运筹学通论I

胡晓东

应用数学研究所

中国科学院数学与系统科学研究院

http://www.amt.ac.cn/member/huxiaodong/index.html

5. 组合优化-算法设计技巧

精确算法

分而治之 (搜索、排大小序、旅行商)
动态规划 (最短路、三角剖分、背包)
分支定界 (整数线性规划、旅行商、工件排序)

近似算法或启发式算法

贪婪策略 (最小生成树、最大可满足、背包、
顶点覆盖、独立集、旅行商)
局部搜索(最大匹配、旅行商、最大割)
序贯法 (工件排序、装箱、顶点着色)
整数规划法 (顶点覆盖、最大可满足、最大割)
随机方法 (最小割、最大可满足、顶点覆盖)
在线算法 (页面调度、k-服务器、工件排序、装箱)
不可近似 (最大团、背包、旅行商、装箱、连通控制集等)

5.3 分支定界

分支定界方法从本质上讲是一类**分而治之**方法，它与**动态规划**方法也十分类似。给定一个优化问题的实例 **I**，它保存一些子实例 $\mathbf{I}_i$ 的集合 ***S***，使得：

子实例 $\mathbf{I}_i$ 的可行解集的并等于原实例 **I** 的可行解集。

初始时，***S*** 仅包含实例 **I**。然后，对问题的实例 **I** 进行一个分解运算，称为**分支**，得到一组互不相交的子实例 $\mathbf{I}_i$。我们用一棵**判定树**来刻画这个过程，树上的每一个节点都对应一个子实例。

这个方法的关键是，对产生的新的子实例运用**回溯技巧**，并保存和更新当前已得到的最优解的信息，从而减少需要考虑和求解的子实例的数目。

5.3 分支定界（续一）

对每一个子实例 $\mathbf{I}_i$，当我们得到它的最优值 $c_{\text{opt}}(\mathbf{I}_i)$ 的一个下界 $c(\mathbf{I}_i)$ 以后，如果 $c(\mathbf{I}_i)$ 比已经得到的最好值要大，那么就没有必要对 $\mathbf{I}_i$ 进行分支运算了，这是因为原实例 $\mathbf{I}$ 的最优解不可能在子实例 $\mathbf{I}_i$ 的可行解集中。此时，我们称 $\mathbf{I}_i$ 被除掉了。否则我们还需要对 $\mathbf{I}_i$ 再进行分支运算，得到若干子子实例。

我们可以引入一个如下控制机制，用来确定那些分支不会产生比目前已经得到最好解更好的解；并停止对这些分支进一步分支，然后回溯到分支所对应的父辈节点（分支）。

5.3 分支定界（续二）

ALGORITHM 6.1 *Branch-and-Bound Algorithm*

Active set $S := \{I_0\}$. (I_0 is the input instance.)
Upper bound $U := \infty$.
Current best solution $s := \emptyset$.
while $S \neq \emptyset$ **do**
    choose an instance $I_i \in S$ as a branching instance,
    $S := S \setminus \{I_i\}$.
    decompose I_i into subinstances $I_{i1}, I_{i2}, \cdots, I_{ik}$,
    compute the lower bounds $\underline{c}(I_{ij})$ for $j = 1, 2, \cdots, k$.
    for $j = 1, 2, \cdots, k$ **do**
        if $\underline{c}(I_{ij}) \geq U$ **then** kill I_{ij}
        else if I_{ij} is a feasible solution to I_0 **then** $U := \underline{c}(I_{ij})$ and $s := I_{ij}$
            else $S := S \cup \{I_{ij}\}$.
    end-for
end-while
return s.

5.3 整数线性规划

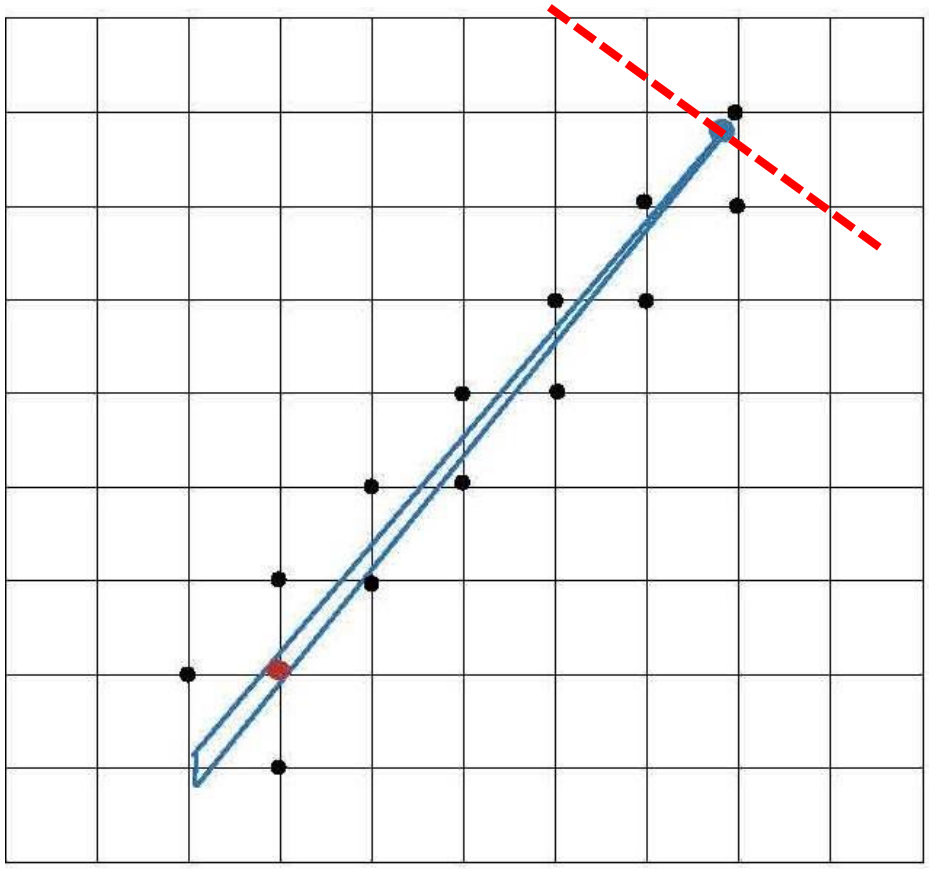
因为大多数的组合优化问题都可以化为整数线性规划问题，所以很多学者花费了很多精力研究求解整数线性规划问题的最优解的性质和求解方法。分支定界算法是最有效的方法之一。

(P_0) **Min** $z = cx$
s.t. $Ax \le b$,
$x \in Z^n$

线性规划松弛 **Min** $z = cx$
s.t. $Ax \le b$

一般的整数线性规划（即使是特例，**0-1 规划**）是**NP-**难解的。 然而，如果不要求变量 x 是整数向量，那么相应的问题就是一个线性规划问题，称为**线性规划松弛**。注意，线性规划问题是多项式时间可解的。

5.3 整数线性规划（续一）



整数线性规划的最优解与其松弛所得线性规划的最优解，可能相差甚远。

5.3 整数线性规划（续二）

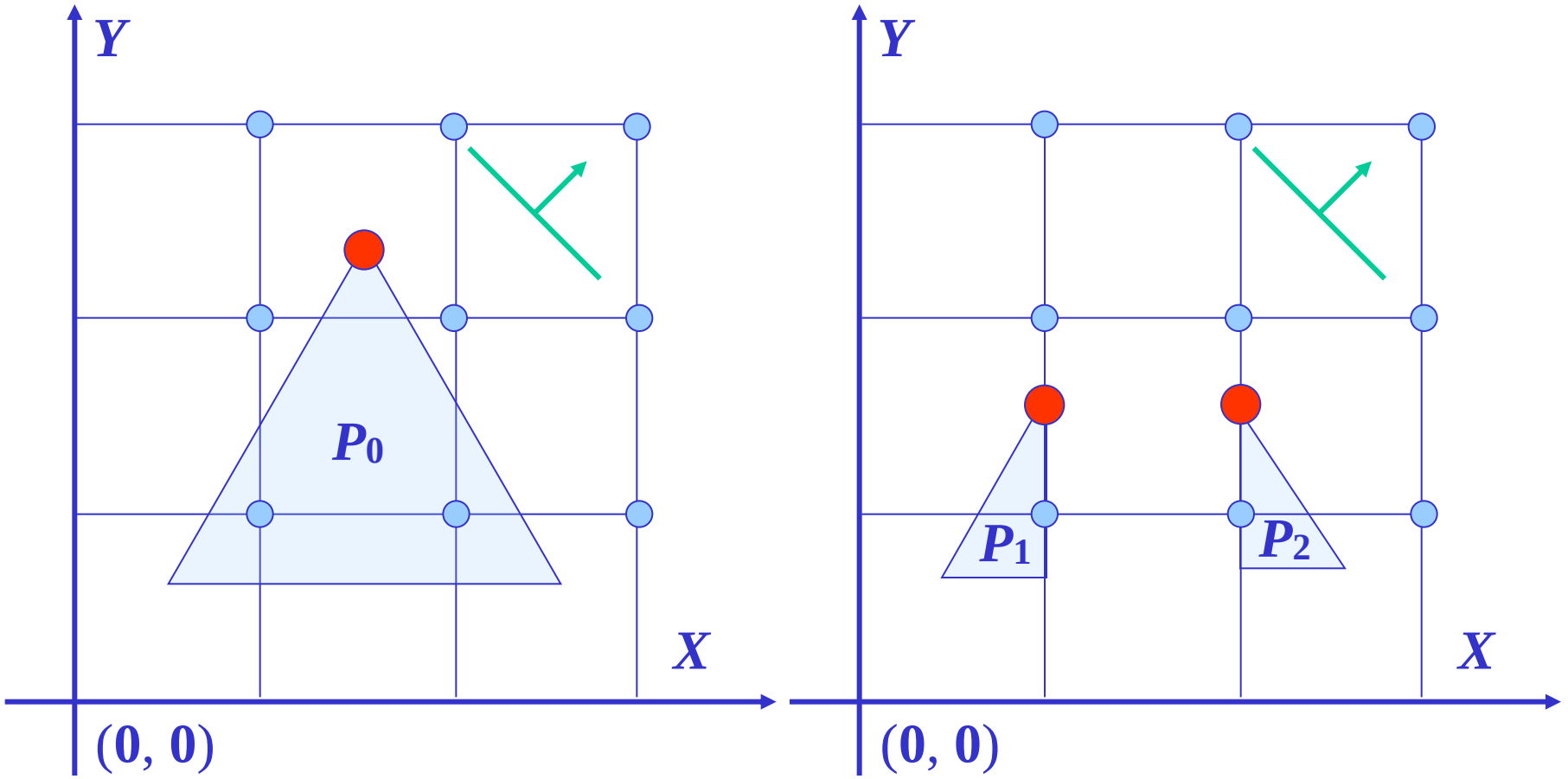
一个简单，但是十分重要，的事实是，如果我们解得线性规划松弛问题的解 x_0，且它是整数解，那么我们就找到了原问题 P_0 的一个最优解；假如 x_0 不是整数解，但是它的函数值 $c^T x_0$ 可以作为问题 P_0 的最优解的目标函数值的一个下界。这就是用分支定界方法求解整数线性规划问题的基本思想，该算法称为 **割平面法** (1958)。我们现在可以在线性规划松弛问题的约束中再增加一个约束条件，将原问题 P_0 分解为两个约束集合不交的子问题，且原问题 P_0 的可行解仍然是新问题的一个可行解：

$$(P_1) \quad \text{Min} \quad z = cx$$
$$\text{s.t.} \quad Ax \le b,$$
$$x_i \le \lfloor x_{i0} \rfloor$$

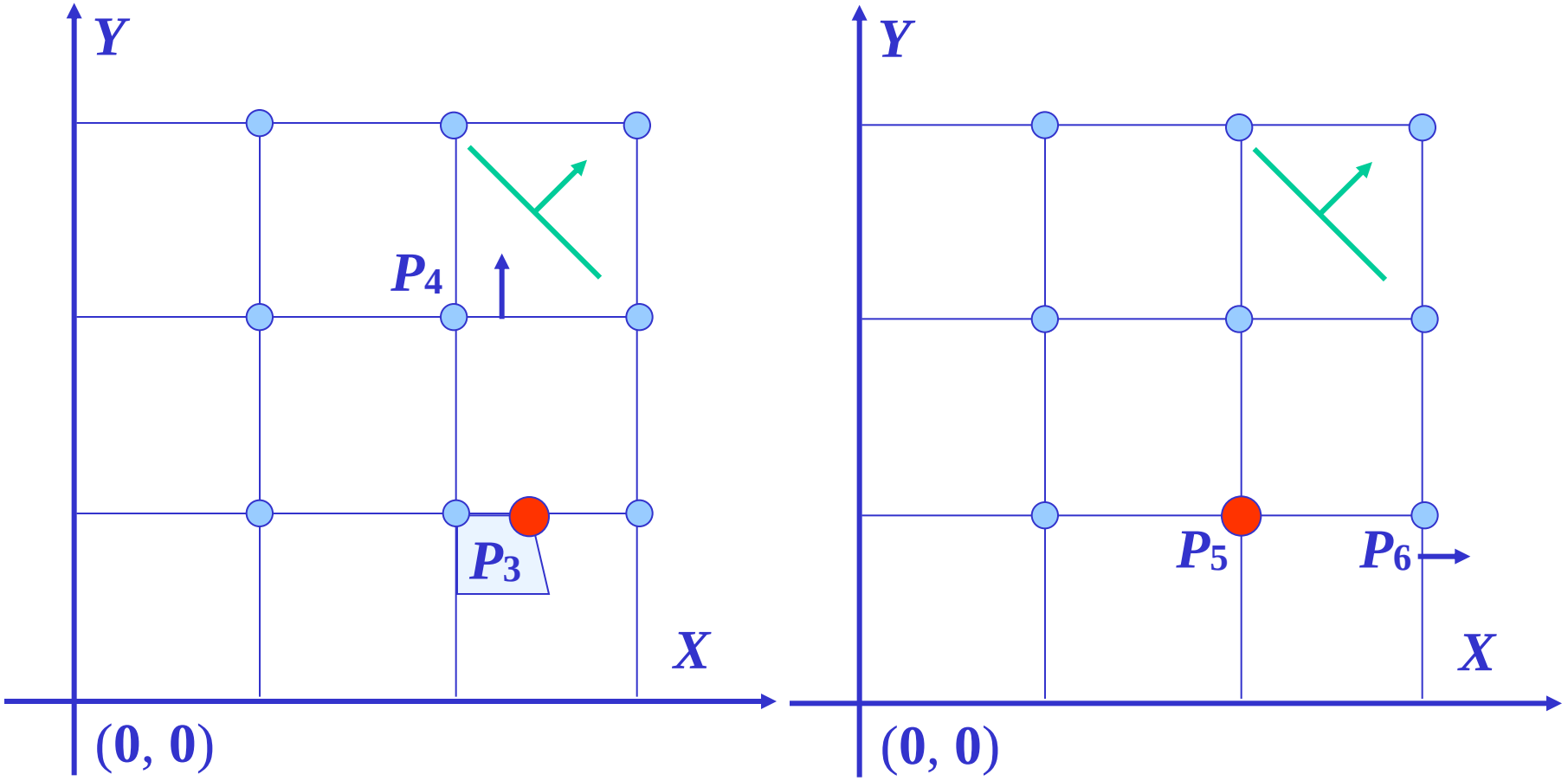
$$(P_2) \quad \text{Min} \quad z = cx$$
$$\text{s.t.} \quad Ax \le b,$$
$$x_i \ge \lfloor x_{i0} \rfloor + 1$$

5.3 整数线性规划（续三）

Min –(x + y)

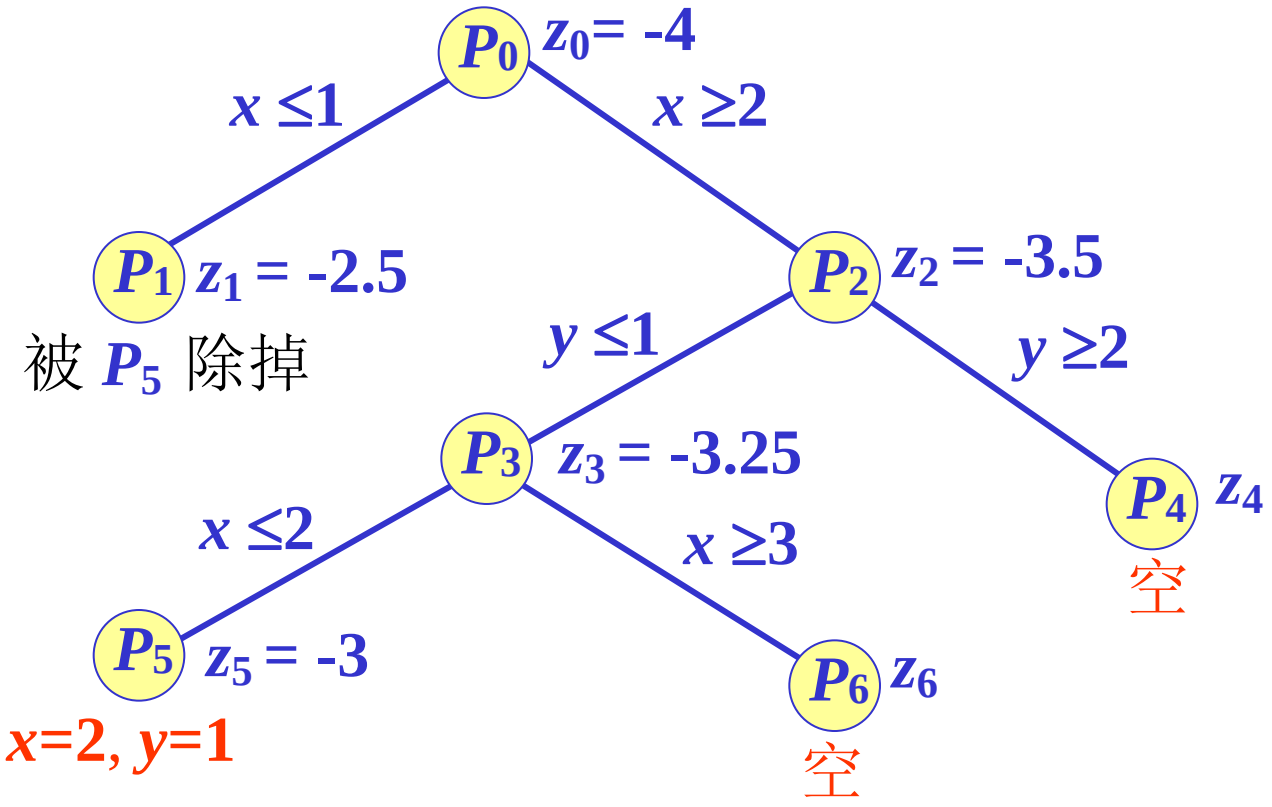


5.3 整数线性规划（续四）



整数线性规划问题的最优解为 $x_{opt} = (2, 1)$。

5.3 整数线性规划（续五）



分支定界法的判定树模型

5.3 整数线性规划（续六）

从上面的分支定界法的描述中可以看出，针对一个具体的组合优化问题，分支定界法取决于它采取的以下三个策略：

- 如何进行分支以使得不需要进一步分支：尽可能快地得到一个不需要再进一步分支的分支。
- 如何得到一个下界：一般是通过求解原问题的某个松弛问题实例 $\mathbf{I}_r$，而得到原问题实例 $\mathbf{I}$ 最优值 $c_{\text{opt}}(\mathbf{I})$ 的一个下界，亦即，去掉原问题实例 $\mathbf{I}$ 中的一些约束条件。
- 在已经生产的所有还没有考虑的子实例中，如何挑选下一个需要考虑的子实例，并对它进行分支和定界。

5.3 一般旅行商

现在考虑**一般旅行商问题**， 它是**欧氏平面上旅行商问题**的一般情形。这里给定一组城市及其城市之间的距离矩阵。目标仍然是求一个相邻城市间距离之和最小的所有城市的一个回路。经过所有城市的一个回路可以用一个循环置换 p_i 表示， $p_i(v_i)$ 表示访问完城市 v_i 紧接着要访问的城市。

问　题： **一般旅行商**

实　例： 一组城市 $C=\{c_0, c_1, \ldots, c_n\}$，

城市之间的距离矩阵 $D_{n\times n}=(d_{ij})$

可行解： n 个城市 C 的一个置换 π。

目　标： 极小化置换的费用，$\sum_{i=1}^{n} d_{i\pi(i)}$

称上述问题为**对称旅行商**问题如果距离矩阵是对称的。

5.3 一般旅行商（续一）

下面我们介绍求解一般旅行商问题的两个分支定界方法。借此说明一个困难问题的子问题是相对容易的。

令 $x_{ij}=1$ 如果边 (i,j) 在回路上，否则 $x_{ij}=1$。旅行商问题的最优解一定满足以下条件：

$$\text{Min } z=\sum_{i,j=1}^{n} d_{ij}x_{ij}$$

$$\sum_{i=1}^{n} x_{ij}=1,\ j=1,2,\ldots,n;$$

$$\sum_{j=1}^{n} x_{ij}=1,\ i=1,2,\ldots,n。$$

5.3 一般旅行商（续二）

上述问题实际上是**指派问题**的一个实例， 它是二部图上的**最大匹配问题**的一种特殊情形。用**匈牙利算法**(1955)即可在多项式时间内求得其最优解。注意，对于原旅行商问题来讲，这个问题的约束是其必要条件而不是充分条件，因为它的解可能包含不止一个圈且仍然保证每一个城市仅在其中的一个圈上。

因此，此问题的一个最优值 z 是原旅行商问题的最优值的一个下界。而且，如果相应的最优解恰是一个回路，那么它也是原旅行商问题的一个可行解。 如果它不是一个回路，那么它一定包含一个长度为 $k < n$ 的子回路：

$$x_{12} = x_{23} = \ldots = x_{k1} = 1。$$

5.3 一般旅行商（续三）

注意，在原旅行商问题的解中，这些变量不可能都为1，因而我们可以在问题的约束中分别添加以下k 约束中一个，将解空间分解为 k 个子集：

$$x_{12} = 0，\ x_{23} = 0，\ \ldots，\ x_{k1} = 0。$$

由此产生 k 个子问题，每一个子问题仍然是一个指派问题。

$$\text{Min } z = \sum_{i,j=1}^{n} d_{ij} x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = 1,\ j = 1, 2, \ldots, n;$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ i = 1, 2, \ldots, n;$$

$$x_{12} = 0.$$

……

$$\text{Min } z = \sum_{i,j=1}^{n} d_{ij} x_{ij}$$

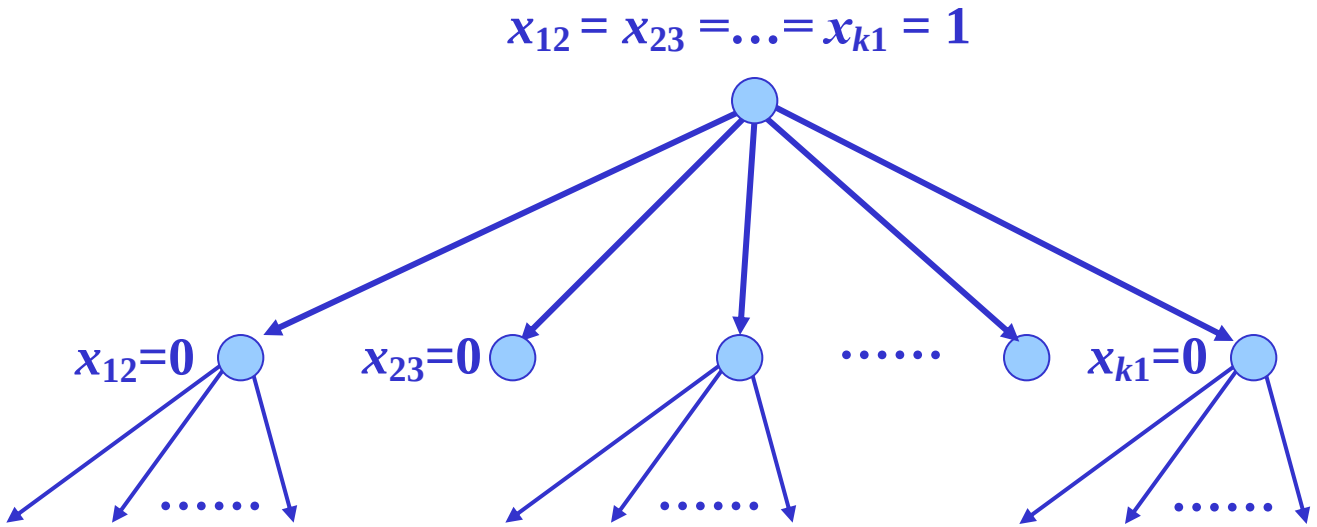
$$\sum_{i=1}^{n} x_{ij} = 1,\ j = 1, 2, \ldots, n;$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ i = 1, 2, \ldots, n;$$

$$x_{k1} = 0.$$

5.3 一般旅行商（续四）

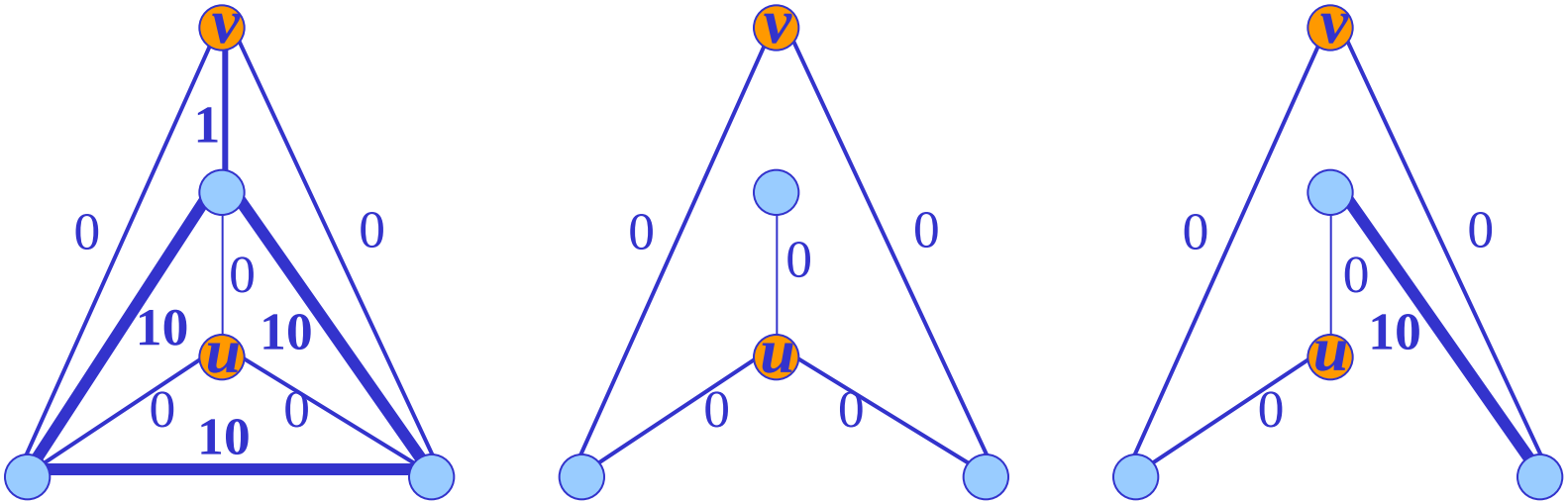
指派问题的解存在一个子回路：



另外一个分支的方法是基于一个简单的事实：最小生成树的最优值是旅行商问题的最优值的一个下界，这是因为将一个回路中的任意一条边去掉都会得到一棵生成树；而且，最小生成树可以在多项式时间内构造出来。

5.3 一般旅行商（续五）

不过，通过引入**1-**树，我们可以得到一个更小的下界。图 ***G*=(*V*, *E*)** 的一棵 **1-树** 是图 ***G*** 的一个子图，它由点集 ***V*\{ *v* }** 的树再加上以 ***v*** 为端点的两条边构成。注意，每一个回路都是一棵 **1-**树，但是反之不成立。而且，最小**1-**树可以在多项式时间内构造出来。因此，最小**1-**树的值可以作为旅行商问题的最优值的一个下界。



5.3 流水线上工件排序

给定一组待加工的工件，每一个工件需要在两台机器上分别完一道加工工序。每一个工件在每台机器上加工需要一定时间，且它在进行第二道加工工序前必须完成第一道加工工序。完工时间定义为所有工件在第二台机器上完工时间的总和。 流水线上工件排序问题就是确定待加工的工件在每台机器上的加工顺序，使得完工时间最小。这是一个 **NP-难**问题。

问　题.　两台机器的流水线上工件排序

实　例：一组工件 $J = \{J_1, \ldots, J_n\}$，每一个工件有两道工序需要在机器 M_1 和 M_2 完成，需要加工的时间为 t_{ij}。

可行解：工件及工序在机器上的加工顺序。

目　标：最小化完工时间。

5.3 流水线上工件排序（续一）

下表给出了一个具有三个待加工工件的实例：

加工时间 t_{ij}	机器 M_1	机器 M_2
工件 J_1	2	3
工件 J_2	3	1
工件 J_3	2	3

注意，要求解上述问题，我们需要确定两个置换，一个是 n 工件的顺序，另一个是工件的工序的置换。不过，下面的引理告诉我们，可以把精力仅放在可以决定整个排序的一个置换上。

定理 1 两台机器的流水线上工件排序问题存在一个最优解，使得工件在两台机器上的加工顺序是完全一样的。

5.3 流水线上工件排序（续二）

所有六个可能的排序方案。惟一的**最优解**的完工时间为18。

$J_1J_2J_3$	M_1	1	1	2	2	2	3	3			
F=19	M_2			1			2		3	3	3
$J_1J_3J_2$	M_1	1	1	3	3	2	2	2			
F=18	M_2			1		3	3	3	2		
$J_2J_1J_3$	M_1	2	2	2	1	1	3	3			
F=20	M_2				2		1		3	3	3
$J_2J_3J_1$	M_1	2	2	2	3	3	1	1			
F=21	M_2				2		3	3	3	1	
$J_3J_1J_2$	M_1	3	3	1	1	2	2	2			
F=19	M_2			3	3	1			2		
$J_2J_3J_1$	M_1	3	3	2	2	2	1	1			
F=20	M_2			3	3	3	2		1		

5.3 流水线上工件排序（续三）

由前述引理可知，求解流水线上工件排序问题的关键是找到 n 个工件恰当的置换。一个自然的分支方法是选择要排序的第一个工件在第一层，第二个工件在第二层，等等。下一步就是要构造一个下界函数。

一个非常有效的方法如下：假设在集合 $J' \subset J$ 中的工件已经排好了顺序，其中 $|J'| = r$。设排序方法 A 将第 k 个工件安排在 a_k 的位置，$k = 1, 2, \ldots, n$。则此排序方法的费用是

$$F = \sum_{j \in J'} F_{2j} + \sum_{j \notin J'} F_{2j}$$

5.3 流水线上工件排序（续四）

注意，如果每一个工件都可以在第一台机器M_1上一加工完，就开始在第二台机器 M_2 上加工， 那么上式的第二个求和项就变成如下的形式：

$$F' = \sum_{k=r+1}^{n} \left(F_{1a_k} + (n-k+1)\, t_{1a_k} + t_{2a_k} \right)$$

若这不可能，如图例中的排序方案 $J_2J_3J_1$，则 F' 只可能增加，所以我们就得到第二个求和项的一个如下的下界

$$\sum_{j \notin J'} F_{2j} \geq F'$$

5.3 流水线上工件排序（续五）

类似地，如果在机器 M_2 的每一个工件可以立刻加工只要排在它前面的工件加工完，那么上述等式的第二个求和项就变成如下形式

$$F'' = \sum_{k=r+1}^{n} \left(\max\{F_{2a_r}, F_{2a_k}\} + (n-k+1)\, t_{2a_k} \right)$$

同样，若这不可能，如图例中给出的排序$J_1J_2J_3$，则可得到第二个求和项的另外一个下界

$$\sum_{j \notin J'} F_{2j} \geq F''$$

因此，我们可以得到排序的下述下界：

$$F \geq \sum_{j \in J} F_{2j} + \max\{F', F''\}$$

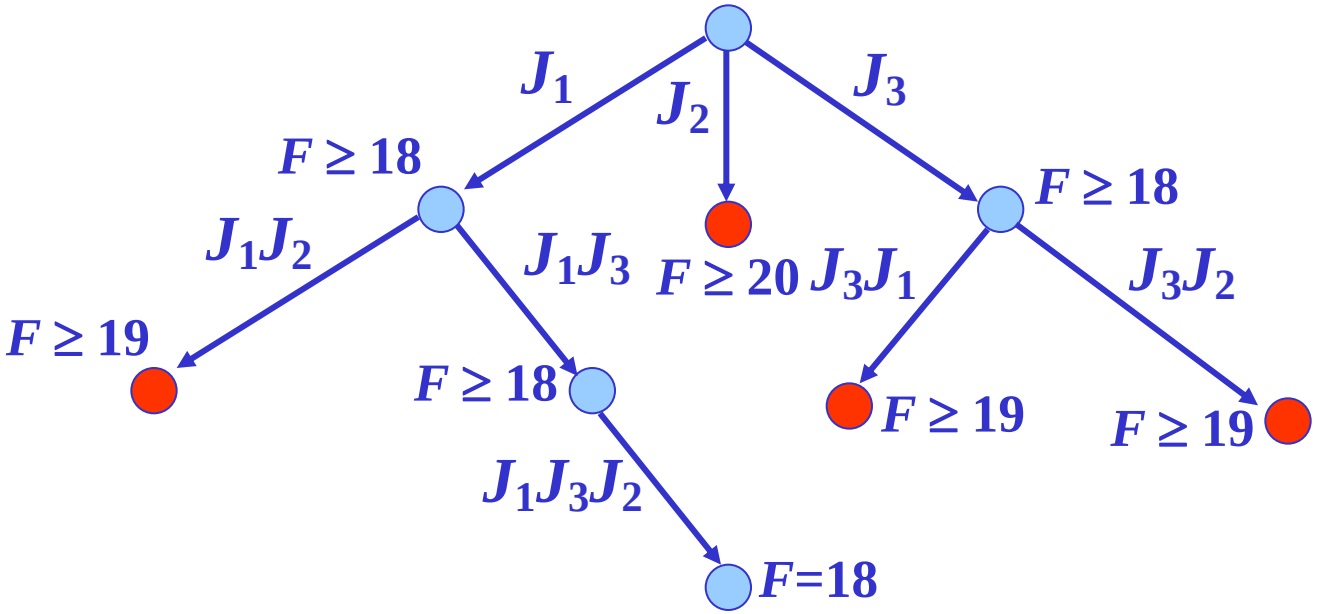
5.3 流水线上工件排序（续六）

注意，所得下界依赖于J'中剩下的工件是如何排序的。然而，这种依赖性可以通过如下方法消除：选取 a_k ($k = r, \ldots, n$) 使得加工时间为 t_{1a_k} 的工件是依照递增的顺序排的 (这样 F' 就可极小化了)，选取 a_k ($k = r, \ldots, n$) 使得加工时间为 t_{2a_k} 的工件是依照递增的顺序排的 (这样 F'' 就可极小化了)。将所得的相应值分别记为 $\overline{F'}$ 和 $\overline{F''}$。因而我们可以得到一个很容易计算的下界

$$F \geq \sum_{j \in J} F_{2j} + \max\{ \overline{F'}, \overline{F''} \}$$

5.3 流水线上工件排序（续七）

下面的图给出了前面所给实例的一个完整的搜索树。对具有最小下界的节点首先进行分支，如果两个节点的相应最小值是一样的，那么从左往右搜索。 一旦得到最优解 $J_1 J_3 J_2$，它就会除掉所有其它的节点。



5.3 总结

分支定界法与**分而治之法**和**动态规划法**在如下两个方面有本质差别：

- 分支定界法在求解给定的求最小值问题 P 时，它将其分解为若干个小的子问题 P_i，每一个子问题的最优值的下界可以很容易求得。它不试图直接求出子问题的最优解，因为这十分困难。
- 尽管 分支定界法的思想是以某种聪明的方式枚举和考虑所有的可行解， 但是在最坏情形下，它并不比简单枚举和考虑所有的可行解更好。因此，即使对一个分支定界算法进行理论分析不是不可能的，也是十分困难的。