数据分析与算法设计
第12章 超越算法能力的极限（Coping with Limitations of Algorithm Power）

李旻
百人计划研究员
浙江大学 信息与电子工程学院
Email: min.li@zju.edu.cn

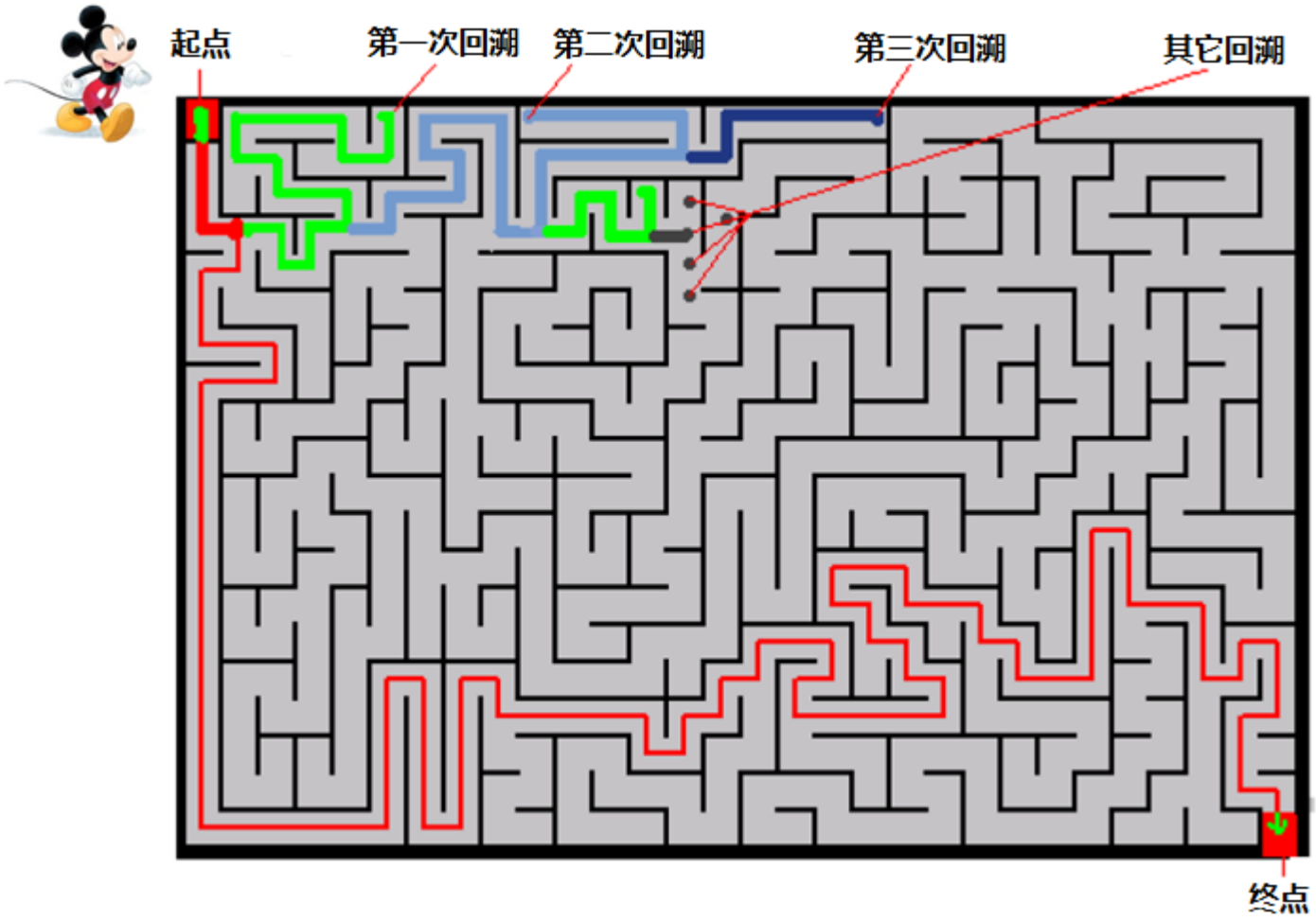
求解复杂组合问题

- 两种常见的思路
  - 使用某些策略能保证准确地解决问题，但并不能保证在多项式时间内完成
    - 蛮力法：遍历所有可能的候选解，从而确定最优解
    - 动态规划：可用于求解部分问题，如背包问题
    - 回溯法（Backtracking）：蛮力法的改进；排除一些不需要考虑的部分解，减小解的搜索空间；对于许多问题的实例可在合理的时间内解决，但最坏的情况下仍然是指数级
    - 分支界限法（branch-and-bound）：回溯法的改进
  - 使用一个近似算法，可以在多项式时间内找到一个近似的（次优）解决方案

目录

- 回溯法
  - n-皇后问题
  - 哈密顿回路问题
  - 子集和问题
- 分支界限法
  - 分配问题
  - 背包问题
  - 旅行商问题
- NP困难问题的近似算法
- 基于人工智能的求解方法

回溯法



回溯法

- 基本思想：每次只构造解的一个分量，然后评估该部分构造解
  - 如果可以进一步构造符合问题约束的部分解，则继续构造解的下一个分量
  - 如果无法对下一分量进行合法的选择，则进行回溯，把部分构造解的最后一个分量替换成它的下一个选择

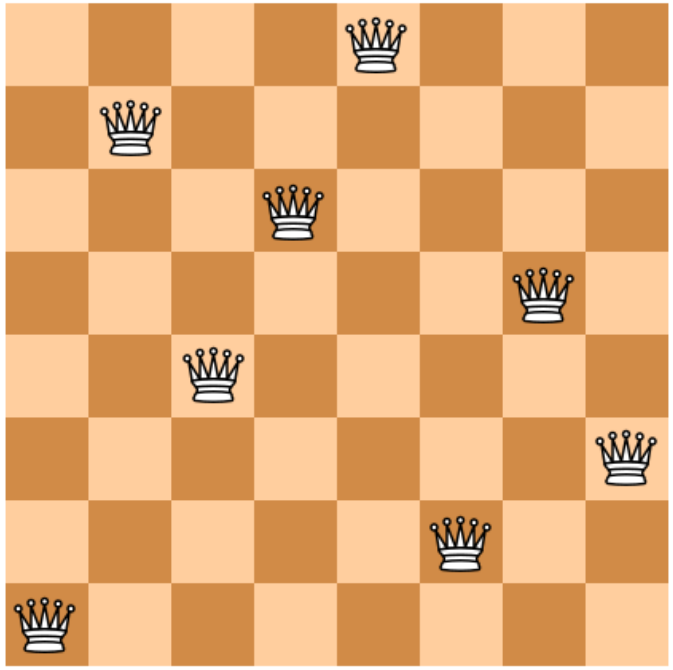
从部分解往前走，能进则进，
不能进则退回来，换一条路再试。

回溯法

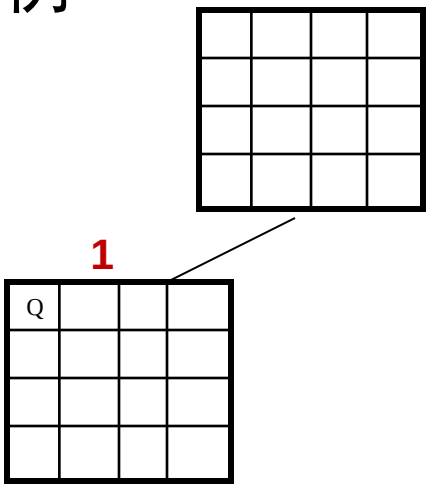
- 算法实现：状态空间树
  - 根：查找解之前的初始状态
  - 第k层节点：代表对解的第k个分量所做的选择
  - 区分有希望与无希望的节点：一个部分构造解仍然可能导致一个完整解，则该部分解对应的节点是有希望的，反之是无希望的节点
  - 空间树生长方式：深度优先
    - 如果当前节点是有希望的，则通过向部分解添加下一个分量的第一个合法选择，生成该节点的一个子女
    - 如果当前节点变得没希望，则回溯到该节点的父母，考部分解的最后一个分量的下一个可能选择，如这种选择不存在，则再回溯到上一层

n-皇后问题

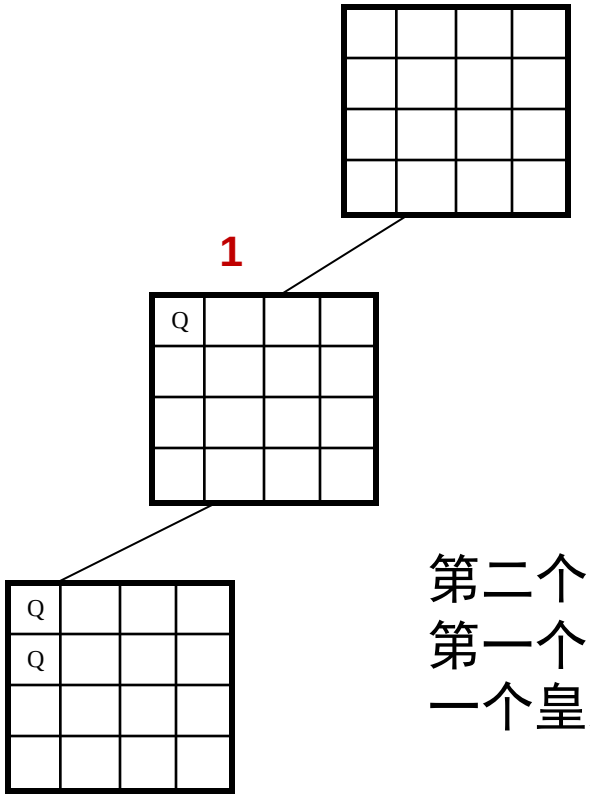
- 问题描述：n个皇后放在一个$n \times n$的棋盘上，要求任何两个皇后不能攻击，即它们不在同行、同列或同一对角线上。设计一个算法求解一个可行的放置位置。



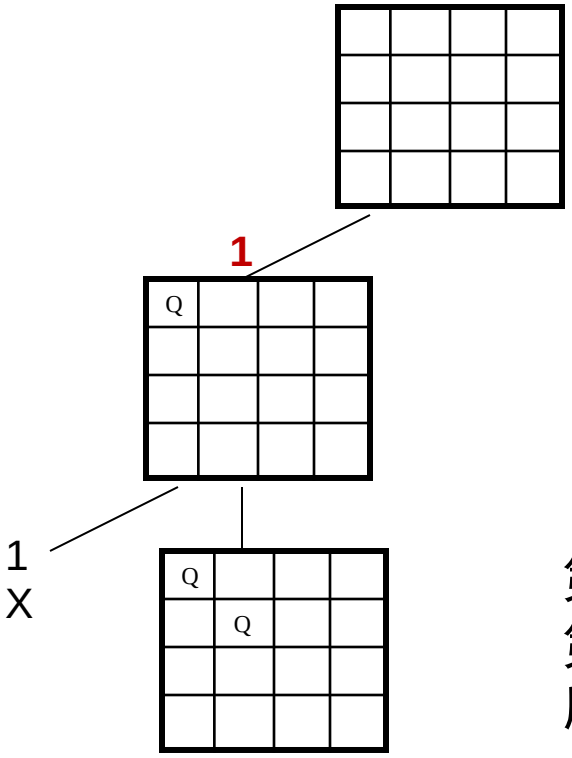
- 回溯法：以n=4为例



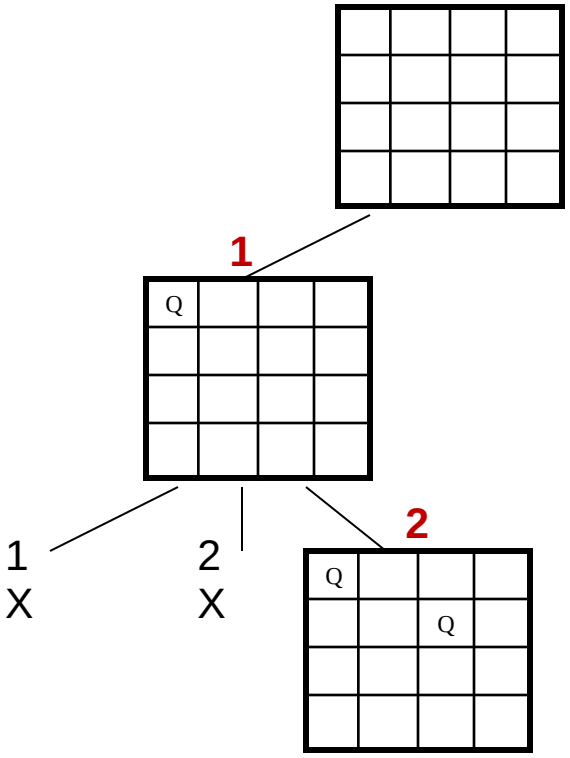
首先放入（1，1）位置



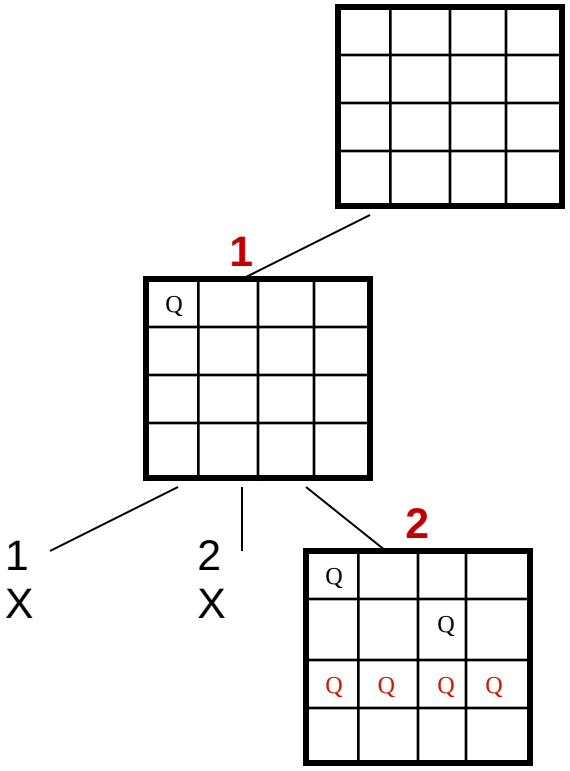
第二个皇后放在（2，1）位置，与第一个皇后位于同一列，所以被第一个皇后攻击。失败。



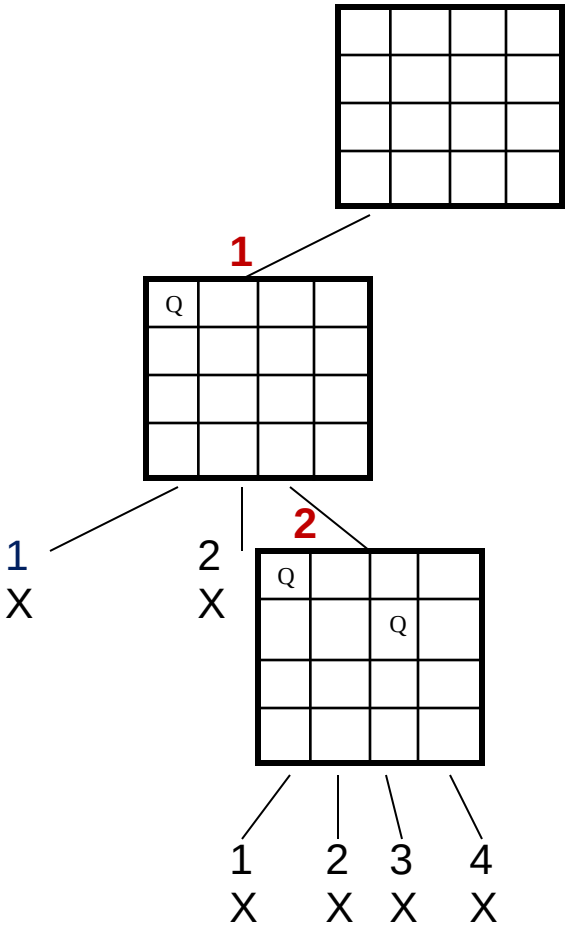
第二个皇后放入（2，2）位置，与第一个皇后位于同一条对角线上，所以被第一个皇后攻击。失败。

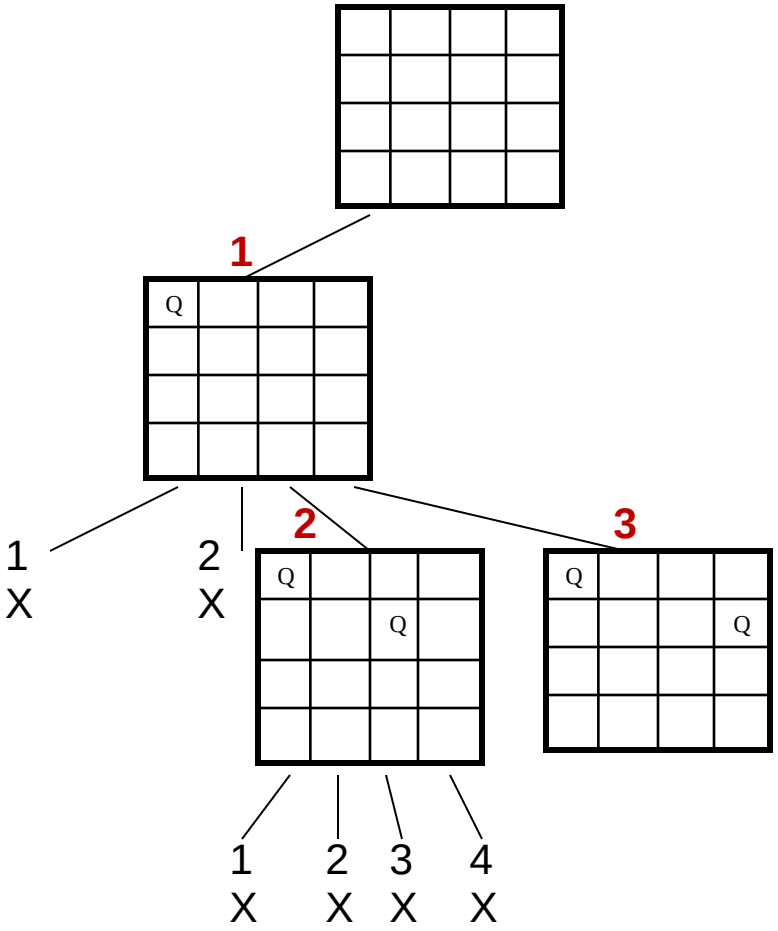


放入（2，3）位置，

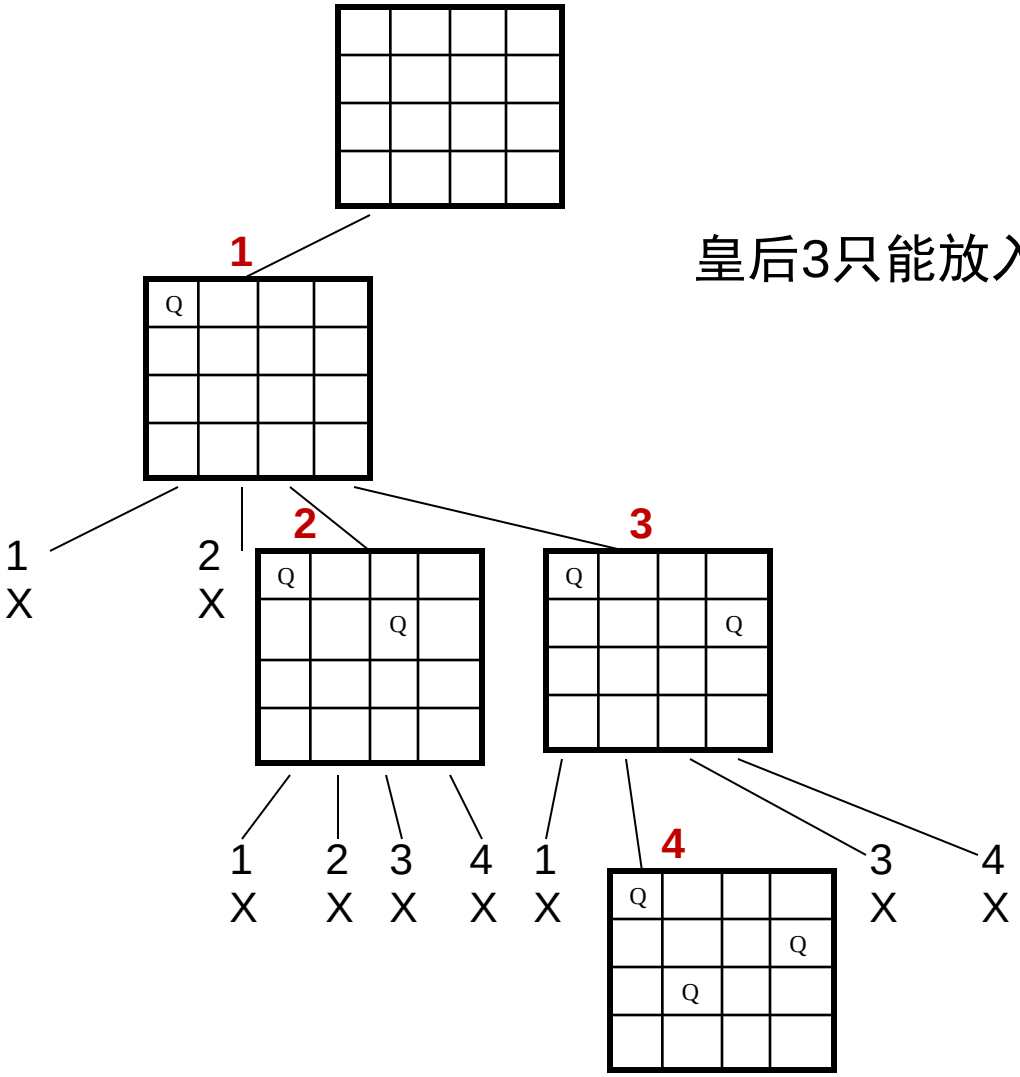


但对于皇后3来说这是一个死胡同，因为皇后3将没位置放。所以失败！

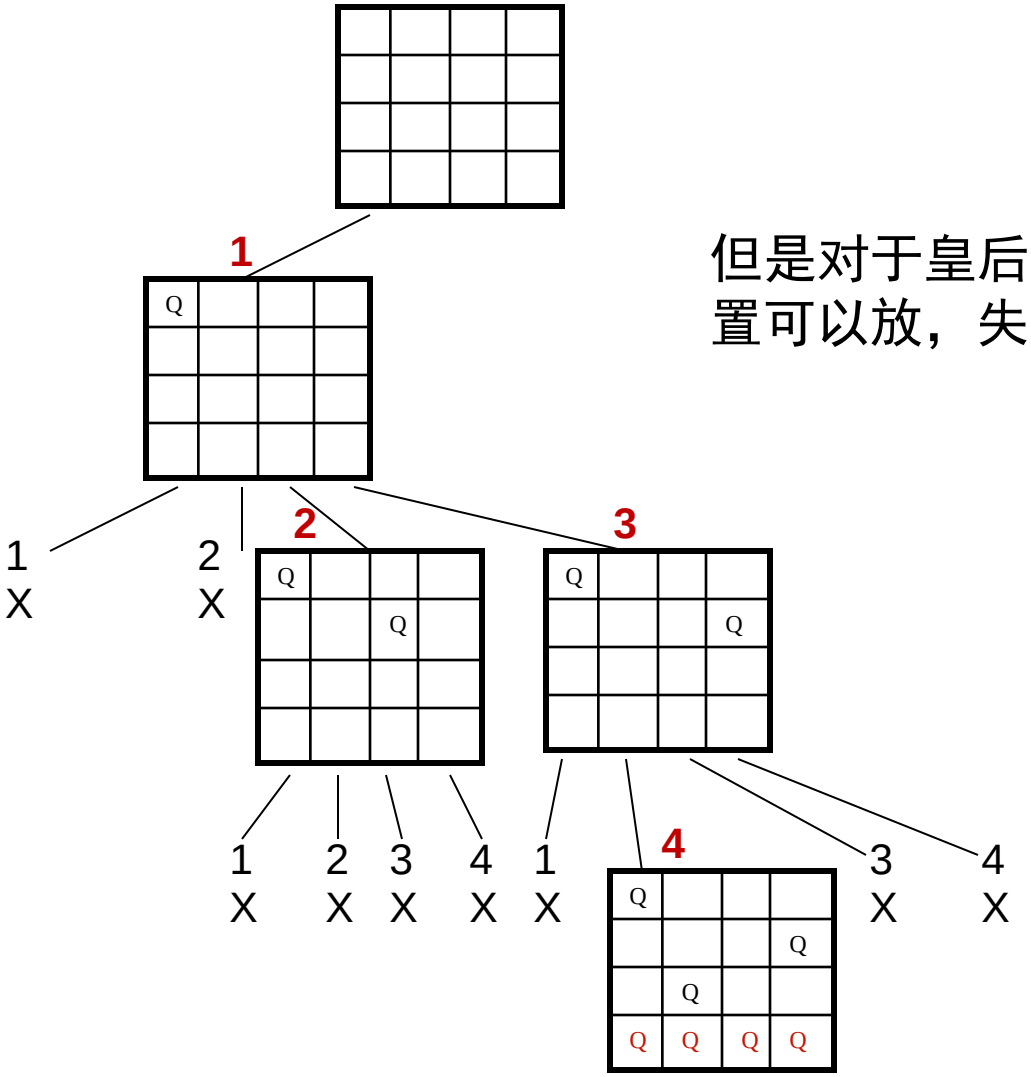

1
Q
1
X
2
X
2
Q
Q
1
X
2
X
3
X
4
X



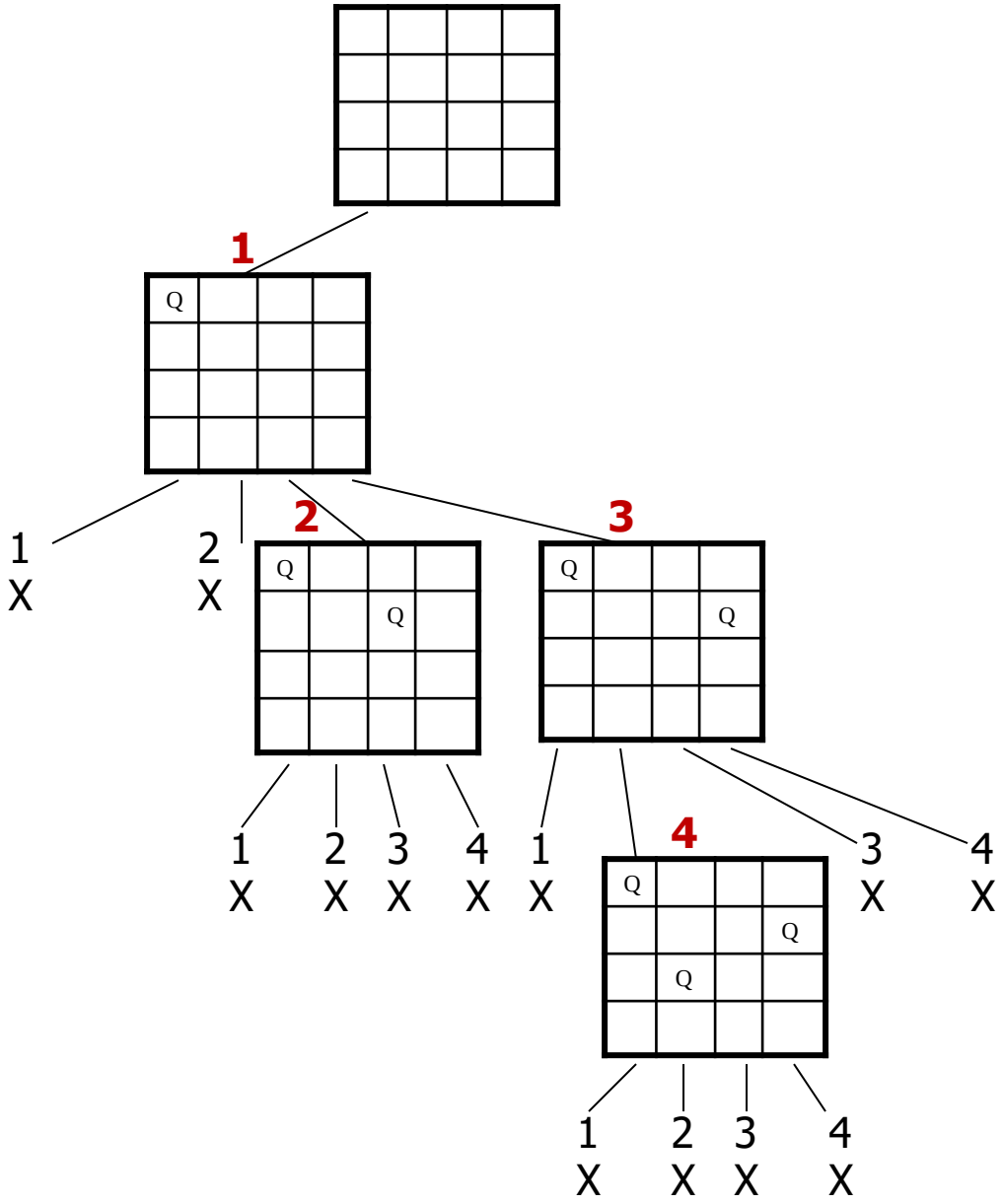
所以将皇后2放入（2，4）位置



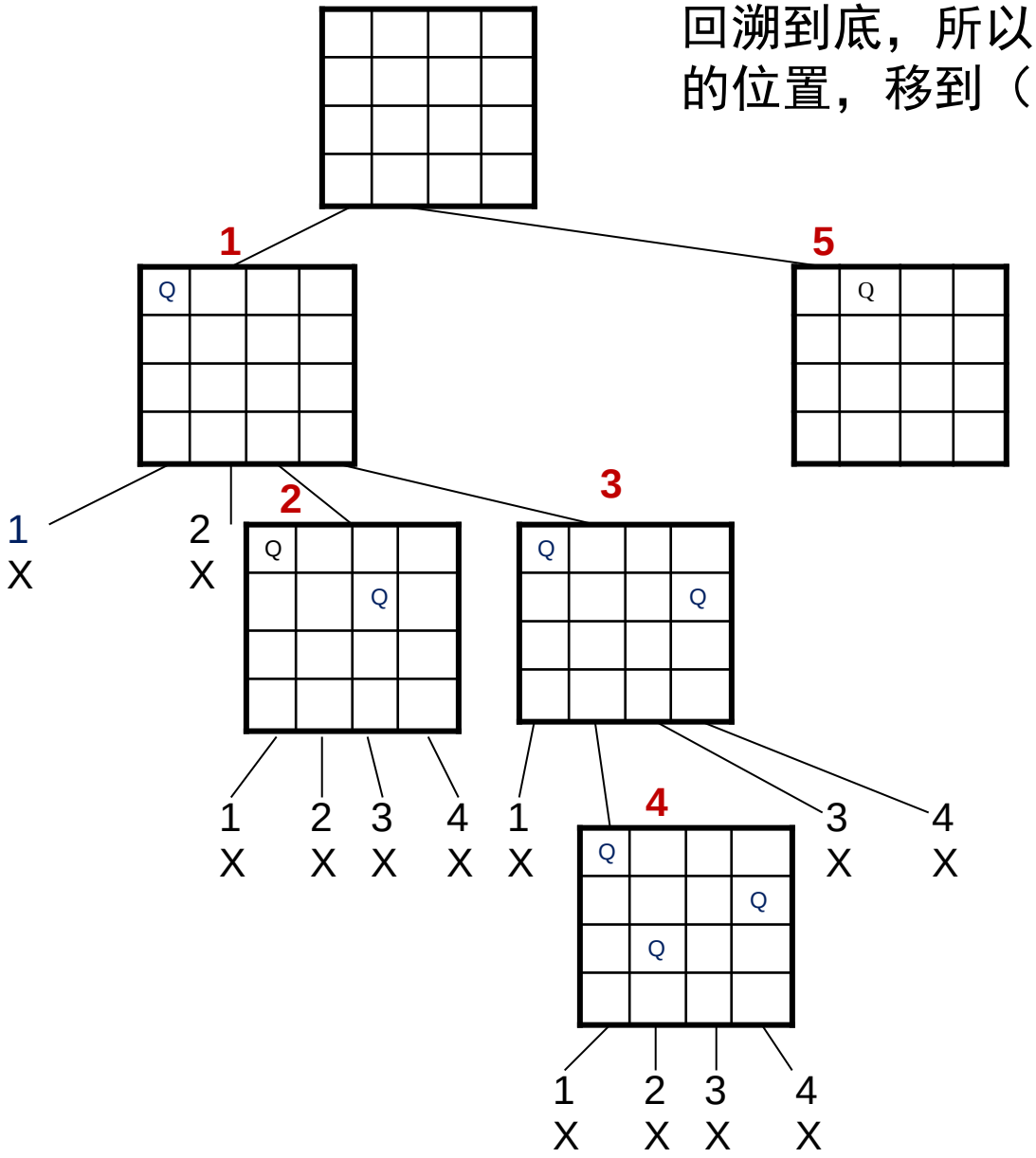
皇后3只能放入（3，2）位置



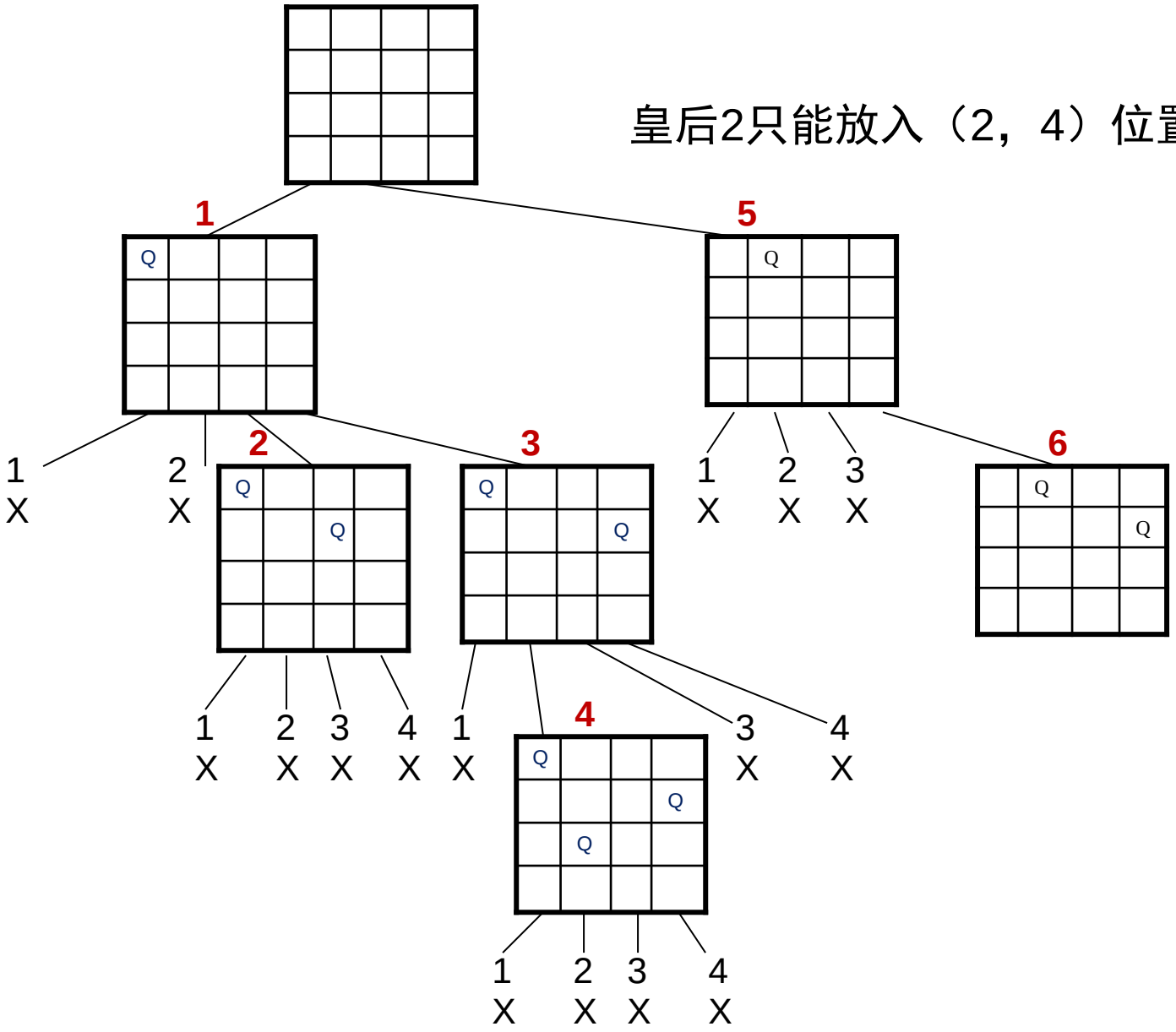
但是对于皇后4，没有位置可以放，失败！

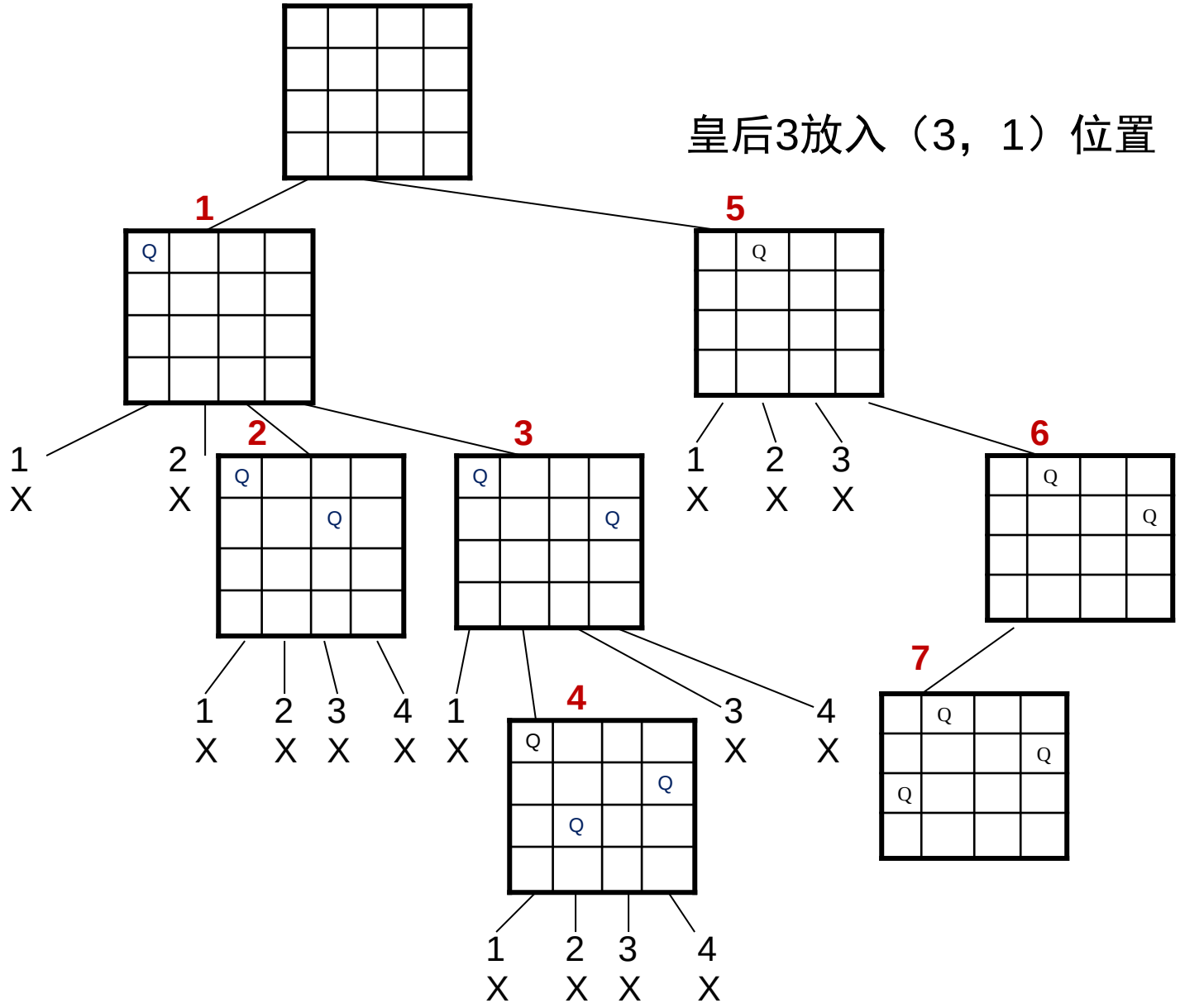

1
Q
2
Q
Q
3
Q
Q
4
Q
Q
Q
1 X
2 X
1 X
2 X
3 X
4 X
1 X
3 X
4 X
1 X
2 X
3 X
4 X

回溯到底，所以只能移动皇后1的位置，移到（1，2）位置

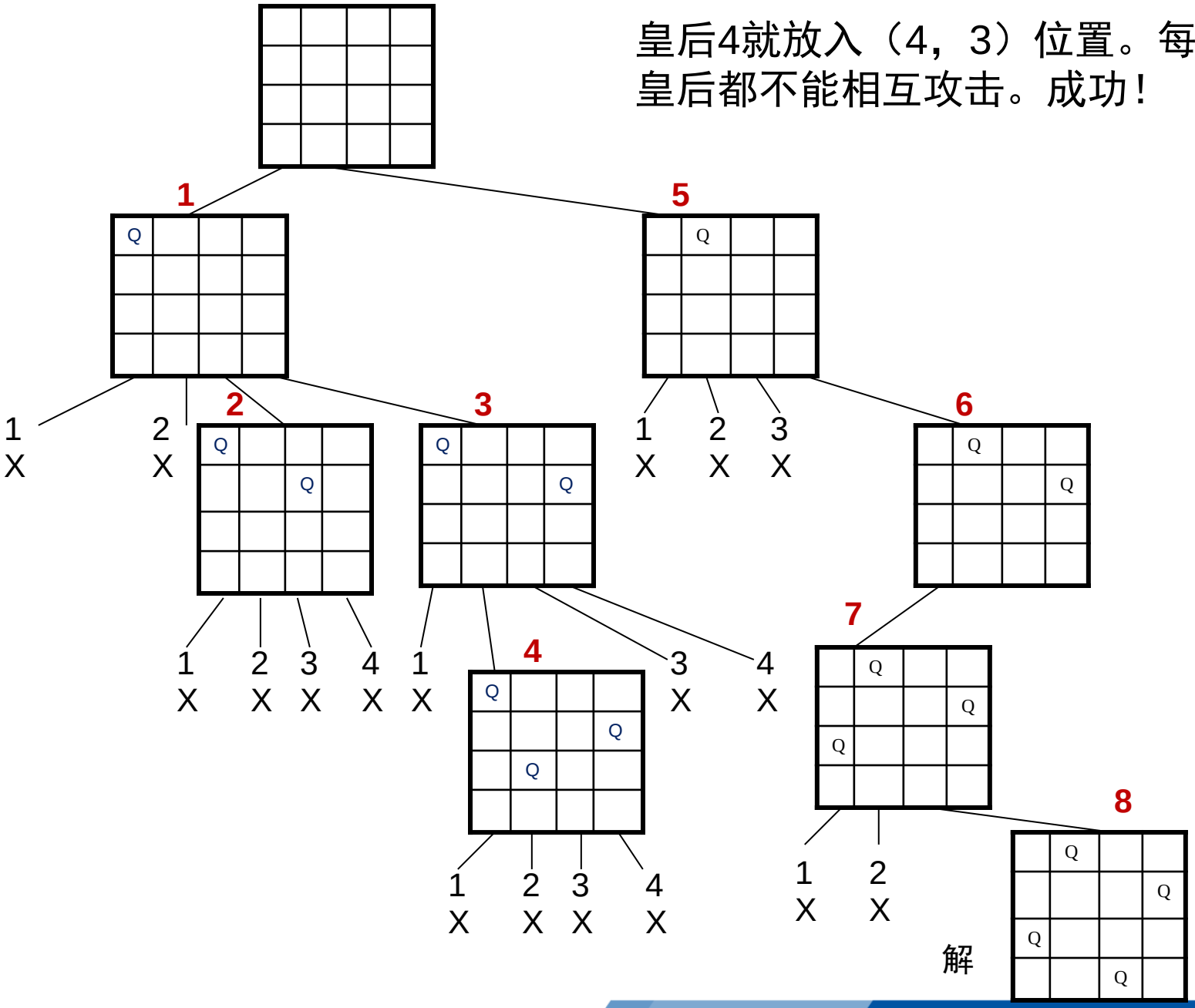


皇后2只能放入（2，4）位置。




皇后3放入（3，1）位置
1
5
2
3
6
7
4
Q
X

皇后4就放入（4，3）位置。每个皇后都不能相互攻击。成功！



n-皇后问题

- 衍生问题：对于任意的$n \geq 4$皇后问题，共有多少种可行解？是否有算法可以生成所有的可行解？
- 对于任意的$n \geq 4$皇后问题都可在线性时间内求解

若$n \bmod 6 \neq 2$且$n \bmod 6 \neq 3$,则可通过(A1)或(A2)导出解序列;

若$n \bmod 6 \neq 2$或$n \bmod 6 \neq 3$,则可通过(B1)或(B2)或(B3)或(B4)导出解序列

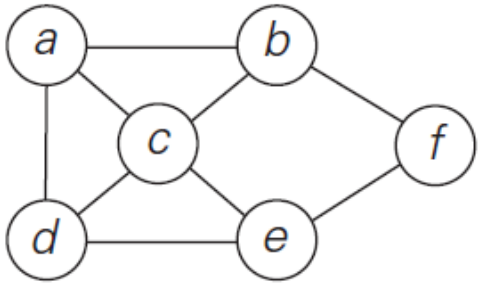
其中，当n是偶数时，$m = \frac{n}{2}$; 当n是奇数时，$m = \frac{n-1}{2}$

$$
\begin{cases}
(A1-n\text{偶}):[2,4,6,8,\ldots,n],[1,3,5,7,\ldots,n-1] \\
(A1-n\text{奇}):[2,4,6,8,\ldots,n-1],[1,3,5,7,\ldots,n-2] \\
(B1-n\text{偶}m\text{偶}):[m,m+2,\ldots,n],[2,4,6,\ldots,m-2],[m+3,m+5,\ldots,n-1],[1,3,5,\ldots,m+1] \\
(B2-n\text{偶}m\text{奇}):[m,m+2,\ldots,n-1],[1,3,5,\ldots,m-2],[m+3,m+5,\ldots,n],[2,4,6,\ldots,m+1] \\
(B3-n\text{奇}m\text{偶}):[m,m+2,\ldots,n-1],[2,4,6,\ldots,m-2],[m+3,m+5,\ldots,n-2],[1,3,5,\ldots,m+1],[n] \\
(B4-n\text{奇}m\text{奇}):[m,m+2,\ldots,n-2],[1,3,5,\ldots,m-2],[m+3,m+5,\ldots,n-1],[2,4,6,\ldots,m+1],[n]
\end{cases}
$$

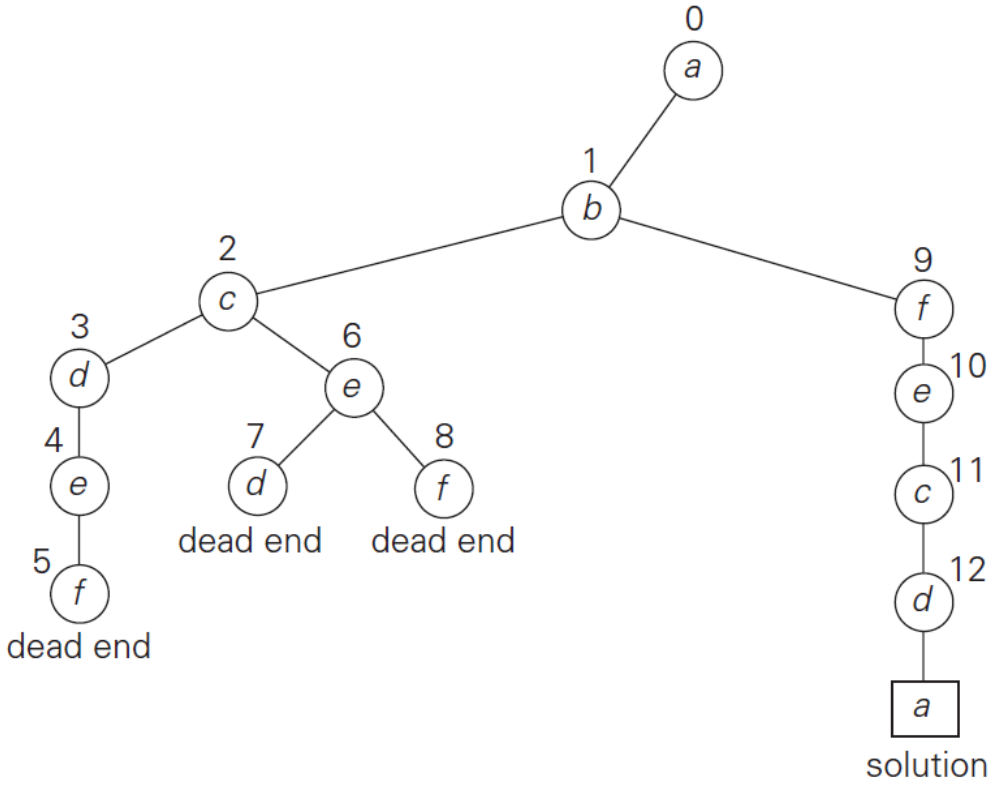
E.J. Hoffman, J.C. Loessi, R.C. Moore, Constructions for the Solution of the m Queens Problem, Mathematics Magazine, 1969, Vol.42, No.2, pp.66-72

哈密顿回路问题

- 哈密顿回路：一条起止于相同顶点的路径，并且只经过其它所有的顶点一次



（a）图



（b）状态空间树及求解过程

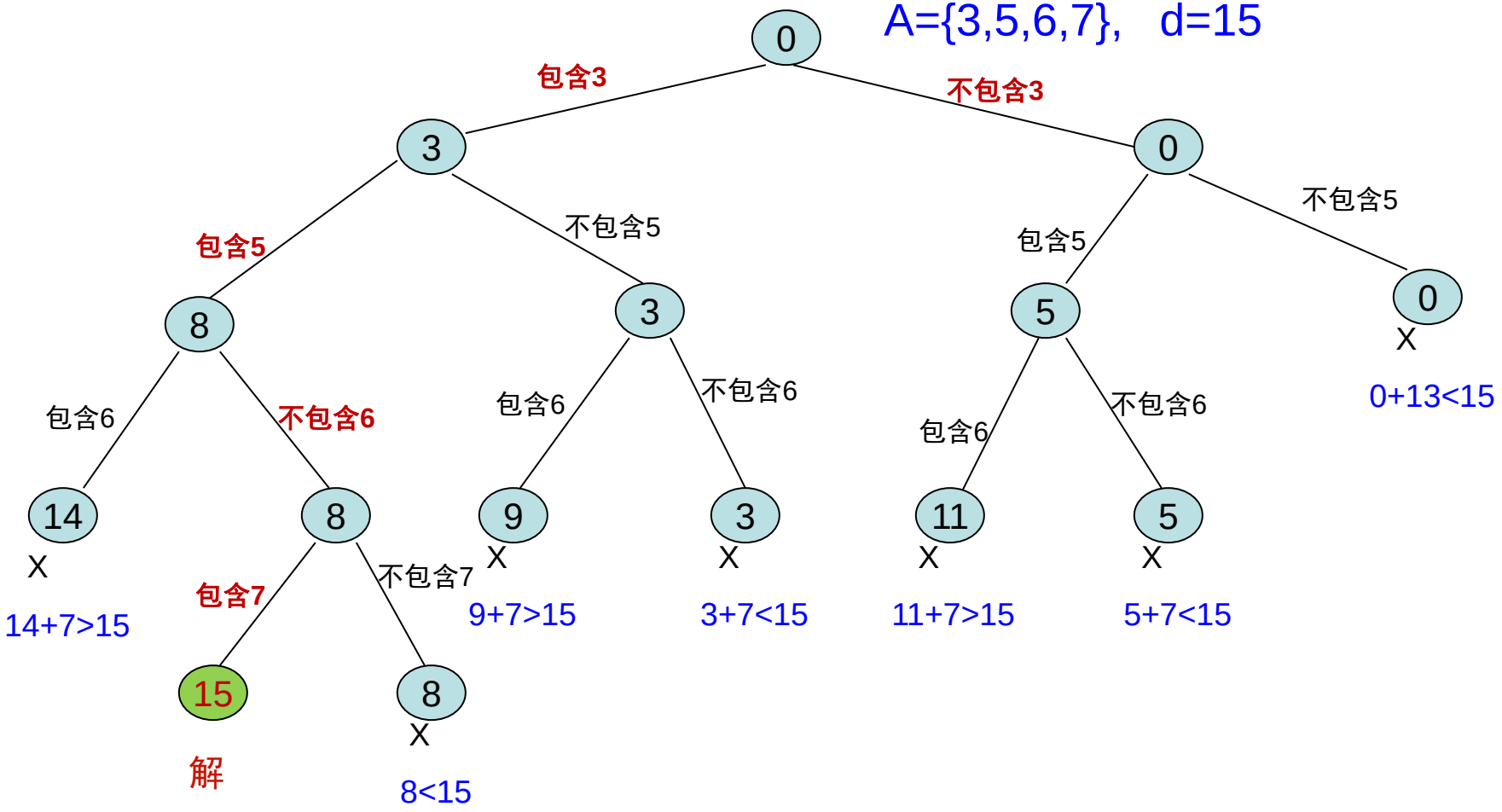
子集和（Subset-Sum）问题

- 问题描述：给定一个正整数集合$A = \{a_1, a_2, \ldots, a_n\}$及某个整数$d$, 求该集合的子集，要求其元素和等于d
  - $A = \{3,5,6,7\}, d = 15$: $\{3,5,7\}$

- 回溯算法：集合元素先按升序排序，再构造状态空间树
  - 根的左右子女分别代表当前子集中包含或不包含a_1，以此类推
  - 从根到某个第i层节点的路径，指出了该节点所代表的子集中包含了前i个元素中的哪些元素
  - 将这些元素的和s记录在节点中，如果$s = \mathrm{d}$, 则求解成功，否则，当下面不等式中的任何一个成立，则将该节点作为无希望的节点终止掉

$$s + a_{i+1} > d \text{ (和 } s \text{ 太大)}$$

$$s + \sum_{j=i+1}^{n} a_j < d \text{ (和 } s \text{ 太小)}$$


A={3,5,6,7}, d=15
0
包含3
不包含3
3
0
包含5
不包含5
包含5
不包含5
8
3
5
0
包含6
不包含6
包含6
不包含6
包含6
不包含6
14
8
9
3
11
5
X
X
X
X
X
X
14+7>15
包含7
不包含7
9+7>15
3+7<15
11+7>15
5+7<15
0+13<15
15
8
X
解
8<15

回溯法：小结

- 回溯法并不总是高效。在最坏的情况下，有可能必须生成一个呈指数增长（或者更快）的状态空间中所有的可能解
- 有一些技巧可以帮助缩小状态空间树的规模：
  - n皇后问题中，棋盘的对称性，第1个皇后放在左半边、右半边是对称的
  - 子集和问题中，先预排序
- 如何估计状态空间树的规模？
  - 随机路径法：
    - 构造解的每个分量时，都是从可能的子女中随机选一个，直到遇见叶子节点，记录相应的状态空间树的规模
    - 多做几次，计算它们的平均值
- 适用于困难的组合问题，有望在可接受时间内精确求解问题

讨论：24点游戏问题

- 问题描述：给定一个长度为4的整数数组 cards，每张卡片上都包含一个范围在［1, 9］的数字。请设计一个算法，判定是否可使用运算符［‘+’，‘–’，‘*’，'/'］和括号 '(' 和 ')' 将这些卡片上的数字排列成数学表达式，以获得值24。
  - 除法运算符 '/' 表示实数除法，而不是整数除法。例如，

    4 /(1 - 2 / 3)= 4 /(1 / 3)= 12
  - 每个运算都在两个数字之间。不能使用 “–” 作为一元运算符。例如，

    如果 cards =[1, 1, 1, 1]，则表达式 “–1 –1 –1 –1” 是 不允许的

示例 1:

```
输入: cards = [4, 1, 8, 7]
输出: true
解释: (8-4) * (7-1) = 24
```

示例 2:

```
输入: cards = [1, 2, 1, 2]
输出: false
```

讨论：24点游戏问题

- 解题思路
  - 从数组中选出2个数字，再选择一种运算操作，计算得到当前结果
  - 从数组剩余数字中再选1个数字及一种运算操作，与当前结果组合得到更新的结果。以此重复，直到数组中为空，看当前结果是否等于24。如果等于，则返回true；如果不等于，则回溯尝试不同操作和数字组合

1. 输入：给定的四个数字列表（如[4, 6, 8, 2]）和一个空的操作符列表。
2. 定义一个回溯函数，该函数将接收当前的计算结果和剩余的数字列表作为参数。
3. 在回溯函数中，首先检查剩余数字列表的长度：
   - 如果剩余数字列表为空并且当前的计算结果等于24，则找到了一个解，可以将操作符列表和数字列表返回。
   - 如果剩余数字列表为空但当前计算结果不等于24，则回溯到上一步。
   - 如果剩余数字列表不为空，则执行以下步骤：
     - 遍历剩余数字列表中的每个数字。
     - 将当前数字与剩余数字列表中的其他数字进行交换位置，以确保每个数字都有机会作为第一个操作数。
     - 为每个数字执行以下操作：
       - 从剩余数字列表中移除该数字，并将其添加到操作符列表中。
       - 对剩余的数字列表和更新后的操作符列表调用回溯函数，传递更新后的计算结果。
       - 如果返回的结果表明找到了一个解，则可以将结果返回。
       - 否则，将操作符列表和数字列表恢复到之前的状态，以便尝试其他的操作数。
4. 返回所有找到的解。

讨论：24点游戏问题

```
function backtrack(nums, operators, currentResult):
    if nums is empty:
        if currentResult is 24:
            return operators
        else:
            return None

    for i from 0 to length of nums:
        selectedNum = nums[i]
        remainingNums = nums excluding selectedNum

        // 尝试不同的操作符和数字组合
        for operator in ['+', '-', '*', '/']:
            updatedOperators = operators + operator
            updatedResult = applyOperation(currentResult, selectedNum, operator)

            // 递归调用回溯函数
            result = backtrack(remainingNums, updatedOperators, updatedResult)

            if result is not None:
                return result

    return None
```

目录

- 回溯法
  - n-皇后问题
  - 哈密顿回路问题
  - 子集和问题
- 分支界限法
  - 分配问题
  - 背包问题
  - 旅行商问题
- NP困难问题的近似算法
- 基于人工智能的求解方法

分支界限法

- 回溯法应用于最优化问题求解中的进一步改进
- 和回溯法相比，分支界限法需要两个额外的条件：
  - 对于一棵状态空间树的每个节点所代表的部分解，需要提供一种方法，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边界值
  - 目前求得的最佳解的值

- 上述信息可以帮助更高效地判定当前节点是否有希望，如无希望，则终止沿着该节点的后续查找

分支界限法

- 路径查找终止条件：
  - 该节点的边界值不能超越目前最佳解的值。这是一个没有希望的节点，需要立即终止，也称为剪枝
  - 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，即是叶子
- 选择目前最有希望取得最佳值的节点，优先展开
- 回溯法只考虑上述条件的后面两条，只适用于寻找可行解，不适合用于寻找最优解

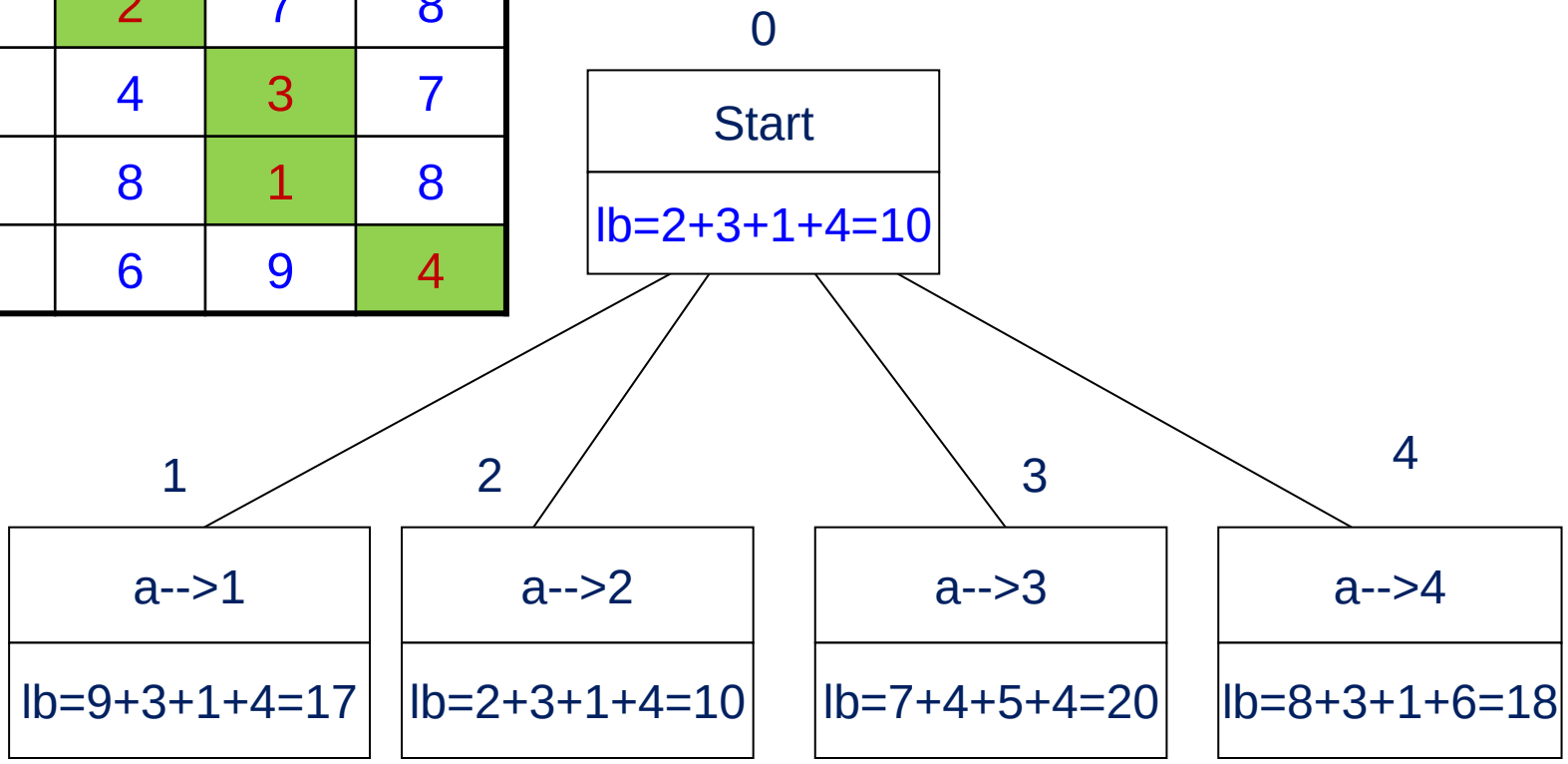
分配问题

- 有n个任务需要分配给n个人执行，一人一个任务。将第j个任务分配给第i个人的成本是C[i,j]，求总成本最小的分配方案

$$C=\begin{array}{c} \begin{matrix} \text{任务 1} & \text{任务 2} & \text{任务 3} & \text{任务 4} \end{matrix} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \end{array} \begin{matrix} \text{人员} & a \\ \text{人员} & b \\ \text{人员} & c \\ \text{人员} & d \end{matrix}$$

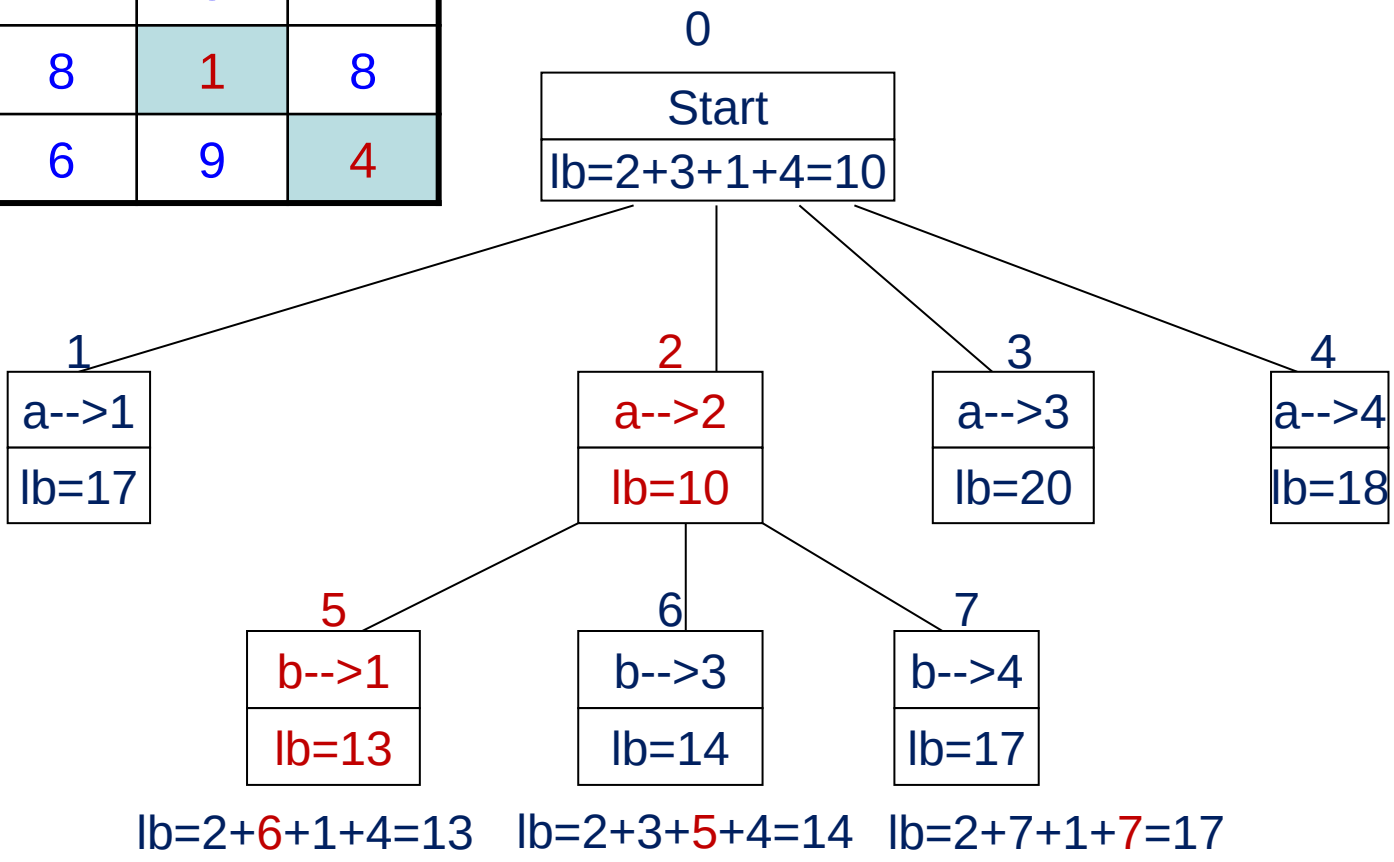
- 蛮力法、匈牙利算法、分支界限法…

	任务1	任务2	任务3	任务4
人员a	9	2	7	8
人员b	6	4	3	7
人员c	5	8	1	8
人员d	7	6	9	4

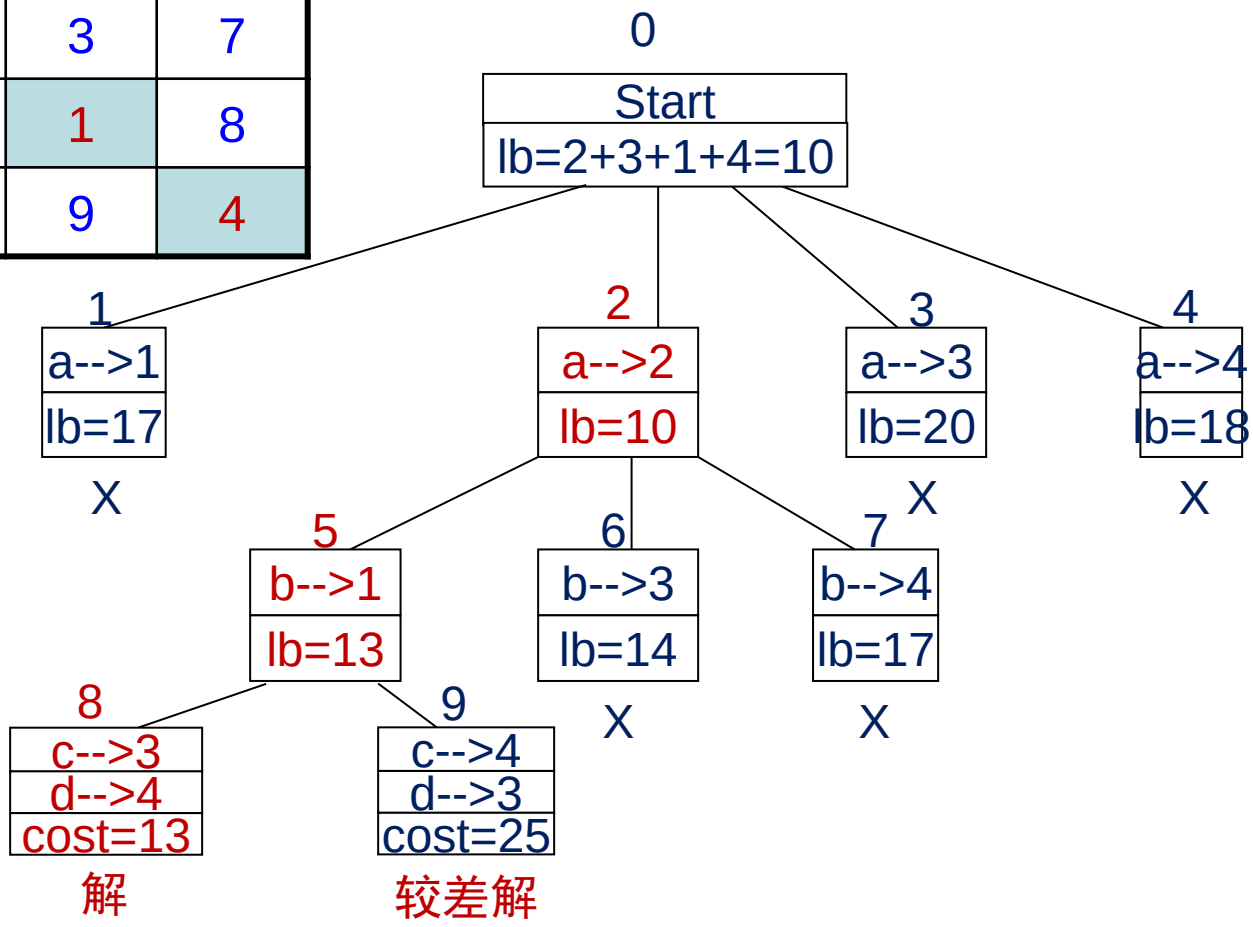


当前树的未终止的叶子节点中选择一个最有希望的，并生成它的全部节点

	任务1	任务2	任务3	任务4
人员a	9	2	7	8
人员b	6	4	3	7
人员c	5	8	1	8
人员d	7	6	9	4



	任务1	任务2	任务3	任务4
人员a	9	2	7	8
人员b	6	4	3	7
人员c	5	8	1	8
人员d	7	6	9	4



背包问题

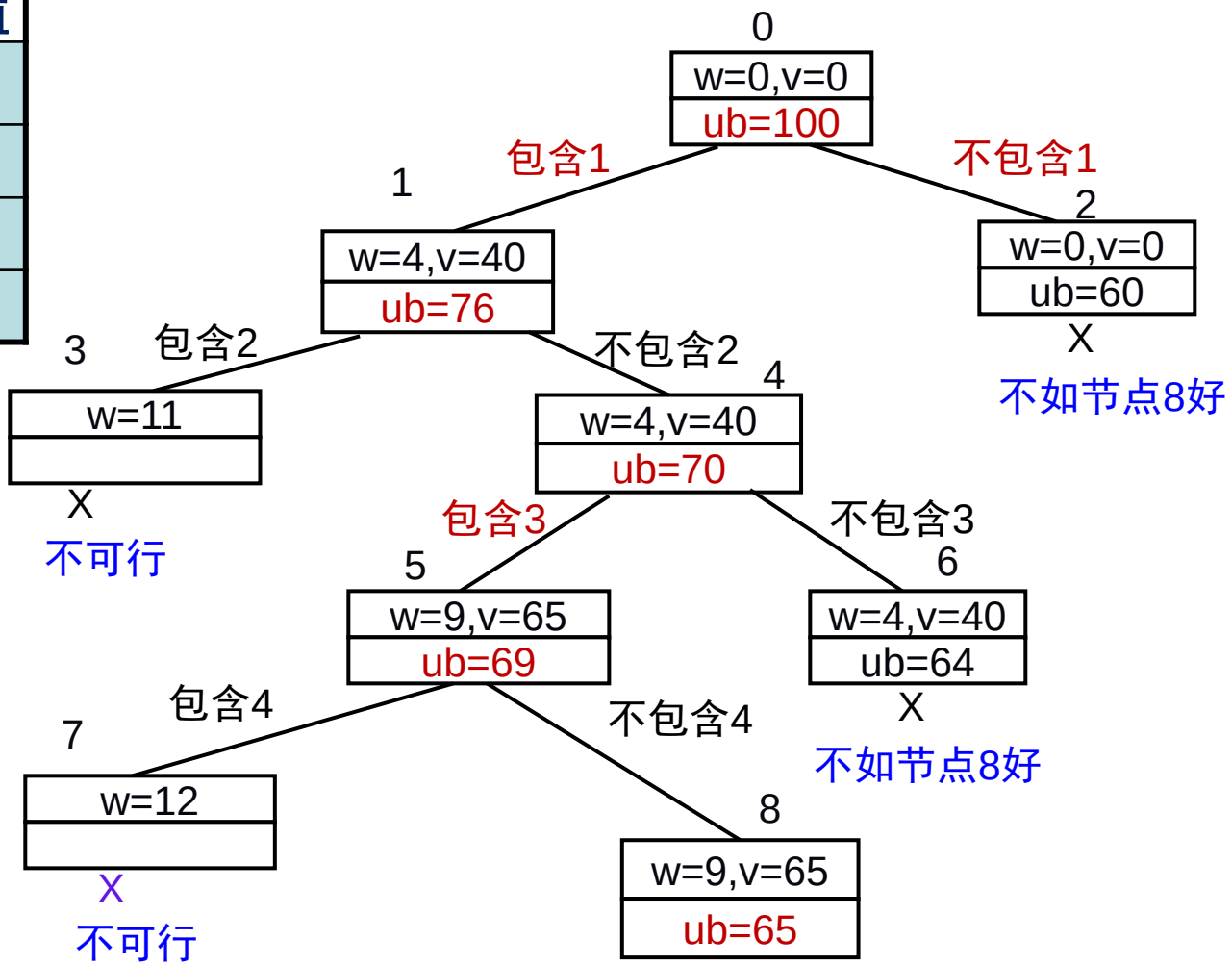
- 分支界限法：
  - 将物品按照单位回报（价值重量比）降序排列
  - 构造状态空间树，向左的分支表示包含下一个物品，向右的表示不包含下一个物品
  - 计算上界的简单办法：

    已有物品价值+剩下物品最高单位回报*剩余容量

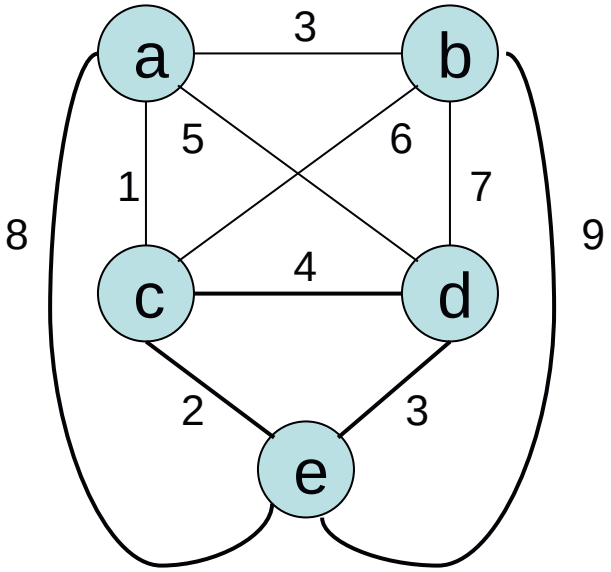
背包问题

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

背包的总承重为10



旅行商问题

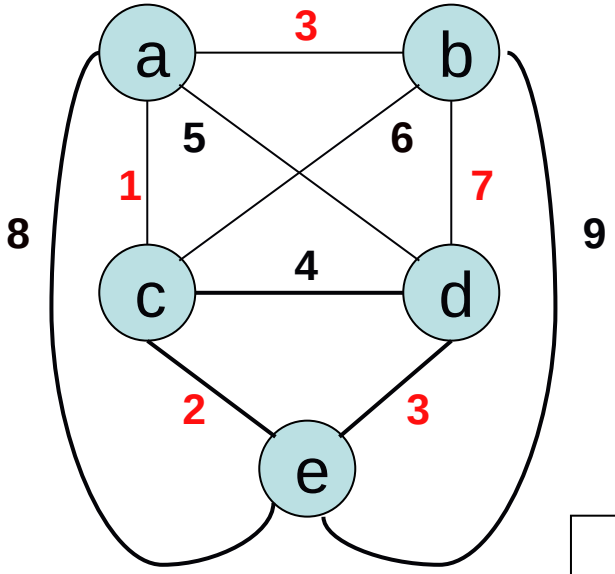


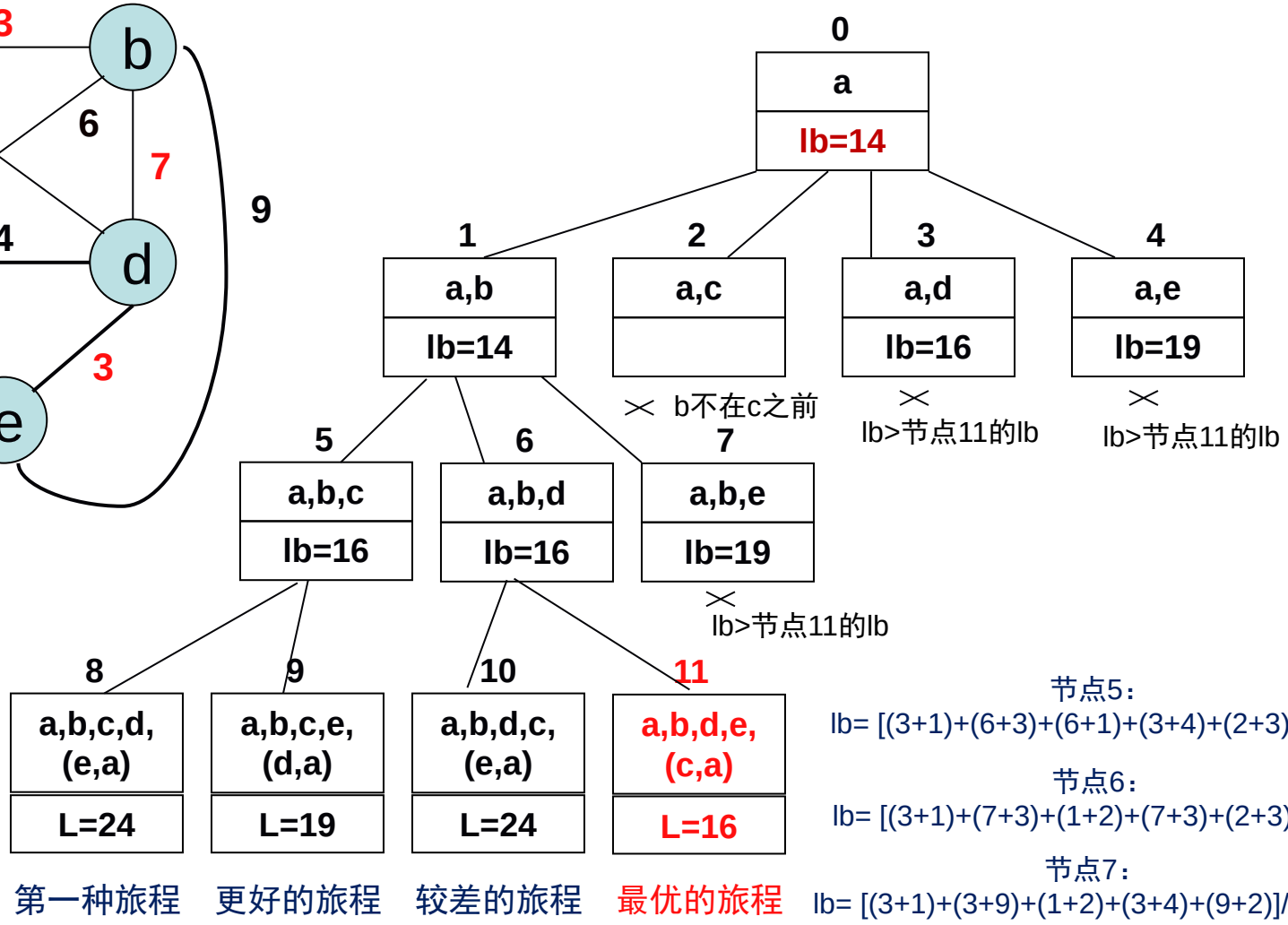
任何旅程的下界可以这样计算：

- 每一个城市i(1≤i≤n)到相邻最近的两个城市距离之和，并计算所有城市的总和，把结果除以2，并取整
- 如果包含某些特定边的旅程，可以在选取最短边前，首先包含特定边

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

旅行商问题





节点5：
lb= [(3+1)+(6+3)+(6+1)+(3+4)+(2+3)]/2=16

节点6：
lb= [(3+1)+(7+3)+(1+2)+(7+3)+(2+3)]/2=16

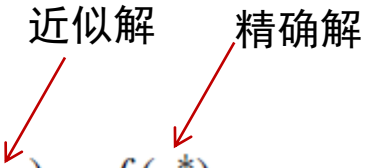
节点7：
lb= [(3+1)+(3+9)+(1+2)+(3+4)+(9+2)]/2=18.5

目录

- 回溯法
  - n-皇后问题
  - 哈密顿回路问题
  - 子集和问题
- 分支界限法
  - 分配问题
  - 背包问题
  - 旅行商问题
- NP困难问题的近似算法
- 基于人工智能的求解方法

近似算法

- 目标：有限时间内快速求得问题的近似解
- 近似解的评价标准
  - 相对误差、精确率



相对误差： $re(s_a) = \dfrac{f(s_a) - f(s^*)}{f(s^*)}$

精确率： $r(s_a) = \dfrac{f(s_a)}{f(s^*)}$ （最小化问题）、 $r(s_a) = \dfrac{f(s^*)}{f(s_a)}$ （最大化问题）

  - 但是，最优解未知的情况下，上述指标无法算出
  - 性能比（performance ratio）

定义 如果对于所讨论问题的任何实例，一个多项式时间的近似算法的近似解的精确率最多是 c，我们把它称作一个 c **近似算法**(approximation algorithm)，其中 $c \geqslant 1$。也就是说：

$$r(s_a) \leqslant c \qquad (12.3)$$

对于问题的所有实例，使不等式(12.3)成立的 c 的最佳(也就是最低)值，称为该算法的性能比(performance ratio)，记作 R_A。

背包问题的近似算法

- 离散背包问题的贪婪算法

第一步： 对于给定的物品，计算其价值重量比 $r_i = v_i/w_i$，$i = 1, 2, \cdots, n$。

第二步： 按照第一步计算出的比率的降序对物品排序(原先的顺序可以忽略)。

第三步： 重复下面的操作，直到有序列表中不留下物品。如果列表中的当前物品能够装入背包，将它放在背包中并处理下一物品；否则，直接处理下一个物品。

- 举例

物　品	重　量	价值/美元
1	7	42
2	3	12
3	4	40
4	5	25

背包承重量为10

物　品	重　量	价值/美元	价值/重量
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

背包问题的近似算法

- 连续背包问题的贪婪算法
  - “连续”：可按任意比例取走给定的物品装入背包

第一步： 对于给定的物品，计算其价值重量比 v_i/w_i，$i = 1, 2, \cdots, n$。

第二步： 按照第一步计算出的比率的降序对物品排序(原先的顺序可以忽略)。

第三步： 重复下面的操作，直到背包已经装满，或者有序列表中不留下物品。如果列表中的当前物品能够完全装入背包，将它放在背包中并处理下一个物品；否则，取出它能够装满背包的最大部分，然后停止。

物　品	重　量	价值/美元
1	7	42
2	3	12
3	4	40
4	5	25

物　品	重　量	价值/美元	价值/重量
1	4	40	10
2	7	42	6
3	5	25	5
4	3	12	4

6/7个物品

背包问题的近似算法

- 近似方案（Approximation Scheme）
  - 通过参数来调节的系列算法，比如，对于任何规模为n的实例来说

$$\frac{f(s^*)}{f(s_a^{(k)})} \leq 1 + 1/k \quad , \quad 其中\ 0 \leq k < n$$

  - 萨尼近似算法
    - 给定参数k, 则生成所有小于等于k个物品的子集，向每一个能够装入背包的子集添加剩余物品，并选择最有价值的子集输出

物　品	重　量	价值/美元	价值/重量
1	4	40	10
2	7	42	6
3	5	25	5
4	1	4	4

背包的承重量 $W = 10$

背包问题的近似算法

$k = 2$

子　集	添加的物品	价值/美元
∅	1, 3, 4	69
{1}	3, 4	69
{2}	4	46
{3}	1, 4	69
{4}	1, 3	69
{1, 2}	不可行	
{1, 3}	4	69
{1, 4}	3	69
{2, 3}	不可行	
{2, 4}		46
{3, 4}	1	69

旅行商问题的近似算法

- 是否可能找到一个具有有限性能比的多项式时间的近似算法，来对旅行商问题的所有实例求解？
- 定理

> **定理 1** 如果 $P \neq NP$，则旅行商问题不存在 c 近似算法。也就是说，该问题不存在多项式时间的近似算法使得所有的实例都满足
>
> $$f(s_a) \leqslant cf(s^*)，\ c \text{ 为常数}$$

  - 证明（反证法）
    - 假设存在这样的近似算法A和常数c，则可证明该算法在多项式时间内求解哈密顿回路问题
      - 假设G为n个顶点任意图，将G中每条边的权重设为1，不邻接的顶点间加上一条权重为cn+1边，则G就映射成为一个加权图G’。应用算法A到G’对应的旅行商问题，求解得到的最短路径长度与cn比，如果<=cn，则原图G存在哈密顿回路，否则，则不存在哈密顿回路
    - 但是，哈密顿回路问题是NP完全问题，除非P=NP，否则就导出矛盾

（1）贪婪算法：最近邻居算法

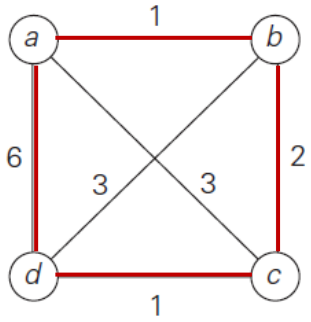
- 基本思想：下一次总是访问最近的未访问城市

第一步： 任意选择一个城市作为开始。

第二步： 重复下面的操作直到访问完所有的城市。访问和最近一次访问的城市 k 最接近的未访问城市(如果有距离相同的城市，可任意选择其一)。

第三步： 回到开始的城市。

- 举例
  - 算法输出：a–b–c–d–a
  - 最优路径：a–b–d–c–a
  - 精确率：$r(s_a)=\frac{f(s_a)}{f(s^*)}=\frac{4+w}{8}=\frac{10}{8}=1.25$（w为路径a–d的距离）
  - 当w足够大时，$r(s_a)$可以任意大，因此性能比$R_A=\infty$



（1）贪婪算法：多片段启发算法

- 基本思想：把问题抽象成求一个给定加权完全图的最小权重边集合，使得所有顶点的连通度都为2

第一步：将边按照权重的升序排列。将要构造的旅途边集合一开始是空集合。

第二步：重复下面的操作直到获得一条节点数为 n 的旅途，n 是待解问题中的城市数量：将排序列表中的下一条边加入旅途边集合，保证本次加入不会使得某个顶点的连通度为 3，也不会产生一条长度小于 n 的回路；否则，忽略这条边。

第三步：返回旅途边集合。

- 针对前述例子，多片段启发算法产生的路径与最近邻居算法相同
- 一般情况下，多片段启发算法优于最近邻居算法，但是它的性能也是没有上界的

（2）基于最小生成树的算法

- 基本思想：先构造加权图的一棵最小生成树，然后在这一良好基础上来构造一条近似最短路径

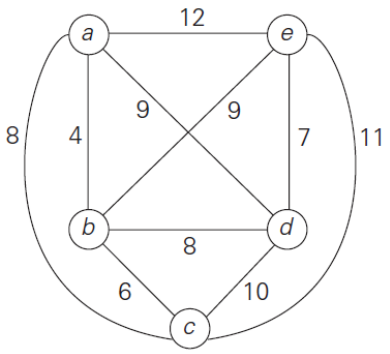
绕树两周算法(twice-around-the-tree algorithm)

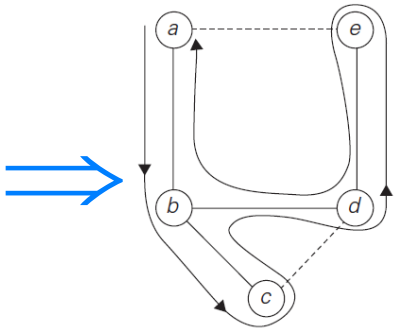
第一步：对一个旅行商问题的给定实例，构造它相应图的最小生成树。

第二步：从一个任意顶点开始，绕着这棵最小生成树散步一周，并记录下经过的顶点。(这可以用深度优先遍历来完成。)

第三步：扫描第二步中得到的顶点列表，从中消去所有重复出现的顶点，但留下出现在列表尾部的起始顶点不要消去。(这相当于在散步中走捷径。)列表中余下的顶点就构成了一条哈密顿回路，这就是该算法的输出。

- 举例



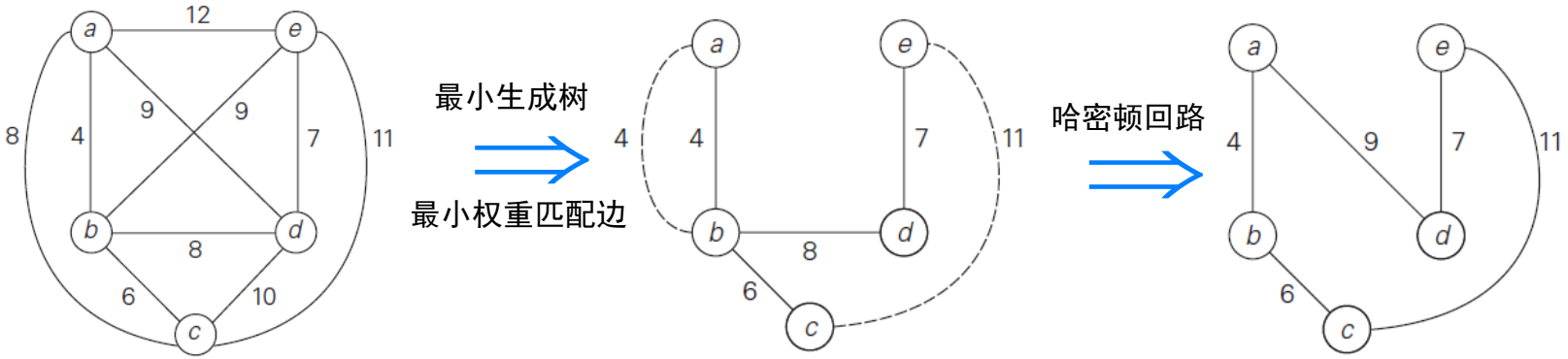


$a,\ b,\ c,\ \cancel{b},\ d,\ e,\ \cancel{d},\ \cancel{b},\ a.$

对于具有欧几里得距离的旅行商问题来说，绕树两周算法是一个 2 近似算法。$f(s_a) \leqslant 2f(s^*)$

（3）Christofides算法

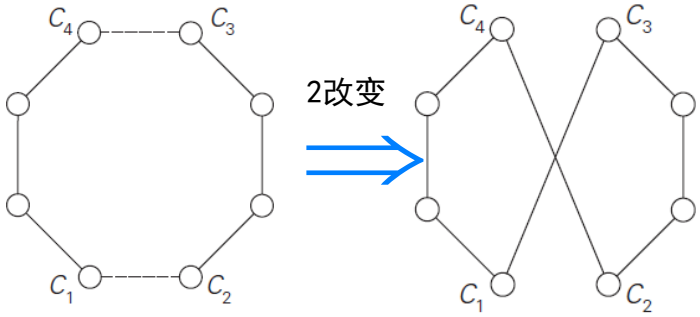
- 基本思想：先构造加权图的一棵最小生成树，然后找出树中所有连通度为奇数的顶点，并把这些顶点的最小权重匹配边加入图中，再求出该多重图的欧拉回路，最后通过走捷径的方法将其转换为哈密顿回路
- 举例

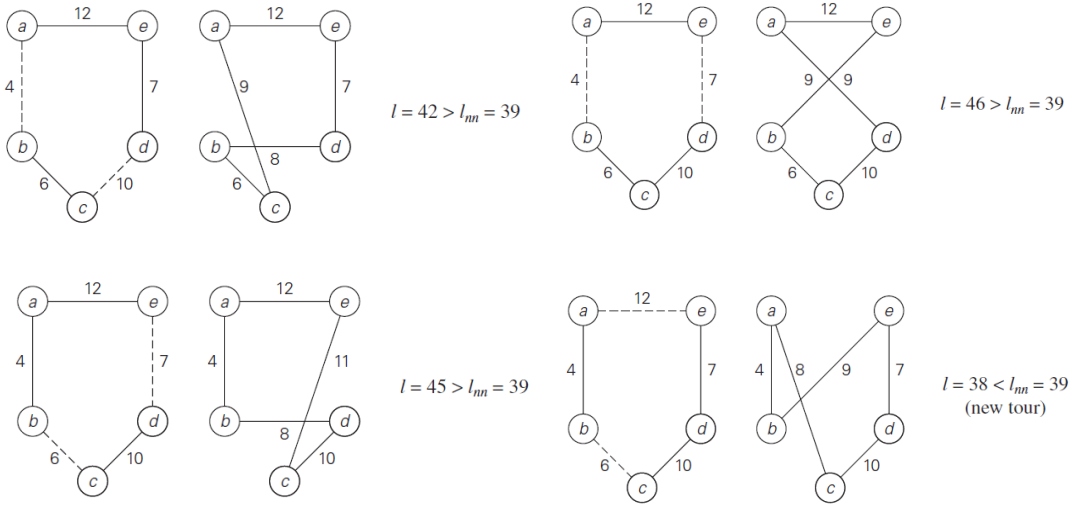


对于欧几里得实例，Christofides 算法的性能比是 1.5

(4) 本地查找启发法

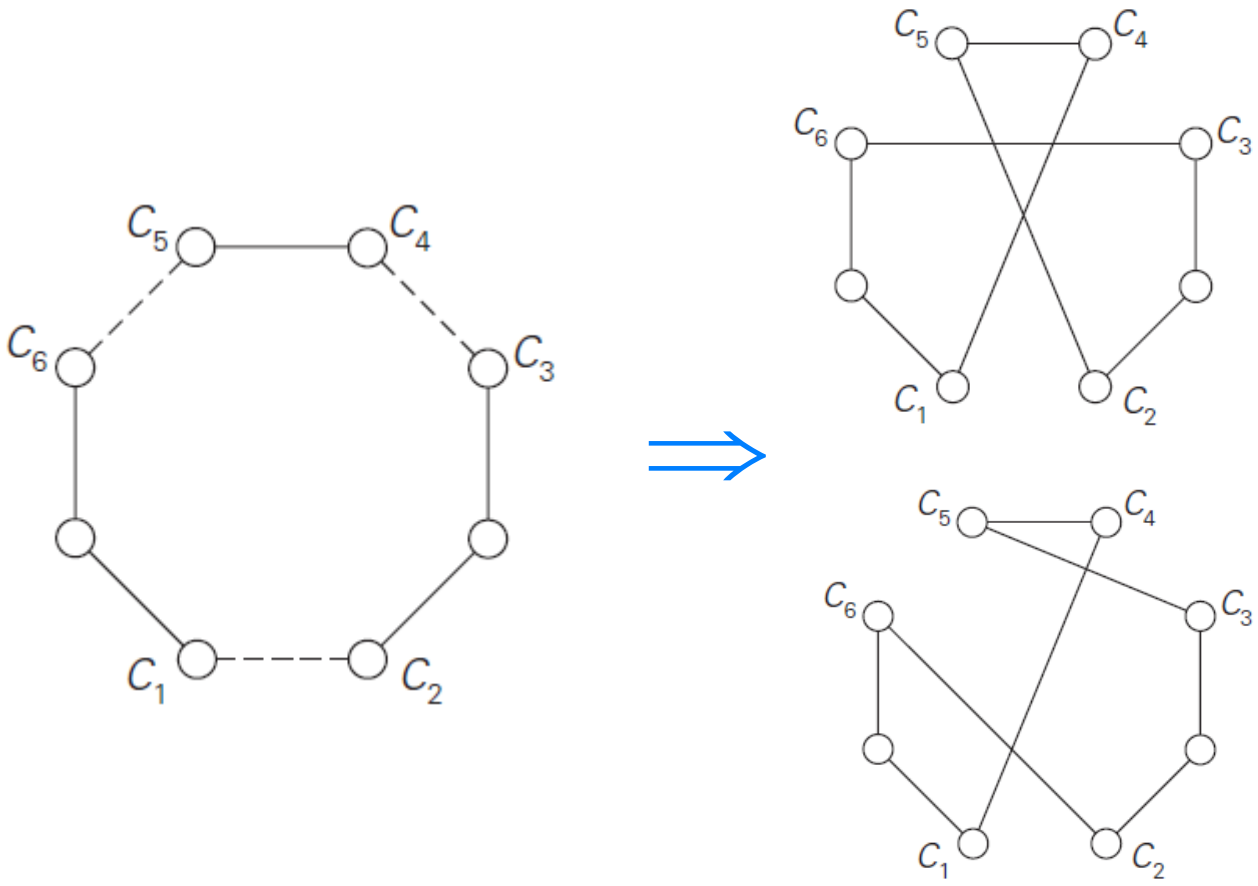
- 基本思想：基于某些简单的算法（如最近邻居）构造初始旅途，然后再进行迭代改进，每次迭代把当前旅途中的一些边用其它边来代替，试图生成更短的旅途
- 2改变算法：删除旅途中的一对非邻接边，然后把这两条边的端点用另一对边重新连起来，以得到另一个旅途





(4) 本地查找启发法

- 2改变的概念可进一步拓展到3改变



目录

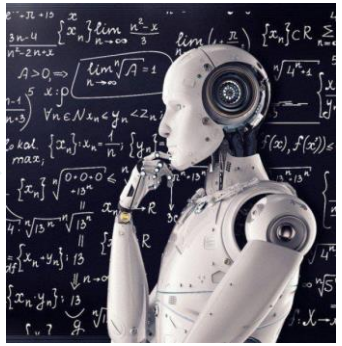
- 回溯法
  - n-皇后问题
  - 哈密顿回路问题
  - 子集和问题
- 分支界限法
  - 分配问题
  - 背包问题
  - 旅行商问题
- NP困难问题的近似算法
- 基于人工智能的求解方法

组合优化问题求解

数学优化算法
- 回溯法
- 分支界限法

启发式算法
- 近似算法
- 贪婪算法

元启发式算法
- 遗传算法
- 蚁群算法
- 模拟退火算法

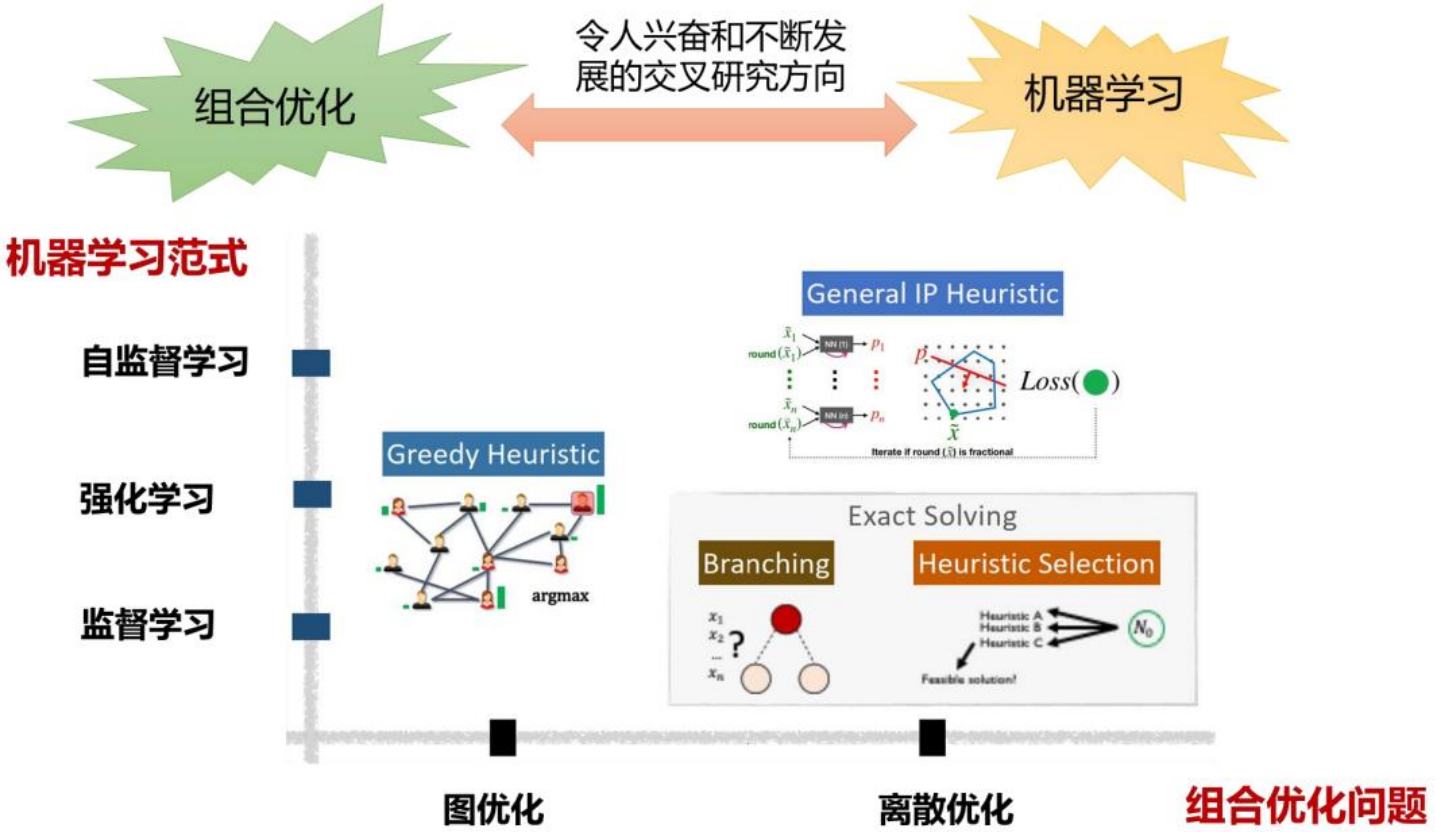


优：能找到全局最优解，有理论保证
劣：最差情况的时间呈指数增长

优：存在最差解的质量保证
劣：求解时间较长，好的近似算法不好构造

优：较短时间得到可行解
劣：算法对参数比较敏感

人工智能与组合优化



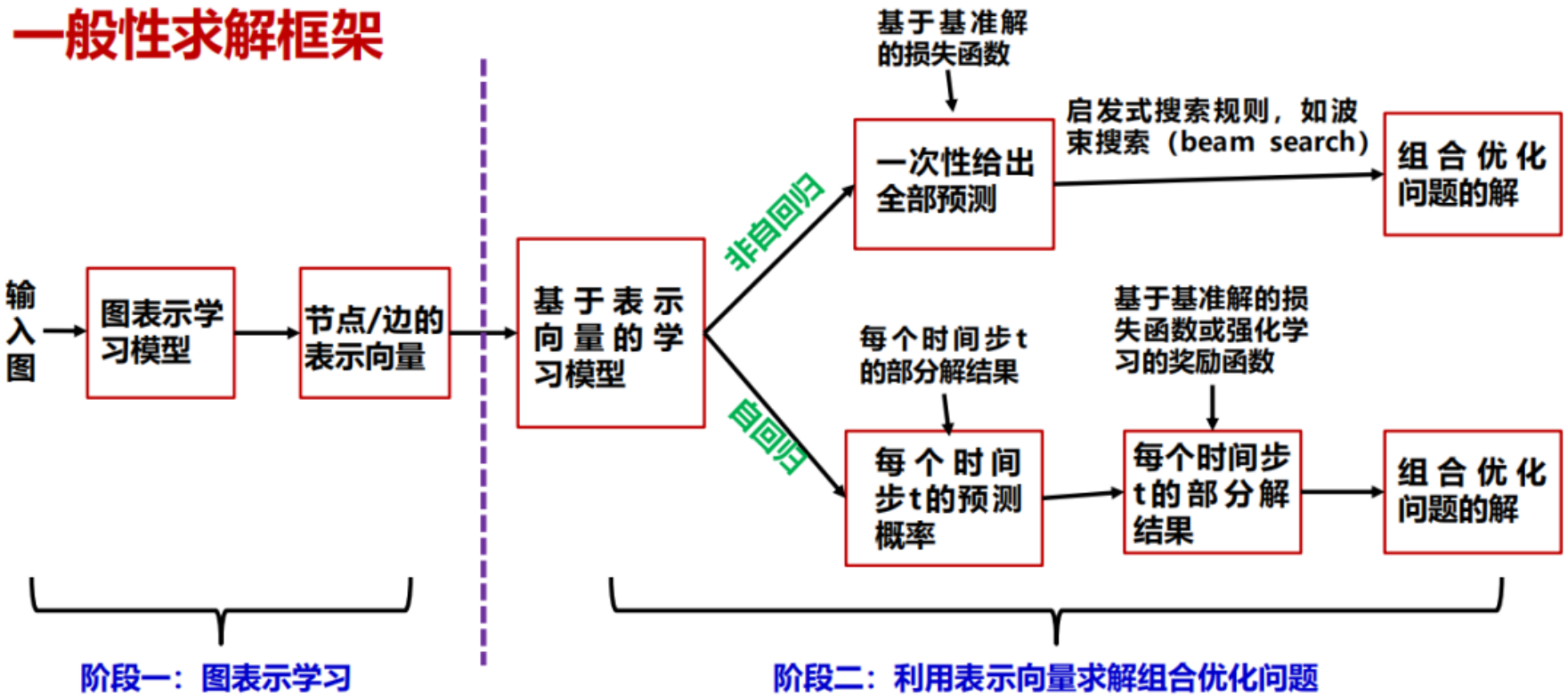
从同分布的数据中学习到通用的规律，能快速求解同一类问题

人工智能与组合优化

- 常见的求解范式
  - 基于端到端求解：输入是问题的实例，输出是问题的结果；使用自回归或者非自回归方法学习一些启发式的策略，得到输入-输出的映射关系
  - 局部改进求解：将改进精确算法或者启发式算法中比较耗时的模块转换成一个学习问题，从而加快求解的速度，如对于分支界限法， 可以将分支选择问题转成一个分类的学习问题

- 图上的组合优化问题
  - 第一阶段：进行图的表示学习，学习节点或边的表示向量
  - 第二阶段：基于表示向量和学习的方法求解优化问题

图上的组合优化问题求解



旅行商问题的智能化求解

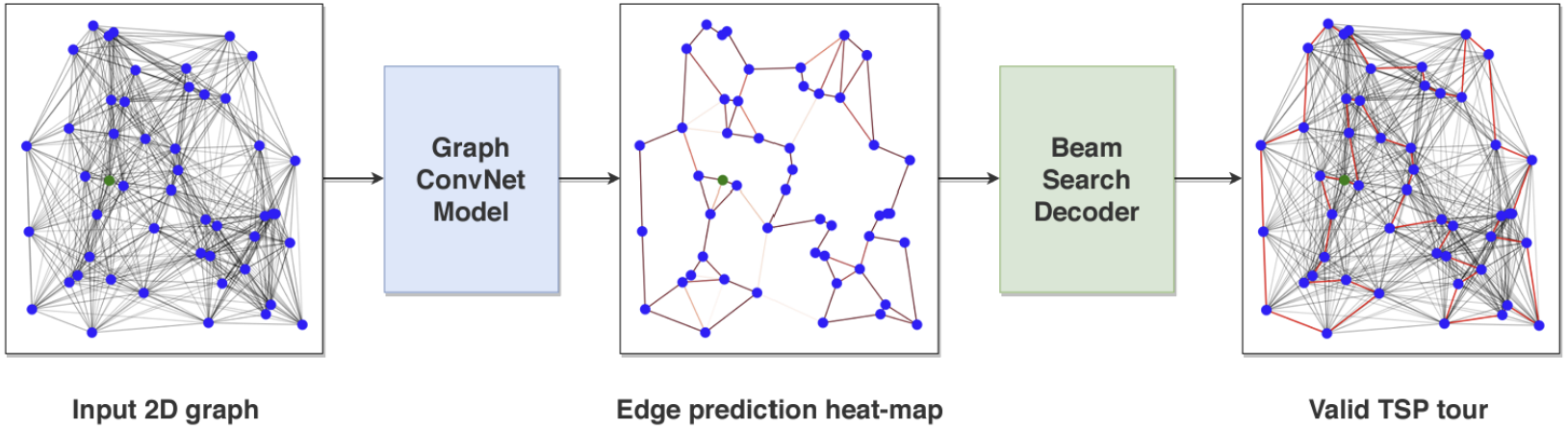
An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem

Chaitanya K. Joshi[1], **Thomas Laurent**[2], **and Xavier Bresson**[1]
[1]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[2]Department of Mathematics, Loyola Marymount University
{chaitanya.joshi, xbresson}@ntu.edu.sg, tlaurent@lmu.edu

[R1] https://arxiv.org/abs/1906.01227, An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem, 2019
[R2] https://arxiv.org/abs/1910.07210, On Learning Paradigms for the Travelling Salesman Problem, 2019
[R3] https://github.com/chaitjo/graph-convnet-tsp

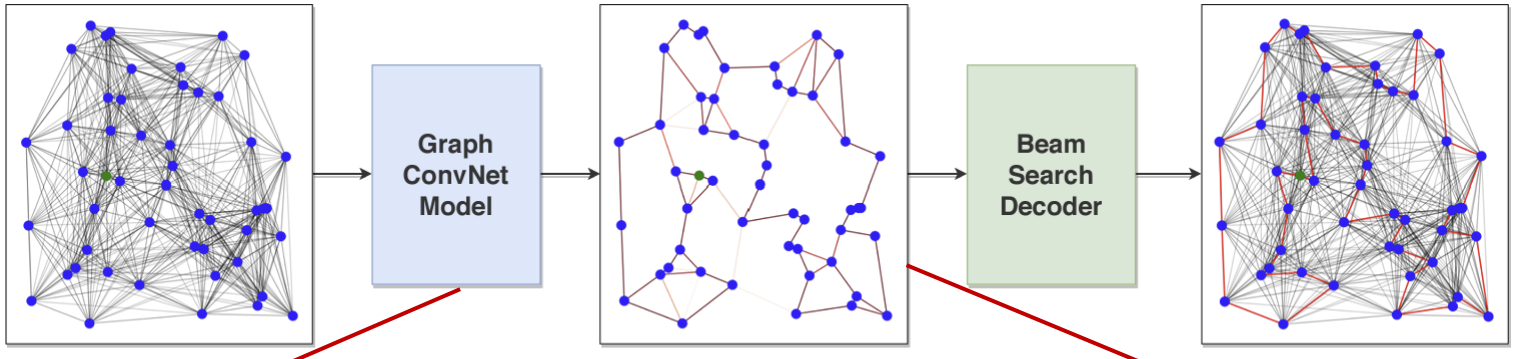
旅行商问题的智能化求解

- 使用深度图卷积网络来构建高效的TSP图表示，并通过高度并行化的波束搜索以非自回归的方式输出2D欧式图上的旅行商问题的解路径



- 输入图：n个城市组成的节点序列 $S=\{x_i\}_{i=1}^{n}$，其中 $x_i \in [0,1]^2$ 为节点（归一化）坐标

旅行商问题的智能化求解



输入层

将输入的节点特征，即二维坐标 $x_i \in [0,1]^2$，嵌入到 h 维特征向量：

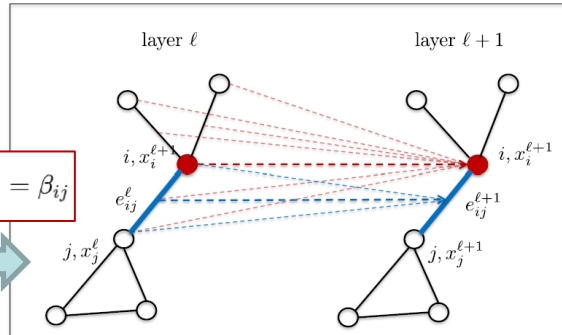
$$\alpha_i = A_1 x_i + b_1$$

将边的距离 d_{ij} 嵌入到 $\frac{h}{2}$ 维特征向量：

$$\beta_{ij} = A_2 d_{ij} + b_2 || A_3 \delta_{ij}^{k-NN}$$

$$x_i^{l=0} = \alpha_i, e_{ij}^{l=0} = \beta_{ij}$$

多层图卷积



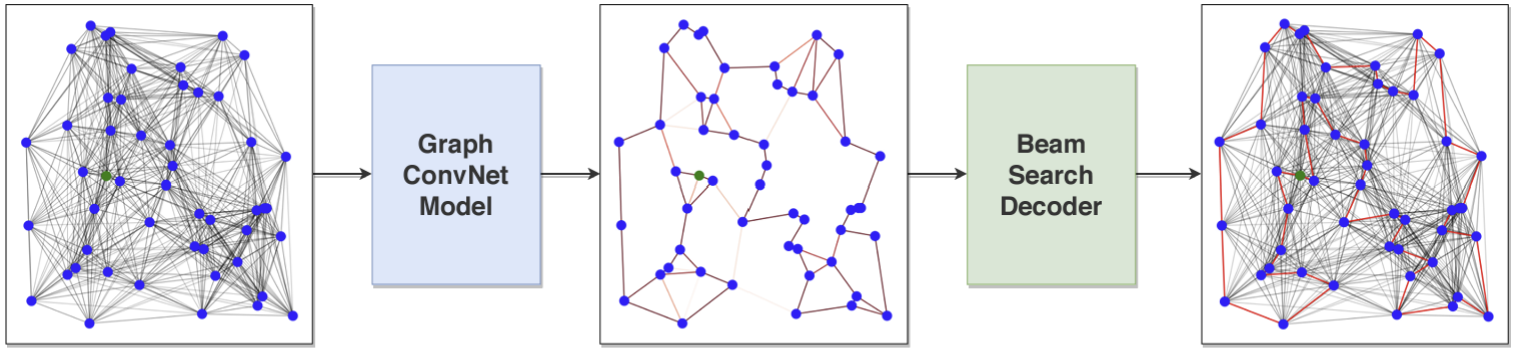
$$x_i^{\ell+1} = x_i^{\ell} + \mathrm{ReLU}\left(\mathrm{BN}\left(W_1^{\ell} x_i^{\ell} + \sum_{j \sim i} \eta_{ij}^{\ell} \odot W_2^{\ell} x_j^{\ell}\right)\right)$$

$$\eta_{ij}^{\ell} = \frac{\sigma(e_{ij}^{\ell})}{\sum_{j' \sim i} \sigma(e_{ij'}^{\ell}) + \varepsilon}$$

$$e_{ij}^{\ell+1} = e_{ij}^{\ell} + \mathrm{ReLU}\left(\mathrm{BN}\left(V_1^{\ell} e_{ij}^{\ell} + V_2^{\ell} x_i^{\ell} + V_3^{\ell} x_j^{\ell}\right)\right)$$

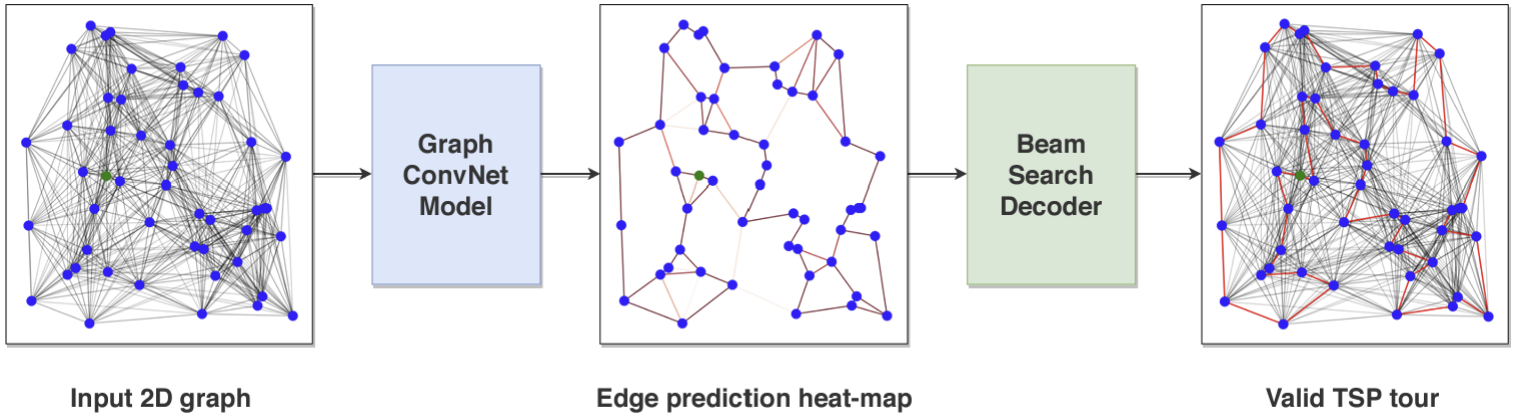
输出：每条边在该图TSP路径中的概率

旅行商问题的智能化求解



- 路径决策
  - 贪婪搜索：从第一个节点开始，贪婪地根据边出现的最高概率从它的邻居中选择下一个节点。当所有节点都被访问后，搜索终止。
  - 波束搜索：从第一个节点开始，通过扩展节点与邻居之间的 b 条最可能的边来探索热图（广度优先查找）。在每个阶段迭代地扩展top- b 部分路径，直到已经访问了图中的所有节点。最终的预测是在波束搜索结束时，选择 b 个完整路径中概率最高的路径（b 指的是波束宽度）。

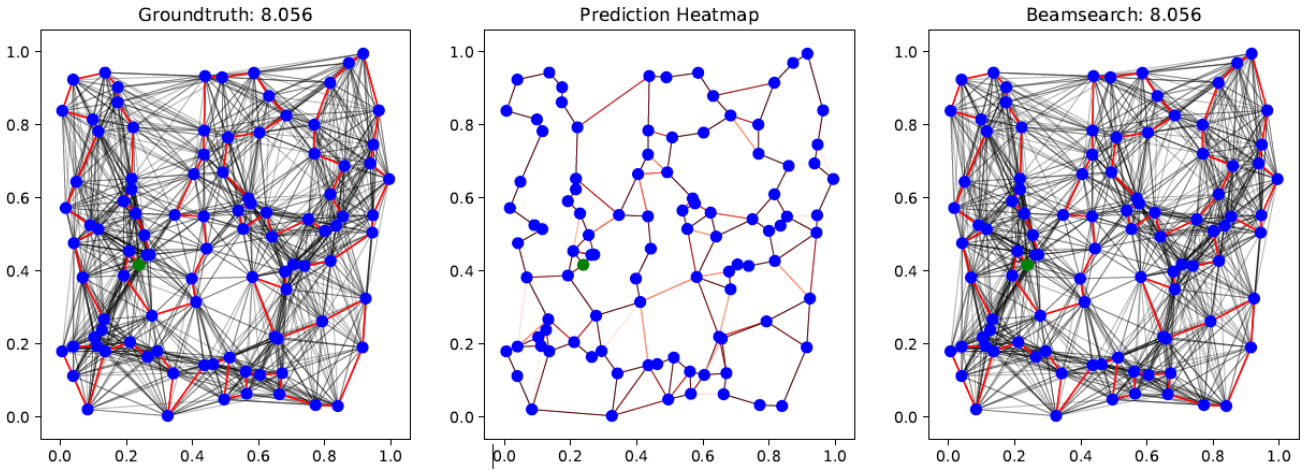
旅行商问题的智能化求解



- 训练方法
  - 数据集：大小为20、50和100个节点的图创建单独的训练、验证和测试数据集。训练集由1000000对问题实例和解组成，验证和测试集各由10000对组成。对于每个TSP实例，n 个节点位置在单位正方形中随机均匀采样。最佳路线 π 由Concorde求解器得到。
  - 损失函数（图卷积网络）：给定TSP路径的排列标签 π，将路径转换成邻接矩阵，其中每个元素的表示在TSP路径中的节点 i 和 j 之间存在或不存在边。最小化在小批量上平均的加权二元交叉熵损失。

旅行商问题的智能化求解

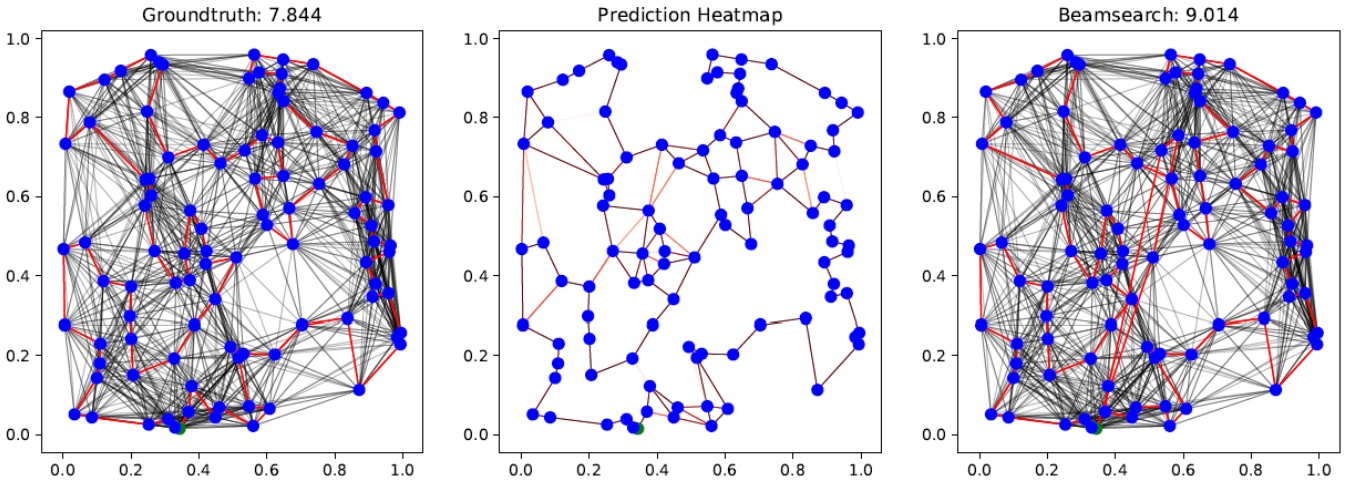
- 数值仿真：TSP100



(a) The complexity of this instance is due to the envelopes from the top-center into the center of the graph in the optimal tour. The model relies on beam search to navigate the complexity and predicts the optimal tour.

旅行商问题的智能化求解

- 数值仿真：TSP100



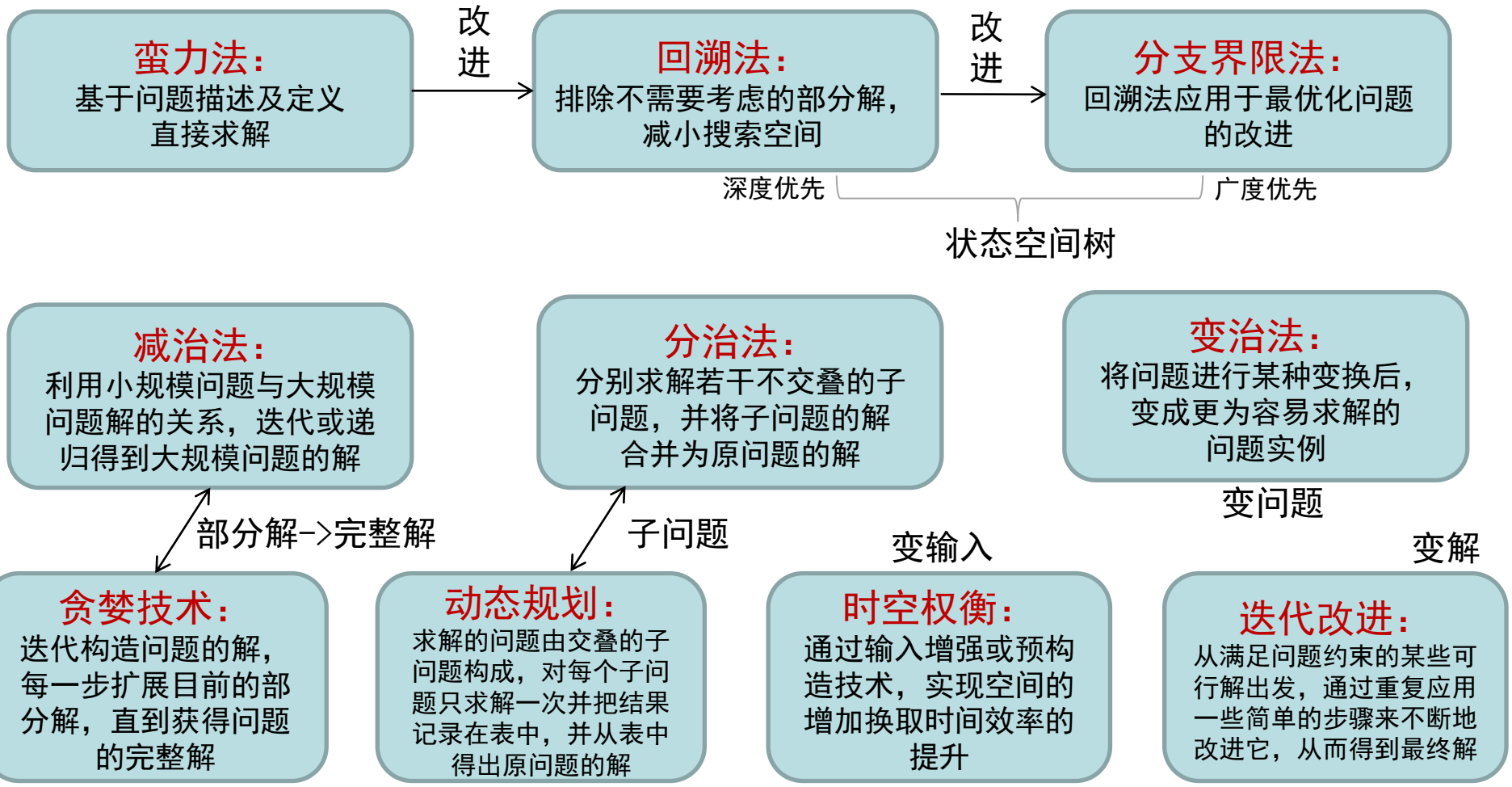
(b) This instance is harder than (a) because there several envelopes towards the center of the graph in the optimal tour. The model is unable to confidently identify these envelopes and beam search is unable to find the optimal contours around the center. This is reflected in extremely long connections being predicted for certain nodes.

旅行商问题的智能化求解

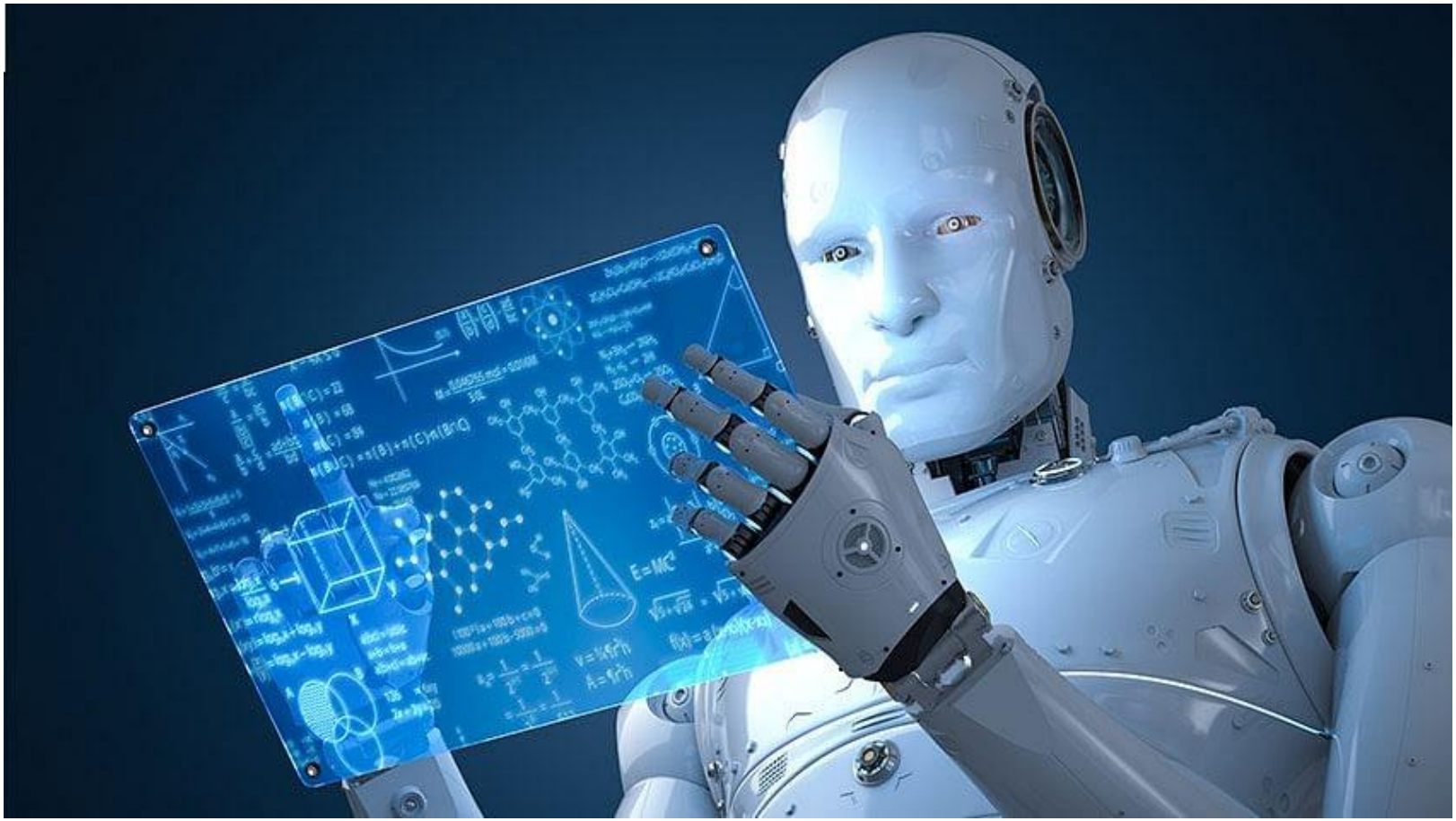
Table 1: Performance of our technique compared to non-learned baselines and state-of-the-art methods for various TSP instance sizes. Deep learning approaches are named according to the type of neural network used. The optimality gap is computed w.r.t Concorde. In the *Type* column, **H**: Heuristic, **SL**: Supervised Learning, **RL**: Reinforcement Learning, **S**: Sampling, **G**: Greedy, **BS**: Beam search, **BS***: Beam search and shortest tour heuristic, and **2OPT**: 2OPT local search.

Method	Type	TSP20			TSP50			TSP100		
		Tour Len.	Opt. Gap.	Time	Tour Len.	Opt. Gap.	Time	Tour Len.	Opt. Gap.	Time
Concorde	Solver	3.84	0.00%	(1m)	5.70	0.00%	(2m)	7.76	0.00%	(3m)
LKH3	Solver	3.84	0.00%	(18s)	5.70	0.00%	(5m)	7.76	0.00%	(21m)
Gurobi	Solver	3.84	0.00%	(7s)	5.70	0.00%	(2m)	7.76	0.00%	(17m)
Nearest Insertion	H, G	4.33	12.91%	(1s)	6.78	19.03%	(2s)	9.46	21.82%	(6s)
Random Insertion	H, G	4.00	4.36%	(0s)	6.13	7.65%	(1s)	8.52	9.69%	(3s)
Farthest Insertion	H, G	3.93	2.36%	(1s)	6.01	5.53%	(2s)	8.35	7.59%	(7s)
Nearest Neighbor	H, G	4.50	17.23%	(0s)	7.00	22.94%	(0s)	9.68	24.73%	(0s)
PtrNet [Vinyals et al., 2015]	SL, G	3.88	1.15%		7.66	34.48%			-	
PtrNet [Bello et al., 2016]	RL, G	3.89	1.42%		5.95	4.46%		8.30	6.90%	
S2V [Dai et al., 2017]	RL, G	3.89	1.42%		5.99	5.16%		8.31	7.03%	
GAT [Deudon et al., 2018]	RL, G	3.86	0.66%	(2m)	5.92	3.98%	(5m)	8.42	8.41%	(8m)
GAT [Deudon et al., 2018]	RL, G, 2OPT	3.85	0.42%	(4m)	5.85	2.77%	(26m)	8.17	5.21%	(3h)
GAT [Kool et al., 2019]	RL, G	3.85	0.34%	(0s)	5.80	1.76%	(2s)	8.12	4.53%	(6s)
GCN (Ours)	**SL, G**	**3.86**	**0.60%**	**(6s)**	**5.87**	**3.10%**	**(55s)**	**8.41**	**8.38%**	**(6m)**
OR Tools	H, S	3.85	0.37%		5.80	1.83%		7.99	2.90%	
Chr.f. + 2OPT	H, 2OPT	3.85	0.37%		5.79	1.65%			-	
GNN [Nowak et al., 2017]	SL, BS	3.93	2.46%			-			-	
PtrNet [Bello et al., 2016]	RL, S		-		5.75	0.95%		8.00	3.03%	
GAT [Deudon et al., 2018]	RL, S	3.84	0.11%	(5m)	5.77	1.28%	(17m)	8.75	12.70%	(56m)
GAT [Deudon et al., 2018]	RL, S, 2OPT	3.84	0.09%	(6m)	5.75	1.00%	(32m)	8.12	4.64%	(5h)
GAT [Kool et al., 2019]	RL, S	3.84	0.08%	(5m)	5.73	0.52%	(24m)	7.94	2.26%	(1h)
GCN (Ours)	**SL, BS**	**3.84**	**0.10%**	**(20s)**	**5.71**	**0.26%**	**(2m)**	**7.92**	**2.11%**	**(10m)**
GCN (Ours)	**SL, BS***	**3.84**	**0.01%**	**(12m)**	**5.70**	**0.01%**	**(18m)**	**7.87**	**1.39%**	**(40m)**

致敬经典技术


蛮力法：
基于问题描述及定义直接求解
改进
回溯法：
排除不需要考虑的部分解，减小搜索空间
改进
分支界限法：
回溯法应用于最优化问题的改进
深度优先
广度优先
状态空间树
减治法：
利用小规模问题与大规模问题解的关系，迭代或递归得到大规模问题的解
分治法：
分别求解若干不交叠的子问题，并将子问题的解合并为原问题的解
变治法：
将问题进行某种变换后，变成更为容易求解的问题实例
变问题
部分解->完整解
子问题
变输入
变解
贪婪技术：
迭代构造问题的解，每一步扩展目前的部分解，直到获得问题的完整解
动态规划：
求解的问题由交叠的子问题构成，对每个子问题只求解一次并把结果记录在表中，并从表中得出原问题的解
时空权衡：
通过输入增强或预构造技术，实现空间的增加换取时间效率的提升
迭代改进：
从满足问题约束的某些可行解出发，通过重复应用一些简单的步骤来不断地改进它，从而得到最终解

拥抱智能计算



课后作业

章 X	节 X. Y	课后作业题 Z	思考题 Z
12	12.1	2	4,11
	12.2	2,5,9	6
	12.3	7	8,9,10

注：只需上交“课后作业题”；以“学号姓名_chX.pdf ”规范命名，提交到“学在浙大”指定文件夹。DDL：2024年5月21日