第 5 章 背包问题

在有限资源条件下如何做出最优选择，是众多实际问题的核心所在。从选择最合适的投资组合、装载货物的最优配置，到分配预算、制定行程安排，我们常常面临“在约束条件下最大化收益”的决策任务。背包问题（Knapsack Problem）的基本形式是：在给定容量限制的背包中，如何选择若干物品，使得所装物品的总价值最大且不超过容量限制。这一问题虽然表述简洁，却蕴含着丰富的算法思想与建模技巧。

作为组合优化领域的基础问题之一，背包问题不仅在理论计算机科学中具有重要地位，更广泛应用于金融决策、物流规划、资源配置等现实场景。它也是众多复杂优化问题的原型，构成动态规划、贪心算法、分支限界等方法的教学范例。通过对背包问题的深入学习，读者将能够掌握基本建模思维，理解多种算法的适用条件与局限性，并为后续更复杂的优化问题奠定理论与实践基础。

5.1 背包问题的介绍

背包问题是一个组合优化问题：给定一组物品，每件物品都有一个重量和一个价值，确定每件物品在集合中的数量，使总重量（权重）在小于或等于给定限制的前提下，总价值尽可能大。它的名称来源于这样一个问题：一个人受到一个固定大小的背包的限制，必须尽可能把有价值的物品装进背包。这个问题经常出现在资源分配中，决策者必须在固定的预算或时间限制下，分别从一组不可分割的项目或任务中做出选择。

5.2 基本的背包问题及数学模型

背包问题是组合优化中的一个经典问题，广泛应用于资源分配、物资装载、投资组合等场景。其中最基本的形式是 0-1 背包问题。设有一个容量为a千克的背包，以及n种不同的物品，每种物品的重量为ω_i千克，价值为数量x_i的函数$c_i(x_i)$。每种物品最多只能选择一件，是否选择第i种物品用变量x_i表示，其中$x_i \in \{0,1\}$，选择该物品则x_i为 1。问题的目标是在不超过背包容量的前提下，选择物品使得总价值最大。其数学模型为：

$$\max f = \sum_{i=1}^{n} c_i(x_i)$$

$$\sum_{i=1}^{n} \omega_i x_i \leq a$$

$$x_i \in \{0,1\}$$

相比之下，一般化的背包问题允许每种物品装入多件，且每种物品的价值可以随着装入数量变化。一般化的背包问题描述如下：

有一个登山者带一个背包上山,其可携带物品重量的限度为a千克。设有n种物品可供他选择装入背包中,这n种物品编号为$1,2,\ldots n$。已知第i种物品每件重量为ω_i千克，在上山过程中的作用(价值)是携带数量x_i的函数$c_i(x_i)$。问登山者应如何选择携带物品(各几件),使所起作用(总价值)最大。此时的数学模型为：

$$\max f = \sum_{i=1}^{n} c_i(x_i)$$

$$\sum_{i=1}^{n} \omega_i x_i \leq a$$

$x_i \geq 0$ 且为整数，$\mathrm{i} = 1$，2，$\cdots, \mathrm{n}$.

5.3 基本背包问题的求解算法

5.3.1 贪婪启发式算法

贪婪算法也可以用于解决这个问题，但并不能保证找到最优解，尤其是在物品不能拆分的情况下（即 0-1 背包问题）。贪婪算法更适用于完全背包问题，即允许部分装入物品的情况。

首先，计算每个物品的单位重量的价值，即物品的价值与物品重量之比。价值密度是衡量物品性价比的标准，高价值密度的物品意味着相对较高的收益与较低的重量，这对于背包的装载策略至关重要。具体计算公式为：价值密度=物品的价值/物品的重量。

接下来，将所有物品按照价值密度从高到低进行排序。排序的目的是优先选择那些单位重量价值较高的物品，这样可以在有限的背包容量内尽可能多地获取高收益。

然后，从排序后的物品列表中，依次选择物品放入背包中。在选择过程中，对于每个物品，首先检查其重量是否能够放入当前的背包。如果能放入，就将物品放入背包，并更新背包的剩余容量；如果不能放入，则跳过该物品，继续考虑下一个物品。

当所有物品都被遍历一遍，或者背包容量已经达到最大值时，算法停止，返回最终选择的物品和背包的总价值。

贪婪算法的核心在于每次都选择当前最优的物品进行装载，虽然这种策略可能并不总是能够得出全局最优解，但它通常能在较短时间内找到一个较为接近最优的解，且计算效率较高，适用于解决许多实际问题。

例题 5.1 某公司准备组织一次外出旅行，每个人可以携带一个背包，背包的最大承载重量是 15 千克。公司有五种物品可以选择，每种物品都有特定的重量和价值，如表所示：

表 5.1 例 5.1 中每种物品的重量和价值

物品	重量（kg）	价值（元）
物品 1	2	30
物品 2	5	60
物品 3	4	50
物品 4	8	88
物品 5	6	60

根据贪婪算法的思想，首先计算每个物品的价值密度，即每千克的价值。物品 1 的价值密度为 15 元/千克，物品 2 的价值密度为 12 元/千克，物品 3 的价值密度为 12.5 元/千克，物品 4 的价值密度为 11 元/千克，物品 5 的价值密度为 10 元/千克。通过比较得出，物品的顺序是：物品 1、物品 3、物品 2、物品 4、物品 5。接下来，算法开始尝试以这个顺序将物品放入背包。

首先考虑物品 1，它的重量为 2 千克，可以放入背包，放入后背包的剩余容量为 13 千克。接着，考虑物品 3，它的重量为 4 千克，可以放入背包，放入后背包的剩余容量为 9 千克。然后考虑物品 2，它的重量为 5 千克，可以放入背包，此时背包剩余容量为 4 千克。此时，背包已经无法再放入物品 4 和物品 5。

因此，通过贪婪算法，最终选择了物品 1、物品 2 和物品 3，总价值为 140 元。

贪婪算法的关键在于每次选择当前价值密度最高的物品，但由于这种策略并不保证得到全局最优解，可能会错过其他更好的组合。虽然这个解在计算上比较高效，但不一定是最优解。在本例中，虽然通过贪婪算法得到的解比较好，但它并不保证是最优解。例如，如果选择物品 1、物品 2 和物品 4，它们的总价值为 178 元，总重量为 15 千克，价值高，且背包刚好装满。因此，在某些情况下，贪婪算法可能无法找到最优解，但它通常能在短时间内得到一个接近最优的解。

5.3.2 动态规划

下面介绍用动态规划的方法来求解背包问题。

设按照可装入物品的n个种类划分为n个阶段。状态变量ω表示装入第1种物品至第k种物品的总重量。决策变量x_k表示装入第k种物品的件数。则状态转移方程为:

$$\widetilde{\omega} = \omega - x_k\omega_k,$$

允许决策集合为:

$$D_k(\omega) = \left\{x_k \mid 0 \leqslant x_k \leqslant \left\lfloor\frac{\omega}{\omega_k}\right\rfloor\right\},$$

最优值函数$f_k(\omega)$是当总重量不超过ω千克，背包中可以装入第1种到第k种物品的最大使用价值。即：

$$f_k(\omega) = \max_{\substack{\sum_{i=1}^{k}\omega_i x_i \leq \omega \\ x_i \geq 0,\text{且为整数}(i=1,2,\cdots,k)}} \sum_{i=1}^{k} c_i(x_i) \quad ,$$

因而可写出动态规划的顺序递推关系为:

$$f_1(\omega) = \max_{x_1=0,1,\cdots,\left\lfloor\frac{\omega}{\omega_1}\right\rfloor} c_1(x_1),$$

$$f_k(\omega) = \max_{x_k=0,1,\cdots,\left\lfloor\frac{\omega}{\omega_j}\right\rfloor} \{c_k(x_k) + f_{k-1}(\omega - \omega_k x_k)\}, 2 \leqslant k \leqslant n,$$

然后，逐步计算出$f_1(\omega), f_2(\omega), \cdots, f_n(\omega)$及相应的决策函数$x_1(\omega), x_2(\omega), \cdots, x_n(\omega)$，最后得出的$f_n(a)$就是所求的最大价值，其相应的最优策略由反推运算即可得出。

动态规划解决问题首先要抽象出一个数字化的状态，然后可以确定状态的初始化状态并且可以写出状态的转移方程。后一个状态的转移计算应该依赖于已经计算出来的前面状态(可能是多个)。为了达到全局最优，前一个状态应该是局部最优的，并且后一个状态的计算依赖于前一个状态计算能够达到最终的全局最优，这被称为问题具有最优子结构。并且每一个局部最优状态不会因为后面的计算而发生改变，这被称为无后效性。

简单来说，动态规划的一个最优状态依赖于多个局部最优的结果而得出，这也就是为什么动态规划需要填表的原因。贪婪算法从原则上来说动态规划的一种特例，贪婪算法后一个状态只依赖于最优的前一个状态，而不是多个。

例题 5.2 有一辆最大货运量为10吨的卡车，用以装载三种货物，每种货物的单位质量及相应单位价值如表所示。应如何装载可使总价值最大？

表 5.2 例题 5.2 中每种货物的单位质量及相应单位价值

货物编号i	1	2	3
单位质量/吨	3	4	5
单位价值c_i	4	5	6

设第i种货物装载的件数为$\mathrm{x}_i(\mathrm{i}=1,2,3)$，则问题可表示为：

$$\begin{aligned} &maxZ = 4x_1 + 5x_2 + 6x_3 \\ &\text{s.t.}\begin{cases} 3x_1 + 4x_2 + 5x_3 \leqslant 10 \\ x_i \geqslant 0 \text{ 且为整数}(i = 1,2,3), \end{cases} \end{aligned}$$

解 可按前述方式建立动态规划模型，由于决策变量取离散值，所以可以用列表法求解，S 表示当前状态下的可用载重。

当$k=1$时，

$$f_1(S)=\max_{\substack{0\leqslant 3x_1\leqslant S\\ x_1\text{为整数}}}\{4x_1\},$$

或

$$f_1(S)=\max_{\substack{0\leqslant x_1\leqslant \frac{S}{3}\\ x_1\text{为整数}}}\{4x_1\}=4\left[\frac{S}{3}\right]$$

计算结果见下表。

表 5.3 例题 5.2 中 k=1 时计算结果

S	0	1	2	3	4	5	6	7	8	9	10
$f_1(S)$	0	0	0	4	4	4	8	8	8	12	12
x_1^*	0	0	0	1	1	1	2	2	2	3	3

当$k=2$时，

$$f_2(S)=\max_{\substack{0\leq x_2\leq \frac{S}{4}\\ x_2\text{为整数}}}\{5x_2+f_1(S-4x_2)\},$$

计算结果见下表

表 5.4 例题 5.2 中 k=2 时计算结果

S	0	1	2	3	4	5	6	7	8	9	10
x_2	0	0	0	0	0，1	0，1	0，1	0，1	0，1，2	0，1，2	0，1，2
c_2+f_1	0	0	0	4	4，5	4，5	8，5	8，9	8，9，10	12，9，10	12，13，10
$f_2(S)$	0	0	0	4	0，5	5	8	9	10	12	13
x_2^*	0	0	0	0	1	1	0	1	2	0	1

当$k=3$时

$$f_3(10) = \max_{\substack{0\le x_3\le 2 \\ x_3\text{为整数}}} \{6x_3 + f_2(10-5x_3)\}$$

$$= \max_{x_3=0,1,2} \{6x_3 + f_2(10-5x_3)\}$$
$$= max\{f_2(10), 6 + f_2(5), 12 + f_2(0)\}$$
$$= max\{13, 6+5, 12+0\}$$
$$= 13,$$

此时$x_3^* = 0$，逆推可得全部策略为：

$x_1^* = 2, x_2^* = 1, x_3^* = 0$，最大价值为13。

5.3.3 遗传算法

基本遗传算法从初始种群开始，通过自然选择、交叉和变异操作产生新的种群，并不断更新，直到找到最优解。针对背包问题，人们采用遗传算法进行了很多研究。各类研究的不同点主要体现在编码方式上。应用遗传算法最关键的一步，就是如何设计编码的问题。

下面分别介绍这三类编码方法。

1. 二进制表达法（Binary representation）

二进制表达，实际上也就是直接将问题的解向量映射为染色体的编码。因为问题的解本身就是$0-1$变量，所以染色体表现为二进制串。二进制串是$0-1$背包问题的自然表达，其中1表示选入，0表示未选入。二进制表达，实际上也就是直接将问题的解向量映射为染色体的编码。因为问题的解本身就是$0-1$变量，所以染色体表现为二进制串。

x=[x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}]

染色体：	0	1	0	1	0	0	0	0	1	0

这表示项目2，4和9被选入背包

但这种表达会产生不可行解，即违反约束条件。当染色体中的1太多时，可能超过背包的承重量。

当存在不可行解时，存在以下两种解决方法：

（1）惩罚法

给每个不可行解指定一个惩罚值。通过在目标函数中加入一个惩罚项来处理约束。如果解不满足约束条件，就在目标函数中对该解进行惩罚，从而降低其适应度。惩罚项的强度通常与违反约束的程度成正比。

Gordon 和 Whitley 给每个不可行解一个简单的罚值。

适值函数构造为：$eval(x) = f(x)p(x)$

极大化问题的惩罚函数为：

$$p(x) = 1 - \frac{\left|\sum_{j=1}^{n} \omega_j x_j - W\right|}{\delta}$$

$$\delta = \max\{W, \left|\sum_{j=1}^{n} \omega_j - W\right|\},$$

$\left|\sum_{j=1}^{n} \omega_j x_j - W\right|$指的是约束违反量或背包剩余量，这里的惩罚函数是根据问题的特点给出的。惩罚函数不仅惩罚超过背包承重量的不可行解，也惩罚背包承重量利用不足的可行解。

对于这类最优解位于可行域与不可行域边界上的问题，这种惩罚函数正是所需要的。

（2）解码法

用贪婪近似法将不可行解转化为可行解。当生成的解不满足约束条件时，应用特定的修复操作将其转换为满足约束的可行解。这种方法可以在不影响算法搜索空间的情况下，保证解的可行性。

首先，将$x_j = 1$的项目按价值与重量比（单位重量的价值）的降序排列。按优先适合启发式选择项目，直到背包不能再装为止。

例如：染色体为[x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}]

1	0	1	1	0	1	0	1	1	0

价值重量比：[0.1 0 0.2 0.8 0 0.3 0 0.5 0.6 0]

排序装包：x_4 x_9 x_8 x_6 x_3 x_1

需要注意的是，编码和解之间不存在简单的一一对应关系。多个表面上不同的染色体，可能对应一个解。

Gordon 和 Whitley 试验了惩罚法和解码法，一个20个项目的较难问题，惩罚法较好；而一个80个项目的较容易问题，解码法较好。

二进制表达的交叉、变异方法，可以采用我们曾经介绍过的基本方法。

2.顺序表达法（Order representation）

顺序表达法的核心思想是通过一个排列来表示物品选择的优先顺序。通常情况下，染色体（个体）由一个长度为n的基因序列组成，其中n是物品的数量,每个基因用一个不同的正整数代表一个项目，表示这些物品放入背包的优先级。

项目在顺序中的位置，可以看作是该项目选入背包的优先级。按项目的顺序用优先适合启发式方法，就可产生一个可行解。

例如：$W = 100$

表 5.5 各项目的重量和价值

项目 j	1	2	3	4	5	6	7
重量ω_j	40	50	30	10	10	40	30
价值c_j	40	60	10	10	3	20	60

初始化时，随机产生一个项目的排列顺序，作为一个染色体。假设 7 个项目的染色体为

1	6	4	7	3	2	5

这表示按照顺序优先选取物品1，然后是物品6，接着是物品4、物品7、物品3、物品2，最后是物品5。在背包容量有限的情况下，按照此顺序依次将物品放入背包，直到容量耗尽或没有更多可放入的物品为止。依此顺序,能进包的项目为[1，6，4，5]。因此，其中可行解为[1，6，4，5]。

$$x_1 = x_4 = x_5 = x_6 = 1$$

$$g(x) = 100 \leq W$$

$$f(x) = 73,$$

对于交叉操作，顺序编码通常使用部分匹配交叉（Partially Matched Crossover，PMX）或顺序交叉（Order Crossover，OX）等专门设计的交叉算子。这些交叉算子能够保证生成的新个体依然是一个合法的排列，即没有重复或遗漏的物品。

对于变异操作，常见的变异操作是交换变异（Swap Mutation），即随机交换染色体中的两个基因位置，以生成新的个体

这种编码方法下，染色体到可行解的映射是多对一的。这个特点使得该方法大大地减少了种群的多样性。因此，应该用大的种群来增加种群的多样性。

3.变长表达法（Variable-length representation）

变长表达法是一种可以动态调整解长度的编码方式，适用于物品数量或解长度动态变化的场景。在背包问题中，变长编码可以表示选择不同数量的物品。

在变长表达法中，每个个体由一个包含选择物品的索引集合表示。该集合的长度可以根据实际的选择情况进行调整。这意味着每个个体的长度是动态的，而不是固定的。

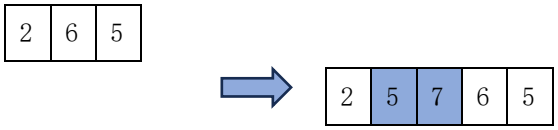
例如，染色体：[2,6,5]、[1,6,4,5]都是可行的背包。

其中，2、6、5表示选择了编号为2、6和5的物品，其他物品未被选择。与固定长度的二进制编码不同，变长表达法允许解的长度根据实际选择的物品数量动态变化，染色体中项目的顺序没有意义。[2,6,5]和[2,5,6]都是一样的。

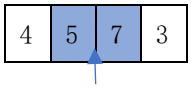
初始化时，产生一个随机的项目顺序，基于这个顺序按优先适合启发式构造一个可行背包。

对于交叉操作，采用插入交叉法。首先在第一个双亲上选择一个断点，在第二个双亲上选一截片断;然后将片断插入第一个双亲的断点处;接着删除片断外的重复基因，得到一个原始后代，最后按优先适合启发式从原始后代中选择基因，即得到一个可行背包。

首先插入片段 

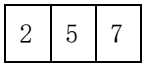


重复基因：5

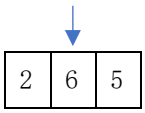


删去重复基因 5

按优先适合启发式选择基因，得到：



对于变异操作，首先随机地删除一个基因，以随机的顺序增加染色体中没有的基因（所有的基因），对于以上形成的原始后代用优先适合启发式产生一个可行的背包。



（随机增加染色体中没有的基因）

这种编码方式以及交叉、变异方法，是针对问题的特点而设计的，可以理解为通过改进遗传算子来处理约束，也就是前面提到的约束处理技术中的“改进遗传算子策略”。

5.4 背包问题的应用

基本背包问题的应用广泛且多样，其中之一是资源分配问题。在实际的企业经营中，常常需要处理多个项目的资源分配，每个项目都有不同的资源消耗和收益。如何在总资源量有限的情况下，优化各个项目的分配，使得企业的总收益最大化，正是背包问题的核心思想。在这种情况下，背包容量对应企业能够分配的资源总量，而每个项目的资源消耗和收益则类似于背包中的物品重量和价值。企

业通过合理选择项目组合，确保在资源有限的条件下实现收益的最优化，避免资源的浪费和投资的低效。

另一个典型的应用是资金预算问题。在企业的投资决策中，资金的限制经常是必须面对的现实。在众多的投资项目中，每个项目的资金需求和预期收益不同。企业的目标是选择合适的项目投资组合，使得在有限的资金投入下，能够获得最大化的净收益。这与背包问题中的“背包容量”相对应，项目的资金需求对应物品的重量，而预期收益对应物品的价值。通过精确选择哪些项目能够带来最高的回报，企业可以在资金有限的情况下实现最佳的投资效果。

此外，工厂下料问题也可以通过背包问题来解决。在生产过程中，工厂需要将原材料进行加工切割，每种加工方式可能会带来不同的利润。原材料总量相当于背包的容量，而每种下料方式的利润则对应物品的价值。为了确保生产过程的经济效益，工厂需要合理规划下料方案，选择最优的下料组合，使得在固定的原材料量下，实现最大利润。这种问题常常出现在制造业中，尤其是在切割、铺设等需要精确分配资源的生产工艺中。

最后，背包问题也在运输装载中得到了广泛应用。例如，在集装箱运输中，运输公司需要根据货物的体积和价值选择合适的货物进行装载。每个货物的体积相当于物品的重量，而货物的价值或利润则对应物品的价值。由于集装箱的体积有限，如何选择合适的货物组合，确保在最大限度地利用空间的同时，货物的总价值也能够最大化，成为运输公司在装载过程中的重要任务。通过运用背包问题的优化方法，可以有效提高运输效率，降低运输成本，达到最佳的经济效益。

5.5 背包问题的变种问题

5.5.1 多选择背包问题

多选择背包问题（Multiple-Choice Knapsack Problem, MCKP）是背包问题的一种变形，在这种问题中，物品被分成多个相互排斥的类，每个类中有若干不同的项目。背包的目标是从每个类中选择一个物品，使得所选物品的总价值最大，同时满足背包的容量限制。这类问题通常出现在产品设计、资源分配和任务调度等实际应用中。

在多选择背包问题中，决策者需要在多个物品类别中做出选择，每个类别只能选择一个物品。这一约束使得问题的复杂度增加，因为在选择时必须考虑如何在多个类别之间进行优化，而不仅仅是从一组物品中选择。该问题的数学模型通常表现为以下形式：

目标是最大化背包中物品的总价值，同时满足背包的容量限制和每组最多选择一个物品的约束。设有n个物品类别，每个类别中有m_i个物品，x_{ij}是决策变量，表示是否选择第i类中的第j个物品。如果选择了该物品，x_{ij}为1；否则，x_{ij}为0。该问题的目标函数为：

$$\max f = \sum_{i=1}^{n}\sum_{j=1}^{m_i} v_{ij}\, x_{ij}\,,$$

v_{ij}为第i类中第j个物品的价值。背包的容量限制条件如下：

$$\sum_{i=1}^{n}\sum_{j=1}^{m_i} w_{ij}\, x_{ij} \le C,$$

其中，ω_{ij}为第i类中第j个物品的重量，C是背包的容量。每个物品类别中的选择约束要求对于每个类别i，只能选择其中的一个物品：

$$\sum_{j=1}^{m_i} x_{ij} = 1, \forall i \in \{1,2,\dots,n\},$$

二元变量约束：

$$x_{ij} = \begin{cases} 1, \text{选择第} i \text{类中的第} j \text{个物品} \\ 0, \text{否则} \end{cases}, i = 1,2,\dots,n, j = 1,2,\dots,m_i.$$

5.5.2 多约束背包问题

多约束背包问题（multi constrained knapsack problem）也称为多维背包问题或者多背包问题，它是带有一组约束（重量、尺寸、可靠性等）的背包问题。

N件物品和一个容量是V的背包，背包能承受的最大重量是M。每件物品只能用一次。体积是v_i，重量是m_i，价值是ω_i。求解将哪些物品装入背包，可使物

品总体积不超过背包容量，总重量不超过背包可承受的最大重量，且价值总和最大。输出最大价值。

目标函数是最大化背包中所有所选物品的总价值：

$$\max f = \sum_{i=1}^{n} \omega_i \, x_i \,,$$

条件如下：

所有选中的物品的总体积不超过背包的体积容量：

$$\sum_{i=1}^{n} v_i \, x_i \leq V,$$

所有选中的物品的总重量不超过背包的最大承重量：

$$\sum_{i=1}^{n} m_i \, x_i \leq M,$$

二元变量约束：

$$x_i = \begin{cases} 1, \text{选择第} i \text{个物品} \\ 0, \text{否则} \end{cases}, i = 1,2, \dots, n.$$

通过解决该问题，可以得到一个物品的最优选择组合，使得在满足体积和重量限制的前提下，背包中物品的总价值达到最大。该问题常见的应用包括资源分配、物流优化、生产调度等多个领域。例如，工厂的生产调度可以通过该模型来优化多个生产过程中的资源分配，最大化产品的利润，同时确保不超过设备的负荷和容量。

在求解多约束背包问题时，常用的算法包括动态规划、分支定界法和启发式算法等，尤其在物品和约束数量较多时，分支定界法和启发式算法（如遗传算法、模拟退火算法）较为常见，能够在较短时间内找到近似最优解。

5.5.3 有界背包问题

在$0-1$背包问题中将$x_j = 0$或1改成$0 \leqslant x_j \leqslant b_j, b_j$为给定的正整数，则得到如下定义的有界背包问题（bounded knapsack problem）：

$$\max f = \sum_{j=1}^{n} p_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} \omega_j x_j \leqslant C$$

$$0 \leqslant x_j \leqslant b_j, x_j \text{为整数}, j \in \{1,2,\cdots,n\},$$

有界背包问题表示同类物品（即价值与重量相同的物品）不止一个，设第j类物品有b_j个，因而根据需要可放入包中至多b_j个该类物品。实际上，即使是同类物品有任意多个的背包问题（称为无界背包问题），也可化成上述有界问题来求解。这是因为由包的容量约束，物品j至多有$\left\lfloor \frac{C}{w_j} \right\rfloor$个可放入包内，因此可取$b_j = \left\lfloor \frac{C}{w_j} \right\rfloor$，从而无界问题就转化成等价的有界问题。

5.5.4 无界背包问题

无界背包问题（unbounded knapsack problem）是背包问题的一个变种，在这种问题中，每个物品可以被选取无限次。也就是说，可以选择任意数量的某种物品，只要总重是或总体积不超过背包的容量，并且希望在此限制下最大化总价值。

设共有n种物品，背包的容量为V。第i种物品的体积（或重量）记为v_i，价值记为ω_i。用决策变量x_i表示第i种物品被装入背包的件数。由于每种物品都可以重复放入，x_i是一个不受上界限制的非负整数。模型的目标是在不超过容量V的前提下，使装入背包中的物品总价值达到最大。

$$\max f = \sum_{i=1}^{n} \omega_i x_i ,$$

条件如下：

所有选中的物品的总体积或重是不能超过背包的容量 V：

$$\sum_{i=1}^{n} v_i x_i \leq V,$$

非负整数约束:物品的选择次数x_i为非负整数，且可以任意取大。（对每种物品选择的数量没有限制）：

$$x_i \geq 0, \forall i \in \{1,2,\ldots,n\},$$

5.6 本章小结

背包问题是一类经典的组合优化问题，在实际应用中具有广泛的意义。通过对基本背包问题的研究，深入探讨了不同求解算法，并分析了其在资源分配、预算优化等实际问题中的应用。介绍了包括贪婪算法、动态规划和遗传算法等多种求解方法，并详细评估了每种方法的优缺点及适用场景。

贪婪算法虽然不能保证最优解，但其高效性使其成为实际问题中常用的求解工具。动态规划通过状态转移方程和递推关系来求解背包问题，特别适用于小规模问题。遗传算法通过模拟自然选择过程来搜索最优解，尤其适用于大规模复杂问题。

此外，背包问题的变种，如多约束背包问题、多选择背包问题和无界背包问题等，也得到了详细的讨论。不同类型的背包问题具有不同的求解策略，能够应对实际中资源分配的复杂性。

本章通过具体的应用案例，展示了背包问题在优化资源配置、资金分配、生产调度等领域的巨大应用潜力，并进一步探讨了如何通过现代算法和技术有效解决实际问题。

研学互通

为了推动理论教学与科研实践的有机融合，通过背包问题这一典型模型，将算法思想的系统学习与其在工程、经济等领域中的广泛应用相结合。这里遴选了若干具有代表性的研究资料，涵盖从基础概念到复杂变种的多个层面，系统梳理了背包问题的建模方式、求解策略及其算法演化。通过资料研读与问题探究，不仅可以掌握背包问题的基本原理与主要解法，还可深入理解其在现实问题建模中的核心地位，提升分析与解决复杂优化问题的综合能力。

(1) Martello, S., & Toth, P. (1990). Knapsack problems: Algorithms and computer implementations. John Wiley & Sons.

系统阐述了各种背包问题变种（0-1,有界/无界,子集和,多维等）的理论基础、

精确算法（分支限界、动态规划的各种优化）、启发式算法（贪婪、邻域搜索等）以及详细的计算机实现技巧。可以对背包问题家族全面、系统、深入的理解，掌握高级精确算法和启发式算法的设计精髓，了解实际实现中的优化技巧。

(2) Kellerer, H., Pferschy, U., & Pisinger, D. (2004). Knapsack problems. Springer-Verlag Berlin Heidelberg.

Martello & Toth 之后又一重要现代专著。内容更新更全面，涵盖了最新的理论进展、算法（包括更高效的动态规划、分支限界法、进化算法等）以及更广泛的应用领域（如金融、物流、密码学）。结构清晰，理论深度与实践指导并重。可以进一步了解背包问题研究的现代进展，接触更高效的算法变种和更复杂的应用场景，是进行深入学术研究或解决高难度实际问题的宝贵资源。

(3) Salkin, H. M., & De Kluyver, C. A. (1975). The knapsack problem: A survey. Naval Research Logistics Quarterly, 22(1), 127–144.

这篇经典综述系统总结了背包问题的基本形式、变体（如 0-1 背包、分数背包等）以及主要的求解方法，包括动态规划、分支定界法和贪心算法等。适合初学者了解背包问题的全貌。

(4) Cacchiani, V., Iori, M., Locatelli, A., & Martello, S. (2022). Knapsack problems—An overview of recent advances. Part I: Single knapsack problems. Computers & Operations Research, 143, 105692.

该文献回顾了单背包问题的最新研究进展，涵盖了各种求解算法的改进和实际应用案例，为深入研究提供了丰富的参考。

(5) Wang, Y., Liu, J., & Zhang, Y. (2023). Nature-inspired algorithms for 0–1 knapsack problem: A survey. Neurocomputing, 526, 126630.

文章综述了应用于 0-1 背包问题的自然启发式算法，如遗传算法、粒子群优化等，分析了它们的性能和适用场景，适合对智能优化方法感兴趣的读者。

(6) Pushpa, K. R., & Mrunal, K. (2016). A study of performance analysis on knapsack problem. International Journal of Computer Applications, 162(6), 1–5.

该研究比较了贪心算法、动态规划和分支定界法在解决背包问题时的性能表现，提供了不同算法在效率和准确性方面的实证分析。

思行经世：背包问题与精准扶贫：优化资源配置与社会效益的双赢

在我国当前的精准扶贫工作中，如何在有限的资源下实现最优的资源配置，以便最大化扶贫效益，一直是政策制定者和执行者关注的焦点问题。背包问题作为经典的资源优化问题，其核心思想为在有限资源下如何进行最优选择和配置，这一思想在精准扶贫的资源分配中得到了有效应用。在扶贫工作中，资源有限而需求却多样且复杂，如何科学配置资源，确保扶贫资金、物资和人力得到合理利用，帮助贫困地区提高脱贫致富的能力，成为了精准扶贫战略实施的关键。

在精准扶贫的背景下，背包问题的应用主要体现在如何优化扶贫资源的配置。假设每个贫困地区有不同的扶贫需求，如教育、医疗、基础设施建设、产业扶持等，而扶贫资金和物资是有限的。如何通过背包问题优化资源的分配，确保在有限的扶贫资金下，实现扶贫效果的最大化，成为核心挑战。通过合理地配置扶贫资源，可以实现每一项扶贫项目的最大效益，确保资金、物资和人力的最佳使用。例如，在某些贫困地区，通过对扶贫项目的优先级排序和资源优化配置，能够最大化解决贫困人口的基本需求，并逐步提升他们的生活质量。这不仅提高了扶贫效率，还减少了扶贫中的资源浪费，推动了精准扶贫工作的深入开展。

背包问题的优化帮助扶贫系统提高了资源配置的效率，减少了不必要的浪费，并使有限的扶贫资源得到了最大化的利用。技术创新不仅提升了扶贫工作的精准度，还与社会责任紧密结合，为实现全面建成小康社会的目标贡献了积极力量。精准扶贫不仅仅是对贫困人口的直接援助，更是全社会共同推进共同富裕的重要途径。

通过精准扶贫的案例，可以深入理解背包问题在社会资源优化中的应用，认识到如何在有限资源条件下通过科学的决策实现最大社会效益。应将科技创新与社会责任相结合，践行社会公平与共同富裕的理念，为扶贫事业做出贡献，推动社会的和谐与进步。背包问题在精准扶贫中的应用，不仅提升了资源的使用效率，减少了扶贫过程中的浪费，还为社会的可持续发展做出了贡献。通过这种方式，

能够理解背包问题在实践中的价值和意义，激发为社会的进步和共同富裕贡献智慧和力量的决心。

习题

习题 5.1 用贪婪算法分别求解下列 0-1 背包问题的实例:

(1)价值向量为$(p_1,\cdots,p_8)=(15,100,90,60,40,15,10,1)$

重量向量为$(w_1,\cdots,w_8)=(2,20,20,30,40,30,60,10)$, 容量为 105

(2)价值向量为$(p_1,\cdots,p_6)=(50,50,64,46,50,5)$

重量向量为$(w_1,\cdots,w_6)=(56,59,80,64,75,17)$, 容量为 190

习题 5.2 为保障航天飞机在为期三周的深空探测任务中，宇航员能够获得足量能量补给，后勤团队从地面实验室精选了六种高能航天食品。航天飞机货舱可用容积仅剩 20L，发射总质量限制为 15kg。请在满足体积与质量双重约束的前提下，从下表六种食品中选取若干，使所携带食品的总卡路里最大化。

表 5.6 习题 5.2 中食品体积、质量及其卡路里

食品	体积(L)	质量(kg)	卡路里(kcal)
冻干牛肉粒	3	2	300
坚果能量棒	4	5	500
速溶燕麦粥	1	1	150
蛋白质布朗尼	3	2	320
高热量巧克力	4	6	600
脱水蔬菜脆片	2	2	280

请使用遗传算法求解该背包问题，并给出遗传算法求解思路（编码、适应度、选择、交叉、变异及终止条件）。

习题 5.3 某人外出旅游，需要将五件物品装入背包，但背包质量有限，总质量不超过 13 千克。物品质量及其价值的关系如下表所示，试问如何装这些物品才使整个背包的价值最大？试着用贪心算法求解该问题并给出代码。

表 5.7 习题 5.3 中物品质量及其价值的关系

物品	质量/千克	价值/元
A	7	4
B	5	9
C	4	0.5
D	3	2
E	1	3

习题 5.4 考虑如下 0-1 线性背包问题，用动态规划法求解：

$$\begin{aligned}&\max 3x_1 + x_2 + 2x_3 + 2x_4 + 6x_5 + 4x_6,\\&\text{s.t. } 2x_1 + 6x_2 + 2x_3 + 4x_4 + 3x_5 + 9x_6 \leqslant 17,\\&x_j \in \{0,1\}, j = 1, \cdots, 6.\end{aligned}$$