研究生课程讲义

# 典型优化问题的模型与算法

## (Models and Algorithms for Typical Optimization Problems)

东北财经大学 管理科学与工程学院

感谢东北大学系统工程研究所课程组

# 第六章 装箱问题及求解算法

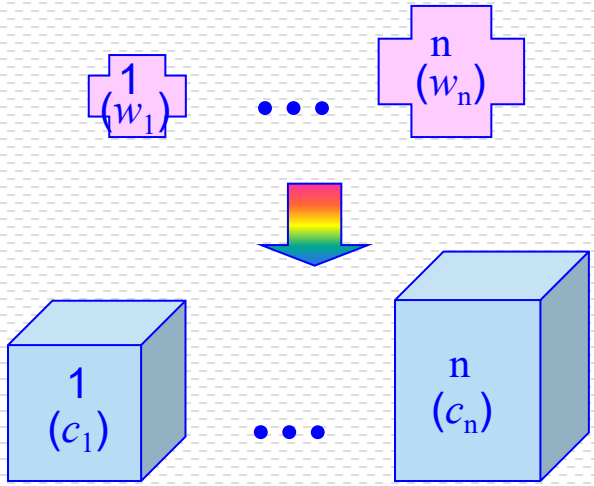
## 装箱问题 (Bin-Packing Problem)

- 装箱问题实例与特征
- 装箱问题的数学描述
- 装箱问题的分类与变种
- 装箱问题的启发式算法
- 装箱问题的GA求解算法

# 装箱问题及特征

- 将 $n$ 个物品装入许多箱子(最多 $n$ 个箱子)。
- 每个物品有重量（$w_j>0$），每个箱子有重量限制 ($c_j>0$)。
- 问题是：寻找最优的将物品分配到箱子的方案，从而使每个箱子中物品的重量之和不超过其限制，而使用的箱子数量最少。

| 物品 | 1 | 2 | … | $n$ |
|---|---|---|---|---|
| 重量 | $w_1$ | $w_2$ | … | $w_n$ |
| 重量限制 | $c_1$ | $c_2$ | … | $c_n$ |




- 通常，所有箱子有相同的重量限制 ($c>0$)。
- 显然，将每个物品放入一个箱子是一个可行解，但不是最优解。

# 模型描述

- 数学表示为：

$$\min \quad f(\boldsymbol{y}) = \sum_{i=1}^{n} y_i$$

$$\text{s.t.} \quad g_i(x, y) = \sum_{j=1}^{n} w_j x_{ij} \le c y_i, \quad i \in N = \{1,2,\ldots,n\}$$

$$g_j(x) = \sum_{i=1}^{n} x_{ij} = 1, \quad j \in N$$

$$y_i = 0 \text{ or } 1, \quad i \in N$$

$$x_{ij} = 0 \text{ or } 1, \quad i, j \in N$$

where

$$y_i = \begin{cases} 1, & \text{if bin } i \text{ is used} \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{if object } j \text{ is assigned to bin } i \\ 0, & \text{otherwise} \end{cases}$$

- 第一个约束是箱子的容量限制，
- 第二个约束表明，一个物品只能放入一个箱子。

# 特点

- NP 完全问题
- 不同于一般的 0-1 整数规划问题
- 仅有一些启发式算法

# 应用

- 在建筑中经常需要从长度一定的棍子上切割不同长度的棒。
- 电气布线中的电线来自长度一定的线卷。
- 贴墙纸需要从给定长度的纸卷中得到。
- 具有相同宽度、不同长度的石料需要用平板架运送到建筑工地。
- 在金属制造工业中，钢片需要从大块钢片中切除。
- 在运输工业中，卡车具有给定的最大承重，通常将不同负荷的重量作为考虑对象。
- 电子工业中不同长度(字节)的微代码程序需要存储至微处理器中具有固定尺寸的存储器中。
- 在作业管理中，不同执行时间的任务需要在满足完成所有任务时间限制的前提之下被分配到不同的工人。
- 重要的生产流水线平衡(沿着生产流水线将任务分配到工作站)问题就是带有附加优先约束的装箱问题。
- 装箱问题是容量限制的工厂选址问题的特例之一。

# 贪婪启发式算法

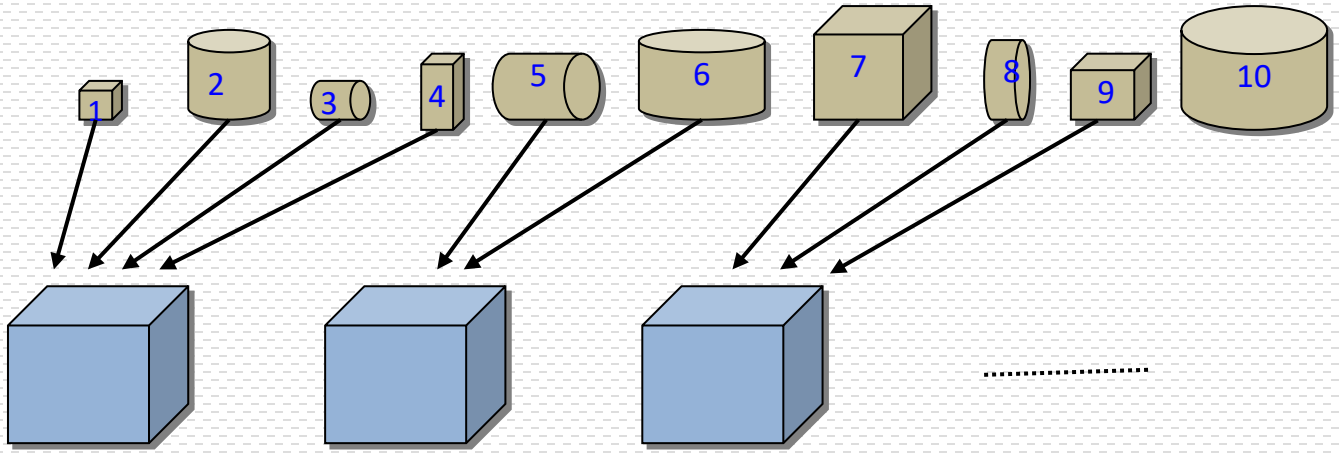
- 装箱问题属于NP-完全问题。目前仅有一些启发式算法。
- 而大多数启发式算法以贪婪方法为特点，采用了某些简单规则，比如：
  - 次优配合
  - 优先配合
  - 最佳配合。
- Garey和Johnson指出：
  - 简单启发式算法的结果可以不差于(但也不好于)最优解乘以一个相当小的系数。

# 一般的启发式求解方法

- 次优配合启发式方法(next-fit heuristic，*NF*)
  - 第 1 个物品放入第 1 个箱子，然后是根据下标上升的顺序放入第 2，…，*n* 个物品。
  - 如果当前箱子的容量允许，则每个物品放入当前箱子，否则放入一个新的箱子。于是新箱子成为当前箱子。
  - 对于给定的问题 *I*，算法给出的解值 *NF(I)* 满足下面约束：

$$NF(I) \le 2z(I) \qquad \text{其中} z(I) \text{代表最优解}$$

  - 算法的复杂性是： *O*(*n*)。
  - 物品的排列不同，所获得的问题的解也不同

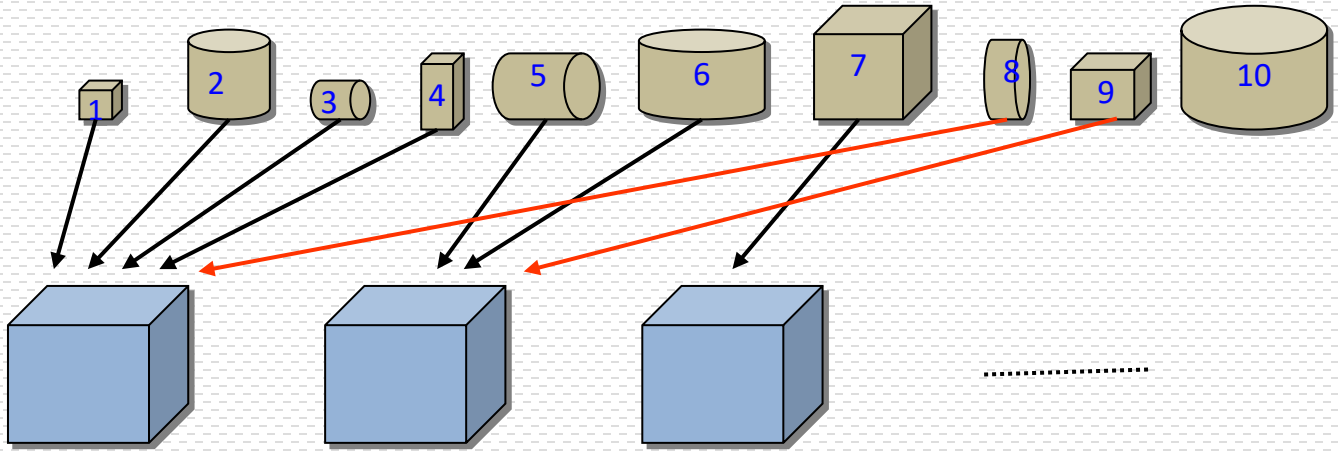


# 一般的启发式求解方法

- 优先配合启发式方法(first-fit heuristic，*FF*)
  - 也根据下标上升的顺序考虑物品。但将每个物品放入其能够放入的最小下标的已初始化的箱子。
  - 如果当前物品无法放入任何已初始化的箱子,则放入一个新箱子。对于所有装箱问题有:

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

  - 时间复杂性是：$O(n \log n)$

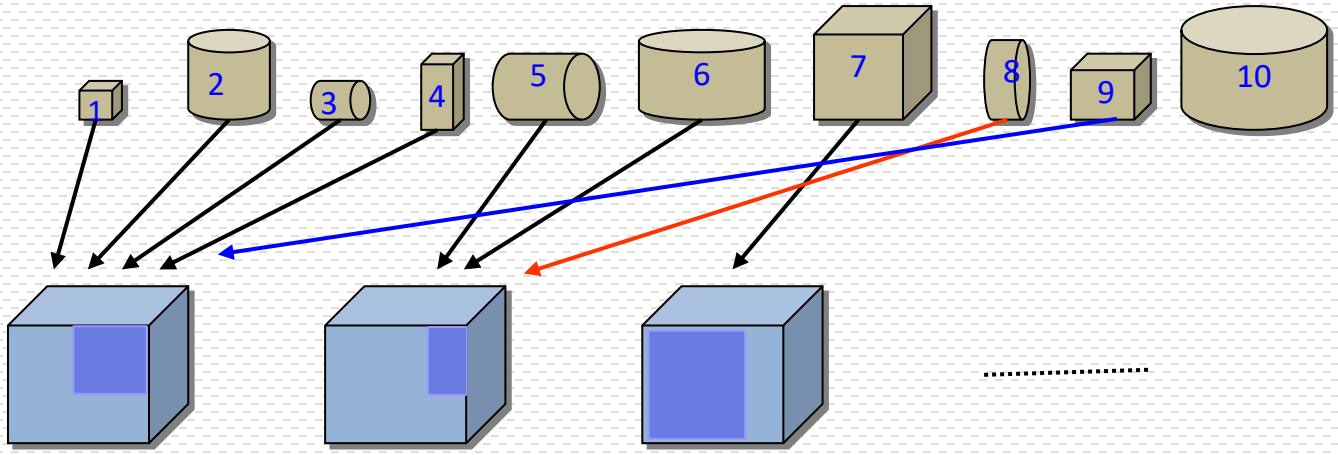


# 一般的启发式求解方法

- 最佳配合启发式方法(best-fit heuristic，*BF*)
  - 最佳配合算法从 *FF* 算法改进而来
  - 它将当前物品放入具有最小剩余容量的可行箱子中
  - 该算法摒弃了有利于最小下标箱子的思想
  - *BF* 满足与 *FF* 相同的最坏情况边界

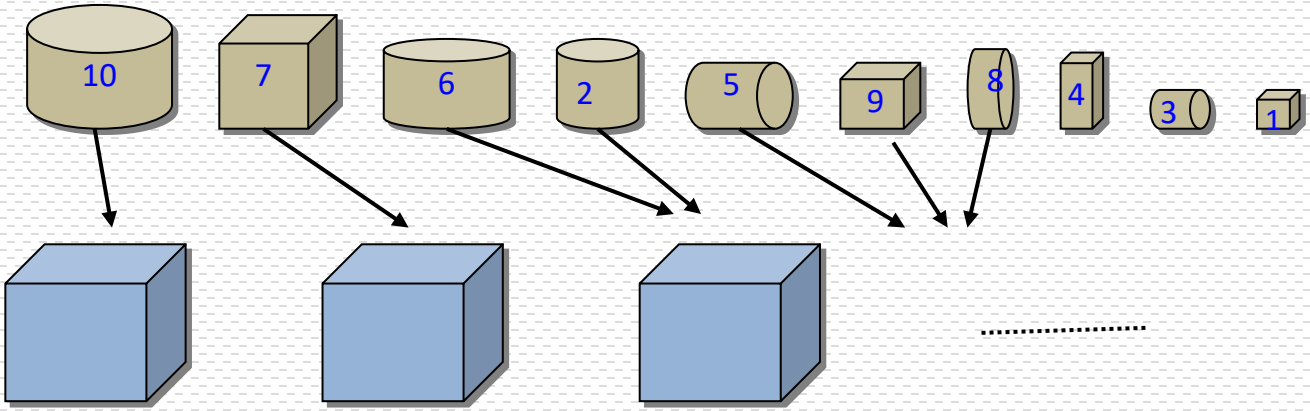
$$BF(I) \le \frac{17}{10} z(I) + 2$$

  - 时间复杂性是：*O*(*n log n*)



# 一般的启发式求解方法

- 其它启发式方法
  - 如果物品按照重量降序排列 ($w_1 \geq w_2 \geq \ldots \geq w_n$)
  - 然后采用 *NF*, *FF* 或 *BF* 得到的算法相应称作
    - 次优配合降序(next-fit decreasing, *NFD*)
    - 优先配合降序(first-fit decreasing, *FFD*)
    - 最佳配合降序(best fit decreasing, *BFD*)
  - 这些算法的时间复杂度都是： $O(n \log n)$
  - 通常用于测试新提出的用于装箱问题的算法的有效性。
  - 启发式算法通常被嵌入遗传算法来增强遗传搜索能力，从而为装箱问题寻找到更接近最优解的答案。

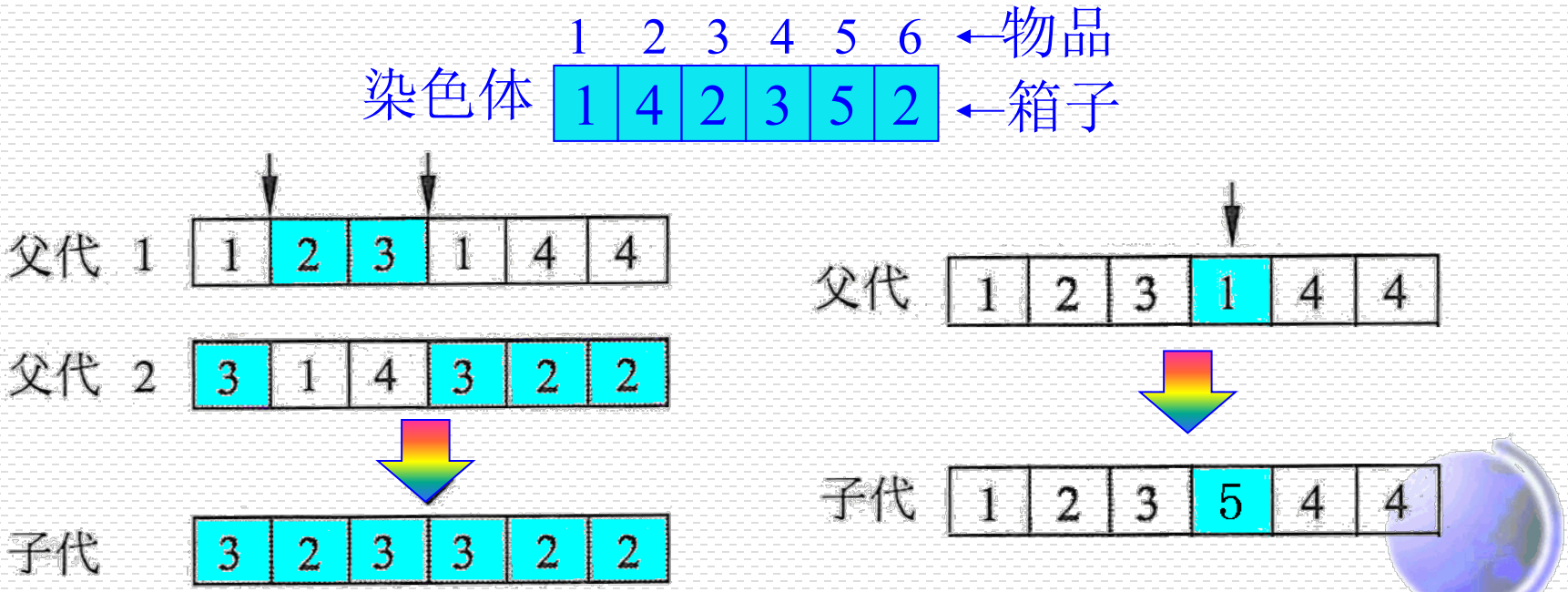


# GA求解

- 编码
  - 基于箱子的表示 (bin-based representation)
    - 最直接的方法就是对物品的从属关系进行编码。
  - 基于物品的表示 (object-based representation)
    - 对物品的排列进行编码，然后应用解码器得到对应的解
  - 基于群体的表示 (group-based representation)
    - 对于装箱问题，好的表示方法需要包含两个部分
      - 第 1 部分提供关于哪个物品属于哪个箱子(群体)的信息。
      - 第 2 部分对使用的箱子进行编码。

# GA求解

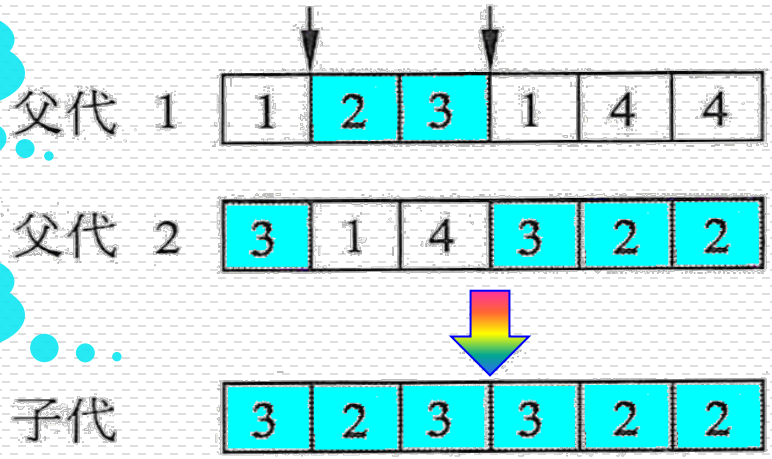
- 基于箱子的表示
  - 基因的位置表示物品，基因的值表示该物品放入的箱子号
  - 优点：
    - 染色体具有恒定长度(与物品的数量相同)，可以采用标准的遗传算子



# GA求解

- 基于箱子的表示
  - 缺点：
    - 高度冗余。例如，123144 和 314322 就是对问题相同解的不同编码。这种冗余随箱子数量（问题规模）的增加而指数上升，导致搜索空间比解空间大，因而削弱了GA的搜索能力。
    - 会使某基因的含义依赖于其他基因，这种情况在采用标准杂交算子时是非常不希望发生的。可能导致：两个相同的父代杂交后却产生了不同的子代的现象。
    - 可能由于一个箱子被放入了超量的物品而产生不可行解。

相同的父代，使用4个箱子




不同的后代，使用2个箱子

第1和第4个物品放入同一个箱子，第5和第6个物品放入同一个箱子，第2和第3个物品分别被放入两个箱子。

# GA求解

- 基于物品的表示
  - 对物品的排列进行编码，应用解码器得到相应的解
    - 例如：根据给定的染色体排列从左到右将物品放入箱子从而得到问题的解（次优配合启发式）
  - 设有6个物品需要装箱，编码就是数字1到6的排列

物品编号 1 2 3 4 5 6，编码为：

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 3 | 2 | 1 | 4 | 5 | 6 |
| 4 | 5 | 2 | 1 | 3 | 6 |

  - 优点：
    - 与基于箱子的表示不同，这种方法从不产生不可行解。

# GA求解

- 基于物品的表示
  - 缺点：
    - 编码是高度冗余的。冗余度随着问题规模的增加而指数上升，遗传算法的能力受到严重影响。

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 |
| 3 | 2 | 1 | 4 | 5 | 6 |
| 2 | 1 | 3 | 4 | 5 | 6 |

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | 5 | 4 | 6 |
| 3 | 2 | 1 | 5 | 4 | 6 |
| 4 | 5 | 2 | 1 | 3 | 6 |

例如：假设可以如上进行划分，就是说，有3个箱子，一个装物品1到3，第2个装物品4和5，第3个装物品6。于是同一箱子里的物品的任意排列均产生新的染色体，而其代表的解却是相同的（假设后三个染色体中，5号和2号物品不能放入前面物品构成的箱子）。

    - 某物品放入的箱子号依赖于染色体的左边位置的物品。给定基因的位置与问题的费用函数没有任何关系。导致这种表示方法无法在进化过程中维持从父代继承的信息。

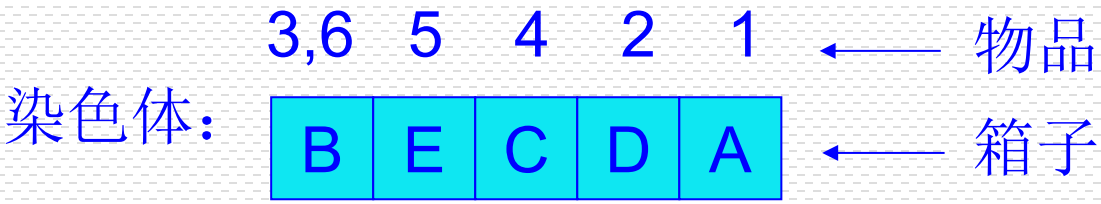
# GA求解

- 上述两种表示方法对于装箱这样的*群体问题*不适合，原因在于这些染色体是面向项(item)的，而不是面向群体(group)的。
- 简言之，由于装箱问题的费用函数依赖于箱子中物体的群体，上述编码不适合用作这类问题的费用函数优化。

- 对于装箱问题，更好的表示方法需要包含两个部分。
  - 第1部分提供关于哪个物品属于哪个箱子(群体)的信息。
  - 第2部分对使用的箱子进行编码（装不同物品的箱子有不同的编码）。
- 基于这样的考虑，Falkenauer提出了一种“基于群体”的表示方法，这种方法用一个基因表示一个箱子。

# GA求解

- 基于群体的表示
  - 第 1 部分提供关于哪个物品属于哪个箱子（群体）的信息。
  - 第 2 部分对使用的箱子进行编码。
  - 群体的意思是将某个箱子中的物品的集合用一个字母来表示。



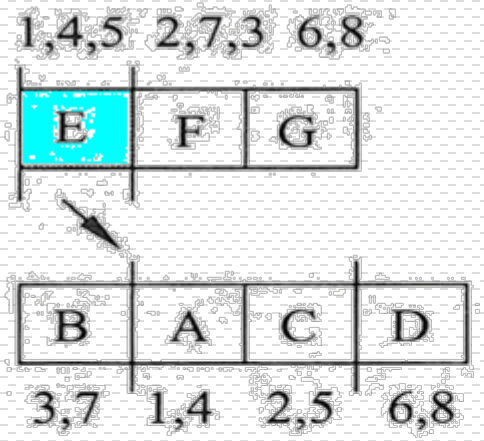

  - 这种编码使得基因既表示物品又表示群体(箱子)
    - 一个合法的染色体，应包含所有的物品且不能重复。
  - 对于装箱问题来说，群体是有意义的积木块，基因在交叉变异后的子代中意义不变，可以继承父代的信息。
  - 这种表示的关键之处就是遗传算子对于染色体的群体部分进行操作，其物品部分仅用于判定由哪些物品形成该群体。
  - 特别指出：这种表示意味着算子需要处理不同长度的染色体。

# GA求解

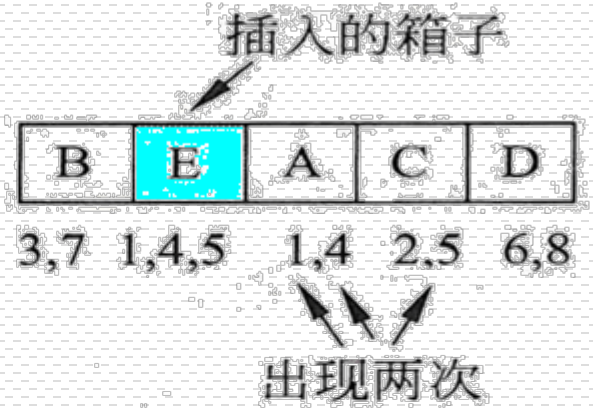
- 交叉
  - 基于群体的表示包含两个部分: 箱子和物品的群体。
  - 因此交叉需要处理可变长的染色体。
- 交叉的过程
  1. 随机选择两个杂交位置，对每个父代选定杂交部分。
  2. 将第 1 个父代杂交部分的内容插入到第 2 个父代第 1 个杂交位置之前。由于杂交对染色体的部分群体进行操作，这就意味着从第 1 个父代插入一些群体 (箱子) 到第 2 个父代中。
  3. 从产生的后代中原有的箱子中去掉所有重复出现的物品，使得这些物品原先的从属关系让位于“新”插入的箱子。因此产生的后代中的某些群体发生了改变。他们不再包含与先前相同的物品，原因是消除了一些物品。
  4. 如果必要，根据问题的约束和待优化的费用函数来调整新产生的箱子。在本阶段，可以采用如 *FFD* 等局部搜索算法。
  5. 改变两个父代的角色并重新应用 2 到 4 步生成第 2 个子代。

# GA求解

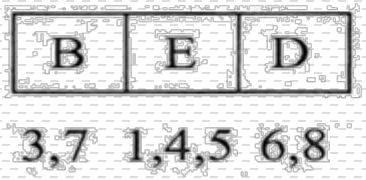
- 交叉过程的图示




(a) 选择杂交部分



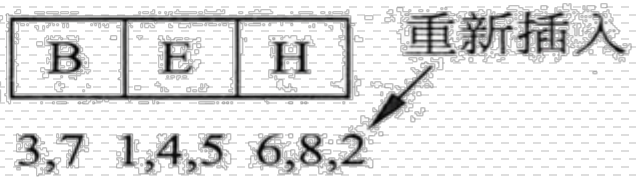

(b) 插入杂交部分





放到一边

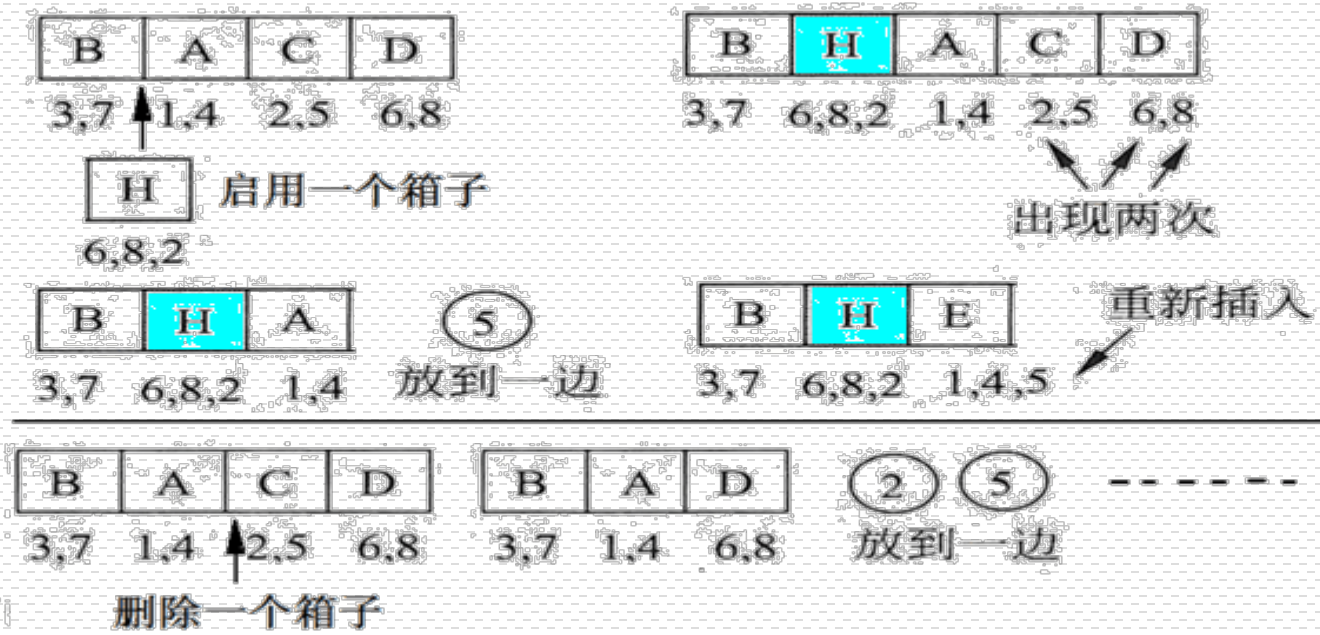
(c) 消除空箱子和重复的箱子




(d) 重新插入消失的物品

# GA求解

- 变异
  - 装箱问题的变异算子必须针对群体 (箱子) 而不是物品进行操作
  - 变异的一般性策略：
    - 或启用一个新的箱子
    - 或消除一个已经使用的箱子。
  - 如果变异后解中缺少某些物品，可以采用 *FF* 或 *FFD* 启发式方法来按照随机的顺序将其重新放入箱子。



# GA求解

- 适值函数
  - 存在两种评价目标函数的方法
    - 最小化使用的箱子数量；
    - 最小化使用的箱子数量同时尽量装满所有使用的箱子
  - Falkenauer 和 Delchambre 对装箱问题提出了下面的评价函数 (属于上述第二种方法)

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

where

$N$：解中使用的箱子数,

$F_i$: 第 $i$ 个箱子中所有物品的重量之和 (箱子的填充程度),

$C$：箱子的容量,

$k$：常数, $k > 1$. 表示对装满箱子的重视程度。经验值 $k = 2$

  - GA优化的目标是最大化 $f_{BPP}$。一方面，使用的箱子数量越少，$f_{BPP}$越大。另一方面 $F_i$ 越接近 $C$，$f_{BPP}$越大

# GA求解

- 种群初始化
  - 装箱问题有两种产生初始种群的方法
    - 随机的方法
      - 根据所有的物品，随机产生一些可行的箱子（包含随机个物品），如果不存在则对箱子进行编码，构成一个染色体；反复进行，产生初始种群。
    - 启发式的方法
      - 产生一个物品的随机序列，然后用 *FF* 启发式方法装箱，如果不存在则对箱子进行编码；反复进行直至达到种群规模。
      - 种群可能由箱子 (基因) 数相差很大的个体组成。

# GA求解

- 选择
  - 实际应用中，通常有两种选择模式
    - 按比例选择；
    - 基于次序的选择。
  - 按比例选择：基于种群中染色体适值的相对关系选择染色体。
    - 轮盘赌选择 (Roulette wheel selection)
    - 随机通用选择 (Stochastic universal selection)
    - 随机剩余选择 (Stochastic remainder selection)
  - 基于次序的选择：对染色体按适值进行排序，依据排序结果进行选择。
    - 竞争选择 (Tournament selection)
    - $\mu+\lambda$ 选择 ($\mu+\lambda$ selection)
    - 截断选择 (Truncation selection)
    - 线性排序选择 (Linear ranking selection)
  - 选择会对进化产生一定的影响。

# 第六章 装箱问题及求解算法

## 装箱问题 (Bin-Packing Problem)

- 装箱问题实例与特征
- 装箱问题的数学描述
- 装箱问题的分类与变种
- 装箱问题的启发式算法
- 装箱问题的GA求解算法

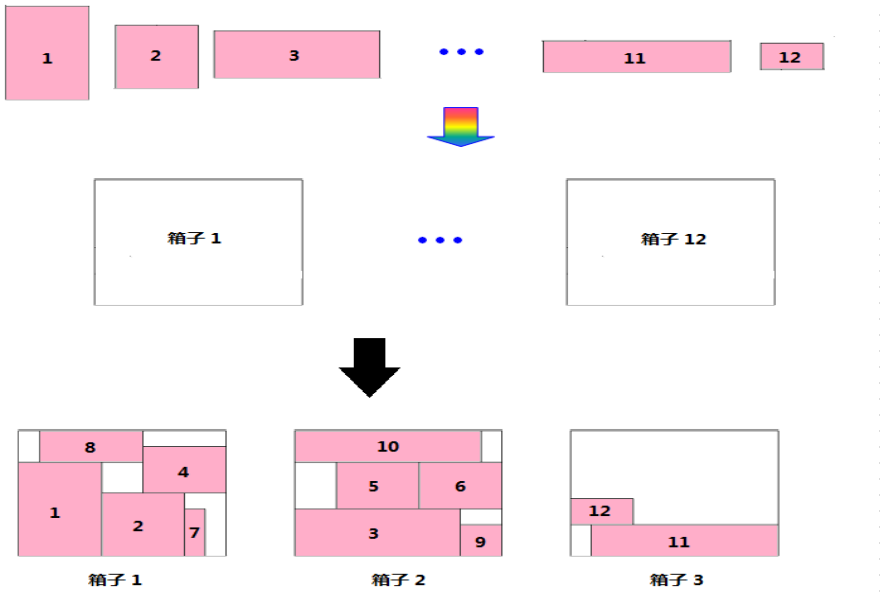
# 第六章 装箱问题及求解算法

## 装箱问题 (Bin-Packing Problem)

- 装箱问题实例与特征
- 装箱问题的数学描述
- 装箱问题的启发式算法
- 装箱问题的GA求解算法
- 二维装箱问题及启发式算法

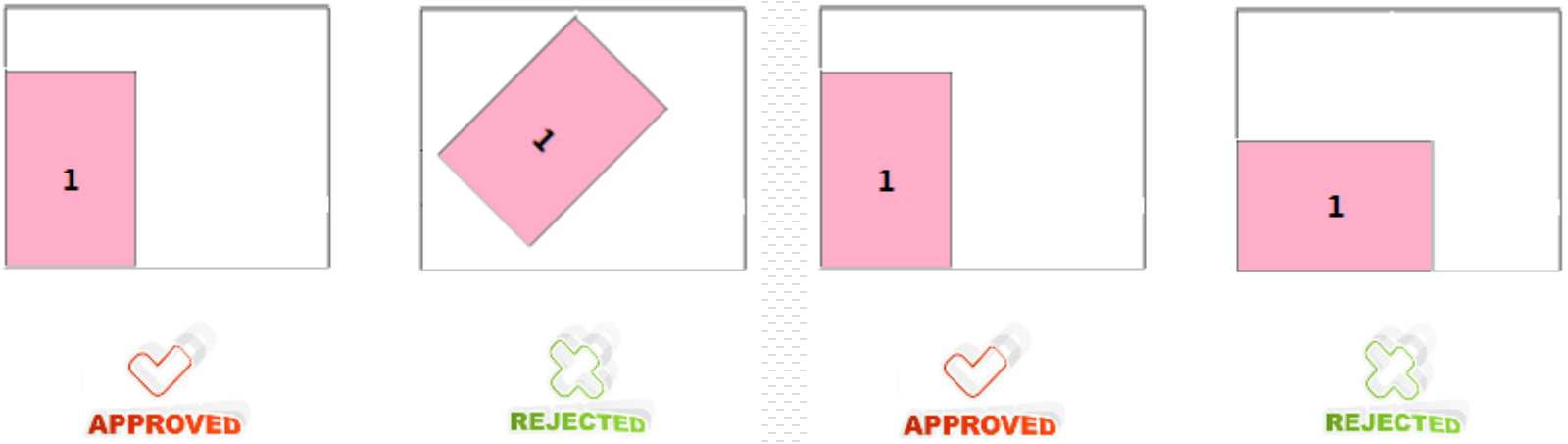
# 二维装箱问题及特征

- 给定 $m$ 个矩形箱子和 $n$ 个矩形物品（$m>=n$）
- 每个物品有长度（$l_j>0$）和宽度（$w_j>0$），每个箱子也有长度（$L_i>0$）和宽度（$W_i>0$）。
- 问题是：寻找最优的方案将矩形物品分配到矩形箱子中，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。



# 二维装箱问题及特征

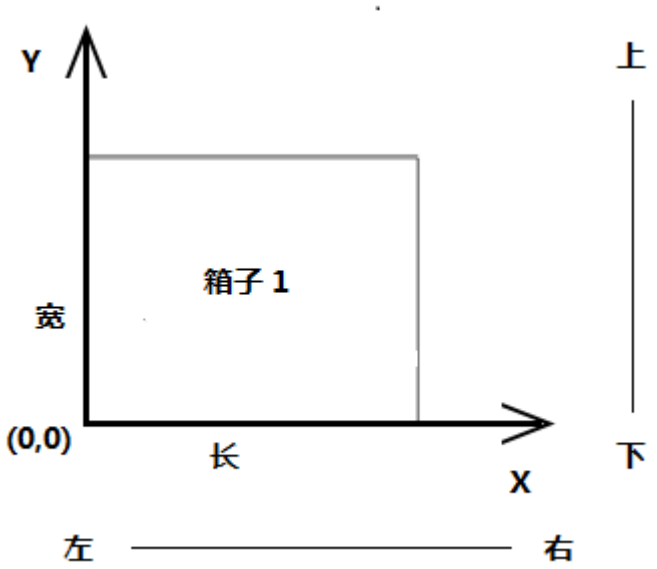
- 通常，所有箱子有相同的长度和宽度，$L_i = L, W_i = W, \forall\, i = 1,2,\ldots,m$。
- 物品的边缘应该平行于它所在的箱子的边缘。
- 在某些情况下，我们要求物品不可以旋转90度。
- 显然，如果每一个物品的宽度都等于箱子的宽度 $W$，那么二维装箱问题就被简化为一维装箱问题。




边缘平行　　不可旋转

# 模型描述

- 假设
  - 每一个箱子都被放置在一个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，箱子的左下角与坐标原点重合。

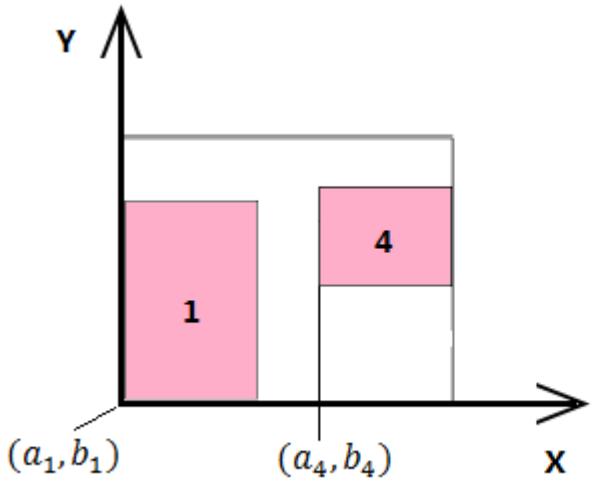


# 模型描述

- 数学模型中使用的变量
  - $y_i$：0-1变量，$y_i = 1$表示箱子 $i$被使用，$y_i = 0$表示箱子 $i$没有被使用
  - $x_{ij}$：0-1变量，$x_{ij} = 1$表示物品 $j$被装在箱子 $i$中，$x_{ij} = 0$表示物品 $j$没有被装在箱子$i$中
  - $(a_j, b_j)$：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
  - $p_{aj}$：0-1变量，$p_{aj} = 1$表示物品 $j$ 的长平行于$X$ 轴
  - $p_{bj}$：0-1变量，$p_{bj} = 1$表示物品 $j$ 的长平行于$Y$ 轴
  - $q_{aj}$：0-1变量，$q_{aj} = 1$表示物品 $j$ 的宽平行于$X$ 轴
  - $q_{bj}$：0-1变量，$q_{bj} = 1$表示物品 $j$ 的宽平行于$Y$ 轴

# 模型描述

- $c_{jk}$: 0-1变量，$c_{jk}=1$表示物品 $j$ 在物品$k$的左面 $(a_j < a_k)$
- $d_{jk}$: 0-1变量，$d_{jk}=1$表示物品 $j$ 在物品$k$的右面 $(a_j > a_k)$
- $e_{jk}$: 0-1变量，$e_{jk}=1$表示物品 $j$ 在物品$k$的下面 $(b_j < b_k)$
- $f_{jk}$: 0-1变量，$f_{jk}=1$表示物品 $j$ 在物品$k$的上面 $(b_j > b_k)$




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

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

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

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

# 模型描述

1. $\min \quad f(\boldsymbol{y}) = \sum_{i=1}^{m} y_i$
2. $\text{s.t.} \sum_{i=1}^{m} x_{ij} = 1, \forall\, j \in N = \{1,2,\dots,n\}$
3. $\sum_{j=1}^{n} x_{ij} \le M y_i, \forall\, i \in I = \{1,2,\dots,m\}$
4. $a_j + l_j p_{aj} + w_j q_{aj} \le a_k + (1 - c_{jk})M, \forall\, j \in N, i \in I$
5. $a_k + l_k p_{ak} + w_k q_{ak} \le a_j + (1 - d_{jk})M, \forall\, j \in N, i \in I$
6. $b_j + l_j p_{bj} + w_j q_{bj} \le b_k + (1 - e_{jk})M, \forall\, j \in N, i \in I$
7. $b_k + l_k p_{bk} + w_k q_{bk} \le b_j + (1 - f_{jk})M, \forall\, j \in N, i \in I$
8. $c_{jk} + d_{jk} + e_{jk} + f_{jk} \ge x_{ij} + x_{ik} - 1, \forall\, j \in N, i \in I$
9. $a_j + l_j p_{aj} + w_j q_{aj} \le L_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
10. $b_j + l_j p_{bj} + w_j q_{bj} \le W_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
11. $p_{aj} + p_{bj} = 1, \forall\, j \in N$
12. $q_{aj} + q_{bj} = 1, \forall\, j \in N$
13. $p_{aj} + q_{aj} = 1, \forall\, j \in N$
14. $p_{bj} + q_{bj} = 1, \forall\, j \in N$
15. $a_j \ge 0, b_j \ge 0, \forall\, j \in N$
16. $p_{aj}, p_{bj}, q_{aj}, q_{bj}, c_{jk}, d_{jk}, e_{jk}, f_{jk} \in \{0,1\}, \forall\, j,k \in N, i \in I$

*M is a large constant*

# 集合覆盖模型

- 装箱方案 (a packing pattern)：如果一组物品可以装入一个箱子，那么我们称这组物品为一个装箱方案。
- 这样的packing pattern的数量 $K$ 可能是指数级的。
- 我们用长度为 $n$的列向量 $A^k$表示一个packing pattern：$A^k$的每一个元素 $a_j^k$指示物品$j$ 是否在这个packing pattern中，如果存在则 $a_j^k = 1$，否则 $a_j^k = 0$。
- 由所有packing pattern的列向量组成的矩阵可以表示为 $A$。

$$\min \ f(\boldsymbol{x}) = \sum_{k=1}^{K} x_k$$

$$\text{s.t.} \sum_{k=1}^{K} a_j^k x_k = 1, \forall\, j \in \{1,2,\dots,n\}$$

$$x_k \in \{0,1\}, \ \forall\, k \in \{1,2,\dots,K\}$$

- $x_k = 1$表示第 $k$个packing pattern出现在解中，否则 $x_k = 0$

# 特点

- NP 难问题
- 不同于一般的整数规划问题
- 仅有一些启发式算法可以在有限的计算空间和时间约束下得到较好的解

# 应用

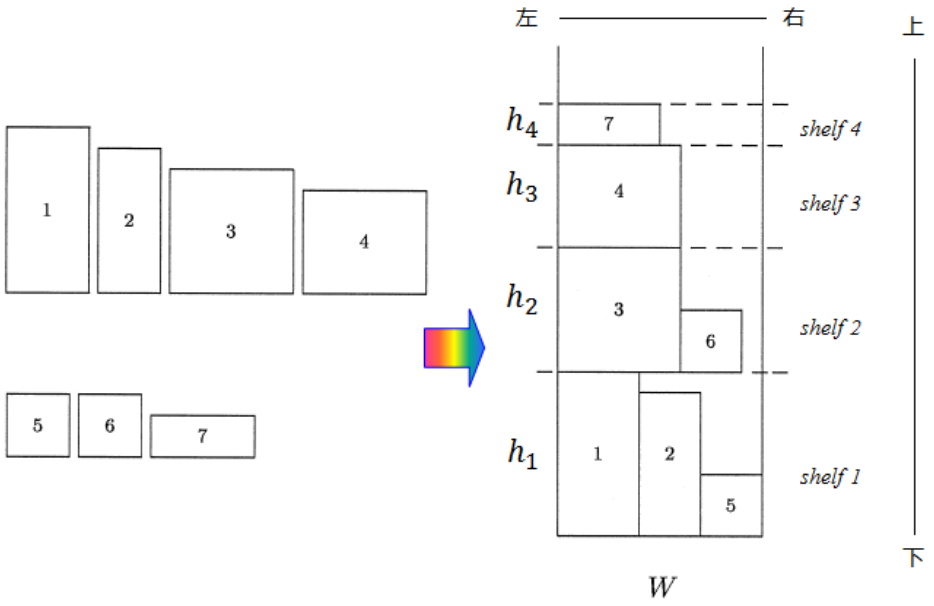
- 具有不同宽度和长度的石料需要用平板架运送到建筑工地。
- 具有不同宽度和长度的货品需要被摆放在仓库的货架上。
- 在金属制造工业中，特定形状的钢片需要从大块钢片中切出。
- 在木材制造行业中，特定形状的木板需要从原始大块木板上裁出。
- 在玻璃制造行业中，特定形状的玻璃板需要从原始大块玻璃材料上裁出。
- 包装材料裁切、服装布料裁切、皮鞋制作过程中考虑怎样裁切使得材料、布料、皮革浪费最少。
- 报纸杂志排版。
- 堆场中各功能区域划分。
- 停车场区位划分。

# 贪婪启发式算法

- 装箱问题属于NP-完全问题。目前最有效的计算方法是一些启发式算法。
- 在二维装箱问题中常用的启发式算法包括：
  - ***Shelf packing*** ***（SP）*** 算法
  - ***Normal position packing*** ***（NPP）*** 算法

# *Shelf packing* 算法

- 在*shelf packing*算法中物品被成排的装入箱子中，类似于将物品摆放在在一个一个“架子（*shelf*）”上。
- 显然，所有的架子的宽度都等于箱子的宽度，而每一个架子的高度由这一排物品中的第一个物品决定。

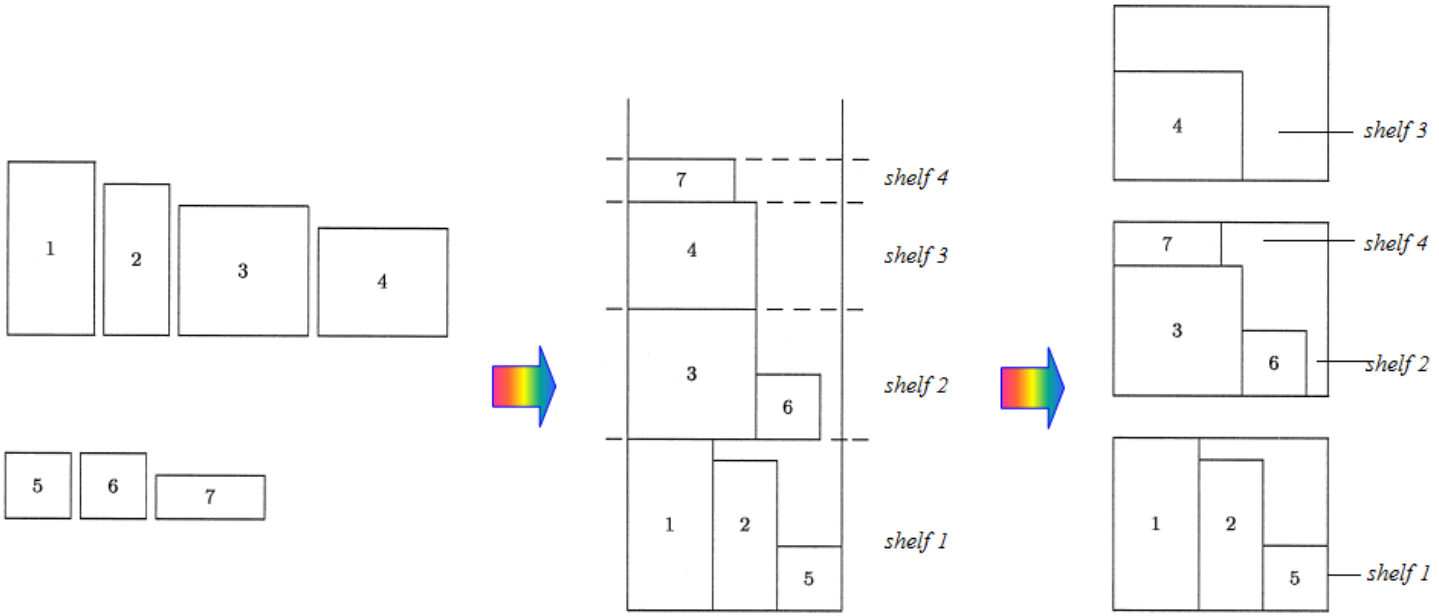


# *Shelf packing* 算法

- 两段式 *shelf packing* 算法
  - 步骤一：将所有物品摆放在不同的架子上
    - 假设存在*n*个宽度为*W*、高度待定的空架子，
      - ▲ 将所有物品按照长度递减的顺序排列（通常矩形两边中较长的一边为长度），并依次将物品按照最佳配合的方法或者优先配合的方法摆放在架子上。
      - ▲ 当一个架子上出现第一个物品时，这个架子的高度就被确定为与这个物品的长度相等。
      - ▲ 最佳配合方法：
        - ★ 如果当前待装物品（当前长度最长的物品）不可以被摆放在任何一个已经使用的高度一定的架子上，那么将此物品摆放在一个空架子上，并将这个新架子的高度设定为此物品的长度。
        - ★ 否则，选择一个已使用的架子摆放当前物品，使得架子的剩余宽度最小。

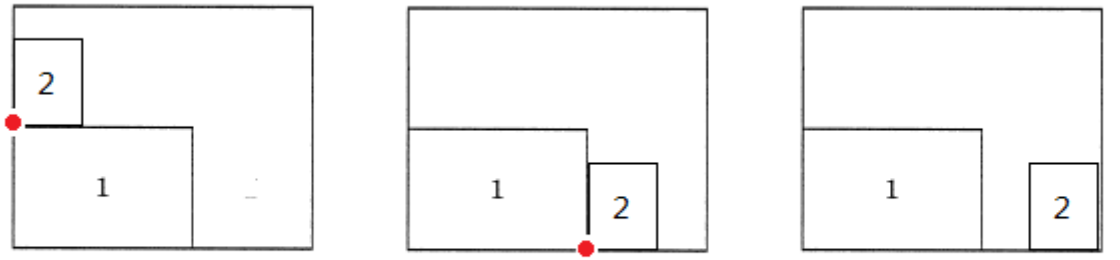
# *Shelf packing* 算法

- 两段式 ***shelf packing*** 算法
  - 步骤二：将所有摆放了物品的架子装入箱子中
    - 解一维装箱问题
- 时间复杂性：***O(n log n)***
- 算例：在步骤一和步骤二中都采用最佳配合方法



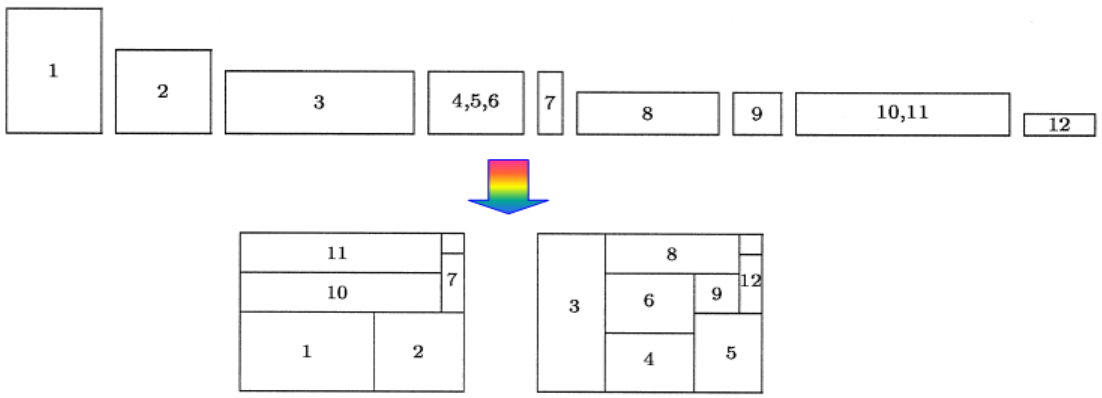
# *Normal position packing* 算法

- **Normal position：一个物品被放置在normal position时，这个物品的左边界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边界，而这个物品的下边界应该触碰箱子的下壁或者其他物品的上边界。**



# *Normal position packing* 算法

- 将所有物品按照面积递减的顺序排序，并依次装入箱子中
  - 如果当前物品无法被装入任何一个已使用的箱子，则初始化一个新箱子，并将该物品放在新箱子的左下角（新箱子中唯一的一个**normal position**）。
  - 否则，使用以下打分机制为每一个箱子中的**normal position**打分，并选择分数最高的一个**normal position**来放置此物品。
    - 打分机制：当一个物品被放置在一个**normal position**时，物品的边界与箱子或者其他物品的边界相接触，其中接触的最多的（或者接触率最高的）**normal position**分数最高。
      - 如果分数相同，那么选择剩余面积最小的箱子。
- 算例



# Tabu Search 求解示例

## 1. 初始化

1) 物品序列Seq←将给定的*n*个物品按照面积降序排列;
2) sol ← 将Seq作为输入，使用NPP算法求解初始解;
3) 当前最优解 best = sol;
4) 令禁忌表为空。

## 2. 判断算法终止条件是否满足？若是，则结束算法并输出结果best；否则，继续以下步骤。

# Tabu Search 求解示例

## 3. 从当前最优解的邻域中选择部分候选解，并更新最优解

1) 邻域操作：随机交换物品序列中的两个物品的位置；
2) 通过对物品序列Seq进行邻域操作获得若干个新的物品序列；
3) 把每一个新的物品序列当做输入，使用NPP算法求解。注意及时更新最优解best，更新Seq为最优解对应的物品序列，同时记录Seq是通过怎样的邻域操作得到的，也就是交换位置的物品对（p, q）。

# Tabu Search 求解示例

## 4. 更新禁忌表

1) 将步骤3中得到的物品对（p, q）添加入禁忌表中。
2) 如果禁忌表已满，则用（p, q）替换禁忌表中最早加入的物品对。

## 5. 转步骤 2。

# 二维装箱问题的变种问题

- **箱子尺寸不同的二维装箱问题(2D Variable-Sized Bin Packing Problem)：**
  - 寻找最优的装箱方案，使得装入同一个箱子的所有物品不会相互重叠，而使用的箱子的总面积最少。
- **箱子尺寸不同费用也不同的二维装箱问题(2D Bin Packing Problem with Variable Sizes and Costs)：**
  - 寻找最优的装箱方案，使得装入同一个箱子的所有物品不会相互重叠，而使用的箱子的总费用最小。
- **二维条装箱问题(2D Strip Packing Problem)：**
  - 将所有物品装入一个宽度一定但是长度无限制的长条中，使得所有物品不会重叠，而总长度最小。

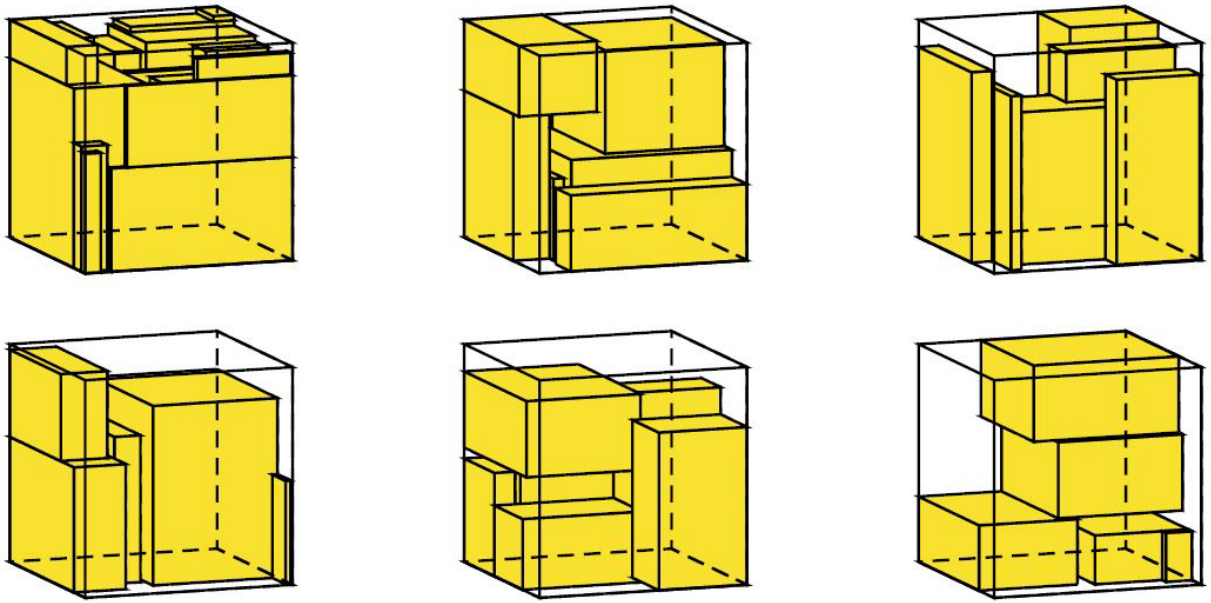
# 第六章 装箱问题及求解算法

## 装箱问题 (Bin-Packing Problem)

- 装箱问题实例与特征
- 装箱问题的数学描述
- 装箱问题的启发式算法
- 装箱问题的GA求解算法
- 三维装箱问题及启发式算法

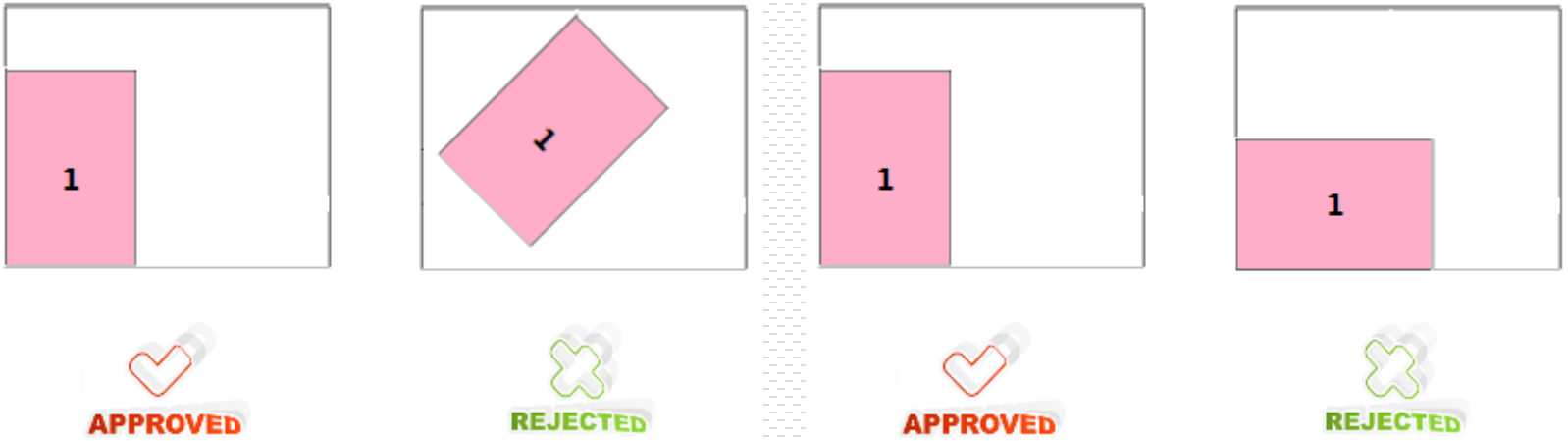
# 三维装箱问题及特征

- 给定 $m$ 个长方体箱子和 $n$ 个长方体物品（$m>=n$）
- 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。



# 三维装箱问题及特征

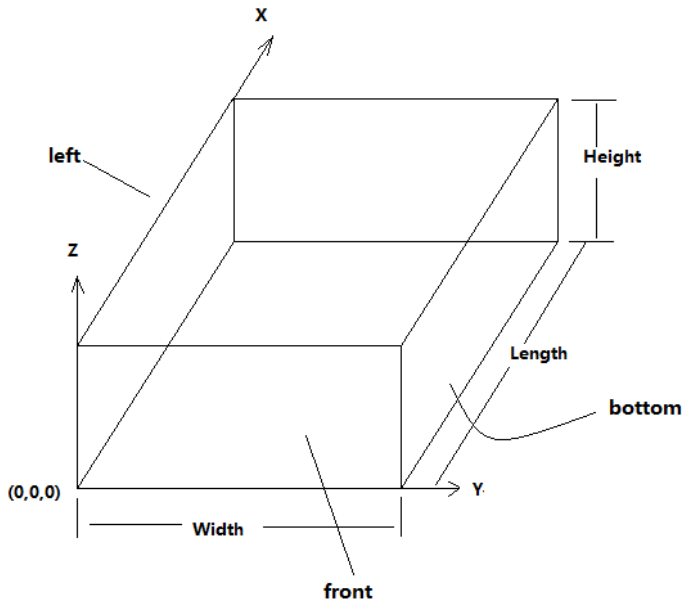
- 通常，所有箱子有相同的长宽高，$L_i = L, W_i = W, H_i = H, \forall\, i = 1,2,\ldots,m$。
- 物品的边缘应该平行于它所在的箱子的边缘。
- 在某些情况下，我们要求物品不可以旋转。
- 显然，如果每一个物品的高度都等于箱子的高度 $H$，那么三维装箱问题就被简化为二维装箱问题。




边缘平行　　不可旋转

# 模型描述

- 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。

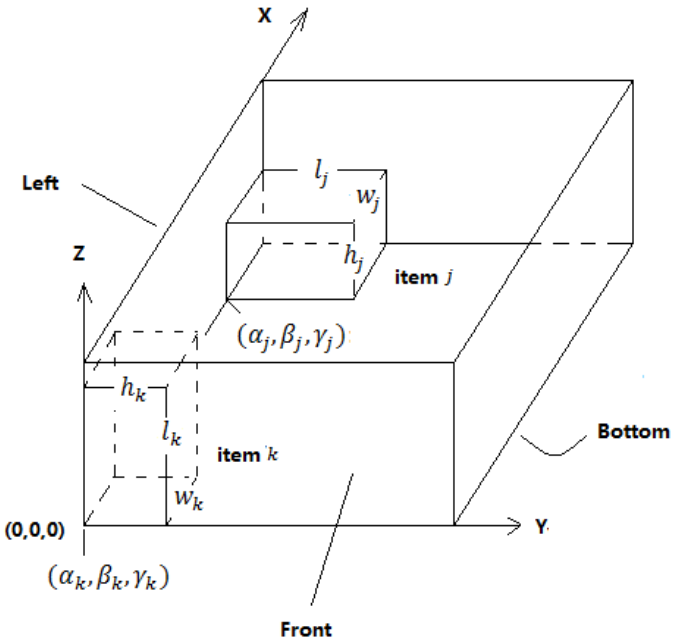


# 模型描述

- 数学模型中使用的变量
  - $y_i$: 0-1变量，$y_i = 1$表示箱子 $i$被使用，$y_i = 0$表示箱子 $i$没有被使用
  - $x_{ij}$: 0-1变量，$x_{ij} = 1$表示物品 $j$被装在箱子 $i$中，$x_{ij} = 0$表示物品 $j$没有被装在箱子$i$中
  - $(\alpha_j, \beta_j, \gamma_j)$: 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
  - $p_{\alpha j}$: 0-1变量，$p_{\alpha j} = 1$表示物品$j$的长平行于$X$轴
  - $p_{\beta j}$: 0-1变量，$p_{\beta j}: = 1$表示物品$j$的长平行于$Y$轴
  - $p_{\gamma j}$: 0-1变量，$p_{\gamma j} = 1$表示物品$j$的长平行于$Z$轴
  - $q_{\alpha j}$: 0-1变量，$q_{\alpha j} = 1$表示物品$j$的宽平行于$X$轴
  - $q_{\beta j}$: 0-1变量，$q_{\beta j} = 1$表示物品$j$的宽平行于$Y$轴
  - $q_{\gamma j}$: 0-1变量，$q_{\gamma j} = 1$表示物品$j$的宽平行于$Z$轴
  - $r_{\alpha j}$: 0-1变量，$r_{\alpha j} = 1$表示物品$j$的高平行于$X$轴
  - $r_{\beta j}$: 0-1变量，$r_{\beta j} = 1$表示物品$j$的高平行于$Y$轴
  - $r_{\gamma j}$: 0-1变量，$r_{\gamma j} = 1$表示物品$j$的高平行于$Z$轴

# 模型描述

- $a_{jk}$: 0-1变量，$a_{jk}=1$表示物品$j$在物品$k$的前面，准确来说 $a_{jk}=1$表示 $\alpha_j < \alpha_k$
- $b_{jk}$: 0-1变量，$b_{jk}=1$表示物品$j$在物品$k$的后面，准确来说 $b_{jk}=1$表示 $\alpha_j > \alpha_k$
- $c_{jk}$: 0-1变量，$c_{jk}=1$表示物品$j$在物品$k$的左面，准确来说 $c_{jk}=1$表示 $\beta_j < \beta_k$
- $d_{jk}$: 0-1变量，$d_{jk}=1$表示物品$j$在物品$k$的右面，准确来说 $d_{jk}=1$表示 $\beta_j > \beta_k$
- $e_{jk}$: 0-1变量，$e_{jk}=1$表示物品$j$在物品$k$的下面，准确来说 $e_{jk}=1$表示 $\gamma_j < \gamma_k$
- $f_{jk}$: 0-1变量，$f_{jk}=1$表示物品$j$在物品$k$的上面，准确来说 $f_{jk}=1$表示 $\gamma_j > \gamma_k$




$$p_{\alpha j}=0, q_{\alpha j}=1, r_{\alpha j}=0$$

$$p_{\beta j}=1, q_{\beta j}=0, r_{\beta j}=0$$

$$p_{\gamma j}=0, q_{\gamma j}=0, r_{\gamma j}=1$$

$$p_{\alpha k}=0, q_{\alpha k}=1, r_{\alpha k}=0$$

$$p_{\beta k}=0, q_{\beta k}=0, r_{\beta k}=1$$

$$p_{\gamma k}=1, q_{\gamma k}=0, r_{\gamma k}=0$$

$$a_{jk}=0, b_{jk}=1$$

$$c_{jk}=0, d_{jk}=0$$

$$e_{jk}=0, f_{jk}=0$$

# 模型描述

1. $\min \quad f(\boldsymbol{y}) = \sum_{i=1}^{m} y_i$
2. $\text{s.t.} \sum_{i=1}^{m} x_{ij} = 1, \forall\, j \in N = \{1,2,\dots,n\}$
3. $\sum_{j=1}^{n} x_{ij} \le M y_i, \forall\, i \in I = \{1,2,\dots,m\}$
4. $\alpha_j + l_j p_{\alpha j} + w_j q_{\alpha j} + h_j r_{\alpha j} \le \alpha_k + (1 - a_{jk})M, \forall\, j \in N, i \in I$
5. $\alpha_k + l_k p_{\alpha k} + w_k q_{\alpha k} + h_k r_{\alpha k} \le a_j + (1 - b_{jk})M, \forall\, j \in N, i \in I$
6. $\beta_j + l_j p_{\beta j} + w_j q_{\beta j} + h_j r_{\beta j} \le \beta_k + (1 - c_{jk})M, \forall\, j \in N, i \in I$
7. $\beta_k + l_k p_{\beta k} + w_k q_{\beta k} + h_k r_{\beta k} \le \beta_j + (1 - d_{jk})M, \forall\, j \in N, i \in I$
8. $\gamma_j + l_j p_{\gamma j} + w_j q_{\gamma j} + h_j r_{\gamma j} \le \gamma_k + (1 - e_{jk})M, \forall\, j \in N, i \in I$
9. $\gamma_k + l_k p_{\gamma k} + w_k q_{\gamma k} + h_k r_{\gamma k} \le \gamma_j + (1 - f_{jk})M, \forall\, j \in N, i \in I$
10. $a_{jk} + b_{jk} + c_{jk} + d_{jk} + e_{jk} + f_{jk} \ge x_{ij} + x_{ik} - 1, \forall\, j \in N, i \in I$
11. $\alpha_j + l_j p_{\alpha j} + w_j q_{\alpha j} + h_j r_{\alpha j} \le L_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
12. $\beta_j + l_j p_{\beta j} + w_j q_{\beta j} + h_j r_{\beta j} \le W_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
13. $\gamma_j + l_j p_{\gamma j} + w_j q_{\gamma j} + h_j r_{\gamma j} \le H_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
14. $p_{\alpha j} + p_{\beta j} + p_{\gamma j} = 1, \forall\, j \in N$
15. $q_{\alpha j} + q_{\beta j} + q_{\gamma j} = 1, \forall\, j \in N$
16. $r_{\alpha j} + r_{\beta j} + r_{\gamma j} = 1, \forall\, j \in N$
17. $p_{\alpha j} + q_{\alpha j} + r_{\alpha j} = 1, \forall\, j \in N$
18. $p_{\beta j} + q_{\beta j} + r_{\beta j} = 1, \forall\, j \in N$
19. $p_{\gamma j} + q_{\gamma j} + r_{\gamma j} = 1, \forall\, j \in N$
20. $\alpha_j \ge 0, \beta_j \ge 0, \gamma_j \ge 0, \forall\, j \in N$
21. $p_{\alpha j}, p_{\beta j}, p_{\gamma j}, q_{\alpha j}, q_{\beta j}, q_{\gamma j}, r_{\alpha j}, r_{\beta j}, r_{\gamma j}, a_{jk}, b_{jk}, c_{jk}, d_{jk}, e_{jk}, f_{jk} \in \{0,1\}, \forall\, j,k \in N, i \in I$

***M is a large constant***

# 集合覆盖模型

- 装箱方案 (a packing pattern)：如果一组物品可以装入一个箱子，那么我们称这组物品为一个装箱方案。
- 这样的packing pattern的数量 $K$ 可能是指数级的。
- 我们用长度为 $n$的列向量 $A^k$表示一个packing pattern：$A^k$的每一个元素 $a_j^k$指示物品$j$ 是否在这个packing pattern中，如果存在则 $a_j^k = 1$，否则 $a_j^k = 0$。
- 由所有packing pattern的列向量组成的矩阵可以表示为 $A$。

$$\min \ f(\boldsymbol{x}) = \sum_{k=1}^{K} x_k$$

$$\text{s.t.} \sum_{k=1}^{K} a_j^k x_k = 1, \forall\, j \in \{1,2,\dots,n\}$$

$$x_k \in \{0,1\}, \ \forall\, k \in \{1,2,\dots,K\}$$

- $x_k = 1$表示第 $k$个packing pattern出现在解中，否则 $x_k = 0$

# 特点

- NP 难问题
- 不同于一般的整数规划问题
- 仅有一些启发式算法可以在有限的计算空间和时间约束下得到较好的解

# 应用

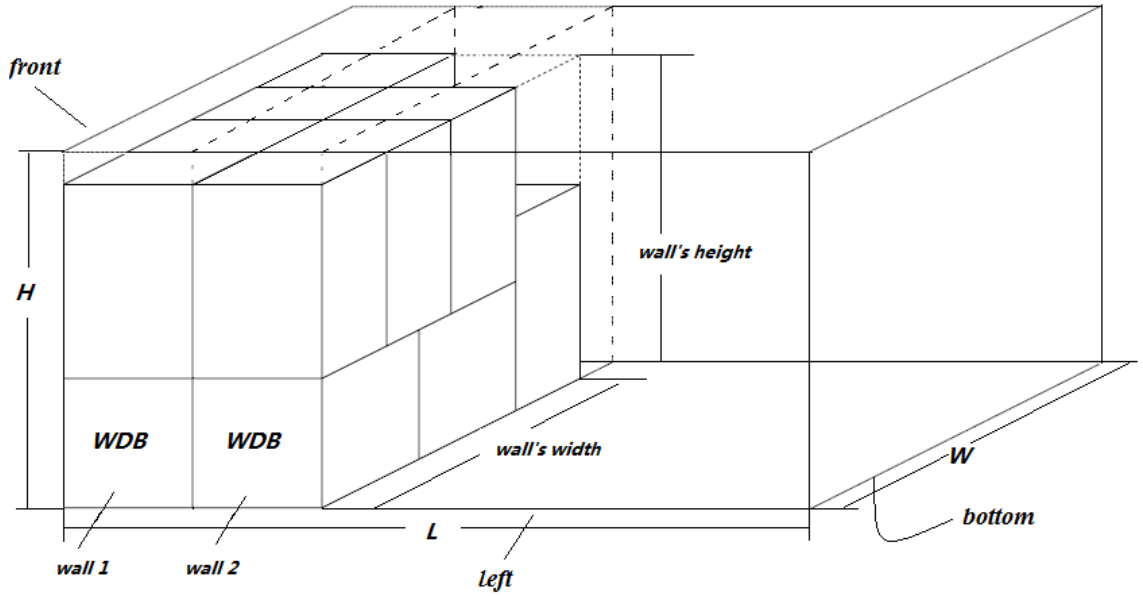
- 具有不同长宽高的石料需要用平板架运送到建筑工地。
- 具有不同长宽高的货品需要被摆放在仓库的货架上。
- 在家具制造行业中，特定形状的3D模板需要从大块塑料泡沫中裁出。
- 在金属制造工业中，特定形状的钢块需要从大块钢材中切出。
- 在木材制造行业中，特定形状的木块需要从原始木材上裁出。
- 在供应链或运输行业中，将不同形状的货物装入集装箱。

# 贪婪启发式算法

- 装箱问题属于NP-完全问题。目前最有效的计算方法是一些启发式算法。
- 在三维装箱问题中常用的启发式算法包括：
  - ***Wall building*** 算法
  - ***Tower building*** 算法
  - ***Corner position packing*** （***CPP***）算法

# *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-下位置上的物品中决定。我们把这个物品称为建造这面墙的关键物品 (*Wall Determining Box，WDB*)。

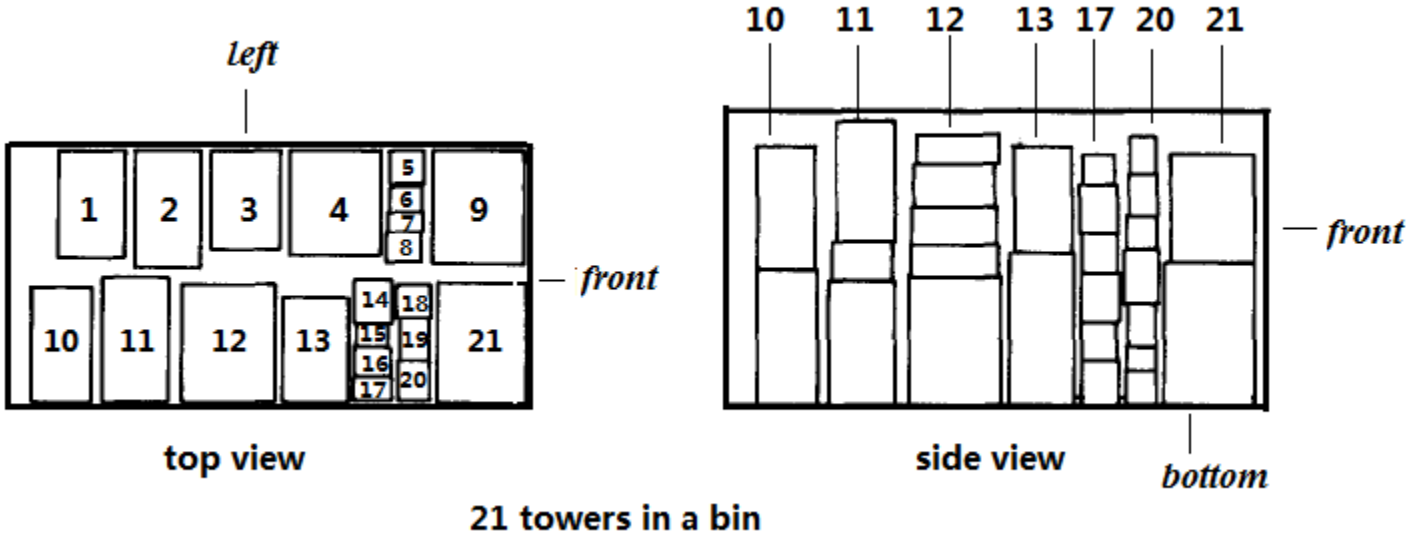


# *Wall building*算法

- 两段式 *Wall building* 算法
  - 步骤一：将所有物品堆砌成宽度小于*W*、高度小于*H*的墙
    - 将所有物品按照体积递减的顺序排列，并依次将物品按照优先配合的方法堆放成一面一面物品墙。
    - 虽然物品可以旋转，但是关键物品必须以长度平行于X轴的方式放入墙中。所以每一面墙的长度等于其关键物品的长度。
    - 优先配合方法：
      - 如果没有任何一个已经建立起来的物品墙可以容纳当前待装物品（当前体积最大的物品），那么将此物品作为关键物品并开启一面新墙。
      - 否则，将当前物品放入第一面可以容纳它的墙中，并且使得这个物品在墙中的位置尽可能靠近前面、左面和下面。
  - 步骤二：将所有物品墙装入箱子中
    - 解一维装箱问题

# *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*）。

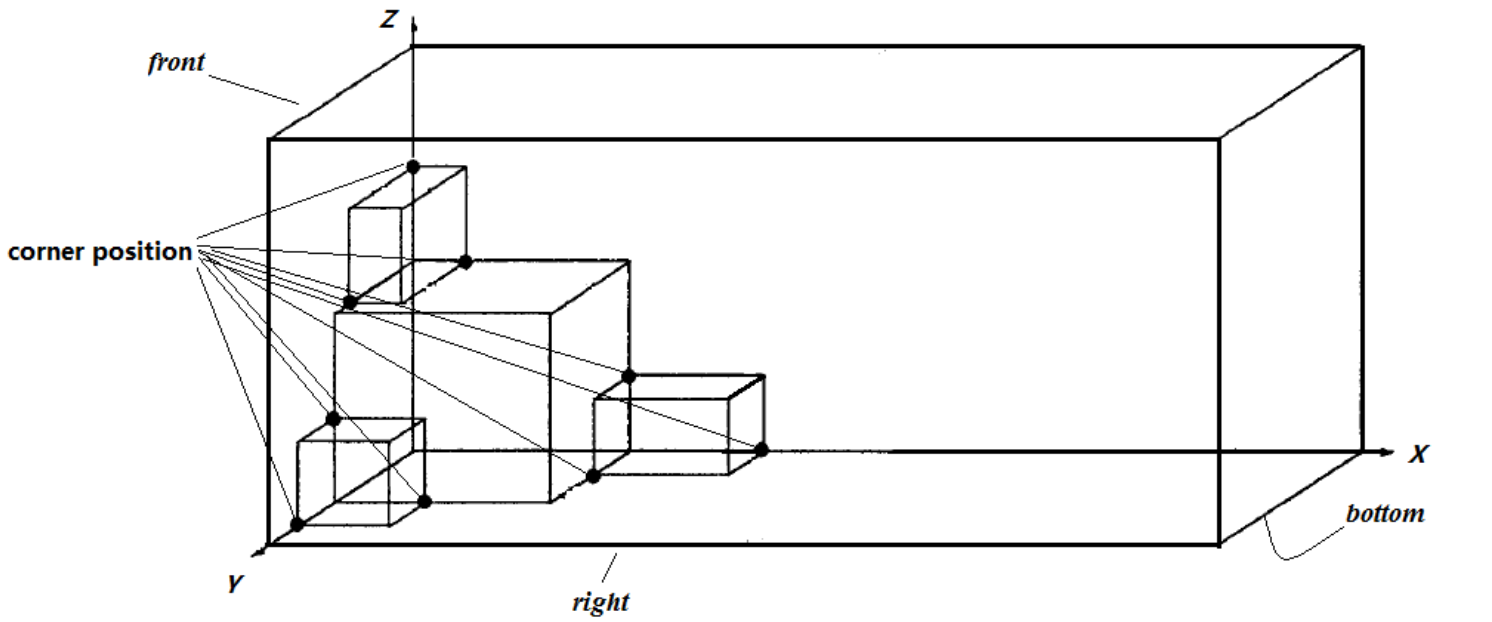


# *Tower building* 算法

- 两段式 ***Tower building*** 算法
  - 步骤一：将所有物品堆成高度小于$H$ 的塔
    - 将所有物品按照体积递减的顺序排列，并依次将物品按照优先配合的方法堆放成一座一座的物品塔。
    - 虽然物品可以旋转，但是关键物品必须以面积最大的平面朝下的方式摆放。所以每一座塔的底面应该是其关键物品六个面中最大的面。
    - 优先配合方法：
      - 如果没有任何一个已经建立起来的物品塔可以容纳当前待装物品（当前体积最大的物品），那么将此物品作为关键物品并开启一座新塔。
      - 否则，将当前物品放入第一座可以容纳它的塔中，并且使得这个物品在塔中的位置尽可能靠近前面、左面和下面。
  - 步骤二：将所有物品塔装入箱子中
    - 解二维装箱问题

# 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。**



# *Corner position packing* 算法

- 将所有物品依次装入箱子中：
  - 如果当前物品无法被装入任何一个已使用的箱子，则初始化一个新箱子，并将该物品放在新箱子的前-左-下角（新箱子中唯一的一个corner position）。
  - 否则，使用以下打分机制为每一个箱子中的corner position打分，并选择分数最高的一个corner position来放置此物品。
    - 打分机制：当一个物品被放置在一个corner position时，物品的前壁、左壁和下壁与箱子或者其他物品的边界相接触，其中接触的最多的（或者接触率最高的）corner position分数最高。
      - 如果分数相同，那么选择剩余体积最小的箱子。

# 三维装箱问题的变种问题

- **箱子尺寸不同的三维装箱问题(3D Variable-Sized Bin Packing Problem/ Multiple Container Loading Problem):**
  - 寻找最优的装箱方案，使得装入同一个箱子的所有物品不会相互重叠，而使用的箱子的总体积最少。
- **箱子尺寸不同费用也不同的三维装箱问题(3D Bin Packing Problem with Variable Sizes and Costs/ Multiple Container Loading Cost Minimization Problem):**
  - 寻找最优的装箱方案，使得装入同一个箱子的所有物品不会相互重叠，而使用的箱子的总费用最小。
- **三维条装箱问题(3D Strip Packing Problem):**
  - 将所有物品装入一个长度和宽度固定但是高度无限制的长条中，使得所有物品不会重叠，而总长度最小。

# 三维装箱问题的变种问题

- 三维装箱问题在货物运输行业非常常见，但是一般的三维装箱算法无法用于实际生产生活，因为货物装箱问题中有很多特殊情况和限制条件，比如：
  - 并不是所有的物品都是长方体
  - 重量限制(weight limitation)：将货物装入卡车时除了要考虑空间几何限制外，还要考虑卡车的限重
  - 重量分布(weight distribution)：为了防止翻车，装货物时要考虑水平轴和垂直轴上的重量分布
  - 货物自身的承重能力(loading bearing strength)：重物不可以压在易碎物品上
  - 货物稳定性(stability constraint)：在运输易碎物品时由其重要，如果物品之间的缝隙太大，卡车运行过程中的颠簸会造成货物的移动甚至跌落

# End

研究生课程讲义

# 典型优化问题的模型与算法

## (Models and Algorithms for Typical Optimization Problems)

东北财经大学 管理科学与工程学院

感谢东北大学系统工程研究所课程组

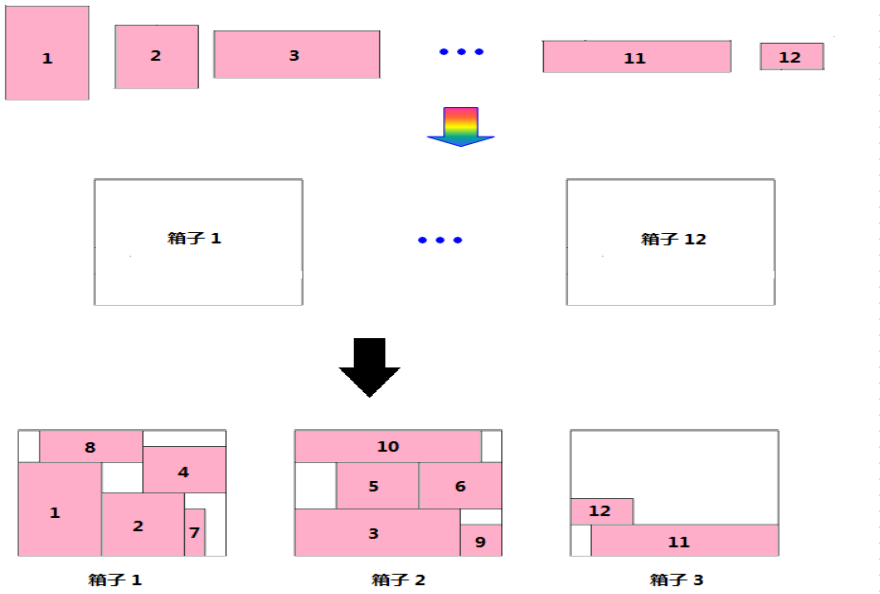
# 第六章 装箱问题及求解算法

## 装箱问题 (Bin-Packing Problem)

- 装箱问题实例与特征
- 装箱问题的数学描述
- 装箱问题的分类与变种
- 装箱问题的启发式算法
- 装箱问题的GA求解算法

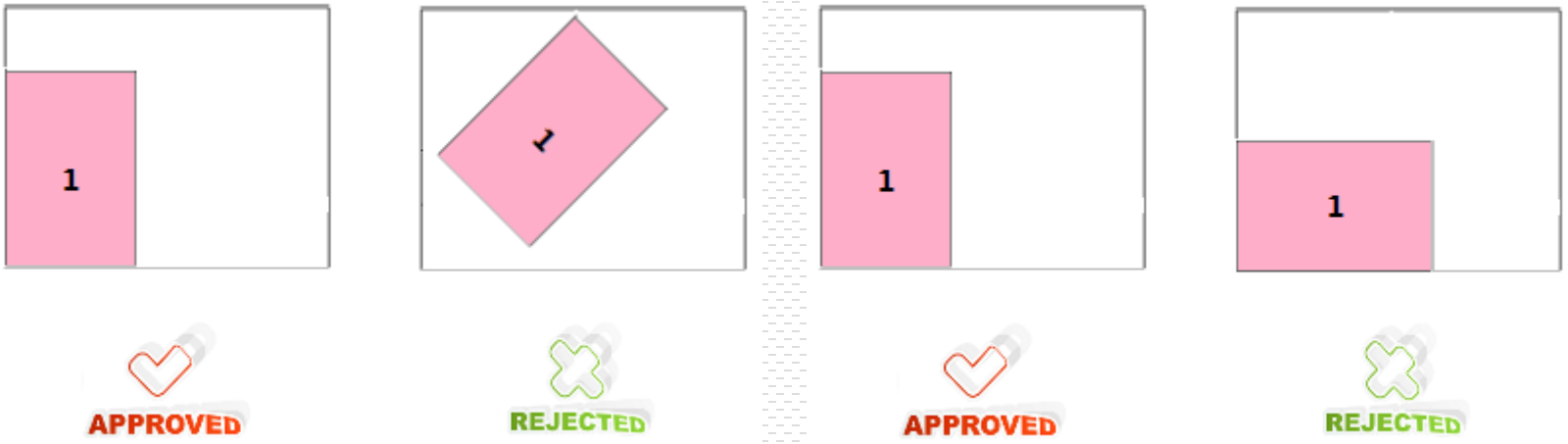
# 二维装箱问题及特征

- 给定 $m$ 个矩形箱子和 $n$ 个矩形物品（$m>=n$）
- 每个物品有长度（$l_j>0$）和宽度（$w_j>0$），每个箱子也有长度（$L_i>0$）和宽度（$W_i>0$）。
- 问题是：寻找最优的方案将矩形物品分配到矩形箱子中，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。



# 二维装箱问题及特征

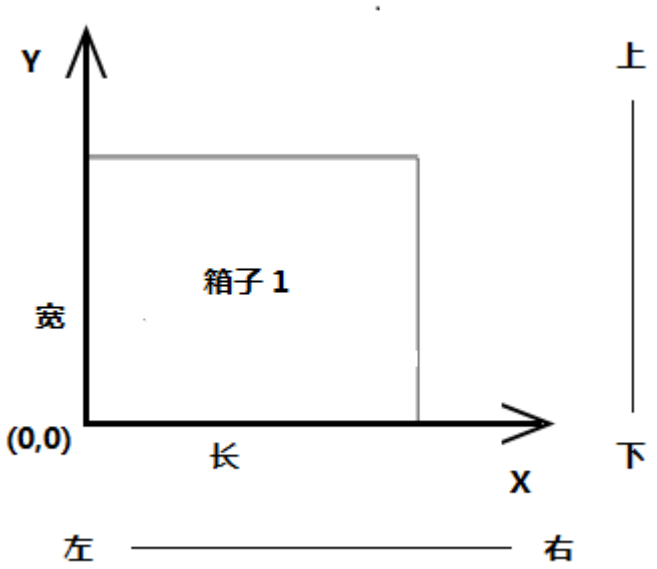
- 通常，所有箱子有相同的长度和宽度，$L_i = L, W_i = W, \forall\, i = 1,2,\ldots,m$。
- 物品的边缘应该平行于它所在的箱子的边缘。
- 在某些情况下，我们要求物品不可以旋转90度。
- 显然，如果每一个物品的宽度都等于箱子的宽度 $W$，那么二维装箱问题就被简化为一维装箱问题。




边缘平行　　不可旋转

# 模型描述

- 假设
  - 每一个箱子都被放置在一个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，箱子的左下角与坐标原点重合。

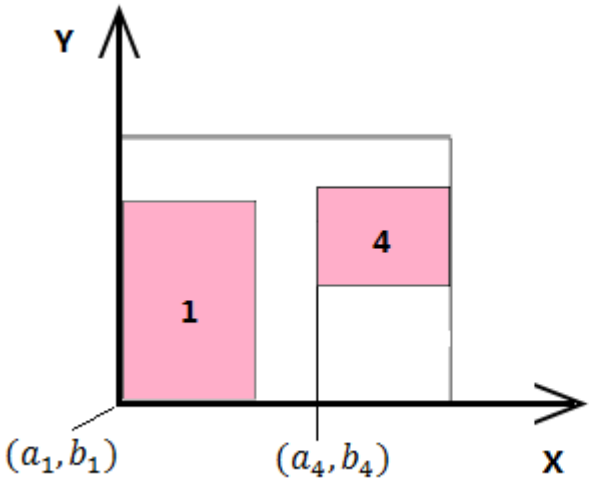


# 模型描述

- 数学模型中使用的变量
  - $y_i$: 0-1变量，$y_i = 1$ 表示箱子 $i$ 被使用，$y_i = 0$ 表示箱子 $i$ 没有被使用
  - $x_{ij}$: 0-1变量，$x_{ij} = 1$ 表示物品 $j$ 被装在箱子 $i$ 中，$x_{ij} = 0$ 表示物品 $j$ 没有被装在箱子 $i$ 中
  - $(a_j, b_j)$: 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
  - $p_{aj}$: 0-1变量，$p_{aj} = 1$ 表示物品 $j$ 的长平行于 $X$ 轴
  - $p_{bj}$: 0-1变量，$p_{bj} = 1$ 表示物品 $j$ 的长平行于 $Y$ 轴
  - $q_{aj}$: 0-1变量，$q_{aj} = 1$ 表示物品 $j$ 的宽平行于 $X$ 轴
  - $q_{bj}$: 0-1变量，$q_{bj} = 1$ 表示物品 $j$ 的宽平行于 $Y$ 轴

# 模型描述

- $c_{jk}$: 0-1变量，$c_{jk}=1$ 表示物品 $j$ 在物品$k$的左面 $(a_j < a_k)$
- $d_{jk}$: 0-1变量，$d_{jk}=1$ 表示物品 $j$ 在物品$k$的右面 $(a_j > a_k)$
- $e_{jk}$: 0-1变量，$e_{jk}=1$ 表示物品 $j$ 在物品$k$的下面 $(b_j < b_k)$
- $f_{jk}$: 0-1变量，$f_{jk}=1$ 表示物品 $j$ 在物品$k$的上面 $(b_j > b_k)$




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

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

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

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

# 模型描述

1. **min** $f(\boldsymbol{y}) = \sum_{i=1}^{m} y_i$
2. **s.t.** $\sum_{i=1}^{m} x_{ij} = 1, \forall\, j \in N = \{1,2,\dots,n\}$
3. $\sum_{j=1}^{n} x_{ij} \le M y_i, \forall\, i \in I = \{1,2,\dots,m\}$
4. $a_j + l_j p_{aj} + w_j q_{aj} \le a_k + (1 - c_{jk})M, \forall\, j \in N, i \in I$
5. $a_k + l_k p_{ak} + w_k q_{ak} \le a_j + (1 - d_{jk})M, \forall\, j \in N, i \in I$
6. $b_j + l_j p_{bj} + w_j q_{bj} \le b_k + (1 - e_{jk})M, \forall\, j \in N, i \in I$
7. $b_k + l_k p_{bk} + w_k q_{bk} \le b_j + (1 - f_{jk})M, \forall\, j \in N, i \in I$
8. $c_{jk} + d_{jk} + e_{jk} + f_{jk} \ge x_{ij} + x_{ik} - 1, \forall\, j \in N, i \in I$
9. $a_j + l_j p_{aj} + w_j q_{aj} \le L_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
10. $b_j + l_j p_{bj} + w_j q_{bj} \le W_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
11. $p_{aj} + p_{bj} = 1, \forall\, j \in N$
12. $q_{aj} + q_{bj} = 1, \forall\, j \in N$
13. $p_{aj} + q_{aj} = 1, \forall\, j \in N$
14. $p_{bj} + q_{bj} = 1, \forall\, j \in N$
15. $a_j \ge 0, b_j \ge 0, \forall\, j \in N$
16. $p_{aj}, p_{bj}, q_{aj}, q_{bj}, c_{jk}, d_{jk}, e_{jk}, f_{jk} \in \{0,1\}, \forall\, j,k \in N, i \in I$

***M is a large constant***

# 集合覆盖模型

- 装箱方案 (a packing pattern)：如果一组物品可以装入一个箱子，那么我们称这组物品为一个装箱方案。
- 这样的packing pattern的数量 $K$ 可能是指数级的。
- 我们用长度为 $n$ 的列向量 $A^k$ 表示一个packing pattern：$A^k$ 的每一个元素 $a_j^k$ 指示物品 $j$ 是否在这个packing pattern中，如果存在则 $a_j^k = 1$，否则 $a_j^k = 0$。
- 由所有packing pattern的列向量组成的矩阵可以表示为 $A$。

$$\min \ f(\boldsymbol{x}) = \sum_{k=1}^{K} x_k$$

$$\text{s.t.} \quad \sum_{k=1}^{K} a_j^k x_k = 1, \forall\, j \in \{1,2,\dots,n\}$$

$$x_k \in \{0,1\}, \ \forall\, k \in \{1,2,\dots,K\}$$

- $x_k = 1$ 表示第 $k$ 个packing pattern出现在解中，否则 $x_k = 0$

# 特点

- NP 难问题
- 不同于一般的整数规划问题
- 仅有一些启发式算法可以在有限的计算空间和时间约束下得到较好的解

# 应用

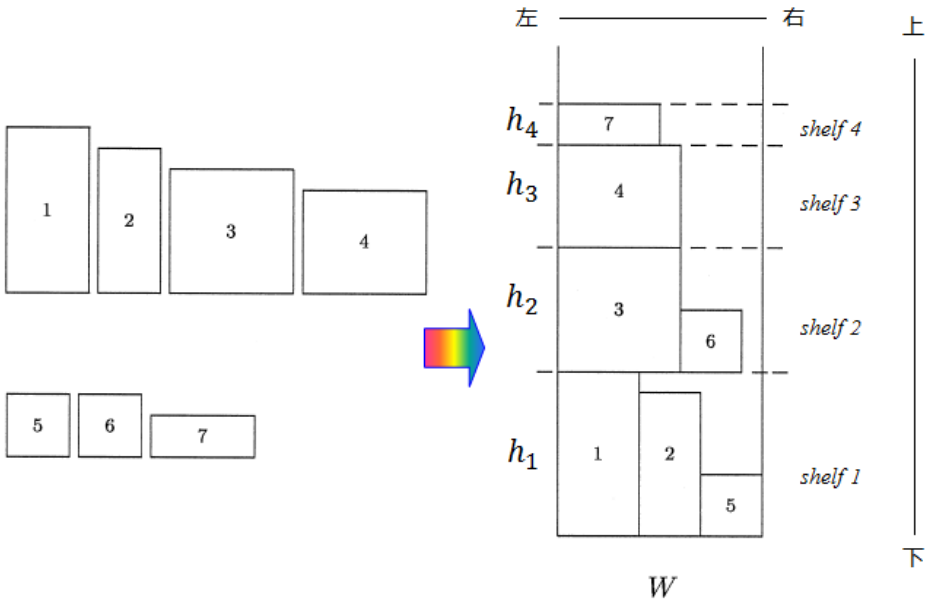
- 具有不同宽度和长度的石料需要用平板架运送到建筑工地。
- 具有不同宽度和长度的货品需要被摆放在仓库的货架上。
- 在金属制造工业中，特定形状的钢片需要从大块钢片中切出。
- 在木材制造行业中，特定形状的木板需要从原始大块木板上裁出。
- 在玻璃制造行业中，特定形状的玻璃板需要从原始大块玻璃材料上裁出。
- 包装材料裁切、服装布料裁切、皮鞋制作过程中考虑怎样裁切使得材料、布料、皮革浪费最少。
- 报纸杂志排版。
- 堆场中各功能区域划分。
- 停车场区位划分。

# 贪婪启发式算法

- 装箱问题属于NP-完全问题。目前最有效的计算方法是一些启发式算法。
- 在二维装箱问题中常用的启发式算法包括：
  - ***Shelf packing*** ***（SP）*** 算法
  - ***Normal position packing*** ***（NPP）*** 算法

# *Shelf packing* 算法

- 在*shelf packing*算法中物品被成排的装入箱子中，类似于将物品摆放在在一个一个“架子（*shelf*）”上。
- 显然，所有的架子的宽度都等于箱子的宽度，而每一个架子的高度由这一排物品中的第一个物品决定。

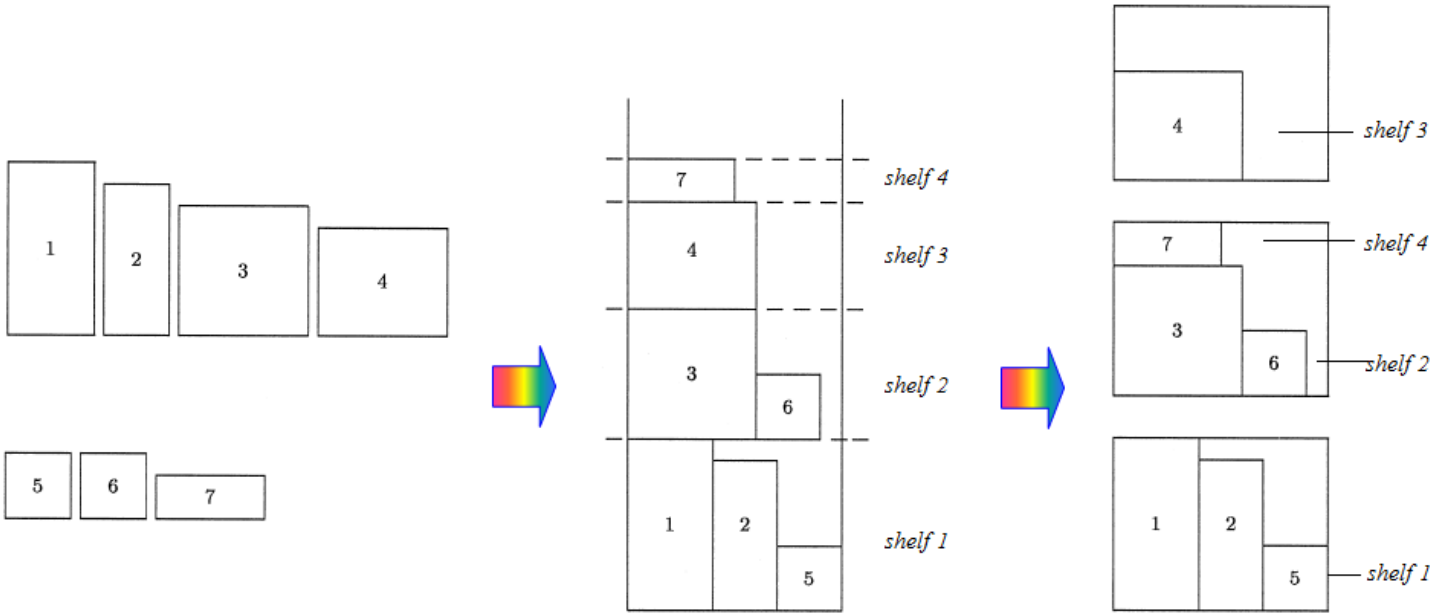


# *Shelf packing* 算法

- 两段式 *shelf packing* 算法
  - 步骤一：将所有物品摆放在不同的架子上
    - 假设存在*n*个宽度为*W*、高度待定的空架子，
      - ▲ 将所有物品按照长度递减的顺序排列（通常矩形两边中较长的一边为长度），并依次将物品按照最佳配合的方法或者优先配合的方法摆放在架子上。
      - ▲ 当一个架子上出现第一个物品时，这个架子的高度就被确定为与这个物品的长度相等。
      - ▲ 最佳配合方法：
        - ★ 如果当前待装物品（当前长度最长的物品）不可以被摆放在任何一个已经使用的高度一定的架子上，那么将此物品摆放在一个空架子上，并将这个新架子的高度设定为此物品的长度。
        - ★ 否则，选择一个已使用的架子摆放当前物品，使得架子的剩余宽度最小。

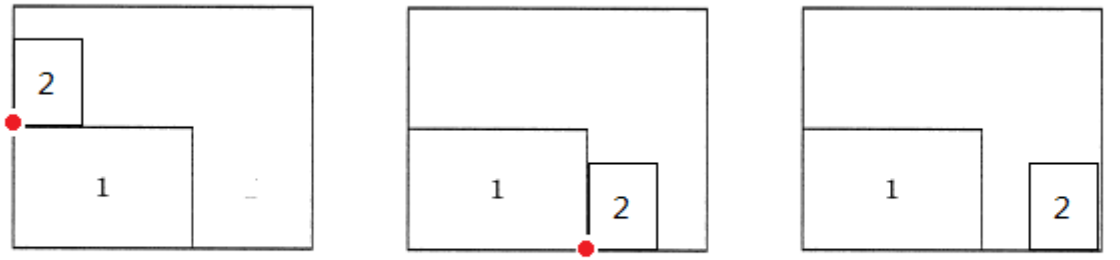
# *Shelf packing* 算法

- 两段式 ***shelf packing*** 算法
  - 步骤二：将所有摆放了物品的架子装入箱子中
    - 解一维装箱问题
- 时间复杂性：***O(n log n)***
- 算例：在步骤一和步骤二中都采用最佳配合方法



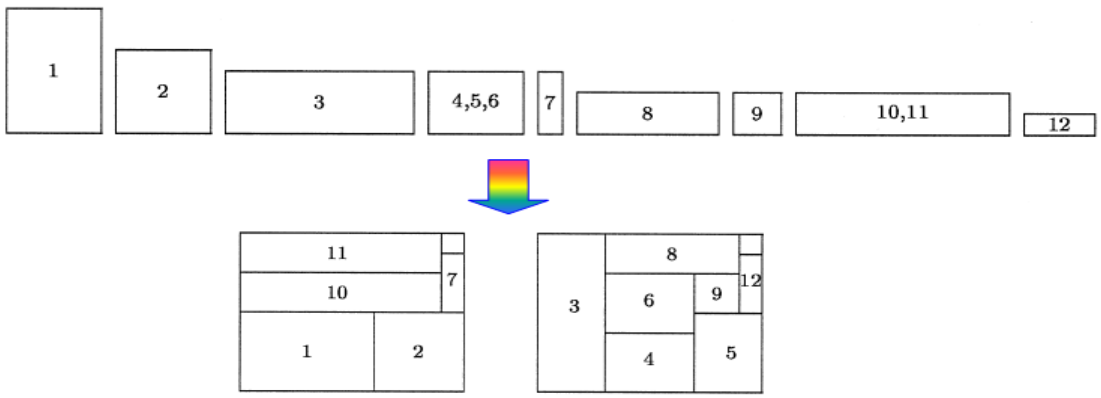
# *Normal position packing* 算法

- **Normal position：一个物品被放置在normal position时，这个物品的左边界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边界，而这个物品的下边界应该触碰箱子的下壁或者其他物品的上边界。**



# Normal position packing 算法

- 将所有物品按照面积递减的顺序排序，并依次装入箱子中
  - 如果当前物品无法被装入任何一个已使用的箱子，则初始化一个新箱子，并将该物品放在新箱子的左下角（新箱子中唯一的一个**normal position**）。
  - 否则，使用以下打分机制为每一个箱子中的**normal position**打分，并选择分数最高的一个**normal position**来放置此物品。
    - 打分机制：当一个物品被放置在一个**normal position**时，物品的边界与箱子或者其他物品的边界相接触，其中接触的最多的（或者接触率最高的）**normal position**分数最高。
      - 如果分数相同，那么选择剩余面积最小的箱子。
- 算例



# Tabu Search 求解示例

## 1. 初始化

1) 物品序列Seq←将给定的*n*个物品按照面积降序排列;
2) sol ← 将Seq作为输入，使用NPP算法求解初始解;
3) 当前最优解 best = sol;
4) 令禁忌表为空。

## 2. 判断算法终止条件是否满足？若是，则结束算法并输出结果best；否则，继续以下步骤。

# Tabu Search 求解示例

## 3. 从当前最优解的邻域中选择部分候选解，并更新最优解

1) 邻域操作：随机交换物品序列中的两个物品的位置；
2) 通过对物品序列Seq进行邻域操作获得若干个新的物品序列；
3) 把每一个新的物品序列当做输入，使用NPP算法求解。注意及时更新最优解best，更新Seq为最优解对应的物品序列，同时记录Seq是通过怎样的邻域操作得到的，也就是交换位置的物品对（p, q）。

# Tabu Search 求解示例

## 4. 更新禁忌表

1) 将步骤3中得到的物品对（p, q）添加入禁忌表中。
2) 如果禁忌表已满，则用（p, q）替换禁忌表中最早加入的物品对。

## 5. 转步骤 2。

# 二维装箱问题的变种问题

- **箱子尺寸不同的二维装箱问题(2D Variable-Sized Bin Packing Problem)：**
  - 寻找最优的装箱方案，使得装入同一个箱子的所有物品不会相互重叠，而使用的箱子的总面积最少。
- **箱子尺寸不同费用也不同的二维装箱问题(2D Bin Packing Problem with Variable Sizes and Costs)：**
  - 寻找最优的装箱方案，使得装入同一个箱子的所有物品不会相互重叠，而使用的箱子的总费用最小。
- **二维条装箱问题(2D Strip Packing Problem)：**
  - 将所有物品装入一个宽度一定但是长度无限制的长条中，使得所有物品不会重叠，而总长度最小。

研究生课程讲义

# 典型优化问题的模型与算法

## (Models and Algorithms for Typical Optimization Problems)

东北财经大学 管理科学与工程学院

感谢东北大学系统工程研究所课程组

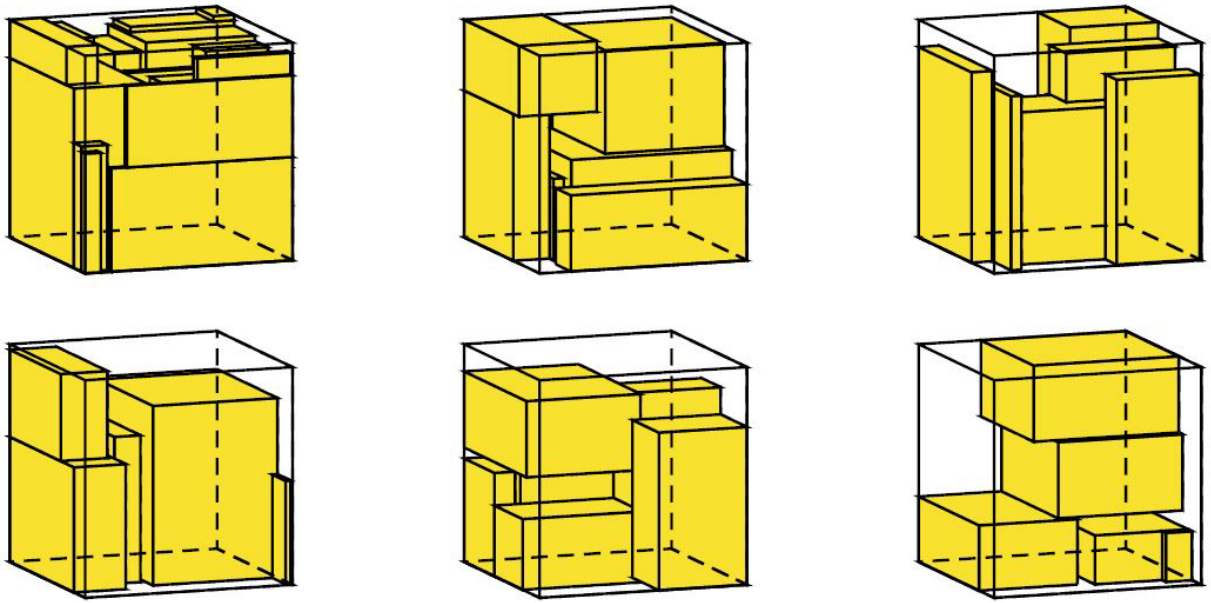
# 第六章 装箱问题及求解算法

## 装箱问题 (Bin-Packing Problem)

- 装箱问题实例与特征
- 装箱问题的数学描述
- 装箱问题的分类与变种
- 装箱问题的启发式算法
- 装箱问题的GA求解算法

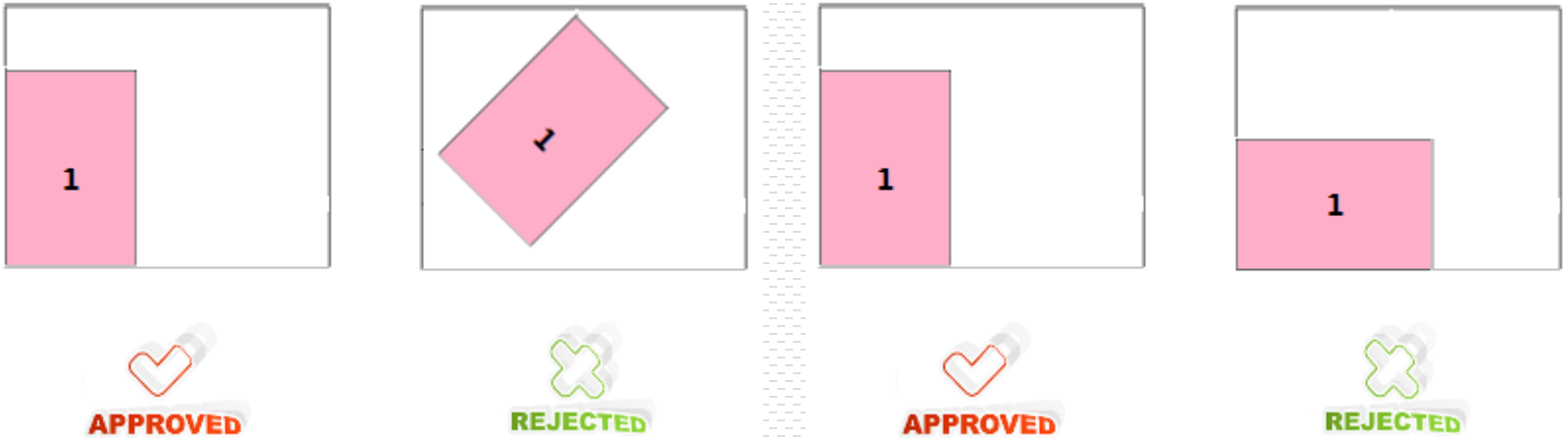
# 三维装箱问题及特征

- 给定 $m$ 个长方体箱子和 $n$ 个长方体物品（$m>=n$）
- 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。



# 二维装箱问题及特征

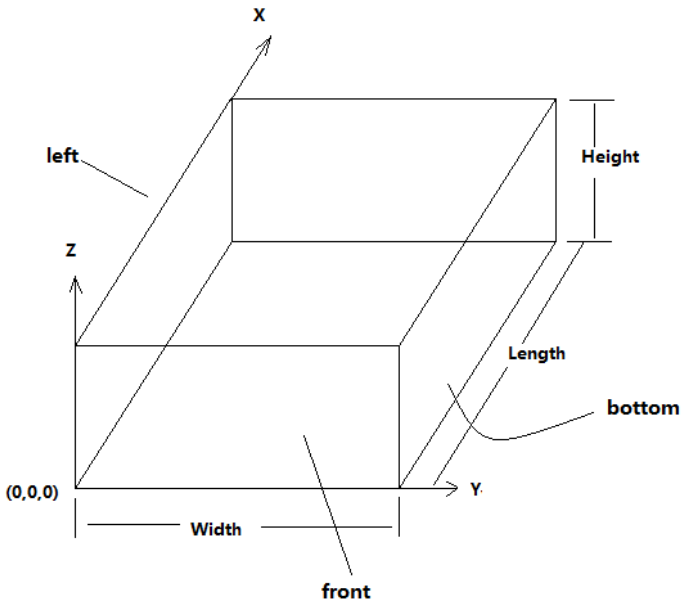
- 通常，所有箱子有相同的长宽高，$L_i = L, W_i = W, H_i = H, \forall\, i = 1,2,\ldots,m$。
- 物品的边缘应该平行于它所在的箱子的边缘。
- 在某些情况下，我们要求物品不可以旋转。
- 显然，如果每一个物品的高度都等于箱子的高度 $H$，那么三维装箱问题就被简化为二维装箱问题。




边缘平行　　不可旋转

# 模型描述

- 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。

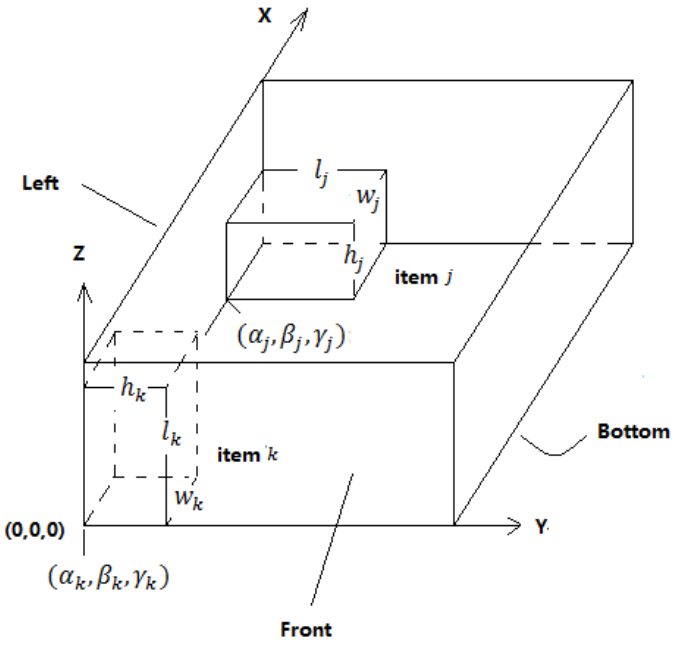


# 模型描述

- 数学模型中使用的变量
  - $y_i$: 0-1变量，$y_i = 1$ 表示箱子 $i$ 被使用，$y_i = 0$ 表示箱子 $i$ 没有被使用
  - $x_{ij}$: 0-1变量，$x_{ij} = 1$ 表示物品 $j$ 被装在箱子 $i$中，$x_{ij} = 0$ 表示物品 $j$ 没有被装在箱子$i$ 中
  - $(\alpha_j, \beta_j, \gamma_j)$: 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
  - $p_{\alpha j}$: 0-1变量，$p_{\alpha j} = 1$ 表示物品$j$的长平行于$X$轴
  - $p_{\beta j}$: 0-1变量，$p_{\beta j}: = 1$ 表示物品$j$的长平行于$Y$轴
  - $p_{\gamma j}$: 0-1变量，$p_{\gamma j} = 1$ 表示物品$j$的长平行于$Z$轴
  - $q_{\alpha j}$: 0-1变量，$q_{\alpha j} = 1$ 表示物品$j$的宽平行于$X$轴
  - $q_{\beta j}$: 0-1变量，$q_{\beta j} = 1$ 表示物品$j$的宽平行于$Y$轴
  - $q_{\gamma j}$: 0-1变量，$q_{\gamma j} = 1$ 表示物品$j$的宽平行于$Z$轴
  - $r_{\alpha j}$: 0-1变量，$r_{\alpha j} = 1$ 表示物品$j$的高平行于$X$轴
  - $r_{\beta j}$: 0-1变量，$r_{\beta j} = 1$ 表示物品$j$的高平行于$Y$轴
  - $r_{\gamma j}$: 0-1变量，$r_{\gamma j} = 1$ 表示物品$j$的高平行于$Z$轴

# 模型描述

- $a_{jk}$: 0-1变量，$a_{jk}=1$ 表示物品$j$在物品$k$的前面，准确来说 $a_{jk}=1$ 表示 $\alpha_j < \alpha_k$
- $b_{jk}$: 0-1变量，$b_{jk}=1$ 表示物品$j$在物品$k$的后面，准确来说 $b_{jk}=1$ 表示 $\alpha_j > \alpha_k$
- $c_{jk}$: 0-1变量，$c_{jk}=1$ 表示物品$j$在物品$k$的左面，准确来说 $c_{jk}=1$ 表示 $\beta_j < \beta_k$
- $d_{jk}$: 0-1变量，$d_{jk}=1$ 表示物品$j$在物品$k$的右面，准确来说 $d_{jk}=1$ 表示 $\beta_j > \beta_k$
- $e_{jk}$: 0-1变量，$e_{jk}=1$ 表示物品$j$在物品$k$的下面，准确来说 $e_{jk}=1$ 表示 $\gamma_j < \gamma_k$
- $f_{jk}$: 0-1变量，$f_{jk}=1$ 表示物品$j$在物品$k$的上面，准确来说 $f_{jk}=1$ 表示 $\gamma_j > \gamma_k$




$$p_{\alpha j}=0, q_{\alpha j}=1, r_{\alpha j}=0$$

$$p_{\beta j}=1, q_{\beta j}=0, r_{\beta j}=0$$

$$p_{\gamma j}=0, q_{\gamma j}=0, r_{\gamma j}=1$$

$$p_{\alpha k}=0, q_{\alpha k}=1, r_{\alpha k}=0$$

$$p_{\beta k}=0, q_{\beta k}=0, r_{\beta k}=1$$

$$p_{\gamma k}=1, q_{\gamma k}=0, r_{\gamma k}=0$$

$$a_{jk}=0, b_{jk}=1$$

$$c_{jk}=0, d_{jk}=0$$

$$e_{jk}=0, f_{jk}=0$$

# 模型描述

1. **min** $f(\boldsymbol{y}) = \sum_{i=1}^{m} y_i$
2. **s.t.** $\sum_{i=1}^{m} x_{ij} = 1, \forall\, j \in N = \{1,2,\dots,n\}$
3. $\sum_{j=1}^{n} x_{ij} \le M y_i, \forall\, i \in I = \{1,2,\dots,m\}$
4. $\alpha_j + l_j p_{\alpha j} + w_j q_{\alpha j} + h_j r_{\alpha j} \le \alpha_k + (1 - a_{jk})M, \forall\, j \in N, i \in I$
5. $\alpha_k + l_k p_{\alpha k} + w_k q_{\alpha k} + h_k r_{\alpha k} \le a_j + (1 - b_{jk})M, \forall\, j \in N, i \in I$
6. $\beta_j + l_j p_{\beta j} + w_j q_{\beta j} + h_j r_{\beta j} \le \beta_k + (1 - c_{jk})M, \forall\, j \in N, i \in I$
7. $\beta_k + l_k p_{\beta k} + w_k q_{\beta k} + h_k r_{\beta k} \le \beta_j + (1 - d_{jk})M, \forall\, j \in N, i \in I$
8. $\gamma_j + l_j p_{\gamma j} + w_j q_{\gamma j} + h_j r_{\gamma j} \le \gamma_k + (1 - e_{jk})M, \forall\, j \in N, i \in I$
9. $\gamma_k + l_k p_{\gamma k} + w_k q_{\gamma k} + h_k r_{\gamma k} \le \gamma_j + (1 - f_{jk})M, \forall\, j \in N, i \in I$
10. $a_{jk} + b_{jk} + c_{jk} + d_{jk} + e_{jk} + f_{jk} \ge x_{ij} + x_{ik} - 1, \forall\, j \in N, i \in I$
11. $\alpha_j + l_j p_{\alpha j} + w_j q_{\alpha j} + h_j r_{\alpha j} \le L_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
12. $\beta_j + l_j p_{\beta j} + w_j q_{\beta j} + h_j r_{\beta j} \le W_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
13. $\gamma_j + l_j p_{\gamma j} + w_j q_{\gamma j} + h_j r_{\gamma j} \le H_i + (1 - x_{ij})M, \forall\, j \in N, i \in I$
14. $p_{\alpha j} + p_{\beta j} + p_{\gamma j} = 1, \forall\, j \in N$
15. $q_{\alpha j} + q_{\beta j} + q_{\gamma j} = 1, \forall\, j \in N$
16. $r_{\alpha j} + r_{\beta j} + r_{\gamma j} = 1, \forall\, j \in N$
17. $p_{\alpha j} + q_{\alpha j} + r_{\alpha j} = 1, \forall\, j \in N$
18. $p_{\beta j} + q_{\beta j} + r_{\beta j} = 1, \forall\, j \in N$
19. $p_{\gamma j} + q_{\gamma j} + r_{\gamma j} = 1, \forall\, j \in N$
20. $\alpha_j \ge 0, \beta_j \ge 0, \gamma_j \ge 0, \forall\, j \in N$
21. $p_{\alpha j}, p_{\beta j}, p_{\gamma j}, q_{\alpha j}, q_{\beta j}, q_{\gamma j}, r_{\alpha j}, r_{\beta j}, r_{\gamma j}, a_{jk}, b_{jk}, c_{jk}, d_{jk}, e_{jk}, f_{jk} \in \{0,1\}, \forall\, j,k \in N, i \in I$

***M is a large constant***

# 集合覆盖模型

- 装箱方案 (a packing pattern)：如果一组物品可以装入一个箱子，那么我们称这组物品为一个装箱方案。
- 这样的packing pattern的数量 $K$ 可能是指数级的。
- 我们用长度为 $n$ 的列向量 $A^k$ 表示一个packing pattern：$A^k$ 的每一个元素 $a_j^k$ 指示物品 $j$ 是否在这个packing pattern中，如果存在则 $a_j^k = 1$，否则 $a_j^k = 0$。
- 由所有packing pattern的列向量组成的矩阵可以表示为 $A$。

$$\min \ f(\boldsymbol{x}) = \sum_{k=1}^{K} x_k$$

$$\text{s.t.} \quad \sum_{k=1}^{K} a_j^k x_k = 1, \forall\, j \in \{1,2,\dots,n\}$$

$$x_k \in \{0,1\}, \ \forall\, k \in \{1,2,\dots,K\}$$

- $x_k = 1$ 表示第 $k$ 个packing pattern出现在解中，否则 $x_k = 0$

# 特点

- NP 难问题
- 不同于一般的整数规划问题
- 仅有一些启发式算法可以在有限的计算空间和时间约束下得到较好的解

# 应用

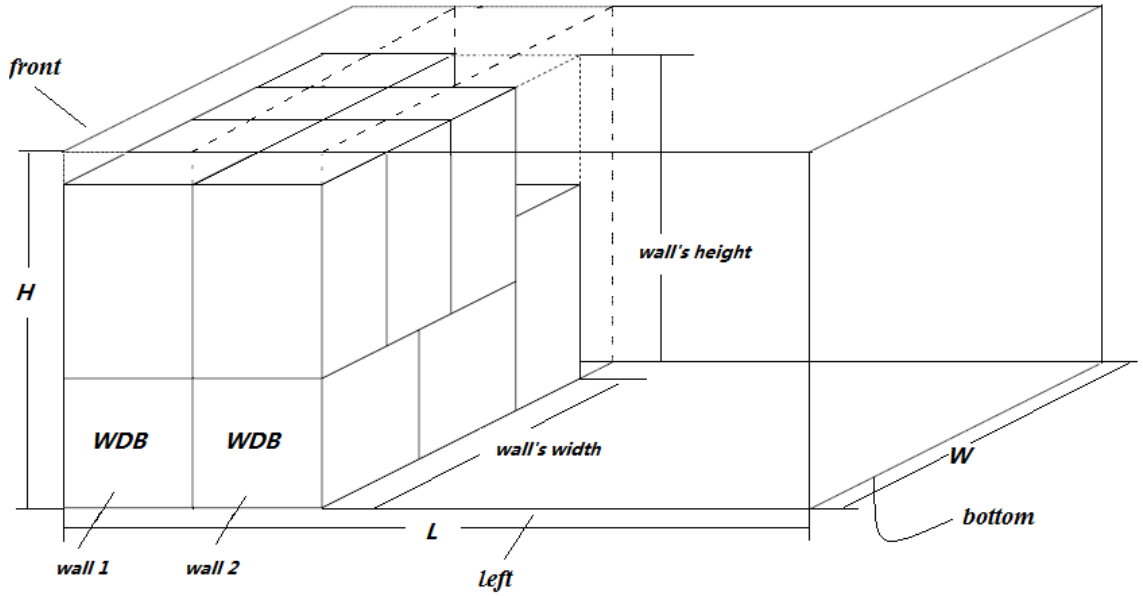
- 具有不同长宽高的石料需要用平板架运送到建筑工地。
- 具有不同长宽高的货品需要被摆放在仓库的货架上。
- 在家具制造行业中，特定形状的3D模板需要从大块塑料泡沫中裁出。
- 在金属制造工业中，特定形状的钢块需要从大块钢材中切出。
- 在木材制造行业中，特定形状的木块需要从原始木材上裁出。
- 在供应链或运输行业中，将不同形状的货物装入集装箱。

# 贪婪启发式算法

- 装箱问题属于NP-完全问题。目前最有效的计算方法是一些启发式算法。
- 在三维装箱问题中常用的启发式算法包括：
  - ***Wall building*** 算法
  - ***Tower building*** 算法
  - ***Corner position packing*** （***CPP***）算法

# *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-下位置上的物品中决定。我们把这个物品称为建造这面墙的关键物品 (*Wall Determining Box，WDB*)。

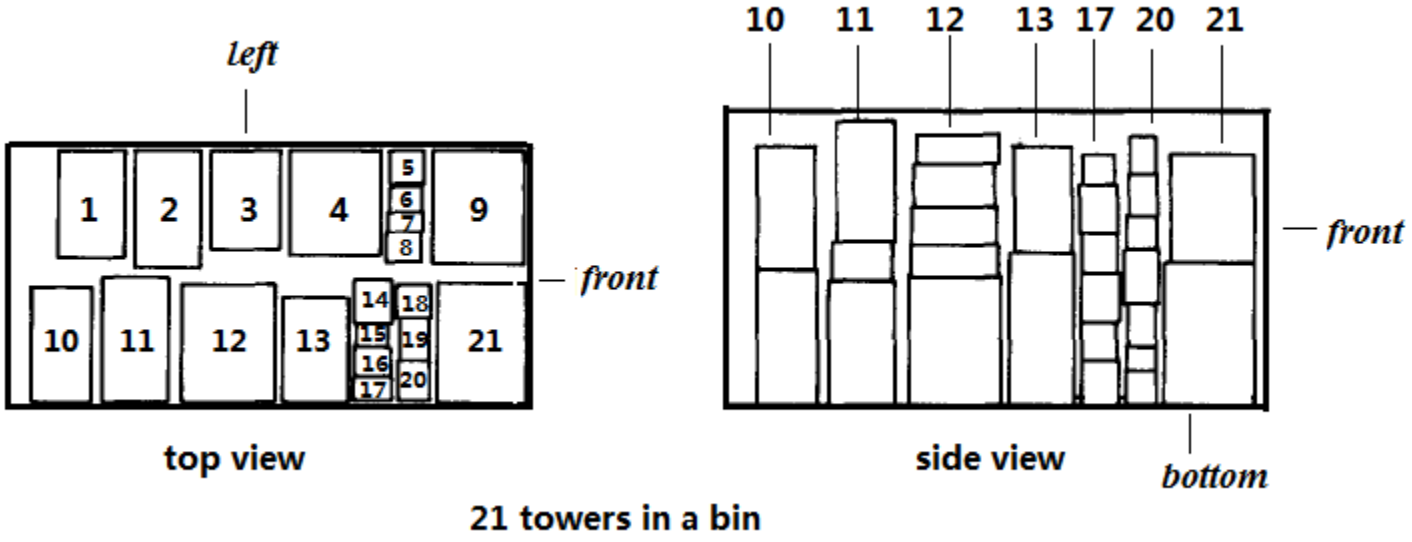


# *Wall building*算法

- 两段式 ***Wall building*** 算法
  - 步骤一：将所有物品堆砌成宽度小于$W$、高度小于$H$的墙
    - 将所有物品按照体积递减的顺序排列，并依次将物品按照优先配合的方法堆放成一面一面物品墙。
    - 虽然物品可以旋转，但是关键物品必须以长度平行于X轴的方式放入墙中。所以每一面墙的长度等于其关键物品的长度。
    - 优先配合方法：
      - 如果没有任何一个已经建立起来的物品墙可以容纳当前待装物品（当前体积最大的物品），那么将此物品作为关键物品并开启一面新墙。
      - 否则，将当前物品放入第一面可以容纳它的墙中，并且使得这个物品在墙中的位置尽可能靠近前面、左面和下面。
  - 步骤二：将所有物品墙装入箱子中
    - 解一维装箱问题

# *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*）。

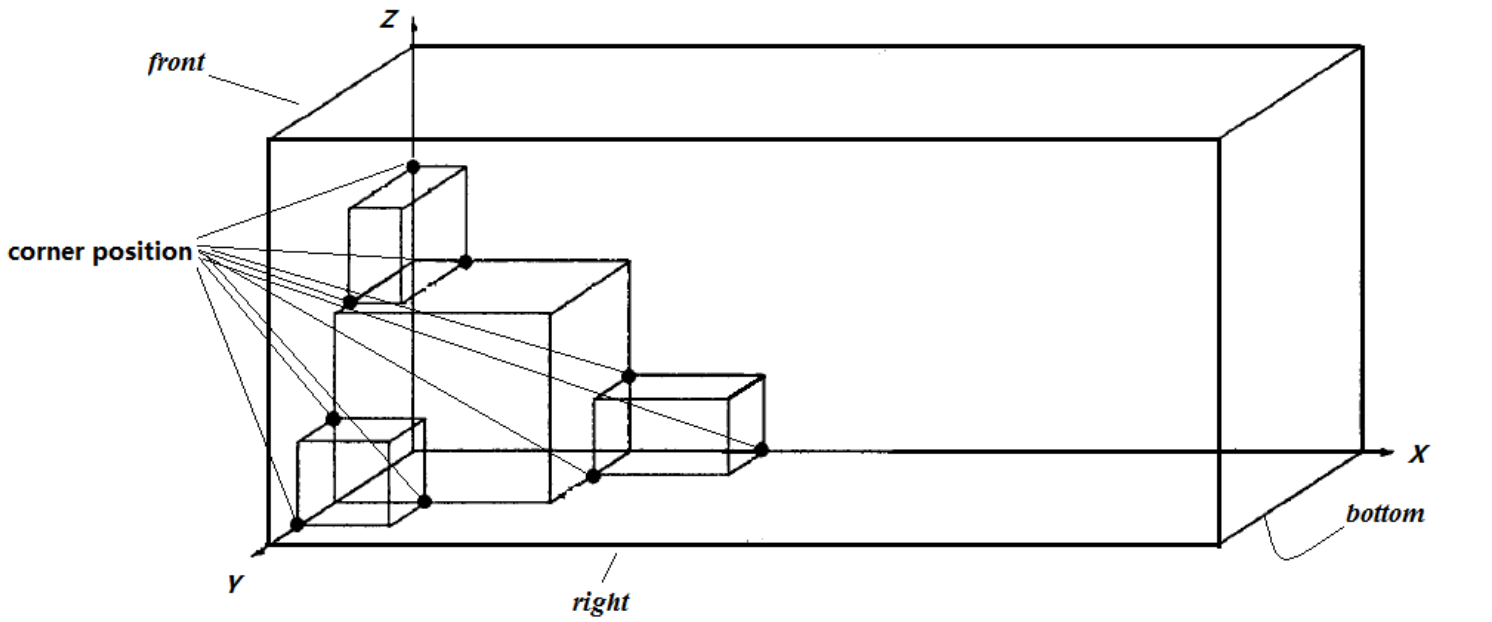


# *Tower building* 算法

- 两段式 *Tower building* 算法
  - 步骤一：将所有物品堆成高度小于$H$ 的塔
    - 将所有物品按照体积递减的顺序排列，并依次将物品按照优先配合的方法堆放成一座一座的物品塔。
    - 虽然物品可以旋转，但是关键物品必须以面积最大的平面朝下的方式摆放。所以每一座塔的底面应该是其关键物品六个面中最大的面。
    - 优先配合方法：
      - 如果没有任何一个已经建立起来的物品塔可以容纳当前待装物品（当前体积最大的物品），那么将此物品作为关键物品并开启一座新塔。
      - 否则，将当前物品放入第一座可以容纳它的塔中，并且使得这个物品在塔中的位置尽可能靠近前面、左面和下面。
  - 步骤二：将所有物品塔装入箱子中
    - 解二维装箱问题

# *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。**



# *Corner position packing* 算法

- 将所有物品依次装入箱子中：
  - 如果当前物品无法被装入任何一个已使用的箱子，则初始化一个新箱子，并将该物品放在新箱子的前-左-下角（新箱子中唯一的一个corner position）。
  - 否则，使用以下打分机制为每一个箱子中的corner position打分，并选择分数最高的一个corner position来放置此物品。
    - 打分机制：当一个物品被放置在一个corner position时，物品的前壁、左壁和下壁与箱子或者其他物品的边界相接触，其中接触的最多的（或者接触率最高的）corner position分数最高。
      - 如果分数相同，那么选择剩余体积最小的箱子。

# 三维装箱问题的变种问题

- **箱子尺寸不同的三维装箱问题(3D Variable-Sized Bin Packing Problem/ Multiple Container Loading Problem):**
  - 寻找最优的装箱方案，使得装入同一个箱子的所有物品不会相互重叠，而使用的箱子的总体积最少。
- **箱子尺寸不同费用也不同的三维装箱问题(3D Bin Packing Problem with Variable Sizes and Costs/ Multiple Container Loading Cost Minimization Problem):**
  - 寻找最优的装箱方案，使得装入同一个箱子的所有物品不会相互重叠，而使用的箱子的总费用最小。
- **三维条装箱问题(3D Strip Packing Problem):**
  - 将所有物品装入一个长度和宽度固定但是高度无限制的长条中，使得所有物品不会重叠，而总长度最小。

# 三维装箱问题的变种问题

- 三维装箱问题在货物运输行业非常常见，但是一般的三维装箱算法无法用于实际生产生活，因为货物装箱问题中有很多特殊情况和限制条件，比如：
  - 并不是所有的物品都是长方体
  - 重量限制(weight limitation)：将货物装入卡车时除了要考虑空间几何限制外，还要考虑卡车的限重
  - 重量分布(weight distribution)：为了防止翻车，装货物时要考虑水平轴和垂直轴上的重量分布
  - 货物自身的承重能力(loading bearing strength)：重物不可以压在易碎物品上
  - 货物稳定性(stability constraint)：在运输易碎物品时由其重要，如果物品之间的缝隙太大，卡车运行过程中的颠簸会造成货物的移动甚至跌落