基于优化理论的任务定价与分配模型

摘要

APP 拍照任务定价问题是一个任务发布者（APP 平台）和任务完成者（会员）间双向决策问题。任务发布者希望在任务成本较小的情况下，任务的完成量尽可能的大；而作为任务完成者追求任务完成后的收益尽可能的大。

问题 1：我们得到任务定价主要与任务附近范围内的会员密集程度、给定区域内任务集中程度、任务难易程度有关。根据所给数据，通过定量分析方法，得到任务的定价与上述三个因素满足规律 $P_i = P_0 + 0.5R_i + S_i - Q_i$。针对该定价规律，我们分析了任务未完成的原因主要有：（1）任务本身定价较低，使会员对该任务的满意度未达到期望值；（2）高收入地区会员期望值较高，导致在其他地区能够完成的任务在该地区未被完成；（3）高信誉会员的任务额较高，而且有优先选择任务的权力，预约任务过多，导致很多任务没及时完成。

问题 2：首先，针对问题一定价规律中存在的缺陷，我们对问题一中的定价模型进行改进。（1）对于经济发达的地区，会员的收益期望值较高，在定价时，应当适当提高了该地区任务的定价；（2）考虑到高信誉会员优先选择任务和限额的特点，在计算会员密集程度时，把一个高信誉会员当做若干个会员处理。建立了新的定价模型。其次，根据新的定价模型，以任务有效完成量最大为目标，以会员预定限额、会员接单期望、不同城市经济发展水平的用户期望值等条件为约束，建立了任务分配的优化模型。考虑到模型约束条件比较复杂，现有方法很难对模型进行求解，我们设计了一种基于最大流的启发式算法，利用 MATLAB 编程，对模型进行求解。求解结果与问题一比较，在总费用基本不变的情况下，使得任务的完成率提高了 24.5%。

问题 3：为了进一步提高任务的完成率，借鉴了商品的打包销售方法，我们给出三条具体的打包原则：（1）距离相近的任务（集中度较高的任务）应考虑打包发布；（2）未完成的任务应尽量与自己距离相近的已完成的任务打包发布；（3）距离相近的价格差比较大的任务应尽量考虑打包发布。按照该原则，我们对任务进行打包处理，把每个包看作一个新的任务。类似于问题二，重新建立了打包情况下的任务定价模型和任务分配优化模型。通过模型的求解，得到在总费用大致不变的情况下，使任务的总完成率在问题二的基础上又提高了 7.2%。

问题 4：对于给定的任务地理位置和会员的分布情况，由于此时任务的难易程度未知，在问题三的定价模型中，舍弃了问题难易度因素，只利用会员密集程度、给定区域内任务集中程度、不同地区会员的期望值不同等因素，建立了任务定价模型和任务分配优化模型。得到任务的完成率是 87.8%，较好地解决了任务的定价和分配问题。

定价模型考虑了会员密集程度、任务集中度、任务难易程度等因素。任务优化分配模型提高了任务的有效完成率，基于最大流的启发式算法计算精度高、运算时间短。

关键词： 任务定价 定价规律 打包原则 优化模型

1 问题重述

1.1 问题的背景

随着移动互联网的高速发展，“拍照赚钱”这种基于移动互联网的自助式劳务众包平台应运而生。相比传统的市场调查，它不但节省调查成本，而且有效地保证了调查数据真实性，缩短了调查的周期。APP 作为“拍照赚钱”平台运行的核心。用户在 APP 上注册会员，完成所领取的拍照任务后即可赚取标定的酬金。而 APP 中任务的定价是维持平台运行的核心要素，APP 拍照任务定价问题是一个任务发布者（APP 平台）和任务完成者（会员）间双向决策问题。任务发布者希望在任务成本较小的情况下，任务的完成量尽可能的大；而作为任务完成者追求任务完成后的收益尽可能的大。若定价不合理，会使任务无人问津。因此，如何合理定价成为当下一个非常热门而重要的话题。

1.2 问题的相关信息

根据题目提供的相关信息，可知如下数据条件：

附件 1 给出了 835 个任务的位置、定价和完成情况，其中“1”表示完成，“0”表示未完成；

附件 2 包含了 1877 个会员的位置、信誉值、任务开始预订时间和预订限额。原则上会员信誉越高，越优先挑选任务，其配额就越大；

附件 3 给出了 2066 个任务的位置信息。

1.3 需解决的问题

问题一：分析附件 1 中的任务数据，建立相应的数学模型，研究项目的任务定价规律，并对未完成的任务找出原因；

问题二：为附件 1 中的项目设计一种新的任务定价方案，并与原方案进行比较；

问题三：为了提高项目的完成数量，给出将任务联合打包的方案，改进问题二中的定价模型，并分析打包对任务完成情况的影响。

问题四：利用所建立的定价模型给出附件 3 中新项目的任务定价方案，并评价该方案的实施效果。

2 模型假设和符号说明

2.1 模型的假设

（1）假设任意两点之间的实际路程可以用直线距离近似代替；

（2）假设每个会员都处于在线状态，是否接单取决于对任务的满意度是否大于期望值；

（3）假设会员在其预定任务开始时间 30 分钟内挑选任务。预定任务开始时间，会员越早开始挑选任务；

（4）假设所有会员理性挑选任务，不存在会员随意抢单的情况。

2.2 符号的说明

符号	说明
$d(i,j)$	会员 j 和任务 i 之间的距离
κ_i	任务 i 完成的可能性指标
$\gamma_1,\gamma_2,\gamma_3,\gamma_4$	佛山、广州、东莞和深圳四块区域期望值阈值
V_j	会员 j 愿意接受任务点的集合
e_j	会员 j 所能接受任务限额
Q_j	会员 j 的信誉值
t_j	会员 j 开始预定任务的时间
t_j'	会员 j 完成任务的时间
d_{jA}	会员 j 经过集合 A 中所有点的最短距离
D	任务完成所需总金额限制

3 数据预处理

3.1 经纬度-距离转换[1]

为了得到任意两个任务、会员或会员与任务间的直线距离，我们对经纬度进行转换。

地球上任意一点地理坐标都可以用有序数对表示为 (u,v)，u 为经度，v 为纬度。以地心 O 为坐标原点，赤道平面为 xOy 平面，0 度经线圈所在的平面为 xOz 平面建立三维直角坐标系，则

$$\begin{cases} x = R\cos u\cos v \\ y = R\sin u\cos v \\ z = R\sin v \end{cases}, \qquad (3—1)$$

其中，$R=6370$ 为地球半径。

根据解析几何的知识，任意两点 $A(u_A,v_A),B(u_B,v_B)$ 间实际距离为

$$d = R\arccos\left(\frac{OA\cdot OB}{|OA|\cdot|OB|}\right),$$

将式（3—1）代入化简得

$$d = R\arccos[\cos(u_A - u_B)\cos v_A \cos v_B + \sin v_A \sin v_B].$$

4 问题一的模型建立与求解

4.1 数据的描述分析

根据附件 1、2 中的经纬度信息，我们将任务、会员标注在二维地图上，并利用任务标价绘制热图。图 4-1 绘制了项目任务的地理位置分布，其中红点表示已完成任务，黑点表示未完成任务；图 4-2 绘制了会员的地理位置分布。根据任务定价绘制热图，颜色从蓝过渡到红，表示定价从低到高。

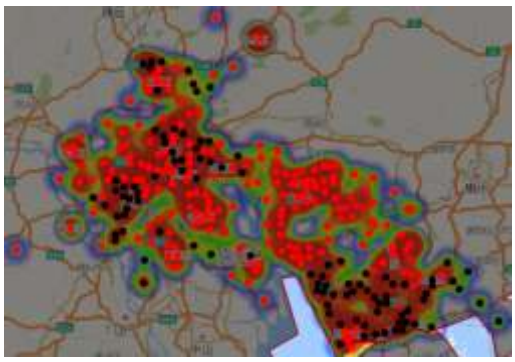


图 4-1 任务分布与标价关联图

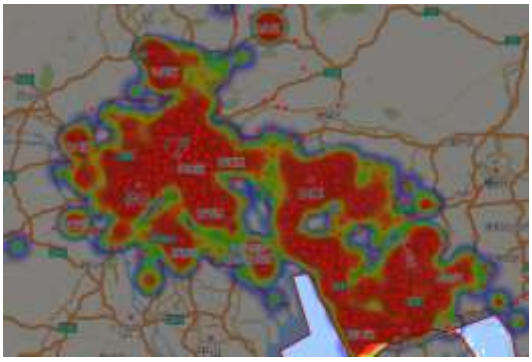


图 4-2 会员分布与标价关联图

从上述两图中可以看出，价格较高的任务主要集中在城市内部，且大致可以按城市划分成佛山、广州、东莞和深圳四块区域。左图反映了任务价格较高的点与任务的密集程度有关，且任务越密集，价格越高，但是价格的高低不能完全决定任务的完成情况；右图反映了会员的分布情况，会员在该区域的分布较为均匀。

4.2 影响定价的指标因素

参照出租车的定价策略[2]，我们分析得到任务附近邻域内会员密集程度、任务集中程度、任务难易程度、不同经济地区会员的期望值是影响任务定价 P_i 的主要因素。

会员密集程度 m_i：

设 m_i 是以任务点 i 为中心，半径 d 范围内的会员数，以 m_i 作为任务点 i 会员集中程度的指标。定义函数

$$\chi_{ij} = \begin{cases} 1, d(i,j) \le d \\ 0, d(i,j) > d \end{cases}, i \in E, j \in F,$$

其中，$d(i,j)$ 为任务 i 与会员 j 两点间的直线距离，F 为会员集合，E 为任务集合。

因此，该范围内的会员数可表示为

$$m_i = \sum_{j=1}^{n_2} \chi_{ij}.$$

利用 MATLAB 软件，绘制价格与会员数 m_i 的二维散点图 4—3。

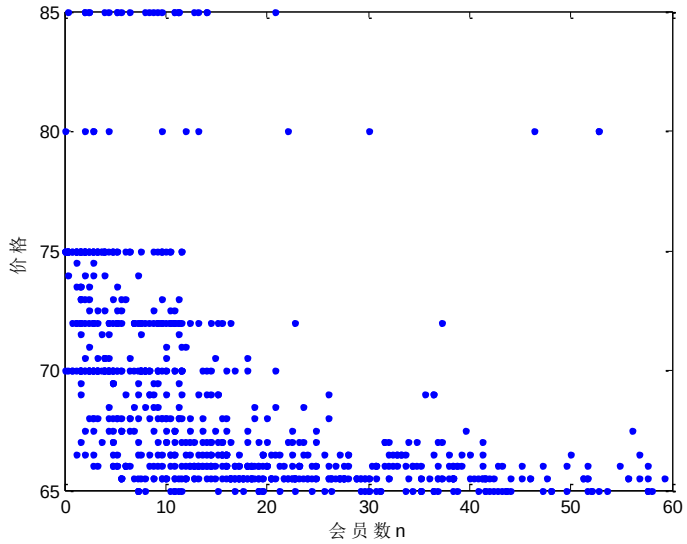


图 4-3 任务定价与会员数关系图

会员密集程度反映了会员在某一区域内的集中情况。会员集中程度越高，该任务被完成的可能性越大，此时该任务的定价就越低，则会员数 m_i 与定价呈负相关。

任务集中程度 q_i：

设 q_i 为以任务点 i 为中心，半径 d 范围内的任务数，以 q_i 作为任务点 i 附近任务集中程度的指标。定义函数

$$\delta_{ij}=\begin{cases}1, d(i,j)\le d\\ 0, d(i,j)>d\end{cases}, i\in E, j\in F,$$

因此，任务集中程度

$$q_i=\sum_{j=1}^{n_1}\delta_{ij}.$$

利用 MATLAB 软件，绘制价格与任务集中程度 q_i 的二维散点图 4—4。

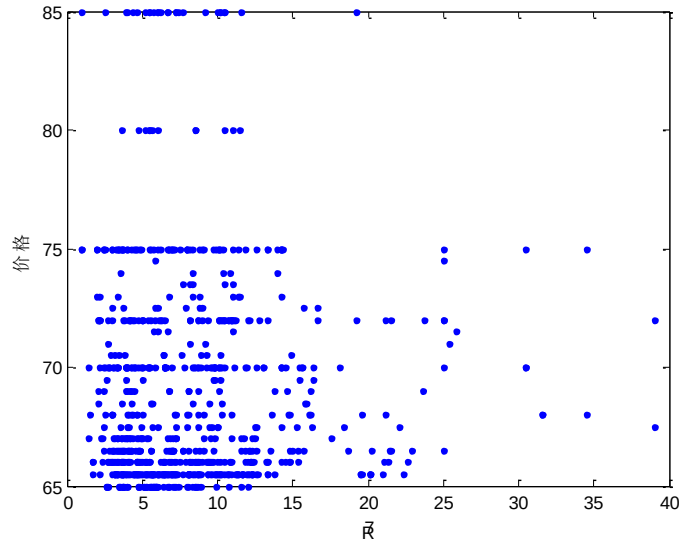


图 4-4 定价与任务集中程度关系图

任务密集程度反映了任务在某一区域内的集中情况。任务的集中程度越高，供用户挑选的任务也就越多，在集中度高，距离近的情况下，任务的定价较低。

任务难易程度 s_i：

由于各项任务的性质不同，完成它的难易程度也不同。当某一任务较难完成时，为了提高它的吸引力，就要给出较高的定价。我们将任务按照难易程度分为三个档次：很难、较难、一般。

4.3 任务定价规律

为了保证任务能够顺利完成和会员的基本收益，对每项任务设定它的基础价格为 P_0，结合上述的三个指标，我们给出第 i 个任务的定价 P_i：

$$P_i = P_0 + 0.5R_i + S_i - Q_i \qquad (4—1)$$

这里，P_0 为基础定价，设为 65，其余指标转换值按如下方式给出：

表 4-1 会员密集程度 m_i 与 R_i 对应表

m_i	1~2	3~4	5~6	7~8	8~9	10~11	12~13	14~15	16~17	18~19
R_i	20	19	18	17	16	15	14	13	12	11
m_i	20~21	22~23	24~25	26~27	28~29	30~31	32~33	34~35	36~37	>37
R_i	10	9	8	7	6	5	4	3	2	1

表 4-2 任务难易程度 s_i 与 S_i 对应表

s_i	很难	较难	一般
S_i	10	5	0

表 4-3 任务集中程度 q_i 与 Q_i 对应表

q_i	0~15	16~30	>30
Q_i	5	2.5	0

说明：考虑到任务发布者与会员双方的利益，我们限定任务最低定价为 65 和最高定价为 85，一旦当 P_i 小于 65 时，规定定价取值 65；当 P_i 大于 85 时，规定定价取值 85。将附件 1 中的实际数据代入公式比较，多数数据满足公式（4-1）；部分与公式有较大出入的定价，是由于任务的难易程度影响造成的。根据这些数据，我们也能计算出各个任务的难易程度。

例：任务号码 A0773，已完成，会员密集程度 $m_i = 6$，任务集中程度 $q_i = 2$，实际定价为 85，已完成，则可带入公式 $85 = P_0 + 0.5R_i + S_i - Q_i = 65 + 0.5 \times 18 + S_i - 0$，解得 $S_i = 11$，则任务难易程度为“很难”。

4.4 未完成任务原因分析

（1）原因一：某些任务邻近区域内，任务密集，而会员人数远少于任务数，人均任务量远远大于其实际完成能力。

例：任务号码 A0436，会员密集程度 $m_i = 3$，任务集中程度 $q_i = 7$，任务难易程度 $s_i =$“一般”，理论定价为 $P_i = P_0 + 0.5R_i + Q_i - S_i = 65 + 0.5 \times 19 + 0 - 2.5 = 72$，实际定价为 75，虽然定价基本合理，但是由于会员人数少，任务多，会员会找性价比更高的任务来完成，

而放弃该项任务。

（2）原因二：任务本身定价较低，使会员完成该任务的收益达不到期望值。某些任务远离会员密集地，若会员要完成需花费较高的费用，且任务定价较低，使得回报低于成本，故出现无人接单的情况。

例：任务号码 A0350，会员密集程度 $m_i=8$，任务集中程度 $q_i=6$，任务难易程度 $s_i=$“一般”，理论定价为 $P_i=P_0+0.5R_i+Q_i-S_i=65+0.5\times17+0-2.5=71$，实际定价为 $65<71$，定价过低，满足不了会员的期望。

（3）原因三：高收入地区会员对收益要求较高，导致很多任务没人接单。例如深圳地区居民普遍收入较高，对收益也有较高的期望值，使得在其他地区能够完成的任务产生的收益值不足以吸引他们去完成。

例：任务号码 A0468，会员密集程度 $m_i=8$，任务集中程度 $q_i=2$，任务难易程度 $s_i=$“一般”，理论定价为 $P_i=P_0+0.5R_i+Q_i-S_i=65+0.5\times17+0-5=67.5$，实际定价为 $80>67.5$，此任务所在地位深圳市中心，居民收入水平较高，即使此任务性价比高，也难以吸引会员。

（4）原因四：高信誉会员的限额较高，预约过多任务，导致很多任务没及时完成。某些任务周围有较多低信誉会员，但此任务被高信誉会员优先预约，而高信誉会员由于多任务在身或此任务距离较远而没有完成。

（5）原因五：任务难度过高，导致任务性价比不高，或者任务所在地的周边环境，如有交通故障难以通行、危险地带等原因，导致任务无法及时完成。

各任务点未完成原因如下图所示：

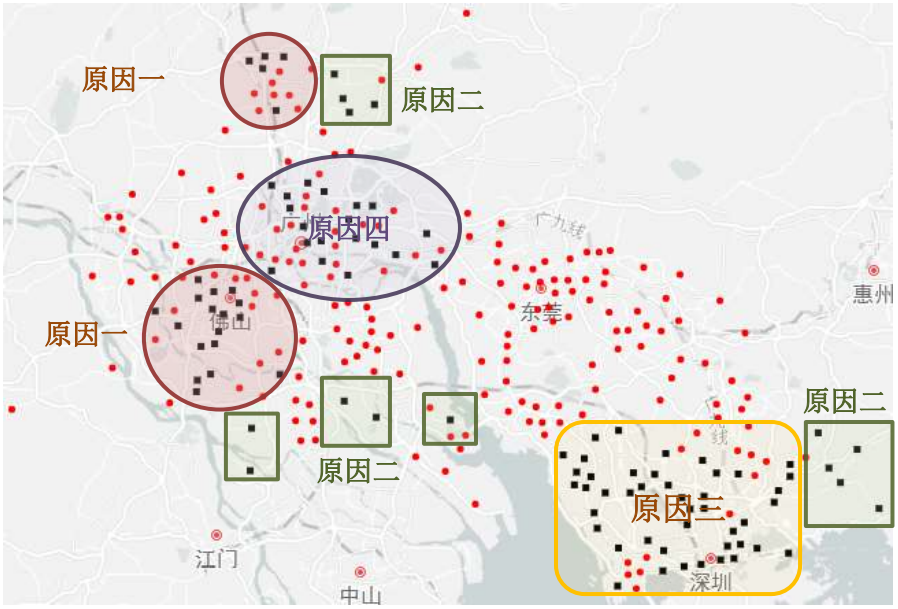


图 4-5 未完成任务点原因分析图

5 问题二的模型建立与求解

5.1 问题二的分析

首先，分析问题一中定价模型存在的不足主要有：未考虑到不同地区会员对收益的期望值，以及信誉度高的会员接单限额和开始预约时间对任务定价的影响。结合这些因素，我们给出新的定价方案。其次，影响任务完成与否的主要因素为会员对收益的期望值，所以我们定义了会员对任务的满意度。结合给出的定价模型，建立以任务完成量最大为目标的优化模型。

5.2 定价模型

5.2.1 模型的建立

由于问题一给出的定价规律没有很好地考虑到不同地区经济发展的不平衡，不同地区会员的收益期望值不同，高信誉会员优先选择任务和限额的特点，导致大量的任务未被完成。所以，针对上述原因，我们建立了新的任务定价模型。

①对于经济发达的地区，会员的收益期望值较高，相同的定价在其他地区可以被完成，所以在定价时，对该地区任务的定价给予一定的提高。

②考虑到高信誉会员优先选择任务和限额的特点，在计算会员密集程度时，把一个高信誉会员按其所能完成的任务限额，把他当做若干个会员处理。

③由于问题一中，相当一部分任务并不是由于价格过低而未被完成的，所以我们不再限制最低价格和最高价格。

基于上述分析，建立定价模型如下：

$$P_i' = P_0 + 0.5R_i + S_i - Q_i + T_i \tag{5-1}$$

其中，m_i' 表示 m_i 个会员在任务限额限定下能完成的任务总数；

图 5-1 会员限额总数 m_i 与 R_i 对应表

m_i'	1~3	4~6	7~9	10~12	13~15	16~18	19~21	22~24	25~27	28~30
R_i	20	19	18	17	16	15	14	13	12	11
m_i'	31~33	34~36	37~39	40~42	43~45	46~48	49~51	52~54	55~57	>57
R_i	10	9	8	7	6	5	4	3	2	1

$$T_i = \begin{cases} 5, \text{任务}i\text{位于经济发达地区} \\ 0, \text{任务}i\text{不位于经济发达地区} \end{cases};$$

P_0，S_i，Q_i 与问题一模型中相同。

5.2.2 模型的求解

将附件 1，附件 2 中的数据和问题一中得到的各任务的难易程度，结合上述公式，利用 MATLAB 编程，得到各个任务的定价如下（部分结果），详细结果见附件二：

表 5-1 任务定价情况表

任务号码	原定价	现定价
A0022	65	65
A0012	65.5	70.5
A0468	80	86
A0018	66	71
A0006	75	77.5
A0450	85	87
A0693	65.5	69.5
A0264	67	72
A0835	85	82
A0830	85	79
A0468	80	90
A0734	75	90

表 5-2 优化前后总定价和完成率情况表

	总定价	任务平均定价
原方案	57641.5	68.87
现方案	58732.6	70.17

5.3 用户满意度

对于会员是否接单，最关键考虑的是任务的性价比，也就是收入支出比。当某一会员对任务的满意度大于某一阈值时，该会员有意愿接受此任务，即任务被完成。这里，我们定义会员 j 对任务 i 的满意度为

$$\kappa_{ij}=\frac{P_i'}{c\cdot d(i,j)},$$

其中，P_i' 表示任务 i 的定价，c 表示单位距离所需的费用，$d(i,j)$ 表示会员 j 和任务 i 之间的距离。

考虑到会员的信誉度会对任务的完成有影响，因此，我们定义罚函数 $c_i=1-\frac{Q_j}{maxQ_j}$，其中，Q_j 表示会员 j 的信誉度，$maxQ_j$ 表示以任务 i 为中心，r 为半径范围内会员信誉的最大值。建立衡量任务 i 是否完成的可能性指标为

$$\kappa_i=max\left\{\frac{P_i'}{c\cdot d(i,j)}\left(1-\frac{Q_j}{maxQ_j}\right)\middle| d(i,j)\le r\right\}. \qquad (5—2)$$

当任务点 i 的指标值 κ_i 大于或等于某一阈值 γ 时，我们认为存在会员愿意接受该任务；否则，该任务未完成。式（5—2）作为衡量指标，不但考虑了会员的实际选择依据，而且能解释当性价比接近时，平台优先安排信誉度较高的会员的原则。

5.4 任务优化分配模型

5.4.1 模型的建立

在给出模型之前，我们对定义一些基本符号：

n_1表示任务数，n_2表示会员数；

$d(i,j)$为会员 j 和任务 i 之间的距离；

κ_i 为任务 i 完成的可能性指标；

γ 为阈值；

V_j 表示会员 j 愿意接受任务点的集合；

e_j 表示会员 j 所能接受任务限额；

$x_{ij}=\begin{cases}1, i\in V_j\\0, i\notin V_j\end{cases}$，当 $i\in V_j$ 时，任务 i 由会员 j 完成；

Q_j 表示会员 j 的信誉值；

t_j 表示会员 j 开始预定任务的时间；

C 表示任务完成所支付总金额限制；

（1）目标函数的确定

在总费用一定的限制情况下，一个合理的定价方案，最关键的是要保证平台所给出的任务要尽量多地完成。因此，我们以任务的完成量最大为目标函数，即：

$$max \quad \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} x_{ij}\ .$$

（2）约束条件的确定

约束条件一：同一个会员的预定任务数量有限额限制，设会员 j 接受任务点的集合可表示为 $A_j=\{i\,|\,x_{ij}=1\}$，即：

$$\sum_{i\in A_j} x_{ij}\le e_j'\ .$$

其中，e_j' 为重新定义的用户限额。考虑到每个会员完成任务的能力有限，我们定义每个会员的限额最大不超过 5，即：

$$e_j'=\begin{cases}e_j, e_j\le 5\\5, e_j>5\end{cases}.$$

约束条件二：会员只接受的任务满足自己期望的任务。考虑到附件 1 中每个区域的用户对任务的期望值不同，我们对上述优化模型中的阈值 γ 按区域重新设定。根据附件一中的经纬度，我们按城市将用户分成佛山、广州、东莞和深圳四块区域，并将期望值设成 γ_1，γ_2，γ_3 和 γ_4.

考虑不同地区用户的不同期望阈值，定义特征函数

$$y_{jz}=\begin{cases}1, j\in T_z\\0, j\notin T_z\end{cases},\quad z=1,2,3,4,$$

其中，T_1，T_2，T_3 和 T_4 分别为佛山、广州、东莞和深圳四块区域的用户集合。

会员 j 愿意接受任务点的集合为

$$V_j=\left\{i\left|\frac{P_i'}{c\cdot d(i,j)}\left(1-\frac{Q_j}{maxQ_j}\right)\geq\gamma_z,d(i,j)\leq r,y_{iz}=1\right.\right\},$$

即：

$$A_j\subset V_j.$$

约束条件三：同一个任务最多只能被一位会员完成，即：

$$\sum_{j=1}^{n_2}x_{ij}\leq 1,\ \ d(i,j)\leq r.$$

约束条件四：开始预约任务时间早的会员优先挑选任务，如果任务 i 是会员 j_1 和 j_2 都满意的，那么开始时间较早的会员优先挑选。即：

$$x_{ij_1}t_{j_1}\geq x_{ij_2}t_{j_2},\ \ \forall i\in V_{j_1}\cap V_{j_2}.$$

约束条件五：平台支付给用户的总费用有一定限制，即：

$$\sum_{i=1}^{n_1}P_i'\leq C.$$

综上所述，建立任务分配优化模型如下：

$$max\quad\sum_{i=1}^{n_1}\sum_{j=1}^{n_2}x_{ij}$$

$$s.t.\begin{cases}P_i'=P_0+0.5R_i'+S_i-Q_i+T_i\\ \sum\limits_{i\in A_j}x_{ij}\leq e_j'\\ \sum\limits_{j=1}^{n_2}x_{ij}\leq 1,d(i,j)\leq r\\ x_{ij_1}t_{j_1}\geq x_{ij_2}t_{j_2},\forall i\in V_{j_1}\cap V_{j_2}\\ A_j\subset V_j\\ \sum\limits_{i=1}^{n_1}P_i'\leq C\\ i\in\{1,2,\cdots,n_1\},j\in\{1,2,\cdots,n_2\}\end{cases}$$

5.4.2 基于最大流的启发式算法[1]

考虑到上述优化模型的约束条件比较复杂，一般的线性规划方法无法求解。为了求解最大任务完成数量，我们考虑到在每一个任务发布后，所有对该任务感兴趣的会员都可以对该任务发起请求。在这样的优化前提之下，我们可以将问题转化为一个网络流的最大流问题去求解。我们设计了一种基于最大流的启发式算法对模型进行求解，具体步骤如下：

Step1: 计算 $(\alpha_1,\alpha_2,\alpha_3)$ 的大致范围，设定初始温度 T_0 、点每次移动一步的长度、迭代深度；

Step2: 向 $(\alpha_1,\alpha_2,\alpha_3)$ 随机投点，投点时删除会使总价格高出额定值的点；

*Step3：*对集合中每个点向 8 个方向移动，并调用算法二计算其在新的位置的最大任务接单数，算法二步骤如下：

*-Step3.1：*建图，为任务和会员添加相对应的点，并添加源点和汇点；

*-Step3.2：*每个会员和汇点之间连一条边，边的流量是这个用户的任务分配限额，每个任务与源点之间连一条边，边的流量是 1；

-Step3.3： 每个会员 *i* 和任务 *j* 之间两两配对，计算会员 *i* 对任务 *j* 的满意度。如果会员 *i* 对任务 *j* 感到满意，那么就在会员 *i* 和任务 *j* 之间连一条边；

*-Step3.4：*调用 Dinic 算法，计算源点到汇点之间的最大流；

如果新的结果更好，则将新的节点添加进正在计算的点集合中，如果没有比原来的结果更好，则以概率 P 接受该结果；

*Step4：*当迭代深度超过一定的深度时，退出算法，输出最优解。

（算法介绍见附录一）

5.4.3 求解结果

对上述算法，利用 MATLAB 编程（具体代码见附录二），得到现任务定价及完成情况表（部分），具体详见附件一。

表 5-3 原方案与现方案任务定价及完成情况表

任务号码	原定价	原完成情况	现定价	现完成情况
A0022	65	1	65	1
A0012	65.5	0	70.5	1
A0468	80	0	86	1
A0018	66	0	71	1
A0006	75	0	77.5	1
A0450	85	0	87	1
A0693	65.5	0	69.5	1
A0264	67	0	72	1
A0835	85	1	82	1
A0830	85	1	79	1
A0468	80	0	90	0
A0734	75	0	90	0

表 5-4 原方案与现方案总定价和完成率情况表

	总定价	任务平均定价	完成率
原方案	57641.5	68.87	62.1%
现方案	58732.6	70.17	86.6%

5.4.4 结果分析

考虑到深圳等地区的任务定价有一定的提高，使得现方案的总费用比原方案大约增加了 1.9%，但现方案的任务完成率比原方案的任务完成率提高了 24.5%，说明现方案比原方案大大提高了系统整体效益。

在$(56000,76000)$范围内随机改变初始设定的价格总和上限D，求得在不同价格上限下任务完成数量的最大值，如下图所示：

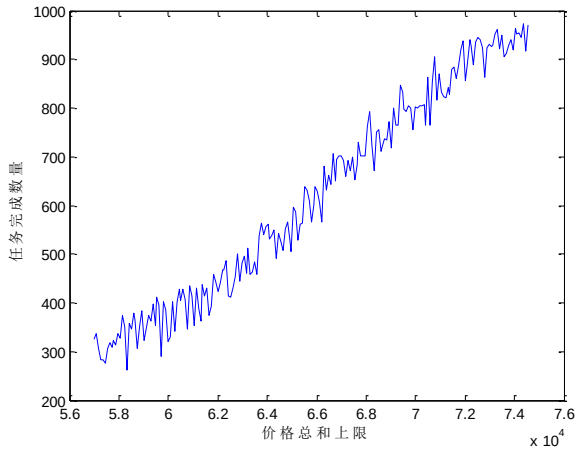


图 5-1 任务最大完成量与价格总和上限关系图

由上图可知，任务最大完成量与价格总和上限总体成正相关，价格上限较低，在56000~60000 范围内，任务最大完成数量变化不大，在 330 件上下波动；价格在60000~71500 范围内，任务最大完成数量随价格总和上限的提高而增加；价格上限较低，在 71500~75000 范围内，任务最大完成数量变化不大，在 950 件上下波动。

6 问题三的模型建立与求解

6.1 问题三的分析

APP 拍照任务定价问题是一个任务发布者（APP 平台）和任务完成者（会员）间双向决策问题。任务发布者希望在任务成本较小的情况下，任务的完成率尽可能的大，完成任务的时间尽可能的短；而作为任务完成者追求任务完成后的收益尽可能的大。分析问题一与问题二中任务的定价规律和任务完成情况，发现会存在下面的一些问题：

（1）有些任务由于定价的不合理，或者是由于任务难度等其他原因不能达到所有会员的满意程度，导致该任务未被完成，影响到任务的完成率。

（2）由于会员选择任务时不合理性，例如一个信誉高的会员选择多个项目，没有考虑到项目的分散程度，在完成任务过程中增加了成本而减少了收益。而对 APP 平台来讲，会导致任务完成的时间延长，甚至无法完成影响到任务的完成率。

为了解决这些问题，我们借助一般商品的打包销售原理，对问题一与问题二中任务的定价规律和模型进行优化，建立打包定价模型。

6.2 打包模型的建立

6.2.1 打包原则

（1）距离相近的任务（集中度较高的任务）应考虑打包发布；

（2）未完成的任务应尽量与自己距离相近的已完成的任务打包发布；

（3）距离相近的价格差比较大的任务应尽量考虑打包发布.

打包原则（1）可以有效提高任务的吸引力，节省会员完成任务的成本，缩短任务

完成的时间。某任务未完成的主要原因是由于该任务的吸引力达不到用户对该任务的满意度，若将直接将若干距离较近且未完成的任务打包，显然不能提高任务的吸引力，有效提高完成率。打包原则（2）、（3）能够提升那些吸引力较低或者吸引力达不到用户对该任务的满意度的任务的吸引力，提升任务的完成率。

6.2.2 二次打包模型

根据上述三条打包原则，建立二次打包模型如下：

第一次打包：根据原则（1），集中度较高的已完成任务进行打包。设T是所有任务集合，T_1是已被完成任务的集合，$d(i,j)$表示两点之间的直线距离，当T_1中两点间的距离$d(i,j)\le r$时，两任务打包发布。

第二次打包：根据原则（1），（2），（3），针对未完成的任务进行分配。对于未完成的任务，我们考虑将它们分配到第一次打包完成后的任务包内，设第一次打包时将任务集T_1划分成$\{S_1,S_2,\cdots,S_l\}$，定义任务点i到集合S_j的距离为

$$d(i,S_j)=\min d(i,k),k\in S_j.$$

定义任务点i与集合S_j中任务的价格差为

$$\xi(i,S_j)=\left|P_i'-\frac{\sum_{k\in S_j}P_k'}{|S_j|}\right|.$$

为简化模型，我们采用平均价格定义任务集合$S_j'=\{i\}\cup S_j$的满意度

$$\kappa_{S_j'}=\frac{\sum_{k\in S_j'}P_k'}{|S_j|+1}.$$

综合上述三条原则，当任务点i满足如下约束时，我们认为可以将未完成的任务i打包分配到集合S_j，

$$\begin{cases} d(i,S_j)=\min d(i,k)\le d_0,k\in S_j,i\in T/T_1 \\ \kappa_{S_j'}=\dfrac{\sum_{k\in S_j'}P_k'}{|S_j|+1}\ge\kappa_0 \\ S_j'\le N \end{cases}.$$

这里，我们取$N=5$.

若上述两限制都满足时，将任务点i合并到价格差最大的集合，即：

$$\max\xi(i,S_j)=\left|P_i'-\frac{\sum_{k\in S_j}P_k'}{|S_j|}\right|.$$

6.3 基于二次打包的定价与任务分配优化模型

对任务进行打包后，原先的任务点被合并成任务集合（也可以为单点集），我们对任务集合重新定义部分变量，其余均与模型二中相同。

l 表示所有任务点划分成任务集合的个数；

任务集合 S_i' 的定价为 $P_{S_i'}' = \dfrac{\sum_{j \in S_i'} P_j'}{|S_i'|}$；

用户 j 到任务集合 S_i' 的距离为 $d'(S_i', j) = \dfrac{\sum_{k \in S_i'} d(j,k)}{|S_i'|}$；

用户 j 对任务集合 S_i' 的满意度为

$$\kappa_{S_i'j}' = \frac{P_{S_i'}'}{c \cdot d'(S_i', j)},$$

其中，$\kappa_{S_i'j}'$ 为用户 j 对任务集合 i 的满意度。

建立衡量任务 i 是否完成的可能性指标为

$$\kappa_{S_i'}' = max\left\{ \frac{P_{S_i'}'}{c \cdot d'(S_i', j)} \left(1 - \frac{Q_j}{maxQ_j} \right) \middle| d(S_i', j) \le r' \right\}.$$

增加约束条件：

会员 j 接受了 S_i' 中的一项任务，则集合 S_i' 应包含在会员 j 完成的任务集中，即：

$$S_i' \subset A_j .$$

综上所述，建立优化模型如下：

$$max \quad \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} x_{ij}$$

$$s.t. \begin{cases} P_i' = P_0 + 0.5R_i' + S_i - Q_i + T_i \\ \sum_{i \in V_j} x_{ij} \le 5 \\ \sum_{j=1}^{n_2} x_{ij} \le 1, d(i,j) \le r \\ x_{ij_1} t_{j_1} \ge x_{ij_2} t_{j_2}, \forall i \in V_{j_1} \cap V_{j_2} \\ A_j \subset V_j \\ S_i' \subset A_j \\ \sum_{i=1}^{n_2} P_i' \le C \\ i \in \{1,2,\cdots,n_1\}, j \in \{1,2,\cdots,n_2\} \end{cases} .$$

6.4 模型的求解

采用类似于问题二中启发式算法，得到部分任务定价及完成情况如下表（详细结果见附件一）：

表 6-1 部分任务定价及完成情况表

任务号码	原方案定价	原完成情况	优化后定价	现完成情况
A0026	66	1	66	1
A0012	65.5	0	69	1
A0468	80	0	83	1
A0118	69	0	71	1
A0216	65.5	1	65.5	1
A0452	68	1	67	1
A0603	65.5	0	69.5	1
A0361	67	0	72	1
A0745	83	1	82	1
A0711	86	1	84	1
A0465	80	0	88	0
A0344	75	0	86	0

表 6-2 打包前后总定价和完成率情况表

	总定价	任务平均定价	完成率
原方案	57641.5	68.87	62.1%
问题二方案	58732.6	70.17	86.6%
打包方案	57698.3	68.93	93.8%

相比于问题二的方案，打包方案在总定价降低了 1.79%，而它的任务完成率达到了 93.8%，比问题二方案的任务完成率提高了 7.2%，比原方案提高了 31.7%。说明打包方案具有更高的系统整体效益。

7 问题四的模型建立与求解

7.1 问题分析

对于给定的任务地理位置和会员的分布情况，由于此时任务的难易程度未知，在问题三的定价模型中，舍弃了任务难易度因素，只利用会员密集程度、给定区域内任务集中程度、不同地区会员的期望值不同等因素不同建立了任务定价模型和任务分配模型。

7.2 模型的建立

不考虑任务难易度因素的定价模型如下：

$$P_i' = P_0 + 0.5R_i' - Q_i + T_i$$

任务分配优化模型如下：

$$max \quad \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} x_{ij}$$

$$s.t.\begin{cases} P_i' = P_0 + 0.5R_i' - Q_i + T_i \\ \sum_{i\in V_j} x_{ij} \le 5 \\ \sum_{j=1}^{n_2} x_{ij} \le 1, d(i,j) \le r \\ x_{ij_1} t_{j_1} \ge x_{ij_2} t_{j_2}, \forall i \in V_{j_1} \cap V_{j_2} \\ A_j \subset V_j \\ S_i' \subset A_j \\ \sum_{i=1}^{n_2} P_i' \le C \\ i \in \{1,2,\cdots,n_1\}, j \in \{1,2,\cdots,n_2\} \end{cases}$$

7.2 模型的求解

采用类似于问题二中启发式算法，得到部分新项目任务定价及完成情况如下表（详细结果见附件二）：

表 7-1 新项目任务定价及完成情况表

任务号码	定价	完成情况	任务号码	定价	完成情况
C0002	66.5	1	C0486	70.5	0
C0014	71.5	0	C0510	72	1
C0019	72.5	1	C0615	77.5	1
C0044	72	1	C0639	66.5	1
C0051	74.5	1	C1230	75.5	0
C0065	72.5	0	C1548	68	0
C0265	77	1	C1689	71.5	0
C0276	70.5	1	C2049	79.5	1

表 7-2 新项目总定价和完成情况统计表

	总定价	任务平均定价	完成任务数	完成率
新项目	149280.5	72.25	1468	71.1%

由于没有考虑任务的难易程度，所以在部分任务定价时会产生一定的偏差，进而影响到任务的完成率，所以该项目的任务完成率偏低，只有 71.1%。为了克服这方面的缺陷，在实际应用过程中，我们可以通过多方面的渠道，提前获得任务的难易程度，建立更加合理准确的打分模型。

8 模型的评价

8.1 模型的优点

（1）定性定量分析了各个参数对任务定价的影响，并设置了扰动参数，合理地解释附件一中所给的数据；

（2）考虑了会员集中度、任务集中度、任务难易程度等因素和不同城市会员满意度的影响，分别建立了任务定价模型和任务分配优化模型；

（3）建立了二次打包模型，给出的打包原则较好地解决了问题一、问题二中任务未完成的情况。

（4）本文采用的启发式算法计算精度高、运算时间短。

8.2 模型的缺点

问题四中没有考虑任务的难易程度，在部分任务定价时会产生一定的偏差，进而影响到任务的完成率。

参考文献

[1] 司守奎, 孙兆亮, 孙玺菁. 数学建模算法与应用[M]. 国防工业出版社, 2015.

[2] 张佳彤. 打车软件参与下出租车动态定价策略研究[J]. 唐山学院学报, 2016, 29(6):78-84.

附录

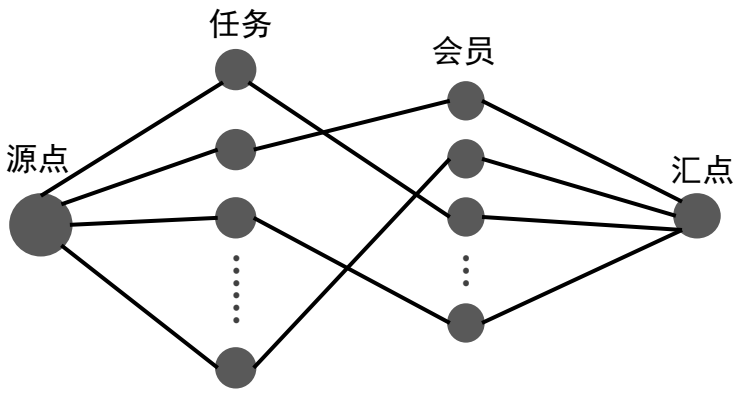
附录一：基于最大流的启发式算法介绍

考虑该优化模型是在多约束情况下求解某个最优目标，因而为得到最优解且搜索速度快，我们采用模拟退化算法进行求解。为确定定价，我们需要求解修正系数 $\delta_i = \alpha_1\rho_i^1 + \alpha_2\rho_i^2 + \alpha_3\overline{Q_i}$ 中的 α_1，α_2，α_3。将这三个值视为三维空间中的一个点 $(\alpha_1,\alpha_2,\alpha_3)$，在 $(\alpha_1,\alpha_2,\alpha_3)$ 的可行范围内均匀投点作为该算法的起始点。在每次迭代中，每个点都可以往 8 个方向进行一个步数的移动，每次移动后计算在该情况下的最大任务完成数量。如果得到的结果比原先的结果好，我们就接受这个结果，如果没有比原先更好，我们就以一定概率接受这个结果。这里我们设定当新的值不如旧的值好时，概率函数为

$$P(x', x_0) = -\frac{(f(x') - f(x_0))}{T_0}$$

为了求解最大任务完成数量，我们设计最大交易数求解算法。我们考虑在每一个任务发布的过程后会有一段公示期，所有对该任务感兴趣的人都可以对该任务发起请求，公示期后，发布任务的公司会根据自己的业务需求选择将任务分配给一些用户。在这样的优化前提之下，我们可以将问题转化为一个网络流的最大流问题去求解。

将每个会员和任务都视为图中的一个点，如果会员 *i* 对任务 *j* 感到满意，则发起请求。于是我们在会员 *i* 和任务 *j* 之间连一条边，同时添加一个超级源点和一个超级汇点，如下图：



最大交易数求解算法示意图

附录二：基于最大流的启发式算法代码

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
#include<iostream>
using namespace std;

//****************************************
// 求解网络流最大流的 Dinic 算法
```

```
const int INF=0x3f3f3f3f;
const int MAXN=100 + 100000;//点数的最大值
const int MAXM=100 + 200000;//边数的最大值

struct Node{
	int from,to,next;
	int cap;
}edge[MAXM];
int tol;

int dep[MAXN];//dep 为点的层次
int head[MAXN];

int n;
void init(){
	tol=0;
	memset(head,-1,sizeof(head));
}
void addedge(int u,int v,int w) {    //第一条变下标必须为偶数
	edge[tol].from=u;
	edge[tol].to=v;
	edge[tol].cap=w;
	edge[tol].next=head[u];
	head[u]=tol++;
	edge[tol].from=v;
	edge[tol].to=u;
	edge[tol].cap=w;
	edge[tol].next=head[v];
	head[v]=tol++;
}

int BFS(int start,int end){
	int que[MAXN];
	int front,rear;
	front=rear=0;
	memset(dep,-1,sizeof(dep));
	que[rear++]=start;
	dep[start]=0;
	while(front!=rear){
		int u=que[front++];
		if(front==MAXN)front=0;
		for(int i=head[u];i!=-1;i=edge[i].next){
			int v=edge[i].to;
			if(edge[i].cap>0&&dep[v]==-1){
```

```
                dep[v]=dep[u]+1;
                que[rear++]=v;
                if(rear>=MAXN)rear=0;
                if(v==end)return 1;
            }
        }
    }
    return 0;
}
int dinic(int start,int end){
    int res=0;
    int top;
    int stack[MAXN];//stack 为栈，存储当前增广路
    int cur[MAXN];//存储当前点的后继
    while(BFS(start,end)){
        memcpy(cur,head,sizeof(head));
        int u=start;
        top=0;
        while(1){
            if(u==end){
                int min=INF;
                int loc;
                for(int i=0;i<top;i++)
                    if(min>edge[stack[i]].cap){
                        min=edge[stack[i]].cap;
                        loc=i;
                    }
                for(int i=0;i<top;i++){
                    edge[stack[i]].cap-=min;
                    edge[stack[i]^1].cap+=min;
                }
                res+=min;
                top=loc;
                u=edge[stack[top]].from;
            }
            for(int i=cur[u];i!=-1;cur[u]=i=edge[i].next)
                if(edge[i].cap!=0&&dep[u]+1==dep[edge[i].to])
                    break;
            if(cur[u]!=-1){
                stack[top++]=cur[u];
                u=edge[cur[u]].to;
            }
            else{
                if(top==0)break;
```

```
                    dep[u]=-1;
                    u=edge[stack[--top]].from;
                }
            }
        }
        return res;
}
//*****************************************

struct User{
    User(){}
    User(double _tasklimts, double _credit):tasklimits(_tasklimts), credit(_credit){}
    int id;
    double tasklimits;
    double credit;
};
vector<User> users;

struct Task{
    double p[3];
    vector<int> user_vct;
};
vector<Task> tasks;

typedef pair<double, double> Range;
struct Ans{
    Ans(){}
    Ans(double _a1, double _a2, double _a3){
        a[0] = _a1;
        a[1] = _a2;
        a[2] = _a3;
    }
    double a[3];
};
Range ansrange;

int n, m;

double getPrice(const Ans& ans, const Task& task){
    double ret = 70;
    for (int i = 0; i < 3; i++){
        ret += ans[i].a[i]*task[i].p[i];
    }
    return ret;
```

```
}

double myrand(int range){
    return double(rand()%range)/range;
}

double Countsatisfaction(){ //计算用户满意度的函数，因为两个模型在这里有不同的表现，所以这里留空

}

double ThresholdValue = 300; //  用户接受一个任务的阈值，这里设置为 300

double step = 1; //任务变化的过程中每次向四周行走的距离

int getPerfromance(Task task){
    int src = 0, dest = m+n+1;
    init(); //初始化建图
    for (int i = 0; i < task.size(); i++)
        addedge(src , i, 1);
        for (int j = 0; j < task[i].user_vct.size(); i++){
            if (Countsatisfaction() > ThresholdValue){
                addedge(i, n+task[i].user[j], 1);
            }
        }
    for (int j = 0; j < m; j++){
        addedge(task.size(), dest, user[j].tasklimit);
    }
    return dinic(src, dest);
}

int dx[] = {-1, 1, 0, 0, 0, 0};
int dy[] = {0, 0, -1, 1, 0, 0};
int dz[] = {0, 0, 0, 0, -1, 1};
void SA(double ){
    double T = 1;
    vector<Ans> host, guest;

    for (int i = 0; i < 1000; i++){//初始化解
        Ans tmp;
        for (int j = 0; j < 3; j++){
            tmp[j] = (ansrange[j].second - ans[j].first)*rand(1000) + ansrange[j].first:
        }
        host.push_back(tmp);;
```

```
    }
    double e = .00000001;
    double at = 0.999;
    double L = 20000;
    for (int i = 0; i < L; i++){
        guest.clear();
        for (int k = 0; k < host.size(); k++)
            const Task& cur = host[k];
            int cur_v = getPerfromance(cur);
            for (int j = 0; j < 6; j++){
                Task& newtask;
                newtask.p[0] += dx[j]*step;
                newtask.p[1] += dy[j]*step;
                newtask.p[2] += dz[j]*step;
                int new_performance = getPerfromance(newtask);
                if (new_performance > cur_v)
                    guest.push_back(newtask);
                else if (exp(cur_v - new_performance)/T > myrand(1000)){
                    host.push_back(newtask);
                }
            }
    }
}
typedef pair<int, int> pir;
int main(){
    srand(time(0));
     std::ios::sync_with_stdio(false);cin.tie(0);
    cin >> n >> m; //读入任务的数据，以及总人数
    for (int i = 0;    i < n; i++){
        Task tmp;
        for (int j = 0; j < 3; j++){
            cin >>tmp.p[j];
        }
        tasks.push_back(tmp);
        int m; cin >> m; // 读取在该任务一定范围内用户的数据，该数据由另一个程序处理得到。
        tmp.user_vct.clear();
        for (int i = 0; i < m; i++){
            User usertmp; cin >> usertmp.id >> usertmp.tasks >> usertmp.credits;
            tmp.user_vct.push_back(usertmp);
        }
    }
    for (int i = 0; i < 3; i++){
        cin >> ansrange[i].first >> ansrange[i].second;
```

```
        }
        return 0;
}
```