研究生算法课课堂笔记

上课日期：2014-10-13　　第(3)节课

组长：范非凡　　组员：李晨

组员：李肯　　组员：高晓旸

注意：请提交 Word 格式文档。

Greedy Clustering Algorithm

概念 1： Single-linkage k-clustering Algorithm

(1) 从图上点集 U，联通到 n 个簇

(2) 找到最接近的两个对象，每个对象都是不同的一对集群，和他们之间加一条边。

(3) 重复 n–k 次直到恰有 k 个簇。

概念 2： 聚类问题两个标准：

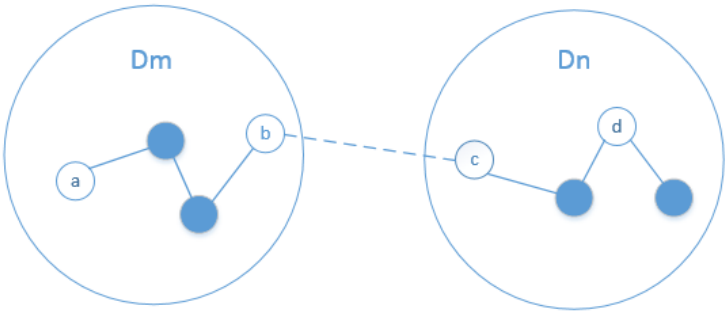
(1) 不同 cluster 之间的最小距离尽可能的大

(2) 同一 cluster 内两个点之间的最大距离尽可能的小

问题 1： 对于标准(1)有效的贪心聚类算法

1. 对于标准(1)，可以采用 Kruskal 算法，将反复地合并聚类问题转化为计算一个最小生成树并删除最贵的边的问题：即在添加最后的 k-1 条边时算法终止，相当于删除了 k-1 条最贵的边。
2. 证明对于 k-clustering 问题，采用 Kruskal 算法得到的聚类结果的最大间隔不比最优解小，即其划分的 k 个聚类最优。

   思路： 设 C 为采用 Kruskal 算法得到的聚类$C_1, C_2, \cdots, C_k$，则 C 的间隔为 Kruskal 算法得到最小生成树中第$k-1$条最贵的边的长度d^*。设 D 为最优解$D_1, D_2, \cdots, D_k$。只需证明其间隔至多为d^*。



考虑到聚类 C 和 D 不同，故一定存在某个C_i不是 D 中任意个集合的子集（否则 C=D，因为$\{v \mid v \in C_i, 1 \le i \le k\} \subseteq \{v \mid v \in D_i, 1 \le i \le k\}$且两个集合相同），假设点 a 和点 d 都

属于聚类C_i，且$a \in D_m, d \in D_n$ 那么点 a 与点 d 路径上的所有边都是在 Kruskal 算法终止之前添加的，即路径上每条边的长度$\leq d^*$。令 b 和 c 为 a-d 路径上聚类D_m和聚类D_n将路径断开两边的点，则点 b 和点 c 的间隔$\leq d^*$，从而最优解 D 的间隔至多为$dis(b,c) \leq d^*$，从而 Kruskal 贪心策略得到的聚类结果 C 间隔不比最优解 D 差。

问题 2： 对于标准(2)可能的有效的聚类算法

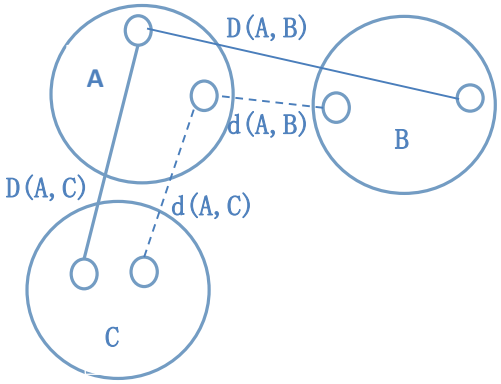
同学思路 1：

n 个顶点为 n 个集合，离得最近两点合并，变为 n-1 个集合。之后两个集合间的距离，定义为最远的点对，合并时取集合间距离最小的合并

反例说明：

由上述策略会导致出现聚类内的最大距离比聚类间的最小距离要大的情况，显然不符合聚类的要求（即同一聚类内点之间的距离尽可能小，不同聚类间的距离尽可能的大，并且聚类内的最大距离应比聚类间的最小距离要小）

如下图所示，假设当前有聚类 A，B，C，那么按照上述策略选取两个集合聚类时，A，C 距离认为较远的 D(A,C),A,B 间距离认为较远的 D（A，B），由于 D(A,B)>D(A,C)，则该策略将集合 A 和集合 C 聚类在一起。但集合 A，B 实际距离为最近的两点间的距离 d(A,B),集合 A,C 实际距离为最近的两点间的距离 d(A,C)，且 d(A,B)<D(A,C)，而合并 A，C 后的聚类内两点间最大距离不小于 D(A,C)，但其与 B 的两点间的最小距离不大于 d(A,B)，这样显然是矛盾的（聚类内的距离大于聚类间的距离）



同学思路 2：

完全图删边，从最大的删起，直到产生 k 个集合，再多删一条边就变为 k+1 个集合

反例说明：

按照上述策略得到的实际为满足标准（1）的聚类方法，即不同聚类间的最小距离尽可能的大而不是同一聚类间的最大距离尽可能的小。可以通过 Kruskal 算法生成最小生成树然后删去最重的 k-1 条边得到符合标准（1）的聚类过程来理解，按照上述完全图删边的策略相当于采用 Reverse-Delete Algorithm 得到最小生成树 MST 后继续删除前 k-1 条最重的边，即使聚类内的最大距离尽可能的小。

概念 3： P 问题，NP 问题，NPC 问题，规约（补充）

P 问题：能够在多项式时间内解决的**决策**问题

NP 问题：多项式时间内能够**验证**的问题

验证：给定一个问题的实例(Instance)，验证这个实例为这个问题的正确答案。

NPC 问题：目前不能用多项式时间解决的问题，但是还不能证明这个问题不能用多项式解决，定义为：

A problem Π is $\mathcal{NP}$-completeness if

1. $\Pi \in \mathcal{NP}$;

2. $\forall \Pi' \in \mathcal{NP}, \Pi' \leq_P \Pi$.

(1) P 问题同时也是 NP 问题

(2) P 问题是不是等于 NP?

目前答案是“不是”，但还不能证明，如果能够证明一类 NPC 问题是 P 问题，则 NP=P

规约（Reduction）：如果我们要证明一个问题 A 是 NPC 问题，则只需要首先证明他是 NP 问题，然后只要找一个你所知道的 NPC 问题规约到 A 即可。

如果 A 到 B 规约成功，则说明：B 至少比 A 要难，即只要有一个解决 B 的黑盒子算法，则就能解决 A 问题。

证明 B 为 NPC 问题的规约过程：

(1)证明 B 是 NP 问题。

(2)知道一个已知的 NPC 问题 A。

(3)给出一个规约过程，并证明此规约过程是多项式时间的。

对于 A 中的任意一个实例：

(4)如果 A 有一个真的实例，则 B 也有一个真的实例。

(5)如果 B 有一个真的实例，则 A 也有一个真的实例。

问题 3：证明标准(2)的问题为 NPC 问题

引入 1：k-着色问题（NPC）

图的 k-着色问题——若一个图最少需要 k 种颜色才能使图中任意相邻的 2 个顶点着不同颜色，则称这个数 k 为该图的色数。求一个图的最小色数 k 的问题称为 k-着色问题。

思路：将 k-着色问题规约到标准(2)的问题

图着色问题即将所有顶点分为 K 个颜色组，每个组形成一个独立集，即其中没有相邻的顶点。根据标准(2)，假设同一集合中任意两点的最大距离为 d，我们可以简单的定义不相连的点距离为 d-1，相连的点距离定义为 d+1，那么图的 k 着色方案就对应了聚类问题的一个合法解，其中每个聚类中包含的点就是原图中着同一种颜色的顶点集合。当然这是 $k \geq 3$ 时的情况。

扩展 1：K=2 时的情况

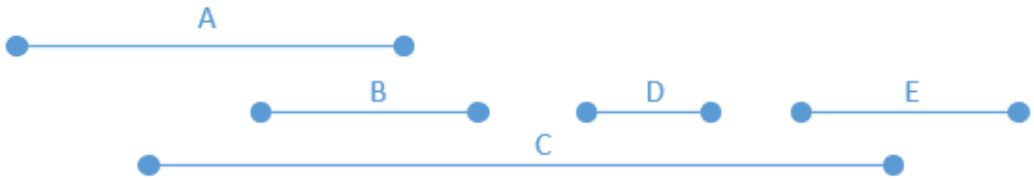
上面我们考虑了 $k \geq 3$ 的情况，若 k=2，看还能否找到一种办法将原图分为两个集合，满足同一集合内部点距小于等于 d。我们可以采用这样的办法：两点距离大于 d 则相连，小于等于 d 则不相连，然后判断所形成的图是否是个二部图。

问题 4：第四章习题 15（指导委员会的轮班调度问题）

思路 1：

将轮班表按照结束时间的顺序排列，（1）第一个区间必须有指导委员会同学的轮班时间与其重叠，故选取与第一个区间相重叠的开始时间最晚的轮班时间的同学，然后去掉该区间能覆盖（有重叠）的其他时间区间（2）选取顺序排列中未与上一个指导委员会同学的轮班时间重叠的时间区间作为第一个区间，重复步骤（1）直至所有轮班表区间遍历完全。

存在反例如：



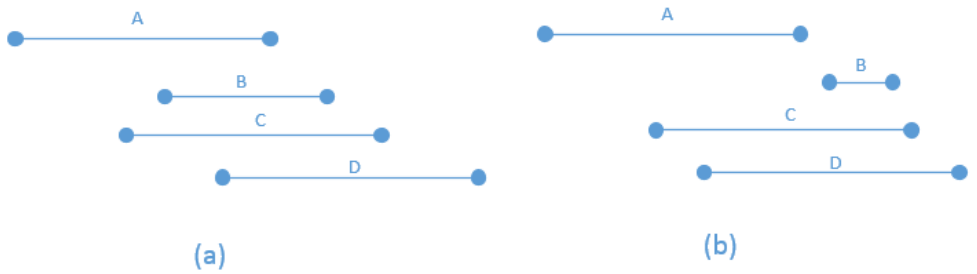
假设 A 为第一个时间区间，B 和 C 为与 A 有重叠的时间区间，则按照上述方法会选择开始时间最晚的 B，这样 D 和 E 区间仍需要指导委员会同学来覆盖，而如果选择 C 的话，只需要一个区间就能与所有轮班时间重叠。

思路 2：

将轮班表按照结束时间的顺序排列，在思路 1 的方法上改进，选择与第一个区间有重叠的结束时间最晚的轮班区间的同学。

证明：只需要说明最优解能覆盖的区间，采用结束时间最晚的方案也都能覆盖即可。

如下图所示：假设 A 为算法每次循环中的第一个区间，C 为最优解，B 为最优解能覆盖的区间，D 为与 A 相重叠的结束时间最晚的区间。则 B 有两种情况，一种与 A 相交(a)，一种 A 不相交(b)（按照结束时间排序说明 B 结束时间比 A 晚，不会出现内含的情况）。对于情况(a)，考虑到 A 与 B 相交，而 D 与 A 相交，故 B 和 D 至少在 A 结束的时间点有重叠；对于情况(b)，考虑到 C 能覆盖 B（或与 B 区间有重合），而 D 区间的结束时间比 C 区间晚，故 D 一定能与 B 有重叠。因此最优解 C 能覆盖的区间，结束时间最晚的 D 一定能覆盖。



问题 5：圆柱体装箱（往年期末考试题）

题目：N 个圆柱体物体要被装到一些规定固定的桶里面，所有的桶的长度都是 L，而所有的物理的高度都小于等于 L。桶的内部直径和物体的直径一样，一个桶只允许放 1 个和 2 个物

体，桶里的物体高度和不能超过桶的高度。请问至少需要多少个桶才能装下所有物体？

思路：先将所有圆柱体按照高度从低到高排列，然后采用贪心策略，每次选最高的和最低的组合放入桶中，如果高度和超过 L，则只放最高的，然后选取下一个最高的和最低的尝试放入一个桶中，依次类推。

时间复杂度分析：（1）圆柱体按照从高到低排列：O(nlogn) （2）尝试放最高和最低的圆柱体，最优情况为 O(N/2), 最差情况为 O(N)，算法整体复杂度为 O(nlogn+n)=O(nlogn)，伪代码如下：

```
Input: N 个圆柱体的高度数组 A[N]
Output:  最少需要的圆桶数组 M
MinCount = 0
QuickSort(A, N)
Min = 0
Max = N – 1
While Min <= Max
    If (A[Min] + A[Max]) < L then
        Min += 1
        Max -= 1
        MinCount += 1
    Else
        Max -= 1
        MinCount += 1
    Endif
EndWhile
Return MinCount
```

证明：贪心策略得到的装桶方法最优，即使用的桶的数目最少。

设 N 的圆柱体按照高度从高到低顺序为$\{\mathrm{L}_1,\mathrm{L}_2,\mathrm{L}_3,\cdots,\mathrm{L}_n\}$，贪心策略的装桶方案为$U=\{\mathrm{S}_1,S_2,S_3,\cdots,S_u\}$，最优解的装桶方案为$V=\{T_1,T_2,T_3,\cdots,T_v\}$，其中 U 和 V 按照每个桶中最大圆柱体的高度从高到低排序，如S_i比S_{i+1}中最高桶要高。那么可以证明对于每个T_i替换为S_i依然是最优解：

假设 U 和 V 前 k-1 项相同，那么只需证明将S_k替换T_k依然是最优解即可。

(1) 当 k==1 时，$S_1=(L_1,L_n)$或(L_1)

若$S_1=(L_1,L_n)$则$T_1=\left(L_1,L^*\right)$或$\left(L_1\right)$，其中$L_n\le L^*$，显然用S_1替换T_1依然是最优解（考虑到$L_1+L_n\le L$，$L_1+L^*\le L$，且L_1最大，故交换 V 中L_1和L^*的位置，依然是最优解）

若$S_1=(L_1)$则根据贪心策略得知$T_1=(L_1)$（没有圆柱体能和L_1放在一个桶中）

故用S_1替换T_1，V依然是最优解

(2) 假设 U 和 V 前 k-1 项（k≥2）相同，那么对于第 k 项：

假设$S_k=(L_i)$或$(L_i,L_j),i<j$（显然有$L_i>L_j$）

若$S_k=(L_i)$，根据贪心策略可知，此时剩下的圆柱体都不能和L_i同时放在一个桶中（高度和不小于 L），故$T_k=(L_i)$（因为 U，V 均按照每个桶中最高圆柱体的从高到低的顺序排列，当前L_i最高）

若$S_k=(L_i,L_j)$，则$T_k=(L_i,L_r)$ 或(L_i)；若$T_k=(L_i)$，显然用S_k替换T_k能得到更好的结果；若$T_k=(L_i,L_r)$，则$L_j\le L_r$（根据贪心策略可知，S_k选择的是当前最高的圆柱体和当前最低的圆柱体），由$L_i+L_j<L$且$L_i+L_r<L$可知，对于最优方案 V 来说，交换L_j和L_r的位置，依然是最优解（使用桶数不变）即S_k替换T_k依然是最优解

综合(1)(2)可知，使用贪心策略得到的方案使用桶的数目和最优解相同即使用桶的数目最少。