1, p_2, ··· , p_{j−1}; p_j, q)
8:          end if
9:      end for
10:     return E_n
11: end function
```

最小圆覆盖算法

Algorithm 5 最小圆覆盖算法 MinimumDisk2 函数

```
1:  function MinimumDisk2(p_1, p_2, ··· , p_n; q_1, q_2)
2:      F_0 ← q_1, q_2 确定的最小圆（这是唯一的）
3:      for k ← 1, 2, ··· , n do
4:          if p_k ∈ F_{k−1} then
5:              F_k ← F_{k−1}
6:          else
7:              F_k ← p_k, q_1, q_2 确定的唯一圆
8:          end if
9:      end for
10:     return F_n
11: end function
```

最小圆覆盖算法的正确性

- 在这个经典的算法中用到了非常丰富的几何性质（包括最开始的引理），我们可以总结并推广如下

定理

设 $P = \{p_1, p_2, \cdots, p_n\}$ 为平面上的一个点集，Q 是另一个点集且 $P \cap Q = \varnothing$，则：

- 若存在圆 D 覆盖 P 中的所有点满足 $\forall q \in Q, q \in \partial D$，则存在唯一半径最小的圆 D 满足该条件，记之为 $D_{P,Q}$
- 若 $p_n \in D_{P\setminus\{p_n\},Q}$，则 $D_{P,Q} = D_{P\setminus\{p_n\},Q}$
- 若 $p_n \notin D_{P\setminus\{p_n\},Q}$，则 $D_{P,Q} = D_{P\setminus\{p_n\},Q\cup\{p_n\}}$

最小圆覆盖算法分析

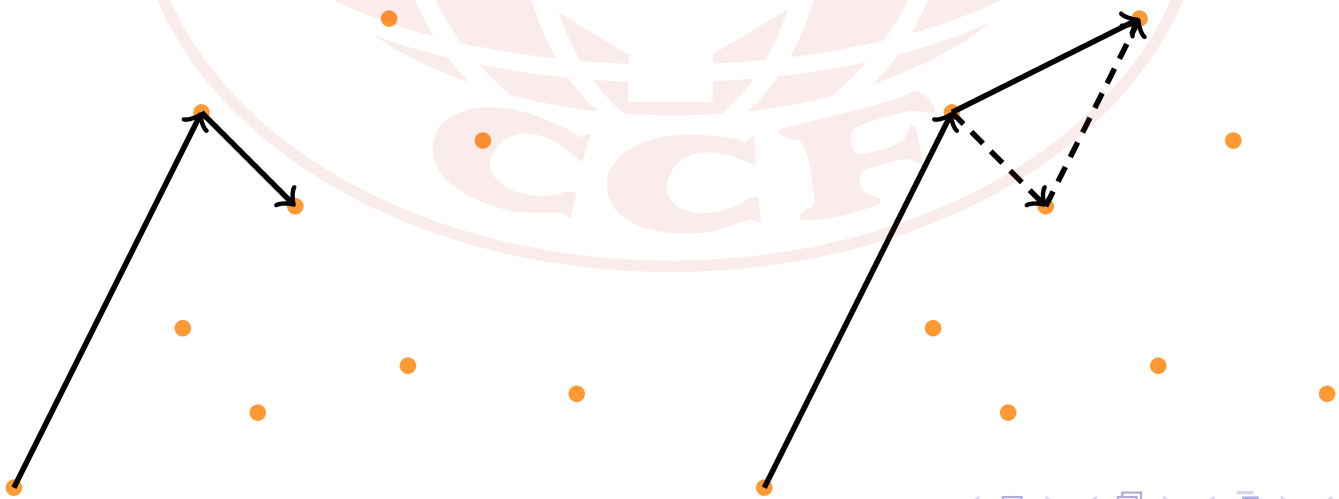
- 接下来我们来考虑一下最小圆覆盖随机增量法的期望时间复杂度，首先对于 MinimumDisk2 函数，显然单次调用的时间复杂度为 $O(n)$
- 对于 MinimumDisk1 函数，除了调用 MinimumDisk2 函数外的部分总复杂度为 $O(n)$，单次调用 MinimumDisk2 函数的时间复杂度为 $O(j)$，考虑调用的概率，可以发现当且仅当 p_j 是确定最小覆盖圆的至少三个点之一时去掉 p_j 才可能使得答案发生变化，这个概率（也就是期望）不超过 $\frac{2}{j}$，所以单次调用 MinimumDisk1 函数的期望复杂度为 $O(n)$
- 类似的，可以证明最小圆覆盖主程序部分的期望复杂度为 $O(n)$，至此整个算法分析完毕

凸包问题

- 凸包：设 S 是平面上的一个点集，则 S 的凸包 $\mathcal{CH}(S)$ 定义为平面上包含 S 的最小凸集，也即包含 S 的所有凸集的交，在非退化的情况下，$\mathcal{CH}(S)$ 一定是一个多边形
- 二维凸包的求解是比半平面交更加经典的问题，在这一节中我们首先对这个算法进行简单回顾，然后主要着眼于三维凸包的求解和分析，并引出高维凸包问题的一些困难，最后我们讨论凸包和半平面交之间的关系作为结尾

二维凸包

- 求解 n 个点的二维凸包的复杂度下界为 $\Omega(n \log n)$，有很多已知的算法能够达到这个复杂度，这里我们回顾一下最经典的“卷包裹”法
- 简言之，我们将凸包分为上凸包和下凸包两个部分来求解，在每个部分中首先对所有点按照 x 坐标排序，之后沿着点从左向右走，保证每次走到的点相对之前的点来说都是相同的转弯方向即可



二维凸包算法

Algorithm 6 二维凸包算法（上凸包部分）

1: 将 $S = \{p_1, p_2, \cdots, p_n\}$ 按照 x 坐标排序
2: $\mathcal{CH}(S)_{upper} \leftarrow [p_1, p_2]$
3: **for** $i \leftarrow 3, 4, \cdots, n$ **do**
4: $\quad \mathcal{CH}(S)_{upper} \leftarrow \mathcal{CH}(S)_{upper} + [p_i]$
5: $\quad$ **while** $|\mathcal{CH}(S)_{upper}| \geq 3$ 且 $\mathcal{CH}(S)_{upper}$ 中最后三个元素不构成右转 **do**
6: $\quad\quad$ 去掉 $\mathcal{CH}(S)_{upper}$ 中倒数第二个元素
7: $\quad$ **end while**
8: **end for**

从二维凸包到三维凸包

- 在我们开始具体讨论三维凸包前，首先注意到一个问题：三维凸包可能有多大?
- 这个问题和线段求交里的问题是类似的，在后者中我们如果想求出所有交点，则付出的复杂度至少是交点个数级别的，但如果只希望知道交点个数就有稍好一点的复杂度
- 对于二维凸包，很容易知道凸包上的点数和边数都不会超过给定的点集大小，三维凸包稍微麻烦一点，不过我们也能得到较好的性质

三维凸包的大小

定理

对一个 n 个顶点的凸多面体，其包含的边数不会超过 $3n-6$ 条，包含的面数不会超过 $2n-4$ 张

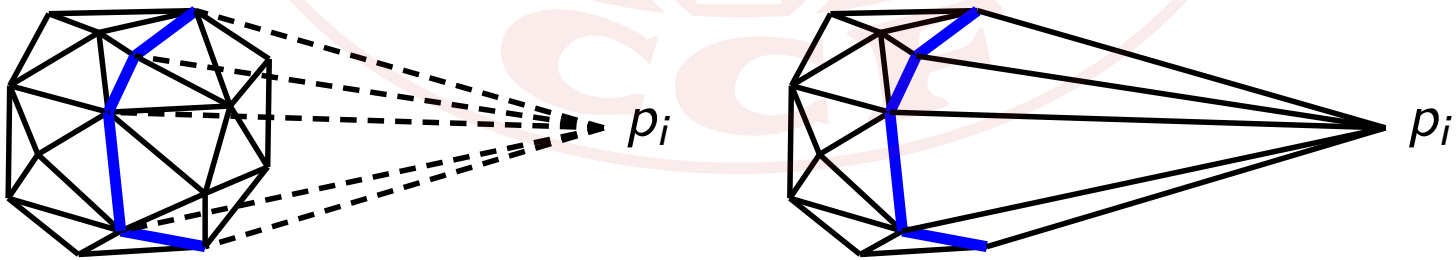
- 显然在非退化情形下，n 个点的三维凸包是一个凸多面体，所以它的大小不会超过 $O(n)$
- 对于退化的情况，至少 n 个点都是共面的，我们可以投影之后用二维凸包的算法来解决，接下来的讨论中不再考虑退化的情况

三维凸包随机增量法

- 设 $P = \{p_1, p_2, \cdots, p_n\}$ 是三维空间中的 n 个点，下面我们使用随机增量法来构造它们的三维凸包 $\mathcal{CH}(P)$，首先我们描述整个算法的过程
- 作为增量法的开始，我们要先找到 4 个不共面的点，由它们构成一个四面体作为最开始的凸包，因为非退化，所以可以先任意找 2 个不同点，再任意找一个与这两点不共线的第 3 点，最后任意找一个与这三点不共线的第 4 点即可
- 将 $p_5, \cdots, p_n$ 的顺序随机打乱，设 $P_i = \{p_1, p_2, \cdots, p_i\}$，对 $\forall i \geq 5$，如果 p_i 落在 $\mathcal{CH}(P_{i-1})$ 的内部或边界上则可以直接令 $\mathcal{CH}(P_i) = \mathcal{CH}(P_{i-1})$，否则我们考虑如何将其加入凸包中得到 $\mathcal{CH}(P_i)$

多面体的可见性

- 如下图所示，当 P_i 不在 $\mathcal{CH}(P_{i-1})$ 内时，它是能够“看到” $\mathcal{CH}(P_{i-1})$ 的一些面和一条地平线（也即图中的蓝线）的，那么我们需要做的操作就是删去这些可见的面，并沿着地平线和 p_i 组成一些新的面加入 $\mathcal{CH}(P_i)$ 中
- 注意边界情况：刚好和 p_i 的视线“擦肩而过”的这些面不会被删去，也不应该添加新的面，而是应该在原有的面中插入一个新的点



三维凸包的增量法

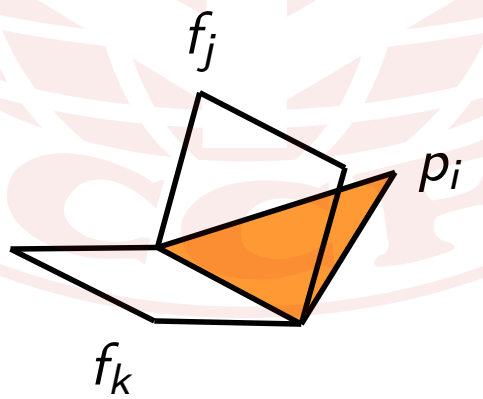
- 实际上现在已经可以给出一个三维凸包的求解方法：我们只需要每次暴力检查哪些面会被 p_i 看到即可，这等价于 p_i 和对应面的外法向量在同侧，对于刚好擦过的情况则表示连线和法向量垂直
- 因为凸包上的面数在第 i 步之后是 $O(i)$ 的，所以该做法的时间复杂度为 $O(n^2)$
- 这个方法就称作三维凸包的增量法，奇怪的是这个做法看起来和增量的顺序没什么关系，它基本都会达到最坏情况，实际上我们应该考虑用更好的方法来维护这个可见性，随机增量在那里起到作用

Conflict 图

- 对 $\mathcal{CH}(P_i)$，我们构造所谓的 Conflict 图来维护可见性，这是一张二分图，两个点集分别代表点 $p_{i+1}, p_{i+2}, \cdots, p_n$ 和 $\mathcal{CH}(P_i)$ 上的面 $f_1, f_2, \cdots, f_m$，如果点 p_j 能看到面 f_k，则两个点之间有一条边
- Conflict 的图的初始化和删去点与面时要进行的操作十分简单，我们主要关注产生新的面时 Conflict 图如何变化

产生新面的变化

- 如果增量 p_i 时一个面只是新插入了这个点，则其对后续点的可见性不会有任何变化
- 考虑产生一个新的三角形面的情况，如下图所示，不难发现新的面的可见点只需要检查 f_j, f_k 的可见点列表——进行验证即可



三维凸包随机增量法分析

- 随机增量法的时间复杂度证明较为复杂，其梗概可以用两个定理进行描述

定理

随机增量法中生成的新面个数期望不超过 $6n-20$

定理

设每次生成新面时地平线上一条边 e 的两个相邻面中可见点的并的集合为 $P(e)$，则随机增量法的时间复杂度瓶颈为 $O(\sum|P(e)|)$，且可以进一步证明这是 $O(n\log n)$ 的

凸包与半平面交的对偶关系

- 现在我们着眼于凸包与半平面交的关系，考虑将点和直线做一个对偶，一个点 p 对偶后得到的直线记为 p^*，一条直线 l 对偶后得到的直线记为 l^*，且 $l^* \in p^*$ 当且仅当 $p \in l$，l^* 在 p^* 的上方当且仅当 p 在 l 的上方
- 为一组下半平面计算交集的问题，在此对偶意义下可以转化为计算某个上凸包的问题，同理为一组上半平面计算交集的问题可以转化为计算某个下凸包的问题
- 如果熟知算法竞赛中经典的半平面交求解方法，则不难理解这个对偶关系，事实上三维凸包和三维空间的半空间交也是这样的一组对偶，因此三维凸包的求解方法可以用于求解三维空间的半空间交问题

高维凸包的求解 *

- 正如前面所说，n 个顶点的 d 维凸多胞体的大小是 $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$ 的，因此对于维数 ≥ 4 的凸包问题不太能有一个满意的复杂度
- 目前最优秀高维凸包算法（以及高维半空间交算法）可以做到 $O(n \log k + (nk)^{1-\frac{1}{\lfloor \frac{d}{2} \rfloor + 1}} \log^{o(1)} n)$ 的时间复杂度[3]，显然这对我们并没有什么意义:)

[3] https://www.semanticscholar.org/paper/Output-sensitive-results-on-convex-hulls%2C-extreme-Chan/5ac22bef1ed172488ba60fd4b158ee4220cdc69e?p2df