第 6 章 装箱问题

在现代生产与物流体系中，如何高效地组织有限空间，以容纳形状各异、数量不定的物品，是一项基础却又充满挑战的任务。从仓库货架的自动布局，到电子元件的批量包装，再到云计算中任务资源的分区分配，这类问题本质上都涉及到：在空间受限的容器中，合理安排对象的位置与组合方式，使所需容器数量尽可能少。这类问题在理论上被统一描述为装箱问题，其核心任务是用尽可能少的“箱子”装下所有待分配的“物品”，而每个箱子的容量都是固定的。

与背包问题强调“选取哪些物品”不同，装箱问题更加关注“如何划分和放置”。装箱问题已经发展出多个变种模型，并成为工业工程、计算机科学及运筹优化中广泛研究的重要问题。接下来的内容将从其基本模型出发，逐步展开对算法策略、问题变体与实际应用的系统讲解。

6.1 装箱问题介绍

装箱问题（Bin Packing Problem, BPP）是组合优化领域的经典问题，其本质在于将若干大小不一的物品装入尽可能少的固定容量容器中，使每个容器不超载。该问题最早由 Garey 和 Johnson 于 1979 年证明为 NP-hard 问题，至今仍是工业界和学术界的研究热点。该问题不仅具有清晰的数学结构，更广泛存在于现实生产生活之中。例如，在电商仓储系统中，订单拣选完成后需要将多个商品打包装箱以发货。假设某仓库使用的标准快递箱容量为 1 立方米，而用户订单中的商品尺寸各不相同，如衣物、玩具、小型家电等，仓库管理系统必须根据这些商品的体积高效规划打包方案，以减少所用箱子数量、降低物流成本并提升打包效率。这一过程的抽象模型就是典型的装箱问题。

装箱问题广泛存在于工业物流、资源分配及日常生活场景中。例如，在物流运输中，需将若干货物装入尽可能少的集装箱；又如将多个文件存储到容量受限的磁盘时，如何最小化磁盘使用数量。

装箱问题可描述为：给定容量为 C 的许多箱子(假设箱子数量足够多)，以及n个物品，其重量分别为$w_1, w_2, \dots, w_n$（满足$0 < w_i < C$），需将所有物品装入箱

子，并使得所用箱子数目最少。

6.2 装箱问题数学模型

为建立整数线性规划模型，定义以下决策变量：

$$y_i = \begin{cases} 1, \text{第 i 个箱子被使用} \\ 0, \text{否则} \end{cases}, i = 1,2,\dots,n$$

$$x_{ij} = \begin{cases} 1, \text{第 j 个物品被装入第 i 个箱子} \\ 0, \text{否则} \end{cases}, i,j = 1,2,\dots,n,$$

目标是最小化总使用箱数，即：

$$\min z = \sum_{i=1}^{n} y_i ,$$

约束条件包括：

容量约束：每个箱子中物品总尺寸不超过其容量。若箱子i被使用（$y_i = 1$），则其装载物品的总尺寸需满足：

$$\sum_{j=1}^{n} w_j x_{ij} \le C y_i, \forall i = 1,2,\dots,n,$$

全覆盖约束：每个物品必须且仅被装入一个箱子：

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

变量取值约束：

$$y_i \in \{0,1\}, x_{ij} \in \{0,1\}, \forall i,j = 1,2,\dots,n,$$

该模型通过变量y_i控制箱子的使用状态，并通过x_{ij}确保物品分配的可行性。由于装箱问题是强 NP-困难问题，精确算法（如分枝定界法）的求解效率受限于问题规模，因此实际中常依赖启发式或近似算法获取可行解。

6.3 装箱问题求解

6.3.1 次优配合启发式方法

次优配合启发式方法（Next-Fit heuristic, NF）的基本逻辑是按物品的输入顺序逐一处理，确保每个物品能够被有效地放入箱子中。具体操作中，若当前箱子的剩余容量足够容纳当前物品，则直接将物品放入该箱子中。如果当前箱子无法容纳该物品，算法则会开启一个新的箱子，并将该新箱子作为当前的箱子来继续装载后续物品。

如下图所示，在整个过程中，物品是依照输入的顺序逐个处理的。举例来说，物品 1、物品 2 和物品 3 被依次放入箱子 A，。此时，物品 4 被处理时发现箱子 A 的剩余容量无法容纳物品 5，因此算法启动了一个新的箱子——箱子 B，并将物品 5 和物品 6 依次放入箱子 B。接着，当箱子 B 的剩余空间不足以容纳下一个物品时，算法再次启动了一个新的箱子——箱子 C，并将物品 7、物品 8 和物品 9 依次放入箱子 C。在最后，算法判断物品 10 是否能被放入箱子 C，如果箱子 C 的剩余容量不足以容纳物品 10，则会启动一个新的箱子，以此类推，直到所有物品都被成功装入箱子中。

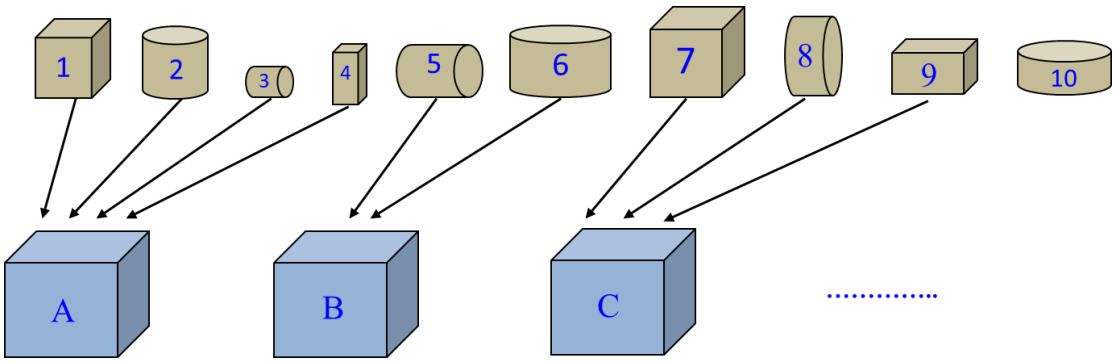


图 6.1 次优配合启发式方法演示图

该算法的目标是通过最小化所需箱子的数量来完成装箱任务。虽然该算法的解决方案并不一定是最优的，但它满足一个重要的约束条件：$NF(I) \leq 2z(I)$，其中$z(I)$代表问题的最优解。

此外，NF 算法的时间复杂度为$O(n)$。尽管物品的排列顺序可能影响最终的解，但无论如何，该算法通过顺序处理和动态开启新箱子的方式，有效地解决了

装箱问题。在实践中，虽然算法可能不总是给出最优解，但其计算速度和较小的解空间使得它成为一种非常实用的启发式方法。

6.3.2 优先配合启发式方法

优先配合启发式方法（First-Fit heuristic, FF）的基本逻辑是在处理物品时，优先尝试将当前物品放入已有的箱子中，而非立即开启新箱，从而尽可能提高箱子的利用率。该算法的处理流程如下：首先，物品按照输入顺序依次处理。对于每一个待放入的物品，算法从已开启的箱子中按照顺序依次查找，试图将其放入第一个容量足以容纳它的箱子中；若当前没有任何一个已有箱子可用，算法则立即开启一个新箱子，并将该物品放入其中。

如图所示，在整个过程中，物品被依照其编号顺序依次处理。首先处理物品 1、物品 2、物品 3 和物品 4，并依次将它们放入箱子 A 中，直到箱子 A 的剩余空间不足以继续容纳物品 5。此时，算法会开启一个新的箱子——箱子 B，物品 5 和物品 6 被放入箱子 B 中。若后续的物品无法放入现有的箱子中，算法则继续开启新的箱子并将物品放入。在处理物品 7 时，算法发现箱子 A 和箱子 B 均无法容纳物品 7，因此开启了新的箱子 C，并将物品 7 放入其中。接下来的物品 8 和物品 9 被依次放入箱子 A 和箱子 B 中，因为它们能够分别适应这两个箱子中剩余的空间。

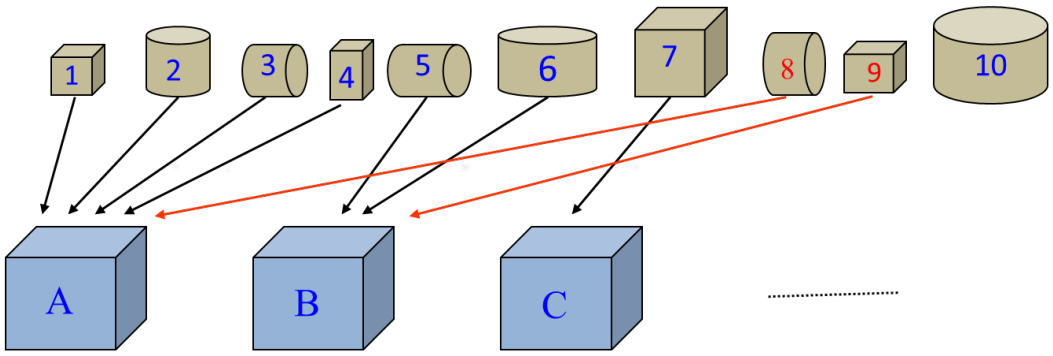


图 6.2 优先配合启发式方法演示图

相比于 NF 算法，FF 算法在处理过程中允许回溯检查已有所有箱子的容量状况，因此在箱子利用率上往往表现得更为紧凑。虽然其计算过程较 NF 算法略为复杂，但它在最坏情况下的性能也有理论保证。对于任意装箱问题I，FF 算法

所使用的箱子数量$FF(I)$满足如下不等式：

$$FF(I) \leq \frac{17}{10} z(I) + 2,$$

其中，$z(I)$表示问题的最优解所需的最少箱子数。由此可见，尽管 FF 算法并非总能产生最优结果，但其解在最坏情况下距离最优解的偏差是有严格上界的，这使得 FF 算法成为一种理论性能与实践表现均较为平衡的启发式方法。

此外，FF 算法的时间复杂度为$O(n \log n)$。这是由于在查找可容纳当前物品的第一个合适箱子时，若结合适当的数据结构（如平衡树、堆或索引表）进行管理，便可以加快搜索速度。

6.3.3 最佳配合启发式方法

最佳配合启发式方法（best-fit heuristic, BF）的基本逻辑是在处理物品时，每次将物品放入剩余空间最小的可行箱子中。如果没有箱子可以容纳当前物品，则开启一个新箱子。该算法的步骤如下：物品按照编号顺序进行处理，尝试将物品放入剩余空间最小的可行箱子中；若没有适合的箱子，则开启一个新的箱子。

如图所示，在整个过程中，物品被依照其编号顺序依次处理。首先处理物品 1、物品 2、物品 3 和物品 4，由于箱子 A 中有剩余空间，它们被依次放入箱子 A 中，当处理物品 5 时，箱子 A 空间不足，因此开启新的箱子——箱子 B，物品 5 和物品 6 被放入箱子 B 中。在处理物品 7 时，算法发现箱子 A 和箱子 B 的剩余空间均无法容纳物品 7，因此开启了新的箱子 C，并将物品 7 放入其中。此时箱子 8 的剩余空间最小，且能容纳物品 8，因此物品 8 被放入箱子 B 中。处理物品 9 时，箱子 A 时剩余空间最小的可行箱子，因此物品 9 被放入箱子 A 中，物品 10 同理。

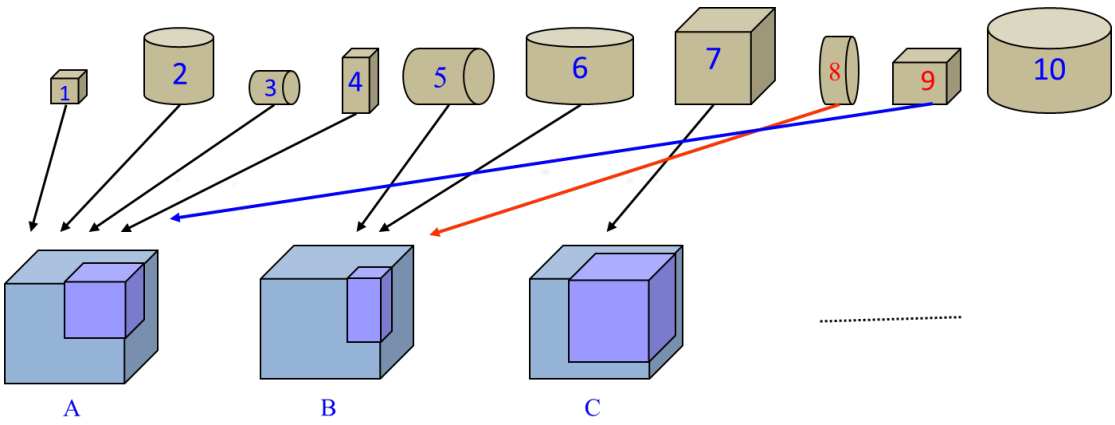


图 6.3 最佳配合启发式方法演示图

Best-fit（BF）算法通过优先选择剩余空间最小的箱子来提高空间利用率。它确保每次放置物品时，都能将物品放入当前箱子中剩余空间最小但足够容纳的箱子，从而减少新箱子的开启。BF 算法在最坏情况下的性能满足如下公式：

$$BF(I) \leq \frac{17}{10} z(I) + 2,$$

其中，$z(I)$为问题的最优解所需的箱子数。BF 算法的时间复杂度为$O(n \log n)$，其中n为物品数量。该复杂度来源于每次查找合适箱子时需要的排序和查找操作。

例题 6.1 物品重量列表：[4,2,3,5,1]，箱子容量 C=6。要求通过 NF（Next Fit）、FF（First Fit）、BF（Best Fit）三种算法来装入箱子，并找出每种方法所需的箱子数量。

解 (1).NF（Next Fit）方法：

在这种方法下，物品依次按顺序放入当前箱子中，如果当前箱子容量不够容纳下一个物品，就开启一个新箱子。

第一步，将物品 1、2 放入第一个箱子，剩余容量为 0；第二步，将物品 3 放入第二个箱子，但是第二个箱子剩余容量为 3，放不下物品 4，所以下一个箱子无法容纳它，因此必须打开一个新箱子；第三步，将物品 4、5 放入第三个箱子，剩余容量为 0；最终，共使用了 3 个箱子

(2).FF（First Fit）方法：

在 FF 方法中，物品按顺序依次尝试放入当前已有的箱子中，选择第一个能容纳该物品的箱子。

第一步，将物品 1 放入第一个箱子，剩余容量为 2；第二步，物品 2 放入第一个箱子，剩余容量为 0；第三步，物品 3 放入第二个箱子，剩余容量为 3；第四步，物品 4 无法放入第二个箱子（剩余容量为 3），于是放入第三个箱子，剩余容量为 1；第五步，物品 5 放入第一个容纳它的箱子，也就是第二个箱子。最终，共使用了 3 个箱子。

(3).BF（Best Fit）方法：

在 BF 方法中，我们尝试将物品放入最适合它的箱子中，也就是剩余容量最小、还能容纳当前物品的箱子。

第一步，将物品 1 放入第一个箱子，剩余容量为 2；第二步，物品 2 放入第一个箱子，剩余容量为 0；第三步，物品 3 放入第二个箱子，剩余容量为 3；第四步，物品 4 无法放入第二个箱子，于是放入第三个箱子，剩余容量为 1；第五步，此时第二个箱子剩余容量为 3，第三个箱子剩余容量为 1，于是物品 5 放入剩余容量最小的可行箱子，也就是第三个箱子中。最终，共使用了 3 个箱子。

6.3.4 遗传算法求解

（1）编码策略

在遗传算法中，编码策略是至关重要的，它直接影响到解的可行性和搜索效率。对于装箱问题，选择合适的编码方式有助于提高算法的表现。

Ⅰ.基于箱子的表示

基于箱子的表示（Bin-Based Representation）是一种常见的遗传算法编码策略，主要应用于装箱问题中。在这种编码方式中，每个基因的位置代表一个物品，而基因的值则表示该物品所属的箱子编号。举例来说，假设有物品编号 1 至 6，染色体可能为[1,1,2,3,2,3]，这意味着物品 1 和物品 2 被放入箱子 1，物品 3 和物品 5 被放入箱子 2，物品 4 和物品 6 被放入箱子 3。这样的编码方式直观地将物品与其所属的箱子进行了关联，便于在遗传算法中进行操作。

该方法的主要优点在于其结构的简单性，使得可以直接应用标准的遗传算子，如单点交叉和位变异。这使得遗传算法的实现变得更加简便，且具有较好的执行效率。然而，基于箱子的表示也存在一些缺点。首先是冗余性问题，即同一解可能会对应多个不同的染色体表示。例如，染色体[1,1,2,2]和[2,2,1,1]实际上表示的是相同的解，但它们的编码方式却不同，这样的冗余性增加了搜索空间的复杂度。其次，由于在交叉或变异操作过程中，可能会生成超重箱子的情况，因此该编码方法在某些情况下可能会导致不可行解的出现。换句话说，某些箱子的物品总重量可能超过了其容量限制，导致解不满足实际约束。

为了解决这些问题，通常可以采用改进策略。例如，引入修复算子（Repair Operator），该算子可以在交叉或变异操作后对生成的解进行检查，及时发现并调整超重的箱子，确保每个箱子所装物品的总重量不超过其容量限制。通过这种修复机制，遗传算法能够在搜索过程中维持解的可行性，避免生成不符合问题约束的解，从而提高算法的稳定性和效率。

Ⅱ.基于物品的表示

基于物品的表示（Object-Based Representation）是一种在遗传算法中应用的编码策略，其主要思想是通过对物品的排列顺序进行编码，并利用解码器将该排列转化为对应的装箱解。与基于箱子的表示不同，这种编码方式并不直接记录物品属于哪个箱子，而是将物品按照某种顺序排列。通过解码过程，不同的算法将物品放入箱子中，最终得到问题的解。举例来说，假设物品编号为 1 至 6，染色体可以表示为[1，2，3|4，5|6]，表示物品 1 到 3 放入一个箱子，物品 4 和 5 放入第二个箱子，物品 6 放入第三个箱子。

该方法的一个重要优点是与基于箱子的表示不同，它从不产生不可行解。由于物品的排列顺序是通过解码器进行逐步转换的，因此无论如何排列，生成的解始终符合约束条件，不会出现箱子超重或容量违规的问题，这为解决装箱问题提供了更为稳定和可靠的基础。

然而，基于物品的表示也存在一定的缺点。最主要的问题在于编码的冗余性，这种冗余性随着问题规模的增加而急剧上升，导致遗传算法的效率受到显著影响。具体来说，在这种表示方法中，同一箱子内物品的任意排列都会生成不同的染色

体表示，而这些不同的染色体实际上对应的是相同的解。随着问题规模的增大，这种冗余性将指数性地增加，造成大量重复的计算，从而大幅度增加了搜索空间的复杂性，降低了遗传算法的搜索效率。例如，若有 3 个箱子，分别装物品 1、2、3、物品 4 和 5、物品 6，不同的排列顺序会生成大量不同的染色体，而这些染色体其实表示的是相同的解。由于冗余度的存在，遗传算法的搜索能力可能会受到严重的限制，因此需要采取额外的措施来降低这种冗余，优化算法的表现。

III.基于群体的表示

上述基于物品和箱子的表示方法存在局限性，原因在于它们都是面向单个物品的，而装箱问题实际上是一个群体问题。在这种问题中，费用函数依赖于箱子中物品的组合和群体特征，而非单独物品的分布。因此，基于物品或箱子的编码方法无法有效地优化装箱问题的费用函数。

对于装箱问题，更好的编码方式应该能够同时处理两个关键部分。首先，它需要提供物品与箱子之间的关系，即明确每个物品属于哪个箱子。其次，它还需要对使用的箱子进行编码，因为每个装有不同物品的箱子实际上是不同的群体。为此，Falkenauer 提出了“基于群体”的表示方法。在这种方法中，每个基因代表一个箱子，而整个染色体则由多个基因组成，每个基因内包含该箱子中所有物品的信息。这样的表示方式能够更好地反映装箱问题中的群体特性，从而有效地优化费用函数。

基于群体的表示（Group-Based Representation）是一种更为适合装箱问题的编码方法。在这种方法中，每个基因代表一个箱子（群体），并包含该箱子内的物品列表。通过这种结构，染色体能够直接反映装箱的实际情况。例如，染色体[[1,3],[2,4]]表示两个箱子，分别装入物品 1 和 3，以及物品 2 和 4。这种编码方式能够清晰地表达每个箱子的内容，适应装箱问题的群体优化需求。

基于群体的表示方法的优点在于它能够直接反映装箱结构，避免了冗余编码的产生，使得遗传算法操作变得更加简洁且高效。同时，该方法还保留了父代的群体特征，因为交叉和变异操作是在群体级别进行的，这样可以避免在遗传操作中破坏已有的优良分组，提高了搜索过程中的稳定性和效果。

然而，这种表示方法也存在一些缺点。首先，由于染色体的长度是可变的，这就需要在遗传算法中设计特殊的算子来处理这种变长的情况，例如变长交叉。其次，基于群体的表示方式在实现上较为复杂，需要处理群体合并和冲突解决等问题，这增加了算法的实现难度。因此，尽管基于群体的表示具有较好的优化潜力，但其实现和操作的复杂性也是需要考虑的挑战。

（2）适值函数设计

在遗传算法中，适值函数是用来评价解优劣的核心指标，决定了遗传算法在搜索过程中如何选择和优先保留较优的解。在装箱问题中，适值函数的设计尤为重要，因为它需要同时考虑最小化箱子数量和最大化空间利用率这两个相互矛盾的目标。具体而言，最小化箱子数量可以帮助减少运输或存储成本，而最大化空间利用率则可以有效利用每个箱子的空间，减少空间浪费。由于这两个目标通常是相互制约的，减少箱子数量可能导致部分箱子未能充分填满，而追求高填充率又可能需要增加箱子数量，因此，适值函数的设计需要在这两个目标之间找到合适的平衡。

为此，Falkenauer 和 Delchambre 对装箱问题提出了一种适值函数的数学形式。该函数的形式为：

$$f_{\mathrm{BPP}} = \frac{\sum_{i=1}^{N} \left(\frac{F_i}{C}\right)^k}{N},$$

其中，N代表解中使用的箱子数量，F_i是第i个箱子中物品的总重量（即箱子的填充程度），C是箱子的容量，而k是一个调节因子$(\mathrm{k} > 1)$，用于控制对高填充率箱子的偏好程度。通常情况下，经验值取$k = 2$。通过这个适值函数，遗传算法的目标是最大化f_{BPP}的值。具体而言，当使用的箱子数量N较少时，f_{BPP}值较大；同样，当每个箱子中的物品总重量F_i接近箱子的容量C时，f_{BPP}值也会较大。这样，适值函数能够有效地平衡箱子数量和空间利用率之间的矛盾，指导遗传算法朝着既减少箱子数量又提高空间利用率的方向进行优化。

（3）遗传运算

Ⅰ.交叉操作

在遗传算法中，交叉操作是通过结合两个父代个体的优良特征，生成新的子代个体，从而推动算法寻找最优解的过程。在装箱问题中，交叉操作需要特别关注解的可行性和群体结构，因为这种问题涉及将一组物品合理地放入多个箱子中，因此在生成新的个体时，必须保证解的合理性，即所有物品必须能够放入箱子，并且没有重复分配。

如下图所示，交叉操作的基本流程可以从选择父代开始，通常采用锦标赛选择或轮盘赌选择的方式，从当前种群中选择两个父代个体。接下来，进行随机切割操作，在父代 1 中随机选择一个切割点，将切割点之前的部分箱子组复制到子代中。然后，在父代 2 中选择剩余的部分，将其箱子组按顺序加入子代的对应位置，并且去除重复的物品。此时，已经完成了初步的子代生成，但此时可能有些物品未能被纳入子代。

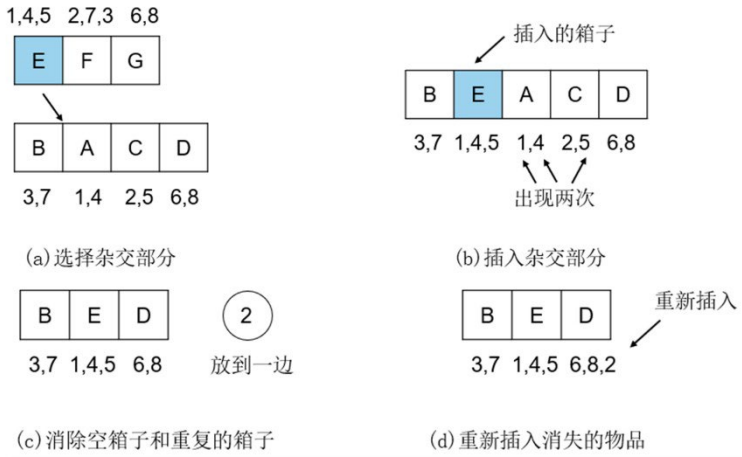


图 6.4 遗传算法求解装箱问题时交叉操作演示图

为了补全缺失的物品，算法将依次尝试将未包含的物品放入现有的箱子中。如果发现某个物品无法放入现有箱子中，则需要开启一个新箱子来容纳该物品。通过这样的补全过程，可以确保子代个体符合装箱问题的约束条件，且具备较高的适应度。

交叉操作的特点在于，首先，它能够保留父代个体的优良特征。具体而言，子代继承了父代 1 的部分箱子组和父代 2 的部分箱子组，因此，子代有较高的潜

力继承并发扬父代的优势。其次，通过去除重复物品的策略，交叉操作能够避免冗余，确保每个物品仅被分配到一个箱子里，这也是确保解合法性的一个关键因素。最后，交叉操作具备较高的灵活性，能够生成多样化的子代，同时保持解的可行性。因此，这种操作方式适应性强，能够有效地探索解空间，提高遗传算法的搜索效率。

Ⅱ.变异操作

在装箱问题的求解中，变异操作的应用尤为重要，它能够在解空间中探索更多的可能性，提高算法的搜索能力，避免陷入局部最优解。

变异的基本流程首先是选择变异的类型，常见的变异类型包括启用一个新的箱子或消除一个已经使用的箱子。

启用一个新的箱子是指将一个新箱子插入到某个位置，剩余的操作类似于交叉。删除一个已经使用的箱子是指对被删除的箱子中的物品采用 FF 或 FFD 的方法，将其重新放入箱子。

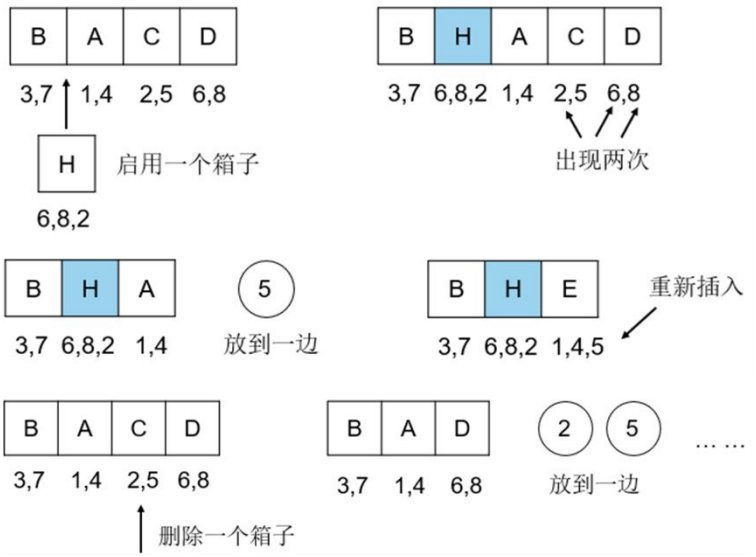


图 6.5 变异操作演示图

在进行变异操作时，可能会出现解的不可行性问题，即变异后的解可能会导致某些物品没有被正确放置。为了解决这一问题，可以采用 FF（First Fit）等启发式方法。通过这些方法，可以按照一定的顺序将缺失的物品重新放入箱子中，从而保证变异后的解是可行的。

变异操作的特点主要体现在几个方面。首先，它通过拆分或合并箱子的方式，能够引入多样性，生成新的解结构，拓宽搜索空间。其次，变异操作还具有修复机制，可以确保变异后的解仍然满足容量约束，不会出现不可行解。最后，变异操作具有局部优化的效果，通过重新分配物品，能够优化箱子的填充率，从而进一步提升解的质量。

（4）种群初始化

种群初始化在遗传算法中是一个至关重要的步骤，因为初始种群的质量直接影响到算法的搜索效率和最终解的质量。在装箱问题的求解中，初始化种群的策略有着重要作用。常用的初始化方法大体可以分为两类：随机初始化和启发式初始化。

Ⅰ.随机初始化

随机初始化是一种较为简单的初始化方法，其操作步骤如下：首先，随机生成若干可行的箱子组合，每个箱子中的物品数量是随机确定的，但总重量必须不超过箱子的容量限制。如果生成的箱子组合未能覆盖所有物品，则会将剩余的物品用新的箱子进行包装。通过不断重复这些步骤，直到达到指定的种群规模为止。

随机初始化的优点是操作简便，生成速度较快，且种群的多样性较高，可以有效避免初期解的过度集中。然而，随机初始化也有其缺点，最明显的是它可能会产生大量的低质量解，这些解往往需要在后续的进化过程中进行大量的优化和改进。

Ⅱ.启发式初始化

启发式初始化则更加注重初始解的质量，能够在一定程度上加速算法的收敛过程。其操作步骤如下：首先，生成物品的随机排列序列，然后利用经典的启发式算法，如 First Fit（FF）、Best Fit（BF），将物品装入箱子中。接着，使用特定的编码方式将装箱方案表示为染色体，并基于群体表示进行初始化。通过重复这一过程，直到种群规模满足要求。

启发式初始化的优点是生成的初始解质量较高，从而能够加速遗传算法的收敛过程。然而，这种方法也存在一定的局限性，其主要缺点是可能会限制种群的

多样性，从而增加早熟收敛的风险。因此，启发式初始化通常需要与变异操作相结合，以保证算法的多样性，避免过早陷入局部最优解。

总的来说，选择适合的初始化方法对于遗传算法的性能至关重要。随机初始化适合探索广泛的解空间，而启发式初始化则能够提供较高质量的初始解，促进快速收敛。在实际应用中，这两种方法可以根据具体问题的需求进行灵活组合，以获得更好的优化效果。

（5）选择策略

在遗传算法应用于装箱问题的求解过程中，通过适当的选择策略，遗传算法能够有效地从当前种群中筛选出优质的个体，使得问题的求解更为高效，最终获得较优的装箱方案。选择操作通过适值函数来筛选适应度较高的个体进入下一代，从而推动算法向更优解的方向发展。

在装箱问题的遗传算法中，常见的选择策略包括按比例选择和基于次序的选择。

Ⅰ.按比例选择

第一种模式是按比例的选择，这类选择策略根据种群中个体的适应度进行选择。具体来说，在这些方法中，个体的适应度越高，其被选中的概率就越大。常见的基于适应度的选择方法包括：轮盘赌选择、随机通道选择和随机余数选择。

轮盘赌选择是一种根据适应度比例分配选择概率的经典方法。在这种方法中，每个个体的适应度值决定了其在选择过程中所占的比例，适应度高的个体在“轮盘”中占据更大的区域，因此有更大的概率被选择。随机通道选择是在轮盘赌选择的基础上进行改进，通过多点选择来减少选择中的偏差，从而提高算法的稳定性。随机余数选择则进一步优化了选择过程，它通过余数的方式来分配选择概率，以达到更加均匀且随机的选择效果。

Ⅱ.基于次序的选择

第二种选择模式是基于次序的选择，即通过对种群中的个体进行排名，再根据排名结果进行选择。该方法主要通过对个体的适应度进行排序，并根据排序结

果来决定每个个体的选择概率。这类方法包括：竞争选择、 $\mu+\lambda$ 选择、截断选择和线性排序选择。

竞争选择是一种通过随机选择若干个个体进行“比赛”的方法，选择其中适应度最好的个体进入下一代。这种方法简单有效，尤其适用于较大规模的种群。$\mu+\lambda$ 选择则是将父代个体和子代个体一同进行竞争，并保留适应度较高的个体，常用于保持种群的多样性并防止过早收敛。截断选择通过对种群中的个体按适应度值进行排序，保留最优的前若干个个体，淘汰适应度较低的个体，从而加速进化过程。线性排序选择通过根据个体的适应度进行排序，为每个个体分配一个选择概率，这种方法避免了轮盘赌选择中由于适应度差异过大可能导致的选择偏差。

这些选择策略对遗传算法的优化效果产生了重要影响。通过合理的选择策略，能够有效提高种群的质量，促进算法的收敛，同时避免过早收敛的问题。

例题 6.2 考虑一个装箱问题，给定 5 个物品，其重量分别为[4,5,7,3,6]，每个箱子的容量为 10。我们要求通过遗传算法来求解，找到最少箱子数的装箱方案。

解 算法思路：在本问题中，我们通过遗传算法来进行求解。遗传算法的核心思路包括通过模拟自然选择和进化过程，通过选择、交叉、变异等操作来逐步找到最优解。下面是本算法的主要步骤：

编码方式：采用基于箱子的表示方式来编码解。每个染色体由多个箱子组成，每个箱子包含一组物品。这种表示方式能够有效地反映装箱问题的群体结构

适值函数：适值函数的设计基于 Falkenauer 公式，主要考虑两方面：一是箱子数量要尽可能少，二是每个箱子的填充率要高。通过适值函数来评估个体解的质量，以此引导算法朝着更优的解进化。

交叉操作：交叉操作通过交换父代的箱子群体并去除重复物品来产生新的子代。该操作的目标是保留父代的优良特征，同时引入新的解空间。

变异操作：变异操作是随机增加或删除一个箱子，并重新分配物品。为了确保解的合法性，使用 First Fit（FF）算法来修复变异后的解。

选择策略：采用轮盘赌选择（Roulette Wheel Selection）作为选择策略。通过

这种方式，适应度较高的个体将有更大的概率被选中进行繁殖。

6.4 装箱问题的应用

装箱问题在实际管理中的应用广泛而深入，尤其在资源受限的环境下，如何在效率与成本之间取得平衡，已成为企业运营过程中不可回避的关键挑战。在物流运输领域，这一问题典型地表现为集装箱的装载优化。企业需在满足载重与体积等多重约束的前提下，尽可能以最少的集装箱完成货物配送任务，从而有效降低运输成本，提升仓储周转效率和配送的及时性。在制造业中，装箱问题转化为原材料的裁剪与利用优化。企业往往需要从整块材料中切割出尽可能多的合格部件，同时最大限度地减少边角废料，这直接关系到材料利用率和生产成本控制，进而影响利润水平。

在云计算领域，装箱问题则以资源分配的形式体现。在多租户的运行环境下，云服务提供商需将不同规格的虚拟机合理部署到有限的物理服务器中，既要确保服务器资源的高效利用，又要避免计算资源碎片化，同时满足各租户之间的性能隔离与服务质量要求。而在零售仓储管理中，装箱问题被具体化为空间利用的优化问题。如何在有限的货架空间中，结合商品的体积、重量以及销售周转率等因素，设计出既能提升存储密度，又便于拣选操作的布局方案，是仓库运营效率的重要决定因素。

这些管理实践尽管行业背景各异，但其核心问题具有高度的一致性：企业均需在明确的资源约束条件下，通过合理的数学建模与优化算法，在降低运营成本的同时提高系统效率，实现资源配置的最优解。这正是装箱问题之所以在各行业中具有持续研究与实践价值的根本原因。

6.5 二维装箱问题

6.5.1 问题描述

二维装箱问题是一个经典的组合优化问题，它是指将n个矩形物品装入m个矩形箱子中每个物品有长度$l_j > 0$和宽度$w_j > 0$，每个箱子有长度$L_i > 0$和宽度$W_i > 0$。该问题的核心目标是找到一种最优的装箱方案，使得在同一个箱子内的

物品不重叠，同时箱子的使用数量尽可能少。

在实际问题中，通常假设所有箱子的长度和宽度相同，也就是说，$L_i = L$，$W_i = W$，$\forall i = 1,2,\dots,m$。这种设定使得问题的求解变得更加规范和统一，能够减少计算的复杂度。物品在装箱时，需要保证物品的边缘与箱子的边缘平行，因此物品不可以随意旋转。

在某些特殊情况下，二维装箱问题可以被简化为一维装箱问题。例如，当每个物品的宽度恰好等于箱子的宽度W时，问题就可以转化为一个一维的装箱问题，其中物品只能沿箱子的长度方向排列，从而大大简化了问题的复杂度。

6.5.2 数学模型

在这个二维装箱问题的数学模型中，首先假设每个箱子被放置在一个直角坐标系的第一象限内，箱子的长边与X轴平行，宽边与Y轴平行，并且箱子的左下角与坐标原点重合。接下来，定义了一些关键的决策变量来描述装箱过程。

接下来，定义了一些关键的决策变量来描述装箱过程。

$$y_i = \begin{cases} 1, \text{第 i 个箱子被使用} \\ 0, \text{否则} \end{cases}, i = 1,2,\dots,m$$

$$x_{ij} = \begin{cases} 1, \text{第 j 个物品被装入第 i 个箱子} \\ 0, \text{否则} \end{cases}, i = 1,2,\dots,m, j = 1,2,\dots,n$$

$$p_{aj} = \begin{cases} 1, \text{物品} j \text{的长平行于} X \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$p_{bj} = \begin{cases} 1, \text{物品} j \text{的长平行于} Y \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$q_{aj} = \begin{cases} 1, \text{物品} j \text{的宽平行于} X \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$q_{bj} = \begin{cases} 1, \text{物品} j \text{的宽平行于} Y \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$c_{jk} = \begin{cases} 1, \text{物品} j \text{在物品} k \text{的左面} (a_j < a_k) \\ 0, \text{否则} \end{cases}, j,k = 1,2,\dots,n$$

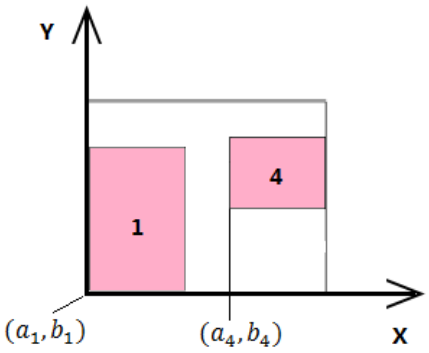
$$d_{jk} = \begin{cases} 1, \text{物品} j \text{在物品} k \text{的右面} (a_j > a_k) \\ 0, \text{否则} \end{cases}, j,k = 1,2,\dots,n$$

$$e_{jk}=\begin{cases}1,\text{物品}j\text{在物品}k\text{的下面}(b_j<b_k)\\0,\text{否则}\end{cases},j,k=1,2,\dots,n$$

$$f_{jk}=\begin{cases}1,\text{物品}j\text{在物品}k\text{的上面}(b_j>b_k)\\0,\text{否则}\end{cases},j,k=1,2,\dots,n,$$

(a_j,b_j)为连续变量，表示物品j的左下角的坐标，也叫物品j的起始坐标。

为便于理解，举例说明以上决策变量如何描述装箱过程，如图所示：



$$p_{a1}=0,p_{b1}=1,q_{a1}=1,q_{b1}=0$$

$$p_{a4}=1,p_{b4}=0,q_{a4}=0,q_{b4}=1$$

$$c_{14}=1,d_{14}=0$$

$$e_{14}=1,f_{14}=0$$

图 6.6 二维装箱过程演示图

该问题的目标是最小化所使用箱子的数量，目标函数的数学表达式为：

$$\min f(y)=\sum_{i=1}^{m}y_i,$$

约束条件：

每个物品必须放入一个箱子：

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

箱子容量限制：

$$\sum_{j=1}^{n}x_{ij}\le My_i,\forall i\in I=\{1,2,\dots,m\},$$

物品在箱子内不重叠：

$$a_j + l_j p_{aj} + w_j q_{aj} \le a_k + (1 - c_{jk})M, \forall j,k \in N, i \in I$$

$$a_k + l_k p_{ak} + w_k q_{ak} \le a_j + (1 - d_{jk})M, \forall j,k \in N, i \in I$$

$$b_j + l_j p_{bj} + w_j q_{bj} \le b_k + (1 - e_{jk})M, \forall j,k \in N, i \in I$$

$$b_k + l_k p_{bk} + w_k q_{bk} \le b_j + (1 - f_{jk})M, \forall j,k \in N, i \in I,$$

物品必须完全在箱子内：

$$a_j + l_j p_{aj} + w_j q_{aj} \le L_i + (1 - x_{ij})M, \forall j \in N, i \in I,$$

$$b_j + l_j p_{bj} + w_j q_{bj} \le W_i + (1 - x_{ij})M, \forall j \in N, i \in I,$$

方向约束：

$$p_{aj} + p_{bj} = 1, \forall j \in N$$

$$q_{aj} + q_{bj} = 1, \forall j \in N$$

$$p_{aj} + q_{aj} = 1, \forall j \in N$$

$$p_{bj} + q_{bj} = 1, \forall j \in N,$$

物品之间的相对位置约束：

$$c_{jk} + d_{jk} + e_{jk} + f_{jk} \ge x_{ij} + x_{ik} - 1, \forall j,k \in N, i \in I,$$

0-1 约束：

$$p_{aj}, p_{bj}, q_{aj}, q_{bj}, c_{jk}, d_{jk}, e_{jk}, f_{jk} \in \{0,1\}, \forall j,k \in N, i \in I.$$

6.6.3 启发式算法

尽管二维装箱问题相较一维问题更为复杂，但在实际场景中，如物流打包、货架陈列、裁剪排样等，都必须面对物品在二维空间中如何高效排布的问题。为此，许多学者设计出多种规则明确、计算高效的启发式策略，用于快速获得近似最优解。本节将重点介绍两种在二维装箱中广泛应用的启发式方法：Shelf Packing（架式装箱）算法与 Normal Position Packing（标准位置装箱）算法。

1.Shelf Packing（SP）算法

在 Shelf Packing 算法中，物品被按照一定的规则摆放在装入箱子的架子（shelf）上，类似于将物品放置在一排排的架子中。每个架子的宽度与箱子的宽度相等，而每个架子的高度由该架子上第一个物品的高度决定。算法的核心目标是优化物品在箱子中的排列，减少空间浪费。该算法通常分为两大步骤：首先将物品摆放到架子上，然后将这些架子装入箱子中。

在第一步中，假设存在n个物品，箱子的宽度为W，架子的高度待定。所有物品按照长度递减的顺序排列（通常长边为长度）。然后，物品依次被放置到架子上，优先使用最佳配合方法（BF）。如果当前待装物品不能放入任何一个已经使用的架子中，且这些架子的高度已经固定，则需要新开一个空架子，并将该架子的高度设定为当前物品的高度。否则，选择一个已使用的架子，使得该架子剩余的宽度最小，并能容纳当前物品。此步骤的目标是尽量减少物品在架子上的摆放空间，确保每个物品能以最合适的方式被放置，从而节省空间。

在第二步中，所有物品已摆放好的架子将依次装入箱子。此时，实际上是在解一个一维装箱问题，因为每个架子的宽度已被固定为箱子的宽度W。我们将已装满物品的架子看作一维物品，接着通过一维装箱问题的求解算法将这些架子装入箱子中，确保箱子空间的使用最为高效。

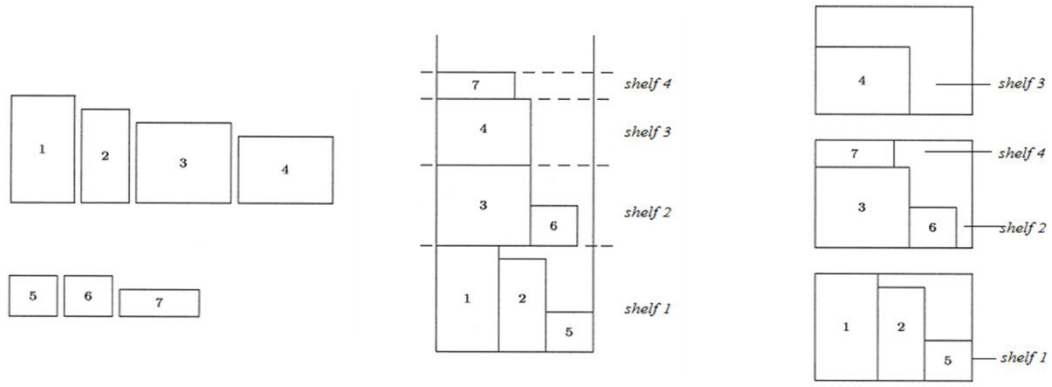


图 6.7 Shelf Packing 算法演示图

例题 6.3 已知箱子尺寸其高度 H=5，宽度 W=8。现有一组待装箱物品，它们的尺寸分别为(6×4)、(5×3)、(4×5)、(3×2)、(2×2)。请使用 Shelf Packing 算法，

将这些物品合理地装入箱子，并给出详细的操作步骤。

解 Shelf Packing 算法的第一步通常是对物品进行排序。这里我们按照物品的长度进行降序排列。排序后物品列表为(6×4)、(5×4)、(5×3)、(3×2)、(2×2)，满足降序要求。这一步的目的是为后续生成架子的过程提供一个有序的物品处理顺序，有助于更高效地利用箱子空间。

物品 1（尺寸为 6×4）：

由于此时还没有已有的架子，我们需要新建一个架子，记为架子 1。根据物品 1 的尺寸，架子 1 的高度设定为物品 1 的宽度，即 4。而架子 1 的剩余宽度则为箱子的宽度减去物品 1 的长度，也就是 8−6=2。这样，架子 1 就初步建立起来，等待后续物品的放置。

物品 2（尺寸为 5×4）：

接下来，我们检查架子 1 的剩余宽度是否能够容纳物品 2。经比较，架子 1 的剩余宽度 2 小于物品 2 的长度 5，物品 2 无法放入架子 1 中，需要新建架子 2，所以物品 2 需放入放入架子 2 中。放入后，架子 2 的剩余宽度变为 8-5=3，架子 2 的高度设定为物品 2 的宽度 4。

物品 3（尺寸为 5×3）：

继续查看架子 1 和架子 2 的剩余宽度（架子 1 剩余 2，架子 2 剩余 3），发现物品 3 无法放入架子 1 和 2 中。于是，我们需要新建一个架子，即架子 3。架子 3 的高度设定为物品 3 的宽度 3，剩余宽度为 8−5=3。

物品 4（尺寸为 3×2）：

先看架子 1 的剩余宽度 2，显然小于物品 4 的长度 3，物品 4 不能放入架子 1。再检查架子 2 的剩余宽度 3，它等于物品 4 的长度 3，所以物品 4 可以放入架子 2，且宽度不超过架子 2 的高度，放入后架子 2 的剩余宽度变为 0。

物品 5（尺寸为 2×2）：

分别对架子 1 和架子 2 的剩余宽度进行判断，发现架子 1 的剩余宽度 2 等于物品 5 的长度 2。因此，物品 5 可以放入架子 1，且宽度不超过架子 1 的高度。

经过上述步骤，我们得到了三个架子，分别是架子 1（高度为 4）、架子 2（高度为 4）、架子 3（高度为 3）。此时，我们要将这些架子装入高度为 5 的箱子中。最终，我们使用了 3 个箱子完成了所有架子的装载。

通过 Shelf Packing 算法进行装箱操作，我们得到了如下结果：箱子 1 中装有物品(6×4)、(2×2)，箱子 2 中装有物品(5×4)、(3×2)，箱子 3 中装有物品(5×3)。

2.Normal Position Packing （NPP）算法

Normal Position Packing（NPP）算法核心思想是通过将物品放置在特定的“Normal Position”来优化箱子的空间利用率。所谓“Normal Position”，是指物品的左边界必须紧贴箱子的左壁或已放置物品的右边界，同时物品的下边界必须紧贴箱子的下壁或已放置物品的上边界。这种放置方式能够有效减少空间浪费，并提高装箱效率。

首先，输入物品的尺寸（宽度和高度）以及箱子的尺寸。为了优化装箱顺序，算法会按照物品的面积从大到小进行排序。面积较大的物品优先处理，以减少空间碎片。

对于每一个物品，算法会尝试将其放入已使用的箱子中。如果当前物品无法放入任何一个已使用的箱子，则初始化一个新箱子，并将该物品放置在新箱子的左下角。如果物品可以放入多个箱子，算法会为每个箱子中的“Normal Position”打分，选择得分最高的位置进行放置。

对于能够放入已存在箱子的物品，会使用打分机制来选择合适的“normal position”进行放置。打分机制是根据物品的边界与箱子或其他物品的边界接触的情况来打分。接触最多的（或者接触率最高的）“normal position”得分最高。如果有多个“normal position”得分相同，则选择剩余面积最小的箱子。

例题 6.4 已知箱子尺寸为 6×6。现有一组待装箱的物品，它们的尺寸分别为物品 A（3×3）、物品 B（2×2）、物品 C（4×2）、物品 D（1×5）和物品 E（2×3）。请使用 Normal Position Packing 算法，将这些物品装入箱子，并详细展示整个求解过程。

解 首先根据物品的面积进行排序，物品 A 的面积最大为 9，物品 C 的面积

为 8，物品 E 的面积为 6，物品 D 的面积为 5，物品 B 的面积最小，为 4。因此，按照面积递减的顺序，物品的排列为物品 A、物品 C、物品 E、物品 D、物品 B。

接下来，我们将开始按照算法将这些物品逐一放入箱子中。首先，物品 A 的面积最大，我们将其放置在箱子的左下角，左边界紧贴箱子的左壁，下边界紧贴箱子的下壁，完全占据了箱子的左下角区域，位置为（0,0）到（3,3）。接着，我们放置物品 C，物品 C 的尺寸为 4×2，无法放入第一个箱子的空余空间中，因此我们初始化了一个新的箱子，将物品 C 放置在新箱子的左下角，占据了（0,0）到（4,2）的区域。

接下来是物品 E，物品 E 的尺寸为 2×3，可以放入第一个箱子的空余空间。根据打分机制，物品 E 应该放置在物品 A 的右侧，所以下边界仍然贴着箱子的下壁，左边界与物品 A 的右边界对齐，占据了（3,0）到（5,3）的区域。物品 D 的尺寸为 1×5，相对较长，它可以放置在第一个箱子的剩余空余空间中，具体位置是在物品 E 的右方，占据了（5,0）到（6,5）的区域。

最后，物品 B 的尺寸为 2×2，比较小。物品 B 可以放在第一个箱子的剩余空余空间里，它的放置位置是在物品 A 的上方，占据了（0,3）到（2,5）的区域。通过以上步骤，所有的物品都被顺利放入了箱子中。第一个箱子包含了物品 A、物品 E、物品 D 和物品 B，而第二个箱子则装了物品 C。

6.6 三维装箱问题

6.6.1 问题描述

给定m个长方体箱子和n个长方体物品，每个物品有长度（$l_j > 0$）、宽度（$w_j > 0$）和高度（$h_j > 0$），每个箱子也有长度（$L_i > 0$）、宽度（$W_i > 0$）和高度（$H_i > 0$）。问题是：寻找最优的方案将物品分配到箱子中，使得装入同一个箱子的所有物品不会相互重叠，而使用的箱子数量最少。

通常，所有箱子有相同的长宽高，$L_i = L, W_i = W, H_i = H, \forall i = 1,2,\dots,m$。物品的边缘应该平行于它所在的箱子的边缘。在某些情况下，我们要求物品不可以旋转。显然，如果每一个物品的高度都等于箱子的高度H，那么三维装箱问题就

被简化为二维装箱问题。

假设每一个箱子都被放置在一个三维直角坐标系的第一象限，并且箱子的长平行于X轴，宽平行于Y轴，高平行于Z轴，箱子的下-左-前角与坐标原点重合。

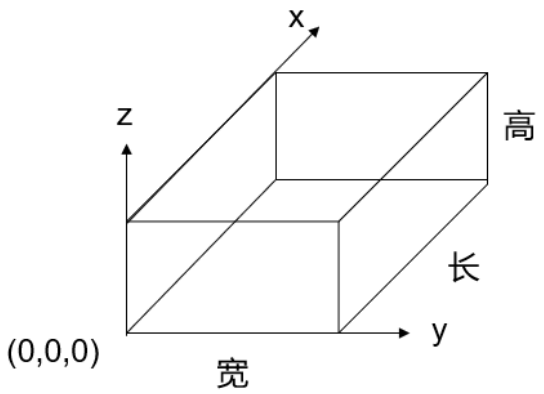


图 6.8 三维装箱坐标示意图

6.6.2 模型描述

数学模型中使用的变量：

$$y_i = \begin{cases} 1, \text{第 i 个箱子被使用} \\ 0, \text{否则} \end{cases}, i = 1,2,\dots,m$$

$$x_{ij} = \begin{cases} 1, \text{第 j 个物品被装入第 i 个箱子} \\ 0, \text{否则} \end{cases}, i = 1,2,\dots,m, j = 1,2,\dots,n$$

$$p_{\alpha j} = \begin{cases} 1, \text{物品} j \text{的长平行于} X \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$p_{\beta j} = \begin{cases} 1, \text{物品} j \text{的长平行于} Y \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$p_{\gamma j} = \begin{cases} 1, \text{物品} j \text{的长平行于} Z \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$q_{\alpha j} = \begin{cases} 1, \text{物品} j \text{的宽平行于} X \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$q_{\beta j} = \begin{cases} 1, \text{物品} j \text{的宽平行于} Y \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$q_{\gamma j} = \begin{cases} 1, \text{物品} j \text{的宽平行于} Z \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$r_{\alpha j} = \begin{cases} 1, \text{物品} j \text{的高平行于} X \text{轴} \\ 0, \text{否则} \end{cases}, j = 1,2,\dots,n$$

$$r_{\beta j}=\begin{cases}1,\text{物品}j\text{的高平行于}Y\text{轴}\\0,\text{否则}\end{cases},j=1,2,\dots,n$$

$$r_{\gamma j}=\begin{cases}1,\text{物品}j\text{的高平行于}Z\text{轴}\\0,\text{否则}\end{cases},j=1,2,\dots,n$$

$$a_{jk}=\begin{cases}1,\text{物品}j\text{在物品}k\text{的前面}(\alpha_j<\alpha_k)\\0,\text{否则}\end{cases},j,k=1,2,\dots,n$$

$$b_{jk}=\begin{cases}1,\text{物品}j\text{在物品}k\text{的后面}(\alpha_j>\alpha_k)\\0,\text{否则}\end{cases},j,k=1,2,\dots,n$$

$$c_{jk}=\begin{cases}1,\text{物品}j\text{在物品}k\text{的左面}(\beta_j<\beta_k)\\0,\text{否则}\end{cases},j,k=1,2,\dots,n$$

$$d_{jk}=\begin{cases}1,\text{物品}j\text{在物品}k\text{的右面}(\beta_j>\beta_k)\\0,\text{否则}\end{cases},j,k=1,2,\dots,n$$

$$e_{jk}=\begin{cases}1,\text{物品}j\text{在物品}k\text{的下面}(\gamma_j<\gamma_k)\\0,\text{否则}\end{cases},j,k=1,2,\dots,n$$

$$f_{jk}=\begin{cases}1,\text{物品}j\text{在物品}k\text{的上面}(\gamma_j>\gamma_k)\\0,\text{否则}\end{cases},j,k=1,2,\dots,n,$$

$(\alpha_j,\beta_j,\gamma_j)$为连续变量，表示物品 j 的前-左-下角的坐标，也叫做物品 j 的起始坐标。

该问题的目标是最小化所使用箱子的数量，目标函数的数学表达式为：

$$\min f(y)=\sum_{i=1}^{m}y_i,$$

约束条件：

每个物品必须放入一个箱子：

$$\text{s.t.}\sum_{i=1}^{m}\chi_{ij}=1,\forall j\in N=\{1,2,\dots,n\},$$

箱子容量限制：

$$\sum_{j=1}^{n} x_{ij} \le My_i, \forall i \in I = \{1,2,\dots,m\},$$

在同一个箱子中的物品不能重叠：

$$\alpha_j + l_j p_{\alpha j} + w_j q_{\alpha j} + h_j r_{\alpha j} \le \alpha_k + \left(1 - a_{jk}\right)M, \forall j \in N, i \in I$$

$$\alpha_k + l_k p_{\alpha k} + w_k q_{\alpha k} + h_k r_{\alpha k} \le a_j + \left(1 - b_{jk}\right)M, \forall j \in N, i \in I$$

$$\beta_j + l_j p_{\beta j} + w_j q_{\beta j} + h_j r_{\beta j} \le \beta_k + \left(1 - c_{jk}\right)M, \forall j \in N, i \in I$$

$$\beta_k + l_k p_{\beta k} + w_k q_{\beta k} + h_k r_{\beta k} \le \beta_j + \left(1 - d_{jk}\right)M, \forall j \in N, i \in I$$

$$\gamma_j + l_j p_{\gamma j} + w_j q_{\gamma j} + h_j r_{\gamma j} \le \gamma_k + \left(1 - e_{jk}\right)M, \forall j \in N, i \in I$$

$$\gamma_k + l_k p_{\gamma k} + w_k q_{\gamma k} + h_k r_{\gamma k} \le \gamma_j + \left(1 - f_{jk}\right)M, \forall j \in N, i \in I$$

$$a_{jk} + b_{jk} + c_{jk} + d_{jk} + e_{jk} + f_{jk} \ge x_{ij} + x_{ik} - 1, \forall j \in N, i \in I,$$

每一个物品都必须完全存在于某一个箱子中：

$$\alpha_j + l_j p_{\alpha j} + w_j q_{\alpha j} + h_j r_{\alpha j} \le L_i + \left(1 - x_{ij}\right)M, \forall j \in N, i \in I$$

$$\beta_j + l_j p_{\beta j} + w_j q_{\beta j} + h_j r_{\beta j} \le W_i + \left(1 - x_{ij}\right)M, \forall j \in N, i \in I$$

$$\gamma_j + l_j p_{\gamma j} + w_j q_{\gamma j} + h_j r_{\gamma j} \le H_i + \left(1 - x_{ij}\right)M, \forall j \in N, i \in I,$$

确保每个物品 j 必须选择一种长（宽、高）的放置方式：

$$p_{\alpha j} + p_{\beta j} + p_{\gamma j} = 1, \forall j \in N$$

$$q_{\alpha j} + q_{\beta j} + q_{\gamma j} = 1, \forall j \in N$$

$$r_{\alpha j} + r_{\beta j} + r_{\gamma j} = 1, \forall j \in N,$$

确保每个物品 j 的长或宽或高平行于 X 轴（Y 轴、Z 轴）：

$$p_{\alpha j} + q_{\alpha j} + r_{\alpha j} = 1, \forall j \in N$$

$$p_{\beta j} + q_{\beta j} + r_{\beta j} = 1, \forall j \in N$$

$$p_{\gamma j} + q_{\gamma j} + r_{\gamma j} = 1, \forall j \in N,$$

变量的基本约束：

$$\alpha_j \geq 0, \beta_j \geq 0, \gamma_j \geq 0, \forall j \in N$$

$$p_{\alpha j}, p_{\beta j}, p_{\gamma j}, q_{\alpha j}, q_{\beta j}, q_{\gamma j}, r_{\alpha j}, r_{\beta j}, r_{\gamma j}, a_{jk}, b_{jk}, c_{jk}, d_{jk}, e_{jk}, f_{jk} \in \{0,1\}, \forall j, k \in N, i \in I.$$

6.6.3 启发式算法

这些启发式算法，通常被嵌入到 GA 或者 Tabu Search 当中，来增强 GA 和 Tabu Search 的搜索能力，所以简单做一下介绍。

1.Wall building 算法

在 Wall building 算法中，我们先把物品堆砌成一面一面的“墙（wall）”，然后将这些“物品墙”装入箱子中。

每一面墙的宽和高不会超过箱子的宽和高，而其长度由最前-左-下位置上的物品决定。我们把这个物品称为建造这面墙的关键物品（Wall Determining Box，WDB）。

2.Tower building 算法

在 Tower building 算法中，我们先把物品堆砌成一座一座“塔（tower）”，然后将这些“物品塔”装入箱子中。

每一座塔的底面积由其最下面的物品决定。我们把这个物品称为构成这座塔的关键物品（Tower Determining Box，TDB）。

3.Corner position packing 算法

Corner position packing：一个物品被放置在 Corner position 时，这个物品的左面应该触碰箱子的左壁或者触碰已经存放于这个箱子的其他物品的右壁，其前面应该触碰箱子的前壁或者已经存放于这个箱子的其他物品的后壁，而其下面应

该触碰箱子的底面或者其他物品的上壁。

6.7 本章小结

本章主要讨论了装箱问题及其求解方法。装箱问题是一个经典的组合优化问题，广泛应用于物流、资源分配和生产管理等领域。通过数学建模和算法求解，我们旨在优化资源使用，最小化箱子的数量。在传统的装箱问题中，我们探讨了多种经典启发式算法，包括次优配合启发式（NF）、优先配合启发式（FF）和最佳配合启发式（BF）。这些算法虽无法保证最优解，但能在较短时间内提供可行解，尤其适用于实际问题的求解。

在进一步研究遗传算法应用于装箱问题时，详细介绍了遗传算法的编码策略、适值函数设计、遗传运算（交叉与变异操作）以及选择策略。这些操作帮助遗传算法在大规模和复杂约束的装箱问题中进行全局搜索，避免陷入局部最优，确保较高的解决效率。遗传算法的编码方式分为基于箱子、物品和群体的表示方法，每种方法都有其优缺点，适用于不同类型的装箱问题。

此外，二维和三维装箱问题作为装箱问题的扩展，增加了对空间位置和物品布局的要求，在实际应用中尤为重要。二维装箱问题可应用于集装箱装载和切割问题，三维装箱问题则广泛应用于大型货物和多维物品的装载。

通过对各类装箱问题及求解方法的探讨，本章展示了装箱优化算法在提高资源利用率、降低成本以及提升效率方面的应用潜力。装箱问题不仅是理论上的研究课题，还是多个行业中提升效率和降低成本的核心问题。

研学互通

为了通过引介装箱问题的典型文献资源，拓展读者对该问题研究脉络与前沿动态的理解。这里精选了具有代表性的研究论文，涵盖一维、二维与三维装箱模型，兼顾算法设计、性能分析与工程应用等多个角度。借助这些文献，读者不仅能够掌握装箱问题的建模思想与求解策略，还能进一步理解其在供应链优化、物流集成与智能排产等实际场景中的运用价值。

(1) Coffman, E. G., Jr., Csirik, J., Galambos, G., Martello, S., & Vigo, D. (2013).

Bin packing approximation algorithms: Survey and classification.

本文是目前关于一维装箱问题近似算法最权威和系统的综述之一，不仅梳理了首次适用于该问题的各种经典近似策略如 First Fit、Best Fit、Next Fit 等，也对其时间复杂度和近似比进行了详细分类和对比。文章的价值在于，它为研究者提供了一个极为清晰的知识框架，能够帮助读者识别不同算法在理论最优性和实际效率之间的平衡点。

(2) Garey, M. R., & Johnson, D. S. (1981). Approximation algorithms for bin packing problems: A survey.

这篇早期的文献奠定了装箱问题作为 NP 难问题的重要地位。文章回顾了当时最主要的近似算法，并从计算复杂性理论角度探讨了为何装箱问题在给定精度下不能被多项式时间算法精确求解，是学习 NP 问题与近似算法关系的核心材料之一。

(3) Christensen, H. I., Khan, A., Pokutta, S., & Tetali, P. (2017). Approximation and online algorithms for multidimensional bin packing: A survey. Computer Science Review, 24, 63–79.

本文聚焦于多维装箱问题（包括二维与三维），系统介绍了针对在线场景和高维空间下的装箱策略，特别强调了算法在面对动态物品到达、不同尺寸限制和优先权约束时的适应性。它对于深入理解复杂环境中装箱策略的设计与性能评估具有重要参考价值。

(4) Martello, S., & Toth, P. (1995). A hybrid improvement heuristic for the one-dimensional bin packing problem. European Journal of Operational Research, 84(2), 303–315.

本文提出了一种混合改进启发式算法，针对一维装箱问题提出了创新的解决方案，通过结合不同的启发式方法来优化箱子的利用率，并减少使用的箱子数量，具有较强的实际应用价值。

(5) Pisinger, D., & Røpke, S. (2007). Algorithms for the two-dimensional bin packing problem with partial conflicts. Computers & Operations Research, 34(10),

3067 – 3087.

本文针对二维装箱问题，提出了一种处理部分冲突（如物品之间互斥）的算法。这些算法不仅关注物品的有效排列，还考虑了物品之间的冲突限制，适用于一些特殊的装箱场景，在实际应用中具有较高的价值。

思行经世：义甬集装箱海铁联运通道的绿色发展实践

随着全球环保意识的提升与绿色低碳发展理念的普及，交通运输行业作为全球二氧化碳排放的主要来源之一，正面临着巨大的转型压力。传统的运输方式，如公路运输，能源消耗大、排放高，因此亟需通过科技创新与优化措施来减少碳排放、节能降耗。义甬集装箱海铁联运通道的建设正是响应这一转型趋势的重要实践之一。该通道连接浙江义乌与宁波港，采用了“铁水联运”模式，融合铁路与水路运输的优势，通过优化集装箱的装载方案，不仅提高了运输效率，降低了物流成本，还在绿色低碳发展方面取得了显著成果。

在绿色发展成效方面，义甬集装箱海铁联运通道的实施有效减少了碳排放，累计减少 4.6 万吨碳排放，显著推动了绿色物流的实践。此外，铁路运输替代了部分公路运输，极大地降低了燃油消耗。过去三年中，该通道累计节省了 1668.7 万升燃油，按照当前油价计算，节省的能源成本约为 12515.6 万元。这一改变不仅降低了运输成本，也减少了对环境的负担。铁路运输相比于公路运输，噪音污染更少，且通过减少公路上的运输车辆数量，降低了交通噪音，改善了城市及周边环境的生活质量。

装箱优化在这一绿色物流实践中发挥了关键作用。通过优化集装箱的装载方案，实现了运输资源的最大化利用，减少了多余的运输频次和能源消耗。具体来说，装箱优化技术通过合理的货物分类与装载，使每个集装箱的空间得到充分利用，减少了不必要的空载和运输过程中的浪费。这一过程不仅提高了运输效率，还与节能减排、环境保护等社会效益密切相关。在绿色低碳发展的背景下，装箱问题不仅仅是为了提高运输效率，它还推动了低碳物流目标的实现，进一步促进了环境友好型社会的发展。

义甬集装箱海铁联运通道的绿色发展实践，不仅体现了物流运输领域的技术

创新，也体现了国家战略和社会责任的结合。作为全球绿色低碳转型的重要参与者，中国在交通运输领域的绿色转型尤为关键。通过这一项目的实施，技术创新为国家战略和社会责任提供了具体的支持。此外，优化装箱方案和运输模式不仅降低了企业的运营成本，也为社会创造了更多的经济和环境价值。绿色低碳发展不仅符合国家可持续发展战略，还为民众提供了更加清洁、高效的运输方式，推动了社会向更加绿色、可持续的方向发展。

习题

习题 6.1 讨论装箱问题的现实应用场景，并举例说明装箱问题在工业界或日常生活中的一个实际应用。

习题 6.2 某国际航运公司正在进行集装箱运输，每个集装箱的最大容量为 25 单位重量。该公司需要运输一批电子产品和家电产品。这些物品的重量分别如下：$w = [12,10,8,6,15,4,3,11,7,9]$，每个集装箱的最大容量为 25 单位重量。公司的目标是使用最少的集装箱将所有物品运送到目的地。

（1）使用 Best Fit（BF）算法，计算最少需要多少个集装箱才能装下所有物品，并详细列出每个步骤中所用集装箱的情况。

（2）如果每个集装箱的容量是 15 单位重量，使用 First Fit（FF）算法计算所需的箱子数量，并说明每个步骤中所用箱子的装载情况。

习题 6.3 某制造公司生产金属零件，并需要从大块钢板中切割出多个不同尺寸的零件。每块钢板的宽度固定为 100 单位长度，公司要根据零件的尺寸将其切割并尽量减少废料。以下是需要从钢板上切割的零件长度：$w = [30, 40, 60, 20, 70, 50]$，每块钢板的长度为 100。

（1）使用 Next Fit（NF）算法，计算最少需要多少块钢板才能切割出所有零件，并列出每块钢板的切割情况。

（2）使用 First Fit（FF）算法，计算所需的钢板数量，并列出每块钢板的切割情况。

（3）使用 Best Fit（BF）算法计算所需的钢板数量，并列出每块钢板的切割

情况

习题 6.4 某跨国公司正在优化其全球仓储管理系统。该公司希望在装箱过程中同时实现以下两个目标：最小化箱子的数量，并最大化箱子的空间利用率。物品的重量如下：$w = [6,2,9,4,3]$

每个箱子的最大容量为 12 单位重量。要求设计一个适值函数，用于评估装箱方案，并用遗传算法求解。