整数线性规划简介

提纲

提纲

整数规划 (Integer Programming, IP)

- 要求一部分或全部决策变量取整数值的规划问题称为整数规划(integer programming).
- 不考虑整数约束条件，由余下的目标函数和约束条件构成的规划问题称为该整数规划问题的松弛问题(relaxed problem).
- 若该松弛问题是一个线性规划，则称该整数规划为整数线性规划(integer linear programming, ILP).

整数线性规划数学模型的一般形式：

$$\begin{aligned} \min_x / \max_x \quad & z = c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \\ & x \text{ 的部分或全部分量为整数} \end{aligned}$$

通过对松弛问题的最优解进行“舍入化整" 得到的点，或者不可行，或者可行但非最优。因此，必须对整数规划另行研究。

Classification of Integer Programming

- 纯整数线性规划(pure integer linear programming)：指全部决策变量都必须取整数值的整数线性规划。
- 混合整数线性规划(mixed integer linear programming)：决策变量中有一部分必须取整数值，另一部分可以不取整数值的整数线性规划。
- 0-1型整数线性规划(0-1 programming)：决策变量只能取值0或1的整数线性规划。

Classification of Integer Programming

- We call the problem mixed integer programming due to the presence of continuous variables.
- In some problems x are allowed to take on values only 0 or 1. Such variables are called binary. Integer programs involving only binary variables are called binary integer programs (BIPs). ($x \in \mathbb{B}^n$: $\{0,1\}^n$, or $x \in \{-1,1\}^n$)
- Pure Integer Linear Programming:

$$(IP) \quad \min \{c^T x \mid Ax \leq b, x \in \mathbb{Z}_+^n\}.$$

- Mixed 0-1 Programming:

$$(MIP) \quad \min\{c^T x + h^T y \mid Ax + Gy \leq b, x \in \mathbb{B}^n, y \in \mathbb{R}_+^p\}.$$

Classification of Integer Programming

- A general (nonlinear) integer programming problem can be formulated as:

$$
\begin{aligned}
(NLIP) \qquad & \min \ f(x) \\
& \text{s.t.} \ \ g_i(x) \leq b_i, \ \ i = 1, \ldots, m, \\
& \qquad x \in X,
\end{aligned}
$$

  where f and g_i, $i = 1, \ldots, m$, are real-valued functions on $\mathbb{R}^n$, and X is a finite subset in $\mathbb{Z}^n$, the set of all integer points in $\mathbb{R}^n$.

- Mixed-integer nonlinear programming problem

$$
\begin{aligned}
(MNLIP) \qquad & \min \ f(x, y) \\
& \text{s.t.} \ \ g_i(x, y) \leq b_i, \ \ i = 1, \ldots, m, \\
& \qquad x \in X, \ \ y \in Y,
\end{aligned}
$$

  where f and g_i, $i = 1, \ldots, m$, are real-valued functions on $\mathbb{R}^{n+q}$, X is a finite subset in $\mathbb{Z}^n$, and Y is a continuous subset in $\mathbb{R}^q$.

How hard is Integer Programming

- First thought (a bit naive):
  - Total enumeration: For Stone Problem and 0-1 Knapsack Problem. To list all the feasible points, a super computer with speed 10^8 (Yi) basic operations per second needs:
  - $n = 30$, $2^{30} \approx 10^9$, 10 seconds.
  - $n = 60$, $2^{60} \approx 10^{18}$, 360 years
  - $n = 100$, $2^{100} \approx 10^{30}$, 4×10^{14} years
  - ...

How hard is Integer Programming

- Most of the integer programs are NP-complete or NP-hard, which means the problem is "as difficult as a combinatorial problem can be", if we knew an efficient algorithm for one problem, we would be able to convert it to an efficient algorithm to solve any other combinatorial problem.
- Solving the linear programming relaxation results in a lower bound on the optimal solution to the IP.
- Rounding to a feasible integer solution may be difficult or impossible.
- The optimal solution to the LP relaxation can be arbitrarily far away from the optimal solution to the MIP (except totally unimodular case).
- Solving general integer programs can be much more difficult than solving linear programs or convex optimization problems.
- This is more than just an empirical statement. There is a whole theory surrounding it

提纲

Partition Problem

Given n stones of positive integer weights (i.e., given n positive integers $a_1, \ldots, a_n$), check whether you can partition these stones into two groups of equal weight, i.e., check whether a linear equation

$$\sum_{i=1}^{n} a_i x_i = 0$$

has a solution with $x_i \in \{-1, 1\}, \forall i$.

NP-complete, but usually can be solved efficiently in practice.

Knapsack Problem

A burglar has a knapsack of size b. He breaks into a store that carries a set of items n. Each item has profit c_j and size a_j . What items should the burglar select in order to optimize his heist?
Let

$$x_j = \begin{cases} 1, & \text{Item } j \text{ is selected} \\ 0, & \text{Otherwise} \end{cases}$$

The problem can be modeled as a 0-1 integer program:

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_j x_j \leq b, \\ & x_j \in \{0,1\}, j = 1, \ldots, n. \end{aligned}$$

Assignment Problem

- n people available to carry out n jobs. Each person is assigned to carry out one job. The cost for person i to do job j is c_{ij}. The problem is to find a minimum cost assignment.

$$x_{ij} = \begin{cases} 1, & \text{if person } i \text{ does job } j \\ 0, & \text{otherwise} \end{cases}$$

- The problem can be formulated as:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1, i = 1, \ldots, n \\ & \sum_{i=1}^{n} x_{ij} = 1, j = 1, \ldots, n, \ x_{ij} \in \{0, 1\}. \end{aligned}$$

Set Covering Problem

- Given a certain number of regions, the problem is to decide where to install a set of emergency service centers. For each possible center the cost of installing a service center, and which regions it can be service are known. The goal is to choose a minimum cost set of service centers so that each region is covered.
- Let $M = \{1, \ldots, m\}$ be the set of regions, and $N = \{1, \ldots, n\}$ the set of potential centers. Let $S_j \subseteq M$ be the regions that can be serviced by a center at $j \in N$, and c_j its installing cost.
- Define a 0-1 incidence matrix A such that $a_{ij} = 1$ if $i \in S_j$ and $a_{ij} = 0$ otherwise. Let

$$x_j = \begin{cases} 1, & \text{if center } j \text{ is selected} \\ 0, & \text{otherwise} \end{cases}$$

Set Covering Problem

- The problem can be formulated as

$$\begin{aligned} \min & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j \geq 1, i = 1, \ldots, m \\ & x \in \{0,1\}^n. \end{aligned}$$

Facility Location Problem

- Given a set $N = \{1, \ldots, n\}$ of potential facility locations and a set of clients $M = \{1, \ldots, m\}$. A facility placed at j costs f_j for $j \in N$. The profit from satisfying the demand of client i from a facility at j is c_{ij}. The problem is to choose a subset of the locations at which to place facilities, and then to assign the clients to these facilities so as to maximize the total profit.
- Let $x_j = 1$ if a facility is placed at j and $x_j = 0$ otherwise.
- Let y_{ij} be the fraction of the demand of client i that is satisfied from a facility at j.
- The problem can be modeled as as

$$\begin{aligned} \max & \sum_{i \in M} \sum_{j \in N} c_{ij} y_{ij} - \sum_{j \in N} f_j x_j \\ \text{s.t.} & \sum_{j \in N} y_{ij} = 1, \quad \forall i \in M, \\ & y_{ij} \leq x_j, \quad \forall i \in M,\ j \in N, \\ & x \in \{0,1\}^n,\ y \in \Re_+^{mn}. \end{aligned}$$

Traveling Salesman Problem

- A traveling salesman must visit each of n cities before returning home. He knows the distance between each of the cities and wishes to minimize the total distance traveled while visiting all of the cities.
- Problem: In what order should he visit the cities?

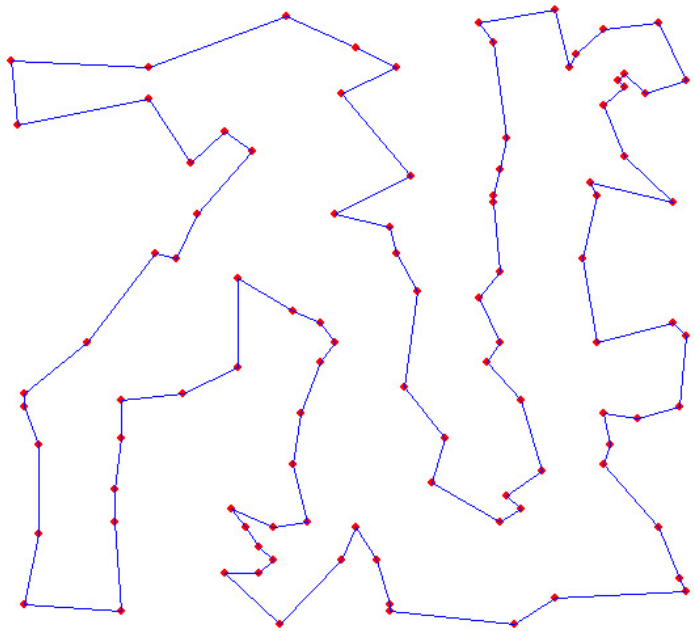


Figure: Traveling salesman problem

Traveling Salesman Problem

- Let the distance from city i to city j be c_{ij}. Let

$$x_{ij} = \begin{cases} 1, & \text{if he goes directly from } i \text{ to } j \\ 0, & \text{otherwise} \end{cases}$$

- Constraints:
  - He leaves city i exactly once:

$$\sum_{j \neq i} x_{ij} = 1, \ i = 1, \ldots, n.$$

  - He arrives at city j exactly once:

$$\sum_{i \neq j} x_{ij} = 1, \ j = 1, \ldots, n.$$

Traveling Salesman Problem

- Subtour elimination constraints:

$$\sum_{i\in S}\sum_{j\notin S} x_{ij} \geq 1, \forall S \subset N = \{1,\ldots,n\}, S \neq \emptyset,$$

or

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1, \forall S \subset N, 2 \leq |S| \leq n-1.$$

- TSP can be formulated as

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{j\neq i} x_{ij} = 1, \ i = 1,\ldots,n, \\
& \sum_{i\neq j} x_{ij} = 1, \ j = 1,\ldots,n, \\
& \sum_{i\in S}\sum_{j\in S} x_{ij} \leq |S| - 1, \forall S \subset N, 2 \leq |S| \leq n-1, \\
& x \in \{0,1\}^n.
\end{aligned}$$

Traveling Salesman Problem

Figure: TSP 1000 cities

Traveling Salesman Problem

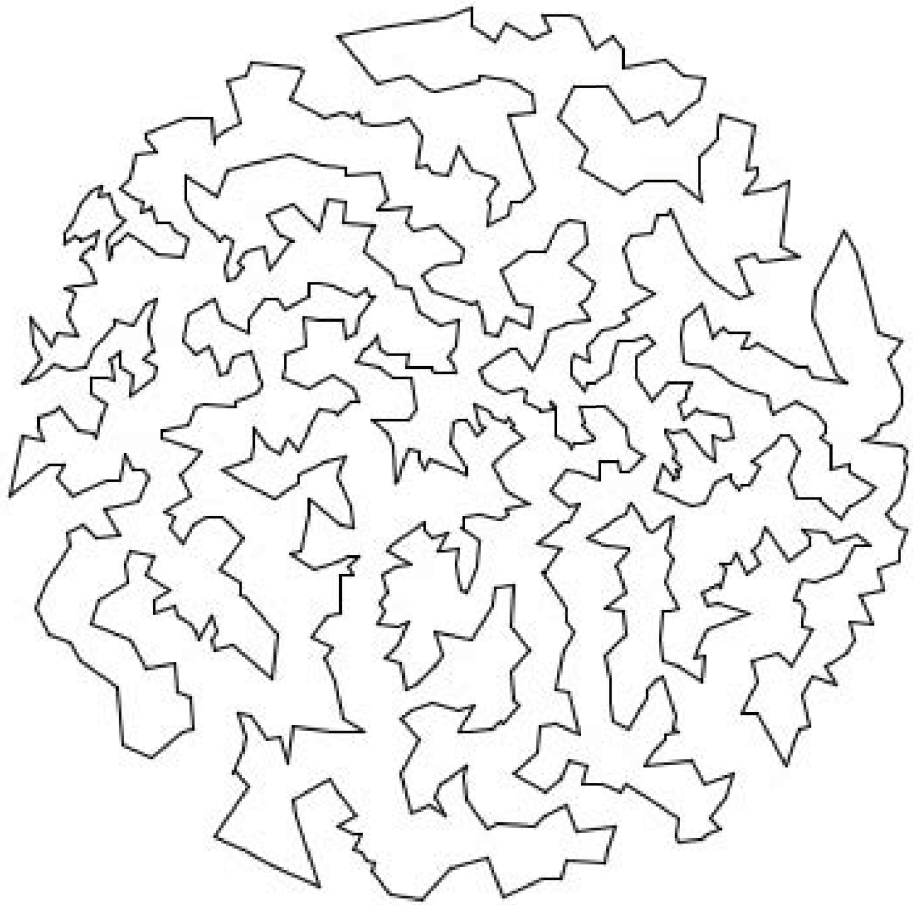


Figure: Optimal tour for the TSP 1000 cities

Generalized Assignment Problem

Given m machines and n jobs, find a least cost assignment of jobs to machines not exceeding the machine capacities. Each job j requires a_{ij} units of machine i' s capacity b_i. The problem can be modeled as

$$\begin{aligned}
\min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{n} a_{ij}x_j \le b_i, \ \forall i, \quad \text{(machine capacity)} \\
& \sum_{i=1}^{m} x_{ij} = 1, \ \forall j, \quad \text{(assign all jobs)} \\
& x_{ij} \in \{0,1\}, \ \forall i,j
\end{aligned}$$

Minimum cost network flows

- A digraph $G = (V, A)$ with arc capacity h_{ij} (integer), for $(i, j) \in A$, demand b_i at node $i \in V$, unit flow cost c_{ij} for $(i, j) \in A$.
- Find feasible integer flow with the minimum cost.
- x_{ij}: the flow for arc (i, j), $V_i^+ = \{k \mid (i, k) \in A\}$, $V_i^- = \{k \mid (k, i) \in A\}$.
- The model:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{k\in V_i^+} x_{ik} - \sum_{k\in V_i^-} x_{ki} = b_i, \ i \in V \\ & 0 \le x_{ij} \le h_{ij}, x_{ij} \text{ integer}. \end{aligned}$$

Algorithms

- Branch and Bound
- Cutting plane method
- Dynamic programming
- Lagrangian Relaxations
- Column generation and Danzig-Wolfe decomposition
- Branch and Price

Nonlinear Integer Programming

- Quadratic 0-1 optimization
- Max-cut problem
- Quadratic knapsack problem
- ...

整数规划的例子

工厂A1和A2生产某种物资。由于该种物资供不应求，故需要再建一家工厂。相应的建厂方案有A3和A4两个。这种物资的需求地有B1, B2, B3, B4四个。各工厂年生产能力、各地年需求量、各厂至各需求地的单位物资运费c_{ij}，见下表

	B1	B2	B3	B4	年生产能力
A1	2	9	3	4	400
A2	8	3	5	7	600
A3	7	6	1	2	200
A4	4	5	2	5	200
年需求量	350	400	300	150	

工厂A3或A4开工后，每年的生产费用估计分别为1200万或1500万元。现要决定应该建设工厂A3还是A4, 才能使今后每年的总费用最少。

整数规划的例子

这是一个物资运输问题，特点是事先不能确定应该建A3 还是A4中哪一个，因而不知道新厂投产后的实际生产物资。为此，引入0-1变量：

$$y_i = \begin{cases} 1, & \text{若建厂;} \\ 0, & \text{若不建厂.} \end{cases} \quad (i = 1, 2).$$

整数规划的例子

设x_{ij}为由A_i 运往B_j 的物资数量，单位为千吨, z表示总费用，单位万元。则该规划问题的数学模型可以表示为：

$$
\begin{aligned}
\min_{x,y} \quad & z = \sum_{i=1}^{4}\sum_{j=1}^{4} c_{ij}x_{ij} + 1200y_1 + 1500y_2 \\
\text{s.t.} \quad & x_{11} + x_{21} + x_{31} + x_{41} = 350 \\
& x_{12} + x_{22} + x_{32} + x_{42} = 400 \\
& x_{13} + x_{23} + x_{33} + x_{43} = 300 \\
& x_{14} + x_{24} + x_{34} + x_{44} = 150 \\
& x_{11} + x_{12} + x_{13} + x_{14} = 400 \\
& x_{21} + x_{22} + x_{23} + x_{24} = 600 \\
& x_{31} + x_{32} + x_{33} + x_{34} = 200y_1 \\
& x_{41} + x_{42} + x_{43} + x_{44} = 200y_2 \\
& x_{ij} \geq 0\,(i,j = 1,2,3,4), y_i \in \{0,1\}\,(i = 1,2).
\end{aligned}
$$

整数规划的例子

Remark

1. 最优解中y_1 与y_2 不会同时为0, 因为若不建厂，供给小于需求；
2. 最优解中y_1 与y_2 不会同时为1, 因为若同时建厂花费增加；
3. 分别计算$y = (1, 0)$ 与$y = (0, 1)$的花费再决定如何选择，相当于枚举法；当变量个数较大时，枚举法不可行。

整数规划的例子

现有资金总额为B, 可供选择的投资项目有n个，项目j所需投资额和预期收益分别为a_j 和c_j, $j = 1, 2, \ldots, n$. 此外由于种种原因，有三个附加条件：

- 若选择项目1, 就必须同时选择项目2. 反之不一定;
- 项目3 和4 中至少选择一个;
- 项目5, 6, 7 中恰好选择2 个。

应该怎样选择投资项目，才能使总预期收益最大。

整数规划的例子

对每个投资项目都有被选择和不被选择两种可能，因此分别用0和1表示，令x_j表示第j 个项目的决策选择，记为：

$$x_j = \begin{cases} 1, & \text{对项目}j\text{投资;} \\ 0, & \text{对项目}j\text{不投资.} \end{cases} \quad (j = 1, 2, \ldots, n).$$

投资问题可以表示为:

$$\begin{aligned} \max_x \quad & z = \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_j x_j \le B, \\ & x_2 \ge x_1, \\ & x_3 + x_4 \ge 1, \\ & x_5 + x_6 + x_7 = 2, \\ & x_j \in \{0, 1\}, \ j = 1, 2, \ldots, n. \end{aligned}$$

指派问题(分配问题)

人事部门欲安排四人到四个不同岗位工作，每个岗位一个人。经考核四人在不同岗位的成绩（百分制）如表所示，如何安排他们的工作使总成绩最好。

人员＼工作	A	B	C	D
甲	85	92	73	90
乙	95	87	78	95
丙	82	83	79	90
丁	86	90	80	88

设$x_{ij} = \begin{cases} 1, & \text{分配第}i\text{人做}j\text{工作;} \\ 0, & \text{不分配第}i\text{人做}j\text{工作.} \end{cases}$

指派问题(分配问题)

$$
\begin{aligned}
\max \quad & z = 85x_{11} + 92x_{12} + 73x_{13} + 90x_{14} \\
& +95x_{21} + 87x_{22} + 78x_{23} + 95x_{24} \\
& +82x_{31} + 83x_{32} + 79x_{33} + 90x_{34} \\
& +86x_{41} + 90x_{42} + 80x_{43} + 88x_{44} \\
\text{s.t.} \quad & x_{11} + x_{12} + x_{13} + x_{14} = 1 \\
& x_{21} + x_{22} + x_{23} + x_{24} = 1 \\
& x_{31} + x_{32} + x_{33} + x_{34} = 1 \\
& x_{41} + x_{42} + x_{43} + x_{44} = 1 \\
& x_{11} + x_{21} + x_{31} + x_{41} = 1 \\
& x_{12} + x_{22} + x_{32} + x_{42} = 1 \\
& x_{13} + x_{23} + x_{33} + x_{43} = 1 \\
& x_{14} + x_{24} + x_{34} + x_{44} = 1 \\
& x_{ij} \in \{0, 1\}, \ i, j = 1, 2, 3, 4.
\end{aligned}
$$

整数规划问题解的特征

- 整数规划问题的可行解集合是它松弛问题可行解集合的一个子集，任意两个可行解的凸组合不一定满足整数约束条件，因而不一定仍为可行解。
- 整数规划问题的可行解一定是它的松弛问题的可行解（反之不一定）。
- 整数规划问题的最优解的目标函数值不会优于其松弛问题的最优解的目标函数值。

例子

设整数规划问题如下

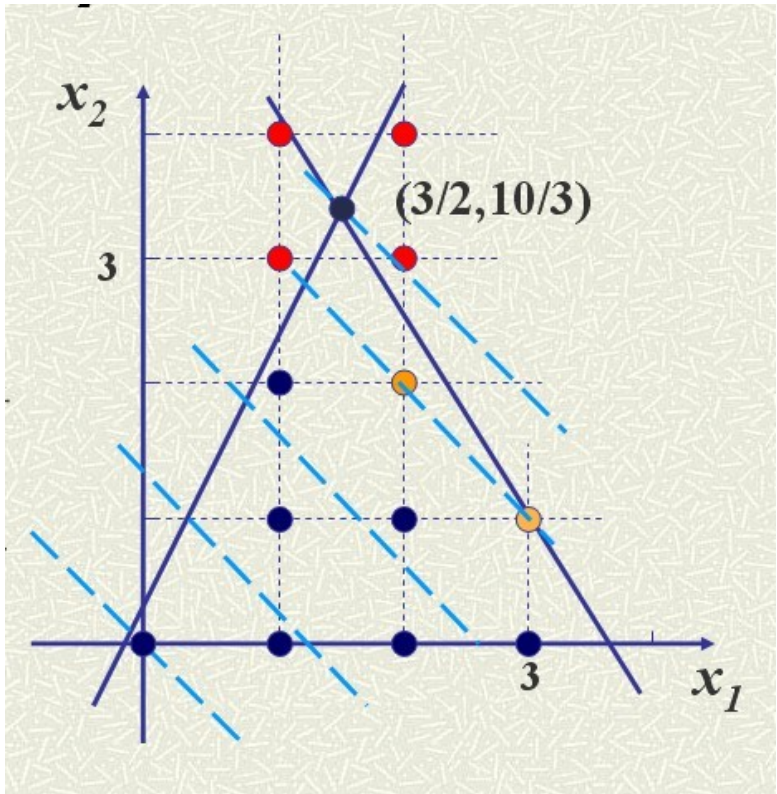
$$\begin{aligned} \max \quad & z = x_1 + x_2 \\ \text{s.t.} \quad & 14x_1 + 9x_2 \le 51 \\ & -6x_1 + 3x_2 \le 1 \\ & x_1, x_2 \ge 0 \text{ 且为整数.} \end{aligned}$$

首先不考虑整数约束，得到松弛线性规划问题

$$\begin{aligned} \max \quad & z = x_1 + x_2 \\ \text{s.t.} \quad & 14x_1 + 9x_2 \le 51 \\ & -6x_1 + 3x_2 \le 1 \\ & x_1, x_2 \ge 0. \end{aligned}$$

例子

用图解法求出最优解为：$(x_1, x_2) = (3/2, 10/3)$, 且有$z = 29/6$. 如用舍入取整法可得到4个点, 即$(1,3)$, $(2,3)$, $(1,4)$, $(2,4)$. 它们都不是整数规划的最优解。按整数规划约束条件，其可行解肯定在线性规划问题的可行域内且为整数点。故整数规划问题的可行解集是一个有限集。易见，$(2,2)$, $(3,1)$ 是该整数线性规划的最优解。



提纲

提纲

分支定界法(Branch and bound method) 的解题步骤

1. 求整数规划的松弛问题最优解；若松弛问题的最优解满足整数要求，则得到整数规划的最优解，否则转下一步；
2. 分支：任意选一个非整数解的变量x_i, 在整数规划中加上约束：

$$x_i \leq [x_i], \quad x_i \geq [x_i] + 1$$

   组成两个新的整数规划问题，称为分支。
3. 求解所有分支问题的松弛问题，得到各松弛问题的解及目标函数值。
   - 若某分支的松弛问题的解是整数，并且目标函数值优于其它分支问题的松弛问题的目标值，则将其它分支剪去不再计算；
   - 若还存在某分支，其松弛问题的解为非整数，并且目标值优于目前找到的最优的整数解的目标值，则需要(1) 为原整数规划问题的最优目标函数值定下界与上界; (2) 继续分支，再检查，直到得到最优解。

分支定界法– 例子

用分支定界法求解整数规划问题

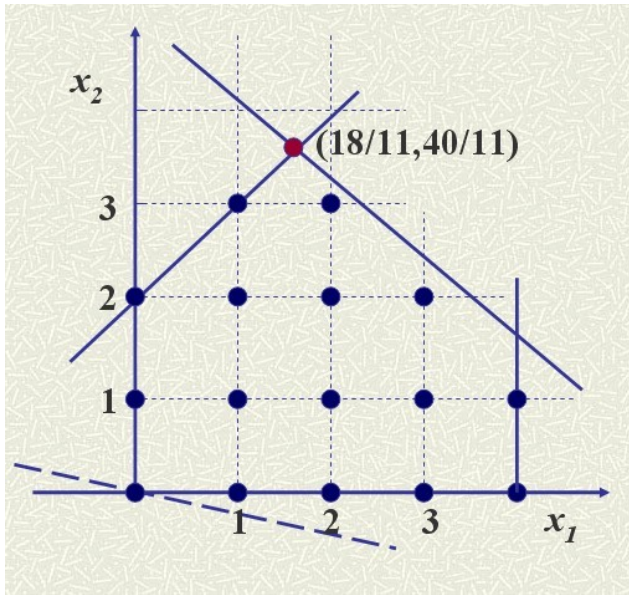
$$
\begin{aligned}
\text{ILP:} \quad \min \quad & z = -x_1 - 5x_2 \\
\text{s.t.} \quad & x_1 - x_2 \geq -2 \\
& 5x_1 + 6x_2 \leq 30 \\
& x_1 \leq 4 \\
& x_1, x_2 \geq 0 \text{ 且均为整数.}
\end{aligned}
$$

分支定界法– 例子

首先去掉整数约束，变成一般线性规划问题(即原整数规划问题的松驰问题)

$$
\begin{aligned}
\text{LP:} \quad \min \quad & z = -x_1 - 5x_2 \\
\text{s.t.} \quad & x_1 - x_2 \geq -2 \\
& 5x_1 + 6x_2 \leq 30 \\
& x_1 \leq 4,\ x_1, x_2 \geq 0.
\end{aligned}
$$

解得$x = (18/11, 40/11)$, $z = -218/11 \approx -19.8$, 此亦为原整数规划问题最优函数值的下界.



分支定界法– 例子

对于$x_1 = 18/11 \approx 1.64$, 利用约束条件$x_1 \leq 1$ 与$x_1 \geq 2$ 将原整数规划问题分支为ILP1 与ILP2:

$$
\begin{aligned}
\text{ILP1:} \quad \min \quad & z = -x_1 - 5x_2 \\
\text{s.t.} \quad & x_1 - x_2 \geq -2 \\
& 5x_1 + 6x_2 \leq 30 \\
& x_1 \leq 4 \\
& x_1 \leq 1 \\
& x_1, x_2 \geq 0 \text{ 且均为整数.}
\end{aligned}
$$

$$
\begin{aligned}
\text{ILP2:} \quad \min \quad & z = -x_1 - 5x_2 \\
\text{s.t.} \quad & x_1 - x_2 \geq -2 \\
& 5x_1 + 6x_2 \leq 30 \\
& x_1 \leq 4 \\
& x_1 \geq 2 \\
& x_1, x_2 \geq 0 \text{ 且均为整数.}
\end{aligned}
$$

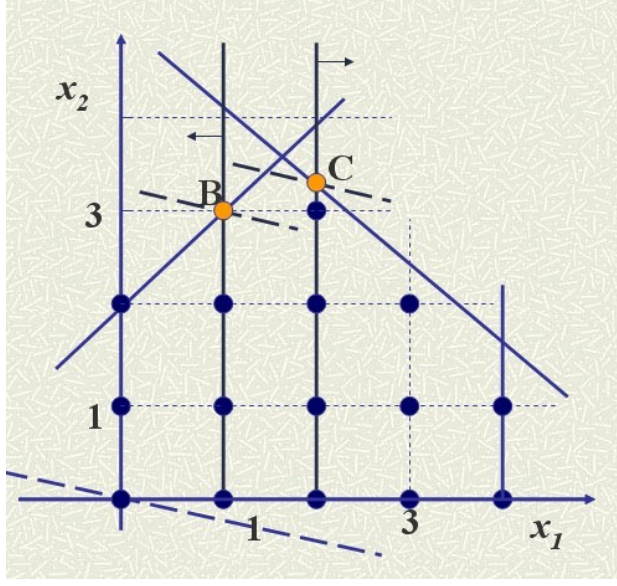
分支定界法– 例子

称与ILP1/ILP2 相应的松弛问题为LP1/LP2. 解LP1 与LP2 得

$$LP1: \quad x = (1,3), z = -16$$

$$LP2: \quad x = (2,10/3), z = -56/3 \approx -18.7.$$

- 分支ILP1 已探明（最优整数解为$x = (1,3)$, 最优函数值$z = -16$），此分支停止计算；
- 由于LP2的最优函数值$-56/3 < -16$, 故分支ILP2 可能存在函数值小于-16的整数解, 因此该分支需继续计算。
- 定界：$-56/3 \leq$ ILP 的最优函数值 ≤ -16.



分支定界法– 例子

分支较多时，为ILP 的最优函数值定界的具体步骤如下：

1. 将所有分支分为两类：松弛问题的最优解为整数解的分支、松弛问题的最优解不是整数解的分支；
2. 上界*ub* 应定为目前找到的最好的整数解所对应的函数值；若尚未找到整数解，则定为$+\infty$;
3. 若某分支所对应的松弛问题的最优函数值大于*ub*, 则应剪去该分支(无论松弛问题的最优解是否为整数解)；
4. 而下界*lb* 则应定为尚未找到整数解的所有分支(已剪去的分支除外)所对应的松弛问题的最优函数值的最小者。若剪去所有可以剪掉的分支后，剩余分支均已找到整数解，则最优解已定，停止结算。

分支定界法– 例子

在ILP2 中分别再加入条件$x_2 \le 3$ 与$x_2 \ge 4$, 得如下两分支：

$$
\begin{aligned}
\text{ILP21:} \quad \min \quad & z = -x_1 - 5x_2 \\
\text{s.t.} \quad & x_1 - x_2 \ge -2 \\
& 5x_1 + 6x_2 \le 30 \\
& x_1 \le 4, x_1 \ge 2 \\
& x_2 \le 3 \\
& x_1, x_2 \ge 0 \text{ 且均为整数.}
\end{aligned}
$$

$$
\begin{aligned}
\text{ILP22:} \quad \min \quad & z = -x_1 - 5x_2 \\
\text{s.t.} \quad & x_1 - x_2 \ge -2 \\
& 5x_1 + 6x_2 \le 30 \\
& x_1 \le 4, x_1 \ge 2 \\
& x_2 \ge 4 \\
& x_1, x_2 \ge 0 \text{ 且均为整数.}
\end{aligned}
$$

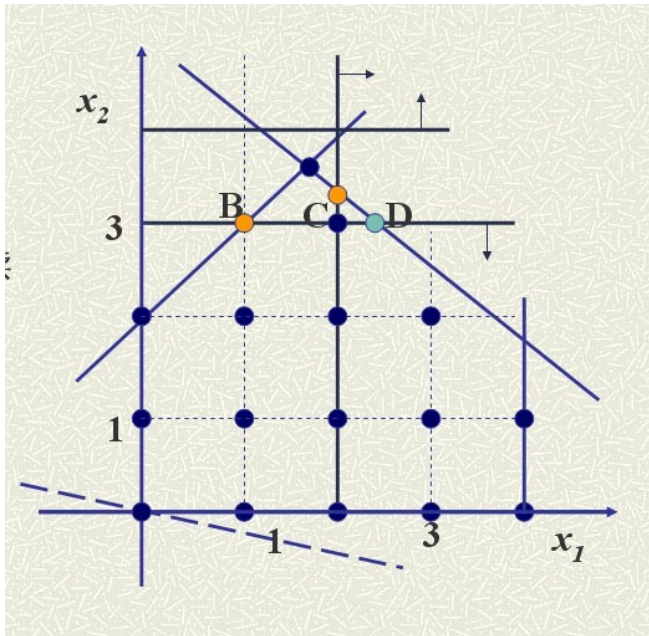
分支定界法– 例子

分别求出松弛问题LP21和LP22的最优解.

$$LP21: \quad x = (2.4, 3), z = -87/5 \approx -17.4$$

$$LP22: \quad \text{无可行解, 剪支.}$$

因为没有找到更好的整数解，故ub无需更新；LP21 的最优解不是整数点，且最优函数值小于$ub = -16$, 因此$lb = -87/5$, 需继续对ILP21 进行分支。



分支定界法– 例子

加入$x_1 \le 2$ 与$x_1 \ge 3$, 继续对ILP21 进行分支:

$$
\begin{aligned}
\text{ILP211:} \quad \min \quad & z = -x_1 - 5x_2 \\
\text{s.t.} \quad & x_1 - x_2 \ge -2 \\
& 5x_1 + 6x_2 \le 30 \\
& x_1 \le 4, x_1 \ge 2, x_2 \le 3 \\
& x_1 \le 2 \\
& x_1, x_2 \ge 0 \text{ 且均为整数.}
\end{aligned}
$$

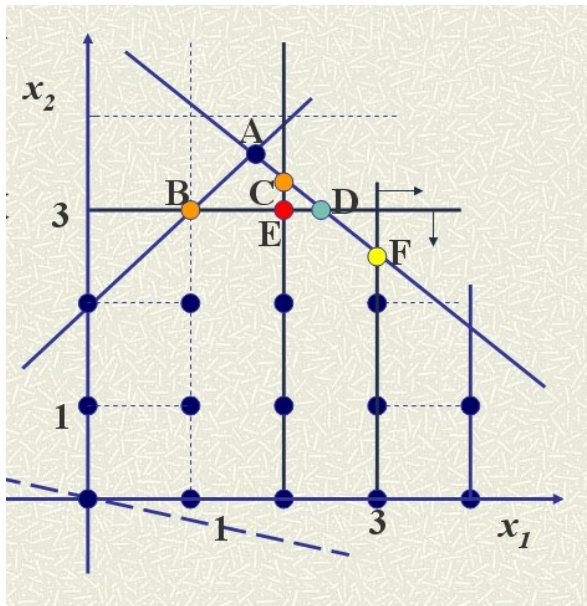
$$
\begin{aligned}
\text{ILP212:} \quad \min \quad & z = -x_1 - 5x_2 \\
\text{s.t.} \quad & x_1 - x_2 \ge -2 \\
& 5x_1 + 6x_2 \le 30 \\
& x_1 \le 4, x_1 \ge 2, x_2 \le 3 \\
& x_1 \ge 3 \\
& x_1, x_2 \ge 0 \text{ 且均为整数.}
\end{aligned}
$$

分支定界法– 例子

分别求出松弛问题LP211和LP212的最优解.

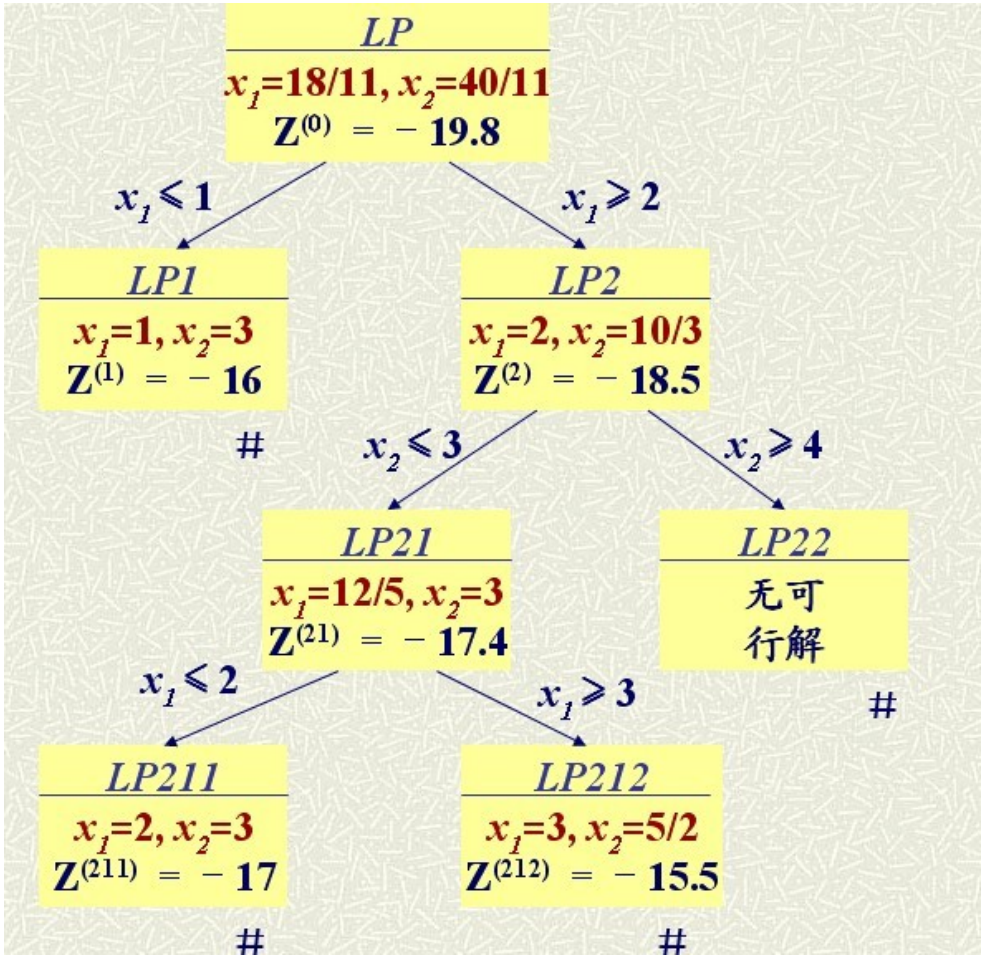
$$LP211: \quad x = (2,3), z = -17$$
$$LP212: \quad x = (3,2.5), z = -15.5.$$

找到更好的整数解，故更新$ub = -17$; LP212 的最优函数值大于ub, 故可以剪支。此时，找到最优解$(2,3)$.

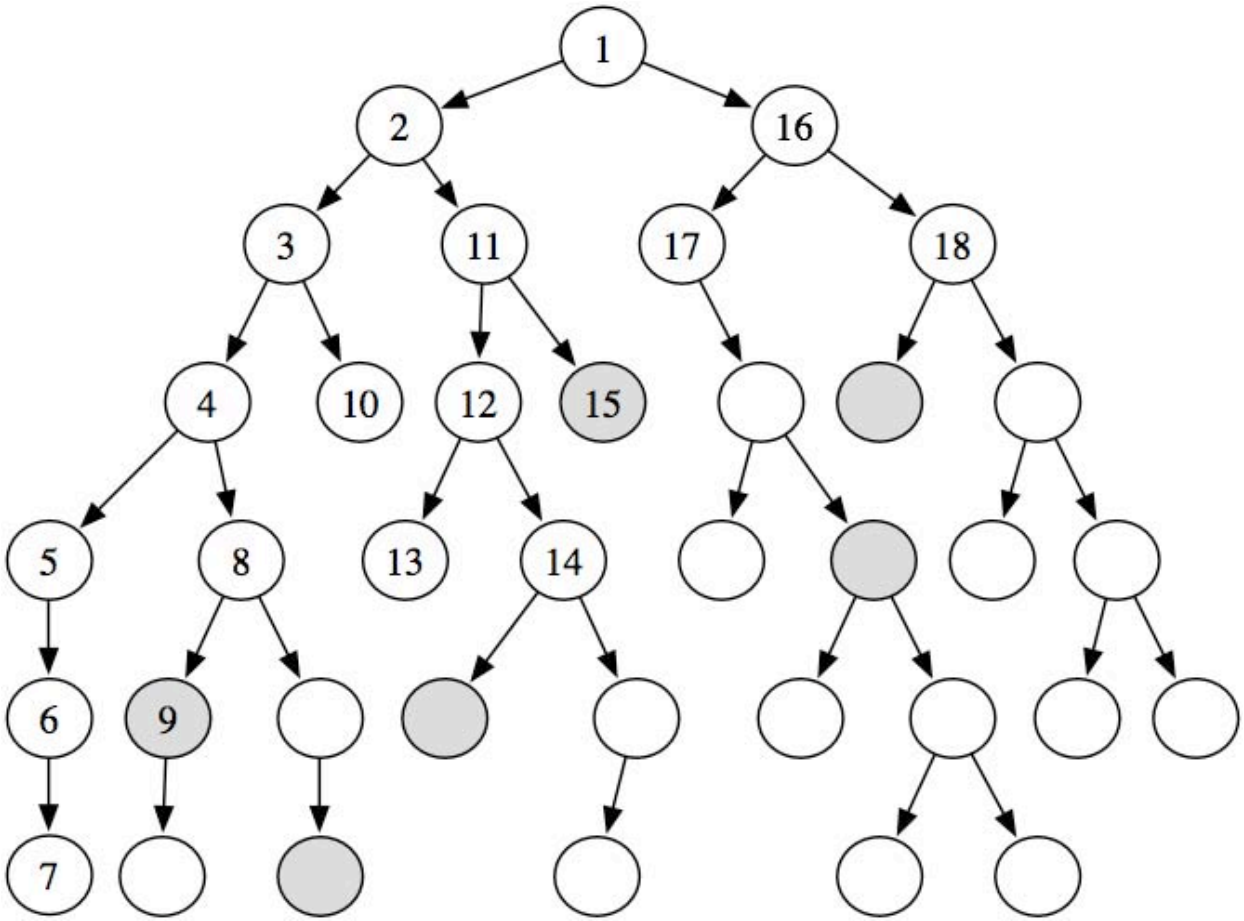


分支定界法– 例子

原整数规划问题的最优解为$(2,3)$, 最优函数值为-17.



Branch-and-Bound



分支定界法

优点： 通过解LP 来解ILP, 而LP 的理论、算法、软件等已相当成熟。

缺点： 对大、难的ILP 问题，分支分层可能会非常多，本质上相当于枚举法；分支分层很多时，需解的LP数量相当可观；分层越深，LP 的约束条件越多，程序实现需考虑到简化约束条件、热启动等加速计算的技巧。

其他问题： 如何快速改善最优函数值的上下界；如何分支更合理；如何加速剪枝；与割平面法结合使用；推广到解混合整数线性规划问题...

作业

用分支定界法求解（松弛问题可借助计算机或用图解法求解）

$$
\begin{aligned}
\min \quad & z = -4x_1 - 3x_2 \\
\text{s.t.} \quad & 3x_1 + 2x_2 \le 25 \\
& 4x_1 + 5x_2 \le 50 \\
& x_1, x_2 \ge 0, \text{且均为整数.}
\end{aligned}
$$

提纲

割平面法(cutting plane method)

割平面法仍然是用解线性规划的方法去解整数线性规划问题，其基本思想如下：

首先不考虑变量是整数这一条件，解松弛问题。若得到非整数的最优解，则增加能割去非整数解的线性约束条件(或称为割平面)，使得原可行域被切割掉一部分，而这部分只包含非整数解，不包含任何整数可行解。

割平面方法就是指出怎样找到适当的割平面(通常需要反复切割很多次)，使得切割后最终得到这样的可行域：它的一个整数极点恰好是问题的最优解。

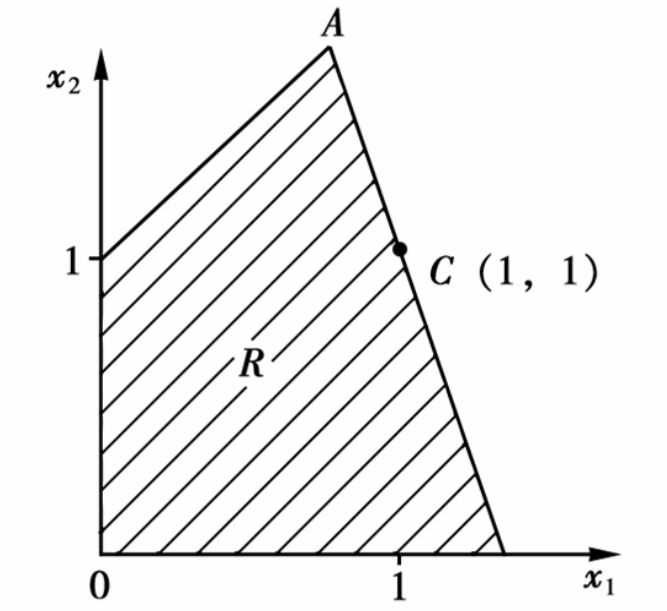
以下仅讨论纯整数线性规划的情形。

割平面法– 例子

$$\begin{aligned} \min \quad & z = -x_1 - x_2 \\ \text{s.t.} \quad & -x_1 + x_2 \le 1 \\ & 3x_1 + x_2 \le 4 \\ & x_1, x_2 \ge 0 \text{ 且为整数.} \end{aligned}$$

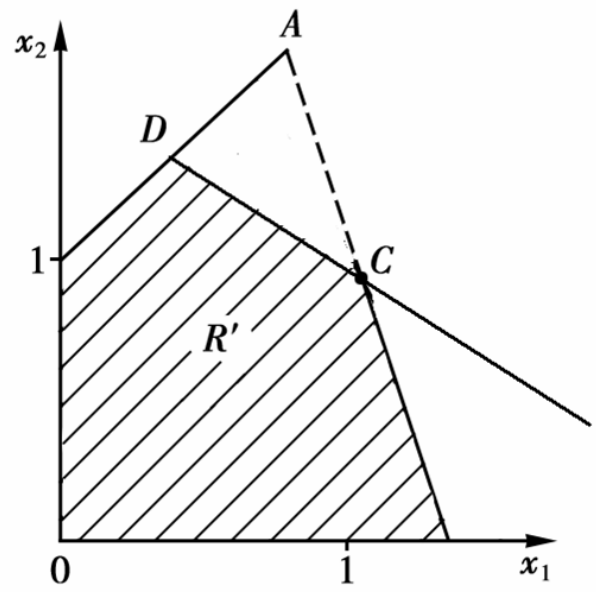
先不考虑整数约束条件，求得相应的松弛问题的最优解为：

$$x_1 = 3/4, x_2 = 7/4, \min z = -5/2.$$



割平面法– 例子

现设想，如能找到像CD 那样的直线去切割可行域R, 去掉三角形域ACD, 那么具有整数坐标的C 点(1, 1) 就是域R' 的一个极点。



若在域R' 上优化目标函数, 而得到的最优解又恰巧在C 点，则得到原整数线性规划问题的解。

解法的关键是怎样构造一个这样的“割平面" CD, 尽管它可能不是唯一的，也可能不是一步能求到的。

割平面法– 例子

在原问题的前两个不等式中增加非负松弛变量x_3 与x_4, 使两式变成等式约束：

$$\begin{aligned} -x_1 + x_2 + x_3 &= 1 \\ 3x_1 + x_2 + x_4 &= 4. \end{aligned}$$

不考虑整数约束条件，用单纯形表解题，见下表：

x_1	x_2	x_3	x_4	b
-1	1	1	0	1
3	1	0	1	4
-1	-1	0	0	0

x_1	x_2	x_3	x_4	b
1	0	-1/4	1/4	3/4
0	1	3/4	1/4	7/4
0	0	1/2	1/2	5/2

割平面法– 例子

由于目前得到的解不满足整数要求，需要考虑将可行域割去一部分，再求最优解，而割去的部分不含整数可行点。可从最终计算表中得到非整数变量对应的关系式：

$$\begin{aligned} x_1 - \frac{1}{4}x_3 + \frac{1}{4}x_4 &= \frac{3}{4} \\ x_2 + \frac{3}{4}x_3 + \frac{1}{4}x_4 &= \frac{7}{4}. \end{aligned}$$

为了得到整数最优解，将上式变量的系数和常数项都分解成整数和非负真分数两部分之和：

$$\begin{aligned} (1+0)x_1 + (-1+\frac{3}{4})x_3 + (0+\frac{1}{4})x_4 &= 0 + \frac{3}{4} \\ (1+0)x_2 + (0+\frac{3}{4})x_3 + (0+\frac{1}{4})x_4 &= 1 + \frac{3}{4}. \end{aligned}$$

割平面法– 例子

将整数部分与分数部分分开，分别移到等式左右两边，得到：

$$\begin{aligned} x_1 - x_3 &= \frac{3}{4} - (\frac{3}{4}x_3 + \frac{1}{4}x_4) \\ x_2 - 1 &= \frac{3}{4} - (\frac{3}{4}x_3 + \frac{1}{4}x_4). \end{aligned}$$

由x_1, x_2 为非负整数可知x_3, x_4 亦为非负整数，故上式左端为整数，因此右端也为整数。因此有

$$\frac{3}{4} - (\frac{3}{4}x_3 + \frac{1}{4}x_4) \leq 0$$

或

$$-3x_3 - x_4 \leq -3.$$

此即为一个切割方程，将其作为约束条件加入原来的问题中，再解。引入非负松弛变量x_5:

$$-3x_3 - x_4 + x_5 = -3.$$

割平面法– 例子

x_1	x_2	x_3	x_4	x_5	b
1	0	-1/4	1/4	0	3/4
0	1	3/4	1/4	0	7/4
0	0	-3	-1	1	-3
0	0	1/2	1/2	0	5/2

用对偶单纯形法解之：

x_1	x_2	x_3	x_4	x_5	b
1	0	0	1/3	-1/12	1
0	1	0	0	1/4	1
0	0	1	1/3	-1/3	1
0	0	0	1/3	1/6	2

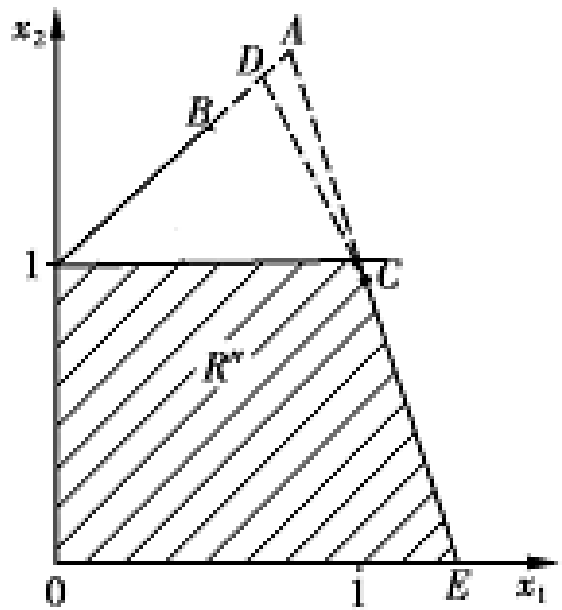
由于x_1, x_2 的值已都是整数，解题完成。

割平面法– 例子

注：新得到的约束条件$-3x_3 - x_4 \leq -3$ 等价于$x_2 \leq 1$, 仅需将

$$\begin{aligned} x_3 &= 1 + x_1 - x_2 \\ x_4 &= 4 - 3x_1 - x_2, \end{aligned}$$

代入即可。而$x_2 \leq 1$ 将可行域中$x_2 > 1$ 的部分切割掉，被切割掉的部分不含整数点，且在切割后的可行域上优化目标函数恰好得到整数解。



求切割方程的步骤

1. 设x_i 是相应线性规划最优解中为分数值的一个基变量（若不存在，则已经最优），由最终的单纯形表得

$$x_i + \sum_{j \in K} y_{ij} x_j = b_i,$$

其中$i \in Q$ (Q 表示基变量的下标集合), $j \in K$ (K 表示非基变量的下标集合).

2. 将b_i 和y_{ij} 都分解成整数部分N 与非负真分数f 之和，即

$$\begin{aligned} b_i &= N_i + f_i \\ y_{ij} &= N_{ij} + f_{ij}, \end{aligned}$$

其中$0 < f_i < 1, 0 \le f_{ij} < 1$. 代入得

$$x_i + \sum_j N_{ij} x_j - N_i = f_i - \sum_j f_{ij} x_j,$$

求切割方程的步骤

3. 考虑变量(包括松弛变量)为整数的约束条件(当然还有非负的条件)。这时，上式由左边看必须是整数, 但从由右边看，因为$0 < f_i < 1$, 所以不能为正，即

$$f_i - \sum_{j \in K} f_{ij} x_j \leq 0.$$

此即为一个切割方程。

易见：（1）切割方程真正进行了切割，因为至少把非整数最优解这一点割掉了；（2）没有割掉整数解，这是因为相应的线性规划的任意整数可行解都满足切割方程。

Remark

(1) 先对切割方程化简，使系数均为整数，再引入松弛变量；*(2)* 切割方程加入最后一张单纯形表后，右端项一定为负（其他分量均为正），因此使用对偶单纯形法仅需一次旋转运算。

割平面法

割平面法自1958年由Gomory 提出后便引起广泛关注。但割平面法经常遇到收敛很慢的情形，因而很少单独使用。割平面法可与其他方法，如分支定界法，配合使用，通常是有效的。

作业

用割平面法解下面问题：

$$
\begin{aligned}
\min \quad & -4x_1 - 3x_2 \\
s.t. \quad & 6x_1 + 4x_2 \leq 30 \\
& x_1 + 2x_2 \leq 10 \\
& x_1, x_2 \geq 0 \text{ 且为整数.}
\end{aligned}
$$

提纲

0-1整数线性规划 (Binary Integer Programming)

$$\begin{aligned} \min_x / \max_x \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \in \{0,1\}^n. \end{aligned}$$

Equivalent to

$$\begin{aligned} \min_x / \max_x \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & 0 \le x \le 1 \\ & x\text{的分量为整数}. \end{aligned}$$

Thus, can be solved by branch and bound method or cutting plane method.

解 BIP 的隐枚举法 I

枚举法：2^n 种情况，首先用可行性条件进行过滤，进一步比较函数值选出最优解。仅当 n 较小时有效。

隐枚举法是在枚举法的基础上进行了改进，对最小化问题，求解的基本步骤如下：

1. 寻找一个初始可行解 x_0, 得到最优目标函数值的上界 z_0.
2. 按照完全枚举法列出 2^n 种情况，当组合解 x_j 对应的目标函数值 z_j 大于 z_0 时则将其过滤掉；当 $z_j \leq z_0$ 时，再进一步检验是否满足约束条件，得到可行解。
3. 根据 z_j 的值确定最优解。
4. 这里的上界 z_0 可以逐渐减小，当某个可行解对应的目标函数值 z_j 小于 z_0 时，则将 z_j 作为新的上界。

(分支定界法是一种隐枚举法)

解 BIP 的隐枚举法 II

用隐枚举法求解下面问题：

$$
\begin{aligned}
\min \quad & z = -6x_1 - 2x_2 - 3x_3 - 5x_4 \\
s.t. \quad & 4x_1 + 2x_2 + x_3 + 3x_4 \leq 10 \\
& 3x_1 - 5x_2 + x_3 + 6x_4 \geq 4 \\
& 2x_1 + x_2 + x_3 - x_4 \leq 3 \\
& x_1 + 2x_2 + 4x_3 + 5x_4 \leq 10 \\
& x \in \{0,1\}^4.
\end{aligned}
$$

(1) 首先，第一个约束条件“$4x_1 + 2x_2 + x_3 + 3x_4 \leq 10$" 可去掉，因为对于任意的 $x \in \{0,1\}^4$, 该约束都是满足的。

(2) 观察得到可行解 $x_0 = (1,0,0,1)$, 对应 $z_0 = -11$.

解 BIP 的隐枚举法 III

(3) 将$-6x_1 - 2x_2 - 3x_3 - 5x_4 \leq -11$ 加入约束条件：

$$
\begin{aligned}
\min \quad & z = -6x_1 - 2x_2 - 3x_3 - 5x_4 \\
s.t. \quad & -6x_1 - 2x_2 - 3x_3 - 5x_4 \leq -11 \quad (a) \\
& 3x_1 - 5x_2 + x_3 + 6x_4 \geq 4 \quad (b) \\
& 2x_1 + x_2 + x_3 - x_4 \leq 3 \quad (c) \\
& x_1 + 2x_2 + 4x_3 + 5x_4 \leq 10 \quad (d) \\
& x \in \{0,1\}^4.
\end{aligned}
$$

(4) 列出 2^4 种情况，分别带入约束条件，判断是否可行：

解 BIP 的隐枚举法 IV

	(x_1, x_2, x_3, x_4)	(a)	(b)	(c)	(d)	目标函数值
1	(0,0,0,0)	×				
2	(0,0,0,1)	×				
3	(0,0,1,0)	×				
4	(0,0,1,1)	×				
5	(0,1,0,0)	×				
6	(0,1,0,1)	×				
7	(0,1,1,0)	×				
8	(0,1,1,1)	×				
9	(1,0,0,0)	×				
10	(1,0,0,1)	√	√	√	√	-11
11	(1,0,1,0)	×				
12	(1,0,1,1)	√	√	√	√	-14
13	(1,1,0,0)	×				
14	(1,1,0,1)	√	√	√	√	-13
15	(1,1,1,0)	√				
16	(1,1,1,1)	√	√	√	×	

解 BIP 的隐枚举法 V

- 选择不同的初始可行解，计算量不一样。
- 一般地，当求最大化问题时，首先考虑目标函数系数最大的变量等于 1。
- 当目标函数求最小值时，先考虑目标函数系数最大的变量等于 0. 对于上例，目标函数为

$$-6x_1 - 2x_2 - 3x_3 - 5x_4,$$

应将其重新排列为

$$-2x_2 - 3x_3 - 5x_4 - 6x_1,$$

再对 (x_2, x_3, x_4, x_1) 取值为

$$(0,0,0,1), (0,0,1,0), (0,0,1,1), (0,1,0,0), \ldots$$

进行逐一排查。

- 在计算过程中，当目标函数值得到改进时，应使用新的界，以减少计算量。

作业

用隐枚举法求解如下问题：

$$
\begin{aligned}
\min \quad & z = 3x_1 + 7x_2 - x_3 + x_4 \\
s.t. \quad & 2x_1 - x_2 + x_3 - x_4 \geq 1 \\
& x_1 - x_2 + 6x_3 + 4x_4 \geq 8 \\
& 5x_1 + 3x_2 + x_4 \geq 5 \\
& x \in \{0,1\}^4.
\end{aligned}
$$

提纲

指派问题 I

$$\begin{aligned} \min / \max \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n. \\ & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n. \\ & x_{ij} \in \{0, 1\}, \quad i, j = 1, 2, \ldots, n. \end{aligned}$$

称如下矩阵 C 为系数矩阵或效率矩阵：

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{pmatrix}.$$

同样可定义记号 $X = (x_{ij})_{i,j=1,2,\ldots,n}$. 则目标函数可表示为 $f(X) = \langle C, X \rangle$. 指派问题有如下性质：

指派问题 II

性质：将效率矩阵 C 的某一行或某一列加上一个任意常数 m 得到矩阵 D. 以 D 为效率矩阵的指派问题与以 C 为效率矩阵的指派问题具有相同的解。

证明：不妨设 D 是由 C 的第一行加上常数 m 得到的。令X满足可行性条件。则

$$
\begin{aligned}
f_C(X) &= \langle C, X\rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
&= (c_{11}x_{11} + c_{12}x_{12} + \ldots + c_{1n}x_{1n}) + \sum_{i=2}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
&= ((c_{11}+m)x_{11} + (c_{12}+m)x_{12} + \ldots + (c_{1n}+m)x_{1n}) \\
&\quad -m(x_{11} + x_{12} + \ldots + x_{1n}) + \sum_{i=2}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
&= \langle D, X\rangle - m \\
&= f_D(X) - m.
\end{aligned}
$$

匈牙利算法

匈牙利算法是利用上述性质求解指派问题的算法，由 W. W. Kuhn 于 1955 年提出，利用了匈牙利数学家 D. Konig 关于矩阵中独立元素的定理。该算法被称之为匈牙利算法。适用条件：

1. 最小化问题；
2. 人与事的数目相等；
3. 效率 $c_{ij} \geq 0 \ \forall i, j$.

不满足上述条件，应首先将问题进行转化以满足上述条件，之后再利用匈牙利算法求解。

匈牙利算法的一般步骤 I

以如下效率矩阵为例，说明匈牙利算法的计算步骤：

$$C = \begin{pmatrix} 4 & 8 & 7 & 15 & 12 \\ 7 & 9 & 17 & 14 & 10 \\ 6 & 9 & 12 & 8 & 7 \\ 6 & 7 & 14 & 6 & 10 \\ 6 & 9 & 12 & 10 & 6 \end{pmatrix}.$$

步骤一：变换系数矩阵。

1. 每一行都减去所在行的最小元素；
2. 在所得矩阵中，每一列都减去所在列的最小元素。

匈牙利算法的一般步骤 II

这样，系数矩阵中每行及每列都至少有一个零元素，同时不出现负元素。转步骤二。

$$C \to \begin{pmatrix} 0 & 4 & 3 & 11 & 8 \\ 0 & 2 & 10 & 7 & 3 \\ 0 & 3 & 6 & 2 & 1 \\ 0 & 1 & 8 & 0 & 4 \\ 0 & 3 & 6 & 4 & 0 \end{pmatrix} \to \begin{pmatrix} 0 & 3 & 0 & 11 & 8 \\ 0 & 1 & 7 & 7 & 3 \\ 0 & 2 & 3 & 2 & 1 \\ 0 & 0 & 5 & 0 & 4 \\ 0 & 2 & 3 & 4 & 0 \end{pmatrix} = C'.$$

位于不同行、不同列的零元素称之为独立零元素。注：

(a) 对于效率非负的指派问题，并且目标是最小化，若能在系数矩阵中找到 n 个位于不同行不同列的零元素，则对应的指派方案总费用为 0, 从而一定是最小的（因为效率非负）.

匈牙利算法的一般步骤 III

(b) 在选择零元素时，当同一行（或列）上有多个零元素时，如选择其一，则其余的零元素就不能再被选择，而成为多余的（被选中的零元素的位置对应的x_{ij}将被赋值为 1. 因为一个人只能做一件事，而一件事也只能由一个人来做）。因此，关键并不在于有多少个零元素，而要看它们是否恰当地分布在不同行和不同列上，即独立零元素的数目。

步骤二：在变换后的效率矩阵中确定独立零元素。方法如下：

(c) 为了确定独立零元素，可以在只有一个零元素的行（或列）中加圈（标记为◎），这表示此人只能做该事，或此事只能由该人来做。每圈出一个0，同时要把位于同列（或同行）的其他零元素划去（标记为∅），表示此事已不能再由其他人来做（或此人已不能做其他事）。如此反复进行，直至系数矩阵中所有的零元素都被圈去或划去为止。在此过程中，如遇到在所有的行和列中，零元素都不止一个的情况，可任意选取其中一个0元素加圈，同时划去其同行和同列中其他零元素。当过程结束时，被画圈的元素即是独立零元素。

匈牙利算法的一般步骤 IV

$$C' = \begin{pmatrix} \emptyset & 3 & \circledcirc & 11 & 8 \\ \circledcirc & 1 & 7 & 7 & 3 \\ \emptyset & 2 & 3 & 2 & 1 \\ \emptyset & \circledcirc & 5 & \emptyset & 4 \\ \emptyset & 2 & 3 & 4 & \circledcirc \end{pmatrix}$$

1. 若独立零元素为 n 个，则已得到最优解（独立零元素所对应的变量 x_{ij} 取值为 1, 其余所有的 x_{ij} 都取值为 0）.
2. 若独立零元素的个数不足 n 个，表示还不能确定最优指派方案，此时需画出能覆盖所有零元素的最少数目的直线集合，然后转步骤三。

匈牙利算法的一般步骤 V

画出覆盖所有零元素的最少数目的直线集合：

1. 对没有 ⊙ 的行打“√".
2. 在已经打 √ 的行中，对“∅" 所在的列打 √.
3. 在已经打 √ 的列中，对“⊙" 所在的行打 √.
4. 重复 2 和 3, 直到再也不能找到可以打 √ 的行或者列为止.
5. 对没有打 √ 的行画横线（删除线），对已经打 √ 的列画垂线（删除线）.

（对于此例，第二、三行打 √, 第一列打 √. 最后，第一、四、五行、第一列画删除线）

步骤三：继续变换系数矩阵。方法是在未被直线覆盖的元素中找出最小的元素。对未被直线覆盖的元素所在的行（或列）中各元素（包括被划去的元素）都减去这一最小元素。这样，在未被直线覆盖的元素中会出现新的零元素，同时已被直线覆盖的元素中

匈牙利算法的一般步骤 VI

出现负数。为了消除负数，在所在的列都加上该最小元素即可。返回步骤二。

$$C' = \begin{pmatrix} 0 & 3 & 0 & 11 & 8 \\ 0 & 1 & 7 & 7 & 3 \\ 0 & 2 & 3 & 2 & 1 \\ 0 & 0 & 5 & 0 & 4 \\ 0 & 2 & 3 & 4 & 0 \end{pmatrix} \to \begin{pmatrix} 1 & 3 & 0 & 11 & 8 \\ 0 & 0 & 6 & 6 & 2 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 0 & 5 & 0 & 4 \\ 1 & 2 & 3 & 4 & 0 \end{pmatrix} = C''.$$

回到步骤2，对C''加圈。

$$C'' = \begin{pmatrix} 1 & 3 & \circledcirc & 11 & 8 \\ \emptyset & \circledcirc & 6 & 6 & 2 \\ \circledcirc & 1 & 2 & 1 & \emptyset \\ 1 & \emptyset & 5 & \circledcirc & 4 \\ 1 & 2 & 3 & 4 & \circledcirc \end{pmatrix}.$$

匈牙利算法的一般步骤 VII

已经有5个独立零元素，得到最优解：

$$x = \begin{pmatrix} & & 1 & & \\ & 1 & & & \\ 1 & & & & \\ & & & 1 & \\ & & & & 1 \end{pmatrix}.$$

非标准形式的指派问题 I

1. 最大化指派问题：设最大化指派问题的效率矩阵为 $C = (c_{ij})_{n\times n}$, 其中最大元素为 m. 令矩阵$B = m - C$. 则以 B 为系数矩阵的最小化指派问题和以 C 为系数矩阵的原最大化指派问题有相同的最优解，同时 $B \geq 0$.
2. 一个人可做几件事的指派问题：若某个人可做 k 件事，则将该人化作相同的 k 个“人”来接受指派。这 k 个“人”做同一件事的费用系数当然都一样。
3. 若人多事少，则添加上一些虚拟的“事”。这些虚拟的“事”被各个人做的费用系数取为 0, 理解为这些虚拟的事是非常容易的，任何人都可以完美的完成。

非标准形式的指派问题 II

4. 若人少事多（例如m人n事，$m < n$）
   - 添加$n - m$个虚拟的“人”。这些虚拟的人做各事的费用系数取做0，理解为这些虚拟的人可以不需要任何代价并且完美的完成任何一件事。此时，指派结果会筛选出m件事指派给m人，其余$n - m$ 件事将不会被指派。
   - 若第i人可以做t_i件事，则将第i人的价值系数拷贝t_i份。若最后行数多于列数，则添加零列（不需要做的虚拟的事）使得行列数相等；若最后行数仍少于列数，则添加零行（无所不能的人），最终只能是有些事不会被指派。
5. 某事一定不能由某人做的指派问题：若某事一定不能由某人来做，则可将相应的费用系数取做足够大的数 M.

例子 I

把5件事指派给3个人，每人最多只能做两件事，费用系数矩阵如下(每行对应一人，每列对应一事)，如何指派?

$$C = \begin{pmatrix} 4 & 8 & 7 & 15 & 12 \\ 7 & 9 & 17 & 14 & 10 \\ 6 & 9 & 12 & 8 & 7 \end{pmatrix}.$$

解：因为每人最多做两件事，可将每个人化身为相同的两个人，得到新的费用系数矩阵如下

$$C' = \begin{pmatrix} 4 & 8 & 7 & 15 & 12 \\ 4 & 8 & 7 & 15 & 12 \\ 7 & 9 & 17 & 14 & 10 \\ 7 & 9 & 17 & 14 & 10 \\ 6 & 9 & 12 & 8 & 7 \\ 6 & 9 & 12 & 8 & 7 \end{pmatrix}.$$

1-2行对应第一人，3-4行对应第2人，5-6行对应第3人。

例子 II

现在是：6人5事，故可以创造一个虚拟的事，任何人都不需要花费任何代价就可以完成的事：

$$C'' = \begin{pmatrix} 4 & 8 & 7 & 15 & 12 & 0 \\ 4 & 8 & 7 & 15 & 12 & 0 \\ 7 & 9 & 17 & 14 & 10 & 0 \\ 7 & 9 & 17 & 14 & 10 & 0 \\ 6 & 9 & 12 & 8 & 7 & 0 \\ 6 & 9 & 12 & 8 & 7 & 0 \end{pmatrix}.$$

最后一列对应虚拟的事。之后再用匈牙利算法。

Totally Unimodular

指派问题的约束条件非常特殊。把其中的等式约束条件写成 $Ax = b$ 的形式, 其中

$$x = (x_{11}, x_{21}, \ldots, x_{n1}, x_{12}, x_{22}, \ldots, x_{n2}, \ldots, x_{1n}, x_{2n}, \ldots, x_{nn})^T.$$

矩阵 A 的元素 $a_{ij} \in \{0, 1\}$. 并且，矩阵 A 的任意子方阵的行列式为 0, 1 或 -1. 因此，矩阵 A 是 totally unimodular 的。据此可证明，指派问题等价于如下线性规划：

$$\begin{aligned} \min / \max \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, 2, \ldots, n. \\ & \sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, 2, \ldots, n. \\ & 0 \le x_{ij} \le 1, \quad i, j = 1, 2, \ldots, n. \end{aligned}$$

作业

求解下列费用系数矩阵的最小化指派问题：

$$\begin{pmatrix} 10 & 11 & 4 & 2 & 8 \\ 7 & 11 & 10 & 14 & 12 \\ 5 & 6 & 9 & 12 & 14 \\ 13 & 15 & 11 & 10 & 7 \end{pmatrix}, \quad \begin{pmatrix} 3 & 6 & 2 & 6 \\ 7 & 1 & 4 & 4 \\ 3 & 8 & 5 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 2 & 4 & 3 \\ 5 & 7 & 6 & 2 \end{pmatrix}$$

提纲

贪婪算法

贪婪算法是求解问题的一种思路，不同问题表现形式不同，每步都采取（当前）最优的做法，其优点是简单。这里每步的最优是局部的最优，最终得到的解可能是全局最优，也可能不是。

教室调度问题

假如有如下课表，你希望将尽可能多的课程安排在某间教室。

课程	开始时间	结束时间
美术	9:00AM	10:00AM
英语	9:30AM	10:30AM
数学	10:00AM	11:00AM
计算机	10:30AM	11:30AM
音乐	11:00AM	12:00PM

如何选出尽可能多且时间不冲突的课程？贪婪算法如下：

1. 选出结束最早的课，它就是要在这间教室上的第一堂课。
2. 接下来，必须选择第一堂课结束后才开始的课。同样，选择结束最早的课作为在该教室上的第二堂课。

答案：美术、数学、音乐；并且这样找到的解是最优的。

背包问题

一个贪婪的小偷，背着容量为35个单位的背包，在商场伺机盗窃各种可装入背包的商品。小偷力图往背包中装入价值最高的商品，使用那种算法那？贪婪算法非常简单，具体步骤如下：

1. 盗窃可装入背包的最贵重的商品；
2. 再盗窃还可以装入背包的最贵重的商品，以此类推。

例设可盗窃的商品有如下三种：

物品	价值	体积
音响	3000	30
笔记本电脑	2000	20
吉他	1500	15

贪婪算法给出的答案：音响（价值3000，体积30）。不是最优解，但比较接近。很多时候，只需要找到一个能够大致解决问题的算法，贪婪算法非常合适，因为它实现起来非常容易，得到的解与最优解接近。

集合覆盖问题 I

在m个候选地点上建设服务中心，服务某市的n个居民区，建成后的每个服务中心都能够覆盖若干个居民区。假设建设每个服务中心的费用是相同的，如何确定在哪些候选地点上建设服务中心使得总费用最小（等价于建设尽可能少的服务中心），同时所有n个居民区均被覆盖到？

可能的服务中心集合有2^m个，全部列出后再在其中选择能够覆盖n个居民区的集合，计算量太大。假设每秒钟可计算10个子集，所需时间如下：

m	时间
5	3.2s
10	102.4s
32	13.6y
100	4×10^{21}y

集合覆盖问题是NP-完全问题，没有任何算法可以足够快地解决该问题（除非 $P = NP$）！

集合覆盖问题 II

使用贪婪算法可得到近似解（不太坏的解），步骤如下：

1. 选出这样一个候选地点，并建设服务中心，该服务中心能够服务最多的尚未被覆盖的居民区（即便该服务中心覆盖了一些已经被覆盖了的居民区）。
2. 重复第一步，直到所有的居民区都被覆盖。

贪婪算法是一种近似算法(approximation algorithm)，在获得精确解（最优解）需要的时间太长时，可使用近似算法。判断近似算法优劣的标准如下：

1. 速度是否快
2. 得到的近似解与最优解的接近程度。

贪婪算法不仅简单，而且通常运行速度很快。在集合覆盖问题中，贪婪算法的运行时间为$O(m^2)$ (why?).

NP-完全问题

为解决集合覆盖问题，需要考虑所有可能的子集的集合，当m变大时，待检查的集合数目指数增长。

再比如，可行解的数目同样增长非常快的还有旅行商（货郎担、TSP）问题。给定 m 个城市，不同城市间有不同的距离。如何排列城市的顺序，使得从某一个城市出发经过每个城市一次且仅一次，最后回到出发的城市，使得总距离最短？

1. 若第一个城市可以任意选，则有 $m!$ 种可行的方案;
2. 若第一个城市可以任意选，则有 $(m-1)!$ 种可行的方案.

$(5! = 120, 10! = 3628800)$

旅行商问题和集合覆盖问题的共同之处是：需要在所有的可行方案中一一检查才能确保找到最优解，都属于NP完全问题。NP完全问题非常难解，很多非常聪明的脑袋都认为，根本不可能找到可以快速求解这类问题的算法。

如何识别NP-完全问题 I

假设你是一位橄榄球教练，正在为球队挑选队员。作为一名优秀的教练，你对球队提出了一些基本要求：优秀的四分卫、优秀的跑位，擅长雨中作战，能承受压力等等。候选球员及能力如下：

球员	能力
1	跑卫
2	外接手/能够承压
3	四分卫/雪中送碳
...	...

组建的球队要满足所有的要求，但名额有限。这是一个集合覆盖问题，是NP-完全问题！贪婪/近似算法如下：

1. 找出符合最多要求的球员；
2. 找出符合最多的尚未被满足的要求的球员，不断重复该过程，直到球队满足要求（或名额已满）。

如何识别NP-完全问题 II

NP-完全问题无处不在！若能判断出要解决的问题属于此类就好了，这样就不用去追求完美解决方案了，而是使用近似算法寻求近似解即可。然而，要判断一个问题是不是NP完全问题很难！易于解决的问题(P问题)和NP完全问题的差别往往很小。例如：

- 最短路问题是P问题：在一个网络中，寻找从A点到B点的最短路。但是，如果要找出经由指定几个点的最短路，则是NP-完全问题，因为它包括了旅行商问题。
- 最大流/最小割问题是P问题（线性规划），然而最大割问题是NP-完全问题。

如何识别NP-完全问题 III

简言之，没有简答的办法判断一个问题是不是NP完全问题，但还是有一些蛛丝马迹可循：

- 元素较少时算法运行速度很快，但随着元素数量的增加，速度会变的非常慢；
- 设及“所有可能的组合”的问题通常是NP完全问题；
- 不能将问题分成小问题，必须考虑各种可能的情况，则可能是NP-完全问题；
- 如果问题设及序列（如旅行商问题中的城市序列）且难以解决，则可能是NP-完全问题；
- 如果问题设及集合（如集合覆盖问题）且难以解决，则可能是NP-完全问题；
- 如果问题可转换为集合覆盖问题或者旅行商问题，则肯定是NP-完全问题.

如何识别NP-完全问题 IV

NP-完全问题的例子：

- 有个邮递员负责给n个家庭送信，需要找出经过这n个家庭的最短路径。
- 在一堆人中找出最大的朋友圈（即其中任何两人都认识）。
- 要制作衣一副中国地图，需要使用不同的颜色标出相邻的省份。为此，需要确定最少需要使用多少种颜色，才能确保任何两个相邻的省份的颜色都不同。

提纲

动态规划

动态规划是一种解决棘手问题的方法，它将大问题分成小问题，先着手解决小问题，以逐步达到解决大问题（原问题）的方法。与线性规划等不同的是，动态规划是一种解决问题的思想，不是某一具体数学模型。

- 动态规划可帮助在给定约束条件下找到最优解（例如：背包问题中的包容量）.
- 在问题可分解为彼此独立且离散的子问题时，就可以使用动态规划来解决.
- 每种动态规划解决方案都设及网格.
- 网格中的值通常就是要优化的值（例如：背包问题中商品的价值）.
- 每个单元格都是一个子问题，因此应考虑如何将问题分解成子问题，有助于找出网格的坐标轴.

背包问题 I

小偷的背包容量为4个单位，供待选的物品、价值及体积如下：

物品	价值	体积
音响	3000	4
笔记本电脑	2000	3
吉他	1500	1

- 简单算法：尝试各种可能的组合，共计2^n种情况。
- 贪婪算法：近似算法，得到的不一定是最优解。
- 动态规划法：先解决小问题，再逐步解决大问题，最终得到全局最优解。

背包问题 II

制作表格，每一行对应一件物品，每列为不同容量的背包：

	容量1	容量2	容量3	容量4
吉他(G)				
音响(S)				
笔记本电脑(L)				

逐行填入数字，表格填完后，解题完成。

- 第一行为吉他行，意味着小偷尝试将吉他装入背包。在每个单元格，都需要做一个决定：偷还是不偷？（目标：找出价值最高的物品组合）。与在该行，只有吉他供选择。与这个单元格一样，每个单元格都将包含当前可装入背包的所有商品。

	容量1	容量2	容量3	容量4
吉他	1500, G	1500, G	1500, G	1500, G
音响				
笔记本电脑				

背包问题 III

如果只有一个吉他供选择，可装入其中的商品的最大价值为1500.

- 音响行：此时可供选择的商品为1个吉他和1个音响。在每一行，可偷的商品都为当前的商品以及之前各行对应的商品。因此，当前还不能选择笔记本电脑。

	容量1	容量2	容量3	容量4
吉他	1500, G	1500, G	1500, G	1500, G
音响	1500, G	1500, G	1500, G	3000, S
笔记本电脑				

- 电脑行：同样的方式处理。

	容量1	容量2	容量3	容量4
吉他	1500, G	1500, G	1500, G	1500, G
音响	1500, G	1500, G	1500, G	3000, S
笔记本电脑	1500, G	1500, G	2000, L	3500, L & G

背包问题 IV

用$A[i][j]$ 表示第i 行第j 列填入的值，其中i 表示行，j 表示列。对此例：$i = 1, 2, 3$, $j = 1, 2, 3, 4$. 则

$$A[i][j] = \max(A[i-1][j], \text{当前商品的价值} + A[i-1][j-\text{当前商品的体积}]).$$

其中$A[i-1][j]$为第i 行所对应的商品可供选择之前，第j 列容量的背包可装入商品的最大价值。$A[i-1][j-$ 当前商品的体积$]$ 为选择了第i 行所对应的商品后，剩余容量可装入的最大价值。

背包问题动态规划法 FAQ I

- 再增加一件可供选择的商品，例如iPhone (价值2000, 体积1). 此时不需要重新计算前面的表格，仅需要将iPhone放到最后一行，继续填写表格即可。

	容量1	容量2	容量3	容量4
吉他	1500, G	1500, G	1500, G	1500, G
音响	1500, G	1500, G	1500, G	3000, S
笔记本	1500, G	1500, G	2000, L	3500, L & G
iPhone	2000, i	3500, i & G	3500, i & G	4000, i & L

练习：

- 每一列从上往下，最大价值会降低吗？
- 若还有一件MP3 (价值1000, 体积1), 要偷吗？
- 行的排列顺序发生变化时，结果会发生变化吗？

背包问题动态规划法 FAQ II

- 增加一件更小的商品将如何呐？如多出一个可供选择的项链 (价值1000, 容量0.5). 前面的商品体积都是整数，但若选择偷项链，则余下容量3.5. 在容量为3.5的背包中可装入的商品最大价值是多少？由于项链的加入，需要考虑粒度更细的表格：0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4.
- 可以偷走商品的一部分吗？若行窃地点为杂货店，有成袋的扁豆和大米供选择，整袋装不下，可打开包装，再将背包倒满。在这种情况下，不再是要么偷，要么不偷，而是可以偷商品的一部分。此时，动态规划失效！使用动态规划，要么拿走整件商品，要么不拿，而没有办法判断该不该拿走商品的一部分。使用贪婪算法可以轻松处理这种情况: 首先尽可能多的拿走价值最高的商品；如果拿光了，再尽可能多地拿走价值次高的商品，以此类推。

旅游行程最优化 I

假设你要去伦敦度假，假期两天，但想去的地方很多，无法前往每个地方游览。因此，列表单如下：

名胜	时间	评分
某著名教堂 (W)	0.5天	7
环球剧场 (G)	0.5天	6
国家美术馆 (N)	1天	9
大英博物馆 (B)	2天	9
圣保罗大教堂 (S)	0.5天	8

根据此清单，如何做决定？这是一个背包问题！约束条件不是背包容量，而是有限的时间；不是决定该装入哪些商品，而是决定该选择哪些名胜。如何用动态规划问题解决此问题？

旅游行程最优化 II

- 动态规划可以处理相互依赖的情况吗？假设你还想去巴黎，因此在前述清单中又加入几项：

名胜	时间	评分
埃菲尔铁塔	1.5天	8
卢浮宫	1.5天	9
巴黎圣母院	1.5天	7

去这些地方游览需要很长时间，因为你要先从伦敦前往巴黎，耗时0.5天；而到达巴黎后，游览每个地方的时间降为1天；将埃菲尔铁塔装入“背包”后，卢浮宫和巴黎圣母院将变的更便宜。如何使用动态规划建模？

没办法建模！动态规划功能强大，它能通过解决小问题逐步解决大问题。但是，仅当每个子问题都是离散的、且是相互独立的，动态规划才有效。

背包问题的动态规划法 FAQ

- 最优解可能导致背包没装满吗？完全可能！假设可供选择偷窃的还有一个钻石，个头非常大，占去3.5单位的容量，价值100万，比其他商品都值钱多了！毫无疑问，肯定偷！此时，剩余的0.5单位容量就什么都装不下了。

练习

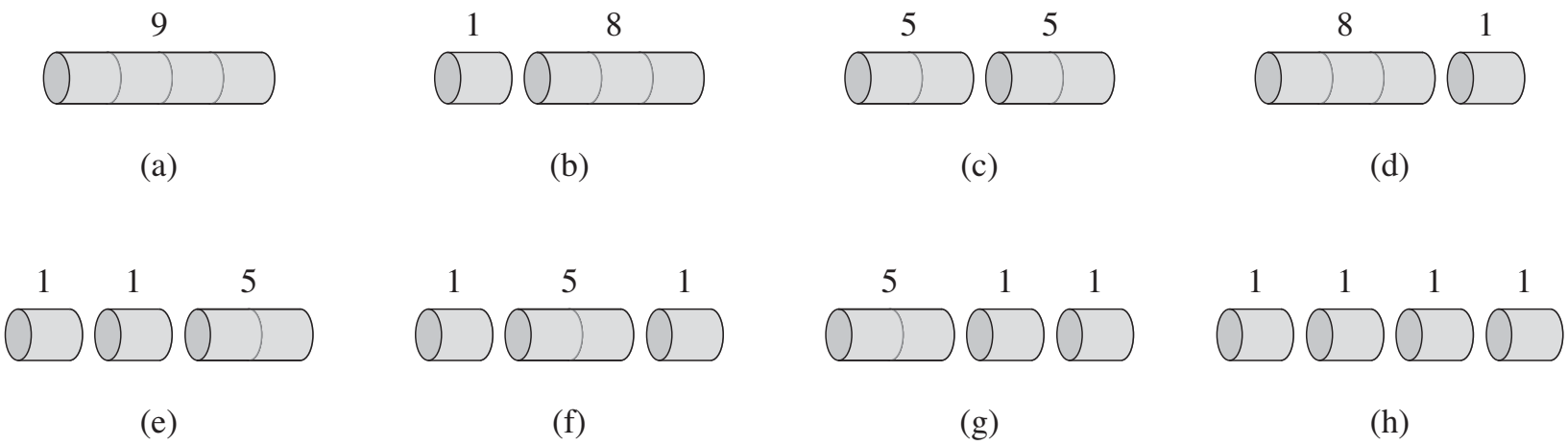
假设你要去野营，有一个容量为6的背包，需要决定该携带下面哪些东西。其中每样东西都有相应的价值，价值越大意味着越重要：

物品	价值	体积
水	10	3
书	3	1
食物	9	2
夹克	5	2
相机	6	1

钢条切割问题

某厂家想要将一根长度为n的钢条切开销售，每种钢条的长度i所对应的价格p_i给定$(i = 1, 2, ..., n)$。怎样切割，厂家才能获得最大的利润？这里要求切割前后钢条的长度均为整数。

例子： 假设一根钢条的长度为4，那么我们有8种方法来切割这根钢条，如下图所示：



每段钢条所对应的价格如下：

长度i	1	2	3	4
价格p_i	1	5	8	9

钢条切割问题

逐一检查我们就可以判断出切割的最优策略是 (c), 即切割成两段长度为2的钢条就可以获得最大利润10。

最优解的特点：在长度为n的钢条想获得最优分割时，可能无需分割，整体卖出就可获得最大利润；但只要需要分割，在第一次分割之后，会产生两个较短的钢条。这两个较短的钢条必须也被最优分割，组合之后才能得到原钢条的最优分割。

矩阵链乘法

给定一列矩阵 A_i $(i = 1, 2, \ldots, n)$, 矩阵 A_i 的维数是 $p_{i-1} \times p_i$, $i = 1, ..., n$, 给它们的乘积 $A_1 A_2 ... A_n$ 添加适当的括号，使得计算时所需的标量乘法次数最少。

假设我们需要计算A_1, A_2, A_3 的乘积$A_1 A_2 A_3$，这三个矩阵的维数分别是$2 \times 10, 10 \times 3, 3 \times 30$。我们有两种添加括号的方式，分别为：

1. $((A_1 A_2)A_3)$：即先把A_1, A_2相乘，然后把结果和A_3相乘。则所需乘法次数为 $2 \times 10 \times 3 + 2 \times 3 \times 30 = 240$。

2. $(A_1(A_2 A_3))$：即先把A_2, A_3相乘，然后把结果和A_1相乘。则乘法的次数为$10 \times 3 \times 30 + 2 \times 10 \times 30 = 1500$。

可以看出，矩阵链相乘时的先后顺序不同，运算量也不同，而且差别很大。我们的目的就是找到最好的运算次序，求出最小的运算量。

矩阵链乘法

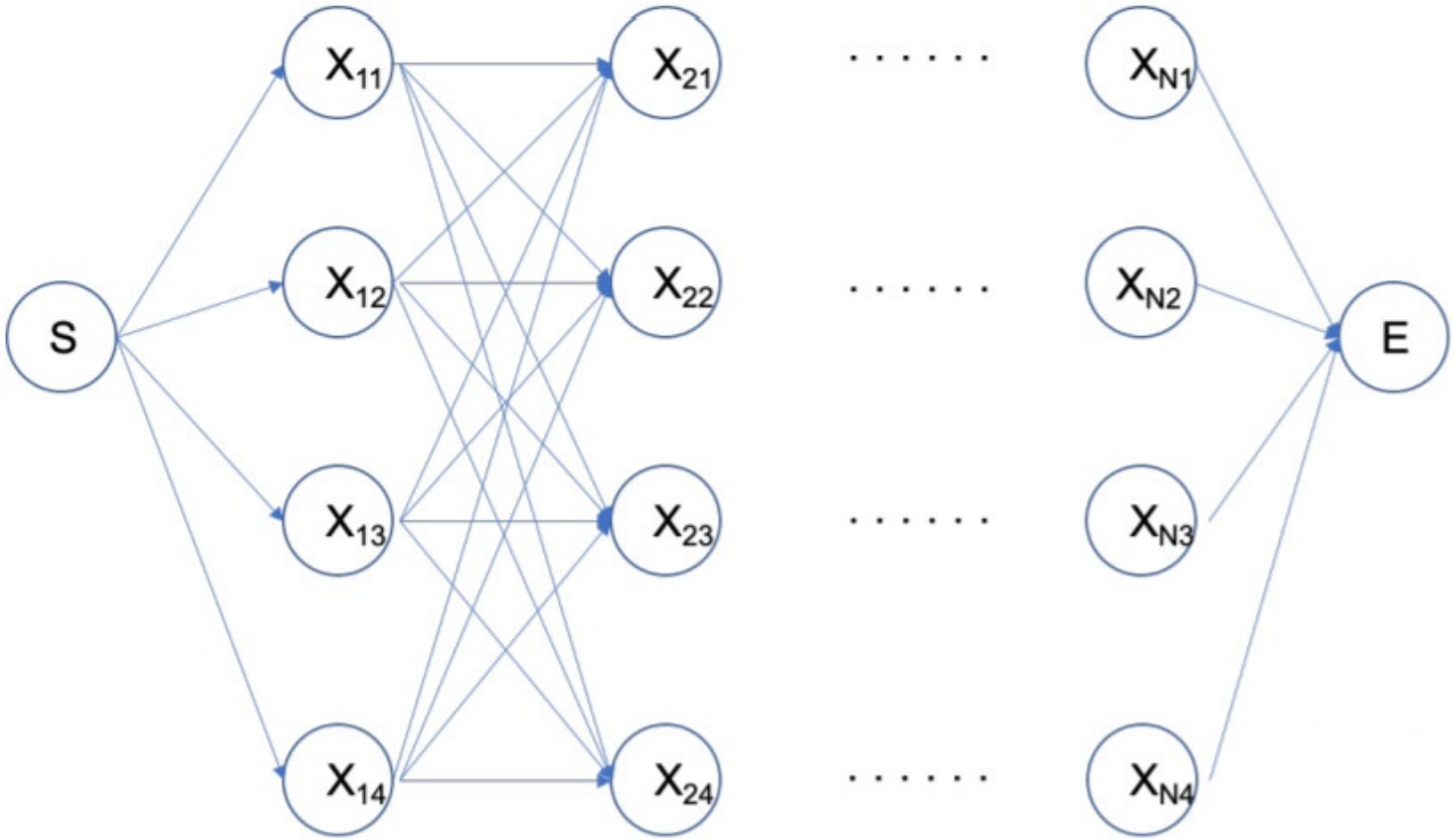
最优解的特点： 按最优的顺序计算 $A_1A_2...A_n$ 时，一定存在 $k(1 \le k \le n-1)$ 使得上述计算的最后一步是 $(A_1...A_k)$ 乘上$(A_{k+1}...A_n)$.

在矩阵链$A_1A_2...A_n$的最优括号化方案中，对子链$A_1...A_k$进行的括号化，应该就是独立求解它时的最优括号化方案。如果不是这样，我们可以将独立求解子链$A_1...A_k$的最优括号化方案带入矩阵链$A_1A_2...A_n$的最优括号化方案来代替原来它对子链$A_1...A_k$采取括号化的方案，这样就可以得到对矩阵链$A_1A_2...A_n$更好的括号化，产生了矛盾。

对子链$A_{k+1}...A_n$，我们也有类似的结论：在原问题$A_1A_2...A_n$的最优括号化方案中，对子链$A_{k+1}...A_n$进行括号化的方法，就是它自身的最优括号化方案。

其他常见的可用动态规划求解的问题

- 最长公共子串问题
- 最优二叉搜索树
- Lattic Network Shortest Path Problem



- 期中考试时间：11月26日14:00-16:00
- 地点：仙 I-212, 仙 I-213
- 11 月 28 日周四，停课一次

提纲

软件

解整数线性规划问题的软件：

- CPLEX: 提供灵活的高性能优化程序，解决线性规划(Linear Programming)、二次方程规划(Quadratic Programming)、二次方程约束规划(Quadratically Constrained Programming) 和混合整型规划(Mixed Integer Programming) 问题。
- Gurobi for solving mixed integer linear and quadratic programming problems.

软件

Matlab 中解混合整数线性规划问题的程序“intlinprog" (方法：分支定界，个平面), 求解问题形式：

$$\begin{aligned} \min_{x} \quad & f^T x \\ \text{s.t.} \quad & Ax \leq b \\ & Cx = d \\ & lb \leq x \leq ub \\ & x(intcon) \text{ are integers} \end{aligned}$$

调用格式：

x = intlinprog(f,intcon,A,b,C,d,lb,ub,options).

Examples in Matlab

- Mixed-Integer Linear Programming Basics
- Travelling Salesman Problem
- Solve Sudoku Puzzles Via Integer Programming
- Mixed-Integer Quadratic Programming Portfolio Optimization
- ...

Further reading

Alexander Schrijver, A Course in Combinatorial Optimization,
https://homepages.cwi.nl/~lex/