第14讲 什么是有限时间内能求解
—问题的计算复杂性与分类

战德臣

哈尔滨工业大学计算学部教学委员会主任
国家教学名师

关于问题求解的疑问

【疑问】

什么是有限时间内能求解的问题?
什么是难解性问题?

算法复杂性

算法复杂性的概念

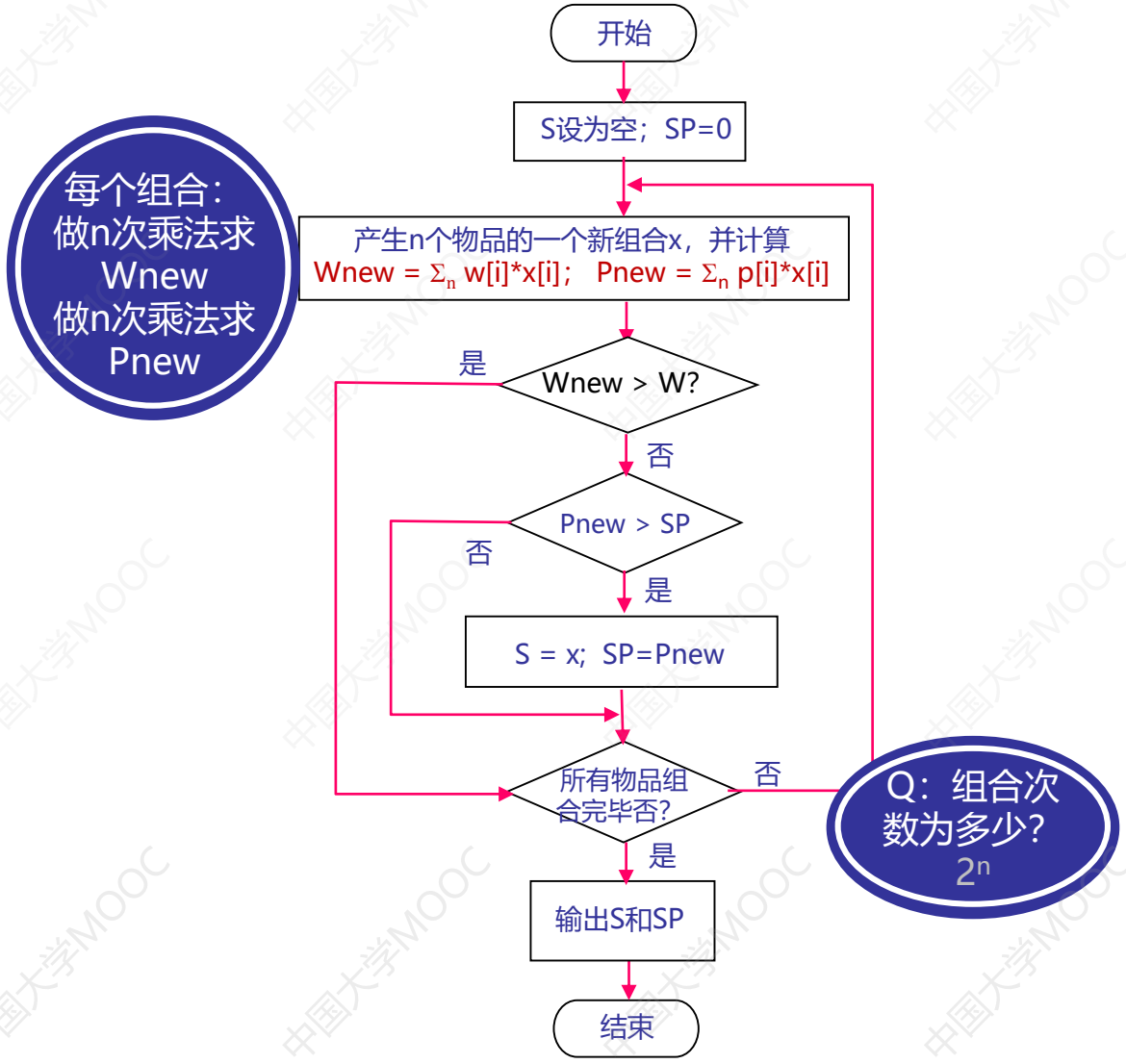
如何评价算法的性能?

算法复杂性，是指求解一个问题的**某一个算法的复杂性**

- **空间复杂性**：衡量算法运行中所需要的存储空间（参照时间复杂性来进一步理解）
- **时间复杂性**：衡量算法的运行速度。
  - ✓ 一个算法由一些基本步骤构成，则算法基本步骤的**执行次数**被称为**时间复杂性。**
  - ✓ 问题通常与自然数n相关联，以反映问题的规模。例如求解5种、50种、50000种物品的组合问题，就是不同规模的问题。
  - ✓ 问题求解算法通常与问题规模**n**相关（典型的就是循环、递归等）。
  - ✓ 算法基本步骤的执行次数通常被表达为关于n的一个函数T(n)，该T(n)被称为时间复杂性或时间复杂度。例如$T(n)=2^n$, 或者$T(n)=2n^2+3n+5$。
  - ✓ 时间复杂性是衡量算法的运行时间随着问题规模的增大而增加的趋势，而不是具体的运行时间。故而时间复杂性通常以量级来衡量，即大O表示法：O(T(n))。例如$O(2n^2+3n+5)=O(n^2)$，表示它们是同一量级的复杂性。

算法复杂性

算法复杂性计算示例



算法复杂性

$=(n+n+1+1+2+1)2^n+3$

$=(2n+5)\ 2^n+3$

$=O(n*2^n)$

算法复杂性

不同量级复杂性在计算时间方面的差异

问题规模n	计算量
10	10！
20	20！
100	100！
1000	1000！
10000	10000！

$20! = 1.216 \times 10^{17}$

$20^3 = 8000$

$O(n!)$与$O(n^3)$、$O(n^3)$与$O(3^n)$的差别

$O(n^3)$	$O(3^n)$
0.2秒	4×10^{28}秒 =1015年

注：每秒百万次，n=60，1015年相当于10亿台计算机计算一百万年

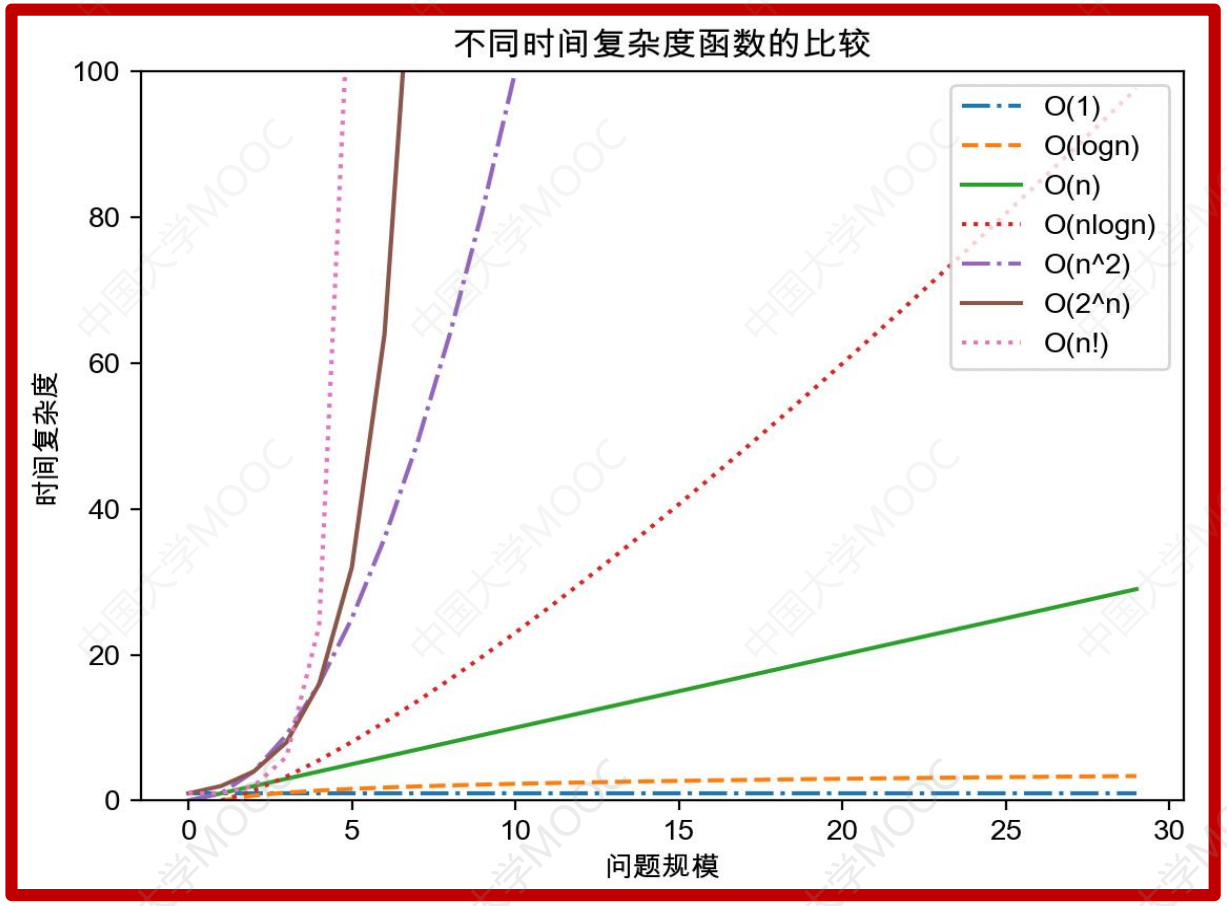
$O(1), O(\log n), O(n), O(n \log n), O(n^b)$

$O(b^n), O(n!)$

算法复杂性

不同量级复杂性在计算时间方面的差异

时间复杂度	问题规模		
	10	100	1000
O(1)	0.001 毫秒	0.001 毫秒	0.001 毫秒
O(logn)	0.003 毫秒	0.006 毫秒	0.0096 毫秒
O(n)	0.01 毫秒	0.1 毫秒	1 毫秒
O(nlogn)	0.03 毫秒	0.66 毫秒	9.97 毫秒
$O(n^2)$	0.1 毫秒	10 毫秒	1 秒
$O(2^n)$	1 毫秒	$4*10^{16}$ 年	$3.4*10^{287}$ 年
O(n!)	3.6 秒	$2.9*10^{144}$ 年	$1.3*10^{2555}$ 年



由算法复杂性到计算复杂性

计算复杂性

◆**算法复杂性**：求解问题的某一个**算法**的复杂性

◆**计算复杂性**：能求问题**精确解**的那个**最快**的**算法**的复杂性

计算复杂性是指问题的一种特性，即利用计算机求解问题的难易性或难易程度

问题的计算复杂性

●计算机在有限时间内能够求解的--【可求解问题】

●计算机在有限时间内难于求解的--【难求解问题】

●计算机完全不能求解的--【不可计算问题】

由算法复杂性到计算复杂性

有限时间内是否能完成算法的执行?

◆当算法的复杂度是一个**多项式**，如$O(n^2)$时，则对于大规模问题，该算法是可以在有限时间内被计算机完成的。例如，某一问题的贪心算法 $O(n^3)$ 。

$O(1), O(\log n), O(n), O(n \log n), O(n^b)$

◆当算法的复杂度是用**指数函数**表示如$O(2^n)$或**阶乘函数**表示如$O(n!)$，则当n很大（如10000）时，该算法是计算机在有限时间内无法处理的。例如某一问题的遍历算法O(n!)。

$O(b^n), O(n!)$

可求解与难求解问题

用计算复杂性划分问题

【P类问题】多项式问题(Polynomial Problem)，即：可以找出一个呈现$O(n^a)$复杂性算法**求出精确解**的问题，其中a为常数。P类问题是指计算机可以在有限时间内能求出精确解的问题，

【NP类问题】非确定性多项式问题(Non-deterministic Polynomial)，即：可以找出一个呈现$O(n^a)$复杂性算法来验证一个可能解是否是正确的问题。

✓有些问题，其答案无法直接计算得到，但可通过间接的猜算/试算来得到，这就是非确定性问题(Non-deterministic)。

✓虽然在多项式时间内难于求解，但给定一个解却不难在多项式时间内验证其正确性的问题，即是NP类问题。

【NPC类问题】完全非确定性多项式问题(NP-Complete)，即：(1)如果NP问题的所有可能解都可以在多项式时间内进行验算，(2)求精确解需要遍历所有可能解，(3)遍历所有可能解的复杂度是指数或阶乘级别的；则为NP-Complete问题。

●一个NP问题，并非其每个解都可以在多项式时间内验算，即：有些解是可以验算的，而有些解是不能验算的。

●如果一个NP问题其需要验算的解空间是用非多项式函数(指数函数或阶乘函数)来表达的，则是NP-Complete问题。

●如果一个NP问题经过约简后得到的问题仍旧是NP问题，则是NP-Complete问题。

问：加密算法应该设计成一个什么问题呢?

可求解与难求解问题

可解性问题与难解性问题

P类问题
(多项式时间内能求精确解)

NP类问题
(多项式时间内能验证一个解)

NPC类问题
(1)多项式时间内能验证每一个解,(2)求精确解需要遍历解空间,(3)指数时间遍历解空间

难解性问题
- 非NP问题
- 多项式时间内不一定能验证一个解

NP-Hard/NP难问题

NPC类问题求解思维

NPC类问题求解—求近似解、满意解

NPC类问题

穷举法或称**遍历法**：对解空间中的每一个可能解进行验证，直到所有的解都被验证是否正确，便能得到精确的结果---**精确解**

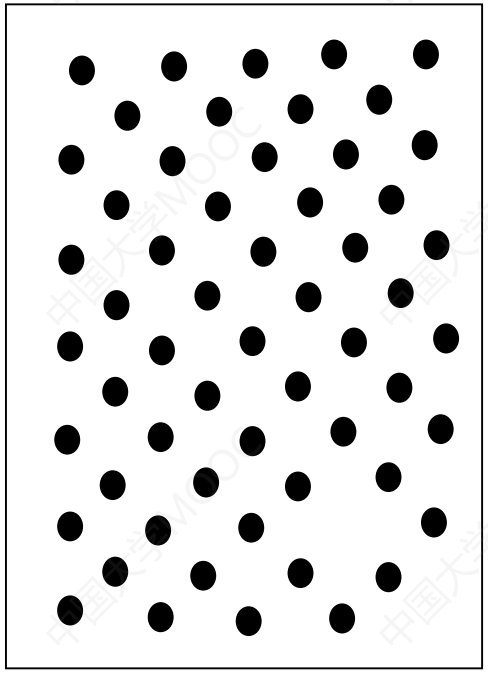
可能是O(n!)或O(a^n)

近似解求解算法---近似解

应该是O(n^a)

Δ = | 近似解 – 精确解 |
满意解： Δ充分小时的近似解

进化算法，**遗传算法**，蚁群算法，蜂群算法，…



穷举，遍历
搜索所有，可找到精确解

NPC类问题求解思维

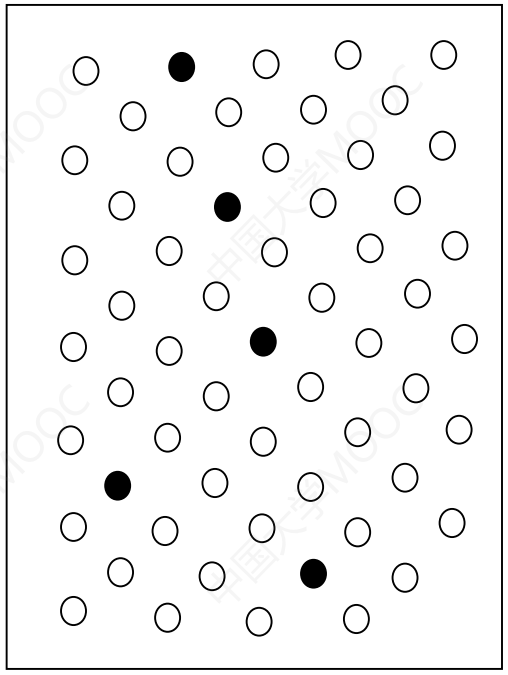
如何降低计算量—随机搜索

NPC类问题

近似解求解算法---近似解

随机搜索法：在解空间中随机选择一些可能解进行验证，求出所选择可能解(子解空间)中的最优解---近似解

- 基于概率论：随机选择
- 子解空间越大，求得满意解的可能性越大，但耗时也会加长
- 求出的近似解并不能保证是满意解



完全随机搜索
随机点之间完全没有联系

NPC类问题求解思维

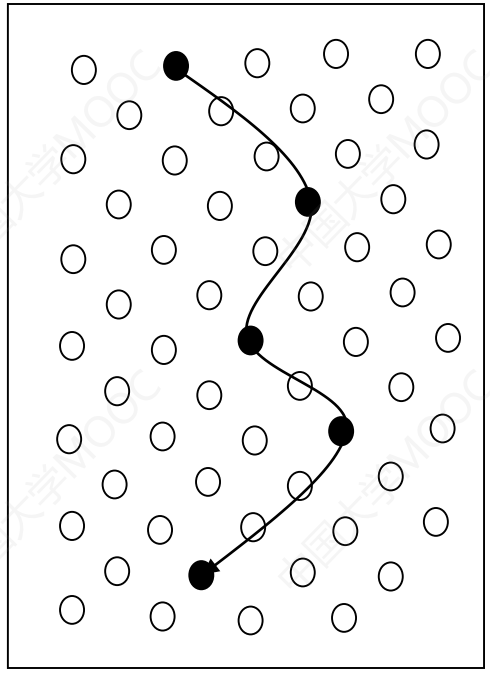
如何提高近似解的质量--导向性随机搜索

NPC类问题

近似解求解算法---近似解

导向性随机搜索法：对随机点的选取进行导向(导向到接近最优解的方向或路径)

- 基于概率论：随机选择
- 随机选择的可能解与前一可能解相比，更偏向于满意解
- 万一初始解就很差怎么办?



导向性随机搜索
随机点之间形成一路径

NPC类问题求解思维

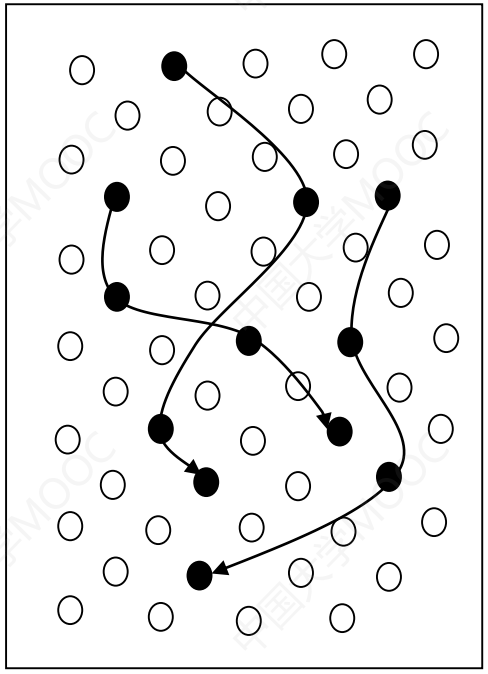
如何避免初识解的偏差--导向性群(体)随机搜索

NPC类问题

近似解求解算法---近似解

导向性群(体)随机搜索法：同时对多个随机点的选取进行导向(导向到接近最优解的方向或路径)，多条搜索路径

- 基于概率论：随机选择
- 多条路径下的最优，总比一条路径的最优要更优一些。
- 遗传算法就是这样一种导向性群随机搜索算法。
- 同一时刻多条路径上的解集合即为一个种群。多次选择，即多代进化。



导向性群(体)随机搜索
多随机点同步进行导向性搜索

问题的计算复杂性与分类

小结