Last Section

机器负荷分配问题

例2 机器负荷分配问题

设机器在高负荷下生产的产量函数为$S_1=8u_1$,年折损率为$a=0.7$;

在低负荷下生产的产量函数为$S_2=5u_2$,年折损率为$b=0.9$。

开始生产时完好机器的数量$x_1=1000$台。按题意要安排好五年的生产计划，使产品的总产量最高。

动态规划应用举例

- 资源分配问题
- 生产与存储问题
- 复合系统工作可靠性问题
- 排序问题
- 设备更新问题

$$\begin{cases} \max[\, g_1(x_1)+g_2(x_2)+.....+g_n(x_n)] \\ x_1+x_2+.....+x_n=a \\ x_i \geq 0 \qquad i=1,2.....,n \end{cases}$$

$$\max P(z_1,z_2,.....,z_n)=\prod_{i=1}^{N} p_i(z_i) \qquad R=\left\{ z \mid \sum_{i=1}^{N} c_i z_i \leq c, \sum_{i=1}^{N} w_i z_i \leq w, z_i \geq 0 \right\}$$

可基于动态规划思想求解的问题与算法

- 计算二项式系数

$$C(n,k) = C(n-1,k-1) + C(n-1,k)$$

- 最长公共子序列

$L[i, j] = 0$ 若i=0或j=0

$=L[i\text{-}1, j\text{-}1] + 1$ 若i>0, j>0 和$a_i = b_j$

$=\max\{ L[i, j\text{-}1], L[i\text{-}1, j]\}$ 若i>0, j>0 和$a_i \neq b_j$

- 矩阵链相乘

$$C[i, j] = \min_{i < k \le j} \{C[i, k-1] + C[k, j] + r_i r_k r_{j+1}\}$$

可基于动态规划思想求解的问题与算法

- **所有点对的最短路径问题**
  - 做n次迭代，使在第k次迭代后，$D_k[i,j]$含有从顶点i到顶点j，且不经过编号大于k的任何顶点的最短路径的长度。

$$d^{k}_{i,j}=l[i,j] \qquad \text{若} k=0$$
$$=\min\{d^{k-1}_{i,j}, d^{k-1}_{i,k}+d^{k-1}_{k,j}\} \qquad \text{若} 1\leqslant k\leqslant n$$

- **0/1 背包问题**
  - 设$V[i,j]$ 表示从前i项$\{u_1, u_2, \ldots, u_i\}$中取出来的装入体积为$j$的背包的物品的最大价值

$$V[i,j]=0 \qquad \text{若} i=0\text{或}j=0$$
$$=V[i-1,j] \qquad \text{若} j<s_i$$
$$=\max\{V[i-1,j], V[i-1,j-s_i]+v_i\} \qquad \text{若} j\geq s_i$$

可基于动态规划思想求解的问题与算法

- **计算有向图的传递闭包**

  Warshall 算法：

  $r_{ij}^{(k)}$置为1意味着存在一条从第i个顶点到第j个顶点的路径，路径中每一个中间顶点的编号都不大于k

  $r_{ij}^{(k)} = r_{ij}^{(k-1)}$ ***or*** $r_{ik}^{(k-1)}$ ***and*** $r_{kj}^{(k-1)}$

- **最优二叉查找树：**

  C[i,j]是在这棵树中成功查找的最小的平均查找次数

  考虑从键$a_i,\ldots,a_j$ (sorted)中选择一个根a_k的所有可能的方法

$$C[i,j] = \min_{i \le k \le j}\{C[i,k-1] + C[k+1,j]\} + \sum_{s=i}^{j} p_s$$

回溯

- 回溯法的主要思想
  - 有组织的穷尽搜索

- 3 着色问题
- 8皇后问题
- 哈密尔顿回路

- 一般回溯算法

分支定界

(Branch and Bound)

一般着色问题

- 3（N）着色问题：

  给定图 G，是否可以用 3（N）种颜色，对其进行合法的着色？

- 一般着色问题：

  给定图 G，对其进行合法着色，最少需要几种颜色？

  e.g.

整数规划

线性规划的数学模型：

目标函数

$$\text{Max(Min)}\ z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

满足约束条件：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots\ldots + a_{1n}x_n \le (=,\ge) b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots\ldots + a_{2n}x_n \le (=,\ge) b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots\ldots + a_{mn}x_n \le (=,\ge) b_m \\ x_1, x_2, \ldots\ldots x_n \ge 0 \end{cases}$$

整数规划

- 例：某厂拟用集装箱托运甲乙两种货物，每箱的体积、重量、可获得利润以及托运所受限制如表：

货物	体积　（每箱）	重量　（每箱）	利润　（每箱）
甲 已	9 7	7 20	40 90
托运限制	56	70	

问两种货物各托运多少箱，可使得利润为最大？

整数规划

- 求解

$$\text{Max } z = 40x_1 + 90x_2 \qquad (1)$$

$$\begin{cases} 9x_1 + 7x_2 \le 56 & (2) \\ 7x_1 + 20x_2 \le 70 & (3) \\ x_1, x_2 \ge 0 & (4) \\ x_1, x_2 \in \text{integer} & (5) \end{cases}$$

整数规划

- 求解

$$\text{Max } z = 40x_1 + 90x_2 \qquad (1)$$

$$\begin{cases} 9x_1 + 7x_2 \le 56 & (2) \\ 7x_1 + 20x_2 \le 70 & (3) \\ x_1, x_2 \ge 0 & (4) \\ x_1, x_2 \in \text{integer} & (5) \end{cases}$$

- 解：先不考虑条件（5），即解相应的线性规划（1）-（4），得最优解

$x_1 = 4.809$，　　$x_2 = 1.817$，　$\max z = 355.890$

整数规划

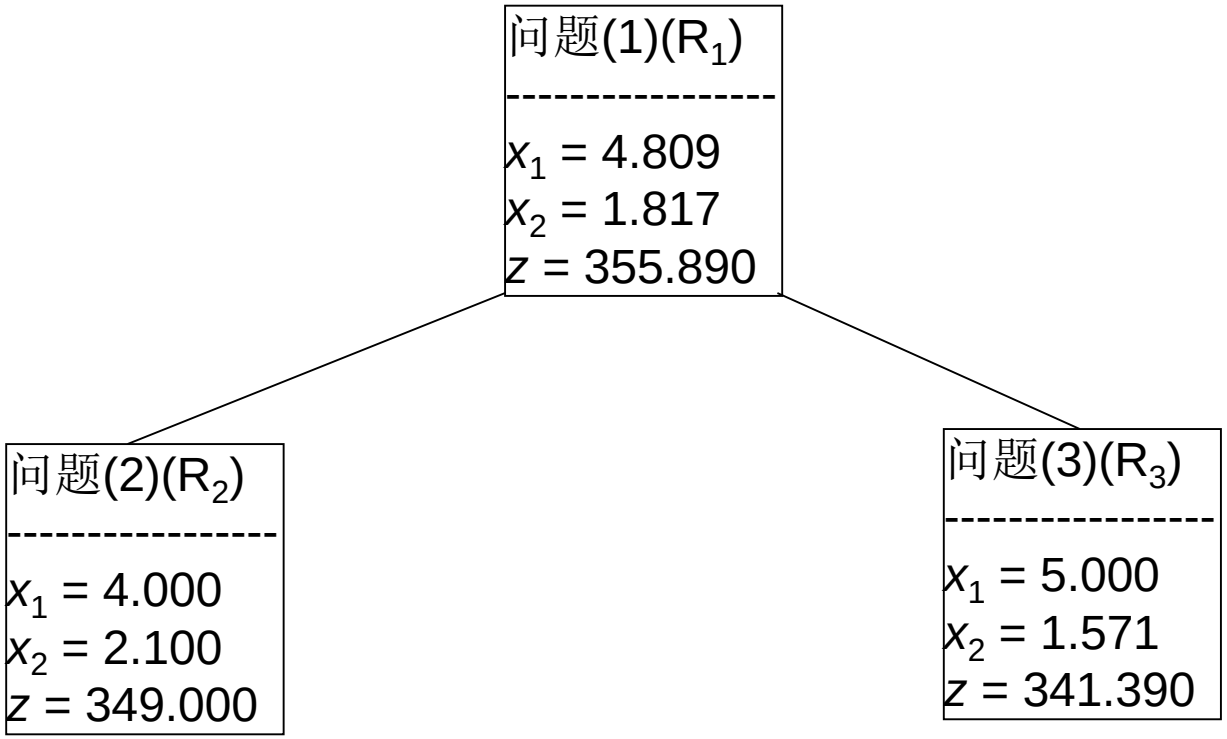
- 分支定界法首先关注其中一个非整数的变量（可以任选），例如选$x_1 = 4.809$，我们可认为最优整数解x_1是$x_1 \leq 4$或$x_1 \geq 5$，而在4与5之间是不合整数条件的，于是把原问题分解成两枝，各枝都增加了约束条件。
- 称原问题为问题（1），它的可行域为R_1；称分解出来的两枝为问题（2）和问题（3），它们的可行域分别为R_2和R_3，依次类推。
- 不考虑整数条件，解问题（2）和问题（3）。

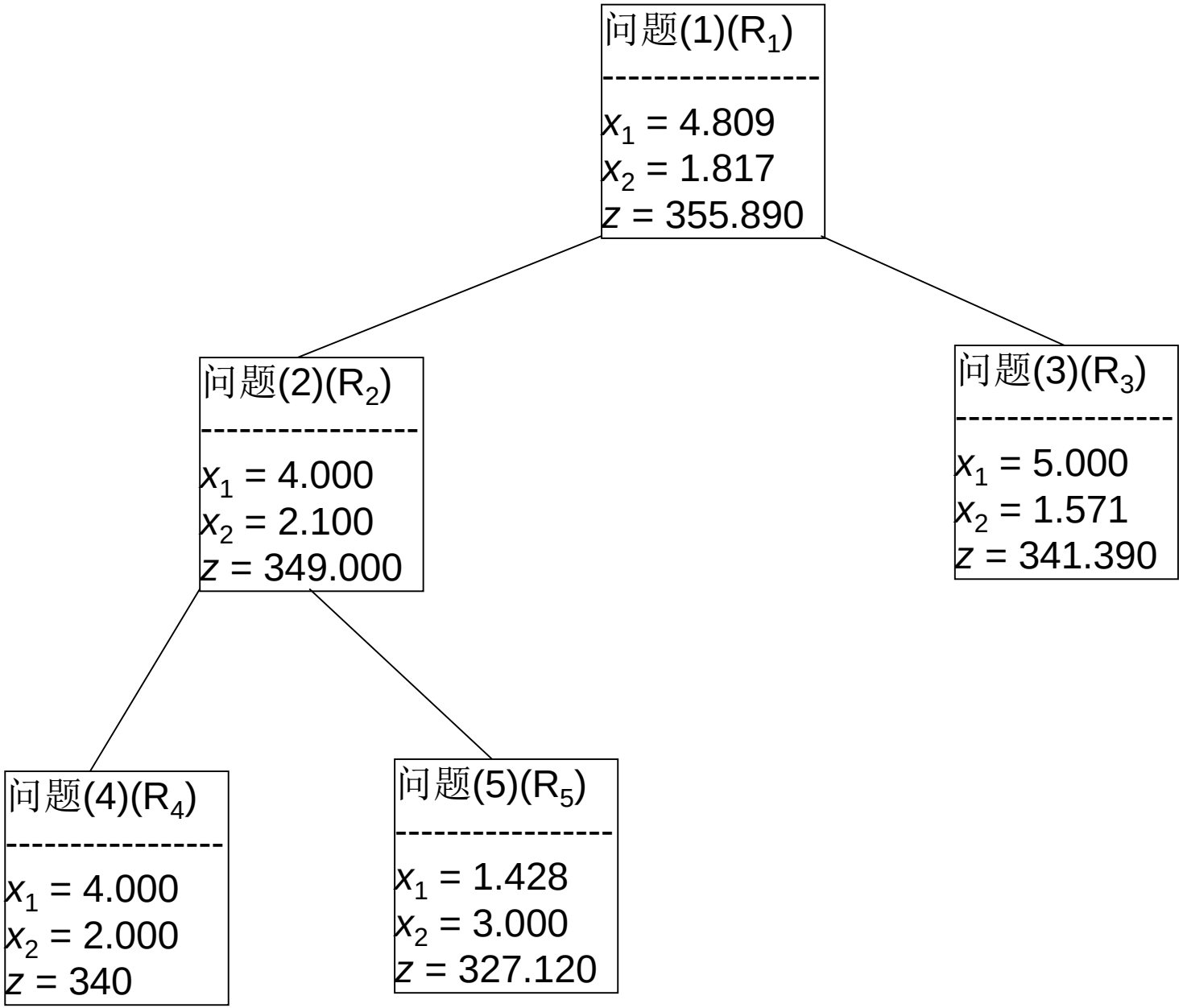
问题(1)(R_1)

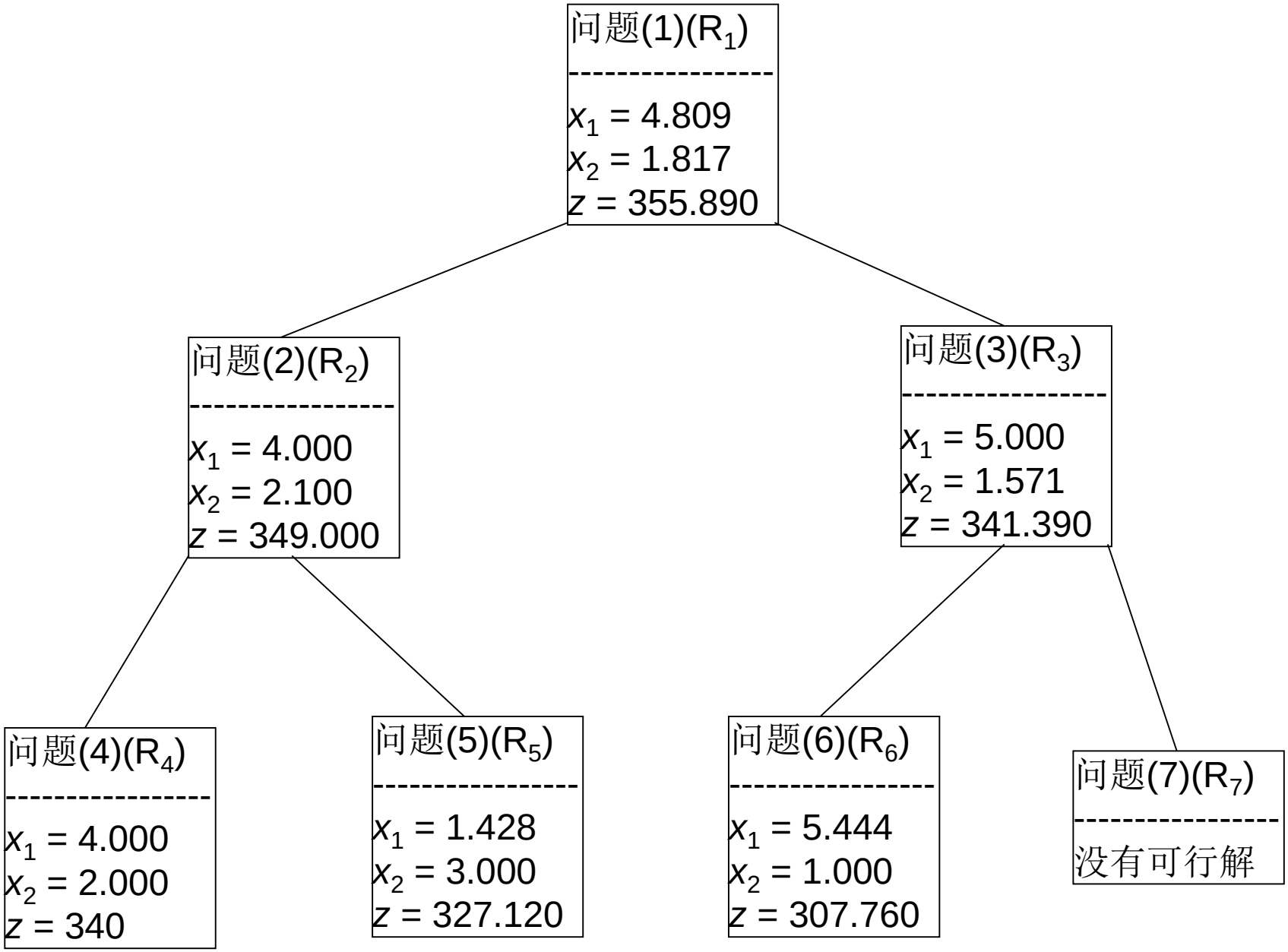
$x_1 = 4.809$

$x_2 = 1.817$

$z = 355.890$


问题(1)(R_1)
x_1 = 4.809
x_2 = 1.817
z = 355.890
问题(2)(R_2)
x_1 = 4.000
x_2 = 2.100
z = 349.000
问题(3)(R_3)
x_1 = 5.000
x_2 = 1.571
z = 341.390


问题(1)(R_1)
x_1 = 4.809
x_2 = 1.817
z = 355.890
问题(2)(R_2)
x_1 = 4.000
x_2 = 2.100
z = 349.000
问题(3)(R_3)
x_1 = 5.000
x_2 = 1.571
z = 341.390
问题(4)(R_4)
x_1 = 4.000
x_2 = 2.000
z = 340
问题(5)(R_5)
x_1 = 1.428
x_2 = 3.000
z = 327.120


问题(1)(R_1)
x_1 = 4.809
x_2 = 1.817
z = 355.890
问题(2)(R_2)
x_1 = 4.000
x_2 = 2.100
z = 349.000
问题(3)(R_3)
x_1 = 5.000
x_2 = 1.571
z = 341.390
问题(4)(R_4)
x_1 = 4.000
x_2 = 2.000
z = 340
问题(5)(R_5)
x_1 = 1.428
x_2 = 3.000
z = 327.120
问题(6)(R_6)
x_1 = 5.444
x_2 = 1.000
z = 307.760
问题(7)(R_7)
没有可行解

分支定界法解整数规划问题的一般步骤

1. 称原整数规划问题为A，称相应的线性规划问题（即不考虑整数条件）为问题B，解问题B。
2. 如问题B没有可行解，即停止。这时问题A也没有可行解。
3. 如求得问题B的最优解，检查它是否合于整数条件：如合于整数条件，它就是问题A的最优解；如不符合整数条件，即转入下步。
4. 在问题B的解中，任选一个不符合整数条件的变量x_j，如x_j的值是b_j，作两个后继问题，它们是对问题B分别增加一个约束条件：

   a) $x_j \leqslant$（小于b_j的最大整数）

   b) $x_j \geqslant$（大于b_j的最小整数）

   不考虑整数条件，解这两个后继问题。
5. 在现有的，且还没分解出后继问题的各可行问题中，选目标函数值为最优的问题。重新称这问题为问题B，回到（3），重复进行。

主要的分支定界法相关概念

- Feasible （solution）
- Infeasible （solution）
- Partial Solution
- Optimal Solution
- Branch
- Bound (tight)
- Traverse
- Backtracking
- Prune
- DFS, BFS, Frontier Search
- Initial solution
- Relax

分支定界法的主要思想

- 最优化问题是根据某些约束寻求目标函数的最大或最小值。
- 可以利用回溯的思想。
- 且回溯的思想得到进一步的强化。
- 和回溯法相比，分支定界法需要两个额外的条件：

1. 对于一棵状态空间树的每一个节点所代表的部分解，我们要提供一种方法，计算出通过这个部分解繁衍出的任何解在目标函数上的最佳值边界。
2. 目前求得的最佳解的值。

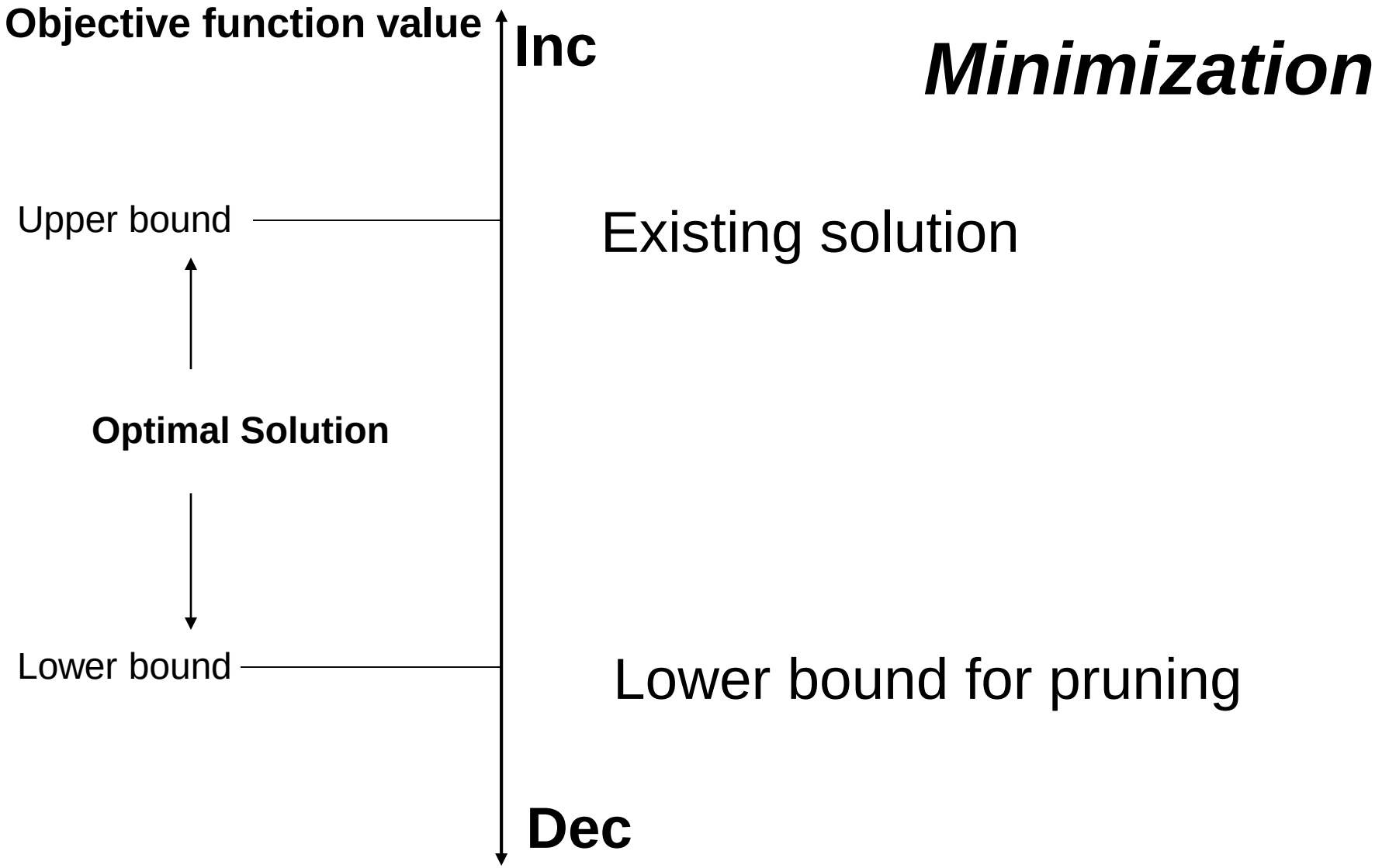
分支定界法的主要思想

- 若能得到1，2两信息，即可拿某个阶段的边界值和目前求得的最佳解值比较：

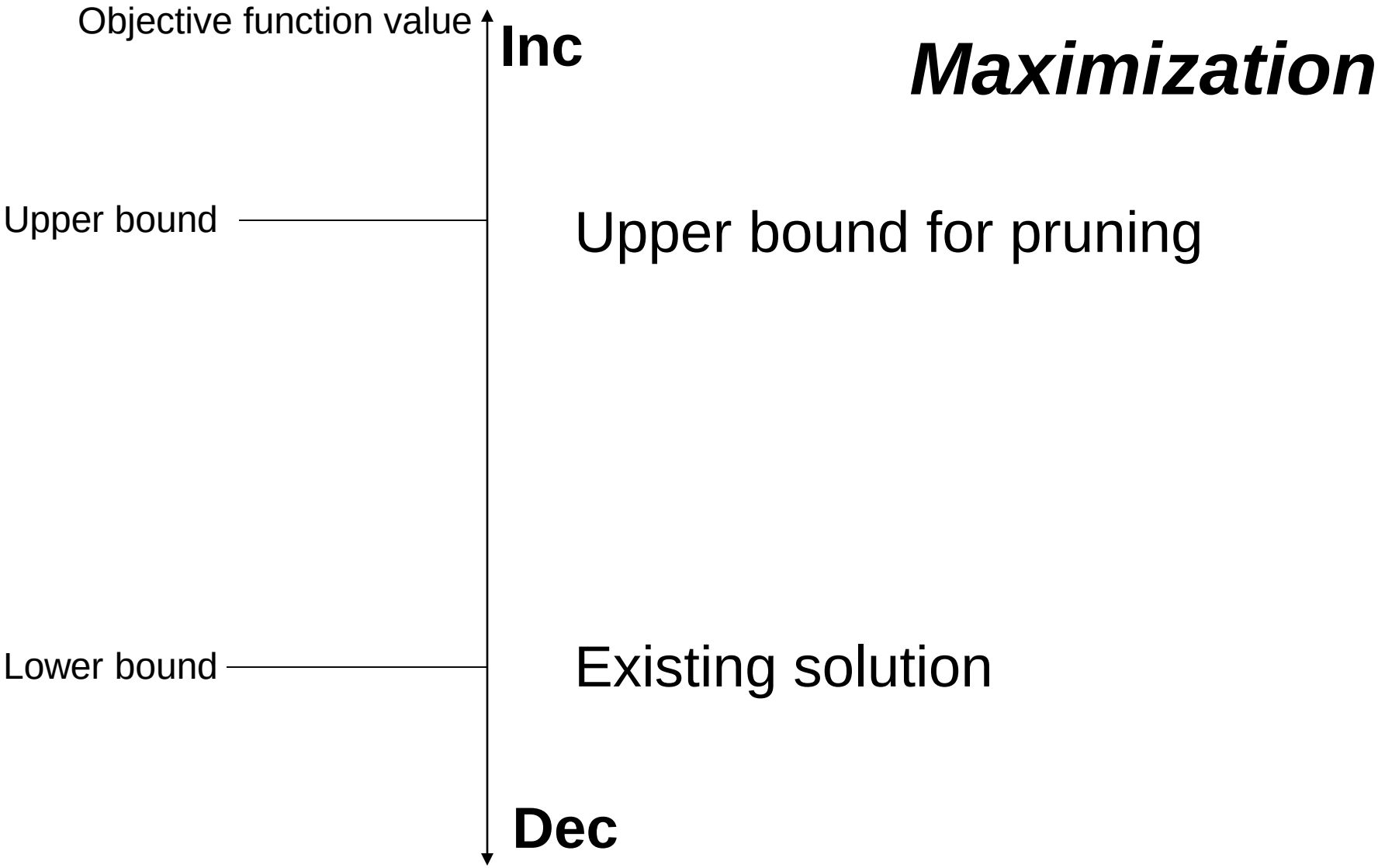
  如果边界值不能超越（也就是说，在最小化问题中不小于，在最大化问题中不大于）目前的最佳解，这个节点就是一个没有希望的节点，需要立即关闭（剪枝），因为从这个节点生成的解，没有一个能比目前已经得到的解更好。

  这就是分支界限技术的主要思想。

Upper Bound and Lower Bound



Upper Bound and Lower Bound



分支定界法的主要思想

一般来说，对于一个分支定界算法的状态空间树来说，只要符合下面任一条件，我们就会中止它的在当前节点上的查找路径：

- 该节点的边界值不优于目前最佳解的值。
- 该节点无法代表任何可行解， 因为它已经违反了问题的约束。
- 该节点代表的可行解的子集只包含一个单独的点（因此无法给出更多的选择）。在这种情况下，我们拿这个可行解在目标函数上的值和目前求得的最佳解进行比较，如果新的解更好一些的话，就用前者替换后者。

背包问题

物品	重量	价值
1	3	12
2	4	40
3	5	25
4	7	42

- 背包的承重量*W* 等于10

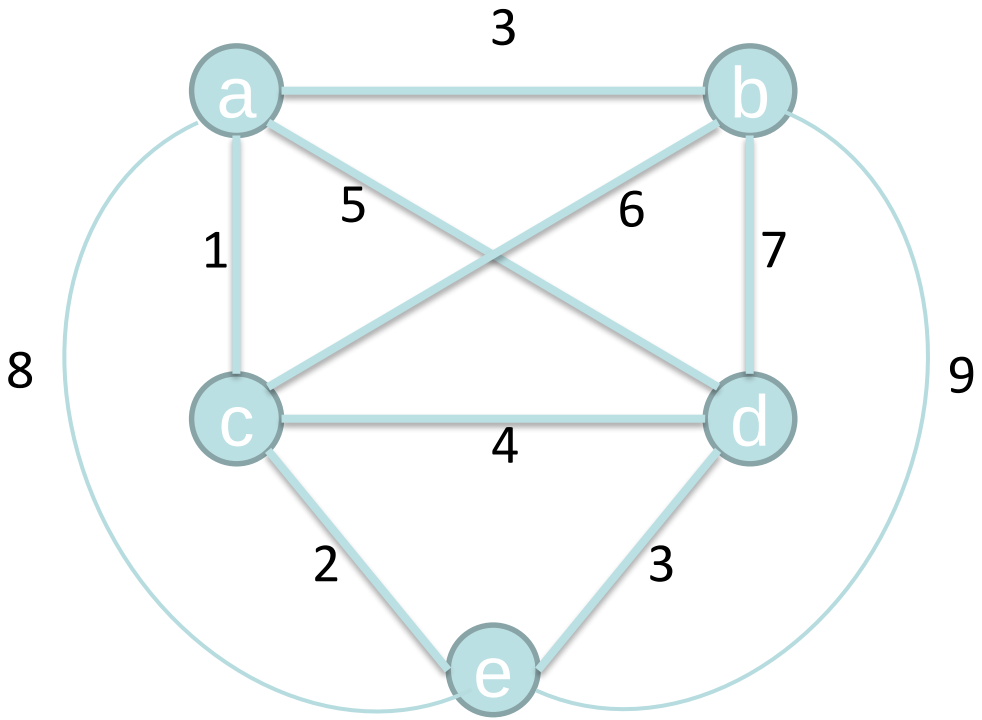
背包问题

物品	重量	价值	价值/重量
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

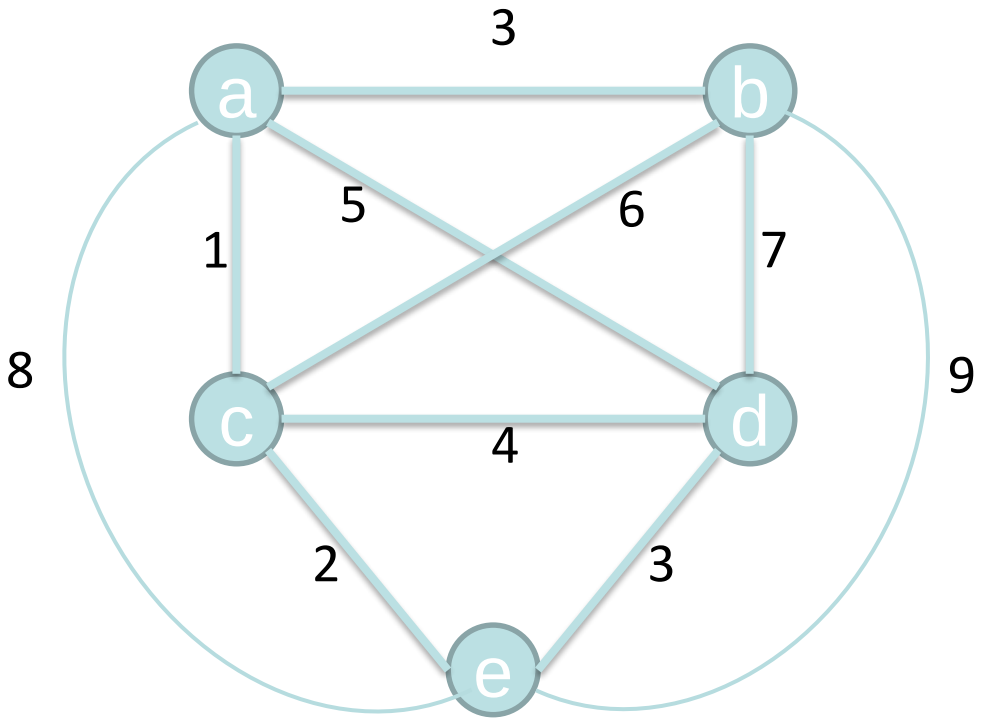
- 背包的承重量W 等于10
- 一种计算上界的方法是把已选物品的总价值v，加上背包剩余重量与剩下物品的最佳单位回报的积：

$$ub = v + (W - w) \times (v_{i+1} / w_{i+1})$$

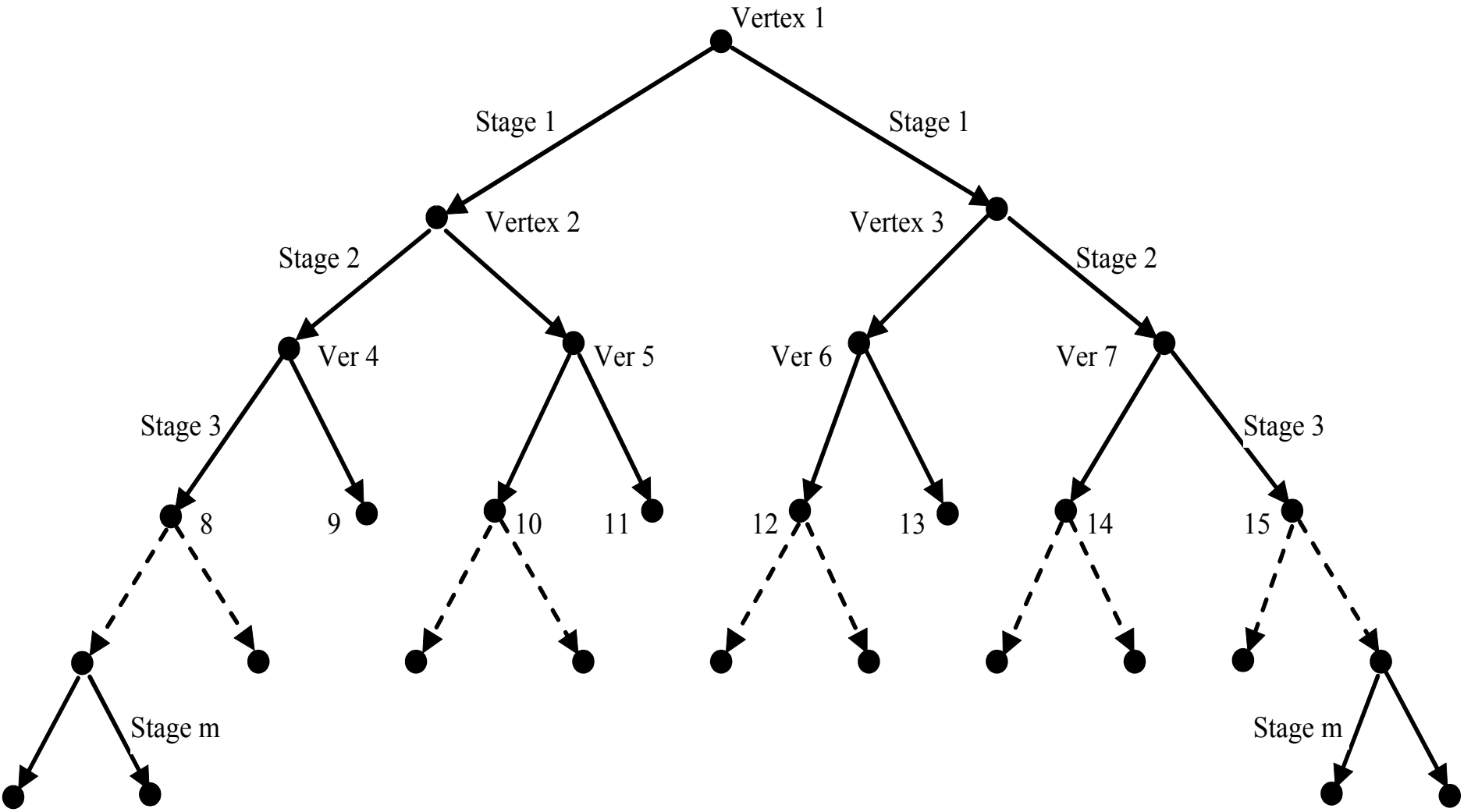
TSP



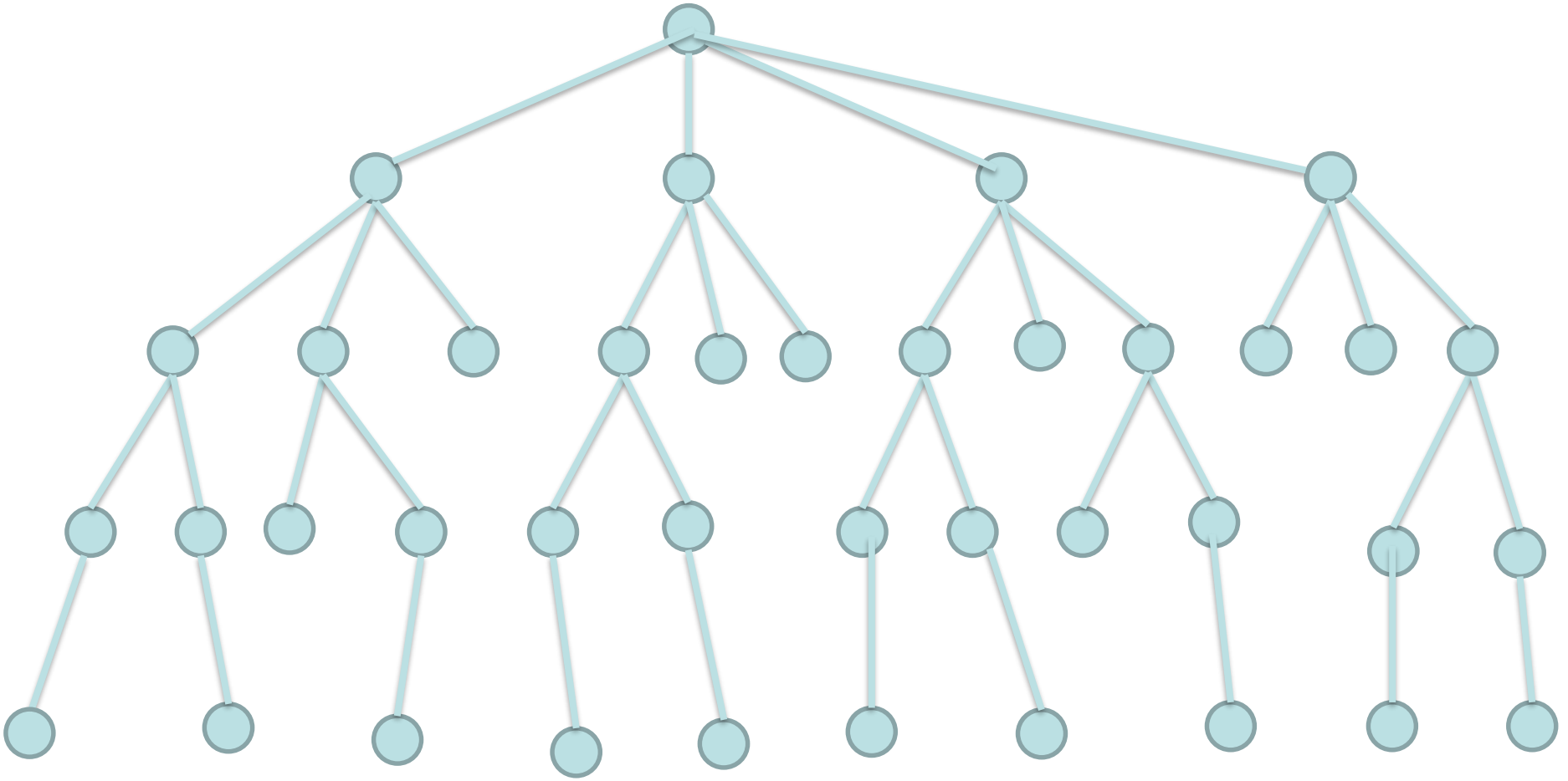
TSP _ 搜索树？



The search tree



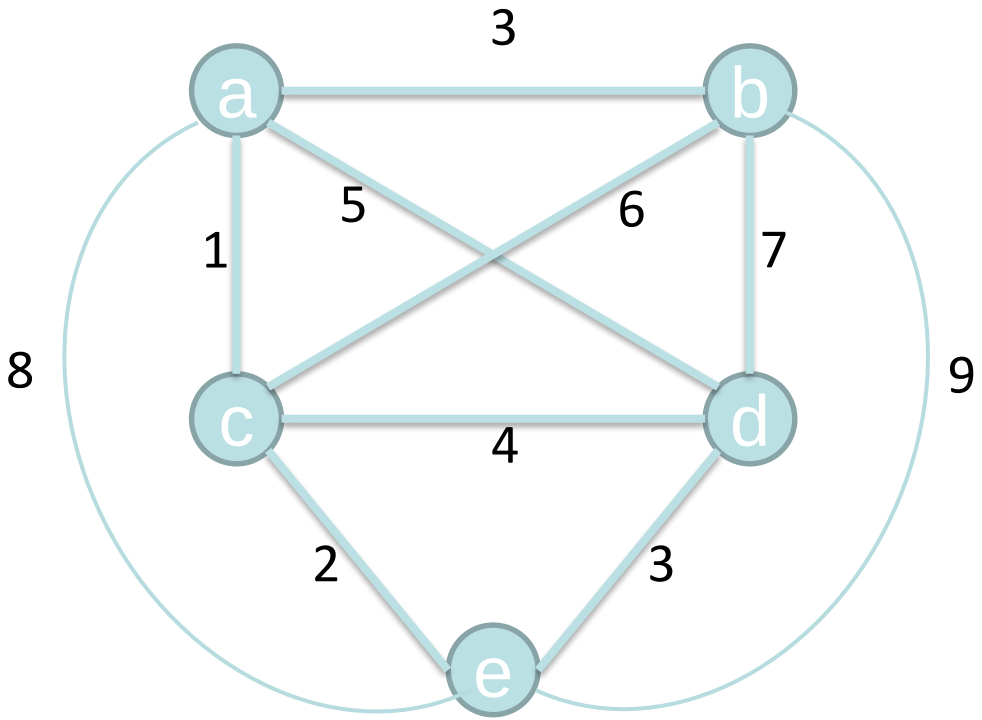
The search tree



TSP _ 下界？

如果能定义下界，即可使用分支定界算法。

TSP

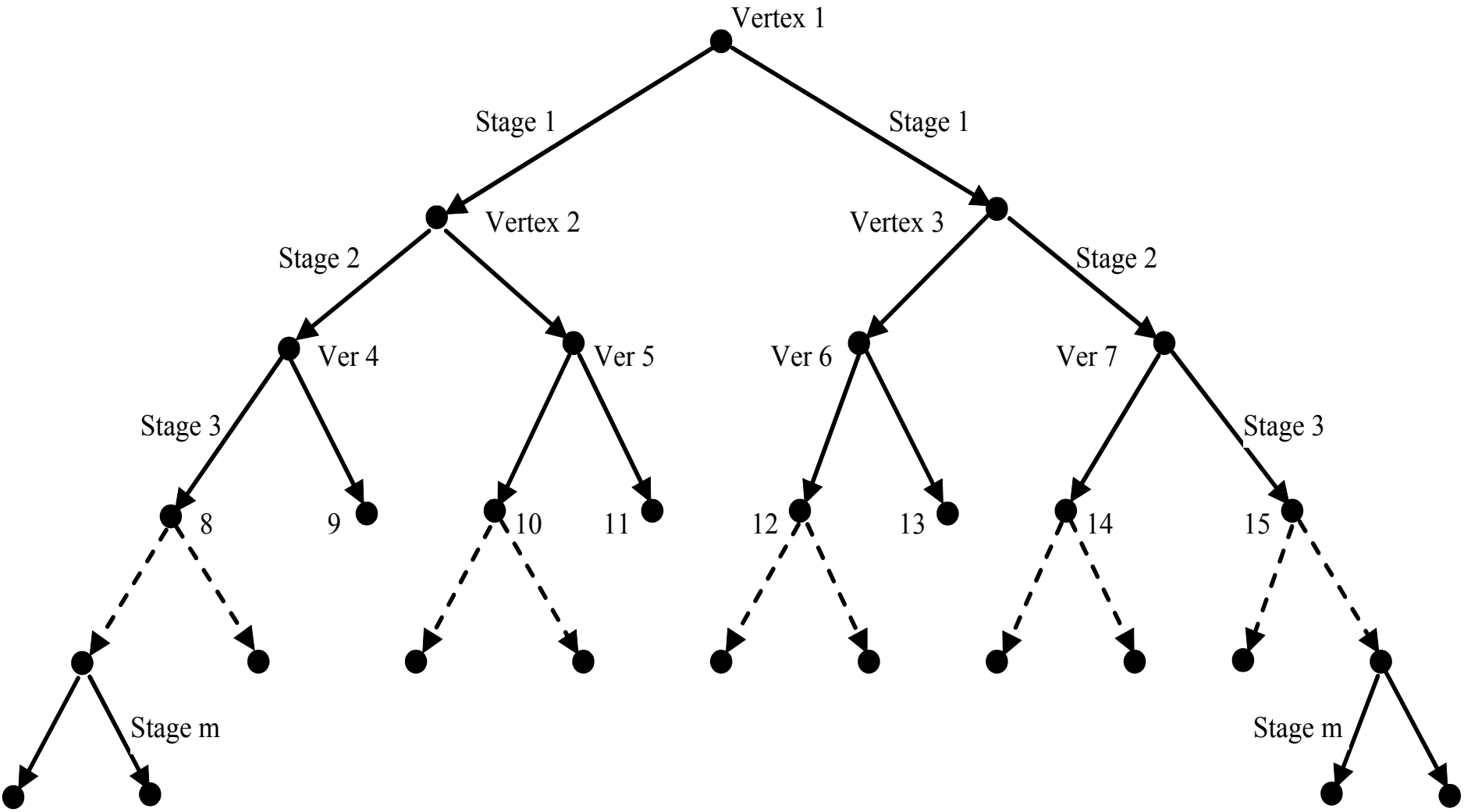


TSP _ 下界？

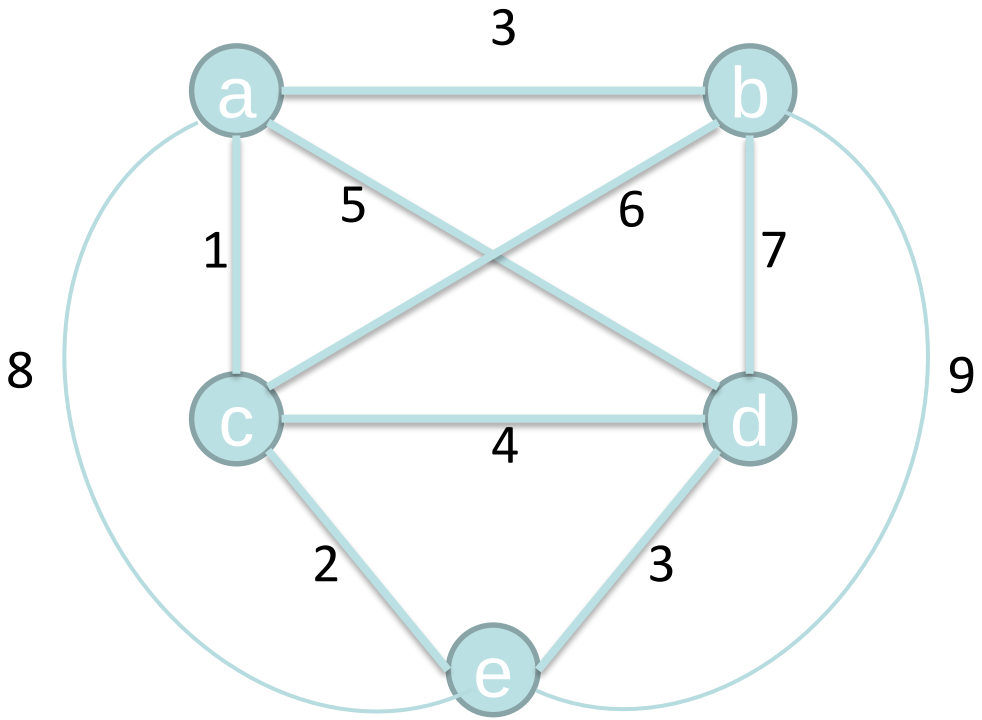
如果能定义下界，即可使用分支定界算法。

- 通过把城市距离矩阵D 的最小元素乘以城市数量n， 我们可以得到一个非常简单的下届。

Does a Lower Bound Work ?



TSP



TSP _ 下界？

如果能定义下界，即可使用分支定界算法。

- 通过把城市距离矩阵D 的最小元素乘以城市数量n， 我们可以得到一个非常简单的下届。

- 稍紧一些的下届：最小的n个元素的和

好的下界

- 发现一个好的边界函数常常不是一件简单的任务。一方面，我们希望这个函数容易计算。但另一方面又不能过于简单(可能过松)。
- 任何旅行长度 l 的下届：对于每一个城市i， $1 \le i \le n$,求出从城市i到最近的两个城市的距离之和s_i；计算出这n个数字的和s，并把结果除以2；而且，如果所有的距离都是整数，还可将这个结果向上取整（更紧）

$$lb = \lceil s/2 \rceil$$

$$lb = \lceil [(1+3)+(3+6)+(1+2)+(3+4)+(2+3)]/2 \rceil = 14$$

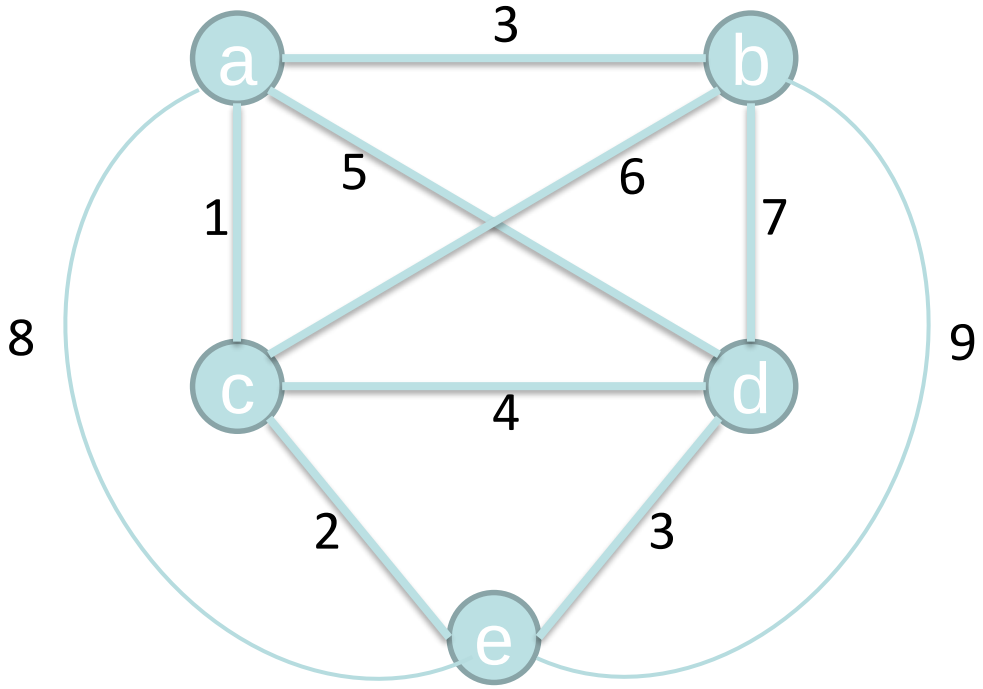
更紧的下界

- 此外，如果一个给定图的任何旅程子集都必须包含某些特定的边，例如，如果在图中，要求图的所有哈密顿回路都必须包含边（*a*,*d*），我们可以分别把*a*,*d* 两个顶点附带的两条最短边，和要求附带的边（*a*,*d*）和（*d*,a）相加，以得到下面这个下界：

$$\left\lceil [(1+5)+(3+6)+(1+2)+(3+5)+(2+3)]/2 \right\rceil = 16$$

- 求解例题

TSP



$$lb = \lceil [(1+3)+(3+6)+(1+2)+(3+4)+(2+3)]/2 \rceil = 14$$

$$\lceil [(1+5)+(3+6)+(1+2)+(3+5)+(2+3)]/2 \rceil = 16$$

有向图的TSP

- Matrix reduction
  - 每一个完整的tour,包含且仅包含每一行上的一个值和每一列上的一个值
  - 若某行（列）减去一个值w,则总的解值就减少w
  - Matrix reduction：每行、列减去其最小值，使每行、列都至少有一个0，被减值的和就是一个界
- 图例

分配（指派）问题

- 有n项任务要完成，恰好有n个人可以分别去完成其中每一项，但由于任务性质和个人专长不同，因此每个人去完成不同的任务的效率（或所费时间）就有差别。

- 由此提出下述问题：应当指派哪个人去完成哪项任务，使总的效率为最高（或花费的总时间为最小）？

分配（指派）问题

- 一个指派问题给出系数矩阵，或称效率矩阵，矩阵的元素c_{ij}(>0) (i, j=1, 2,…, n)表示指派第i人去完成第j向任务的效率（或时间，成本等）。

- 解题时，引入0-1变量 x_{ij}：

  令 x_{ij}=1，当指派第i人去完成第j项任务时

  =0，当不指派第i人去完成第j项任务时

分配（指派）问题

- 极小化问题的数学模型是：

$$\min z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \qquad ①$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1,2,...,n \qquad ②$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1,2,...,n \qquad ③$$

$$x_{ij} = 0 \text{ 或} 1 \qquad ④$$

- 约束条件②说明第j项任务只能由1个人来完成；约束条件③说明第i人只能完成1项任务。合于约束条件②—④的可行解也可写成表格或矩阵形式。

分配（指派）问题_下界

- 任何解都不会小于每行中最小元素的和。
- 适用于部分解

$$C=\begin{bmatrix}12 & 3 & 9 & 7\\ 5 & 4 & 6 & 8\\ 4 & 10 & 2 & 6\\ 7 & 5 & 8 & 3\end{bmatrix}\begin{matrix}a\\ b\\ c\\ d\end{matrix}$$

匈牙利法

- 匈牙利法的根据是

  指派问题最优解的性质：如果从系数矩阵（c_{ij}）的一行（列）各元素分别减去该行（列）的最小元素，得到新矩阵（b_{ij}） 。那么，以（b_{ij}）为系数矩阵的指派问题的最优解（x_{ij}）和原问题的最优解相同。

- 这个性质成立的理由是，解（x_{ij}）的一行（列）元素中只有一个是1，如果从（c_{ij}）的第i行(列)元素各减一常数k，那么使目标函数z也减去k，而z和z-k的最优解（x_{ij}）当然是相同的。

匈牙利法

- **第一步：使系数矩阵出现0元素；**

- 从系数矩阵的每行元素减去各该行的最小元素；
- 再从所得矩阵的各列减去各该列的最小元素。
- 如果某行某列已有0元素，则不处理。

匈牙利法

- **第二步：试求最优解**。

  经过第一步变换后，矩阵的每行每列都有了0元素，但我们希望能找出n个位于不同行不同列的0元素。如果能找到，我们就以这样的元素对应x_{ij}=1，令其余的x_{ij}=0，这样的解代入目标函数z_b=0，它一定是极小，这就得到了（b_{ij}）问题的最优解（x_{ij}），根据上述性质，它也是原问题的最优解。

  由有0元素最少的行开始，圈出一个0元素，用◎表示，然后划去同行同列的其它0元素，用Ø表示。这样依次做完各行，已划去的不能再圈。如果这样能得到n个◎， 就完成了求解最优的过程。

  否则：

匈牙利法

- **第三步：作能覆盖所有0元素的最少数目的直线集合：**

1. 对没有◎的行打∨号；
2. 对打∨行上的所有0元素的列打∨号；
3. 再对打∨号的列上有◎的行打∨号；
4. 重复（2）（3）直到得不出新的打∨号的行列为止；
5. 对没有打∨号的行画横线，所有打∨号的列画纵线，这就是能覆盖所有0元素的最少数直线集合。

匈牙利法

- **第四步：变换矩阵使0元素移动，以达到每行都有◎元素的目的。**
- 为此，在没被直线复盖的部分中找出最小元素：
- 对没画直线的行的各元素都减去这最小元素；
- 对画直线的列的各元素都加这最小元素，使行（未画直线）与列（画线）交叉处的0不变，这样得到新系数矩阵（它的最优解当然和原问题的相同）。
- 如已有n个不同行不同列的0元素，则求解过程已完成；
- 如还没得到n个，则回到第三步重复进行。

匈牙利法

$$\begin{bmatrix} 12 & 7 & 9 & 7 & 16 \\ 8 & 9 & 6 & 6 & 6 \\ 7 & 17 & 12 & 14 & 12 \\ 15 & 14 & 6 & 6 & 10 \\ 4 & 10 & 7 & 10 & 6 \end{bmatrix} \longrightarrow \begin{bmatrix} 5 & 0 & 2 & 0 & 9 \\ 2 & 3 & 0 & 0 & 0 \\ 0 & 10 & 5 & 7 & 5 \\ 9 & 8 & 0 & 0 & 4 \\ 0 & 6 & 3 & 6 & 2 \end{bmatrix}$$

匈牙利法

- 上述匈牙利法为解最小化问题的方法。
- 对于最大化问题：

  可作一新矩阵

  $B = (b_{ij})$　使　　$b_{ij} = M - c_{ij}$

  其中M是足够大的常数（例如，取c_{ij}中最大的元素即可），这时$b_{ij} \geq 0$合于匈牙利法的条件。

$$\min z' = \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij} x_{ij}$$

  所得的极小解（x_{ij}）就是原问题的极大解 。

匈牙利法

$$\sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij}x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n}(M - c_{ij})x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} Mx_{ij} - \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} = nM - \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

- 所以当$\Sigma\Sigma b_{ij}x_{ij}$ 取得极小时，必使 $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$ 取得极大。

一些说明

- 分支界限法类似于回溯法， 也是一种在问题的解空间树T上搜索问题解的算法。
- 但在一般情况下，分支界限法与回溯法的求解目标不同。回溯法的求解目标是找出T中满足约束条件的所有解，而分支界限法的求解目标是找出满足约束条件的一个解，或是在满足约束条件的解中找出使某一目标函数值达到极大或极小的解。

一些说明

- 由于求解目标不同，导致分支界限法与回溯法在解空间树T上的搜索方式也不相同。

  回溯法以深度优先的方式搜索解空间树T，而分支界限法则以广度优先或以最小耗费优先的方式搜索解空间树T。
- 分支界限法的搜索策略是，在扩展结点处，先生成其所有儿子的结点(分枝)，然后再从当前的活结点表中选择下一个扩展结点。为了有效地选择下一扩展结点，以加速搜索的进程，在每一活结点处，计算一个函数值（界限），并根据这些已计算出的函数值，从当前活结点表中选择一个最有利的结点作为扩展结点，使搜索朝着解空间树上有最优解的分枝推进，以便尽快地找出一个最优解。

一些说明

- 从活结点表中选择下一扩展结点的不同方式导致不同的分支界限法。最常见的有以下两种方式：

1. 队列式（FIFO）分支界限法

   队列式分支界限法将活结点表组成一个队列，并按队列的先进先出原则选取下一个结点为当前扩展结点。

2. 优先队列式分支界限法

   优先队列式的分支界限法将活结点表组织成一个优先队列，并按优先队列中规定的结点优先级选取优先级最高的下一个结点成为当前扩展结点。

Branch and Bound

Minimum Cost Network Flow Problem

问题描述

- 该类设计问题的目标是，在一组节点之间设计一最小耗费的网络以满足所有的流量需求。

- 描述该问题的信息：
  - 每个源节点和目的节点对(O-D)之间的流量需求
  - 在每条边(i,j)上负载流量的价值函数

    $C_{ij}=f(t_{ij})$

The Problem

- let $\mathbf{G}$ be the set of all undirected graphs on n nodes with G a member of this set. G is represented by its upper triangular node-node adjacency matrix B with elements b_{ij}.

- The problem is to find a $G^* \in \mathbf{G}$ which minimizes the cost of transporting required origin-destination flows subject to specified arc and node capacity, node degree and chain hop limit constraints.

问题描述

- 设**G**表示在n个点上的所有无向图的集合，G是该集合中的一个元素。G以节点的邻接矩阵B的上三角阵来表示，其元素为$b_{ij.}$

- 该问题就是要找到一个$G^* \in \mathbf{G}$，使得在满足给定的边容量限制的前提下，实现传输所有源节点到目的节点的流量需求所用的花费最小。

Next ?

The Problem

- Formulation-1
- Interpretation of the formulation
- Too complicated? Then what?

Transform and conquer

- Problem transformation
  - Constant arc costs
  - Tree solution

Consider a connected undirected graph $G = (V, A, d, c)$ with node set $V = \{1,2,...,n\}$ and arc set A.

There is a certain amount of traffic demand d_{ij} between each pair of nodes i and j in V.

c_{ij} represents the cost of using arc (i, j) in A.

To satisfy the capacity constraint, the flow on any arc (i, j) must not exceed a given capacity k_{ij}.

By means of these definitions, the problem is to find a minimum cost tree spanning the node set V where all traffic demand are satisfied, and the capacity constraint k_{ij} is observed for every arc (i, j).

Let f_{ij} denote the total flow on arc (i, j). Define $x_{ij} = 1$, if arc (i, j) is included in the solution, and $x_{ij} = 0$, otherwise.

给定一个连通的无向图G=(V,A,d,c)，

点集V={1,2,…,n}，边集A.

V中的每个节点对i和j间，有一定的流量要求d_{ij}.

c_{ij}表示建立A中的边(i,j)的花费。

每条边(i,j)上的负载不能超过一个给定值k_{ij}.

通过以上这些定义，该问题就是要找到一个可以扩展节点集V的最小生成树，同时满足所有的流量要求，而且每条边(i,j)都符合负载限制k_{ij}.

令f_{ij}表示经过边(i,j) 的总流量。定义

x_{ij}=1,当边(i,j)被选中在解中；

x_{ij}=0,当边(i,j)没有被选中。

Problem II

- Formulation-2

- Interpretation of the formulation

Branch and Bound

- What is the first task?

Branch and Bound

- What is the first task?
- Constructing a search tree
- How to construct the search tree?

Branch and Bound

- What is the first task?
- Constructing a search tree
- How to construct the search tree?

  -- Innumerate，Organize，Bound

Branch and Bound

The search tree

- a tree spanning n nodes must have exactly n-1 arcs
- consider the solution as a collection of n-1 arcs (for n-node problem) chosen from all candidate arcs
- introduce an arc-oriented branch and bound method, branching by using an arc or not.

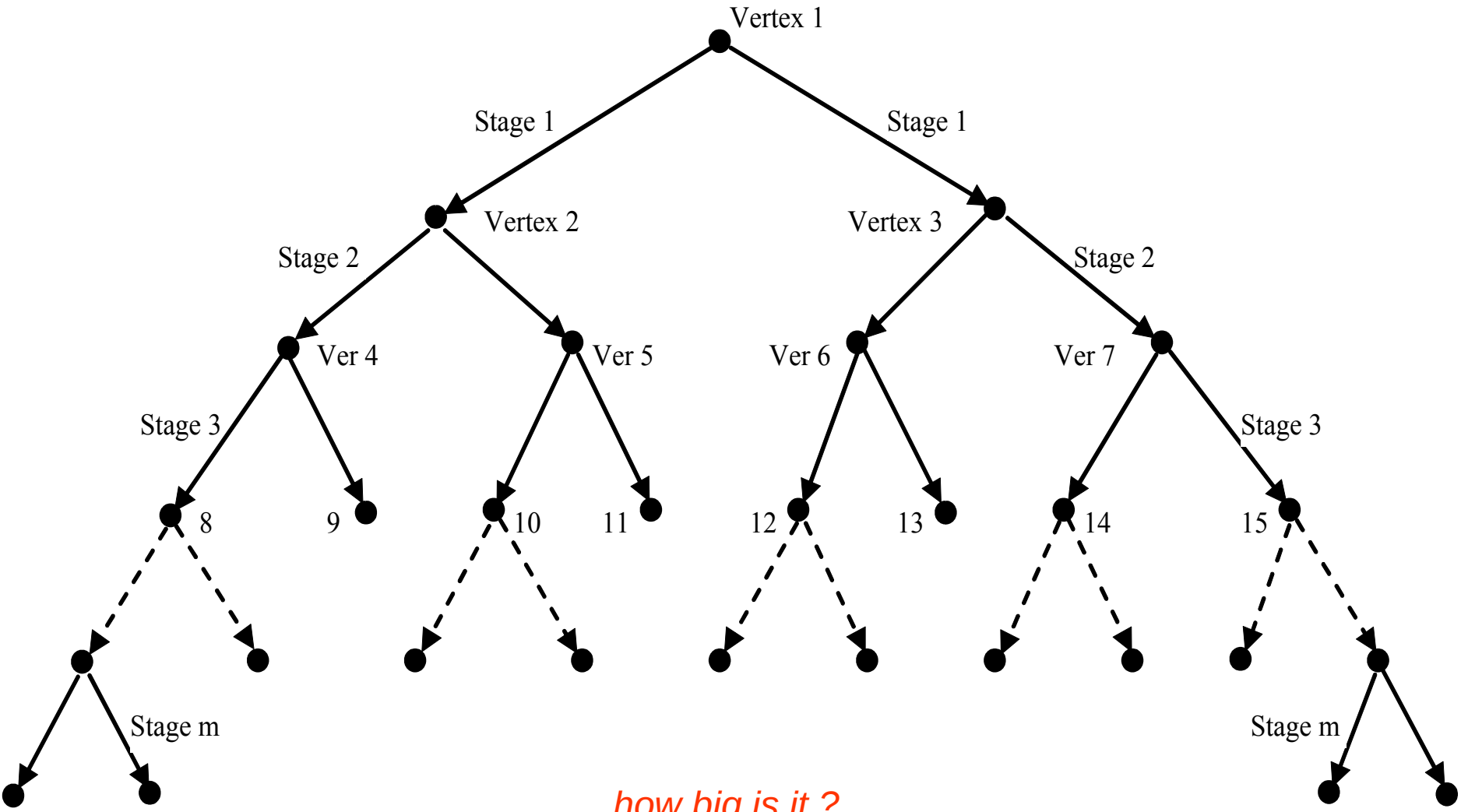
A Binary Search Tree

- Define an m-stage binary search tree for problem with n nodes, where, for complete graph problem, m equals the number of pairs of nodes.
- Give each of the arcs an arc number, from 1 to m, as identification.
- The branching within each stage corresponds to the decision of either adding a certain arc to the solution arc set or not.
- Define traveling along left branch represents for choosing an arc and traveling along right branch represents for not choosing an arc.

二叉搜索树

- 对于n个节点的问题定义一个m层的二叉搜索树，对于完全图问题，m等于节点对的个数。
- 为每条边编号，从1到m。
- 每层的分支对应于是否把某条边添加到解集中的决策。
- 定义沿左分支前进表示选择该边，沿右分支前进表示不选该边。

The search tree



how big is it ?

The search tree

- KEEP The Tree in Mind

The search tree

- KEEP The Tree in Mind
- What’s next?

The search tree

- KEEP The Tree in Mind
- What’s next?
- Regulations for traversing the search tree.

Traversing the search tree

It is obvious that an optimal solution can be found after having visited all the vertices of this binary tree.

1. Always start from the root of the tree (Vertex 1).
2. Always go along the left branch first when descending.
3. Keep descending and backtrack when any of the following situations occurs:
   a. Current selected arcs make the solution infeasible.
   b. A feasible solution has been found.
4. After backtracking to a vertex along its left branch, go forward along its right branch.
5. After backtracking to a vertex along its right branch, go backward to its parent vertex.

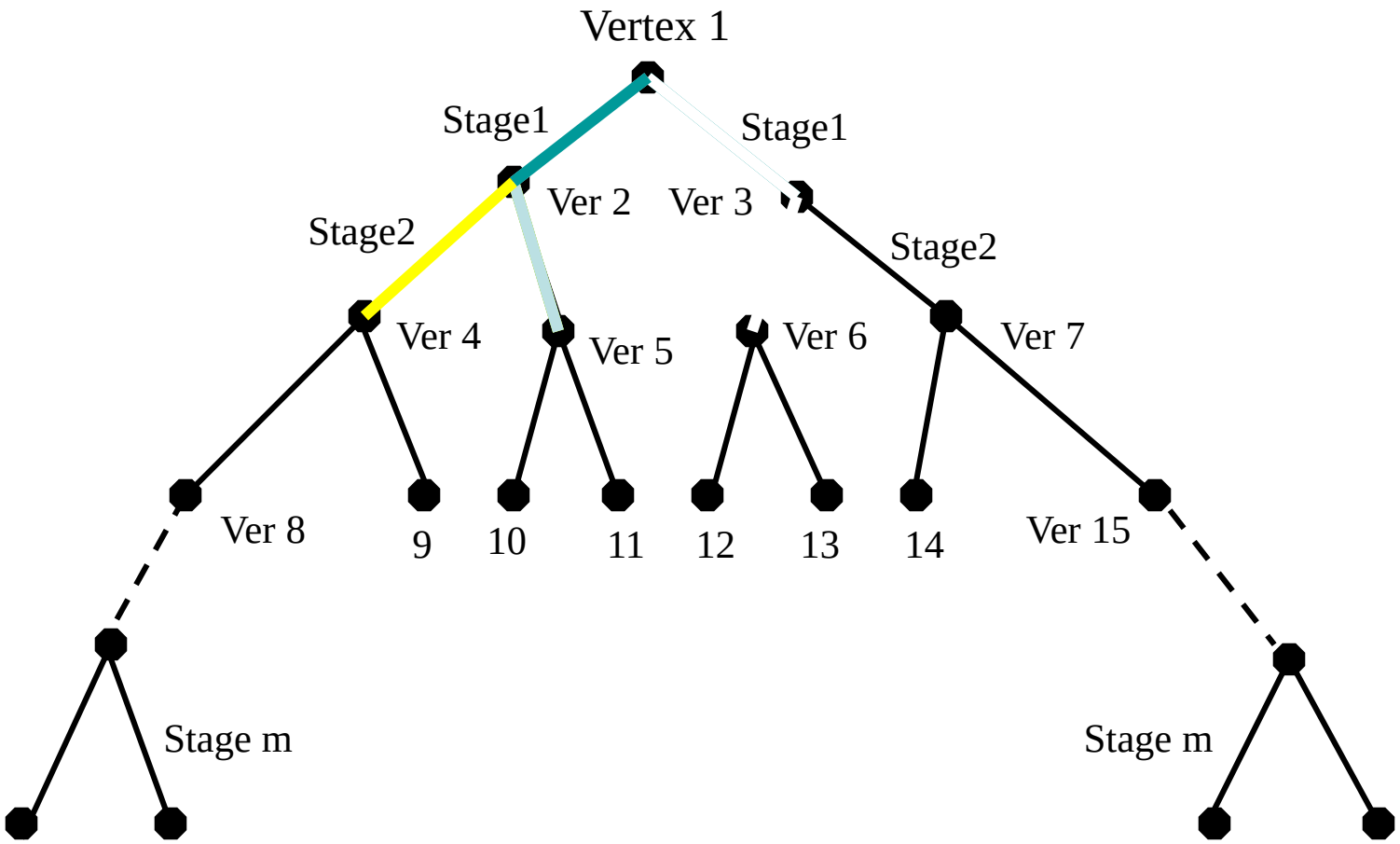
遍历搜索树

明显地，当该二叉树中的所有顶点都被访问过时，可以找到一个最优解。

1. 总是从树的根(Vertex 1)开始。
2. 当下行时，总是先沿左分支进行。
3. 当有如下情况之一发生时，进行回溯：
   a. 当前挑选的边使得解不可行。
   b. 已经找到一个解。

4. 当从左分支回溯到某顶点时，接着沿其右分支向下进行。
5. 当从右分支回溯到某顶点时，接着回溯到其父顶点。

Traversing the search tree



Traversing the search tree

- What’s next?

Traversing the search tree

- What's next?

- Pruning the search tree
  - bound