第 5 章 回溯与分支限界

5.1 内 容 提 要

1. 基本概念

解空间 搜索问题的解所在的集合，又称搜索空间. 解空间通常可以安排成树结构，常用解空间有子集树、排列树等.

回溯算法 遵照某种策略搜索解空间从而找出解的过程. 常用的搜索策略有深度优先、宽度优先、规则优先等.

分支限界算法 回溯算法的一种特例. 在回溯算法运行过程中，为加快算法的速度，尽可能多地在解空间中进行剪枝，设立新的约束条件——代价函数和界. 当代价函数值小于(对于最大化问题)界时即进行剪枝，这种回溯算法称为分支限界算法.

代价函数 以搜索树的任何的结点 v 开始搜索以 v 为根的子树，在这个范围可能得到可行解，代价函数 $f(v)$ 表示在该子树中可行解的目标函数值的一个上界(对于最大化问题).

界函数 函数在搜索树中某结点的函数值是算法搜索到该结点时，已经得到的可行解的目标函数的最大值.

多米诺性质：$P(x_1,x_2,\cdots,x_{i+1})$为真蕴涵 $P(x_1,x_2,\cdots,x_i)$为真.

2. 算法

***n* 皇后问题的回溯算法** n 皇后问题是指，在 $n\times n$ 的棋盘上放置 n 个皇后，使得任何两个皇后不能相互攻击，即棋盘的同一行、同一列、主对角线的平行线、副对角线的平行线上都不能有两个及以上的皇后，给出所有的放置方法. 该问题的回溯算法的解空间是一个树高为 n 的 n 叉完全正则树，树中第 k 层的边表示棋盘中第 k 行的皇后位置，搜索策略是深度优先，在每个结点处检查是否有皇后冲突，若有则回溯，若无则向下分支. 若以检查两皇后是否冲突作为基本运算，该算法的最坏情形复杂性为 $O(3n\times 2n^n)=O(n^{n+1})$.

0-1 背包问题的回溯算法 0-1 背包问题的描述详见主教材第 3 章. 该问题的回溯算法的解空间是一个树高为 n 的 2 叉完全正则树，即子集树，树的第 k 层中，关联每个顶点的两条边分别表示第 k 个物品放入或不放入背包中(分别标记为 1 和 0)，搜索策略是深度优先，在每个结点处检查放入背包物品重量之和是否大于背包的承载量，若是则回溯，否则向下分支. 若以数的大小比较作为基本运算，则该算法的最坏情形复杂性为 $O(2^n)$.

货郎问题(TSP)的回溯算法 货郎问题是指,某售货员要到若干城市去推销商品,各城市之间的距离为已知,他要选定一条从驻地出发经过所有城市最后回到驻地的一条周游路线,使得总的路程最短. 其数学模型是:已知一个带权图完全图(结点代表城市,边代表城市之间的道路,权代表城市之间的距离,如两个城市之间无直接连接的道路,设其权为∞,所以权为正数或无穷),求权和最小的一条哈密顿回路. 该问题的回溯算法的解空间是一个树高为 n 的排列树,树中关联根结点的第1层的边只有一条,表示该售货员所在的城市编号,第2层有 $n-1$ 条边(n 为城市的个数),分别表示下一步要到达的城市编号,第 $k(k\geqslant 2)$ 层中每个顶点有 $n-(k-1)$ 条边,分别表示下一步要到达的城市编号. 搜索策略是深度优先,在每个结点处计算所经过的路径的长度. 若以数的加法作为基本运算,该算法的最坏情形复杂性为 $O((n-1)!)$.

装载问题的回溯算法 装载问题是指,设有重量分别为 w_i 的 n 个集装箱,将其装上两艘载重分别为 c_1 和 c_2 的轮船,已知 $\sum_{i=1}^{n} w_i \leqslant c_1 + c_2$,问是否存在一种合理的装载方案将 n 个集装箱装上轮船?该问题的回溯算法的解空间是一个子集树,搜索策略是深度优先,在每个结点处检查装入的集装箱重量之和是否超过第一艘船的承载量. 遍历整个解空间后得到使得装入第一艘船的集装箱的重量最大. 最后检查剩下集装箱的重量之和是否小于第二条船的承载量,如是则回答"是",否则回答"否". 若以数的加法作为基本运算,该算法的最坏情形复杂性为 $O(2^n)$.

图的 m 着色问题 图的 m 着色问题是指,给定无向连通图 G 和 m 种颜色,用这些颜色给图的顶点着色,每个顶点一种颜色. 如果要求 G 的每条边的两个顶点是不同颜色,给出所有可能的着色方案;如果不存在着这样的方案,则回答"No". 该问题的回溯算法的解空间是一个 m 叉完全正则树,搜索策略还是深度优先. 若以颜色比较作为基本运算,该算法在最坏情形下的复杂性为 $O(nm^n)$.

背包问题的分支限界算法 问题的描述详见主教材第3章.该问题的分支限界算法是在回溯算法的基础上,定义如下代价函数:结点 $<x_1,x_2,\cdots,x_k>$ 的代价函数值是:在 $<x_1,x_2,\cdots,x_k,x_{k+1},\cdots,x_n>$ 中,无论 $x_{k+1},\cdots,x_n$ 取何值,为 $\sum_{i=1}^{n} v_i x_i$ 的一个上界;界函数在结点 $<x_1,x_2,\cdots,x_k>$ 处的函数值是在此结点之前已经找到的方案中,放入背包物品的最大总价值. 若以数的加法作为基本运算,该算法在最坏情形下的复杂性依然为 $O(2^n)$.

最大团问题的分支限界算法 该问题为求无向图的顶点个数最多的团. 该问题的分支限界算法中的解空间是一棵子集树,代价函数是 $F=c_n+n-k$,其中,c_n 为目前形成的团的顶点数(初始为0),k 为目前检索的子集树的结点的层数,即已经检索过的顶点数. 以检查两个结点是否相邻作为基本运算,该算法在最坏情形下的复杂性为 $O(n2^n)$.

货郎问题(TSP)的分支限界算法 在回溯算法的基础上,定义如下代价函数:$L=\sum_{j=1}^{k} c_j + \sum_{i_j \notin B} l_{i_j} + l_{i_k}$,其中,$\sum_{j=1}^{k} c_j$ 为已选定巡回路线的长度,$\sum_{i_j \notin B} l_{i_j} + l_{i_k}$ 为经过剩余结点回到结点1的最短距离的一个下界,界函数值为当前得到的最短巡回路线长度. 若以数的加法作为基本运算,该算法在最坏情形下的复杂性依然为 $O((n-1)!)$.

圆排列问题的分支限界算法 圆排列问题是指，给定 n 个圆，已知每个圆的半径 r_i. 现将它们放到矩形框中，各圆与矩形底边相切，求具有最小长度 l_n 的圆排列. 该问题的分支限界算法中的解空间是一个排列树，搜索策略是深度优先，代价函数 L_k 为 $x_k+(2n-2k+1)r+r_1$，其中，r 为后面待选的 $n-k$ 个圆以及第 k 个圆中最小半径的值，x_k 表示第 k 个圆的圆心的横坐标. 界函数在$<i_1,i_2,\cdots,i_k>$处的值是当前已得到的最小圆排列长度. 若以数的加法作为基本运算，该算法在最坏情形下的复杂性依然为 $O((n+1)!)$.

连续邮资问题的回溯算法 连续邮资问题是指，设有 n 种不同面值的邮票，每个信封至多贴m 张邮票，试给出邮票面值的最佳设计(面值为正整数值)，使得到从 1 开始，增量为 1 的连续邮资区间最大. 该问题的回溯算法中的解空间并不是算法一开始就能确定下来，需要边搜索边构造. 设在结点$<x_1,x_2,\cdots,x_i>$处，邮资最大连续区间为$\{1,2,\cdots,r_i\}$，则 x_{i+1} 的取值范围为$\{x_i+1,x_{i+1}+1,\cdots,r_i+1\}$.

5.2 习 题

要求：对于回溯法，要说明解向量、搜索树结构、搜索策略、代价函数(如果存在)等，并给出最坏情况下的时间复杂度函数. 对于给定的数的输入实例，求出所有的可行解(或一个最优解).

5.1 用回溯法求下列不等式的所有的整数解. 要求给出伪码和解.

$$3x_1+4x_2+2x_3\leqslant 12$$

$$x_1,x_2,x_3 \text{ 为非负整数}$$

5.2 最小重量机器设计问题. 某设备需要 4 种配件，每种 1 件. 有三个供应商提供这些配件，表 5.1 给出了相关配件的价格和每种配件的重量. 试从中选择这 4 种配件，使得总价值不超过 120，总重量最轻.

表 5.1 产品和供应商信息表

配件编号	供应商 1		供应商 2		供应商 3	
	价格	重量	价格	重量	价格	重量
1	10	5	8	6	12	4
2	20	8	21	10	30	5
3	40	5	42	4	30	10
4	30	20	60	10	45	15

5.3 如图 5.1 所示，一个 4 阶 Latin 方是一个 4×4 的方格，在它的每个方格内填入 1，2，3 或 4，并使得每个数字在每行、每列都恰好出现一次. 用回溯法求出所有第 1 行为 1，2，3，4 的 4 阶 Latin 方. 将每个解的第 2 行至第 4 行的数字从左到右写成一个序列. 例如，图 5.1 中的 Latin 方对应于解$<3,4,1,2,4,3,2,1,2,1,4,3>$. 给出所有可能的 4 阶 Latin 方.

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

图 5.1 Latin 方

5.4 应用回溯算法给出$\{1,2,3,4\}$的所有置换.

5.5 给出 8 皇后问题的一个广度优先回溯算法，并分析该算法的时

间复杂度.

5.6 子集和问题. 设 n 个不同的正数构成集合 S,求出使得和为某数 M 的 S 的所有子集.

5.7 分派问题. 给 n 个人分配 n 件工作,给第 i 个人分配第 j 件工作的成本是 $C(i,j)$. 试求成本最小的工作分配方案.

5.8 电路板排列问题. 设 $B=\{1,2,\cdots,n\}$ 是 n 块电路板的集合,$L=\{N_1,N_2,\cdots,N_m\}$ 是 m 块连接块的集合. 对 $j=1,2,\cdots,m$,连接块 $N_j\subseteq B$ 相当于一条导线,把属于 N_j 的所有电路板连起来. 如图 5.2 所示,$B=\{1,2,\cdots,8\}$,其中位置 1,2,…,8 是插槽,每个插槽可以插入一块电路板. 当电路板按照某种排列顺序全部插入后,一些连接块的导线有可能从一块电路板插槽跨到相邻的另一个插槽. 例如,图 5.2 中的 5 块连接块是:

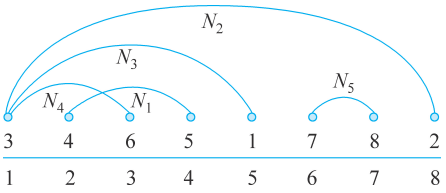

图 5.2 电路板问题的实例

$$N_1=\{4,5,6\},\quad N_2=\{2,3\},\quad N_3=\{1,3\},\quad N_4=\{3,6\},\quad N_5=\{7,8\}$$

如果电路板按照 3,4,6,5,1,7,8,2 的顺序排列,那么横跨插槽 1 和插槽 2 的连线数是 3,即连接块 N_2,N_3 和 N_4;横跨插槽 2 和插槽 3 的连线数是 4,即连接块 N_1,N_2,N_3 和 N_4……横跨插槽 7 和插槽 8 的连线数是 1,即连接块 N_2. 其中最大连线数是 4,称 4 是排列 $X=<3,4,6,5,1,7,8,2>$ 的排列密度,记作 density(X). 对于电路板的不同排列 X 和 X',其排列密度值可能是不一样的. 电路板问题是指,给定集合 B 和 L,求使得排列密度最小的电路板排列.

(1) 对上述电路板问题的实例,求出该实例的最优解.

(2) 不遍历搜索树,用数学方法证明你的解是最优的.

5.9 设有 n 项任务由 k 个可并行操作的机器完成,完成任务 i 所需要的时间是 t_i,求一个最佳任务分配方案,使得完成时间(即从时刻 0 开始计时到最后一台机器停止的时间)达到最短.

5.10 在 5.9 题中,假设每个任务有个完成期限 B_i,以及超出期限的罚款数 f_i. 试求一个最佳任务分配方案,使得完成所有任务的总罚款最少.

5.11 哨兵布置问题. 一个博物馆由排成 $m\times n$ 个矩形阵列的陈列室组成,需要在陈列室中设立哨位,每个哨位上的哨兵除了可以监视自己所在陈列室外,还可以监视自己所在陈列室的上、下、左、右 4 个陈列室,试给出一个最佳哨位安排方法,使得所有陈列室都在监视之下,但使用的哨兵最少.

5.3 习题解答与分析

5.1 34 个解. 即

(0,0,0), (0,0,1), (0,0,2), (0,0,3), (0,0,4), (0,0,5), (0,0,6),
(0,1,0), (0,1,1), (0,1,2), (0,1,3), (0,1,4), (0,2,0), (0,2,1),
(0,2,2), (0,3,0), (1,0,0), (1,0,1), (1,0,2), (1,0,3), (1,0,4),
(1,1,0), (1,1,1), (1,1,2), (1,2,0), (2,0,0), (2,0,1), (2,0,2),
(2,0,3), (2,1,0), (2,1,1), (3,0,0), (3,0,1), (4,0,0)

5.2 按照价格从小到大对配件排序. 设解向量为$<x_1,x_2,\cdots,x_n>$,$x_i=j$ 表示第 i 号配件由 j 号供应商供货. $1\leqslant x_j\leqslant m$. 结点$<x_1,x_2,\cdots,x_k>$表示已经选择了前 k 号配件的供应商,正在处理第 $k+1$ 号配件.

约束条件:选择了下一个配件后总价格不超过 120.

代价函数:

$$\sum_{i=1}^{k} w_{ix_i}+\sum_{j=k+1}^{n}\min_{l=1,2,\cdots,m}\{w_{jl}\}$$

其中 w_{jl} 表示第 l 个供应商 j 号配件的重量.

解:对实例$<3,1,2,3>$,总重量为 31,价值为 119.

5.3 有 24 个 Latin 方,如图 5.3 所示.

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

1	2	3	4
2	1	4	3
3	4	2	1
4	3	1	2

1	2	3	4
2	1	4	3
4	3	1	2
3	4	2	1

1	2	3	4
2	1	4	3
3	4	2	1
4	3	1	2

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

1	2	3	4
2	3	4	1
4	1	2	3
3	4	1	2

1	2	3	4
2	4	1	3
3	1	4	2
4	3	2	1

1	2	3	4
2	4	1	3
4	3	2	1
3	1	4	2

1	2	3	4
3	1	4	2
2	4	1	3
4	3	2	1

1	2	3	4
3	1	4	2
4	3	2	1
2	4	1	3

1	2	3	4
3	4	1	2
2	3	4	1
4	1	2	3

1	2	3	4
3	4	1	2
4	1	2	3
2	3	4	1

1	2	3	4
3	4	2	1
4	3	1	2
2	1	4	3

1	2	3	4
3	4	2	1
2	1	4	3
4	3	1	2

1	2	3	4
3	4	2	1
4	3	1	2
2	1	4	3

1	2	3	4
4	3	1	2
2	1	4	3
3	4	2	1

1	2	3	4
4	3	1	2
3	4	2	1
2	1	4	3

1	2	3	4
4	3	2	1
2	1	4	3
3	4	1	2

1	2	3	4
4	3	2	1
2	4	1	3
3	1	4	2

1	2	3	4
4	3	2	1
3	1	4	2
2	4	1	3

1	2	3	4
4	3	2	1
3	4	1	2
2	1	4	3

图 5.3 24 个 Latin 方

5.4 解向量为$<x_1,x_2,x_3,x_4>$,搜索空间是排列树. 有 24 个置换.

5.5 解向量为$<x_1,x_2,\cdots,x_8>$,搜索空间是 8 叉树. 在代表部分向量$<x_1,x_2,\cdots,x_k>$的结点处,下一步分支条件是 x_{k+1} 与 $x_1,x_2,\cdots,x_k$ 相容(不在同一行、同一列,也不在同一条斜线上). 搜索是按广度优先顺序遍历这棵树. 对于 n 后问题,最坏情况下的时间复杂度为 $O(n^n)$.

5.6 设 $S=\{a_1,a_2,\cdots,a_n\}$. 求 S 满足条件 $\sum_{a_i\in A}a_i=M$ 的所有的子集 A. 用回溯算法. 解向量为$<x_1,x_2,\cdots,x_n>$,$x_i=0,1$. 其中 $x_i=1$ 当且仅当 $a_i\in A$. 搜索空间为子集树. 部分向量$<x_1,x_2,\cdots,x_k>$表示已经考虑了对 $a_1,a_2,\cdots,a_k$ 的选择. 结点分支的约束条件是:

$$B(i)=\sum_{i=1}^{k}a_ix_i<M \quad 且 \quad a_{k+1}\in S-\{a_1,a_2,\cdots,a_k\}$$

最坏情况下算法的时间复杂度是 $O(2^n)$.

5.7 设 n 个人的集合是$\{1,2,\cdots,n\}$,n 项工作的集合是$\{1,2,\cdots,n\}$,每个人恰好 1 项工作.

$$把工作\ j\ 分配给\ i \Leftrightarrow x_i = j,\quad i,j = 1,2,\cdots,n$$

解向量是 $\boldsymbol{X}=<x_1,x_2,\cdots,x_n>$,分配成本是 $C(X)=\sum_{i=1}^{n} C(i,x_i)$. 搜索空间是排列树. 部分向量$<x_1,x_2,\cdots,x_k>$表示已经考虑了对人 $1,2,\cdots,k$ 的工作分配. 结点分支的约束条件是:

$$x_{k+1} \in \{1,2,\cdots,n\} - \{x_1,x_2,\cdots,x_k\}$$

可以设立代价函数:

$$F(x_1,x_2,\cdots,x_k)$$
$$=\sum_{i=1}^{k} C(i,x_i) + \sum_{i=k+1}^{n} \min\{C(i,t) \mid t \in \{1,2,\cdots,n\} - \{x_1,x_2,\cdots,x_k\}\}$$

界 B 是已得到的最好可行解的分配成本. 如果代价函数大于界,则回溯.

算法最坏情况下的时间复杂度是 $O(nn!)$.

5.8 搜索空间为排列树. 叶结点 $<x_1,x_2,\cdots,x_n>$对应排列 X. 在第 i 层,结点$<x_1,x_2,\cdots,x_i>$表示部分排列. 搜索策略为深度优先. 在结点$<x_1,x_2,\cdots,x_i>$,令 $B=\{x_1,x_2,\cdots,x_i\}$,下一步选择 x_{i+1}. 其约束条件是:

$$x_{i+1} \in \{1,2,\cdots,n\} - B$$

界为目前得到的最小排列密度.

在电路板 x_i 和 x_{i+1}之间,如何计算横跨插槽 x_i 和 x_{i+1}的连线数? 令 total[j]是连接块 j 所连接的电路板总数,now[j]是 $x_1,x_2,\cdots,x_i$ 中已经包含在 N_j 中的电路板数,那么

$$N_j \text{ 的连线跨越位置 } i \text{ 和 } i+1 \text{ 的电路板} \Leftrightarrow \text{now}[j] < \text{total}[j] \text{ 且 } \text{now}[j] > 0$$

即 N_j 的部分连线连接 $x_1,x_2,\cdots,x_i$ 中的电路板(因为 now[j]>0);另一部分连线连接除了 $x_1,x_2,\cdots,x_i$ 之外的其他电路板(因为 now[j]<total[j]). 令

d_i: 从位置为 1 到 i 的电路板的最大排列密度,

$$S_{i+1} = \{j \mid \text{now}[j] < \text{total}[j], \text{now}[j] > 0, j = 1,2,\cdots,m\}$$

针对本题给出的实例,$N_1=\{4,5,6\}$,$N_2=\{2,3\}$,$N_3=\{1,3\}$,$N_4=\{3,6\}$,$N_5=\{7,8\}$. 如果排列是 3,4,6,5,1,7,8,2,当 $i=2$ 时,有

$$\text{now}[1]=\text{now}[2]=\text{now}[3]=\text{now}[4]=1,\quad \text{now}[5]=0$$
$$\text{total}[1]=3$$
$$\text{total}[2]=\text{total}[3]=\text{total}[4]=\text{total}[5]=2$$

因此 N_1,N_2,N_3,N_4 的连线跨越位置为 2 和 3 的电路板. 在分支点$<x_1,x_2,\cdots,x_i>$选择 x_{i+1}后的估计是:

$$d_{i+1} = \max\{d_i, |S_{i+1}|\}$$
$$\text{density}(X) = d_n$$

算法在最坏情况下的时间复杂度是 $O(mn!)$.

5.9 设 n 个任务的标号分别为 $1,2,\cdots,n$, k 个处理器的标号分别为 $1,2,\cdots,k$.

算法主要步骤如下:

1. 给出将 n 个任务分到 k 个处理器上的分配方法.

2. 在每个分配方法下,算出每个处理器上所分配到的任务的执行时间,并求这些执行时间的最大值,该最大值即为这个分配方法的执行时间.

3. 求出所有分配方法的执行时间的最小值.

步骤 1 可以用深度优先策略遍历如图 5.4 所示的 k 元完全正则树实现.

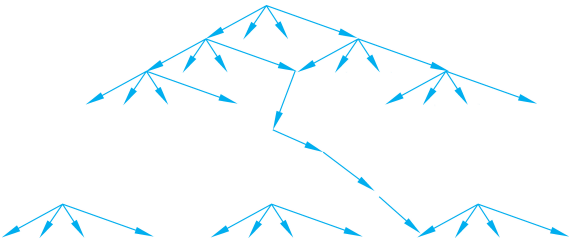

图 5.4　k 元完全正则树

树的根结点分支出来的 k 个条边分别标记 k 个处理器的标号,表示任务 1 所分配的处理器标号;树的第一层每个结点所分支出的 k 条边也分别标记 k 个处理器的标号,表示任务 2 所分配的处理器标号,以此类推.

从根到叶结点路径上的边的标记序列$<i_1,i_2,\cdots,i_n>$(其中 $1\leqslant i_j\leqslant k,1\leqslant j\leqslant n$)表示了 n 个任务在 k 个处理器上的如下分配方案:任务 $1,2,\cdots,n$ 分别分配到标号为 $i_1,i_2,\cdots,i_n$ 的处理器上执行.

按深度优先方法遍历这棵树即可完成了第一步的工作,得到 k^n 个分配方案.

步骤 2 可以按如下算法完成:

在分配方案$<i_1,i_2,\cdots,i_n>$下,如 $i_k=i_l$,则表明任务 k 和 l 被分配到同一个处理器 i_k 上执行.遍历序列 $i_1,i_2,\cdots,i_n$,找到每个处理器上分配到的所有任务 $w_1,w_2,\cdots,w_u$,则在这个分配方案下,该处理器的执行时间为 $t_{w_1}+t_{w_2}+\cdots+t_{w_u}$.再在所有处理器的执行时间中求出最大值,则该最大值即为分配方案$<i_1,i_2,\cdots,i_n>$的执行时间.该步骤的执行时间是 $O(nk^n)$.

步骤 3 是在 k^n 个分配方案的执行时间中求最小值,执行时间是 $O(k^n)$.至此算法结束.总的执行时间是 $O(nk^n)$.

5.10　同上题,设 n 个任务的标号分别为 $1,2,\cdots,n$,k 个处理器标号分别为 $1,2,\cdots,k$.算法基本步骤如下:

1. 如 5.9 题,给出将 n 个任务分到 k 个处理器上的分配方法.

2. 在每个分配方法下,找到每个处理器上所分配到的任务并计算该分配方法的罚款.

2.1　设某个处理器上分配到的所有任务是 $w_1,w_2,\cdots,w_u$,用回溯算法计算出以不同顺序在该处理器上执行任务 $w_1,w_2,\cdots,w_u$ 时,每个任务是否超时以及超时的罚款数,再得出以该顺序执行这些任务时的总罚款数,最后找到总罚款数最小的执行顺序. 该回溯算法执行过程如下:可行解是 $w_1,w_2,\cdots,w_u$ 的一个排列,因此解空间是一棵排列树,与根结点关联的 u 条边表示从任务 $w_1,w_2,\cdots,w_u$ 中选择哪个任务执行;对于第一层每个结点,向下分解出 $u-1$ 条边,表示从剩下的 $u-1$ 任务中选择哪一个执行,以此类推. 在某个结点处,如果从根结点到该结点的路径中边的标号顺序为 $w_{i_1},w_{i_2},\cdots,w_{i_v}$,则任务 w_{i_v} 的执行时间

是 $t_{wi_1}+t_{wi_2}+\cdots+t_{wi_v}$,若 $t_{wi_1}+t_{wi_2}+\cdots+t_{wi_v}>B_{wi_1}$,则增加罚款 f_{wi_v}. 在每个叶结点处即能得到在该顺序下执行 $w_1,w_2,\cdots,w_u$ 的罚款数. 遍历整棵树后计算出所有叶结点的罚款数,在所有这些罚款数中找到最小值和相应的执行顺序(最优执行顺序),则最小值即为该处理器执行 $w_1,w_2,\cdots,w_u$ 时罚款的最小值.

2.2 将每个处理器罚款的最小值相加,即为该分配方法按最优顺序执行时的罚款值. 合并每个处理器按上述顺序执行的任务,即可得到针对这个分配方法的解.

3. 比较步骤2针对每个分配方法的罚款值,找到其中最小罚款值所对应任务分配方法及其在不同处理器上的执行顺序,即可得到该问题的解.

算法在步骤1的执行时间依然是 $O(nk^n)$;步骤2的执行时间是 $O(n!\times n\times k^n)$;步骤3的执行时间是 $O(k^n)$. 故该算法最坏情形下的时间复杂度是 $O(n\times n!\times k^n)$.

5.11 本题的解是一个 $m\times n$ 的0-1矩阵 $\boldsymbol{X}[i,j]$,$\boldsymbol{X}[i,j]=1$ 当且仅当陈列室(i,j)有哨兵. 初始令所有的 $\boldsymbol{X}[i,j]=1$,$i=1,2,\cdots,m$, $j=1,2,\cdots,n$. 算法从$(1,1)$开始直到(m,n)为止,搜索树是二叉树,有 $m\times n$ 层. 每个结点对应一个陈列室位置. 如果令 $\boldsymbol{X}[i,j]=0$,则取消(i,j)位置的哨兵,进入左子树;否则,进入右子树. 在进入左子树时,需要检查房间被监视的情况. 即当取消(i,j)位置的哨兵时,此位置及其上、下、左、右位置是否被监视. 如果有不被监测的情况出现,则该分支不再搜索. 最坏情况下的时间复杂度是 $O(2^{nm})$.

第 6 章 线性规划

6.1 内 容 提 要

1. 数学模型与基本概念

线性规划的一般形式

$$\min(\max) z = \sum_{j=1}^{n} c_j x_j \qquad \text{目标函数}$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \leqslant (=, \geqslant) b_i, \quad 1 \leqslant i \leqslant m \qquad \text{约束条件}$$

$$x_j \geqslant 0, \quad j \in J \subseteq \{1,2,\cdots,n\} \qquad \text{非负条件}$$

$$x_j \text{ 任意}, \quad j \in \{1,2,\cdots,n\} - J \qquad \text{自由变量}$$

满足约束条件和非负条件的 x 称作**可行解**，使 z 取到最小值(对于 min)或最大值(对于 max)的可行解称作**最优解**.

线性规划的标准形

$$\min z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j = b_i \geqslant 0, \quad 1 \leqslant i \leqslant m$$

$$x_j \geqslant 0, \quad 1 \leqslant j \leqslant n$$

标准形的矩阵形式

$$\min z = \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}$$

$$\text{s.t. } \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

$$\boldsymbol{x} \geqslant \boldsymbol{0}$$

设 $\boldsymbol{A}$ 的秩等于 m，$\boldsymbol{A}$ 的 m 个线性无关的列称作**基**，对应的 m 个变量称作**基变量**，其余的变量称作**非基变量**. 基变量记作 $\boldsymbol{x}_B$，非基变量记作 $\boldsymbol{x}_N$. 满足约束条件 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{0}$ 且所有非基变量都等于 $\boldsymbol{0}$ 的 $\boldsymbol{x}$ 称作**基本解**，满足非负条件的基本解称作**基本可行解**，对应的基称作**可行基**.

2. 标准形的可行解的性质

主要公式 设可行基为 $\boldsymbol{B}$，则对应的基本可行解 $\boldsymbol{x}_B = \boldsymbol{B}^{-1}\boldsymbol{b}$，$\boldsymbol{x}_N = \boldsymbol{0}$，简化的目标函数

$$z=z_0+\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{x}$$

其中,$z_0=\boldsymbol{c}_B^{\mathrm{T}}\boldsymbol{B}^{-1}\boldsymbol{b}$ 是该基本可行解的目标函数值,$\boldsymbol{\lambda}^{\mathrm{T}}=\boldsymbol{c}^{\mathrm{T}}-\boldsymbol{c}_B^{\mathrm{T}}\boldsymbol{B}^{-1}\boldsymbol{A}$ 是检验数.

记 $\boldsymbol{B}^{-1}\boldsymbol{A}=(\alpha_{ij})_{m\times n}$,$\boldsymbol{\beta}=\boldsymbol{B}^{-1}\boldsymbol{b}$.

定理 6.1 如果标准形有解,则必有基本可行解.

定理 6.2 如果标准形有最优解,则必存在一个基本可行解是最优解.

定理 6.3 设基本可行解为 $\boldsymbol{x}$,对于 min(max),如果所有检验数 $\lambda_j\geqslant 0(\lambda_j\leqslant 0)$,$1\leqslant j\leqslant n$,则 $\boldsymbol{x}$ 是最优解. 如果存在 $\lambda_k<0(\lambda_k>0)$,且所有 $\alpha_{ik}\leqslant 0(1\leqslant i\leqslant m)$,则目标函数无界.

3. 对偶性

对偶线性规划

原始规划(P)	对偶规划(D)
$\max \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$	$\min \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y}$
s.t. $\boldsymbol{A}\boldsymbol{x}\leqslant\boldsymbol{b}$	s.t. $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}\geqslant\boldsymbol{c}$
$\boldsymbol{x}\geqslant\boldsymbol{0}$	$\boldsymbol{y}\geqslant\boldsymbol{0}$

定理 6.4 对偶规划的对偶规划是原始规划.

定理 6.5 设 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别是原始规划(P)和对偶规划(D)的可行解,则恒有

$$\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}\leqslant\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y}$$

定理 6.6 设 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别是原始规划(P)和对偶规划(D)的可行解,如果 $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}=\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y}$,则 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别是它们的最优解.

定理 6.7 如果原始规划(P)有最优解,则对偶规划(D)也有最优解,且它们的最优值相等;反之亦然.

原始规划和对偶规划的解只有下述3种可能:

(1) 原始规划和对偶规划都有最优解,且最优值相等.

(2) 原始规划有可行解且目标函数无界,则对偶规划无可行解;对偶规划有可行解且目标函数无界,则原始规划无可行解.

(3) 原始规划和对偶规划都没有可行解.

定理 6.8(互补松弛性) 设 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别是原始规划(P)和对偶规划(D)的可行解,则 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别是它们的最优解当且仅当

$$\left(b_i-\sum_{j=1}^{n}a_{ij}x_j\right)y_i=0,\quad 1\leqslant i\leqslant m$$

$$x_j\left(\sum_{i=1}^{m}a_{ij}y_i-c_j\right)=0,\quad 1\leqslant j\leqslant n$$

4. 算法

单纯形法 取初始可行解 $\boldsymbol{x}$. 对于 min(max),若所有 $\lambda_j\geqslant 0(\lambda_j\leqslant 0)$,$1\leqslant j\leqslant n$,则 $\boldsymbol{x}$ 是最优解,计算结束. 否则取 $\lambda_k<0(\lambda_k>0)$,若所有 $\alpha_{ik}\leqslant 0(1\leqslant i\leqslant m)$,则无最优解,计算结束. 如果存在 $\alpha_{lk}>0$,则做基变换,重复进行.

两阶段法 阶段一引入人工变量构造辅助问题,用单纯形法解辅助问题. 若辅助问题的最优值 $w^*>0$,则原问题无可行解,计算结束. 若 $w^*=0$,则删去人工变量得到原问题的一个可行解,进入阶段二用单纯形法解原问题.