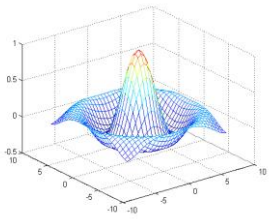
五一数学建模竞赛



题 目 木板最优切割方案的分析

关键词：木板切割，建模

摘　要：原材料作为生产过程中的关键部分，其消耗量的多少直接关系着企业的经济命脉。为了研究木板最优切割方案， 本文基于 DFS 算法，得出优化的木板切割方案，算法较为快捷。并应用本文提出的算法解决了五个问题。

最后，我们对模型进行评价，分析了本模型的优缺点，并指出了优化模型的改进方法。

Hollow Man

一、 问题重述

1.1 问题背景

在家具生产过程中，我们时常会遇到木板加工问题。生产厂家们总希望能够使一块木板的利用率最大，从而降低原材料费用，同时使一块木板获得最大的总利润。现在，利用优良的数学模型，我们能够规划出最优的木材切割方案。

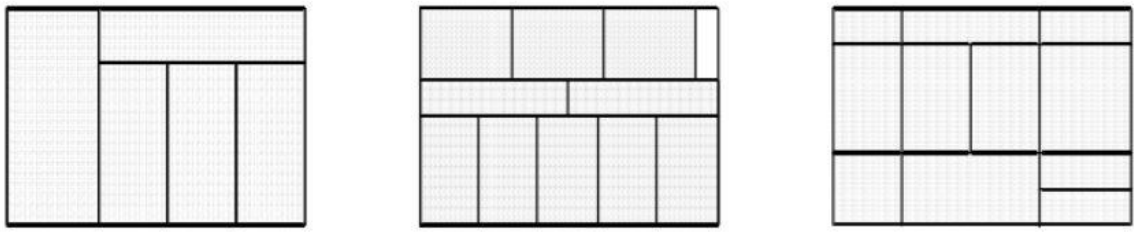
1.2 问题概述

本题给出了一款 3000*1500 mm 型号的原加工材料木板 S_1，同时给出了四种要用原材料木板加工的产品,分别是要生产 774 件的 373*201 mm 的 P_1,要生产 2153 件 477*282 mm 的 P_2，要生产 1623 件 406*229 mm 的 P_3，要生产 1653 件 311*225 mm 的 P_4。问题一要求解切割 S_1制得 P_1的最高利用率。问题二要求解切割 S_1制得 P_1和 P_3产品，并给出按照木板利用率由高到低排序的前 3 种切割方案。问题三要求解切割 S_1使得完成 P_1和 P_3生产任务，并使得模板总利用率最高。问题四要求解切割 S_1使得完成 P_1，P_2，P_3和 P_4生产任务，并使得模板总利用率最高。问题五要求解切割 100 张 S_1使得完成 P_1，P_2，P_3和 P_4生产任务，并使得模板总利用率最高。

二、 模型假设

我们准备使用 DFS 回溯算法建模进行求解。

我们为了优化计算，仅考虑直线切割的方法，即采用“一刀切”，又将一刀切分为准一刀切、正一刀切和亚一刀切。它们的区别在于：准一刀切在切割下料时沿板材两个边线中任意一个方向每次只能找出一条切割线来走刀；亚一刀切可以同时找到多条切割线，但每次能且仅能沿一个维度进行切割；正一刀切则是可以沿板材的任何一个维度方向进行切割，即切割线在排样图上的布局成“井”字型，这是一种最理想的切割状态。图 2.1 为这三种切割方式的示意图：



图(a) 准一刀切　　图(b) 亚一刀切　　图(c) 正一刀切

图 2.1

三、 问题分析

3.1 问题 1 的分析

要求解切割 S1 制得 P1 板的最高利用率，我们首先想到的是枚举出所有的可能性，选择出利用率最高的算法。因而采用 DFS 回溯算法，能够有效地解决这类问题。

因为仅要解切割 S_1 制得 P_1 板的最高利用率，所以我们采用 const 数组存储 P_1 板的长宽信息，然后，通过建立搜索树，进行递归切割，在递归切割的前提下进行回溯剪枝，利用限界函数来缩小时间复杂度。

3.2 问题 2 的分析

在第一问的基础上，此问加入了同时切割 P_3 板的要求。这需要我们对第一问的模型进行一些改进。我们在 const 数组中加入了 P_3 板的长宽信息来进行回溯搜索，并且增加了限界函数的判断条件。

3.3 问题 3 的分析

在第二问的基础上，此问加入了两种模板需要生产的数量要求，并且每块木板的切割方案相同。这需要我们对第二问的模型进行一些额外的改进。计算出每块木板能够切割出的 P_1 和 P_3 的比例。对同一个产品进行考察，随即规定 P_1 和 P_3 的比例系数，从而得出 P_1 和 P_3 在每个产品上需要裁剪的数量。在每次递归之时，增加判断，如果某一产品的裁剪数量已经达到了要求，那么我们便将这种产品从数组中剔除，不再进行裁剪。

3.4 问题 4 的分析

在第三问的基础上，此问加入了同时切割出 P_1，P_2，P_3，P_4 的要求，方法同第二问类似。

3.5 问题 5 的分析

同样，对于第五问，使用贪心算法，先算出每个木板利润与面积的比值，并且将其从大到小排序，优先满足比值高的产品的裁剪。

四、 模型建立与求解

4.1 问题 1

首先对 P_1 进行理论最大值的估算。

因为 P1 的长度为 373mm，宽度为 201mm，所以 P_1 需要木板的总面积为 74973mm^2，将 S1 板的面积 3000 X 1500 = 450000(mm^2) 与 P_1 的面积相除，结果取整约等于 60。所以，在不考虑形状的前提下，利用率最高能达到 99.96%。当然，实际结果和利用率均不会达到如此之高。经过 DFS 计算，我们得出了如表 1 所示的结果。

表 1 问题 1 的结果

P_1 的数量	木板利用率
57	95%

4.2 问题 2

同样地，采用 DFS，我们也可以得出如表 2 所示的结果。

表 2 问题 2 的结果

方案编号	P_1 的数量	P_3 的数量	木板利用率
1	42	13	96.83%
2	43	12	96.43%
3	49	7	96.09%

4.3 问题 3

运行程序，得到问题 3 的结果如表 3 所示。

表 3 问题 3 的结果

木板 S1 的数量	P_1 的数量	P_3 的数量	木板 利用率	备注
20	18	33	98.17%	每块木板切割方案相同
27	15	35	97.3%	同上 此行可根据需要增加
合计数量： 47	774	1623	木板 总利用率: 97.67%	木板总利用率=$\frac{\text{所有产品的总面积}}{\text{所有木板的总面积}}$

4.4 问题 4

运行程序，得到问题 4 的结果如表 4 所示。

表 4 问题 4 的结果

木板 S_1 的数量	P_1 的数量	P_2 的数量	P_3 的数量	P_4 的数量	木板利用率	备注
40	7	13	12	16	97.7%	每块木板切割方案相同
70	5	16	11	10	94.4%	同上 此行可根据需要增加
30	5	17	12	9	97.9%	
合计数量： 140	774	2153	1623	1614	木板 总利用率: 96.09%	木板总利用率 $=\frac{\text{所有产品的总面积}}{\text{所有木板的总面积}}$

4.5 问题 5

首先，我们算出产品的利润和面积的比值。对于 P_1，其面积为 7.4973dm^2，利润为 19.9 元/件，所以其比值为 2.65429 元/dm^2，对于 P_2，其面积为 13.4514dm^2，利润为 23.0 元/件，所以其比值 1.70986 元/dm^2，对于 P_3，其面积为 9.2974dm^2，利润为 21.0 元/件，所以其比值为 2.25870 元/dm^2，对于 P_4，其面积为 6.9975dm^2，利润为 16.0 元/件，所以其比值为 2.28653 元/dm^2。然后对其比值进行排序，依次是 $P_1 > P_4 > P_3 > P_2$。按次序优先满足，得出了表 5 的结论。

表 5 问题 5 的结果

木板 S_1 的数量	P_1 的数量	P_2 的数量	P_3 的数量	P_4 的数量	利润	木板利用率	备注
80	17	8	10	15	77784	96.22%	每块木板切割方案相同
20	15	10	11	13	19350	97.82%	同上 此行可根据需要增加
木板 S_1 合计数量 100	32	18	21	28	总利润: 97134	木板 总利用率: 96.54%	木板总利用率 $=\frac{\text{所有产品的总面积}}{\text{所有木板的总面积}}$

五、 模型的评价

5.1 模型的优点

模型使用 DFS 算法枚举，算法简单，易于实现能够获得较好的解。

5.2 模型的缺点

该模型缺易陷入局部最优的状态，而且算法复杂度较高，遇到大量数据之时，运算速度慢。而且容易出现左侧偏高的现象，材料利用率不高的问题。

5.3 模型的改进方法

针对提出模型的缺点，我们可采用模拟退火算法（SA）和遗传算法（GA）来解决。

模拟退火算法是一种通用的基于 Monte-Carlo 迭代求解策略的随机寻优算法，类似于物理中固体物质的退火过程。其基本思想是从一个给定解开始，通过使用产生器和接受准则，不断通过持续进行产生新解—判断—接受或舍弃的迭代过程（该过程也称冷却过程），将目前结构的解转变为邻近结构的解。最终趋于平衡状态对应于组合优化问题的整体最优解。

遗传算法（GA）是借鉴生物界的进化规律（适者生存，优胜劣汰遗传机制）演化而来的一种随机高效的搜索方法。通过模仿生物染色体中基因的选择、交叉和变异的自然进化程，进行个体重组形成一代代新群体，最终收敛于近似最优解。

这两种算法能够有效地避免局部最优解，从而达到全局最优解。

六、 参考文献

[1] 李蒙，基于一刀切的多原材二维下料协同优化方法研究，http://kns.cnki.net/KCMS/detail/detail.aspx?dbcode=CMFD&dbname=CMFD201401&filename=1013031475.nh&uid=WEEvREcwSlJHSldRa1FhdXNXaEd2UnVnclpZWWQxWFdIS0Z0SkRkdVd5RT0=$9A4hF_YAuvQ5obgVAqNKPCYcEjKensW4IQMovwHtwkF4VYPoHbKxJw!!&v=MjQ3MDZPZll1WnVGeURuV3IzTlZGMjZIYk83SDlYTHFwRWJQSVI4ZVgxTHV4WVM3RGgxVDNxVHJXTTFGckNVUkw=, 20190501;

七、 源代码

7.1 问题 1

输出为余料面积。

Java 实现代码：

```
import java.util.Scanner;

public class rr {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        double W, L, W1, L1;
```

```
        W = in.nextDouble();
        L = in.nextDouble();
        W1 = in.nextDouble();
        L1 = in.nextDouble();
        flag f1 = new flag(W, L, W1, L1, 10000, 10000);
        dfs(f1, 0);
        System.out.println(f1.ss);
        System.out.println(f1.tmpp);
    }

    public static double quyu(double x, double y) {
        int sum = 0;
        while ((sum += y) <= x) {

        }
        return x - (sum - y);
    }

    static public void dfs(flag s, int h) {
        double sss = 0;
        double[][] next = new double[2][2];
        next[0][0] = s.W1;
```

```
next[0][1] = 0;
next[1][0] = 0;
next[1][1] = s.W1;
if (!s.is()) {
    s.tmp = s.W * s.L;
    if (s.tmp < s.ss && s.W >= 0 && s.L >= 0) {
        s.ss = s.tmp;
        if (s.sum < s.tmpp)
            s.tmpp = s.sum;
    }
} else {
    for (int i = 0; i < 2; i++) {
        s.W = s.W - next[i][0];
        s.L = s.L - next[i][1];
        if (i == 0) {
            sss = s.W1 * (quyu(s.L, s.L1));
            s.sum += sss;
        } else {
            sss = s.W1 * (quyu(s.L, s.L1));
            s.sum += sss;
        }
        if (s.flag2[(int) s.W + 100][(int) s.L + 100] == false) {
```

```
                s.flag2[(int) s.W + 100][(int) s.L + 100] = true;
                dfs(s, h + 1);
                s.flag2[(int) s.W + 100][(int) s.L + 100] = false;
            } else
                continue;
        }
    }
}

class flag {
    public boolean[][] flag2;
    public double W;
    public double L;
    public double W1;
    public double L1;
    public double ss;
    public double tmp;
    public double sum;
    public double tmpp;

    public boolean is() {
```

```
        if ((W >= W1 && L >= L1) || (W >= L1 && L >= W1))
            return true;
        else {
            return false;
        }
    }

    public flag(double W, double L, double W1, double L1, double x, double y)
{
        this.W = W;
        this.L = L;
        this.W1 = W1;
        this.L1 = L1;
        this.ss = x;
        tmpp = y;
        flag2 = new boolean[300][300];
    }
}
```