第 8 章 集覆盖问题

在现实世界中，我们经常面临“用最少的资源覆盖尽可能多的需求”的优化任务。例如，如何以最少数量的广播电台覆盖所有城市、如何选择最小的监控摄像头组覆盖整个区域、如何构建最少的课程以涵盖所有教学要求。这类问题的共同特征是：存在一个大的目标集合和若干可以覆盖部分目标的子集合，目标是在覆盖全部目标的前提下，选取最少的子集合。集覆盖问题（Set Covering Problem, SCP）正是对这一类最优化任务的抽象建模，它不仅是组合优化中的基本问题之一，也广泛应用于运筹学、计算机科学、人工智能等多个领域。

集覆盖问题在理论研究和实际应用中都具有重要地位。它不仅是分析算法复杂性和性能边界的重要实例，也是许多近似算法与启发式方法设计的出发点。通过对集覆盖问题的深入探讨，读者不仅可以掌握该问题的数学建模与求解思路，还能进一步理解组合优化问题的基本特性与应对策略。

8.1 集覆盖问题介绍

8.1.1 集覆盖问题的产生和起源

集覆盖问题最早由数学家们在 20 世纪中期提出，并迅速引起了运筹学、计算机科学、组合优化、离散数学等多个领域学者的关注。这个问题的提出与实际应用需求密切相关，特别是在资源分配、网络设计和数据压缩等领域。集覆盖问题的研究不仅在理论上具有重要意义，还在实际应用中得到广泛应用，例如在设施布置、无线网络覆盖、以及数据编码等方面。

集覆盖问题的研究引起了计算复杂性理论的深入探讨，特别是在 1972 年，由 Richard Karp 等人将其归类为 NP-完全问题，这一发现使得该问题成为计算机科学中经典的组合优化问题之一。

1979 年，V. Chvatal 提出了一种基于贪心策略的启发式算法来解决集覆盖问题，并证明了该算法的近似比为 $\ln n$，其中n是全集的元素数目。这一结果对于实际求解大规模集覆盖问题具有重要的指导意义。贪心算法通过每次选择覆盖最多未覆盖元素的子集，不断更新已覆盖元素，直至全集被完全覆盖。尽管该算法不

能保证得到最优解，但它为集覆盖问题提供了一种有效的近似解法。

集覆盖问题的研究不仅体现在理论建模和算法设计上，还体现在如何通过新的启发式算法解决大规模实例上。随着计算技术的进步和应用领域的拓展，集覆盖问题的研究逐步深化，特别是在网络设计、优化问题和数据压缩中得到了越来越广泛的应用。

8.1.2 问题描述

集覆盖问题是一个经典的组合优化问题，广泛应用于诸如资源配置、设施选址、网络覆盖、数据压缩以及调度优化等多个实际领域。该问题的本质是：在给定的一组元素和若干个子集中选择一些子集，使得所有的元素都至少被一个被选中的子集包含，同时使得这些被选中的子集的代价总和最小。为了将该问题形式化表示，通常采用一个 0-1 矩阵作为输入，其中每一行代表一个待覆盖的元素或需求，每一列代表一个可供选择的子集或资源配置选项。矩阵中的元素为 0 或 1，若第i行第j列的元素为 1，则表示第j列所代表的子集可以覆盖第i行所代表的需求。每个子集的选择都伴随着一定的成本，整个问题的目标是找到一个最小成本的子集组合，使得所有的需求都被满足。

换句话说，集覆盖问题的核心在于在一个具有覆盖关系的集合系统中，通过选择合适的列（即子集），实现对所有行（即元素或需求）的覆盖，并且在此基础上最小化选择的代价。由于其组合结构的复杂性，集覆盖问题被证明是 NP-完全问题，这意味着在计算复杂性理论中，它是目前认为没有已知多项式时间算法可以求解所有实例的困难问题之一。因此，在实际应用中，通常依赖于启发式算法、贪心策略、分支定界法或近似算法来寻找高效的可行解，而非精确求解。

8.2 集覆盖问题数学模型

在集覆盖问题中，为了便于形式化分析与求解，可以将其建模为一个 0-1 整数线性规划问题。设有一个 0-1 矩阵$A = [a_{ij}] \in \{0,1\}^{m \times n}$，其中 m 表示需要被覆盖的元素数，n 表示可供选择的子集数量。每一列代表一个子集，每一行代表一个待覆盖的元素。当$a_{ij} = 1$时，表示第 j 个子集能够覆盖第 i 个元素，反之则否。

引入决策变量$x = [x_1, x_2, \dots, x_n]$，其中$x_j \in \{0,1\}$表示是否选择第 j 个子集；若$x_j = 1$，则表示选择了第 j 个子集；若$x_j = 0$，则未选择。各个子集的选取费用以向量$C = [c_1, c_2, \dots, c_n]$表示。

在此基础上，集覆盖问题的数学模型可以写为如下形式：

$$\min \sum_{j=1}^{n} c_j x_j \tag{8.1}$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1，\ i = 1,2, \dots, m \tag{8.2}$$

$$x_j \in \{0,1\}，\ j = 1，\ 2，\ \cdots, n \tag{8.3}$$

上述模型中，目标函数 8.1 表示所选择子集的总费用，优化的目标是最小化这一费用。约束 8.2 表达了覆盖要求：矩阵的第 i 行所对应的元素必须被至少一个被选中的子集覆盖。由于$a_{ij} \in \{0,1\}$，只有当$a_{ij} = 1$且$x_j = 1$时，$a_{ij} x_j$才为正，因此该不等式实际上规定了在所有能够覆盖元素 i 的子集中，至少应有一个被选中。最后，约束 8.3 的二元性约束确保该模型为一个 0-1 整数规划问题，限制了解的可行性空间，同时也增加了问题的复杂性。

例题 8.1 某高校计划在校园的六个不同区域中设立若干个图书资料服务点，以实现对六座主要教学楼的服务覆盖。每个候选区域可以建立一个服务点，建立不同服务点所需的费用各不相同，同时，每个服务点能够覆盖的教学楼也不尽相同。为了在保证所有教学楼都能被服务到的前提下，尽可能降低总建设成本，学校希望确定一个最优的设点方案。

解 假设将六个候选设点区域编号为$S_1, S_2, \dots, S_6$，其对应的建设费用依次为 10 万元、15 万元、11 万元、10 万元、8 万元和 2 万元，用向量形式表示为：

$$C = [10,15,11,10,8,2],$$

学校希望覆盖的目标包括六座教学楼，编号为$e_1, e_2, \dots, e_6$。各区域服务点与教学楼之间的覆盖关系可以表示为一个$\mathbf{6 \times 6}$的$0-1$矩阵 A：

$$A=\begin{bmatrix}1&1&0&1&0&0\\0&0&1&1&0&0\\1&1&1&0&0&0\\0&0&0&0&1&1\\0&0&1&0&0&1\\0&0&0&1&1&0\end{bmatrix},$$

其中第i行第j列的元素为1表示教学楼e_i可被建在区域S_j的服务点所覆盖。例如，第一行显示，教学楼e_1可由设在区域S_1, S_2或S_4的服务点提供服务。

为了构造数学模型，引入 0-1 决策变量$x_j \in \{0,1\}$，代表是否在第j个区域设立服务点。若$x_j = 1$，则表示选择在该区域设点；否则不设。则该问题可表述为以下的整数线性规划模型：

$$\min 10x_1 + 15x_2 + 11x_3 + 10x_4 + 8x_5 + 2x_6$$

$$\text{s.t.}\, x_1 + x_2 + x_4 \geq 1$$

$$x_3 + x_4 \geq 1$$

$$x_1 + x_2 + x_3 \geq 1$$

$$x_5 + x_6 \geq 1$$

$$x_3 + x_6 \geq 1$$

$$x_4 + x_5 \geq 1$$

$$x_j \in \{0,1\}, j = 1,2,\dots,6.$$

该模型的目标函数即为所选服务点的总建设成本。六个不等式约束保证每一座教学楼至少被一个服务点所覆盖。变量的二元性则体现了选址决策的离散性。在求解该问题时，需综合考虑各服务点的建设费用与其所能覆盖的教学楼数量之间的关系。例如，第六个服务点的建设成本最低，仅为 2 万元，但其只能覆盖第 4 与第 5 两座教学楼。而第二个服务点虽然代价高达 15 万元，但其覆盖范围较广。

8.3 集覆盖问题求解

集覆盖问题，根据问题规模大小，有不同的解法。首先，Balas 和 Ho 提出了割平面方法，这是一种经典的最优化方法。该方法通过将原问题分解为多个较小的子问题，从而减小计算复杂度。割平面方法为解决集覆盖问题提供了有效的理论基础。Fisher 和 Kedia 则提出了用于解决最多 200 行 2000 列集覆盖问题的对偶启发式算法。这一算法通过对问题的对偶形式进行优化，能够高效地处理大规模问题，尤其是在求解线性规划时表现出了良好的性能。

Beasley 和 Jornsten 则结合了 Lagrangian 启发式方法、可行解排除约束（Gomory f-cuts）和改进的分支策略，成功解决了最多 400 行 4000 列的集覆盖问题。这些改进方法不仅提高了求解效率，还在处理大规模集覆盖问题时表现出了良好的性能。此外，Harche 和 Thompson 开发了一种名为列减少算法（Column Subtraction Algorithm）的直接方法，专门针对大规模稀疏矩阵的集覆盖问题。该算法在处理稀疏矩阵时显著提高了效率，解决了传统方法在大规模稀疏情况下的不足。

近年来，随着进化计算方法的崛起，集覆盖问题的求解也进入了新的研究阶段。Jacobs 和 Brusco 开发了模拟退火算法，并在求解最多 1000 行 10000 列的大规模问题时取得了显著成功。模拟退火算法在优化问题中具有较强的全局搜索能力，对于解决集覆盖问题中的大规模复杂情况具有良好的适应性。Sen 对模拟退火算法和简单遗传算法的性能进行了比较研究，为选择适当的启发式算法提供了理论支持。通过性能对比，Sen 揭示了两者在不同规模和复杂度问题中的优劣。

另外，Beasley 和 Chu，以及 Gonzalez，Hernandez 和 Corne 等学者进一步探索了遗传算法在求解大规模集覆盖问题中的应用。这些研究不仅拓展了遗传算法的应用范围，还提出了一些有效的算法改进策略，增强了遗传算法在集覆盖问题求解中的表现。

接下来重点介绍提出的求解大规模集覆盖问题的遗传算法。

8.3.1 遗传算法求解集覆盖问题

1.编码

在集覆盖问题的求解中，编码方式扮演着重要的角色。常见的编码方式有基于列的表示和基于行的表示两种。

(1)基于列产生初始种群

基于列的表示方式是将问题的解向量直接映射为染色体的编码，其中每个基因是一个 0-1 变量，表示是否选择该列。这种方式的优势在于它能够简洁地表示问题的核心特性，并且编码的方式直观易懂。具体来说，在集覆盖问题中，每个基因表示一个待选列，如果该列被选中，则该基因值为 1，否则为 0。由于问题解本身就是 0-1 变量，这种编码方式便于直接在遗传算法中使用。

以例 8.1 中数据为例，以列为基因的染色体表示为：

x_1	x_2	x_3	x_4	x_5	x_6
1	0	1	0	1	0

这表示选择第 1，3，5 列的组合来覆盖所有的行。

然而，基于列的表示方式也存在明显的缺点。主要的问题在于，初始化的种群或经过交叉变异后，染色体可能不满足可行性约束，也就是说，解的对应矩阵中并不是所有的行都被覆盖了。这一问题的存在使得有些解可能并不符合实际的需求。因此，在遗传算法的操作中，必须引入一些矫正机制来将不可行的解转换为可行的解，这一过程的设计对算法的效率和优化能力至关重要。

虽然基于列的表示方式有其优势，但如何有效处理不可行染色体并确保每一个解决方案都能满足约束条件，是需要特别注意的问题。实现这一点可能需要通过特定的机制，如惩罚函数、修复操作等，以保证算法的收敛性和解的质量。

基于列的编码方式因其简单、直观的特点而广泛应用于集覆盖问题的求解，但也需要对不可行解进行有效的处理，以确保遗传算法能够在复杂的约束条件下正常工作。

在遗传算法中，处理不可行解的策略至关重要，常见的三种处理不可行解的策略包括拒绝策略、惩罚策略和修补策略。

拒绝策略是一种简单直接的方法，要求在繁殖过程中生成不可行解时直接拒绝这些解。然而，这种策略可能会导致整个种群中的可行解数量骤减，甚至在算法运行过程中无法获得任何可行解。即使通过多次繁殖可能获得可行解，但不可行解的存在会大大拖慢算法的收敛速度，增加计算时间。

惩罚策略是通过对不可行解施加惩罚来减少其适应度，从而抑制不可行解的产生。惩罚策略并不总是能有效改善解的质量，甚至可能导致收敛到次优解。

修补策略则是根据问题的具体特性，通过设计一定的修复规则来调整不可行解，使其转化为可行解。修补策略的效果高度依赖于设计者的经验和问题的特性。一个好的修补策略能够有效地转换不可行解，并优化问题的解决过程，提升算法的效率和解的质量。

下面介绍一种附加启发式算子，专门用于将不可行解转换为可行解。该算子包括两个主要步骤：首先，通过检查当前解中哪些行没有被覆盖，找出所有未满足约束条件的部分；其次，添加必要的列，确保每一行都能被覆盖。添加列之后，为了确保解的质量，算子还会执行局部搜索，进一步优化解中的冗余部分。这样，这种策略不仅能够修复不可行解，还能够提升算法的效率，使得遗传算法在处理大规模问题时表现得更加高效。

变量定义如下：

表 8.1 遗传算法求解集覆盖问题的变量定义

变量	定义
I	所有行的集合
J	所有列的集合
α_i	覆盖行$i(i \in I)$的列的集合
β_j	被列$j(j \in J)$覆盖的行的集合
S	解中列的集合
ω_i	S中覆盖行$i(i \in I)$的列的数量
U	未被覆盖的行的集合

首先需对列集合进行预处理排序。所有列先按费用$\boldsymbol{c_j}$升序排列，当费用相同

时，按该列覆盖的行的数量β_j降序排列，这一排序规则为后续列的选择提供了初始优先级框架，确保在同等费用下优先考虑覆盖能力更强的列。

第一步，需对覆盖计数进行初始化。对于$\forall i \in I$，定义ω_i为初始解集合S与包含行i的列集合α_i的交集大小，即$\omega_i := |S \cap \alpha_i|$，这一步旨在记录每一行初始时被解集合覆盖的次数。

第二步，构建未被覆盖的行集合$U := \{i|\omega_i = 0, \forall i \in I\}$，即找出当前解 S 中没有任何列覆盖的行。

第三步，对于未覆盖行集合U中的每一行i（按照i升序处理），需要在α_i中寻找一个列j，使得$c_j/|U \cap \beta_j|$达到最小值,这里，最小化$c_j/|U \cap \beta_j|$表示希望第j列费用低,而且在当前解中加入第j列后覆盖尚未被覆盖的行数多。列的选择规则是：首先按费用升序排列，若费用相同，则按覆盖行的数量降序排列。这里，c_j是列j的费用，β_j是被列j覆盖的行的集合，而$|U \cap \beta_j|$表示被第j列覆盖但目前尚未被当前解覆盖的行的数量。找到这样的列j后，将其加入解S，并更新相关数据：对于β_j中的每一行i，令$\omega_i := \omega_i + 1$，同时从U中移除β_j中的行。这一过程的目的是选择性价比最高的列，即费用较低且能覆盖较多未覆盖的行。

第四步，对解S中的每一列j（按照j降序处理），检查是否对于所有$i \in \beta_j$，都有$\omega_i \geq 2$。如果条件成立，则令$S := S - j$，$\omega_i = \omega_i - 1$。这一局部搜索步骤旨在移除冗余列，即那些移除后仍能保证所有行被覆盖的列，从而减少解中的冗余部分，提升解的质量。

经过上述步骤后，解S成为一个没有冗余列的可行解，即所有行都被至少一个列覆盖，且不存在多余的列。这一过程确保了算法输出的解既满足约束条件，又具有较高的优化性。

例 8.2 选用例 8.1 的数据，为了方便演示，这里直接使列的费用c_j按升序排列。

$$C = [2,8,10,10,11,15]$$

$$A=\begin{bmatrix}1&1&0&1&0&0\\0&0&1&1&0&0\\1&1&1&0&0&0\\0&0&0&0&1&1\\0&0&1&0&0&1\\0&0&0&1&1&0\end{bmatrix}$$

解 首先计算α_i，覆盖行$i(i\in I)$的列的集合：

$$\alpha_1=\{1,2,4\}$$
$$\alpha_2=\{3,4\}$$
$$\alpha_3=\{1,2,3\}$$
$$\alpha_4=\{5,6\}$$
$$\alpha_5=\{3,6\}$$
$$\alpha_6=\{4,5\}$$

给定初始解为 S={1,3}，即第 1 列和第 3 列被选中，第 1 列(β_1={1,3})覆盖行 1 和行 3，第 3 列(β_3={2,3,5})覆盖行 2、行 3 和行 5。那么S中覆盖行$i(i\in I)$的列的数量为：$\omega_1=1$，$\omega_2=1$，$\omega_3=2$，$\omega_4=0$，$\omega_5=1$，$\omega_6=0$，即行 4 和行 6 未被覆盖：未覆盖行集合:U={4,6}。

按照 i 的升序处理集合 U,对于 i=4，$\alpha_4=\{5,6\}$，$\beta_5=\{4,6\}$，$\beta_6=\{4,5\}$，计算：

$$j=5:|U\cap\beta_5|=|\{4,6\}\cap\{4,6\}|=2,\frac{c_5}{|U\cap\beta_5|}=\frac{11}{2}=5.5$$

$$j=6:|U\cap\beta_6|=|\{4,6\}\cap\{4,5\}|=1,\frac{c_6}{|U\cap\beta_6|}=\frac{15}{1}=15$$

由于 j=5 时，$c_j/|U\cap\beta_j|$达到最小值，将 j=5 加入S，更新S为{1,3,5}，这时$\omega_4=1$，$\omega_6=1$，$U=U-\beta_5=\emptyset$，此时所有行被覆盖。

对于 j=5，$\beta_5=\{4,6\}$，此时$\omega_4=1$，$\omega_6=1$，不存在任意$\omega_i\geq 2$，无需移除；

对于 j=3，$\beta_3=\{2,3,5\}$，此时$\omega_2=1$，$\omega_3=2$，$\omega_5=1$，不存在任意$\omega_i\geq 2$，无需移除；

对于 j=1，$\beta_1=\{1,3\}$，此时$\omega_1=1$，$\omega_3=2$，不存在任意$\omega_i\geq 2$，无需移除；

此时所有行无冗余，可行解为{1,3,5}。

(2)基于行产生初始种群

染色体的长度与问题的行数相同，每个基因对应一行，其值表示覆盖该行的列编号。这种方法的最大优点是，在交叉和变异等遗传操作中，能始终保证解的可行性。因为每个基因都对应一个行，并选择一个覆盖该行的列，无论操作如何变化，解都能满足约束条件。

但这种表示也有缺点：同一解可能对应多种染色体形式。例如，在不同行选择不同列覆盖时，只要列属于同一选中集合，结果相同。这增加了灵活性，但也可能因搜索空间冗余降低效率。

考虑一个覆盖问题，6 行 6 列，覆盖矩阵如下：

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

染色体$x = [2,3,3,6,3,4]$表示各行选择的列：

行	1	2	3	4	5	6
列	2	3	3	6	3	4

对应的基于列的表示为$x = [0,1,1,1,0,1]$。但染色体$x' = [4,3,2,6,3,4]$也能表示相同的解，可见可行性虽然得到了保证，但这种编码到解的多对一映射，会降低种群的多样性，需要靠扩大种群规模来弥补，这会导致遗传算法搜索效率的下降，降低算法的收敛速度。

2.交叉

在遗传算法求解集覆盖问题的研究中，交叉算子的设计需要兼顾解的结构特征与适应值评价，以实现优质基因的有效继承与组合。Beasley 和 Chu 提出的熔合算子（fusion operator）是一种典型的基于适应值的通用交叉算子，其核心思想是在保留父代结构信息的同时，通过适应值加权的概率选择机制提升子代质量。

该算子的执行过程如下：设父代染色体为P_1和P_2，子代染色体为C，$\mathrm{F}(P_1)$

与 $\mathrm{F}(P_2)$分别表示二者的适应值。首先初始化位置计数器$i=1$，逐位处理染色体基因：若P_1与P_2在第i位的基因相同（即$P_1[i]=P_2[i]$），则子代直接继承该位置基因，即$C[i]=P_1[i]=P_2[i]$；若该位置基因不同（$P_1[i]\neq P_2[i]$），则以适应值为权重构建选择概率，其中以概率

$$p=\frac{\mathrm{F}(P_1)}{\mathrm{F}(P_1)+\mathrm{F}(P_2)},$$

让$C[i]=P_1[i]$；以概率$1-p$，让$C[i]=P_2[i]$，从而使适应值更高的父代基因以更大可能性进入子代。重复上述过程直至处理完所有基因位$(i=n)$，最终生成唯一的子代染色体。

与传统交叉算子相比，熔合算子具有显著优势：从编码通用性来看，其适用范围不依赖于染色体编码形式，无论是二进制编码还是整数编码均可直接应用，突破了特定编码结构的限制；从适应值整合机制来看，传统交叉算子大多忽略适应值对基因选择的影响，仅有实数编码下基于方向的交叉算子涉及适应值函数，而熔合算子通过概率加权机制将适应值直接融入基因选择过程，实现了对优质父代特征的定向强化。

3.变异

在遗传算法求解集覆盖问题中，变异操作通过随机翻转$0-1$编码染色体的基因位维持种群多样性。传统变异算子以小概率独立选择变异位置，在保持解结构稳定的同时引入局部扰动，避免早熟收敛。

变异率为变异基因数与种群总基因数的比值（种群总基因数=个体数×染色体长度）。Beasley 和 Chu 提出一种可变的变异方案，变异率随遗传代数的增加，逐渐增大，公式为

$$m=\frac{m_f}{1+\exp\left[-4m_g\dfrac{(c-m_c)}{m_f}\right]},$$

其中，c表示产生的子代个体的数量，m_f表示最终稳态的变异率,由用户指定，m_c是变异率达$m_f/2$时的子代数量，m_g为$c=m_c$时的斜率参数。m_c和m_g表示的值由特定问题的遗传算法收敛所需要的速率来确定。

该变异方法的特点是，因早期子代个体数量 c 较小，分母值较大，故变异率维持在较低水平；当子代个体数量增加至m_c时，变异率恰好达到$m_f/2$；随后，随着子代个体数量继续增加，变异率逐渐递加到m_f。

4.替换

稳态替换：子解将替换种群中随机选择的成员（通常是具有高于平均值的适应度值的成员）。需要注意的是，高于平均值的适合度意味着不太适合。

5.步骤总结

遗传算法的第一步是随机产生 N 个解，构成初始种群，并将迭代次数 t 设为 0。初始种群由 N 个可能的解组成，每个解是一个二进制向量，表示是否选择某些子集来覆盖集合中的元素。这些解的生成是随机的，以确保种群具有足够的多样性，为后续的优化提供广泛的搜索基础。初始化的质量直接影响遗传算法的收敛速度和解的优劣，因此 N 的大小通常需要根据问题的规模进行合理选择。

接下来，采用熔合杂交算子从种群中生成一个新解 C。熔合杂交算子是一种基于父代解特征的杂交方法，通常从种群中选择两个父代解，通过结合它们的优势基因构造新解 C。这种方法能够有效传递父代的优良特性，同时引入新的可能性。新解 C 的生成是遗传算法搜索过程中的核心环节，它通过模拟生物杂交机制，推动种群向更优的方向演化。

随后，对新解 C 中 k 个随机选出的列进行变异操作。变异是通过随机改变 C 的部分元素（例如，将二进制位从 0 翻转为 1 或从 1 翻转为 0）来实现的，k 为预设的变异参数。变异操作的目的是增加种群的多样性，避免算法过早陷入局部最优解。变异后的新解可能暂时偏离可行解的定义，因此需要进一步处理以确保其有效性。

紧接着，采用启发式算子对变异后的新解 C 进行调整，确保其成为集覆盖问题的可行解，并去除冗余列。启发式算子通常结合贪心策略，检查新解是否覆盖了所有元素，若未覆盖，则添加必要的子集；同时，分析已选子集中是否存在冗余（即移除后仍能保持覆盖的部分），并将其删除。这一过程不仅保证了解的可

行性，还优化了解的成本，使其更接近最优解。

在将新解 C 加入种群之前，需要检查其与种群中现有解的唯一性。如果 C 与种群中的任意解完全相同，则返回第二步，重新生成新解；否则，将迭代次数 t 增加 1。这一检查步骤旨在维持种群的多样性，避免重复解的堆积。若新解被判定为唯一，则进入后续的替换阶段。

随后，从种群中随机选出一个适应值低于种群平均适应值的个体，并用新解 C 替代该个体，这一过程称为稳态复制方法。适应值通常根据解的成本或覆盖效率计算，低于平均值的个体被认为是较差的解。稳态复制保持了种群规模的恒定，同时通过引入新解 C 逐步提升种群的整体质量。这种方法在遗传算法中广泛应用，具有稳定性和高效性。

最后，重复第二步至第六步，直至满足终止条件，然后输出种群中的最优解。终止条件可以设定为达到最大迭代次数、最优解在若干代内未改善，或解的质量达到预期目标。算法结束后，种群中成本最低且满足覆盖要求的那一个解被选为最终结果。这一迭代过程体现了遗传算法通过选择、杂交和变异逐步逼近全局最优解的核心思想。

8.3.2 贪婪算法求解集覆盖问题

贪心算法是一种常见的启发式算法，广泛应用于求解各类组合优化问题。在集覆盖问题的求解中，贪心算法通过每次选择覆盖最多未覆盖元素的子集，逐步构建一个解，直到所有元素都被覆盖为止。贪心算法的优点是计算效率较高，且实现简单，但它通常不能保证得到最优解。尽管如此，贪心算法可以为集覆盖问题提供一个接近最优解的近似解，并且特别适用于大规模问题。

在集覆盖问题中，给定一个元素集合$U = \{u_1, u_2, \dots, u_m\}$，以及若干个子集$S_1, S_2, \dots, S_n$，每个子集$S_j$覆盖了若干元素。每个子集$S_j$都有一个费用$c_j$，即选择该子集的代价。目标是从这些子集中选择若干个子集，使得所有元素都至少被一个子集覆盖，同时所选子集的总费用最小。

贪心算法的核心思想是，在每次选择中，选择覆盖最多未覆盖元素的子集，并将该子集加入解集合，直到所有元素都被覆盖。每次选择都基于当前的局部最

优解——即选择当前能覆盖最多未覆盖元素的子集。

贪心算法的步骤是，首先初始化所有待覆盖的元素集合U和所有可选子集的费用。接下来，选择当前能覆盖最多未覆盖元素的子集，将其加入解集合，并更新已覆盖的元素。这个过程重复进行，直到所有元素都被覆盖。需要注意的是，贪心算法的选择标准是覆盖最多的未覆盖元素，但这种选择并不保证最终能得到最优解。

对于贪心算法求解集覆盖问题的实际操作，在每次选择覆盖子集时，通常会选择当前未覆盖元素最多的子集，更新已覆盖的元素集合，并继续此过程，直到所有元素被覆盖。贪心算法并不考虑已选择的子集之间的关系，因此可能会错过最优解。但它的计算效率较高，尤其在面对大规模数据集时，能够在合理时间内得到一个可行的解。

尽管贪心算法无法保证最优解，但在实际应用中，贪心算法仍然被广泛采用，尤其是在问题规模较大时。对于集覆盖问题，贪心算法提供了一种有效的近似解法，能够在短时间内给出较好的解决方案。

例 8.3 某市政府计划在市区的不同位置安装交通监控摄像头，以实现对主要交通路线的监控覆盖。该市区有多个主要路段，市政府希望通过合理选址，保证所有路段都能覆盖到，同时控制安装监控摄像头的总费用。每个候选监控摄像头的安装位置覆盖的路段不同，而每个位置的建设费用也各不相同。政府希望在最小的成本下，确保每一条主要路段都有至少一个监控摄像头进行监控。

解 假设该市区有 5 个主要路段（编号为L_1, L_2, L_3, L_4, L_5），并且有 6 个候选监控摄像头安装位置（编号为$S_1, S_2, S_3, S_4, S_5, S_6$）。每个监控摄像头能够覆盖一定数量的路段，且每个位置的建设费用不同。建设费用和覆盖关系可以用下表表示：

表 8.2 例 8.3 中监控安装费用及覆盖路段

监控位置	安装费用（万元）	覆盖路段
S_1	10	L_1, L_2
S_2	15	L_1, L_3
S_3	12	L_2, L_3, L_4

S_4	9	L_4, L_5
S_5	8	L_3, L_5
S_6	6	L_1, L_5

采用贪心算法解决这个问题。贪心算法的关键是每次选择当前能够覆盖最多未覆盖路段的监控摄像头，并更新已覆盖的路段集合，

第1步，初始时，所有路段L_1, L_2, L_3, L_4, L_5都未被覆盖。选择能覆盖最多未覆盖路段的监控摄像头。通过分析，发现S_3能够覆盖L_2, L_3, L_4，且费用为12万元。因此，选择S_3，已覆盖路段更新为$\{L_2, L_3, L_4\}$，未覆盖的路段为L_1, L_5。

第2步，继续选择覆盖最多未覆盖路段的监控摄像头。此时，未覆盖的路段为L_1, L_5，而S_6刚好可以覆盖，且费用为6万元。因此选择S_6，已覆盖路段更新为$\{L_1, L_2, L_3, L_4, L_5\}$。

第3步，此时，所有路段都已被覆盖。最终，选择的监控摄像头位置为S_3, S_6，总费用为$12 + 6 = 18$万元。

8.4 集覆盖问题应用

集覆盖问题是一类数学优化问题，主要用于选择最少数量的集合，以覆盖所有指定的目标元素。虽然其概念简单，但在现实生活中有广泛的应用。以下是一些典型的应用实例，帮助我们更好地理解这一问题。

在网络性能监测中，集覆盖问题被用来确定最少数量的监测器，以覆盖所有网络链路。网络中的每一条链路都可以视为矩阵中的一行，监测器位置则为矩阵中的列，能监测哪些链路通过 0-1 矩阵表示。问题的核心在于如何选择最少数量的监测器，使其能够覆盖所有链路，从而确保对网络性能的全面监测。

在故障检测中，集覆盖问题被应用于测试点的选择。对于一个系统，要检测所有可能的故障，必须设置多个测试点。每个测试点能检测的故障类型不同，且检测的时间成本各异。我们需要在所有备选测试点中选择最少数量的点，使得所有故障都能被检测到，并且检测的时间总和最小。

物流中心的选址问题也是一种集覆盖问题。物流中心需要为各个需求点提供

服务，每个需求点只能由一个物流中心服务。通过构建 0-1 矩阵，矩阵的行代表需求点，列代表备选的物流中心，矩阵中的值表示某个物流中心是否能为某个需求点提供服务。我们需要选择最少数量的物流中心，既能满足所有需求，又能使建设成本最低。

枢纽站选址问题与物流中心选址问题类似，但其关注的焦点是运输流而非需求点。运输流从起点到终点的流量经过枢纽站，枢纽站之间的“干线流”形成了运输的规模效益。

交通消防设施点布局问题可以通过集覆盖问题来优化。交通事故发生时，交通消防设施点需要在规定时间内到达事故现场。我们将事故发生的地点视为“应急点”，而备选的消防设施点则为矩阵的列。矩阵中的元素为 1 表示某个消防设施能覆盖某个应急点，问题的目标是选取最少数量的消防设施点，以确保所有应急点都能得到及时救援。

软件测试用例的选择也是集覆盖问题的一个应用。在软件测试中，我们需要设计一组测试用例，以覆盖所有可能的测试需求。每个测试用例能覆盖不同的测试需求，通过构建 0-1 矩阵，可以帮助我们选择最少的测试用例，确保每个需求都得到测试，同时减少测试所需的人力和物力。

无线通讯网络的基站布局问题是集覆盖问题的另一应用。在设定的区域内，首先需要选择候选基站位置，并根据不同的设计标准（如最小发射功率等）将移动终端分配到各个基站。问题的目标是选取最少的基站位置，使得基站建设成本最低，同时满足网络覆盖需求。

8.5 集覆盖问题的特例

在经典的集覆盖问题（Set Covering Problem, SCP）中，目标是从给定的若干列中选择一个子集，使得所有的行都至少被某一列覆盖，并且使得所选列的总代价最小。这一问题的复杂性主要来自于列与行之间的覆盖关系不规则，以及每一列具有不同的费用。基于这一基本模型，在实际应用中又出现了一些重要的变种问题，它们在约束形式或目标函数上略有差异，但都可以在相似的建模框架中进行分析和求解。图中的内容提到了两种常见的变种，分别是单一费用问题

（Unicost Set Covering Problem）和集划分问题（Set Partitioning Problem），下面将对这两个变种进行进一步的阐述和扩充。

8.5.1 单一费用问题

单一费用问题（Unicost Set Covering Problem）是集覆盖问题的一个特殊情形，其假设所有的列具有相同的费用，即$c_j = c$对于任意 j 成立。这种情况下，原问题的目标函数就可以简化为最小化选中的列数，因为费用向量退化为常数向量，最小费用就等价于选择最少数量的列。形式化地，该问题可以表述为：

$$\min \sum_{j=1}^{n} x_j$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \forall i = 1, \dots, m$$

$$x_j \in \{0,1\}, \forall j = 1, \dots, n. \tag{8.4}$$

其中，$A = [a_{ij}]$是给定的$0-1$覆盖矩阵，表示第 j 列是否覆盖第 i 行。由于目标函数是对选中列的个数求和，因此该问题在本质上是一个组合优化问题，旨在找到最小的覆盖子集。

8.5.2 集划分问题

集划分问题（Set Partitioning Problem）则是另一类重要的变种，其主要特征在于覆盖约束从“至少覆盖一次”变为“恰好覆盖一次”，即每一行只能被一列所覆盖。这一问题的数学模型如下：

$$\min \sum_{j=1}^{n} c_j \, x_j$$

$$\sum_{j=1}^{n} a_{ij} \, x_j = 1, \forall i = 1, \dots, m \tag{8.5}$$

$$x_j \in \{0,1\}, \forall j = 1, \dots, n.$$

在此模型中，每一行（通常代表一个元素或任务）必须被且只能被一列（一

个集合或资源）覆盖。若将列视作集合，则这一约束意味着每一个元素恰好属于一个集合，也即对全集进行了不重叠的划分。这类问题在实际中有广泛应用，例如航班机组排班、车辆路径问题中的客户分配、任务调度中的资源唯一分配等。由于约束的变化使得问题具有更强的结构性，集划分问题常常在大规模整数规划中通过列生成法进行求解。

需要特别指出的是，尽管两类变种都属于 NP-难问题，但由于目标函数或约束的简化，它们往往可以作为启发式算法的子问题模块，或者作为原问题的近似解模型。在研究算法设计、计算复杂性及理论性质时，它们也常被用作基础模型加以分析。

8.6 本章小结

本章介绍了集覆盖问题，该问题作为经典的组合优化问题，广泛应用于资源分配、设施选址、网络设计等领域。集覆盖问题的核心是从一组给定的子集中选择若干个子集，使得所有元素都被覆盖，同时最小化选择的代价。由于其计算复杂性较高，集覆盖问题被归类为 NP-完全问题，因此在实际应用中，求解该问题通常依赖启发式算法和近似算法。

本章首先阐述了集覆盖问题的数学建模方法，通常采用 0-1 矩阵表示问题，其中每行表示待覆盖的元素，每列表示可供选择的子集。问题的目标是通过选择合适的子集覆盖所有元素，并最小化费用。通过将集覆盖问题转化为整数线性规划问题，能够有效地进行形式化分析和求解。

接着，本章重点介绍了遗传算法在集覆盖问题中的应用。遗传算法作为一种模拟自然选择和遗传学原理的优化方法，通过选择、交叉、变异等操作，有效地在大规模集覆盖问题中寻找近似最优解。在遗传算法的实现中，编码是关键步骤。集覆盖问题通常使用基于列的编码方式，将问题的解向量映射为二进制染色体，每个基因表示是否选择该列。熔合算子（Fusion Operator）被引入，这是一种基于适应值的交叉方法。熔合算子通过对父代基因进行加权选择，确保更优的基因在子代中得到继承，从而提高搜索效率和优化能力。

本章还讨论了常见的求解方法，如贪心算法，广泛应用于集覆盖问题的求解。

贪心算法通过每次选择覆盖最多未覆盖元素的子集，逐步构建一个解，虽然不能保证最优解，但能够提供有效的近似解，并且计算效率较高，尤其适用于大规模问题。

最后，本章还介绍了集覆盖问题的实际应用，包括网络监控、物流选址、软件测试等领域。通过具体的案例，展示了集覆盖问题在多个实际场景中的重要性和应用价值。尽管集覆盖问题的计算复杂性较高，但通过启发式算法的应用，能够为这些问题提供高效的求解方案。

研学互通

为进一步拓展对集覆盖问题的理解，增强理论学习与学术研究的关联性，本模块特别选取代表性文献，涵盖集覆盖问题的基本理论、近似算法、建模方法及其在实际中的应用。通过阅读这些文献，读者可以在掌握教材内容的基础上，了解该问题的研究发展脉络及前沿进展，为后续深入学习与研究提供参考。

(1) Karp, R. M. (1972). Reducibility among combinatorial problems.

本文首次将集覆盖问题列为 21 个典型 NP-完全问题之一，为其在理论计算机科学中的研究奠定了复杂性基础。通过归约方法，展示了集覆盖问题与其他经典问题（如布尔可满足性、图着色等）之间的紧密联系。阅读此文有助于理解集覆盖问题在 NP 理论框架中的历史地位及其复杂性根源。

(2) Feige, U. (1998). A threshold of ln n for approximating set cover. Journal of the ACM, 45(4), 634–652.

本文提出了一个令人震惊的理论结果：除非 P=NP，贪婪算法已经达到了集覆盖问题近似性能的最优界限。这意味着后续所有近似算法都无法在最坏情况下突破 lnn 这个边界。

(3) Williamson, D. P., & Shmoys, D. B. (2011). The design of approximation algorithms. Cambridge University Press.

本书特别强调集覆盖类问题在调度、网络优化、分配中的具体建模与求解方法。读者能通过实际案例学习集覆盖如何抽象出现实难题，并利用近似算法工具

包构造有效解，是理论与实践结合的典范教材。

(4) Balas, E., & Ho, A. (1980). Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. In Combinatorial optimization (pp. 37 – 60).

本文专注于集覆盖问题的算法研究，提出了结合割平面法、启发式方法和次梯度优化的混合算法框架。通过大量的计算实验，展示了这些方法在解决大规模集覆盖问题中的有效性和效率。

思行经世：集覆盖问题与城市公共服务优化中的“共同富裕”理念

以城市公共医疗服务网点选址为例，假设某城市希望在全市范围内合理配置多个医疗服务点，以确保每个社区、乡村都能在最短时间内获得医疗救助。每个医疗服务点能够覆盖不同的区域，同时其建设费用不同。通过运用集覆盖问题的优化模型，城市可以在最小的成本下，合理选择医疗服务点的位置，保证所有区域都得到医疗保障，从而实现基本公共服务的普惠性

这一问题的核心思想不仅仅是为了效率或成本的最小化，更是为了体现“共同富裕”的社会价值。通过合理布局医疗服务点，我们不仅提高了服务的可达性，还能够减少社会贫富差距，尤其是在偏远地区或低收入社区。通过这样的资源配置，技术不再是追求单纯效率的工具，而是服务社会公平、推动社会进步的重要手段

集覆盖问题的求解方法帮助我们突破了传统公共服务供给中存在的不平衡问题。在过去，许多城市的公共服务往往聚集在经济发达区域或核心城区，而偏远地区和贫困区域的资源分配则相对薄弱。集覆盖问题的算法使得我们能够科学地在城市的各个角落合理布局服务点，确保低收入群体和偏远地区居民能够平等享受公共服务。

这一实践凸显了“科技向善”的价值导向。技术不仅要为效率和成本的优化服务，更要关注技术应用的社会效益。在集覆盖问题的应用中，技术决策不仅是为了追求经济效益，更要深入考虑其社会责任和社会影响。例如，在解决城市医疗服务点选址时，运用集覆盖问题的算法来保证每个居民都能获得医疗救助，这

本质上体现了“技术服务于人民”的理念。

通过集覆盖问题的优化，我们可以看到技术创新不仅促进了社会资源的高效利用，还为推动社会公平和实现共同富裕提供了技术支持。这种通过优化资源配置解决公共服务问题的模式，是中国特色社会主义制度下“科技服务民生”的生动体现，体现了科技创新必须与社会责任、社会公平相结合。

习题

习题 8.1 某通信公司计划在某新兴城区部署 5G 基站，需覆盖所有 10 个居民区。候选基站位置有 7 个，每个基站的建设成本不同，且覆盖范围不同。下表列出了各基站的位置、建设成本（万元）及能覆盖的居民区编号：

表 8.3 习题 8.1 中基站建设成本及所覆盖居民区

基站编号	建设成本	覆盖的居民区
S_1	50	1, 2, 3
S_2	30	2, 4, 5
S_3	45	3, 6, 7
S_4	25	4, 5, 8
S_5	40	5, 9, 10
S_6	35	6, 7, 8, 9
S_7	20	8, 9, 10

请给出该问题的数学模型，并通过贪心算法给出一个近似最优解，计算总成本。

习题 8.2 某城市计划在所管辖的 6 个区域修建消防站，要求每个区域与最近消防站之间的行驶时间必须小于 15 分钟。表 8.4 给出了在该城市的各个区域之间行驶所需要的时间。请问该城市最少应修建多少个消防站以满足所有区域的救援需求？消防站应该设置在哪些区域？

表 8.4 习题 8.2 中在该城市的区域之间行驶时需要的时间

从	到

	区域 1	区域 2	区域 3	区域 4	区域 5	区域 6
区域 1	0	10	20	30	30	20
区域 2	10	0	25	35	20	10
区域 3	20	25	0	15	30	20
区域 4	30	35	15	0	15	25
区域 5	30	20	30	15	0	14
区域 6	20	10	20	25	14	0

根据表 8.4，能得到距离每个区域行驶时间小于 15 分钟的区域集合，见表 8.5

表 8.5 习题 8.2 中在给定区域 15 分钟行程内的区域

区域	在 15 分钟行程内的区域
1	1，2
2	1，2，6
3	3，4
4	3，4，5
5	4，5，6
6	2，5，6

习题 8.3 某省突发公共卫生事件，需在 8 个高风险县中设立临时医疗点。每个医疗点可服务的相邻县如下（若两县相邻，则医疗点可覆盖该县），设立医疗点的固定成本为 80 万元/个。目标是覆盖所有高风险县且总成本最低。

各县相邻关系如下：

县 1：与县 2、县 3 相邻

县 2：与县 1、县 4 相邻

县 3：与县 1、县 5、县 6 相邻

县 4：与县 2、县 7 相邻

县 5：与县 3、县 6、县 8 相邻

县 6：与县 3、县 5、县 7 相邻

县 7：与县 4、县 6、县 8 相邻

县 8：与县 5、县 7 相邻

请将此问题建模为集覆盖问题，并设计一种遗传算法求解（需说明编码方式、适应度函数及修复不可行解的策略）。

习题 8.4 如果部分元素可以“容忍未被覆盖”，即只要求覆盖全集的一部分（例如 80%），该如何修改集覆盖模型？若某些集合之间存在“冲突”（不能同时选用），如设点之间距离不能太近，如何处理？