网页广告的排版与招标研究

摘要

不同网站的广告展示区有所区别，如定高或定宽的一维广告条或者是有特定尺寸的二维矩形广告区，网站经营者面临着广告排版和招标的决策问题。在问题一中，我们建立并求解了线性规划模型来求解一维条形广告区排版问题。在问题二在问题三和问题四中，我们采取线性回归方法确定了不同位置下不同面积广告的价格，并且根据这一价格调整目标函数。依据修正后的目标函数约束条件，修改了问题一和二的优化模型，求得使经济收益最大的最优解。

在问题一中，我们需要对于一维条形广告区的排版问题进行建模并给出尽可能饱满的排版方案。我们可以把排版的饱满程度用排版部分的总面积与最大总面积之间的比值来表示，从而得到优化问题的目标函数。考虑到不能超出排版区域以及各个类型的广告具有的最大数量限制两类约束条件，我们建立了整数线性规划模型求解问题一。得到了包括 5 组 150×60,1 组 234×60,2 组 468×60 的情况在内的多种最优解。

在问题二中，我们需要对二维的矩形广告区的排版问题进行建模并给出尽可能饱满的排版方案。我们参考了矩形装箱问题，对矩形广告区排版问题建立了相似的数学规划模型，并参考了矩形装箱问题的 BL 方法，结合广告排版问题中广告的价格变化情况进行模型求解。最后得到的排版效果最优的模型的排版饱满程度约为 91%。

在问题三中，我们采取二元线性回归方法，依据附录中的广告招标数据确定了不同位置的广告的单价，并依据修改后的广告单价改写了问题一中优化模型的约束条件和目标函数。之后我们依据改写得到的优化模型确定了获利最多的排布情况为选择 2 组长 580 的订单和 1 组长为 720 的订单，并依据排布情况在可选范围内选择了能使收益最大化的订单组合，得到的最大收入为 2898.6。

在问题四中，我们依据在问题三中获得的不同位置的广告的单价的计算公式改写了广告价格的计算方法，并且采用了与问题三相似的方法改变了模型关于月度总频次的约束条件。从而求解得到最接近最优划分的可行解。针对可行解，我们在订单约束条件内通过**随机模拟**的方法遍历了各种订单的选取和组合，最终选择了总收入为 15396.5 的最优订单组合。

关键字：整数线性规划　二维矩形装箱问题　随机模拟

一、 问题重述

网页广告是广告投放的重要渠道之一, 也是最能带来流量的一种营销模式。不同网站设计的广告展示区也有所区别, 如定高或定宽的一维广告条或者是有特定尺寸的二维矩形广告区。网站每个季度都会收到大量的广告需求, 网站经营者面临着在考虑竞标广告的尺寸类型、频次要求、竞标价格等数据后的广告排版和招标的决策问题。

在问题一中，我们需要对于一维条形广告区的排版问题进行建模并给出排版方案。其中广告条的尺寸为 1920x60，用于排版的广告为高度为 60 的 A 类条形广告，排版应尽可能饱满（即空白部分尽可能小）。

在问题二中，我们需要对二维的矩形广告区的排版问题进行建模并给出排版方案。其中广告区的尺寸为 1280x480，可以使用 A，B，C 类广告进行填充，排版应当尽可能饱满。

在问题三和四中，我们需要考虑到实际的广告招标与报价情况，加入对广告的播放频次、位置要求、不同位置的报价等因素的考虑。其中，广告排版一般采取定期固定模式, 一个季度更换通常不超过 4 次。广告的报价和不同位置广告的点击率有关，一般满足左上价格 > 右上价格 > 左下价格 > 右下价格的关系。

在满足上述的新增条件的前提下，问题三中我们需要针对尺寸为 1920x60 的条形广告区域的广告需求进行估价并由此确定当季最佳招标方案，同时给出排版轮换计划。

问题四中我们需要对尺寸为 1280x480 的矩形广告区广告需求进行估价并由此确定当季最佳招标方案，同时给出排版轮换计划。

二、 问题分析

在问题一中, 我们需要对于一维条形广告区的排版问题进行建模并给出尽可能饱满的排版方案。我们可以把排版的饱满程度用排版的横向尺寸与最大横向尺寸 (1920) 之间的差来表示，从而得到优化问题的目标函数。考虑到不能超出排版区域以及各个类型的广告具有的最大数量限制两类约束条件，我们可以建立整数线性规划模型求解问题一。

在问题二中, 我们需要对二维的矩形广告区的排版问题进行建模并给出尽可能饱满的排版方案。我们可以把矩形广告区排版问题转化为指将不同尺寸的小矩形放入一个预先设定好尺寸的大矩形中, 以使大矩形的空间利用率达到最大的二维矩形装箱问题。但在模型的约束条件上有一定区别。通过查阅文献，我们发现 RPP 是一个典型的 NP-hard 问题，为了保证求解效率，我们可以参考文章 [1] 提出的启发式算法进行求解。

在问题三中，我们首先采取二元线性回归方法，依据附录中的广告招标数据确定了不同位置的广告的单价，并依据修改后的广告单价改写了问题一中优化模型的目标函数。考虑到第二问中给出了具体的广告时段和广告订单，我们还应该在问题一的模型的基础上增加关于月度和季度总频次的约束条件，从而建立改进后的数学规划模型。我们可以首先确定获利最多的排布情况，并依据排布情况在可选范围内确定能使收益最大化的订单。

在问题四中，我们依据在问题三中获得的不同位置的广告的单价的计算公式改写了广告价格的计算方法，并且采用了与问题三相似的方法改变了模型关于月度总频次的约束条件。从而可以求解几组得到接近最优划分的可行解。针对每组可行解，在订单约束条件内针对各种划分通过随机模拟的方法遍历各种可能的订单的选取和组合，最终选择收入最大的订单组合。

三、 模型假设及符号说明

3.1 模型假设

- 不考虑广告间存在的间隙
- 不考虑广告价格的随机因素
- 具体广告订单的付费按照季度内的最低频次要求确定
- 只在同类广告间进行轮换

3.2 符号说明

符号	意义
K	各种类型的广告组成的集合
x_{ijk}	判断 (i,j) 处是否为广告关键点的变量
w_k, h_k	广告 k 的宽度和长度
$p(x, y)$	(x,y) 处的广告价格
y_{abk}	判断第 a 月第 b 天广告 k 是否展出的变量

四、 问题一

4.1 模型建立

在问题一中，我们需要在要求排版尽可能饱满的前提下，建立 A 类广告在 1920x60 的广告条上排版的数学模型并给出排版方案。由于 A 类广告的高度均为 60，因而我们只需要考虑广告条在水平方向上的排布情况。

在问题一的一维排版问题中，由于广告的排列顺序不会影响广告覆盖的面积，因此我们只需要确定各种 A 类广告对应的数量就可以确定对应的排列方式。可以用向量 $X = (x_1, x_2, \cdots, x_6)$ 代表广告类别 $a_1, a_2, \cdots, a_6$ 的数量，向量 X 可以代表具体的排列方式。

我们可以使用模型实际进行排版的横向长度与最大横向长度 ($X_{max} = 1920$) 之间的差与最大横向长度的比值来表示排版的饱满程度，用向量 $A = (120, 150, 234, 468, 580, 760)^T$ 来表示广告类别 $a_1, a_2, \cdots, a_6$ 的长度，优化问题的目标函数可表示为：

$$\min \quad \frac{X_{max} - AX}{X_{max}} \tag{1}$$

不能超出排版区域进行以及各个类型的广告具有最大数量限制是我们的排版问题的主要约束条件。在确定了约束条件和目标函数后，我们可以建立整数线性规划模型求解问题一。

$$\begin{aligned} \min \quad & \frac{|X_{max} - AX|}{X_{max}} \\ s.t. \quad & \begin{cases} AX \leqslant X_{max} \\ x_i \leqslant 10 (i = 1, 2, \cdots, 6) \end{cases} \end{aligned} \tag{2}$$

4.2 模型求解

通过搜索我们得到了多种可以使得 $X_{max} - AX$ 为 0 的最优分配情况。用不同颜色代表不同的广告位，这里列举五种可行的分配情况：

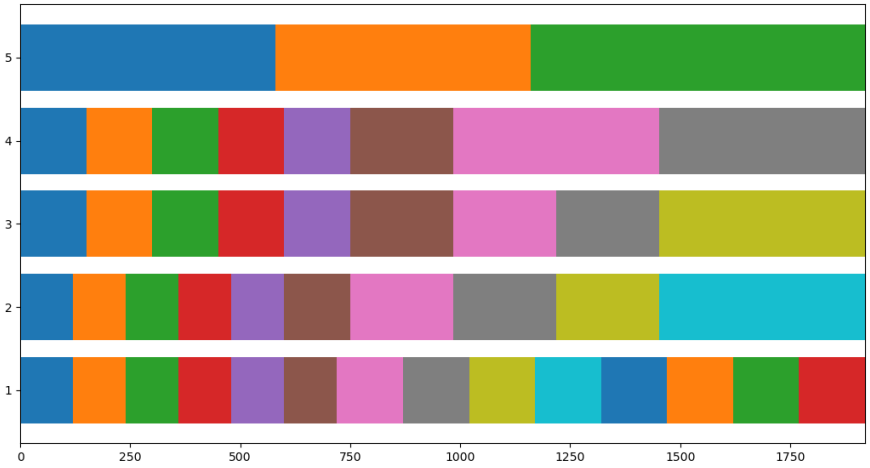


图 1　问题 1 中广告位的几种可能的最优分配情况

通过以上的分析，我们得到了对广告位的最优划分。由于在确定划分后调整划分内广告的相对顺序不会影响广告的总长度，因而在各个分类内不同长度的广告的相对位置可以任意交换。

五、 问题二

5.1 模型建立

二维矩形装箱问题是指将不同尺寸的小矩形放入一个预先设定好尺寸的大矩形中，以使大矩形的空间利用率达到最大的一类问题。在我们的网页广告排版问题中，我们可以把小矩形等价为各类等待排版的广告，并且把大矩形等价为广告区域，并且用与二维矩形装箱问题相似的方法进行模型建立和求解。

我们的问题相对二维矩形装箱问题的主要区别在于：

1. 广告有方向性，不能旋转，而二维装箱问题只限制了小矩形必须正交排布
2. 二维矩形装箱问题预先给定了待装箱的各个小矩形的尺寸，而我们的问题中无法事先确定小矩形的数量。

除了上述的限制之外我们的广告排版问题在在约束条件的确定和目标函数的确定等方面与二维矩形装箱问题有一定的相似性，可以参考这一问题的优化模型建立关于二维广告排版的数学模型，并且采用类似的启发式算法进行求解。

为了将二维广告排版问题进行数学描述，我们可以把广告区域看做一个以广告区域左上角为坐标原点，横向方向为 x 轴，纵向方向为 y 轴的二维直角坐标系。不妨假设所有类型广告的集合 K=0, 1, 2, $\cdots$, 14，则 w_k, h_k 分别表示第 k 种广告的宽和长。我们可

以用广告的右上顶点作为描述广告位置的关键点，根据广告的所属类别 k 得到广告的延伸范围，从而确定广告的具体位置。

我们可以建立描述广告的关键点位置的决策变量 x_{ijk}。变量 C 可以定义为：若 $C_{ij}=k$，则表明像素坐标 (i,j) 是型号为 k 的广告的左上角，此时 $x_{ijk}=1$，否则为 0，即：

$$x_{ijk}=\begin{cases}1 & C_{ij}=k\\ \\ 0 & else\end{cases} \tag{3}$$

在定义了描述广告位置的关键点，并且就各个顶点是否为关键点做出判断后，我们可以给出关于广告的面积和的目标函数。为了使排版尽可能饱满，我们应该使目标函数最大化，即：

$$\max \quad S=\sum_{i=1}^{1280}\sum_{j=1}^{480}\sum_{k=0}^{14}x_{ijk}w_k h_k \tag{4}$$

在实际的广告排版中，为了保证广告的展示效果，广告和广告之间不会重叠；此外，广告也不应当超出广告本身的所属区域，且各种广告的数量不得超过五个，关于这三个约束条件的具体解释如下如下。

关于广告不重叠的约束条件，根据上文的分析，一个广告的位置和范围可以由它所属的类别以及反映它所处位置的关键点来描述。为了使得广告不重叠，可以要求在一个广告的所属范围内只存在一个描述广告位置的关键点。这一约束条件可以表示为：

$$\sum_{i=\hat{i}}^{\hat{i}+h_k}\sum_{j=\hat{j}}^{\hat{j}+w_k}\sum_{k=0}^{14}x_{ijk}\leq 1 \quad \forall i=1,2,\cdots,1280; j=1,2,\cdots,480 \tag{5}$$

关于广告不超出广告区域的约束条件，由于在上面对 x_{ijk} 的定义中，我们考虑到模型的关键点不会出现在坐标系的坐标轴第一象限以外。因而限制 i和j 的值大于 0 就可以限制广告左上角的顶点位于第一象限内。因而这一约束条件可以描述为在 i和j 大于 0 的情况下，广告右下方的顶点位于广告区域内。

$$\forall i=1,2,\cdots,1280; j=1,2,\cdots,480; k=0,1,\cdots,14 \quad i\times x_{ijk}+h_k\leq 1280$$

$$\forall i=1,2,\cdots,1280; j=1,2,\cdots,480; k=0,1,\cdots,14 \quad j\times x_{ijk}+w_k\leq 480$$

关于同型广告的最大需求数量的约束条件，我们把第 k 类广告的需求数量表示为在广告区域内 k 类广告关键点的数目，从而得到下面的约束条件

$$\forall k\in\{0,1,\cdots,14\}\sum_{i=0}^{1280}\sum_{j=0}^{480}x_{ijk}\leqslant 5 \tag{6}$$

我们可以把优化问题整理如下：

$$
\begin{aligned}
&\max S = \sum_{i=1}^{1280}\sum_{j=1}^{480}\sum_{k=0}^{14} x_{ijk} w_k h_k \\
&\text{s.t.} \quad \sum_{i=\hat{i}}^{\hat{i}+h_k}\sum_{j=\hat{j}}^{\hat{j}+w_k}\sum_{k=0}^{14} x_{ijk} \le 1 \quad \forall i = 1,2,\cdots,1280; j = 1,2,\cdots,480 \\
&\forall i = 1,2,\cdots,1280; j = 1,2,\cdots,480; k = 0,1,\cdots,14 \quad i \times x_{ijk} + h_k \le 1280 \\
&\forall i = 1,2,\cdots,1280; j = 1,2,\cdots,480; k = 0,1,\cdots,14 \quad j \times x_{ijk} + w_k \le 480 \\
&\forall k \in \{0,1,\cdots,14\} \quad \sum_{i=0}^{1280}\sum_{j=0}^{480} x_{ijk} \leqslant 5
\end{aligned}
\tag{7}
$$

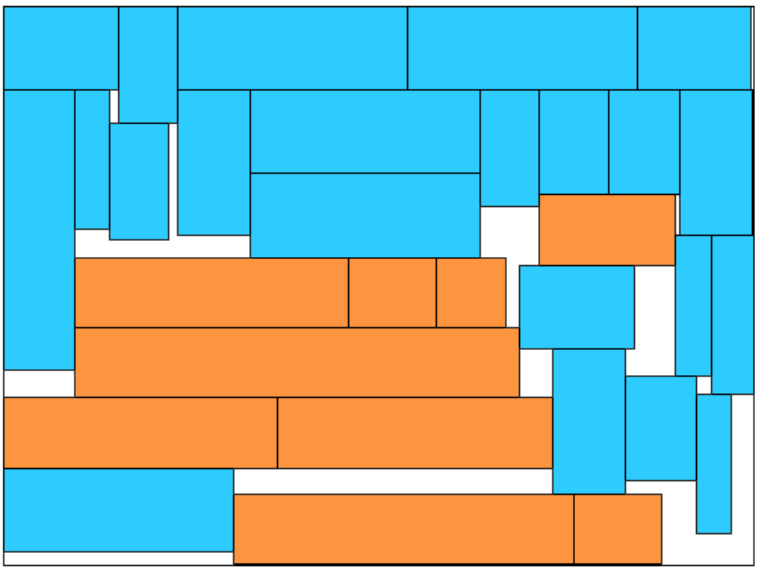
5.2 模型求解

在模型的具体求解中，由于二维矩形装箱问题是 NP-hard 问题，我们的问题相对二维矩形装箱问题由于选择的广告无法提前确定，因而有着比二维矩形装箱更高的时间复杂度，问题和二维矩形装箱问题同样无法在多项式级别的时间复杂度内进行求解。我们参考了二维矩形装箱问题中的 BL 算法这一启发式算法来求解问题。

在二维矩形装箱问题中，BL 算法考虑到装箱的实际规律，在每次装箱时先把随机选择的物品装入箱子的左下角，并且按照从上到下、从右至左的顺序逐步进行排列直到停留在稳定的位置。在我们的网页排版问题中，由于广告区域的左上方往往点击率较高，参考 BL 算法的设计思路，我们在设计广告排布区域是应当在广告区域的左上开始排列。考虑到网站的定价规则，我们选择从下到上、从右至左的排列方式进行排列，直到停留在稳定的位置。

在我们的算法中，不能够采用优先选择面积较大的广告进行放置的贪心策略作为对广告的最优放置策略。这是因为在放置某些较大的广告时，可能由于广告的部分边较长，放置广告时影响了后继广告的放置，从而使得模型整体无法取得最优解。事实上，任何一种简单的选取规则都不能使得全局的最优。因此, 本算法采用随机的方法对所有可行的放置策略进行选取, 并通过多次迭代计算求得最优解。

我们通过迭代算法得到的放置策略如下图所示，图中的蓝色代表 B 类广告，橙色代表 A 类广告。该种放置策略的填充率为 90.6%



六、 问题三

6.1 模型准备

在问题三中，我们需要确定不同位置不同尺寸的广告对应的价格。在实际中左上位置的广告点击率大于右上位置大于左下位置大于右下位置，当定价方案与被利用的广告的点击情况成正比时，可以使广告收入最大化。因而左侧和上侧的广告的价格更高。我们不妨以广告区域的左上角为坐标原点，以水平方向为 x 轴，建立描述广告区域的直角坐标系。

在选取代表广告位置的关键点时，考虑到广告价格的方向性，我们改为使用广告的几何中心来代表广告的所处位置。由于两个广告区的尺寸不同，因而我们可以预先对广告的坐标进行标准化，并且依据标准化后的数据分析 X 和 Y 坐标对广告价格的影响。

我们可以通过线性回归方法对不同位置广告单个像素对应的价格进行拟合，我们需要考虑到广告中心位置的横坐标和纵坐标，并且依据广告的尺寸计算不同位置的广告单个像素出租相同时间（77 天）对应的价格。设广告的中心坐标为 $P = (x, y)$ 我们可以建立下面的二元线性回归模型进行价格求解：

$$y = a * P^3 + b * P^2 + c * P + d \tag{8}$$

由于我们在这里考虑的单个像素在连续展出 77 天时的价格，已经在构建模型前考虑了时间和尺寸因素的影响，这里我们参考了各种模型的拟合效果，最终选择上面的三次多项式模型进行拟合。

针对招标广告 1 和招标广告 2 的数据进行拟合，得到的各个参数的具体值和置信区间如下表：

变量	取值	方差
a_1	[0.00979 , 0.01066]	[0.0325,0.0184]
b_1	[-0.00911 -0.02389]	[0.0469,0.0293]
c_1	[-0.00097 , 0.00687]	[0.0195,0.0135]
d_1	0.03533896	0.0023
a_2	[-0.02820 ,0.94192]	[0.0673,0.2308]
b_2	[0.07449 ,-1.31870]	[0.1037,0.325]
c_2	[-0.03953 , 0.52931]	[0.0465,0.1347]
d_1	0.01878	0.0168

观察上述的拟合结果，整体上 y 坐标的参数取值较大，方差较小，这说明上下方向对于广告定价的影响较大，与题目中左上定价 > 右上定价 > 左下定价 > 右下定价的条件相符。

6.2 模型建立

在问题三中，我们的目标函数不再是让排版尽可能饱满（最小化空白区域的面积），而是确定收益最大的招标方案。此外，在问题三中还给出了更为复杂和具体的广告订单约束情况，这增加了我们的约束条件。因而，我们在问题一建立的数学规划模型的基础上，结合不同位置广告的收益以及广告的具体频次要求和订单类型，建立改进后的数学规划模型求解问题 3。

在模型的目标函数方面，我们需要依据在模型准备部分得到的广告的位置和定价的关系修改问题一的模型中对应的目标函数。考虑到两组给出的广告区域的相同面积下的定价差异较大，考虑到通过回归分析得到的模型对数据的代表性一般，我们使用广告区域一的数据来对优化问题的目标函数进行修正。根据模型准备部分的求解结果，相对坐标为 (x, y) 的位置对应的广告价格为：

$$p(x, y) = -0.0022x - 0.0086y + 0.0377 \tag{9}$$

由于在问题三中广告区域和广告高度确定且相等，因而广告排版问题可以转化为一维直线的分割问题。考虑到位置对于广告价格的影响，我们选取广告区域的几何中心作为广告的关键点，在一维直线上，这一几何中心可以转换为广告在水平方向的中点。

用 $K=\{1,2,\cdots,6\}$ 代表 A 类内部的六种广告。若 $C_i=k$，则代表坐标 i 是 k 类广告的中点，我们可以建立描述广告关键点位置的决策变量 x_{ik}。

$$x_{ik}=\begin{cases}1 & C_i=k\\ 0 & else\end{cases} \tag{10}$$

我们在建立描述广告的关键点位置之后，用 w_k 和 h_k 代表各类广告的宽和长，则作为目标函数的广告总价格可以表示为：

$$\max P=\sum_{i=1}^{1920}\sum_{k=1}^{6}x_{ik}w_kh_k*p(i,30) \tag{11}$$

在模型的约束条件方面，问题三中的模型受到与问题一中模型相似的“广告不能超过广告区域的”范围约束。此外，在问题三中各个类型的订单有各自对应的最大订单数量以及订单排期约束，需要我们在问题一模型的基础上额外考虑。

用 $N=\{n_1,n_2,\cdots,n_6\}$ 代表 k 中各类样本的数量，则问题三中关于订单数量的约束条件可以表示为：

$$\forall k\in K\quad \sum_{i=1}^{1920}x_{ik}<=n_k \tag{12}$$

在确定了上述约束条件和目标函数后，我们可以使用与问题一相似的方法在不考虑具体的订单分配的情况下获得能够使得订单总收入最大的分配方案。在获得分配方案过程中的优化模型整理如下：

$$\begin{aligned}
&\max P=\sum_{i=1}^{1920}\sum_{k=1}^{6}x_{ik}w_kh_k*p(i,30)\\
\text{s.t.}\quad &\forall k\in K\quad \sum_{i=1}^{1920}x_{ik}<=n_k\\
&\sum_{i=\hat{i}}^{\hat{i}+h_k}\sum_{k=0}^{14}x_{ik}\le 1\quad \forall i=1,2,\cdots,1920\\
&\forall i=1,2,\cdots,1920;k=0,1,\cdots,14\quad i\times x_{ik}+h_k\le 1920\\
&p(x,y)=-0.0022x-0.0086y+0.0377;
\end{aligned} \tag{13}$$

在获得了收入最大的分配方案，确定了具体选用的广告类型和数量后，假设 m_k 为第 k 条广告月最低频次要求（$k=1,2,\cdots,73$），q_k 为第 k 条广告季度最低频次要求，K 为所有被展示广告的集合。假设一个季度有 3 个月，一个月有 30 天，我们定义决策变量 y_{abk}（$a=1,2,3;b=1,2,\cdots,30$），当 $y_{abk}=1$ 时，表示第 a 月的第 b 天广告 k 被展示，否则广告 k 不展示。

关于订单的选择过程我们给出下列约束条件：

1. 选择的广告的月展出时间达标

$$\sum_{j=1}^{30} x_{ijk} \geq m_k \quad \forall i = 1, 2, 3; k \in K \tag{14}$$

2. 选择的广告的季度展出时间达标：

$$\sum_{i=1}^{3} \sum_{j=1}^{30} x_{ijk} \geq q_k \quad k \in K \tag{15}$$

3. 在选择特定订单时排版位置受限：这一约束条件不易于用数学方式表示，但在搜索可行解时如果使用到排版受限的订单，应当对订单的具体排版方式提前检查。另外，由于问题一可以等价为在一维区域内排版，因而问题一不必考虑上下方向上的约束。

6.3 模型求解

我们可以根据上面的的优化模型，以最大化总收益为目标进行线性规划。由于不同的样本选择影响到各个类别的订单数目，进而使得样本的选择范围不同，我们可以依据总利润较大的几种情况（如下图）来分析可行的分配方案。得到的能够使得总利润较大的几种情况如下：

考虑到问题需要针对各种分配方案给出选择策略，我们可以采取贪心策略进行同类订单内的选择，即：优先选择时间接近 90 天，或者与其他订单组合后对应的订单时间接近 90 天的订单加入某个特定的订单类中。直到对广告区域内所有的订单类都给出了分配情况为止。通过上述方法计算得到几种能使得总收入最大的订单分配情况如下图：

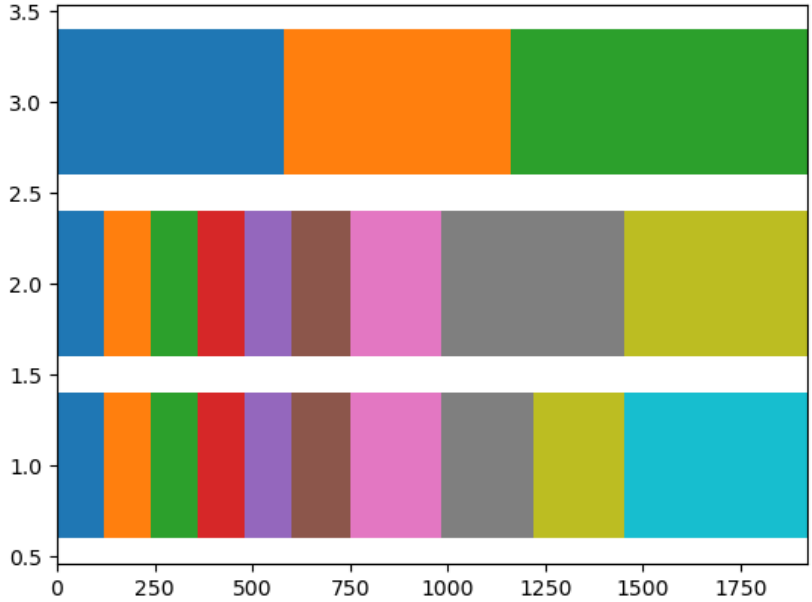


图 2　问题三的部分可行解

上图只包含了问题一结果中的部分情况，这是由于问题三的订单中对于订单数量的约束相对问题一更加严格。

在确定了具体选用的广告类型和数量后，我们可以采取贪心策略辅助订单内的选择，即：优先把订单在满足时间约束的条件下与其他订单组合。之后在所有组合后的较大的订单中随机进行选择。由于在一般情况下进行样本内订单选择的情况较少，因而采取贪心策略选择订单不会产生明显的误差。在把订单组合成较大的订单之后，我们可以为待分配的位置随机分配订单。并选取能够使得收入最多的分配方案。这种方案是选择 2 个 580*60 的广告和一个 760*60 的广告进行排布。

在具体的排布方式上，对于一维空间内的排布，由于左侧的广告价值较高，应该有限把排布较满，空闲时间较少的部分排在左侧。按照从左往右的顺序，最左侧 580*60 的广告位选择订单 17 和 18，而中间的 760*60 的广告位选择订单 23 和 24，并在完成排版任务后，插入时间限制少于或等于 10 天的订单 1,7 以及订单 15 或 16 进行排版，最右的 580*60 的广告位选择订单 22，并在中间广告位完成任务后适当左移。

按照上述的最优分配方案进行广告分配，得到的最大总收入为 2898.6。

七、 问题四

7.1 模型准备

在问题四中，相对广告区域形状相同的问题二，我们的目标函数不再是尽量最小化空白区域的面积，而是确定收益最大的招标方案。此外，在问题四中针对各类订单还给出了更为复杂和具体的，广告订单约束情况，这增加了我们的约束条件。因而，我们在问题一建立的数学规划模型的基础上，结合不同位置广告的收益、广告的具体频次要求和订单类型要求，建立改进后的数学规划模型求解问题 3。

在模型的目标函数方面，为了较好的确定不同位置的广告价格，我们可以沿用问题三中通过二元线性回归模型得到的广告位置与单价的关系，这一关系的具体表达如下：

$$p(x,y) = -0.0022x - 0.0086y + 0.0377 \tag{16}$$

问题四是一个二维空间内的广告排版问题，因而问题四的排版问题不能进行类似问题一的简化。考虑到位置对于广告价格的影响，我们改为选取广告区域的几何中心作为广告的关键点。用 $K = \{0, 1, \cdots, 14\}$ 代表 A 类内部的六种广告。若 $C_i = k$，则代表坐标 (i,j) 是 k 类广告的关键点，我们可以建立描述广告关键点位置的决策变量 x_{ijk}。

$$x_{ik} = \begin{cases} 1 & C_i = k \\ 0 & else \end{cases} \tag{17}$$

我们在建立描述广告的关键点位置之后，用 w_k 和 h_k 代表各类广告的宽和长，则作为目标函数的广告总价格可以表示为：

$$\max P = \sum_{i=1}^{1280}\sum_{j=1}^{480}\sum_{k=0}^{14} x_{ijk} w_k h_k * p(i,j) \tag{18}$$

在模型的约束条件方面，问题四中的模型受到与问题二中模型相似的“广告不能超过广告区域的”以及“广告间不能重叠”的范围约束。此外，在问题四中各个类型的订单有各自对应的最大订单数量以及订单排期约束，需要我们在问题二模型的基础上加以考虑。此外，某些订单有特定的空间约束，这一约束也应当在模型求解时纳入考虑。

用 $N = \{n_0, n_1, \cdots, n_{14}\}$ 代表 k 中各类样本的数量，则问题四中关于订单数量的约束条件可以表示为：

$$\forall k \in K \quad \sum_{i=1}^{1920} x_{ik} <= n_k \tag{19}$$

在确定了上述约束条件和目标函数后，我们可以使用与问题二相似的方法在不考虑具体的订单分配的情况下获得能够使得订单总收入最大的分配方案。在获得分配方案过程中的优化模型整理如下：

$$\begin{aligned}
&\max P = \sum_{i=1}^{1280}\sum_{j=1}^{480}\sum_{k=0}^{14} x_{ijk} w_k h_k * p(i,j) \\
&\text{s.t.} \quad \sum_{i=\hat{i}}^{\hat{i}+h_k}\sum_{j=\hat{j}}^{\hat{j}+w_k}\sum_{k=0}^{14} x_{ijk} \le 1 \quad \forall i = 1,2,\cdots,1280; j = 1,2,\cdots,480 \\
&\forall i = 1,2,\cdots,1280; j = 1,2,\cdots,480; k = 0,1,\cdots,14 \quad i \times x_{ijk} + h_k \le 1280 \\
&\forall i = 1,2,\cdots,1280; j = 1,2,\cdots,480; k = 0,1,\cdots,14 \quad j \times x_{ijk} + w_k \le 480 \\
&\forall k \in K \quad \sum_{i=1}^{1920} x_{ik} <= n_k
\end{aligned} \tag{20}$$

在获得了收入最大的分配方案，确定了具体选用的广告类型和数量及所处位置后，假设 m_k 为第 k 条广告月最低频次要求（$k = 1,2,\cdots,73$），q_k 为第 k 条广告季度最低频次要求，K 为所有被展示广告的集合。假设一个季度有 3 个月，一个月有 30 天，我们定义决策变量 y_{abk}（$a = 1,2,3; b = 1,2,\cdots,30$），当 $y_{abk} = 1$ 时，表示第 a 月的第 b 天广告 k 被展示，否则广告 k 不展示。关于订单选择过程我们给出下列约束条件：1. 选择的广告的月展出时间达标

$$\sum_{j=1}^{30} x_{ijk} \ge m_k \quad \forall i = 1,2,3; k \in K \tag{21}$$

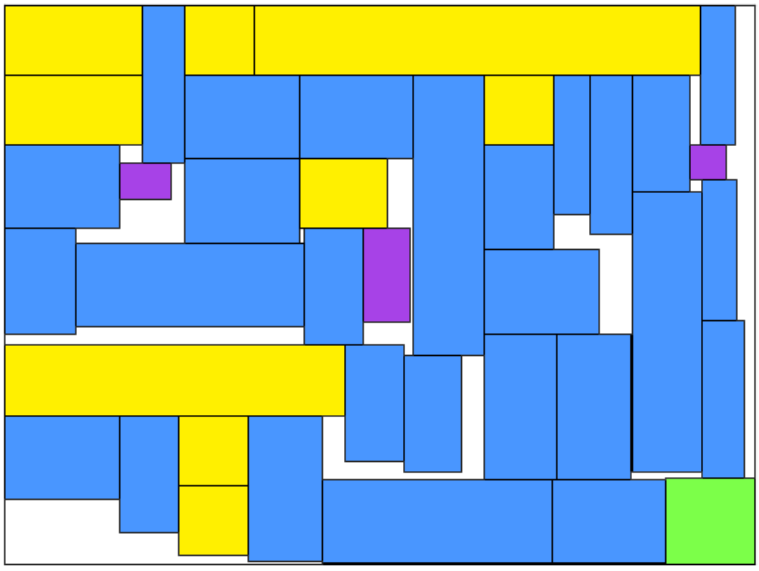
2. 选择的广告的季度展出时间达标：

$$\sum_{i=1}^{3}\sum_{j=1}^{30} x_{ijk} \geq q_k \quad k \in K \tag{22}$$

3. 在选择特定订单时排版位置受限：这一约束条件不易于用数学方式表示，但在搜索可行解时如果需要把排版受限的订单分配到特定位置，应当检查这一位置是否与约束相符。

7.2 模型求解

我们可以根据上面的的优化模型，以最大化总收益为目标进行优化。由于不同的样本选择影响到各个类别的订单数目，进而使得样本的选择范围不同，我们可以首先依据优化模型20，依据总利润最大的情况来分析可行的分配方案。得到的分配方案如下（图中的黄色代表 A 类广告，蓝色代表 B 类广告，紫色代表 C 类广告，绿色代表公益广告）：



在确定了广告类型、位置和数量后，我们可以采取贪心策略对类内的各个订单进行预处理，即：优先选择时间接近 90 天，或者与其他订单组合后对应的订单时间接近 90 天的订单加入某个特定的订单类中。直到对广告区域内所有的订单类都给出分配情况为止。

在采取贪心策略辅助订单内的小订单组合成较大的订单之后，我们可以首先将待分配订单的顺序打乱以避免贪心算法无法获得全局最优解的缺陷。之后针对图中各个待分配的单元格从随机分配后的序列中进行选取。通过多次迭代搜索来获得最优的分配方案。

我们按照上述的最优分配方案进行多次迭代搜索，得到的最优分配方式对应的总收入为 15396.5。

八、 模型评价与推广

8.1 模型优点

- 在问题一和三中通过线性规划方法求得了最优解
- 模型在问题二和四中充分考虑到了各种可能出现的约束条件，并通过随机模拟方法求得了可行解

8.2 模型缺点

- 基于 BL 算法思想的随机模拟算法的时间复杂度较高
- 问题四中广告的分配方式是确定的，不会随着时间的推移而改变，可能会导致模型无法求得最优解

参考文献

[1] 尚正阳, 顾寄南, 丁卫,Enock A.Duodu. 求解二维矩形装箱问题的启发式算法 [J]. 计算机集成制造系统,2018,24(03):583-590.

[2] 金洁. 二维矩形装箱问题及其算法设计 [D]. 云南大学,2015.

[3] Baker B S , Coffman E G , Rivest R L . Orthogonal Packings in Two Dimensions[J]. SIAM Journal on Computing, 1980, 9(4):846-855.

附录 A　问题一中进行线性规划的 Pyhton 源程序

```
import numpy as np
from scipy.optimize import linprog
import cvxpy as cp
c=np.array([120,150,234,468,580,760])
a=np.array([[120,150,234,468,580,760]])
b=np.array([1920])
# res=linprog(-c,a,b,bounds=([0,10],[0,10],[0,10],[0,10],[0,10],[0,10]))
# print(res)
x=cp.Variable(6,integer=True)
obj=cp.Maximize(cp.sum(c@x))
const=[x>=0,x<=10,a@x<=b]
prob=cp.Problem(obj,const)
res=prob.solve(solver=cp.CPLEX)
print(prob.value,x.value)
```

附录 B　求解问题二的 Python 源程序

```
import numpy as np
import random
import matplotlib.pyplot as plt
import matplotlib.patches as mthes

def areaCheck(loc,size,vmatr):
    for i in range(loc[0],loc[0]+size[0]):
        for j in range(loc[1],loc[1]+size[1]):
            if vmatr[i][j]==1:
                return False
    return True

def judgement(loc,size,limit,vmatr):
    if loc[0]+size[0]>=limit[0] or loc[1]+size[1]>=limit[1]:
        return False
    elif not areaCheck(loc,size,vmatr):
        return False
    return True

def turnToVisted(loc,size,vmatr):
    for i in range(loc[0],loc[0]+size[0]):
        for j in range(loc[1],loc[1]+size[1]):
            vmatr[i][j]=1
    return vmatr
```

```
def generateInit(adss):
    res=[]
    for ads in adss:
        for item in ads:
            for time in range(5):
                res.append(item)
    return res

adsA=[(120,60),(150,60),(234,60),(468,60),(580,60),(760,60)]
adsB=[(60,120),(72,136),(100,100),(120,90),(196,72),(125,125),(120,240),(392,72)]
adsC=[(60,30),(88,31),(80,80)]

vmatr=[]
height=1280
width=480

for i in range(height):
    vmatr.append([])
    for j in range(width):
        vmatr[i].append(0)

totsquare=height*width
resquare=0
ax=plt.subplot()
init_choice=generateInit([adsA,adsB])
itetime=20
reschoice=[]

for ite in range(itetime):
    choice=init_choice.copy()
    np.random.shuffle(choice)
    for i in range(height):
        for j in range(width):
            vmatr[i][j]=0
    tempchoice=[]
    tempsquare=0
    for item in choice:
        for i in range(width):
            flag=False
            for j in range(height):
                if judgement([j,i],item,[height,width],vmatr):
                    flag=True
                    tempchoice.append((j,i,item))
                    vmatr=turnToVisted([j,i],item,vmatr)
                    break
            if flag:
                tempsquare+=item[0]*item[1]
```

```
            break
    print(ite,tempsquare/totsquare*100)
    if tempsquare>resquare:
        resquare = tempsquare
        reschoice = tempchoice

for item in reschoice:
    if item[2] in adsA:
        rec = mthes.Rectangle([item[0],item[1]], item[2][0], item[2][1], linewidth=1,
            edgecolor='black', facecolor='r')
    else:
        rec = mthes.Rectangle([item[0], item[1]], item[2][0], item[2][1], linewidth=1,
            edgecolor='black', facecolor='b')
    ax.add_patch(rec)

print('利用率: ',resquare/totsquare*100,'%')
ax.set_xlim([0,1280])
ax.set_ylim([480,0])
plt.show()
```

附录 C　求解问题四的 Python 源程序

```
import numpy as np
import random
import matplotlib.pyplot as plt
import matplotlib.patches as mthes

def areaCheck(loc,size,vmatr):
    for i in range(loc[0],loc[0]+size[0]):
        for j in range(loc[1],loc[1]+size[1]):
            if vmatr[i][j]==1:
                return False
    return True

def judgement(loc,size,limit,vmatr):
    if loc[0]+size[0]>=limit[0] or loc[1]+size[1]>=limit[1]:
        return False
    elif loc[0]+size[0]>=limit[0]-60 and loc[1]+size[1]>=limit[1]-60:
        return False
    elif not areaCheck(loc,size,vmatr):
        return False
    return True

def turnToVisted(loc,size,vmatr):
    for i in range(loc[0],loc[0]+size[0]):
```

```
        for j in range(loc[1],loc[1]+size[1]):
            vmatr[i][j]=1
    return vmatr

def valueFunc(x,y):
    return -0.0022*x-0.0086*y+0.0377

def RelativeCenter(x,y,size,limit):
    return (x+size[0]/2)/limit[0],(y+size[1]/2)/limit[1]

adsA=[(120,60),(150,60),(234,60),(468,60),(580,60),(760,60)]
adsB=[(60,120),(72,136),(100,100),(120,90),(196,72),(125,125),(120,240),(392,72)]
adsC=[(60,30),(88,31),(80,80)]
totads=((120,60),(150,60),(234,60),(468,60),(580,60),(760,60),
        (60,120),(72,136),(100,100),(120,90),(196,72),(125,125),(120,240),(392,72),
        (60,30),(88,31),(80,80))
# freqreq=[[(10,0),(0,49),(15,49),(15,56),(20,63),(20,70)],
#          [(10,49),(15,0),(0,56)],
#          [(10,42),(15,56),(20,63),(0,28),(0,77)],
#          [(10,35),(10,49)],
#          [(10,42),(15,0),(0,56),(20,70),(25,77),(0,84)],
#          [(0,35),(10,42)],
#          [(10,35),(10,42),(15,49),(15,56),(20,63),(0,77)],
#          [(15,49),(0,56),(20,70),(25,77),(0,63),(0,77)],
#          [(0,35),(10,49),(15,49),(20,63),(0,70),(0,77)],
#          [(0,42),(10,42),(10,49),(15,56),(20,63),(0,70),(25,0)],
#          [(10,42),(0,49),(15,49),(15,56),(20,63),(0,70),(0,77)],
#          [(15,56),(20,70),(0,70),(25,77)],
#          [(10,49),(0,56),(15,56),(20,63),(0,63),(0,70)],
#          [(0,63),(20,70),(0,77)],
#          [(0,84)],
#          [(0,84)],
#          [(0,84)]]

Lreq=[(0,2),(9,2)]
Ureq=[(0,3),(3,0),(5,0),(7,5),(11,1)]
LUreq=[(1,1),(6,1),(13,0)]

realreq=[[49, 56, 63, 70, 79],
         [49, 45, 56],
         [42, 63, 77, 84],
         [84],
         [56, 70, 77, 84, 87],
         [84],
         [42, 49, 63, 77, 84],
         [49, 56, 70, 77, 63, 77],
         [49, 63, 70, 77, 84],
```

```
        [49, 56, 63, 70, 75, 84],
        [42, 49, 49, 56, 63, 70, 77],
        [56, 70, 70, 77],
        [49, 56, 56, 63, 63, 70],
        [63, 70, 77],
        [84],
        [84],
        [84]]
reqdict={}
init_choice=[]

for i in range(len(realreq)):
    for j in range(len(realreq[i])):
        init_choice.append((i,j))
        reqdict.update({(i,j):totads[i]})

ax=plt.subplot()
vmatr=[]
height=1280
width=480
itetime=5
reschoice=[]
resvalue=0

for i in range(height):
    vmatr.append([])
    for j in range(width):
        vmatr[i].append(0)

for ite in range(itetime):
    choice = init_choice.copy()
    np.random.shuffle(choice)
    for i in range(height):
        for j in range(width):
            vmatr[i][j] = 0
    tempchoice = []
    tempvalue = 0
    for item in choice:
        siz = reqdict[item]
        idi = item[0]
        idj = item[1]
        x = 0
        y = 0
        for i in range(width):
            flag = False
            for j in range(height):
                if judgement([j, i], siz, [height, width], vmatr):
```

```
                if item in Lreq and j<320:
                    flag = True
                    x = j;y = i
                    vmatr = turnToVisted([j, i], siz, vmatr)
                    break
                elif item in Ureq and i<120:
                    flag = True
                    x = j;y = i
                    vmatr = turnToVisted([j, i], siz, vmatr)
                    break
                elif item in LUreq and j<320 and i<120:
                    flag = True
                    x = j;y = i
                    vmatr = turnToVisted([j, i], siz, vmatr)
                    break
                elif item not in Lreq and item not in Ureq and item not in LUreq:
                    flag = True
                    x = j;y = i
                    vmatr = turnToVisted([j, i], siz, vmatr)
                    break
            if flag:
                rela_x, rela_y = RelativeCenter(x, y, siz, [height, width])
                va=siz[0]*siz[1]*valueFunc(rela_x,rela_y)*realreq[idi][idj]/77
                tempchoice.append((x, y, siz, item,va))
                tempvalue += va
                break
    print(ite, tempvalue)
    if tempvalue > resvalue:
        resvalue = tempvalue
        reschoice = tempchoice
for item in reschoice:
    if item[2] in adsA:
        rec = mthes.Rectangle([item[0],item[1]], item[2][0], item[2][1], linewidth=1,
            edgecolor='black', facecolor='#FFF000')
    elif item[2] in adsB:
        rec = mthes.Rectangle([item[0], item[1]], item[2][0], item[2][1], linewidth=1,
            edgecolor='black', facecolor='#4996FF')
    elif item[2] in adsC:
        rec = mthes.Rectangle([item[0], item[1]], item[2][0], item[2][1], linewidth=1,
            edgecolor='black', facecolor='#A742E7')
    ax.add_patch(rec)
print(reschoice)
print('总价值：',resvalue)
ax.set_xlim([0,1280])
ax.set_ylim([480,0])
plt.show()
```