
队伍编号	21200060022
题号	D

钢材制造业中的钢材切割下料问题

摘 要

二维板材切割问题在实际的工程中有很多的应用，其数学本质是对几何图形进行排样，属于典型的组合优化问题。最优的排样方案可以最大限度地节约材料、提高材料利用率，在经济上制造可观的效益。

本文研究了在满足工艺要求的情形下如何对原材料矩形进行分割（或者对订单矩形排列组合）出最优的方案使得所使用的原料数最少，同时尽量提高原料矩形的利用率等问题，建立了全局最优和局部最优两种整数规划模型，运用了遗传基因算法模型设计和蚁群算法模型设计来求解每张原料的最优排刀方案和使用张数最小的原料数量，所用软件主要为 MATLAB、LINGO 和仁霸板材优化软件。

针对问题一：订单种类只有前五种卷料，首先运用适当穷举法或利用遗传和蚁群算法列出每张原料的所有或最优的“一刀切”（横切）和排刀（纵切）切割出不同数量订单方案，注意到原料 2 和原料 4 的长度显然不满足能切割出卷料订单的要求故可先进行排除，切割原则为尽可能使得每张原料的成材率高（满足两种余料要求的剩余部分原料计入），并用规模为订单数量（5 行） \times 方案数量的矩阵存储，故应有 8 个确定行数的矩阵。接下来进行整数规划，对剩余 8 张原料的不同方案“赋予”不同的未知张数，目标函数为最少原料张数，基本约束条件应包括 8 个矩阵中所有可以切出订单 j ($j=1,2,\dots,5$) 的数量之和等于该订单的需求量，以及累计订单长宽和原料长宽之间的比较关系等，详情请见模型建立与求解部分。

针对问题二：问题二实则是问题一的更多维情况，增加了对板料订单的需求，其规划模型的基本思路是先满足对卷料的需求量，再用遗传模型搜索可在卷料间切割出板料的部分，同时使得最终的废料部分少一些，或满足余料要求的部分多一些。10 个确定行数为 15 的矩阵包含了订单、原料以及他们数量的所有信息，所以问题的复杂性还是在

于每个原料的规划而并不在于维数的增加。为此我们设计的融合了遗传和蚁群智能搜索算法长处的求解模型对寻找局部最优解发挥了重要的作用。

针对问题三：问题三又是问题二的“细节版”，其难度在于不但要使每种原料的成材率高而且要尽量减少换刀次数和机器切割次数，这使得对智能搜索算法施行了限制条件。减少换刀次数意味着要在一张原料里每一块横切的部分实现局部最优排样，这样就能为减少换刀次数提供前提条件；而减少机器切割次数意味着在一张原料里每一块横切的部分能切出更多相同的订单或相同长度的订单。在此基础上再去使用问题二的模型去求解。

针对问题四：问题四开放性较强，在考虑订单长宽浮动比例时更可能会提供比问题二更优化的切割方案（更高的成材率），因为可以尽量使由于订单间长宽不同导致原料剩余部分不好规划而成为废料的可能性降低。

最后给出了模型的结果分析、评价和补充，本文结论正确合理，具有一定的应用价值和普适性。

关键词：最优排样 组合优化 全局最优 局部最优 整数规划 智能搜索算法 遗传基因算法 蚁群算法

目录

一、问题重述	1
二、模型假设	3
三、符号说明	3
四、问题分析	4
五、模型的建立与求解	5
1. 模型一	5
1.1 模型建立	5
1.2 求解方法	7
2. 模型二	9
2.1 寻求原料 i 的最优（或局部最优）切割方案建模	9
2.2 每张原料为最优方案时所需最少原材料张数建模	13
3. 求解结果	14
六、模型的结果分析	24
七、模型的评价与补充	25
参考文献	27
附录一	28

一、问题重述

钢材大量应用于建筑、制造业。在制作时，需要对原料利用率和人力消耗做出最适宜的预算。

已知钢材原材料和订单均为矩形。再进行切割时，由于机器设备问题，只能通过切头剪和小机器进行横向切割，圆盘剪纵向切割，且圆盘剪的刀数不超过 5 把，切割过程中不能进行转弯或停顿。切割次序和排刀方法不同、各种订单规格的搭配不同（即下料策略）不同，材料的损耗及人工的劳力不同。工厂未来一批订单需求如表 1，订单需求数量必须恰好满足（不允许超额供应），工厂现有原料规格及贮存如表 2。根据工厂现有原料切割订单，每种原料的使用数目不能超过其库存。现需解决的实际问题如下：

- （1） 针对给出的所有原料，请使用最少张数的原材料，满足对 5 种卷料的要求（不考虑浮动比例），同时尽量提高总的成材率，给出切割方案。
- （2） 针对给出的所有原料，请使用最少张数的原材料，满足对所有订单的要求（不考虑浮动比例），同时尽量提高总的成材率，给出切割方案。
- （3） 圆盘剪每次排刀需要人工更换刀在排刀架上的位置，同时若有材料需要被移到小机器上再次切割也需要人为操作。为减少人力成本，希望尽量减少换刀数和在小机器上切割数。
- （4） 若订单额外指定了浮动比例，则交付的订单长度可在需求长度的基础上上下浮动。（例：规定浮动比例为 5%，则切割出的长度在原有长度的 95%到 105%之间均满足要求）请重新按照第（2）问的要求，给出切割方案。

表 1 订单需求表

订单编号	长度	宽度	需求量（张）	浮动比例	种类
1	44351.13	422.8 8	36	5%	卷料
2	39229.01	282.8 8	29		卷料
3	54787.74	268.3 6	42		卷料

4	45284.39	277.7	32	1%	卷料
5	53479.79	332.2 9	18	10%	卷料
6	897.32	603.0 6	38		板料
7	896.09	714.7 2	23	10%	板料
8	1096.33	435.8 4	31	10%	板料
9	890.53	343.0 8	40		板料
10	752.61	641.4 5	42		板料
11	970.16	667.2 1	34		板料
12	998.29	472.3	25	5%	板料
13	1024.87	340.5 1	24		板料
14	621.91	476.6	22		板料
15	1243.03	471.2 5	28	10%	板料

表 2 工厂现有原料表

原料编号	长度	宽度	库存（张）
1	148623.9	1519.91	5
2	32960.49	999.35	10
3	75508.72	1232.32	8
4	14091.52	920.62	2
5	75040.31	1573.71	3
6	138570.4	844.99	10
7	98641.28	1184.54	12
8	114074.3	879.02	9
9	104637.7	969.02	3
10	58023.82	1785.45	10

二、模型假设

- （1） 不考虑钢材在切割过程中的消耗。
- （2） 不考虑刀的厚度影响。
- （3） 不考虑 10 中原材料的优先级及成本，只考虑原料的成材率，人工劳力。
- （4） 不考虑切割工艺的不同。
- （5） 不考虑认为损耗。

三、符号说明

L_i	第 i 种原料的长度 ($i=1,2,\dots,V$)
W_i	第 i 种原料的宽度
I_i	第 i 种原料的库存量

l_j	第 j 种订单的长度 ($j=1,2,\dots,J$)
w_j	第 j 种订单的宽度
V	原料的种类
J	订单的种类
d_j	第 j 种订单的需求量
n_i	第 i 种原料切割方案的数量
a_{jk}	用第 i 种原料时第 k 种方案能制成第 j 种订单的个数 ($j=1,2,\dots,J;k=1,2,\dots,n_i$)

四、问题分析

这是一个典型的多目标决策优化问题。

每一问都须解决的基本问题：在固定长宽的矩形原料里，切割出大小不一样但固定的订单件，如何使得原料所用的件数最少及产生的废料面积最少（成材率最高）。

不妨将所要切割的订单件编成一个序列 $T(j)$ ，其中 $j = 1, 2, \dots$ ，则算法的基本思想是按从

下到上、从左到右的顺序规则切割（排列），也就是说将从一块原料的左下角开始切割

T_1 订单，记录下当前位置，接着再从左下角切割 T_2 订单，看能否切割，不能的话向 T 上移动，以此类推，当切割到原料的顶端时，继续从余料件左下角切割，当切割到一定数量的订单后，判断是否还能再进行切割，如能，则继续下去直至此矩形原料所余部分量无法按所需尺寸成订单为止。将剩下的订单继续按照此方法在新原料上进行切割，继续这样的步骤直至切出所要的全部订单为止。这样一个完整的切割方案其实就是一个排序，

以最后余料的大小作为每个排序的评价标准，于是该问题就转化为最优排序问题。

五、模型的建立与求解

1. 模型一

1.1 模型建立

设一张原料 i ($i=1,2,\dots,10$) 理论上有 n_i 种切割方案制成不同数量的订单 j ($j=1,2,\dots,15$)。原料 i 在第一种方案下制成订单 1 的张数为 a_{i1} 张,, 在第一种方案下制成订单 J 的张数为 a_{iJ} 张,, 在第 n_i 种方案下制成订单 1 的张数为 a_{in_i} 张,, 在第 n_i 种方案下制成订单 J 的张数为 a_{Jn_i} 张。

则由以上模型假设可得到如下对应关系矩阵:

$$A_i = \begin{bmatrix} a_{i1} & \cdots & a_{in_i} \\ \vdots & \ddots & \vdots \\ a_{J1} & \cdots & a_{Jn_i} \end{bmatrix}$$

则使用 n_i 种不同切割方案所使用原料 i 的张数为

$$X_i = (x_{i1}, x_{i2}, \dots, x_{in_i})^T, \quad i=1,2,\dots,V$$

而原料 i 的每种切割方案所产生的废料或余料面积为

$$S_{ik} = L_i W_i - (a_{1k} l_1 w_1 + a_{2k} l_2 w_2 + \cdots + a_{Jk} l_J w_J), \\ k=1,2,\dots,n_i$$

对于所有搜索到的余料最省的切割方式，我们由以下模型用来筛选：

主目标函数： \min

$$\sum_{i=1}^V \sum_{k=1}^{n_i} x_{ik}$$

次目标函数： \min

$$\sum_{i=1}^V \sum_{k=1}^{n_i} S_{ik} x_{ik}$$

约束条件：

(1)

$$\sum_{i=1}^V \sum_{k=1}^{n_i} a_{jk} x_{ik} = d_j$$

$$j = 1, 2, \dots, J$$

(2)

$$0 \leq \sum_{k=1}^{n_i} x_{jk} \leq I_i$$

且 x_{jk} 为整数

$$i = 1, 2, \dots, V \quad j = 1, 2, \dots, J$$

(3)

$$\sum_{k=1}^{n_1} a_{jk} l_j \leq L_1$$

.....

$$\sum_{k=1}^{n_V} a_{jk} l_j \leq L_V$$

$$j = 1, 2, \dots, J$$

(4)

$$\sum_{k=1}^{n_1} a_{jk} w_j \leq W_1$$

.....

$$\sum_{k=1}^{n_V} a_{jk} w_j \leq W_V$$

$$j = 1, 2, \dots, J$$

1.2 求解方法

方法是将原料切割成订单后所得余料一起作为一个区间段，将要进行切割的订单与这所有可能的区间段进行长度比较，选择可以切割的位置，对选出的位置进行宽度比较，选择最佳切割区，切割完后，再对区间段进行重新计算，调整其参数。

我们这里为了实现这个算法，采用的是遗传基因法，对于染色体的编码采用整数模式，具体操作步骤如下：

(1) 编码方法

采用随机键表达来产生一串(0,1)之间的随机数。比如，一个10个矩形的染色体为

[0.11,0.14,0.43,0.18,0.53,0.28,0.47,0.73,0.64,0.67]。

我们将随机数按照升序排列，得到如下放置顺序：

8—10—9—5—7—3—6—4—2—1

这条染色体表示先后经过编号为8，10，9，5，7，3，6，4，2，1城市，即表示把编号为8，10，9，5，7，3，6，4，2，1的订单件按前述顺序依次在原料件上进行切割。

（2） 杂交

将选出杂交的两个个体进行顺时针或逆时针旋转排列。

比如被选中的2个个体分别为：

P1: 9—1—2—3—4—5—6—7—8

p2: 5—3—4—9—7—2—8—1—6

按照上述方法杂交后得到：

p11: 1—2—3—4—5—6—7—8—9

P22: 3—4—5—9—7—2—8—1—6

这种杂交方法保证了不生成致死染色体，又保证了前一代的性质遗传性。

（3） 变异

我们考虑采用互换变异，也就是在染色体中选择两个基因，将其互换：

P3: 9—1—2—3—4—5—6—7—8

编译后的染色体：

P33: 9—7—2—3—4—5—6—1—8

2. 模型二

在模型一的基础上改进，由于原料和订单种类数量较大并且原料排刀方案的情况众多，导致 np 类完全问题的时间复杂度和方案数量成指数级，所以接下来试图寻找出每种原料 i 的最优（或局部最优）切割方案，并且设要用 x_i 张原料 i 实现最优（或局部最优）切割。

2.1 寻求原料 i 的最优（或局部最优）切割方案建模

研究发现，蚁群算法和遗传算法随着时间的推移，他们求最优解的效率的总体态势如图一所示。遗传算法在算法的前阶段(t_0-t_a 段)具有较高的向最优解收敛的速度，但算法后阶段（即 t_a 之后）求解效率明显下降；而蚁群算法在算法前阶段（ t_0-t_a 段）由于缺乏初始信息素，求解速度较慢。但由于其正反馈机制，信息素不断积累，在 t_a 之后算法向最优解收敛的速度明显提高。因此，两种算法动态融合的机理就是：在 a 点之前采用遗传算法，充分利用遗传算法的快速性、随机性、全局收敛性，其结果是产生有关问题的初始信息素分布；在 a 点之后采用蚁群算法，在有一定初始信息素分布的情况下，充分利用蚁群算法的并行性、正反馈性、求解效率高等特点（ a 点即为最佳点）。具体实现方法如下：

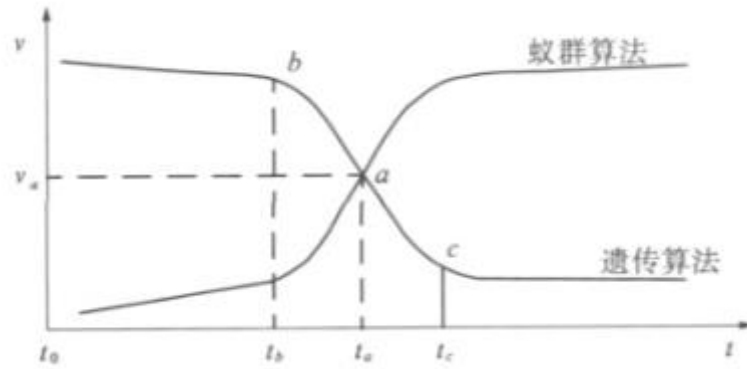


图 1 蚁群算法与遗传算法的速度-时间曲线

- 1) 设置最小遗传迭代次数 $Genemin$ (如 t_b 时刻) 和最大遗传迭代次数 $Genemax$ (如 t_c 时刻);
- 2) 遗传算法迭代过程中统计子代群体的进化率, 并以此设置子代群体最小进化率 $Gnenemin - impro - ratio$;
- 3) 在设定的迭代次数范围内, 如果连续 $Genedie$ 代, 子代群体的进化率都小于 $Genemin - improratio$, 说明这时遗传算法优化速度较低, 因此可终止遗传算法过程, 进入蚁群算法。

2.1.1 订单件排样的遗传算法模型设计如下:

- 1) 编码: 采用十进制整数编码方式。
- 2) 适应度函数: 将适应度函数定义为 $f(R) = Area/Area1$, 其中, $Area$ 是待切割订单件的面积总和, $Area1$ 是所得排样图高度轮廓线以下的原料面积。
- 3) 初始种群: 使用随机函数来生成一定数量的十进制整数序列组成父辈群体, 设定种群规模 $M=50$ 。
- 4) 交叉算子: 采用单点交叉和双点交叉。设置交叉概率 $pc=1$, 单点交叉与双点交叉

各占一半。

5) 变异算子：采用位置变异和旋转变异。设置位置变异概率 $pm1=0.1$ 和旋转变异概率 $pm2=0.1$ 。

6) 选择算子：根据适应度函数值从大到小排列执行完交叉、变异操作的个体，选择前M个个体组成下一代父辈群体。

7) 结束条件：也就是两种算法的最佳动态融合时机。设置

$Genemin=15$ ， $Genedie=3$ ， $Genemax=50$ ， $Genemin-impro-ratio=0.03$ 。

2.1.2 订单件排样的蚁群算法模型设计如下：

状态转移准则 设置订单件的数量为 n ，蚂蚁的数量为 m ，蚂蚁 k ($1 \leq m$)

每一步选择下一订单件时，使用如下规则

$$J = \begin{cases} \arg \max_{i \notin m_k} \{ [\tau(i)]^\alpha \cdot [\eta(i)]^\beta \}, & q \leq q_0, \\ AA, & q > q_0, \end{cases} \quad (1)$$

(1) 式中： $\tau(i)$ 是订单件 i ($1 \leq i \leq n$) 上的信息素浓度； $\eta(i)$ 为订单件 i 的启发信息，取 $\eta(i) = \text{areai} + (L/W)i$ (L, W 分别表示订单件 i 的长与宽， areai 表示订单件 i 的面积)； α 和 β 表示信息素浓度和启发信息的相对重要性；

m_k 表示第 k 只蚂蚁已经选择过的订单件的集合，此集合随着搜索过程作动态调整；

q 是 $[0,1]$ 上的随机数， q_0 是一个适当设定的参数 ($0 \leq q_0 \leq 1$)。若 $q \leq q_0$ ，按式

(1) 中的第 1 种情况选择下一订单件；若 $q > q_0$ ，则按第 2 种情况选择， AA 是根据式 (2) 确定的概率分布。

$$p_k(i) = \begin{cases} \frac{[\tau(i)]^\alpha \cdot [\eta(i)]^\beta}{\sum_{j \notin m_k} [\tau(j)]^\alpha \cdot [\eta(j)]^\beta}, & i \notin m_k, \\ 0, & i \in m_k, \end{cases} \quad (2)$$

其中， $p_k(i)$ 表示第 k 只蚂蚁选择订单件 i 的概率， i, j 意义相同 ($1 \leq j \leq n$)。

信息素初值设置 采用最大—最小蚁群系统的 MMAS 算法。信息素初值设定为

$$\tau_A = \tau_C + \tau_G, \quad (3)$$

式中： τ_C 相当于 MMAS 算法中的 τ_{\min} ，是一个信息素参数； τ_G 是由遗传算法求得的解的转换成的信息素值。

信息素更新模型

对留在订单件上的信息素进行全局更新和局部更新。全局更新是只对全局最优路径上的信息素进行更新，更新规则如下：

$$\tau(i) = (1 - \rho) \cdot \tau(i) + \rho \cdot \Delta\tau(i), \quad (4)$$

$$\Delta\tau(i) = \frac{Q}{Area/Area1}, \quad (5)$$

式中： $\tau(i)$ 是留在订单件 i 上的信息素浓度； ρ 是信息素挥发系数 ($0 \leq \rho \leq 1$)； Q 是一个信息素强度常数； $Area$ 是待排入订单件的面积总和； $Area1$ 表示全局最优排样序列所对应的排样图的高度轮廓线下的原料面积。

局部更新可以动态地改变留在每个订单件上的信息素浓度，更好地为其他蚂蚁提供搜索信息，更新规则如下：

$$\tau(i)=(1-\psi) \cdot \tau(i)+\psi \cdot \tau_A, \quad (6)$$

其中， ψ 是一个初始参数（ $0 \leq \psi \leq 1$ ）。控制参数设置为： $m=10$ ， $\alpha=1.0$ ， $\beta=2.0$ ， $\rho=\psi=0.1$ ， $q_0=0.9$ ， $Q=1000$ 。

2.2 每张原料为最优方案时所需最少原材料张数建模

由模型假设有

$$A_i = \begin{pmatrix} a_{1i} \\ a_{2i} \\ a_{3i} \\ \vdots \\ a_{ji} \end{pmatrix}, \quad i=1,2,\dots,V$$

目标函数： \min

$$\sum_{i=1}^V x_i$$

约束条件：

(1)

$$\sum_{i=1}^V a_{ji} x_i = d_j$$

(2)

$$0 \leq x_i \leq I_i$$

且 x_i 为整数

$$i = 1, 2, \dots, V$$

(3)

$$\sum_{j=1}^J a_{ji} w_j l_j \leq L_i W_i$$

$$i = 1, 2, \dots, V \quad j = 1, 2, \dots, J$$

3. 求解结果

(1)、有多种原材料的前五种订单最优下料策略

在有十种原材料的情况下，单单考虑上述订单中的卷料，进行几种成品料配套优选方案同样的数学模型，只是在订单的切割方案中，增加九种新的原料，用“仁霸板材优化软件”，排出最优的切割方案，根据需要最少的原料张数，建立线性规划方程组，求出所需要原材料的最少张数，根据最佳的方案可以求出原料的利用率。进行卷料的裁剪，通过“仁霸板材优化软件”排出一种最佳方案，从中我们可以看出

表 3 问题一最佳方案

材 料 编号	张数	订 单 1	订 单 2	订 单 3	订 单 4	订 单 5
1	4	16	8	12	0	16
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	7	0	18	13	30	0

7	3	12	3	3	0	0
8	2	3	0	3	1	2
9	1	3	0	2	0	0
10	2	2	0	9	1	0
总计	19	36	29	42	32	18

通过上述的计算求得成材率为：89.41%

(2)、有多种原材料的前五种订单最优下料策略

将模型进行优化处理，可以计算更加复杂的切料模型，并通过 LINGO 软件进行线性最优解的计算，由于中途过于复杂直接得出结果如下图：

表 4 问题二最佳方案

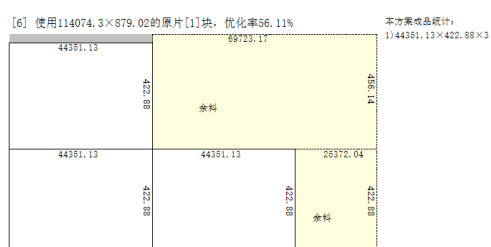
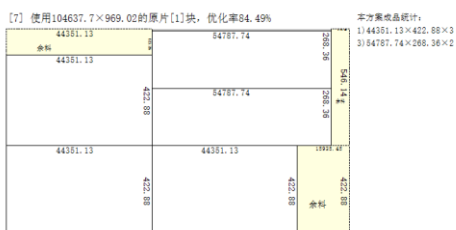
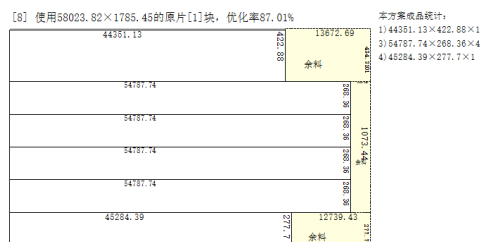
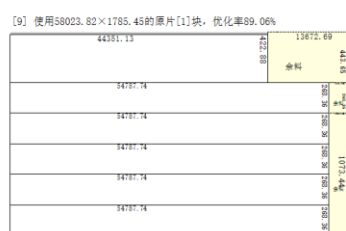
料 编 号	引 数	订 单 1	订 单 2	订 单 3	订 单 4	订 单 5	订 单 6	订 单 7	订 单 8	订 单 9	订 单 10	订 单 11	订 单 12	订 单 13	订 单 14	订 单 15
1	1	0	0	0	0	0	0	0	0	0	0	0	0	4	3	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0	0	0	0	0	3	0	3	4
7	8	0	0	0	0	0	1	0	0	0	0	0	1	0	1	13

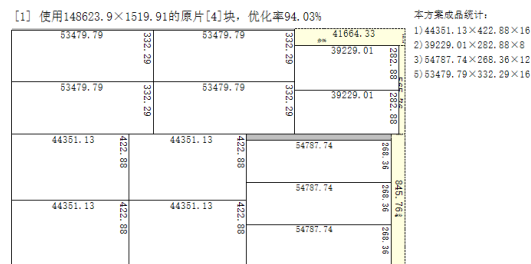
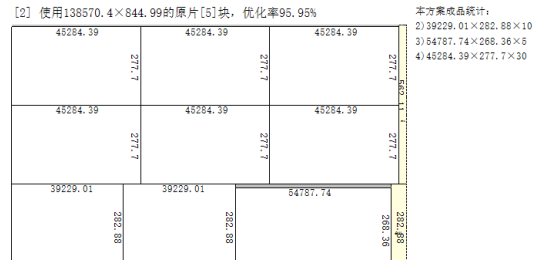
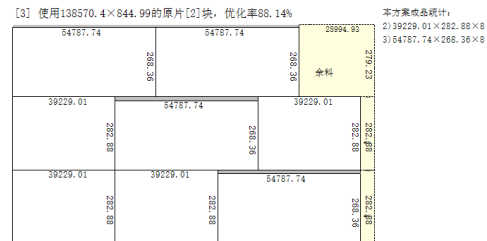
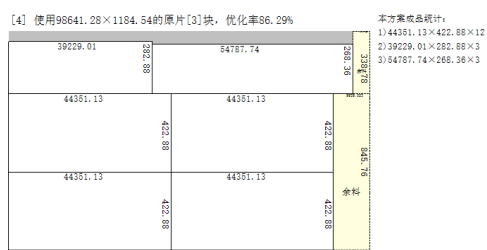
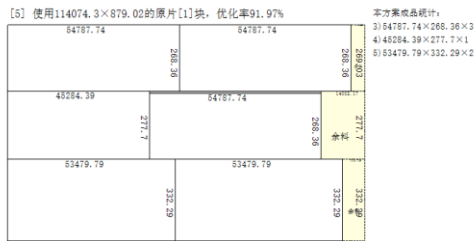
		0	9				4	2			7	4	1		6	
8	9			6		8	0		8	5			1	2	0	11
9	1						2						0	0	0	0
							4									
1																
0	0						0						0	0	0	0
2	3						3						2	2	2	28
计	3	6	9	2	2	8	8	3	1	0	2	4	5	4	2	

得到成材率为 76.56%

问题三、问题四已建立模型，由于本组编写程序能力有限，未能给出具体方案。

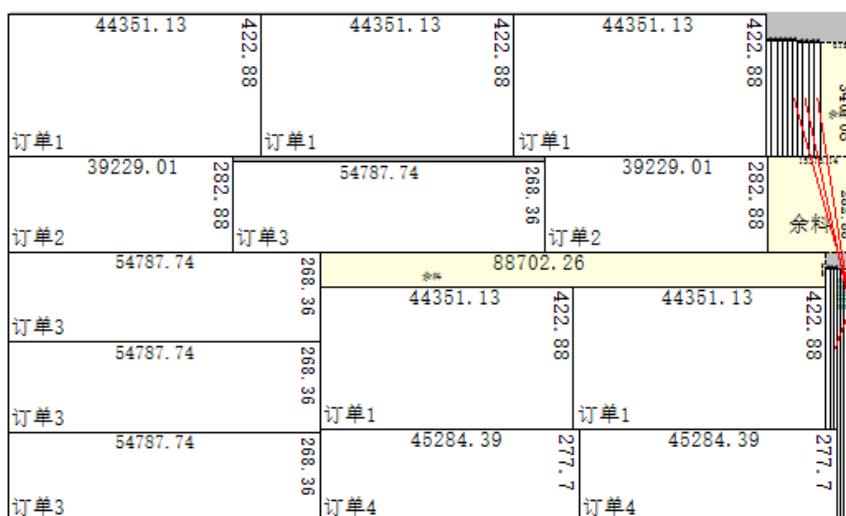
问题一排列图如下:





问题二排列图如下：

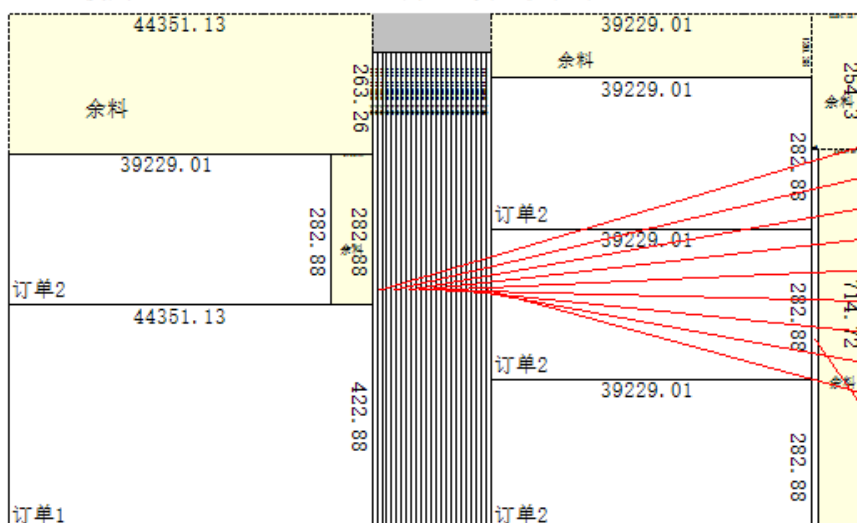
[14] 使用148623.9×1519.91的原片[1]块, 优化率91.40%



本方案成品统计:

- 1) 44351.13×422.88×5
- 2) 39229.01×282.88×2
- 3) 54787.74×268.36×4
- 4) 45284.39×277.7×2
- 9) 890.53×343.08×6
- 10) 752.61×641.45×5
- 13) 1024.87×340.51×4
- 14) 621.91×476.6×3
- 14) 621.91×476.6×3(横)
- 10) 752.61×641.45×4(竖)
- 10) 752.61×641.45×1(竖)
- 9) 890.53×343.08×6(横)
- 13) 1024.87×340.51×2(横)
- 13) 1024.87×340.51×2(横)

[13] 使用104637.7×969.02的原片[1]块, 优化率75.71%



本方案成品统计:

- 1) 44351.13×422.88×1
- 2) 39229.01×282.88×4
- 6) 897.32×603.06×24
- 7) 896.09×714.72×1
- 6) 897.32×603.06×2(竖)
- 6) 897.32×603.06×3(竖)
- 6) 897.32×603.06×3(竖)
- 6) 897.32×603.06×2(竖)
- 6) 897.32×603.06×3(竖)
- 6) 897.32×603.06×2(竖)
- 6) 897.32×603.06×3(竖)
- 6) 897.32×603.06×3(竖)
- 6) 897.32×603.06×3(竖)
- 7) 896.09×714.72×1(横)

[12] 使用114074.3×879.02的原片[5]块，优化率98.42%

54787.74	268.36	54787.74	268.36
订单3		订单3	
53479.79	332.29	53479.79	332.29
订单5		订单5	
54787.74	268.36	54787.74	268.36
订单3		订单3	

本方案成品统计:

- 3) 54787.74×268.36×20
- 5) 53479.79×332.29×10
- 9) 890.53×343.08×25
- 12) 998.26×472.3×10
- 13) 1024.87×340.51×10
- 15) 1243.03×471.25×10
- 13) 1024.87×340.51×1(横)
- 13) 1024.87×340.51×1(横)
- 12) 998.26×472.3×1(横)
- 15) 1243.03×471.25×2(横)
- 12) 998.26×472.3×1(横)
- 9) 890.53×343.08×4(横)
- 9) 890.53×343.08×1(横)

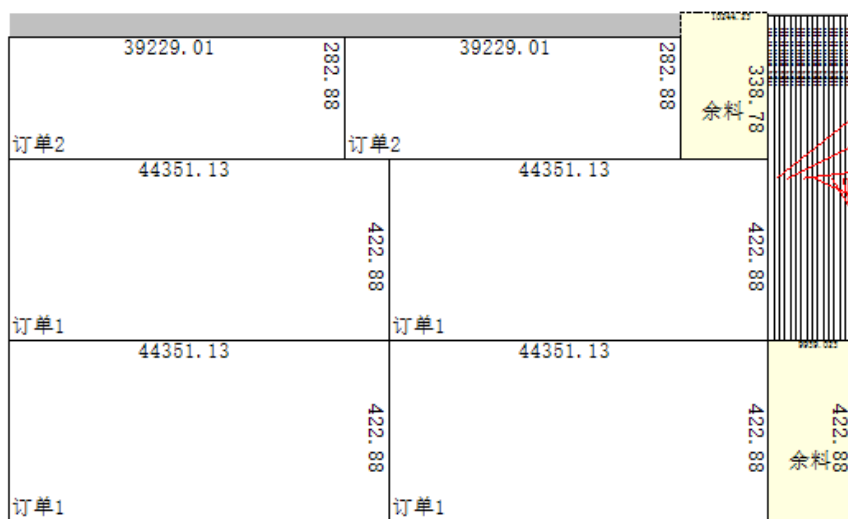
[11] 使用114074.3×879.02的原片[1]块，优化率98.45%

54787.74	268.36	54787.74	268.36
订单3		订单3	
53479.79	332.29	53479.79	332.29
订单5		订单5	
54787.74	268.36	54787.74	268.36
订单3		订单3	

本方案成品统计:

- 3) 54787.74×268.36×4
- 5) 53479.79×332.29×2
- 8) 1096.33×435.84×4
- 12) 998.26×472.3×1
- 13) 1024.87×340.51×4
- 15) 1243.03×471.25×1
- 13) 1024.87×340.51×1(横)
- 13) 1024.87×340.51×1(横)
- 13) 1024.87×340.51×1(横)
- 12) 998.26×472.3×1(横)
- 8) 1096.33×435.84×2(横)
- 8) 1096.33×435.84×2(横)
- 15) 1243.03×471.25×1(横)
- 13) 1024.87×340.51×1(横)

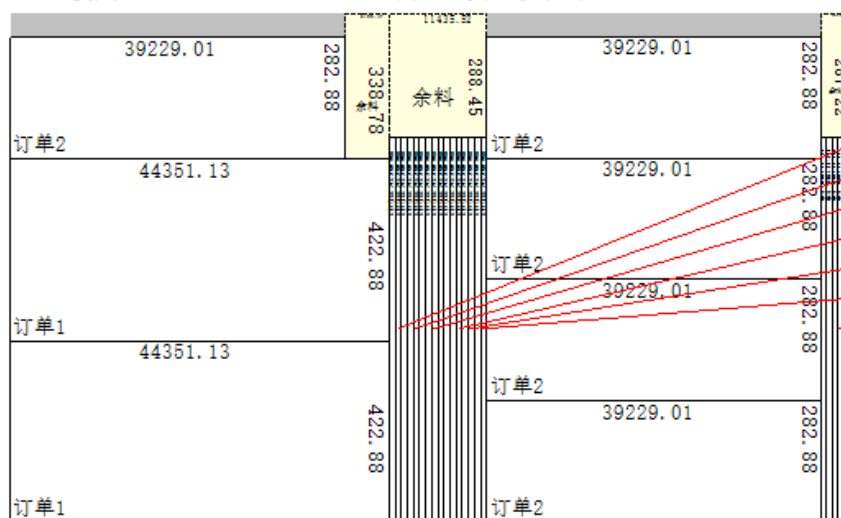
[10] 使用98641.28×1184.54的原片[2]块, 优化率89.40%



本方案成品统计:

- 1) 44351.13×422.88×8
- 2) 39229.01×282.88×4
- 10) 752.61×641.45×30
- 10) 752.61×641.45×2(竖)
- 10) 752.61×641.45×2(竖)
- 10) 752.61×641.45×3(竖)
- 10) 752.61×641.45×2(竖)
- 10) 752.61×641.45×3(竖)
- 10) 752.61×641.45×2(竖)
- 10) 752.61×641.45×1(竖)

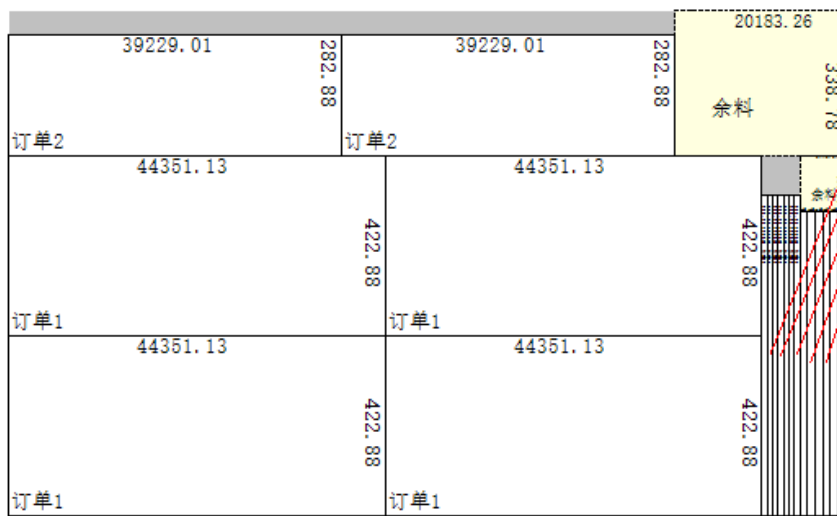
[9] 使用98641.28×1184.54的原片[1]块, 优化率91.14%



本方案成品统计:

- 1) 44351.13×422.88×2
- 2) 39229.01×282.88×5
- 6) 897.32×603.06×6
- 7) 896.09×714.72×16
- 7) 896.09×714.72×2(竖)
- 7) 896.09×714.72×3(竖)
- 7) 896.09×714.72×3(竖)
- 7) 896.09×714.72×4(竖)
- 7) 896.09×714.72×2(竖)
- 7) 896.09×714.72×2(竖)
- 6) 897.32×603.06×4(竖)
- 6) 897.32×603.06×2(竖)

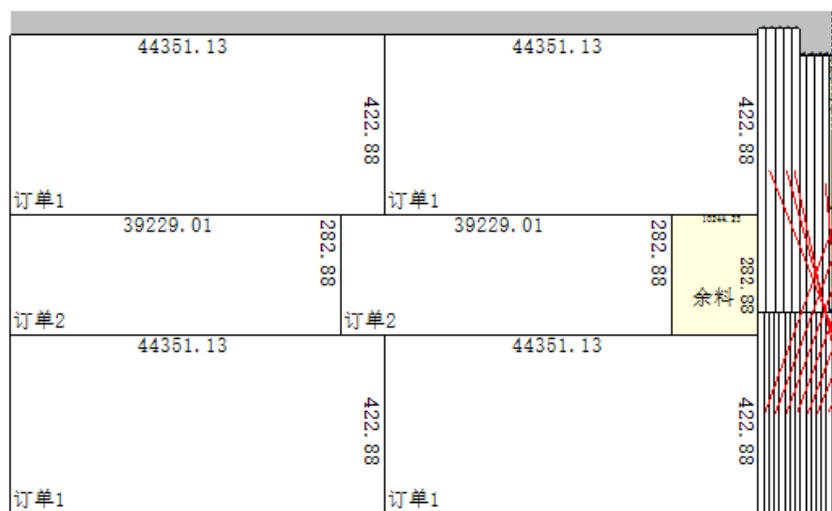
[8] 使用98641.28×1184.54的原片[1]块, 优化率89.38%



本方案成品统计:

- 1) 44351.13×422.88×4
- 2) 39229.01×282.88×2
- 7) 896.09×714.72×6
- 10) 752.61×641.45×7
- 10) 752.61×641.45×2(竖)
- 10) 752.61×641.45×2(竖)
- 10) 752.61×641.45×3(竖)
- 7) 896.09×714.72×2(横)
- 7) 896.09×714.72×2(横)
- 7) 896.09×714.72×2(横)

[7] 使用98641.28×1184.54的原片[1]块, 优化率92.30%



本方案成品统计:

- 1) 44351.13×422.88×4
- 2) 39229.01×282.88×2
- 6) 897.32×603.06×6
- 11) 970.16×667.21×5
- 14) 621.91×476.6×14
- 14) 621.91×476.6×2(横)
- 14) 621.91×476.6×2(横)
- 14) 621.91×476.6×2(横)
- 14) 621.91×476.6×2(横)
- 14) 621.91×476.6×2(横)
- 14) 621.91×476.6×2(横)
- 14) 621.91×476.6×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×1(横)
- 6) 897.32×603.06×4(横)
- 6) 897.32×603.06×1(竖)
- 6) 897.32×603.06×1(竖)

[6] 使用98641.28×1184.54的原片[1]块，优化率92.62%

本方案成品统计：

39229.01		282.88	39229.01	282.88	338.78
订单2	44351.13	422.88	订单2	44351.13	422.88
订单1	44351.13	422.88	订单1	44351.13	422.88
订单1	44351.13	422.88	订单1	44351.13	422.88

- 1) 44351.13×422.88×4
- 2) 39229.01×282.88×2
- 6) 897.32×603.06×2
- 11) 970.16×667.21×9
- 15) 1243.03×471.25×7
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×1(横)
- 15) 1243.03×471.25×4(横)
- 15) 1243.03×471.25×3(横)
- 6) 897.32×603.06×2(竖)

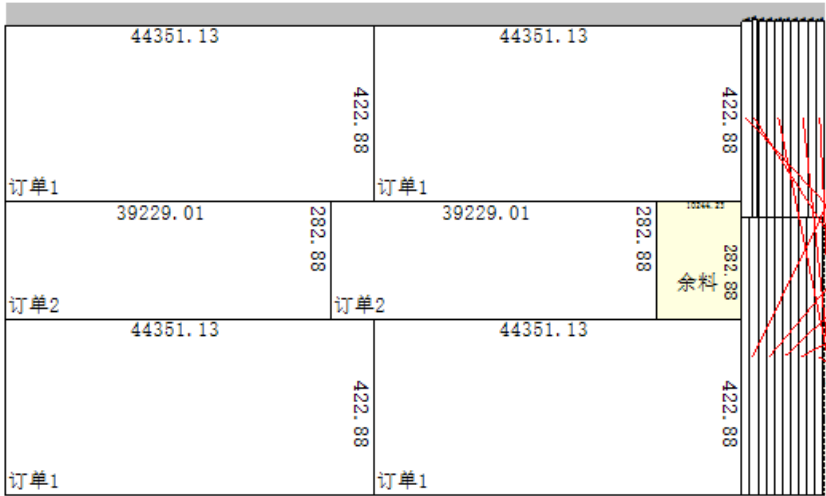
[5] 使用98641.28×1184.54的原片[1]块，优化率92.71%

本方案成品统计：

39229.01		282.88	39229.01	282.88	338.78
订单2	44351.13	422.88	订单2	44351.13	422.88
订单1	44351.13	422.88	订单1	44351.13	422.88
订单1	44351.13	422.88	订单1	44351.13	422.88

- 1) 44351.13×422.88×4
- 2) 39229.01×282.88×2
- 11) 970.16×667.21×10
- 12) 998.26×472.3×3
- 14) 621.91×476.6×1
- 15) 1243.03×471.25×5
- 14) 621.91×476.6×1(横)
- 12) 998.26×472.3×1(横)
- 12) 998.26×472.3×2(横)
- 15) 1243.03×471.25×5(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)

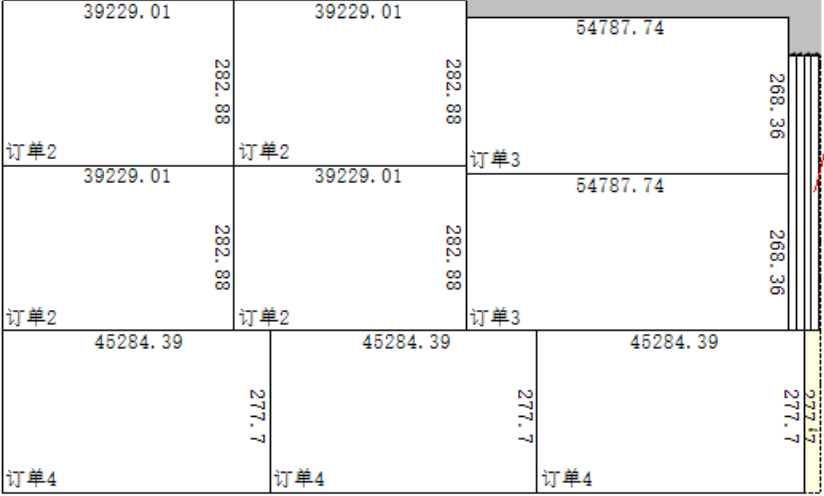
[4] 使用98641.28×1184.54的原片[1]块, 优化率92.72%



本方案成品统计:

- 1) 44351.13×422.88×4
- 2) 39229.01×282.88×2
- 11) 970.16×667.21×10
- 12) 998.26×472.3×8
- 14) 621.91×476.6×1
- 15) 1243.03×471.25×1
- 11) 970.16×667.21×2(横)
- 15) 1243.03×471.25×1(横)
- 14) 621.91×476.6×1(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 11) 970.16×667.21×2(横)
- 12) 998.26×472.3×3(横)
- 12) 998.26×472.3×3(横)
- 12) 998.26×472.3×2(横)

[3] 使用138570.4×844.99的原片[1]块, 优化率97.24%



本方案成品统计:

- 2) 39229.01×282.88×4
- 3) 54787.74×268.36×2
- 4) 45284.39×277.7×3
- 15) 1243.03×471.25×4
- 15) 1243.03×471.25×4(横)

[2] 使用138570.4×844.99的原片[3]块，优化率98.51%

45284.39	45284.39	45284.39	277.7
订单4	订单4	订单4	277.7
45284.39	45284.39	45284.39	277.7
订单4	订单4	订单4	277.7
45284.39	45284.39	45284.39	277.7
订单4	订单4	订单4	277.7

本方案成品统计:

- 4) 45284.39×277.7×27
- 8) 1096.33×435.84×3
- 9) 890.53×343.08×9
- 12) 998.26×472.3×3
- 14) 621.91×476.6×3
- 14) 621.91×476.6×1(横)
- 8) 1096.33×435.84×1(横)
- 12) 998.26×472.3×1(横)
- 9) 890.53×343.08×3(横)

[1] 使用114074.3×879.02的原片[3]块，优化率98.60%

54787.74	54787.74	268.36
订单3	订单3	268.36
53479.79	53479.79	332.29
订单5	订单5	332.29
54787.74	54787.74	268.36
订单3	订单3	268.36

本方案成品统计:

- 3) 54787.74×268.36×12
- 5) 53479.79×332.29×6
- 8) 1096.33×435.84×24
- 13) 1024.87×340.51×6
- 13) 1024.87×340.51×1(横)
- 13) 1024.87×340.51×1(横)
- 8) 1096.33×435.84×6(横)
- 8) 1096.33×435.84×2(横)

六、模型的结果分析

本模型按照逐级优化的方法，通过对下料方案的筛选，当只有一种原料，单一下料的排料最优策略所需要的最少原料也要 157 块，单个原料生产所有订单总体的利用率也

才 86.21%，所以通过每个原料取最优的想法无疑是错误的，在局部最优的情况下，取整体最优的方案可能是无解的，所以我们模型通过对整体的原料进行动态分析，利用遗传算法和蚁群算法可以极大的降低这种片面的最优带给我们在数值上的误差，总体上的成材率达到最大化。

在有十种不同规格的原料的情况下需要原料数如下表：

原料	所需张数
1	4
2	0
3	0
4	0
5	0
6	7
7	3
8	2
9	1
10	2

能达到最大的成材率 89.41%。

不论是只有一种原料、还是有十种原料，所求的成材率基本都是目前最高的，是一种比较好的最优策略。

七、模型的评价与补充

本文主要采用逐级优化的方法，对原料的最优切割策略进行分析计算的方法是基本成功的。事实上，对于一般的二维下料(板材下料)问题，初始切割方式的确定是下料问题中至关重要的内容，如果确定了最优的切割方式，就可以把下料问题化成一个线性规

划问题，也就是要求自动生成线性规划方程的系数矩阵，编程进行计算求解。

蚁群算法是由意大利学者 Dorigo 最早提出的，主要是通过蚂蚁群体之间的信息传递而达到寻优的目的。它具有较强的鲁棒性、优良的分布式计算机制和正反馈机制、易于与其他方法相结合等优点，缺点是求解时间长，初始信息素匮乏，容易出现停滞现象。

遗传算法是一种全局随机搜索算法，它借鉴了生物界的自然选择思想和自然遗传机制，将问题的可行解构成种群，把每一个可能的解看作种群的个体，算法运行时在整个解空间里随机搜索，并按一定的评估策略（或适应度函数）对每一个个体做出评价，然后不断地使用选择、交叉、变异 3 个遗传算子来进化问题的解，直至产生最优解。其优点是：强调概率转移规则，具有快速随机的全局搜索能力，鲁棒性强。缺点是：对于系统中的反馈信息利用不够，当求解到一定范围时往往做大量的冗余迭代，求解效率低。

结合蚁群算法和遗传算法各自的优点，提出了将两种算法相融合来求解矩形件优化排样问题。算法前阶段采用遗传算法，快速获得部分优化排样序列，以此作为蚁群算法的初始信息素分布，然后采用蚁群算法精确求得最优排样序列，得到排样图。实例的计算结果表明，所提出的融合算法使材料的利用率得到提高，优化排样效果更加明显。

参考文献

- [1] 邢文训.谢金星现代优化计算方法 1999
- [2] .尹泽民精通 MATLAB 2002
- [3] 周明.孙树栋遗传算法原理及应用 1999
- [4] 段玉倩遗传算法及其改进 1998(01)
- [5] 张可村工程优化算法与分析 1988
- [6] 姜衍智动态规划原理及应用 1988
- [7] 魏权龄运筹学通论 2001
- [8] 玛晓慧.李菊娥.任春丽装箱问题的一种新算法及其性能比的证明 1998
- [9] 魏凉良.叶家玮一维下料问题的改进自适应遗传算法[期刊论文]-华南理工大学学报(自然科学版) 2003
- [10] 金咏怡产品下料问题的一种求解方法[期刊论文]-湖南商学院学报 2003

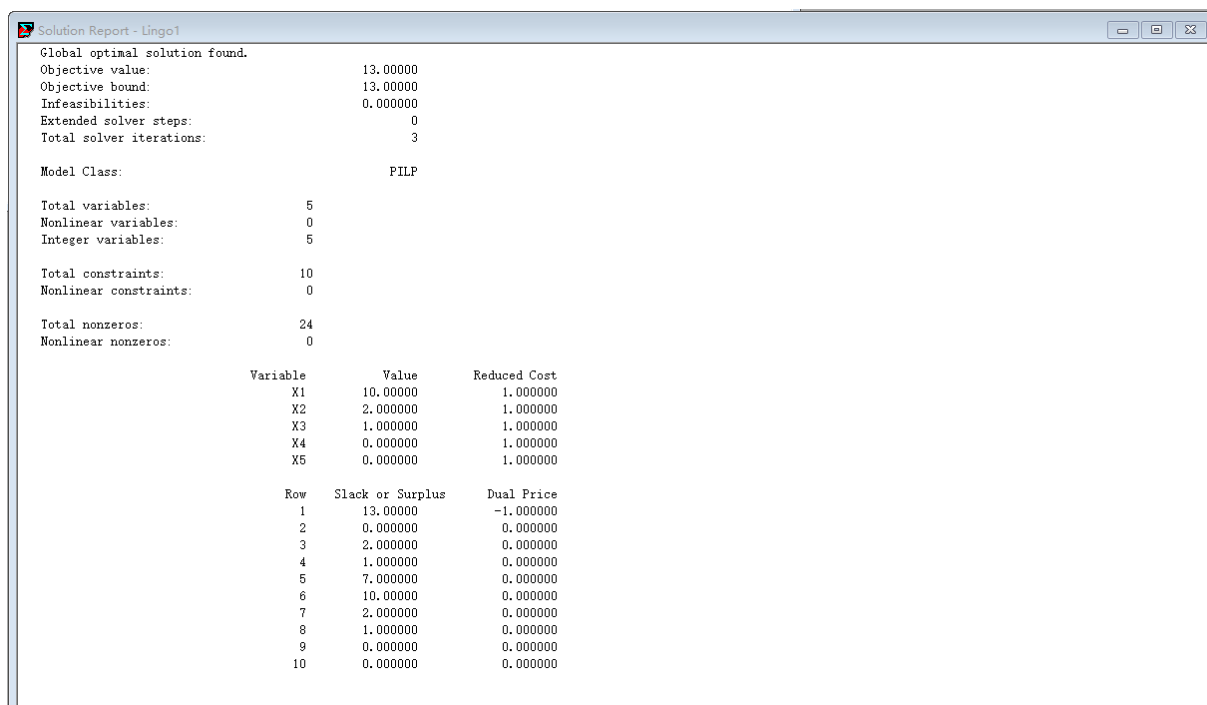
附录一

对于这样一个建立数学模型的问题，采用编写 LINGO 软件程序，利用其中的数学规划功能求解该问题。

对于 LINGO 软件的使用包括以下几个步骤：

1. 编写目标函数，在编程过程中，求解最少原料板使用。
2. 编写约束条件，根据之前已经分析好的约束条件来编写。
3. 点击 LINGO 按钮，得到最终可能性的编辑。

代码 1：LINGO 程序的实现



Solution Report - Lingo1

Global optimal solution found.

Objective value:	13.00000
Objective bound:	13.00000
Infeasibilities:	0.000000
Extended solver steps:	0
Total solver iterations:	3

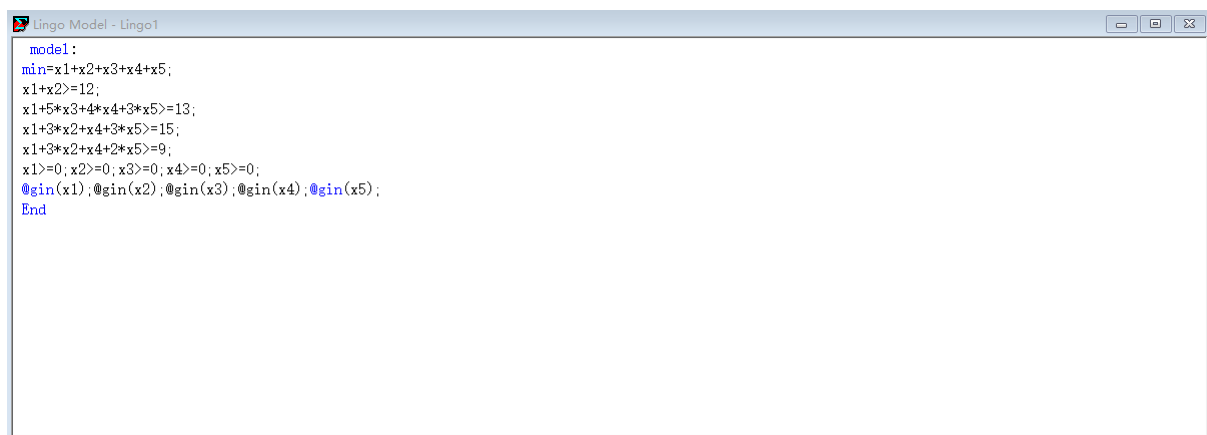
Model Class: PILP

Total variables:	5
Nonlinear variables:	0
Integer variables:	5
Total constraints:	10
Nonlinear constraints:	0
Total nonzeros:	24
Nonlinear nonzeros:	0

Variable	Value	Reduced Cost
X1	10.00000	1.000000
X2	2.000000	1.000000
X3	1.000000	1.000000
X4	0.000000	1.000000
X5	0.000000	1.000000

Row	Slack or Surplus	Dual Price
1	13.00000	-1.000000
2	0.000000	0.000000
3	2.000000	0.000000
4	1.000000	0.000000
5	7.000000	0.000000
6	10.00000	0.000000
7	2.000000	0.000000
8	1.000000	0.000000
9	0.000000	0.000000
10	0.000000	0.000000

代码 2：LINGO 程序的实现



Lingo Model - Lingo1

```
model:
min=x1+x2+x3+x4+x5;
x1+x2>=12;
x1+5*x3+4*x4+3*x5>=13;
x1+3*x2+x4+3*x5>=15;
x1+3*x2+x4+2*x5>=9;
x1>=0; x2>=0; x3>=0; x4>=0; x5>=0;
@gin(x1); @gin(x2); @gin(x3); @gin(x4); @gin(x5);
End
```

代码 3: LINGO 程序的实现

```
model:
min=x1+x2+x3+x4+x5;
x1+x2>=12;
x1+5*x3+4*x4+3*x5>=13;
x1+3*x2+x4+3*x5>=15;
x1+3*x2+x4+2*x5>=9;
x1>=0;x2>=0;x3>=0;x4>=0;x5>=0;
@gin(x1);@gin(x2);@gin(x3);@gin(x4);@gin(x5);
End
```

代码 4: LINGO 程序的实现

```
model:
min=x1+x2+x3+x4+x5;
x1+x2>=12;
x1+5*x3+4*x4+3*x5>=13;
x1+3*x2+x4+3*x5>=15;
x1+3*x2+x4+2*x5>=9;
x1>=0;x2>=0;x3>=0;x4>=0;x5>=0;
@gin(x1);@gin(x2);@gin(x3);@gin(x4);@gin(x5);
End
```

代码 5: 基因遗传算法

```
<!DOCTYPE >
<html>
  <head>
    <meta charset="utf-8"/>
    <title></title>
```



```

</head>
<body>
    <button type="button" id="button">计算结果</button>
    <script type="text/javascript">
        var totalNum = 50, // 种群中染色体的总数
        width=20, // 订单的宽
        length=30, // 订单的长
        N, // 记录需要的订单数量

        bit = 20, // 基因数为 20 位
        total = new Array(); // 种群中的染色体
        bestFitness = 0, // 最佳适应值
        generation = 0, // 染色体代号
        bestGeneration = 0, // 最好的一旦染色体代号
        bestStr = "", // 最好的染色体基因序列
        var size=[], // 记录小矩形块的长宽
        var num=[4,3,5,3,5]; // 记录小矩形块的需求数量
        size[0]=[7,5];
        size[1]=[10,5];
        size[2]=[12,7];
        size[3]=[10,8];
        size[4]=[12,10];
        var sizeTotal=[];
        /*将基因组对应的小矩形块的长宽记录下来*/
        for(var i=0;i<4;i++){
            sizeTotal[i]=size[0];
        }
        for(var i=4;i<7;i++){
            sizeTotal[i]=size[1];
        }
        for(var i=7;i<12;i++){

```

```

        sizeTotal[i]=size[2];
    }
    for(var i=12;i<15;i++){
        sizeTotal[i]=size[3];
    }
    for(var i=15;i<20;i++){
        sizeTotal[i]=size[4];
    }

    /*初始化一条染色体*/
    function initChar() {
        var arr = [];
        for(var i=0;i<bit;i++){
            getx(arr);
        }
        function getx(arr){
            for(var i=0;i>-1;i++){
                var flag = true;
                var num = Math.floor(Math.random()*20);
                for(var i in arr){
                    if(arr[i] == num){
                        flag= false;
                        break;
                    }
                }
                if(flag == true){
                    arr.push(num);
                }
                return;
            }
        }
    }

```

```

        }return arr;
    }

    /*初始化一个种群*/
    function initTotal() {
        for(var i=0;i<totalNum;i++){
            total[i] = initChar()
        }
        return total;
    }

```

```

    /*找出数组中的最大值*/
    function findMax(tmp){
        var max = tmp[0];
        for(var i=1;i<tmp.length;i++){
            if(max<=tmp[i]){
                max=tmp[i];
            }
        }
        return max;
    }

```

```

    /*找出数组中的最小值*/
    function findMin(tmp){
        var min=tmp[0];
        var index=0;
        for(var i=1;i<tmp.length;i++){
            if(min>tmp[i]){
                min=tmp[i];
                index=i;
            }
        }
    }

```

```

    }
    return min;
}

```

/*找出数组中最小值的 index*/

```

function findMaxIndex(tmp){
    var max=tmp[0];
    var index=0;
    for(var i=1;i<tmp.length;i++){
        if(max<tmp[i]){
            max=tmp[i];
            index=i;
        }
    }
    return index;
}

```

/*计算适应度函数（也是直接计算需要的矩形块数量）*/

```

function calculateFitness(arr) {
    var neededNum=1;
    var newArr=[];//记录每一行的矩形块的宽
    var addLenght=0;//累计长
    var remLength=0;//剩余长
    var fitness=0;
    var n=0;//累计在出现 addLenght>length 的情况时 i 循环的次数的个数
    var neededNum=0;
    var index1,index2;
    var count=0;//记录是否出现每行的 remLenth 还能再放一个矩形块的情况

    for(var i=0;i<arr.length;i++){

```

况

```

addLenght+=sizeTotal[arr[i]][0];
n++;
if(addLenght>length){

    if(i<n){
        for(var j=0;j<i;j++){
            newArr[j]=sizeTotal[arr[j]][1];
        }
    }else{
        for(var j=0;j<i;j++){
            newArr[j]=sizeTotal[arr[j]][1];
        }
        for(var a=0;a<i-n;a++){
            newArr.shift();
        }
    }
    remLength=length+sizeTotal[arr[i]][0]-addLenght;
    if(remLength>5){
        outerMost://定义外层循环
        for(var m=i;m<arr.length;m++){
            for(var h=0;h<2;h++){
                if( sizeTotal[arr[m]][h] <= remLength){
                    index1=m;
                    index2=h;
                    count=1;
                    if(index2==0){
                        newArr.push(sizeTotal[arr[index1]][1]);
                    }else{
                        newArr.push(sizeTotal[arr[index1]][0]);
                    }
                }
            }
        }
    }
}

```

```

        if(index1!=i){
            arr.splice(i,0,arr[index1]);
            arr.splice(index1+1,1);
        }
        if(i<arr.length-1){
            i++;
        }

        break outerMost;//终止外层循环
    }
}

}

}

fitness+=findMax(newArr);
addLenght=sizeTotal[arr[i]][0];
n=0;
}

var newArrRest=[];//储存排在最后一行的矩形块的宽
if(i==arr.length-1){
    if(count==0){
        for(var z=0;z<arr.length;z++){
            newArrRest[z]=sizeTotal[arr[z]][1];
        }
        for(var b=0;b<i-n;b++){
            newArrRest.shift();
        }
        fitness+=findMax(newArrRest);
    }else{
        for(var z=0;z<arr.length;z++){
            newArrRest[z]=sizeTotal[arr[z]][1];

```

```

        }
        for(var b=0;b<i-n-1;b++){
            newArrRest.shift();
        }
        fitness+=findMax(newArrRest);
    }
}
if(fitness>width){
    neededNum++;
    fitness=findMax(newArr);
}
count=0;
}
return 1/neededNum;
}

/*轮盘赌选择*/
function select() {
    var evals = new Array(totalNum); // 所有染色体适应值
    var p = new Array(totalNum); // 各染色体选择概率
    var q = new Array(totalNum); // 累计概率
    var F = 0; // 累计适应值总合
    for(var i=0;i<totalNum;i++){ // 记录下种群的最优解
        evals[i] = calculateFitness(total[i]);
        if(evals[i] > bestFitness) {
            bestFitness = evals[i];
            bestStr = total[i];
        }
        F += evals[i];
    }
}

```

```

for(var j=0;j<totalNum;j++){ // 计算累计概率
    p[j] = evals[j]/F;
    if(j == 0){
        q[j] = p[j];
    }
    else{
        q[j]=q[j-1]+p[j];
    }
}
var temp = new Array(totalNum);
for(var k=0;k<totalNum;k++){ //
    var r = Math.random();
    if(r <= q[0]){
        temp = total[0];
        break;
    }
    else{
        for(var z=1;z<totalNum;z++){
            if(r<q[z]){
                temp = total[z];
                break;
            }
        }
    }
}
return temp;
}

```



```

/*染色体交叉
*交叉概率为 70%
*/
function jiaocha(population1,population2){
    var returnPopulation1 = new Array(bit);
    var returnPopulation2 = new Array(bit);
    var temp=Math.random();
    if(temp<0.7){
        for(var i=0;i<bit;i++){
            if(i<bit-2){
                returnPopulation1[i]=population1[i];
                returnPopulation2[i]=population2[i];
            }else{
                returnPopulation1[i]=population2[i];
                returnPopulation2[i]=population1[i];
            }
        }
        //找出重合的基因,并将相同基因改正
        for(var j=bit-1;j>bit-3;j--){
            for(var k=0;k<bit-2;k++){
                if(returnPopulation1[k]==returnPopulation1[j]){
                    returnPopulation1[k]=returnPopulation2[j];
                }
            }
        }
    }else{
        returnPopulation1=population1;
        returnPopulation2=population2;
    }
    return returnPopulation1;
}

```

```
}
```

```
/*染色体变异*/
```

```
function change(arr) {  
    if(Math.random()<0.1){  
        while(1){  
            var temp1=parseInt(Math.random()*20);  
            var temp2=parseInt(Math.random()*20);  
            if(temp1!=temp2){  
                break;  
            }  
        }  
        var temp=arr[temp1];  
        arr[temp1]=arr[temp2];  
        arr[temp2]=temp;  
    }  
    return arr;  
}
```

```
/*执行过程*/
```

```
var d1 = new Date();  
document.getElementById('button').onclick = function () {  
    total = initTotal();  
    var arr=[];  
    var bestArr=[];  
    for(var i=0;i<1000;i++){  
        bestArr[i]=change(jiaocha(select(),select()));  
        arr[i] = calculateFitness(bestArr[i]);  
    }  
}
```

```
//var minFitness=findMin(arr);  
bestStr=bestArr[findMaxIndex(arr)];  
//console.log(bestStr);  
console.log(1/calculateFitness(bestStr));  
var d2 = new Date();  
//var time=d2-d1;  
//console.log('一共执行了'+time+'ms');  
}  
</script>  
</body>  
</html>
```