优化问题

最优化问题

一般最优化问题的数学模型为

$$\min_{x \in D} f(x)$$

其中 x 为决策变量，D 为一个集合称为可行域，f 为 D 上的一个实值函数称为目标函数. 集合 D 中的任一元素称为问题的可行解，如果有一可行解 x^* 满足 $f(x^*) = \min\{f(x) \mid x \in D\}$，则称 x^* 为问题的最优解， 而 $f(x^*)$ 称为最优值. 在大多数情况下，可行域 D 是由一些称之为约束条件来确定的. 求解最优化问题就是寻找问题的最优解. 最优化问题也可以是极大化目标函数，此时若将 f 换作 $-f$， 那么极大化问题可转化为上述极小化问题.

组合优化问题

当最优化问题中的可行域 D 是一个由有限个元素组成的集合时，该最优化问题称为组合优化问题. 通常组合优化问题可表示为

$$\begin{cases} \min & f(x) \\ \text{s.t.} & g(x) \geq 0, \\ & x \in \{x_1, x_2, \cdots, x_n\}. \end{cases}$$

现实生活中大量问题是组合优化问题， 典型的组合优化问题有旅行商问题， 背包问题， 并行机排序问题等等.

旅行商问题 (traveling salesman problem, TSP)：

设有 n 个城市 $1, 2, \cdots, n$， 城市 i 与城市 j 间的距离为 d_{ij}. 一售货商要去这些城市推销货物， 他希望从一城市出发后走遍所有的城市且旅途中每个城市只经过一次， 最后回到起点. 选择一条路径使得售货商所走路线总长度最短， 这就是旅行商问题.

引进决策变量 x_{ij}，若商人从城市 i 出来后紧接着到城市 j， 则 $x_{ij}=1$，否则 $x_{ij}=0$ $(i,j=1,2,\cdots,n)$. 那么 TSP 的数学模型可表示为

$$
\begin{cases}
\min \quad \sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n} d_{ij}x_{ij} \\
\text{s.t.} \quad \sum\limits_{j=1}^{n} x_{ij}=1, \quad i=1,2,\cdots,n, \\
\qquad \sum\limits_{i=1}^{n} x_{ij}=1, \quad j=1,2,\cdots,n, \\
\qquad \sum\limits_{i,j\in S} x_{ij}\le |S|-1, \quad S \text{ 为 } \{1,2,\cdots,n\} \text{ 的非空真子集}, \\
\qquad x_{ij}\in\{0,1\}, \quad i,j=1,2,\cdots,n, i\neq j
\end{cases}
$$

其中 $|S|$ 表示集合 S 中元素的个数.

背包问题(knapsack problem):

设有一个容量为 b 的背包， n 个容积分别为 w_i， 价值分别为 c_i $(i = 1, 2, \cdots, n)$ 的物品， 选择那些物品放入背包中以使装入的物品总价值最大， 这就是背包问题.

引入决策变量 x_i， 若第 i 个物品被放入包中， 则 $x_i = 1$， 否则 $x_i = 0$ $(i = 1, 2, \cdots, n)$. 那么 背包问题的数学模型为

$$\begin{cases} \max & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i \leq b, \\ & x_i \in \{0, 1\}, \quad i = 1, 2, \cdots, n. \end{cases}$$

并行机排序问题(parallel machine scheduling):

设有 m 台同型机器 $M_1, M_2, \cdots, M_m$， n 个相互独立的工件 $J_1, J_2, \cdots, J_n$. 现在要安排这些工件到机器上进行加工， 设每个工件只需在任一台机器上加工， 工件 J_i 的加工时间为 t_i $(i = 1, 2, \cdots, n)$. 如何安排这些工件的加工方案， 以使机器完成所有工作的时间最少. 这就是并行机排序问题.

引入决策变量 x_{ij}， 若工件 J_i 在机器 M_j 上加工， 则 $x_{ij}=1$， 否则 $x_{ij}=0$. 那么并行机排序问题的数学模型为

$$
\begin{cases}
\min \quad T \\
\text{s.t.} \quad \sum_{j=1}^{m} x_{ij} = 1, \quad i = 1, 2, \cdots, n, \\
\qquad T \geq \sum_{i=1}^{n} t_i x_{ij}, \quad j = 1, 2, \cdots, m, \\
\qquad x_{ij} \in \{0, 1\}, \quad i = 1, 2, \cdots, n, j = 1, 2, \cdots, m.
\end{cases}
$$

由于组合优化问题中的可行域是有限集， 所以从理论上看可以将这有限个可行解枚举出来， 一一地计算出他们对应的目标值，然后通过比较大小找出最优解. 对于小规模的组合优化问题， 用这种方法很容易求出最优解. 但对于大规模或稍大规模的组合优化问题， 这种求解方法就不一定可行了.

计算复杂性

我们考虑这样一个问题： 有 n 种作物种子， 把它们分别播种到 n 块地里， 每种种子在不同的地块里的产量不同因而所得产值也就不同， 要求做一个播种方案， 以使总产值最大.

计算复杂性

我们考虑这样一个问题： 有 n 种作物种子， 把它们分别播种到 n 块地里， 每种种子在不同的地块里的产量不同因而所得产值也就不同， 要求做一个播种方案， 以使总产值最大.

所有可能的播种方案有 $n!$ 种， 若把列出一种方案作为一次基本操作， 则需 $n!$ 次基本操作. 用每秒执行一千万次操作的计算机来运算， 当 $n = 19$ 时，至少需要 385.7 年才能完成这些操作找出最优解. 当 $n = 20$ 时，至少需要 7714.6 年才能完成这些操作找出最优解. 这是不可实现的.

计算复杂性

我们考虑这样一个问题： 有 n 种作物种子， 把它们分别播种到 n 块地里， 每种种子在不同的地块里的产量不同因而所得产值也就不同， 要求做一个播种方案， 以使总产值最大.

所有可能的播种方案有 $n!$ 种， 若把列出一种方案作为一次基本操作， 则需 $n!$ 次基本操作. 用每秒执行一千万次操作的计算机来运算， 当 $n = 19$ 时，至少需要 385.7 年才能完成这些操作找出最优解. 当 $n = 20$ 时，至少需要 7714.6 年才能完成这些操作找出最优解. 这是不可实现的.

要了解问题的复杂性， 从而针对性地设计算法解决所研究的问题.

NP问题

- 实例：问题中的参数赋予了具体值的例子称为问题的实例
- 实例的规模：实例中所有参数数值的规模之和
- 数的规模：数在计算机中存储时占据的位数

- 例： 一个正整数 $x \in [2^s, 2^{s+1})$ 的二进制展开

$$x = a_s 2^s + a_{s-1} 2^{s-1} + \cdots + a_1 2 + a_0$$

$$(a_s = 1, a_i \in \{0, 1\}, i = 0, 1, \cdots, s-1)$$

所以 x 在计算机中占据 $s+1$ 位空间. 任何一个正整数 x 的输入规模为 （包含一个符号位和一个数据分隔位）

$$l(x) = \lceil \log_2(|x| + 1) \rceil + 2$$

其中 $\lceil x \rceil$ 表示不小于 x 的最小整数.

- 例： TSP 实例的规模

  该问题的任何一个实例由城市数 n 和城市间的距离 $D = \{d_{ij} \mid 1 \le i, j \le n, i \ne j\}$ 确定. 那么 TSP 的任何一个实例 I 的规模为

$$l(\mathrm{I}) = \lceil \log_2(n+1) \rceil + 2 + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \ne i}}^{n} \{ \lceil \log_2(|d_{ij}| + 1) \rceil + 2 \}$$

记问题的实例为 I, 实例规模为 $I(\mathrm{I})$, 算法 A 在求解 I 时的计算量（基本计算总次数） 为 $C_A(\mathrm{I})$.

- 若存在多项式函数 $g(x)$ 和一个常数 α, 使得

$$C_A(\mathrm{I}) \leq \alpha g(I(\mathrm{I}))$$

对给定问题的所有实例 I 成立, 记 $C_A(\mathrm{I}) = O(g(I(\mathrm{I})))$, 则称算法 A 为解决对应问题的多项式时间算法. 否则称算法 A 为指数时间算法.

- 如果一个问题的每一个实例只有“是” 或 “否” 两种答案， 则这个问题为判定问题. 称有肯定答案的实例为“是”实例， 否定答案的实例为“否”实例.

TSP 对应的判定问题

用 n 个城市的一个排列 $(i_1, i_2, \cdots, i_n)$ 表示商人从城市 i_1 出发依次通过 $i_2, \cdots, i_n$, 最后返回 i_1 这样一个路径. 判定问题为： 给定 z， 是否存在 n 个城市的一个排列 $W = (i_1, i_2, \cdots, i_n)$, 使得

$$f(W) = \sum_{j=1}^{n} d_{i_j i_{j+1}} \leq z?$$

其中 $i_{n+1} = i_1$. 满足 $f(W) \leq z$ 的一个排列 W 称为对应判定问题的一个可行解.

- P 类问题指具有多项式时间算法的问题类
- 若存在一个多项式函数 $g(x)$ 和一个验证算法 A, 对一类判定问题的任何一个“是”的判定实例 I 都存在一个字符串 S 是 I 的“是”回答, 其规模满足 $l(S) = O(g(l(\mathrm{I})))$, 且算法 A 验证 S 为实例 I 的“是”答案的计算时间为 $O(g(l(\mathrm{I})))$, 则称这个判定问题是非确定多项式的, 简记为 **NP**.

- 给定判定问题 Q_1 和 Q_2, 若存在一个多项式函数 $g(x)$ 和一个算法 T 满足:

  (1) T 将 Q_1 问题的一个实例 I_1, 计算得到 Q_2 问题的一个实例 I_2, 且 I_2 为 Q_2 问题的一个"是"实例的充分必要条件是 I_1 为 Q_1 问题的一个 "是"实例;

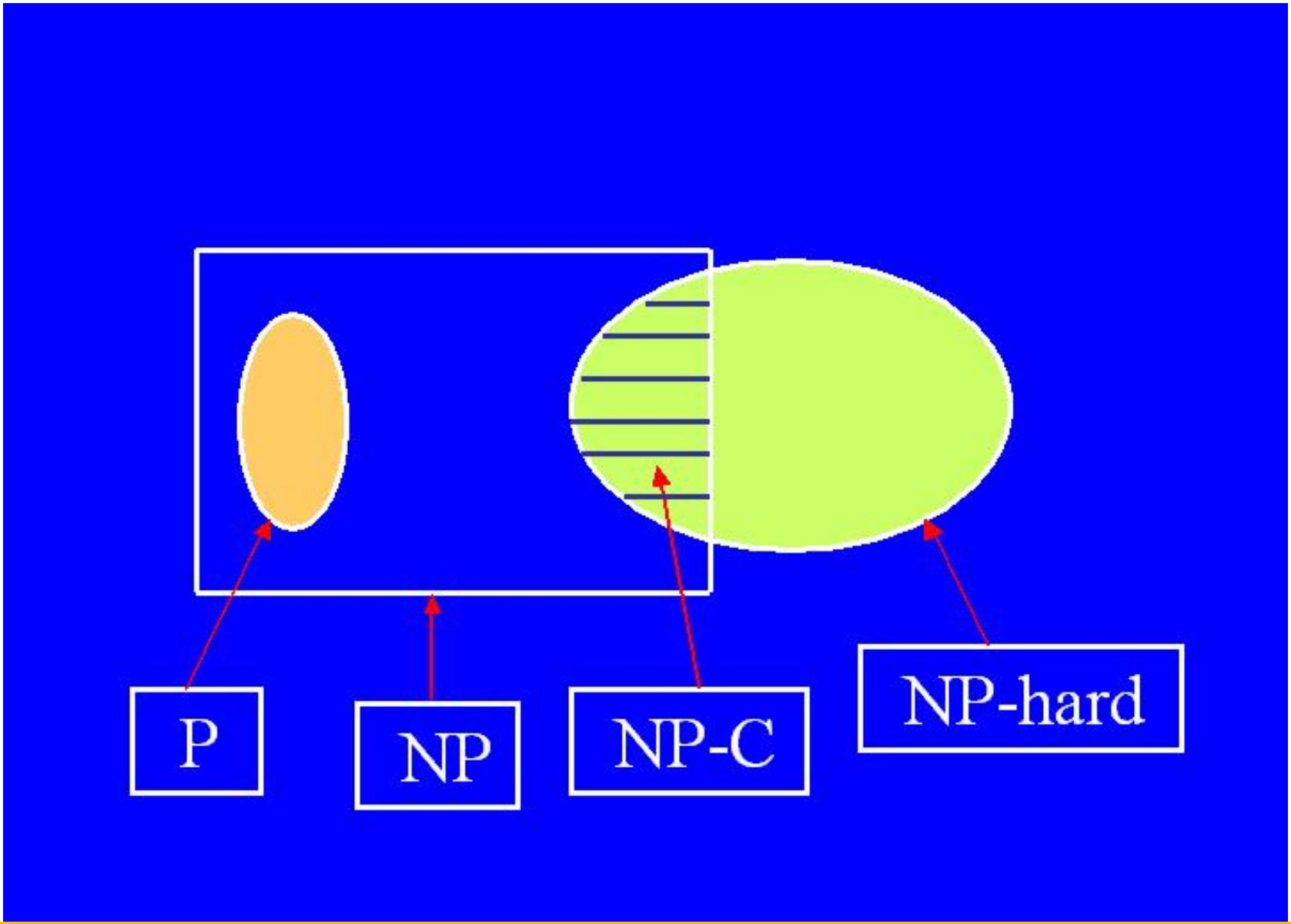
  (2) T 是一个多项式时间算法, 即从实例 I_1 求解得到 I_2 的规模为 $O(g(l(\mathrm{I}_1)))$,

  则称 Q_1 问题多项式转换为 Q_2 问题.

- 当满足下列条件时， 称判定问题 Q_1 多项式归约为 Q_2： 存在 Q_1 的算法和多项式函数 $g(x)$, 算法求解 Q_1 任何实例 I 的过程中， 将 Q_2 的算法作为子程序多次调用. 如果将一次调用 Q_2 算法看成一个单位 （1次基本计算量）， 则这个 Q_1 算法的计算量为 $O(g(I(\mathrm{I})))$.

- 如果判定问题 $Q \in$NP 且 NP 中的任何一个问题都可在多项时间内 归约为 Q, 则称 Q 为 NP完全 （简记为 NP-C）.
- 若 NP 中的任何一个问题都可在多项式时间归约为判定问题 Q， 则称 Q 为 NP难 （简记为 NP-hard）.

四类问题的关系图



邻域

在组合优化中，在一点附件搜索另一个下降的点是组合最优化数值求解 的基本思想.

- 设 D 是一个组合优化问题中决策变量的定义域， 则

$$N : x \in D \to 2^D, \quad x \in N(x)$$

称为一个邻域映射. $N(x)$ 称为 x 的邻域, $y \in N(x)$ 称为 x 的一个邻居.

- 对 TSP 模型， $D = \{x \mid x \in \{0,1\}^{n(n-1)}\}$, 可以定义它的一种邻域为

$$N(x) = \left\{ y \mid |y - x| = \sum_{i,j} |y_{ij} - x_{ij}| \le k, y \in D \right\},$$

k 为一个正整数.

- TSP 问题的另一种表示法为
$D = \{S = (i_1, i_2, \cdots, i_n) \mid i_1, i_2, \cdots, i_n$ **是** $1, 2, \cdots, n$ **的排列** $\}$.
定义它的邻域映射为 2-opt, 即 S 中的两个元素进行对换. 如当 $S = (1,2,3,4)$ 时,
$N(S) = \{(1,2,3,4), (2,1,3,4), (3,2,1,4), (4,2,3,1), (1,3,2,4), (1,4,3,2), (1,2,4,3)\}$.

- 若 $x^* \in F$ 满足

$$f(x^*) \leq f(x), \quad x \in N(x^*) \cap F$$

则称 x^* 为 f 在 F 上的局部最小解（点）. 若

$$f(x^*) \leq f(x), \quad x \in F$$

则称 x^* 为 f 在 F 上的全局最小解（点）.

启发式算法

启发式算法是一种技术， 这种技术使得在可接受的计算费用内去寻找最好的解， 但不一定能保证所得解的可行性和最优性， 甚至在多数情况下， 无法阐述所得解同最优解的近视程度.

如果采用最坏实例下的误差界限来评价启发式算法， 则可以定义近似算法

- 设 Q 是一个问题， 记问题 Q 的任何一个实例 I 的最优解和启发式算法 H 得到的解分别为 $z_{opt}(\mathrm{I})$ 和 $z_H(\mathrm{I})$. 于是对某个 $\varepsilon \geq 0$, 称 H 是 Q 的 ε 近似算法当且仅当

$$|z_H(\mathrm{I}) - z_{opt}(\mathrm{I})| \leq \varepsilon |z_{opt}(\mathrm{I})|, \quad \forall \mathrm{I} \in Q.$$

背包问题的贪婪算法

Step 1. 对物品以 $\frac{c_i}{w_i}$ 从大到小排列， 不妨记为 $\{1, 2, \cdots, n\}$, $k := 1$;

Step 2. 若 $\sum_{i=1}^{k=1} w_i x_i + w_k \leq b$, 则 $x_k = 1$; 否则 $x_k = 0$. $k := k + 1$;

当 $k = n + 1$ 时， 停止； 否则， 重复 step 2.

贪婪算法的计算量为 $O(n \log_2 n)$, 是一个多项式时间算法.

简单的邻域搜索算法

Step 1. 任选一个初始解 $s_0 \in D$.

Step 2. 在 $N(s_0)$ 中按一定规则选择一个 s; 若 $f(s) < f(s_0)$, 则 $s_0 := s$. 否则, $N(s_0) := N(s_0) - \{s\}$; 若 $N(s_0) - \{s_0\} = \emptyset$, 停止; 否则, 重复 step 2.

启发式算法分类

1. 简单直观的算法
   (1) 一步算法. 如背包问题的贪婪算法
   (2) 改进算法. 如邻域搜索算法
2. 数学规划算法. 如分支定界法， 割平面法
3. 智能算法

启发式算法的性能分析

- 最坏情形分析

  对所有实例 I, 评价关系式为

$$\alpha z_{OPT}(\mathrm{I}) + d \geq z_H(\mathrm{I}) \geq z_{OPT}(\mathrm{I}),$$

  其中， $|d| < z_{OPT}(\mathrm{I})$, d 是一个与 I 无关的偏差值， $\alpha \geq 1$.
  α 愈接近 1 说明构造的启发式算法愈好.

  与其相近的方法是确定渐近最坏率

$$AR(H) = \lim_{k\to\infty} \sup_{\mathrm{I}} \left\{ \frac{z_H(\mathrm{I})}{z_{OPT}(\mathrm{I})} \middle| z_{OPT}(\mathrm{I}) = k \right\}$$

- 例： 0-1 背包问题的贪婪算法记为 G 算法， 构造一个实例 I 如下： $c_1 = 1 + \varepsilon$, $w_1 = 1$, $c_2 = K$, $w_2 = K$, $b = K$, $\varepsilon > 0$. 用贪婪算法得到 $x_1 = 1$, $x_2 = 0$, 则 $z_G(\mathrm{I}) = 1 + \varepsilon$, 而 $z_{OPT}(\mathrm{I}) = K$. 故

$$\frac{z_G(\mathrm{I})}{z_{OPT}(\mathrm{I})} = \frac{1+\varepsilon}{K} \to 0, \quad K \to +\infty$$

说明0-1 背包问题的贪婪算法在最坏情况分析下非常不好.

- 概率分析
- 大规模计算分析