队伍编号	MC2311112
题号	（A）

量子计算机在信用评分卡组合优化中的应用

摘 要

通过对银行信用卡业务的最佳风险控制策略分析，本文建立了 0-1 整数规划模型并将其转化为相应的二次无约束二进制优化(QUBO)形式，在此基础上采用 PyQUBO 工具、Gurobi 求解器以及模拟退火算法对银行信用评分卡的组合策略进行了研究。

对于问题一，要求在 100 个信用评分卡中挑选 1 张并决策其对应阈值，使最终收入最大化。该问题以一张信用评分卡及其对应阈值的贷款利息收入和坏账损失的差值最大值为目标函数，以一张信用卡及其对应策略的唯一性为约束条件，建立 0-1 整数线性规划模型。通过分析，模型的决策变量为 0-1 变量，约束为等式约束，将约束转化为对应的哈密顿算符，引入惩罚标量，得出对应的惩罚函数，将其加入到原目标函数中，转化为 QUBO 形式。由于模型的计算复杂度不高且为线性，因此采用 PyQUBO 工具、Gurobi 求解器以及模拟退火算法三种方式进行求解，得出其策略为信用评分卡 49，对应阈值为 1，最大收入为 61172。

问题二在问题一的基础上，增加了信用卡的阈值组合策略，由于总通过率为三张信用卡各自通过率相乘，因此其模型不再线性。该问题以信用卡 1,2,3 及其对应阈值的贷款利息收入和坏账损失的差值最大值为目标函数，以信用卡 1,2,3 各自选择一种阈值为约束条件，最终建立为含有高阶多项式的 0-1 整数规划模型。通过分析，目标函数含有三阶多项式，引入新的 0-1 变量将其降次为二次优化问题，其等式约束按照问题一的方法转化，得到 QUBO 模型。该模型依旧为一个小规模优化问题，计算复杂度不高，同样采用 PyQUBO 工具、Gurobi 求解器以及模拟退火算法三种方式进行求解。得出其信用卡 1,2,3 的阈值策略为 8,1,2，最大收入为 27914.82。

问题三在问题二的基础上，对于信用卡的选取为随机组合，大幅度的增加了决策变量个数。所构建的模型为一个大规模的含有高阶多项式的 0-1 整数规划模型，约束依旧为等式约束，其 QUBO 形式的转化与问题二相同。同样采用 PyQUBO 工具、Gurobi 求解器以及模拟退火算法三种方式进行求解。然而 Gurobi 无法有效解决大规模问题，模拟退火效果不理想，相比之下调用 PyQUBO 工具能够得出其策略组合为信用卡 8 对应阈值 2，信用卡 33 对应阈值 6，信用卡 49 对应阈值 3，总收入为 43880.9712。

针对小规模优化问题，Gurobi 具有高效性和精确性，因此其求解效率最高。PyQUBO 工具适用于 QUBO 模型，所有其求解效率次之。相对而言，模拟退火算法采取逼近最优解的策略，与初始解、逼近方向以及迭代步长有很大关系，所以其收敛效率最差。针对大规模问题，其计算复杂度较高，由于内存限制，Gurobi 求解器一般无法求解，PyQUBO 工具求解效率最好，模拟退火算法求解效率较差。本文的特色在于在算法上增加了 Gurobi 求解器和模拟退火算法与 PyQUBO 工具进行比较，可看出针对大规模问题，将其转换为 QUBO 形式并用 PyQUBO 工具进行求解可以大幅提高计算的效率与准确度。

关键词：0-1 整数规划；QUBO 模型；PyQUBO 工具；Gurobi 求解器；模拟退火算法

目录

一、问题重述与背景介绍

1.1 问题重述

自二十世纪八十年代以来，信用卡作为一种金融产品在我国得以引进并快速发展，信用卡的出现预示着提前消费和信贷消费的时代已经到来[1]。其中，信用等级评定是开展信贷业务的基础，其含义为银行对信贷客户授信之前需要依据各种审核规则评定其信用等级和信贷额度，只有达到评定等级的客户才具有银行的信用或贷款资格。实际上，规则审核过程是通过一重或多重组合规则对目标客户进行打分，为方便起见，可将上述规则称为信用评分卡。信用评分卡可设置不同的阈值，伴随每个阈值可对应不同的通过率和坏账率。在现实场景下，信用评分卡有且仅有一个阈值生效，所以一重信用评分卡策略下确定该阈值则确定了银行的通过率和坏账率即风险控制策略；而多重信用评分卡策略要确定多个阈值，由此会拥有多种通过率和坏帐率的排列组合，需要选择最优的阈值组合实现最佳的风险控制策略。

一般来说，通过率和坏账率具有一定的正相关性，通过率越高，坏帐率也越高，相反的，通过率越低，坏账率也越低。银行的最终收入等于贷款利息收入减去坏账损失，具体来说，通过率越高，通过贷款资格审核的客户数量越多，银行贷款利息收入就越多；但高通过率一般对应高坏帐率，意味着坏账风险越大，坏帐损失也越大。因此，基于题目给出的信用卡评分数据，借助量子计算机和 QUBO 模型量化阈值对通过率和坏账率的影响程度，以最终收入最大化为导向选择最合理的信用评分卡组合是重中之重。

本题提供了包含 100 张信用评分卡的相关数据，其中每张信用评分卡拥有 10 种阈值，每种阈值下均提供对应的通过率和坏账率。根据已有的数据和相关文献资料对下列问题进行探讨：

1、请根据文中叙述和附件 1，选择一重信用评分卡策略，以阈值下的通过率和坏账率为自变量，以银行最终收入最大化为目标函数进行建模，并将建成的模型转为 QUBO 形式并进行求解。

2、依据问题 3 的三重信用评分卡策略，即银行已经确定了三种信用评分卡规则，探索以三种规则下的不同阈值为自变量，以银行最终收入最大化为目标函数的数学模型，并将建成的模型转为 QUBO 形式并进行求解。

3、从附件 1 中 100 个信用评分卡任意选取 3 种信用评分卡，探索以信用评分卡为自变量，银行最终收入最大化为目标函数的数学模型，并将建成的模型转为 QUBO 形式并进行求解。

1.2 背景介绍

1.2.1 二次无约束二进制优化(QUBO)

(1) 二进制优化

组合优化是研究在一定约束条件下，从众多可行解中寻求最优解的一类优化问题。究其本质，它主要研究离散变量(即 0-1 变量)的取值组合，使得目标函数达到最优[2]。现实生活中，组合优化问题随处可见，如路线规划、资源分配、调度安排。

组合优化问题具有 NP 完全性，通常很难求出精确解，因此研究组合优化问题的方法主要包括近似算法、启发式算法和元启发式算法等[3]。近似算法通常可以在多项式时间内求出接近最优解的解，但不能保证解的质量；启发式算法则是通过一些有效的搜索策略来寻找最优解；元启发式算法则是将多个启发式算法结合起来，通过相互协作来提高解的质量和搜索效率。

二进制优化是组合优化问题的一个子类，其变量取值被限制在一个有限的值集内。通俗的，二值优化问题可以表述为：在某些等式和不等式约束下，决策的变量$x_i \in \{0，1\}$

取值使得目标函数$H(\vec{x})$最优。约束限制变量的取值范围，或者划定了可行解的边界，确保决策可行的解决方案。二进制优化的标准形式如下所示：

$$\min\{H(\vec{x}) \mid \vec{x} \in \{0,1\}\}$$

$$f(\vec{x}) = 0, g(\vec{x}) \geq 0$$

二进制优化可以提高程序的效率和性能，减少内存占用，并在多个领域中发挥重要作用。通过分析，不难发现二进制优化有着其独特的应用场景，例如社交网络分析、金融投资组合优化、交通管理和运输调度等。

(2) 多项式无约束二进制优化(PUBO)

多项式无约束二进制优化属二进制优化的一种，其目标函数是由二元多项式组成的。这些二元多项式可以表示为布尔变量之间的逻辑运算[4]，如与、或、非等。PUBO问题可以通过将其转换为整数线性规划(ILP)问题来求解。具体地，可以使用整数变量表示布尔变量，并将布尔运算表示为线性不等式。然后，将目标函数转化为整数线性函数，最后使用整数线性规划算法求解。另外，还可以使用基于启发式算法的方法来求解PUBO 问题，例如模拟退火、遗传算法、蚁群算法等。这些算法可以在较短的时间内获得较好的近似解，但不能保证获得全局最优解。

(3) 二次无约束二进制优化(QUBO)

二次无约束二进制优化(QUBO)是 PUBO 问题的特例，其目标函数是由二元变量的平方项和交叉项组成的二次多项式[5]。这些二元变量通常被编码为 0 或 1，因此可以被视为布尔变量。QUBO 问题的标准形式如下所示[6]：

$$\min\left\{H(\vec{x}) = \sum_{i}^{N} h_i x_i + \frac{1}{2}\sum_{i}^{N}\sum_{j}^{N} J_{ij} x_i x_j \mid \vec{x} \in \{0,1\}\right\}$$

一般情况下，针对实际问题，所构建的模型大多为约束优化模型。此时，便可以将约束优化模型通过松弛或者增加惩罚项变为无约束优化问题。通过转换就可以采取无约束优化问题的一般求解算法(如共轭梯度法、变尺度法等)加以解决，也可以运用启发式算法求解。

QUBO 问题可以被转化为一个特殊的线性二次优化(Quadratic Programming, QP)问题[7]，其中线性约束被表示为线性等式或不等式。通常，这个问题可以通过 Quadratic Unconstrained Binary Optimization Solver (QUBO Solver) 这样的特殊软件来求解[8]。QUBO Solver 通常使用量子退火算法来解决这个问题，它利用了量子比特的优势，可以加速搜索最优解的过程。

QUBO 模型是一种用于解决组合优化问题的数学模型，其可以将多种实际问题转化为 QUBO 模型进行求解，在图着色、旅行商问题、车辆路径优化等领域都得到了广泛的应用。QUBO 模型在金融风控领域也具有重要的应用，信用评分卡是一种常用的风险控制工具，用于评估信用风险，不同的信用评分卡之间存在着一定的差异，如何选择最佳的组合方案以实现最优的风险控制一直是金融业面临的难题。我们通过将不同的信用评分卡组合问题转化为 QUBO 模型，并使用 QUBO 模型的优化算法进行求解，得到了最佳的组合方案以实现最优的风险控制。经过对比证明了与传统方法相比，基于QUBO 模型的算法具有更高的准确性和效率，对银行信用评估提供了有益的参考和借鉴意义。

1.2.2 模拟退火算法(Simulated Annealing，SA)

模拟退火算法(Simulated Annealing，SA)的思想是在给定初温下利用热波动搜索问题最优解[9]，但具有降温速度慢、耗时久、计算量大等缺点。而基于量子波动改进的量子退火算法(Quantum Annealing，QA)能够克服传统模拟退火算法的缺点。量子波动的优势在于它使得量子具有穿透比自身能量更高的是势垒的能力，这一性能称之为量子隧穿效

应。改进后的算法利用量子隧穿效应使算法摆脱局部最优，从而以更大概率实现全局最优，适用于求解各种类型的优化问题[10]。图 1.1 是用模拟退火算法求解 QUBO 模型的一般思路和过程：

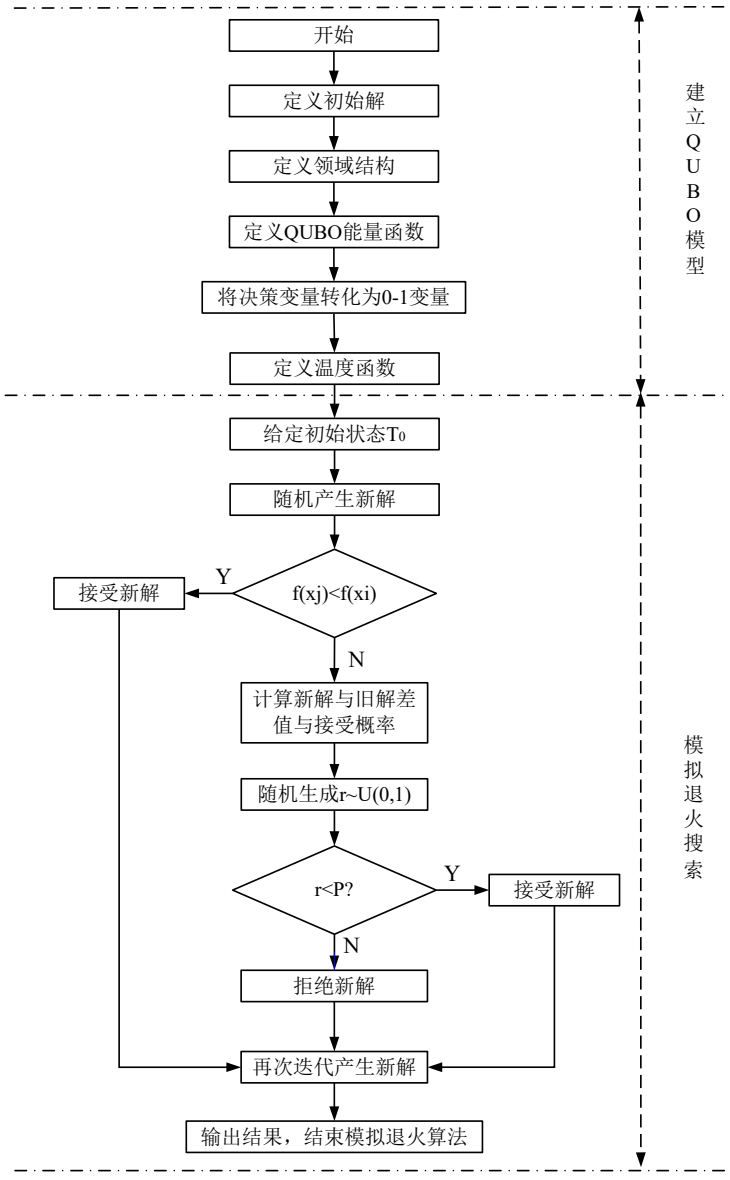

图 1-1 模拟退火算法进行信用卡评分组合优化流程图

(1)定义初始解：随机生成一个二进制数列作为初始解，或者通过一些启发式方法生成一个比较好的初始解。

(2)定义邻域结构：将当前解的某个位置取反，生成一个新的解。

(3)定义能量函数：将 QUBO 模型转化为能量函数的形式，即将目标函数转化为一个关于解的函数，这个函数值越小，解越好。通常可以将 QUBO 模型表示为二次型的形式。

(4)定义温度函数：模拟退火算法通过温度函数来控制搜索过程，从而避免陷入局部最优解。温度函数可以采用不同的方式来定义，但一般情况下都是一个下降函数，随着搜索的进行逐渐降低。

(5)模拟退火搜索：模拟退火算法通过在当前解的邻域结构中进行搜索，以一定的概率接受劣解，从而达到避免陷入局部最优解的目的。

(6)输出结果：输出搜索过程中能量最小的解作为最优解

1.2.3 PyQUBO 工具

量子退火算法(Quantum annealing)的核心思想是将问题的目标函数编码为量子比特的能量函数[11]，然后利用量子退火的过程在量子比特状态空间中寻找目标函数的全局最小值。在量子退火算法中，问题的目标函数被表示为一个哈密顿量，它的本征态对应于问题的解。算法开始时，量子比特被初始化为一个简单的状态，如完全偏置或随机态。然后，系统通过逐渐减小哈密顿量的参数来进行“退火”，从而使系统逐渐从初态演化到目标函数的本征态，即问题的解[12]。量子退火算法是一种 NP 难问题的近似算法，它在求解某些 NP 难问题方面具有很强的潜力，例如组合优化问题等[13]。

PyQUBO 工具是用于编写 QUBO 问题的 Python 库，以下是 PyQUBO 的一些常用功能：

(1) 生成 QUBO 问题：PyQUBO 可以帮助用户将一个实际问题转化为一个 QUBO 问题，并生成 QUBO 问题中的线性和二次项。使用 PyQUBO 可以大大简化生成 QUBO 问题的过程。

(2) 解决 QUBO 问题：PyQUBO 提供了一些优化算法来求解 QUBO 问题，例如 simulated annealing, tabu search, 和量子近似优化算法(QAOA)等。用户可以根据需要选择不同的算法来解决问题。

(3) 显示 QUBO 问题：PyQUBO 可以帮助用户将 QUBO 问题可视化，以便于理解和调试问题。

(4) 与量子计算机的接口：PyQUBO 还提供了与 D-Wave 量子计算机的接口，可以将 QUBO 问题上传到 D-Wave 量子计算机上进行求解。

总的来说，PyQUBO 是一个方便易用的 Python 库，可以帮助用户生成和解决 QUBO 问题，并且可以与量子计算机进行接口。

QUBO 模型算法步骤

Step1：根据待优化问题，构造量子系统的评价函数$H_q = H_{pot} + H_{kin}$，即量子哈密顿函数。其中，$H_{pot}(t)$为势能，即模拟退火算法中的评价函数，$H_{kin}(t)$为动能；

Step2：初始化各个参数，T_0为量子退火的初始温度，Γ为横向场强，变化的横向场强引起不同量子状态之间的量子跃迁，最大迭代次数为$MaxSteps$，初始化状态为x，对应的状态能量为$H_{pot}(x)$；

Step3：随机微扰产生新状态x'，对应的状态能量为$H_{pot}(x')$；

Step4：计算能量差$\Delta H_{pot} = H_{pot}(x') - H_{pot}(x)$以及$\Delta H_q = H_q(x') - H_q(x)$，如果$\Delta H_{pot} < 0$或者$\Delta H_q < 0$则系统接受新解$x = x'$，反之如果$\exp(\Delta H_q / T) < rangdom(0,1)$，则$x = x'$，否则重复执行步骤 3；

Step5：进行退温操作，Γ的变化和模拟退火中的温度T作用类似,横向场强变化形式为$\Gamma = \Gamma - (\Gamma_0 / MaxSteps)$；

Step6：判断是否满足终止条件$\Gamma = 0$，如果满足量子退火算法终止，否则重复步骤 3。

二、模型假设

问题本身的简化：

1、总通过率为所有信用评分卡通过率相乘；

2、总坏帐率为三种信用评分卡对应坏账率的平均值；

3、本次贷款利息收入为：贷款资金*利息收入率*总通过率*（1-总坏账率）；

4、坏帐带来的坏帐损失为：贷款资金*总通过率*总坏账率；

5、银行的最终收入为：贷款利息收入-坏账收入；

模型假设

6、假设银行最终收入只受到通过率和坏账率的影响，不考虑其他因素的影响；

7、为简化问题，本文假设银行的贷款资金为 100 万元，银行的贷款利息收入率为 8%，不考虑评分卡的前序评价流程，只根据附件 1 中阈值进行建模求解；

8、假设对于每一种信用评分卡，最多只能选取一个阈值作为其评判标准；

9、银行决策三种组合策略时，所选取的三种信用卡不能相同。

三、符号说明

建模过程中所用到的变量及其符号说明如表 3-1 所示：

表 3-1 符号说明

参数	含义
l,m,n	表示信用卡种类或者序号
i,j,k	表示选取的阈值
F	银行的贷款资金
η	银行的贷款利息收入率
$t_{l,i}$	信用评分卡l对应于阈值i策略的通过率
$h_{l,i}$	信用评分卡l对应于阈值i策略的坏账率
P	惩罚项数值
$x_{l,i}$	0-1 变量，如果选择信用卡l对应的阈值i，则为 1；否则为 0
t_{ijk}^{last}	信用评分卡 1 选择阈值i，2 选择阈值j，3 选择阈值k策略的总通过率
h_{ijk}^{last}	信用评分卡 1 选择阈值i，2 选择阈值j，3 选择阈值k策略的总坏账率
r_{ijk}	信用评分卡 1 选择阈值i，2 选择阈值j，3 选择阈值k策略的贷款利息收入
l_{ijk}	信用评分卡 1 选择阈值i，2 选择阈值j，3 选择阈值k策略的坏账损失
p_{ijk}	信用评分卡 1 选择阈值i，2 选择阈值j，3 选择阈值k策略的银行最终收入
x_{ijk}	0-1 变量，如果信用 1 选择阈值i，2 选择阈值j，3 选择阈值k，则为 1；否则为 0
x_{ijk}^{lmn}	0-1 变量，如果信用l选择阈值i，m选择阈值j，n选择阈值k，则为 1；否则为 0

四、问题分析

4.1 问题一的分析

问题一要求根据附件 1 中的数据，在 100 张信用评分卡中找出一张及其对应阈值，使最终收入最多，将该模型转为 QUBO 形式并求解。本问题中，只要求在 100 张信用评分卡中选择一张及其对应阈值，根据银行的最终收入的求解公式，需知选择的该张信用评分卡的通过率、坏账率，求出其贷款利息收入、坏账损失得到最终的银行收入。

问题一采用了一种信用评分卡策略，从附件 1 已有数据和问题描述可以知道每一信用评分卡有且只可选择 1 个阈值，那么一种信用评分卡策略的目标对象为选择最佳阈值可使得银行最终收入最大。针对该问题，本文可以使用 0-1 整数线性规划模型。该问题为一个优化问题，目标是银行最终收入的最大值，目标函数为线性函数；约束条件为 100 张信用评分卡中只找出一张及其对应阈值。决策变量为 0-1 变量，为选择的信用评分卡及其对应阈值。

该模型为含有等式约束的线性规划模型。由于目标函数为线性函数，决策变量为 0-1 变量，由 QUBO 的特性$x = x^2$可知，该目标函数可直接转化为 QUBO 形式，其等式约束作为惩罚项加入到目标函数中。

该问题的求解，可以通过建立线性规划模型，使用 Gurobi、Lingo 等求解器；也可以使用一些启发式算法，例如模拟退火算法，设置启发式规则进行求解。对于 QUBO 形式的模型，采用 PyQUBO 工具进行求解。

4.2 问题二的分析

问题二要求在已经选定了数据集中给出的信用评分卡 1、信用评分卡 2、信用评分卡 3 这三种规则下设置其对应的阈值，使最终收入最多，将该模型转为 QUBO 形式并求解。本问题中，已经给定了一组信用卡，需要在给定的信用卡下设置其对应的阈值，根据公式可知，其总的通过率与坏账率不再是一个值，而是一个组合值。

该问题为一个组合优化问题，目标是银行最终收入的最大值，目标函数为高阶多项式函数；约束条件为在信用评分卡 1,2,3 中分别找出一个对应的阈值。决策变量为 0-1 变量，为信用评分卡 1,2,3 对应阈值。

该模型为含有等式约束的高阶多项式优化模型。由于目标函数含有高阶多项式，决策变量为 0-1 变量，而 QUBO 为二次项无约束二值优化模型，因此需要对目标函数进行降次，将高次的问题转化为二次优化问题，其等式约束作为惩罚项加入到目标函数中。

该问题的求解，由于模型不再是线性规划模型，可以将其线性化后使用 Gurobi 求解器；同时为对比求解效果，考虑采用模拟退火算法的启发式算法进行求解。对于 QUBO 形式的求解，依旧采用 PyQUBO 工具进行求解。

4.3 问题三的分析

问题三要求在所给附录中 100 个信用评分卡中任选取 3 种信用评分卡，并设置合理的阈值，使得最终收入最多，将该模型转为 QUBO 形式并求解。问题三与问题二类似，同样为一个组合优化问题，不同的是，问题三中的 3 种信用卡是不确定值，因此增加了决策变量个数。目标依旧是银行最终收入的最大值，目标函数为高阶多项式函数；约束条件为 100 张信用卡中任选三张信用评分卡，且每张信用评分卡对应一个阈值。决策变量为 0-1 变量，为选择的信用卡及其对应阈值。

该模型与问题二都为含有等式约束的高阶多项式优化模型，其 QUBO 的转化形式与问题二类似，其等式约束作为惩罚项加入到目标函数里面。该问题三求解，由于其信用卡与阈值都不确定，因此变量个数与问题一和二相比，大大增加。同样考虑使用 Gurobi 求解器、模拟退火算法以及 PyQUBO 工具进行求解，对比其三种方法的求解效果。

4.4 数据分析

根据附件 1 中包含的 100 张信用评分卡数据，其中每张卡具有 10 种阈值可供选择，并且每种阈值对应有各自的通过率和坏账率，因此一共有 100 项通过率和 100 项坏账率。将 100 张信用评分卡的通过率绘制成变化热力图如图 4.1，颜色越接近红色通过率越高，越接近蓝色通过率越低，过渡颜色为灰色。通过率范围基本上在 0.70-1.00 之间，直观来讲，一半以上的信用评分卡通过率高于 0.85。

同理，将 100 张信用评分卡的坏账率绘制成变化热力图如图 4.2，颜色越接近红色通过率越低，越接近蓝色通过率越高，过渡颜色为灰色。坏账率范围基本上在 0-0.09 之间，直观来讲，一半以上的信用评分卡坏账率低于 0.05。从图中可以看出，通过率、坏帐率与阈值和信用评分卡顺序无相关关系。

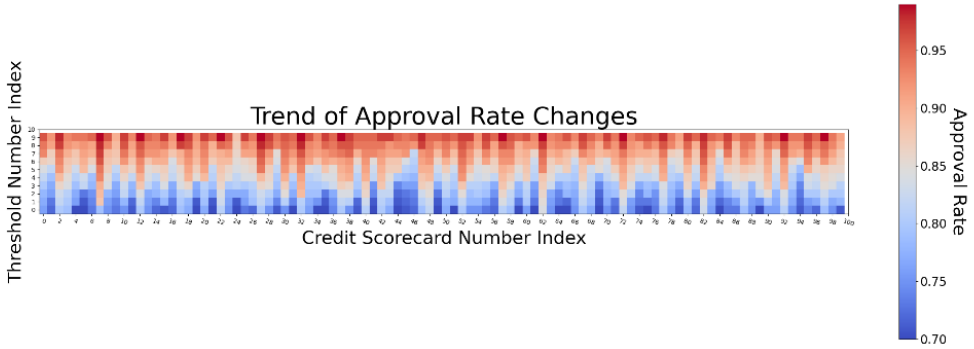

图 4.1 通过率变化热力图

图 4.2 通过率变化热力图

根据附件 1 中包含的 100 张信用评分卡数据，其中每张卡具有 10 种阈值可供选择，并且每种阈值对应有各自的通过率和坏账率，因此一共有 100 项通过率和 100 项坏账率。将 100 张信用评分卡的通过率绘制成变化热力图如图 4.1，颜色越接近红色通过率越高，越接近蓝色通过率越低，过渡颜色为灰色。通过率范围基本上在 0.70-1.00 之间，直观来讲，一半以上的信用评分卡通过率高于 0.85。

同理，将 100 张信用评分卡的坏账率绘制成变化热力图如图 4.2，颜色越接近红色通过率越低，越接近蓝色通过率越高，过渡颜色为灰色。坏账率范围基本上在 0-0.09 之间，直观来讲，一半以上的信用评分卡坏账率低于 0.05。从图中可以看出，通过率、坏帐率与阈值和信用评分卡顺序无相关关系。

图 4.3 银行最终收入变化热力图

根据问题一建立的数学模型得到银行最终收入并绘制变化热力图如图 4.3，最终收入取值范围基本在 10000-60000 之间，其中颜色越接近深红色最终收入越高，颜色越接近淡黄色最终收入越低，可以看出高收入大多出现在大数值阈值下，低收入乃至负收入大多出现在低数值阈值下。

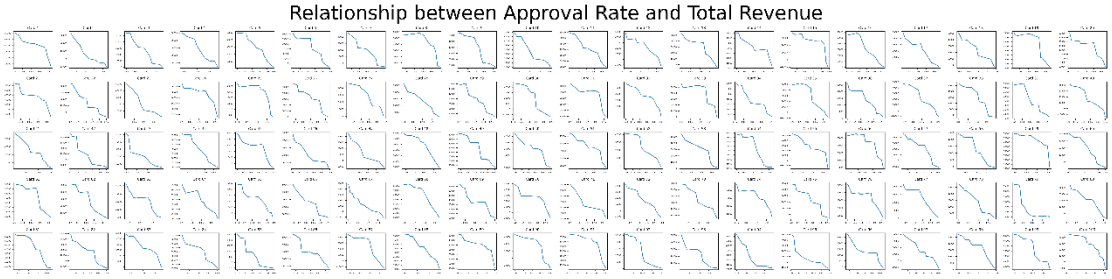

图 4.4 通过率和最终收入的变化关系

根据附件 1 和问题一中 QUBO 模型求解得到的银行最终收入绘制成折线图如图 4.4，其中每个小图纵坐标代表每种信用评分卡的通过率，横坐标代表每种信用评分卡计算后得到的银行最终收入。通过数据可视化后发现通过率与最终收入成反比。

图 4.5 坏账率和最终收入的变化关系

根据附件 1 和问题一中 QUBO 模型求解得到的银行最终收入绘制成折线图如图 4-5，其中每个小图纵坐标代表每种信用评分卡的坏账率，横坐标代表每种信用评分卡计算后得到的银行最终收入。通过数据可视化后发现坏账率与最终收入成反比。

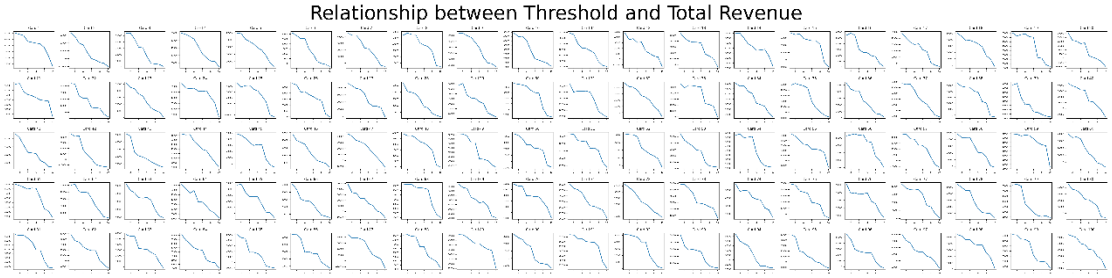

图 4.6 阈值和最终收入的变化关系

根据附件 1 和问题一中 QUBO 模型求解得到的银行最终收入绘制成折线图如图 4.6，其中每个小图纵坐标代表每种信用评分卡的阈值，横坐标代表每种信用评分卡计算后得到的银行最终收入。通过数据可视化后发现阈值与最终收入成反比。

五、模型的建立与求解

5.1 问题一的模型建立与求解

5.1.1 构建数学模型和 QUBO 模型

(1) 构建 0-1 整数规划模型

本问题要求在 100 张信用卡中选取一张，并决策出最优的阈值，使得此信用卡此阈值下的最终收入最大化。通过对问题深度剖析，不难发现问题的约束在只选取一张信用卡且一张信用卡只对应一种阈值。由此，设计相应的决策变量，构建 0-1 整数规划模型。首先，需要选取一张信用卡以及一种阈值，因此建立涉及信用卡序号和阈值两个维度的决策变量$x_{l,i}$。

$$x_{l,i}=\begin{cases}1，选择第l张信用卡中的第i个阈值\\0，否则\end{cases}$$

根据问题所提供的背景知识，计算模型所需要的外部输入，即模型参数。通过题中给出贷款利息收入=贷款资金*利息收入*总通过率*（1-总坏账率），可得银行贷款利息为$F\cdot\eta\cdot x_{l,i}\cdot t_{l,i}\cdot(1-h_{l,i})$,同理可得坏账损失为$F\cdot x_{l,i}\cdot t_{l,i}\cdot h_{l,i}$，以及银行能够获得的最终收入$\sum_{l=1}^{100}\sum_{i=1}^{10}\{[F\cdot\eta\cdot x_{l,i}\cdot t_{l,i}\cdot(1-h_{l,i})]-F\cdot x_{l,i}\cdot t_{l,i}\cdot h_{l,i}\}$。

因此，联合题目信息及上式建立问题一求解模型如下：

目标函数：最终收入最大化，

$$\max Z=\sum_{l=1}^{100}\sum_{i=1}^{10}\{[F\cdot\eta\cdot x_{l,i}\cdot t_{l,i}\cdot(1-h_{l,i})]-F\cdot x_{l,i}\cdot t_{l,i}\cdot h_{l,i}\}\tag{1}$$

约束条件：联系题目可知该模型的约束条件为在 100 张信用评分卡里面选择一张卡，并选择该卡对应的一个阈值，建立约束如下：

$$\sum_{l=1}^{100}\sum_{i=1}^{10}x_{l,i}=1\tag{2}$$

(2) 构建 QUBO 模型

基于上述传统的约束优化模型，我们可以借助 QUBO 的既定规则将其转化为二次无约束优化模型(QUBO 模型)。转化的第一步是借助惩罚的操作手段将约束添加至目标函数，使其成为无约束优化模型的标准形式。这种处理的内涵是：违反约束的情况会使得目标函数极其偏离最优解，意味着要想获取最优目标函数必须保证所有约束条件被满足。

考虑到模型的约束条件为等式约束，于是可直接将约束条件作为惩罚项加入在目标函数中。此外 QUBO 形式下的等式约束$Ax=b$，可以通过$P\left(\sum_{l=1}^{n}\sum_{i=1}^{n}a_{l,i}x_{l,i}-b\right)^2$形式将其作为惩罚项加入到目标函数中。其中$P$是一个惩罚标量，如果$P$足够大，那么就可以保证满足约束条件。

通过分析决策变量$x_{l,i}$，其取值非 0 即 1，不难发现其平方与本身相等。结合问题一建立的整数规划模型和上述 QUBO 的特性转化为$x_{l,i}=(x_{l,i})^2$。

因此，得到问题一求解模型的 QUBO 模型公式如下：

$$\max Z=\sum_{l=1}^{100}\sum_{i=1}^{10}[F\cdot\eta\cdot t_{l,i}\cdot(1-h_{l,i})-F\cdot t_{l,i}\cdot h_{l,i}]\cdot(x_{l,i})^2-P\left(\sum_{l=1}^{100}\sum_{i=1}^{10}x_{l,i}-1\right)^2\tag{3}$$

将上述目标函数进行展开并合并，由此得到二元变量的平方项和交叉项组成的二次

多项式，其中l和m表示两种不同的信用卡。对目标函数进行化简如下：

$$\max Z=\sum_{l=1}^{100}\sum_{i=1}^{10}\{F\cdot t_{l,i}\cdot[\eta\cdot(1-h_{l,i})-h_{l,i}]-P\}(x_{l,i})^2$$

$$-2P\sum_{\substack{l,m=1\\ l\neq m}}^{100}\sum_{i,j=1,}^{10}x_{l,i}x_{m,j}-P \tag{4}$$

由于常数项P不影响我们的最终结果，因此可转换为矩阵形式，矩阵大小为1000*1000的方阵，定义该矩阵Q如下：

$$Q=\begin{bmatrix} F\cdot t_{1,1}[\eta(1-h_{1,1})-h_{1,1}]-P & -P & \cdots & \cdots & -P \\ -P & \ddots & & & \vdots \\ \vdots & & F\cdot t_{l,i}[\eta(1-h_{l,i})-h_{l,i}]-P & & \vdots \\ \vdots & & & \ddots & -P \\ -P & \cdots & \cdots & -P & F\cdot t_{100,10}[\eta(1-h_{1,1})-h_{100,10}]-P \end{bmatrix}$$

5.1.2 问题一求解

(1) Gurobi 求解器求解

Gurobi 求解器是一种商业数学优化求解器，可用于解决线性规划(LP)、混合整数规划(MIP)、二次规划(QP)和非线性规划(NLP)等数学优化问题。Gurobi 求解器是目前业界最先进的求解器之一，具有优秀的求解速度、高效率和易于使用的特点。它可用于多个领域，如金融、能源、医疗保健、交通运输和制造业等，优化决策过程和提高运营效率。

Gurobi 求解器在针对小规模优化问题方面有着得天独厚的优势。运用 Gurobi 求解器求解速度较快，收敛效果良好，能够寻求全局最优解，即精确解。分析本问题所建的初始模型，可以看出其是一个标准的 0-1 线性规划模型，可借助 Gurobi 求解最优解。求解结果如图 5.1 所示。

```
选择的信用评分卡序号为： 49
对应阈值为： 1
最大运营成本： 61172.0
程序用时: 0.028983354568481445 s
```

图 5.1 Gurobi 求解结果

根据 Gurobi 求解结果可知：选取 **49 号**信用卡，并设置其对应**阈值 1**，可得最优目标函数值(最大运营成本)为 61172.0。迭代时间约为 0.02898 秒。

(2) 模拟退火算法求解

模拟退火算法（Simulated Annealing）是一种启发式搜索算法，用于在解空间中寻找全局最优解或近似最优解。它的思想来自于固体物理学中的“退火”过程，即将物质加热并缓慢冷却以达到更稳定的状态。

模拟退火算法用于求解 QUBO 问题具有以下优点：

适用性广：QUBO 问题是一类 NP-hard 问题，通常情况下很难用精确算法解决，但是模拟退火算法可以用于求解多种类型的 QUBO 问题。

全局搜索能力：模拟退火算法是一种启发式搜索算法，具有全局搜索能力，可以找

到全局最优解或接近最优解的解。

可以处理大规模问题：QUBO 问题通常涉及大量变量和约束，而模拟退火算法具有分布式计算的能力，可以处理大规模的 QUBO 问题。

可以处理带噪声的问题：在实际应用中，QUBO 问题常常存在噪声和误差，而模拟退火算法具有一定的容错能力，可以处理带噪声的问题。

可以自适应调节搜索步骤：模拟退火算法中的温度参数和搜索步骤数可以根据具体问题进行自适应调节，从而可以更好地适应不同的问题。

针对于问题一需要在 100 个信用评分卡中找出 1 张及其对应阈值，使最终收入最多，首先将该模型转为 OUBO 形式并利用模拟退火算法进行求解。求解结果如图 5.2 所示。

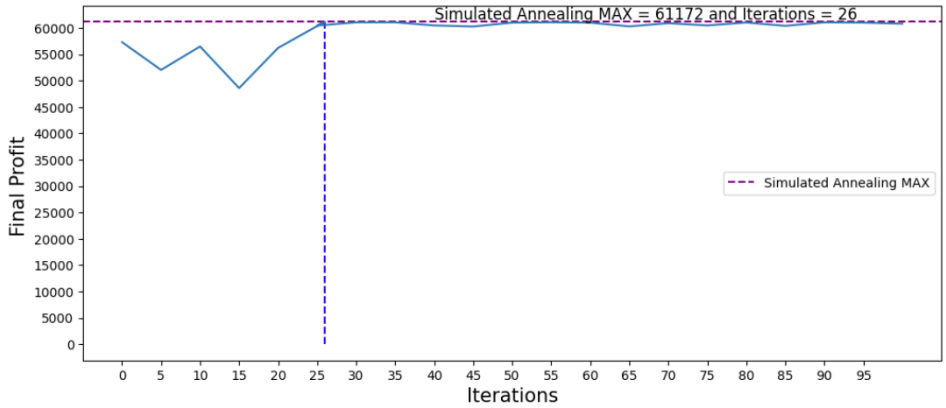

图 5.2 问题一模拟退火迭代过程所有解展示

初始温度设置为 **100 度**

衰减常数设置为 **0.95**

停止准则为迭代 **100 次**

选取 **49** 号信用卡，并设置其对应阈值 **1**，可得最优目标函数值(最大运营成本)为 **61172.0**。求解时间为：**137.451263s**。

产生新解的规则为：在当前解的附近随机产生一个新解，计算新解的目标函数值，重复外循环，在每次迭代过程中，Matlab 模拟退火的函数都只找一个新解，即设置 Markov 链的长度等于 1，这样能节省搜索的时间。

(3) PyQUBO 工具求解

PyQUBO 是 Python 的一个第三方库，可以用来生成和解决 QUBO 问题。PyQUBO 提供了一种简单易用的方法来表示 QUBO 问题，并提供了一些优化算法来求解这些问题，例如 simulated annealing, tabu search, 和量子近似优化算法(QAOA)等。

本问题将模型转化为 QUBO 模型，便可以借助 PyQUBO 进行模型求解，求解结果如图 5.3 所示。

```
[7]': 0, 'x[43][8]': 0, 'x[43][9]': 0, 'x[44][0]': 0,
'x[44][1]': 0, 'x[44][2]': 0, 'x[44][3]': 0, 'x[44]
[4]': 0, 'x[44][5]': 0, 'x[44][6]': 0, 'x[44][7]': 0,
'x[44][8]': 0, 'x[44][9]': 0, 'x[48][5]': 0, 'x[47]
[0]': 0, 'x[48][4]': 0, 'x[47][1]': 0, 'x[85][8]': 0,
'x[48][7]': 0, 'x[47][2]': 0, 'x[85][9]': 0, 'x[48]
[6]': 0, 'x[47][3]': 0, 'x[48][1]': 0, 'x[47][4]': 0,
'x[48][0]': 1, 'x[47][5]': 0, 'x[48][3]': 0, 'x[47]
[6]': 0, 'x[48][2]': 0, 'x[47][7]': 0, 'x[47][8]': 0,
'x[47][9]': 0, 'x[85][7]': 0, 'x[48][8]': 0, 'x[85]
[6]': 0, 'x[48][9]': 0, 'x[50][0]': 0, 'x[50][1]': 0,
'x[50][2]': 0, 'x[50][3]': 0, 'x[50][4]': 0, 'x[50]
[5]': 0, 'x[50][6]': 0, 'x[50][7]': 0, 'x[50][8]': 0,
'x[50][9]': 0, 'x[51][0]': 0, 'x[51][1]': 0, 'x[51]
[2]': 0, 'x[51][3]': 0, 'x[51][4]': 0, 'x[51][5]': 0,
'x[51][6]': 0, 'x[51][7]': 0, 'x[51][8]': 0, 'x[51]
```

图 5.3 PyQUBO 求解结果

由于空间原因，图 5.3 仅仅展示了部分求解结果。在图 5.3 中，只有 0-1 变量“x[48][0]”取值(即图 5.3 蓝色部分)为 1，其余变量取值均为 0。由于 python 索引从 0 开始，所以最终求解结果为：

选择信用卡序号：**49**；

决策出对应阈值：**1**；

最有目标函数值：**61172**；

求解时间为：**67.36457753181458 s**。

5.1.3 三种求解思路效果对比

为进一步分析问题本质以及了解三种求解思路的优化效果。Gurobi 求解器是针对小规模优化问题的有效求解器。在小规模问题求解方面，Guroboi 有着非常很明显的求解效果。所以 Gurobi 求解结果一般被当作标准结果来和其他启发式算法求解效果进行对比。通过和 Gurobi 求解结果对比，可以大致了解启发式算法的收敛效果。收集三种求解过程中的数据，形成效果对比表如表 5-1 所示。

表 5-1 求解效果对比表

决策变量个数：1000	最大收入	迭代次数	迭代时间
Gurobi 求解器	61172.0	-	0.02898s
模拟退火算法	61172.0	12	137.451263s
PyQUBO 工具	61172.0	26	67.36458s

从表 5-1 可以看出，Gurobi 求解速度非常快，求解效果也非常好。其次，运用 PyQUBO 工具求解效果也比较好，收敛速度相对而言比较快，是比较适应 QUBO 模型的求解算法。相对而言，模拟退火算法由于需要设置有效的搜索方向和迭代步长，因此能求解到较为优秀的解，但收敛速度比较缓慢。为进一步展示三种求解思路的效果，绘制如图 5.4 所示的迭代曲线对比图。图中绿色虚线部分表示 gurobi 求解结果，蓝线表示 PyQUBO 求解结果，红线表示模拟退火算法迭代曲线。由图也能看出，运用 PyQUB 工具相比于模拟退火算法能较快的得到最优解。

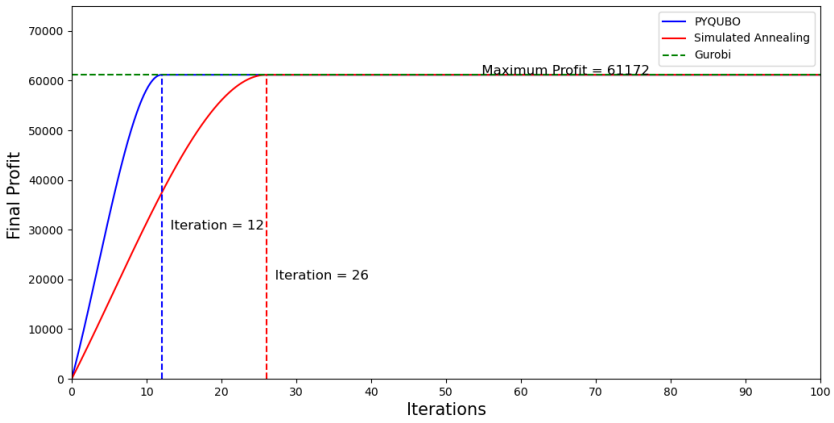

图 5.4 三种求解迭代曲线

5.2 问题二的模型建立与求解

5.2.1 构建数学模型和 QUBO 模型

(1) 构建数学模型

第一种建模方式：

本问题给定三种银行卡 1、2、3，需要决策出对应的阈值取值，使得三种组合策略

下的银行最终收入最大化。因此可以设置变量表述如下：

$$x_{1,i}、x_{2,j}、x_{3,k}(i,j,k=1,2,\cdots,10)$$

其中，信用卡 1 对应阈值i，信用卡 2 对应阈值j，信用卡 3 对应阈值k。

基于所设置的决策变量以及结合题目所给计算标准，可以计算出总通过率为$(x_{1,i}t_{1,i})(x_{2,j}t_{2,j})(x_{3,k}t_{3,k})$，总坏账率为$\frac{1}{3}(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})$，贷款利息收入为$F\eta(x_{1,i}t_{1,i})(x_{2,j}t_{2,j})(x_{3,k}t_{3,k})\left[1-\frac{1}{3}(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})\right]$，银行总坏账损失计算为$F(x_{1,i}t_{1,i})(x_{2,j}t_{2,j})(x_{3,k}t_{3,k})\frac{1}{3}(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})$，由此计算出最终银行收入如下所示：

$$\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10}\left\{F\eta(x_{1,i}t_{1,i})(x_{2,j}t_{2,j})(x_{3,k}t_{3,k})\left[1-\frac{1}{3}(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})\right]\right.$$

$$\left.-F(x_{1,i}t_{1,i})(x_{2,j}t_{2,j})(x_{3,k}t_{3,k})\frac{1}{3}(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})\right\}$$

由此，构建问题 2 的数学优化模型为：

目标函数：收益最大化

$$\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10}\left\{F\eta(x_{1,i}t_{1,i})(x_{2,j}t_{2,j})(x_{3,k}t_{3,k})\left[1-\frac{1}{3}(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})\right]\right.$$

$$\left.-F(x_{1,i}t_{1,i})(x_{2,j}t_{2,j})(x_{3,k}t_{3,k})\frac{1}{3}(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})\right\} \tag{5}$$

约束条件：每张卡片必须选择一个阈值

$$\sum_{i=1}^{10}x_{1,i}=1 \tag{6}$$

$$\sum_{j=1}^{10}x_{2,j}=1 \tag{7}$$

$$\sum_{k=1}^{10}x_{3,k}=1 \tag{8}$$

基于合理推断，问题二的目标函数里面含有三次多项式，需要将其转换为二次多项式。QUBO 三次项转化是针对于高阶问题较为棘手的一步。实际操作过程中，可以将x_1x_2替换为y，并添加约束项使得$x_1x_2=y$，从而将高次的问题转化为二次优化问题。转化过程如下所示：

$$h(x_1,x_2,y)=3y+x_1x_2-2x_1y-2x_2y$$

$$h(x_1,x_2,y)>0 \Leftrightarrow y\neq x_1x_2$$

$$h(x_1,x_2,y)=0 \Leftrightarrow y=x_1x_2$$

对目标函数进行化简分析，公式如下：

$$\begin{aligned}\max Z=&\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10}\Big\{F\eta(x_{1,i}t_{1,i})\,(x_{2,j}t_{2,j})\,(x_{3,k}t_{3,k})\Big[1-\frac{1}{3}(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})\Big]\\&-F(x_{1,i}t_{1,i})\,(x_{2,j}t_{2,j})\,(x_{3,k}t_{3,k})\frac{1}{3}(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})\Big\}\\=&F\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10}t_{1,i}t_{2,j}t_{3,k}(x_{1,i}x_{2,j}x_{3,k})\Big[\eta-\frac{1}{3}(\eta+1)\,(x_{1,i}h_{1,i}+x_{2,j}h_{2,j}+x_{3,k}h_{3,k})\Big]\\=&F\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10}\Big\{\eta t_{1,i}t_{2,j}t_{3,k}(x_{1,i}x_{2,j}x_{3,k})-\frac{1}{3}(\eta+1)t_{1,i}t_{2,j}t_{3,k}\\&\cdot(h_{1,i}x_{1,i}^2x_{2,j}x_{3,k}+h_{2,j}x_{1,i}x_{2,j}^2x_{3,k}+h_{3,k}x_{1,i}x_{2,j}x_{3,k}^2)\}\end{aligned}\tag{9}$$

其中，目标函数含有三次项和四次项，因此需要对函数进行转化，由 QUBO 的特性可知，$x_{i,j}=(x_{i,j})^2$，因此可转化原目标函数如下：

$$\begin{aligned}\max Z=&F\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10}\Big\{\eta t_{1,i}t_{2,j}t_{3,k}(x_{1,i}x_{2,j}x_{3,k})-\frac{1}{3}(\eta+1)t_{1,i}t_{2,j}t_{3,k}\\&\cdot(h_{1,i}x_{1,i}x_{2,j}x_{3,k}+h_{2,j}x_{1,i}x_{2,j}x_{3,k}+h_{3,k}x_{1,i}x_{2,j}x_{3,k})\}\\=&F\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10}\Big\{(t_{1,i}t_{2,j}t_{3,k})\Big[\eta-\frac{1}{3}(\eta+1)\,(h_{1,i}+h_{2,j}+h_{3,k})\Big](x_{1,i}x_{2,j}x_{3,k})\Big\}\end{aligned}\tag{10}$$

通过分析可知，该目标函数只有二次项，因此可以直接纳入 QUBO 公式中；由于引入了$y_{i,j}$0-1 变量，因此需要对其进行约束，将其惩罚项代入进目标函数里面。

其约束项，Rosenberg 多项式表达式如下：

$$\begin{aligned}h(x_{1,i},x_{2,j},y_{i,j})&=3y_{i,j}+x_{1,i}x_{2,j}-2x_{1,i}y_{i,j}-2x_{2,j}y_{i,j}\\h(x_{1,i},x_{2,j},y_{i,j})>0&\Leftrightarrow y_{i,j}\neq x_{1,i}x_{2,j}\\h(x_{1,i},x_{2,j},y_{i,j})=0&\Leftrightarrow y_{i,j}=x_{1,i}x_{2,j}\end{aligned}$$

约束函数依旧是一个等式约束，因此可以采用问题一相同的方式，引入惩罚标量P_1,P_2,P_3,P_4。因此得到的最终 QUBO 形式如下：

$$\begin{aligned}\max Z=&F\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10}\Big\{(t_{1,i}t_{2,j}t_{3,k})\Big[\eta-\frac{1}{3}(\eta+1)\,(h_{1,i}+h_{2,j}+h_{3,k})\Big](x_{1,i}x_{2,j}x_{3,k})\Big\}\\=&F\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10}\Big\{(t_{1,i}t_{2,j}t_{3,k})\Big[\eta-\frac{1}{3}(\eta+1)\,(h_{1,i}+h_{2,j}+h_{3,k})\Big](y_{i,j}x_{3,k})\\&+P_1\Big(\sum_{i=1}^{10}x_{1,i}-1\Big)^2+P_2\Big(\sum_{j=1}^{10}x_{2,j}-1\Big)^2+P_3\Big(\sum_{k=1}^{10}x_{3,k}-1\Big)^2\\&+P_4(3y_{i,j}+x_{1,i}x_{2,j}-2x_{1,i}y_{i,j}-2x_{2,j}y_{i,j})\}\end{aligned}\tag{11}$$

第二种建模方式：

对于第一种建模方式，为了表示三种组合策略的对应关系以及计算模型所需要的参数，设置三个决策变量进行建模。不管是模型还是求解都比较繁琐，可以考虑设置一个决策变量，构建 0-1 整数规划模型。构建变量x_{ijk}，表示如果信用卡 1 选择i，信用卡 2 选择j，信用卡 3 选择k，则变量为 1；其余情况为 0。

$$x_{ijk}=\begin{cases}1,\ \text{如果信用卡1选择阈值}i,\text{信用卡2选择阈值}j,\text{信用卡3选择阈值}k\\0,\ \text{否则}\end{cases}$$

此时计算三重信用卡组合策略下的总通过率t_{ijk}^{last}，总坏账率h_{ijk}^{last}，贷款利息收入r_{ijk}，

坏账损失l_{ijk}，银行最终收入p_{ijk}分别如下：

$$t_{ijk}^{\text{last}} = t_{1i} \cdot t_{2j} \cdot t_{3k}$$
$$h_{ijk}^{\text{last}} = 1/3 \cdot (h_{1i} + h_{2j} + h_{3j})$$
$$r_{ijk} = m \cdot n \cdot t_{ijk}^{\text{last}} \cdot (1 - h_{ijk}^{\text{last}})$$
$$l_{ijk} = m \cdot t_{ijk}^{\text{last}} \cdot h_{ijk}^{\text{last}}$$
$$p_{ijk} = r_{ijk} - l_{ijk}$$

其中，p_{ijk}所表示的更深层次的意义是：当信用卡 1 选择i阈值，信用卡 2 选择j阈值，信用卡 3 选择k阈值，这三种阈值组合策略下带来的银行最终收入。那么，可以得到问题二的目标函数如下：

$$\max z = \sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10} x_{ijk} \cdot p_{ijk} \tag{12}$$

由于信用卡 1、2、3 只能选择一种阈值，那么相应的i,j,k分别只存在一种取值。相应的x_{ijk}累计和也只能为 1。

$$\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10} x_{ijk} = 1 \tag{13}$$

将上述目标函数和约束条件转化为 QUBO 形式为：

$$\max z = \sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10} p_{ijk} \cdot x_{i,j,k}^2 - P\left(\sum_{i=1}^{10}\sum_{j=1}^{10}\sum_{k=1}^{10} x_{ijk} - 1\right)^2 \tag{14}$$

其 Q 矩阵为：

$$Q = \begin{bmatrix} p_{1,1,1} - P & -P & \cdots & \cdots & -P \\ -P & \ddots & & & \vdots \\ \vdots & & p_{i,j,k} - P & & \vdots \\ \vdots & & & \ddots & -P \\ -P & \cdots & \cdots & -P & p_{10,10,10} - P \end{bmatrix}$$

5.2.2 问题二的求解

(1) Gurobi 求解结果

Gurobi 的求解能力非常强大，可以解决各种规模的数学优化问题。具体来说，Gurobi 可以处理线性规划问题(LP)、整数规划问题(IP)、二次规划问题(QP)等类型的数学优化问题。本问题涉及 1000 个决策变量，借助 Gurobi 进行求解，求解结果如图 5.5 所示。

```
信用评分卡1对应阈值为： 8
信用评分卡2对应阈值为： 1
信用评分卡3对应阈值为： 2
最大运营成本： 27914.820479999995
程序用时: 0.13193726539611816 s
```

图 5.5 Gurobi 求解结果图

求解得组合策略为：

信用评分卡 1 对应阈值为：**8**

信用评分卡 2 对应阈值为：**1**

信用评分卡 3 对应阈值为：**2**

最大收入为：27914.82048

(2) 模拟退火算法

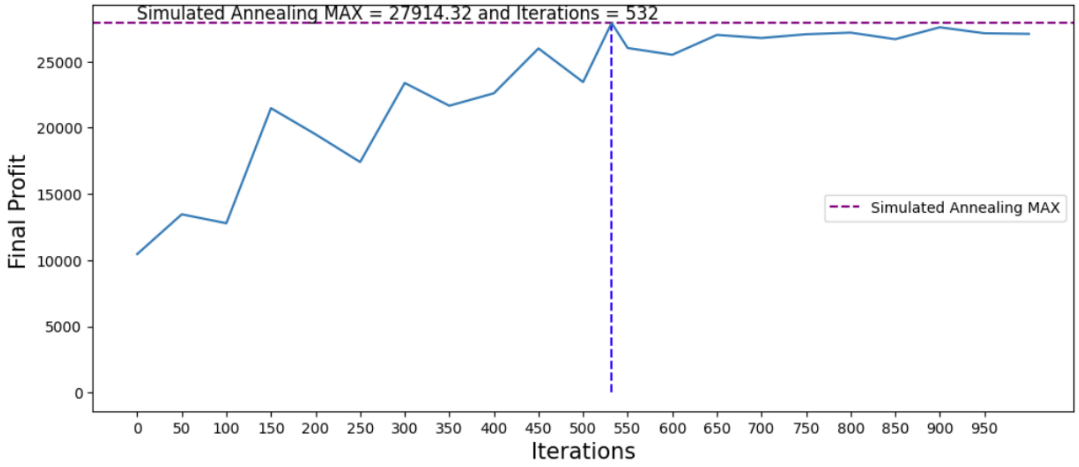

图 5.6 问题二模拟退火迭代过程所有解展示

初始温度设置为 **100 度**；衰减常数设置为 **0.95**；停止准则为迭代 **1000 次**；

求解时间为：**634.99852s**；

选择信用评分卡 1 的阈值为：**8**；

选择信用评分卡 2 的阈值为：**1**；

选择信用评分卡 3 的阈值为：**2**；

最优目标函数值：**27914.820479999995**。

产生新解的规则为：在当前解的附近随机产生一个新解，计算新解的目标函数值，重复外循环，在每次迭代过程中，Matlab 模拟退火的函数都只找一个新解，即设置 Markov 链的长度等于 1，这样能节省搜索的时间。

(3) PyQUBO 工具

本问题将模型转化为 QUBO 模型，便可以借助 PyQUBO 进行模型求解，求解结果如图 5.7 所示。

```
'x[6][3][0]': 0, 'x[6][3][1]': 0, 'x[6][3][4]': 0, 'x[6][3][5]': 0,
'x[6][3][6]': 0, 'x[6][3][7]': 0, 'x[6][5][0]': 0, 'x[6][5][1]': 0,
'x[6][5][2]': 0, 'x[6][5][3]': 0, 'x[6][5][4]': 0, 'x[6][5][5]': 0,
'x[6][5][6]': 0, 'x[6][5][7]': 0, 'x[6][5][8]': 0, 'x[6][5][9]': 0,
'x[6][6][0]': 0, 'x[6][6][1]': 0, 'x[6][6][2]': 0, 'x[6][6][3]': 0,
'x[6][6][4]': 0, 'x[6][6][5]': 0, 'x[6][6][6]': 0, 'x[6][6][7]': 0,
'x[6][6][8]': 0, 'x[6][6][9]': 0, 'x[6][7][8]': 0, 'x[6][7][9]': 0,
'x[6][8][0]': 0, 'x[6][8][1]': 0, 'x[6][8][2]': 0, 'x[6][8][3]': 0,
'x[6][8][4]': 0, 'x[6][8][5]': 0, 'x[6][8][6]': 0, 'x[6][8][7]': 0,
'x[6][8][8]': 0, 'x[6][8][9]': 0, 'x[6][9][8]': 0, 'x[6][9][9]': 0,
'x[7][0][0]': 0, 'x[7][0][1]': 1, 'x[7][0][2]': 0, 'x[7][0][3]': 0,
'x[7][0][4]': 0, 'x[9][1][9]': 0, 'x[7][0][5]': 0, 'x[9][1][8]': 0,
'x[7][0][6]': 0, 'x[7][0][7]': 0, 'x[7][0][8]': 0, 'x[9][1][5]': 0,
'x[7][0][9]': 0, 'x[9][1][4]': 0, 'x[7][1][0]': 0, 'x[7][7][6]': 0,
'x[9][4][5]': 0, 'x[9][7][4]': 0, 'x[7][1][1]': 0, 'x[7][7][7]': 0,
'x[9][4][4]': 0, 'x[9][7][5]': 0, 'x[7][1][4]': 0, 'x[7][7][2]': 0,
'x[9][4][1]': 0, 'x[9][7][0]': 0, 'x[7][1][5]': 0, 'x[7][7][3]': 0,
'x[9][4][0]': 0, 'x[9][7][1]': 0, 'x[7][1][6]': 0, 'x[7][7][0]': 0,
'x[9][4][3]': 0, 'x[9][7][2]': 0, 'x[7][1][7]': 0, 'x[7][7][1]': 0,
```

图 5.7 PyQUBO 求解解果

由于空间原因，图 5.7 仅仅展示了部分求解结果。在图 5.7 中，只有 0-1 变量

“x[7][0][1]”取值(即图 5.7 蓝色部分)为 1，其余变量取值均为 0。由于 python 索引从 0 开始，所以最终求解结果为：

信用评分卡 1 对应阈值为：**8**

信用评分卡 2 对应阈值为：**1**

信用评分卡 3 对应阈值为：**2**

最有目标函数值：27914.82048；

求解时间为：106.15896s。

5.2.3 三种求解思路对比

同样进一步分析问题本质以及了解三种求解思路的优化效果，收集三种求解过程中的数据，形成效果对比表如表 5-2 所示。

表 5-2 三种求解思路效果对比表

决策变量个数：1000	最大收入	迭代次数	迭代时间
Gurobi 求解器	27914.820479999995	-	0.1319372 s
模拟退火算法	27914.820479999995	532	634.99852s
PyQUBO 工具	27914.820479999995	116	106.15896 s

问题 2 也是一个小规模优化问题，从表可看出，Gurobi 求解器具有高效性和精确性，Gurobi 求解器的迭代时间相比于 PyQUBO 工具和模拟退火算法更短，求解速度最快。其次，运用 PyQUBO 工具求解的效果也比较好，收敛速度相较于模拟退火算法更快，适用于 QUBO 模型的求解。模拟退火算法是在解空间内进行全面搜索，从而避免局部最优解的问题，它通过接受劣解的策略在保证全局最优解的概率，因此能求解出较为优秀的解，但收敛速度比较缓慢。为进一步展示三种求解思路的效果，绘制如图所示的迭代曲线对比图。图中绿色虚线部分表示 gurobi 求解结果，蓝线表示 PyQUBO 求解结果，红线表示模拟退火算法迭代曲线。由图也能看出，运用 PyQUB 工具相比于模拟退火算法能较快的得到最优解。

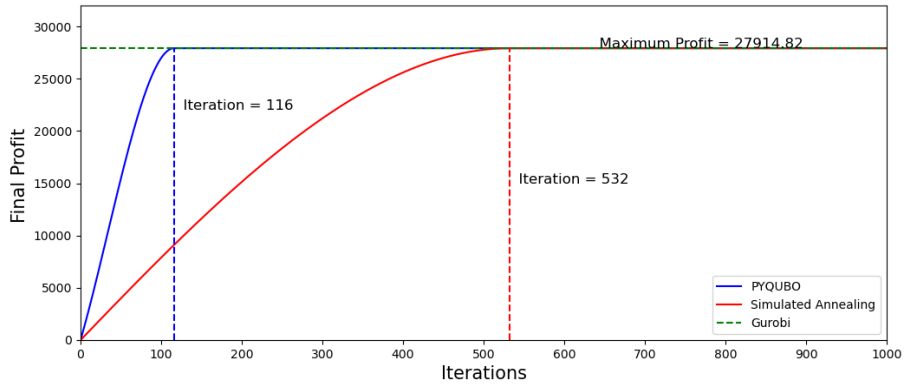

图 5.8 三种求解迭代曲线

5.3 问题三的模型建立与求解

5.3.1 构建数学模型和 QUBO 模型

(1) 构建数学模型

第一种建模思路：

通过对问题三的分析和挖掘附件数据，采用问题二相似的整数规划思想假设从 100 张信用评分卡随机选择 1 张信用卡及其阈值。设置决策变量$x_{l,i}$：

$$x_{l,i}=\begin{cases}1,\text{选择信用卡}l\text{的阈值}i\\0,\text{否则}\end{cases}$$

由于问题三中随机选择三种信用评分卡，所以变量为$x_{l,i}$、$x_{m,j}$、$x_{n,k}$。其中，l,m,n表示信用卡，i,j,k表示阈值。

此时信用卡总通过率$(x_{l,i}t_{l,i})(x_{m,j}t_{m,j})(x_{n,k}t_{n,k})$，总坏账率$\frac{1}{3}(x_{l,i}h_{l,i}+x_{m,j}h_{m,j}+x_{n,k}h_{n,k})$，贷款利息收入$F\eta(x_{l,i}t_{l,i})(x_{m,j}t_{m,j})(x_{n,k}t_{n,k})\left[1-\frac{1}{3}(x_{l,i}h_{l,i}+x_{m,j}h_{m,j}+x_{n,k}h_{n,k})\right]$，总坏账损失为$F(x_{l,i}t_{l,i})(x_{m,j}t_{m,j})(x_{n,k}t_{n,k})\frac{1}{3}(x_{l,i}h_{l,i}+x_{m,j}h_{m,j}+x_{n,k}h_{n,k})$。由此得到目标函数为：

$$\begin{aligned}\min\ z=&\sum_{\substack{l,m,n=1\\l\neq m\neq n}}^{100}\sum_{i,j,k=1}^{10}\left\{F\eta(x_{l,i}t_{l,i})(x_{m,j}t_{m,j})(x_{n,k}t_{n,k})\left[1-\frac{1}{3}(x_{l,i}h_{l,i}+x_{m,j}h_{m,j}+x_{n,k}h_{n,k})\right]\right.\\&\left.-F(x_{l,i}t_{l,i})(x_{m,j}t_{m,j})(x_{n,k}t_{n,k})\frac{1}{3}(x_{l,i}h_{l,i}+x_{m,j}h_{m,j}+x_{n,k}h_{n,k})\right\}\\=&F\sum_{l,m,n=1}^{100}\sum_{i,j,k=1}^{10}\left\{(t_{l,i}t_{m,j}t_{n,k})\left[\eta-\frac{1}{3}(n+1)(h_{l,i}+h_{m,j}+h_{n,k})\right](x_{l,i}x_{m,j}x_{n,k})\right\}\end{aligned}\tag{15}$$

约束条件为：

$$\sum_{l=1}^{100}\sum_{i=1}^{10}x_{l,i}=1\tag{16}$$

$$\sum_{m=1}^{100}\sum_{j=1}^{10}x_{m,j}=1\tag{17}$$

$$\sum_{n=1}^{100}\sum_{k=1}^{10}x_{n,k}=1\tag{18}$$

$$\sum_{\substack{l,m,n=1\\l\neq m\neq n}}^{100}\sum_{i,j,k=1}^{10}(x_{l,i}+x_{m,j}+x_{n,k})=3\tag{19}$$

由 QUBO 的特性可得，$x_{l,i}=x_{l,i}^2$，$x_{m,j}=x_{m,j}^2$，$x_{n,k}=x_{n,k}^2$。

因此，可将目标函数转换为 QUBO 形式如下：

$$\max Z=F\sum_{\substack{l,m,n=1\\l\neq m\neq n}}^{100}\sum_{i,j,k=1}^{10}\left\{(t_{l,i}t_{m,j}t_{n,k})\left[\eta-\frac{1}{3}(\eta+1)(h_{l,i}+h_{m,j}+h_{n,k})\right](x_{l,i}x_{m,j}x_{n,k})\right\}\tag{20}$$

令$y_{i,j}^{l,m}=x_{l,i}x_{m,j}$，$y_{i,j}^{l,m}$为 0-1 变量，转换上述目标函数如下：

$$\max\ z=F\sum_{\substack{l,m,n=1\\l\neq m\neq n}}^{100}\sum_{i,j,k=1}^{10}\left\{(t_{l,i}t_{m,j}t_{n,k})\left[\eta-\frac{1}{3}(\eta+1)(h_{l,i}+h_{m,j}+h_{n,k})\right](y_{i,j}^{l,m}x_{n,k})\right\}\tag{21}$$

引入$y_{i,j}^{l,m}$变量后，其约束项：Rosenberg 多项式表达式如下：

$$h(x_{l,i},x_{m,j},y_{i,j}^{l,m})=3y_{i,j}^{l,m}+x_{l,i}x_{m,j}-2x_{l,i}y_{i,j}^{l,m}-2x_{m,j}y_{i,j}^{l,m}$$

目标函数最终的 QUBO 形式如下：

$$\begin{aligned}\max Z=&F\sum_{\substack{l,m,n=1\\l\neq m\neq n}}^{100}\sum_{i,j,k=1}^{10}\left\{(t_{l,i}t_{m,j}t_{n,k})\left[\eta-\frac{1}{3}(\eta+1)(h_{l,i}+h_{m,j}+h_{n,k})\right](y_{i,j}^{l,m}x_{n,k})\right.\\&-P_1\left(\sum_{l=1}^{100}\sum_{i=1}^{10}x_{l,i}-1\right)^2-P_2\left(\sum_{m=1}^{100}\sum_{j=1}^{10}x_{m,j}-1\right)^2-P_3\left(\sum_{n=1}^{100}\sum_{k=1}^{10}x_{n,k}-1\right)^2\\&\left.-P_4(3y_{i,j}^{l,m}+x_{l,i}x_{m,j}-2x_{l,i}y_{i,j}^{l,m}-2x_{m,j}y_{i,j}^{l,m})\right\}\end{aligned}\tag{22}$$

第二种建模方式：

设置决策变量：

$$x_{ijk}^{lmn}=\begin{cases}1,\text{ 如果信用卡}l\text{对应阈值}i\text{,卡}m\text{对应阈值}j\text{,卡}n\text{对应阈值}k\\0,\text{ 否则}\end{cases}$$

如果在 100 张信用卡里面选择了信用卡l，信用卡m和信用卡n，其对应阈值分别选择i,j,k，此时计算三重信用卡组合策略下的总通过率t_{ijk}^{lmn}，总坏账率h_{ijk}^{lmn}，贷款利息收入r_{ijk}^{lmn}，坏账损失l_{ijk}^{lmn}，银行最终收入p_{ijk}^{lmn}分别如下：

$$t_{ijk}^{lmn}=t_{l,i}t_{m,j}t_{n,k}$$
$$h_{ijk}^{lmn}=1/3\cdot(h_{l,i}+h_{m,j}+h_{n,k})$$
$$r_{ijk}^{lmn}=m\cdot n\cdot t_{ijk}^{lmn}\cdot(1-h_{ijk}^{lmn})$$
$$l_{ijk}^{lmn}=m\cdot t_{ijk}^{lmn}\cdot h_{ijk}^{lmn}$$
$$p_{ijk}^{lmn}=r_{ijk}^{lmn}-l_{ijk}^{lmn}$$

p_{ijk}^{lmn}所表示的更深层次的意义是：当选择了信用卡l，信用卡m和信用卡n，其对应阈值分别选择i,j,k这三种阈值组合策略下，带来的银行最终收入。那么，可以得到问题三的目标函数如下：

$$\max z=\sum_{\substack{l,m,n=1\\l\neq m\neq n}}^{100}\sum_{i,j,k=1}^{10}x_{ijk}^{lmn}\cdot p_{ijk}^{lmn} \tag{23}$$

由于信用卡l,m,n只能选择一种阈值，那么相应的i,j,k分别只存在一种取值。相应的，x_{ijk}^{lmn}累计和也只能为 1。

$$\sum_{\substack{l,m,n=1\\l\neq m\neq n}}^{100}\sum_{i,j,k=1}^{10}x_{ijk}^{lmn}=1 \tag{24}$$

将上述目标函数和约束条件转化为 QUBO 形式为：

$$\max z=\sum_{\substack{l,m,n=1\\l\neq m\neq n}}^{100}\sum_{k=1}^{10}p_{ijk}^{lmn}\cdot(x_{ijk}^{lmn})^2-P\left(\sum_{\substack{l,m,n=1\\l\neq m\neq n}}^{100}\sum_{k=1}^{10}x_{ijk}^{lmn}-1\right)^2 \tag{25}$$

其 Q 矩阵为：

$$Q=\begin{bmatrix}p_1-P & -P & \cdots & \cdots & -P\\ -P & \ddots & & & \vdots\\ \vdots & & p_{ijklmn}-P & & \vdots\\ \vdots & & & \ddots & -P\\ -P & \cdots & \cdots & -P & p_{10^3\cdot C_{100}^3}-P\end{bmatrix}$$

5.3.2 问题三的求解

由于问题三规模较大，变量非常多，所以采用 Gurobi 求解器会发生“记忆错误”。因此，无法用其解决本问题。针对问题三，仅仅采取模拟退火和 PyQUBO 工具来求解。

(1) 模拟退火算法

由于数据量较大，为了将这个约束条件转化为 QUBO 模型的形式，使用布尔满足问题（Boolean Satisfiability Problem，简称 BSP）将“只能选择三张信用评分卡”的约

束条件转化为 QUBO 模型的形式，使用了 D-Wave Leap API 进行量子求解。求解部分结果如图 5.9 所示

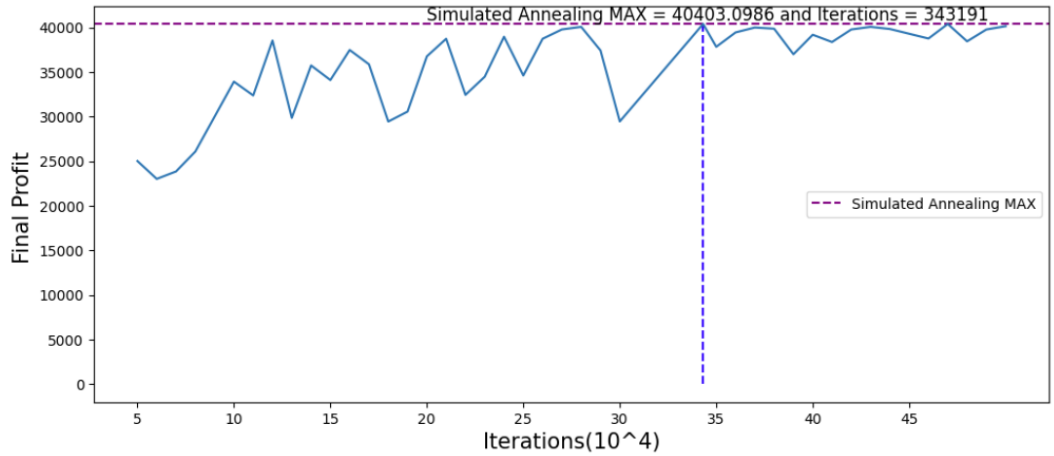

图 5.9 模拟退火算法求解结果

初始温度设置为 100 度；
衰减常数设置为 0.95；
停止准则为迭代 5*10^5 次；
求解时间为：21798.516257s；
选择的信用评分卡为 8 号、33 号、49 号
选择信用评分卡 1 的阈值为：4；
选择信用评分卡 2 的阈值为：8；
选择信用评分卡 3 的阈值为：9；
最优目标函数值：40403.0986。

产生新解的规则为：在当前解的附近随机产生一个新解，计算新解的目标函数值，重复外循环，在每次迭代过程中，Matlab 模拟退火的函数都只找一个新解，即设置 Markov 链的长度等于 1，这样能节省搜索的时间。

(2) PyQUBO 工具

由于 QUBO 模型可以描述许多实际问题，因此 PyQUBO 可以用于求解多种优化问题，包括大规模问题。PyQUBO 可以用于求解许多实际问题，但在处理大规模问题时，需要注意计算资源的限制和算法的有效性。在实际应用过程中，可以通过使用高效的 QUBO 模型或者优化求解参数来提高其求解能力。在描述问题时，应尽可能使用高效的 QUBO 模型，以减少变量和约束的数量。这可以通过对问题进行适当的转化来实现。此外，PyQUBO 提供了一些参数，如退火温度和退火时间等，可以通过调整这些参数来提高求解的效率和准确性。

本问题借助 PyQUBO 工具来求解，得到如图 5.10 所示得结果图。

```
'x[7][32][48][1][4][3]': 0,'x[7][32][48][1][4][4]': 0,
'x[7][32][48][1][4][5]': 0,'x[7][32][48][1][4][6]': 0,
'x[7][32][48][1][4][7]': 0,'x[7][32][48][1][4][8]': 0,
'x[7][32][48][1][4][9]': 0,'x[7][32][48][1][5][0]': 0,
'x[7][32][48][1][5][1]': 0,'x[7][32][48][1][5][2]': 1,
'x[7][32][48][1][5][3]': 0,'x[7][32][48][1][5][4]': 0,
'x[7][32][48][1][5][5]': 0,'x[7][32][48][1][5][6]': 0,
'x[7][32][48][1][5][7]': 0,'x[7][32][48][1][5][8]': 0,
'x[7][32][48][1][5][9]': 0,'x[7][32][48][1][6][0]': 0,
'x[7][32][48][1][6][1]': 0,'x[7][32][48][1][6][2]': 0,
'x[7][32][48][1][6][3]': 0,'x[7][32][48][1][6][4]': 0,
'x[7][32][48][1][6][5]': 0,'x[7][32][48][1][6][6]': 0
```

图 5.10 PyQUBO 求解结果

由于空间原因，图 5.10 仅仅展示了部分求解结果。在图 5.10 中，只有 0-1 变量

“x[7][32][48][1][5][2]”取值(即图 5.10 蓝色部分)为 1，其余变量取值均为 0。由于 python 索引从 0 开始，所以最终求解结果为：

信用评分卡为：**8**　　对应阈值为：**2**
信用评分卡为：**33**　　对应阈值为：**6**
信用评分卡为：**49**　　对应阈值为：**3**
最有目标函数值：**43880.9712**；
求解时间为：5412.26335s。

5.3.3 三种求解思路对比

同样进一步分析问题本质以及了解三种求解思路的优化效果，收集三种求解过程中的数据，形成效果对比表如表 5-3 所示。

表 5-3 三种优化效果对比表

决策变量个数：1000	最大收入	迭代次数	迭代时间
Gurobi 求解器	-	-	-
模拟退火算法	40403.0986	128617	21798.516257s
PyQUBO 工具	43880.9712	343191	5412.26335s

问题 3 为一个大规模优化问题，其决策变量个数相较于问题 1 和问题 2 来说大幅度增加。从表可看，由于求解规模太大，Gurobi 求解器需要大量的内存来存储问题的数据结构和求解过程中的中间结果，出现了记忆出错问题，无法求解出最终结果。其次，PyQUBO 工具使用了量子计算的概念和技术，可以利用量子并行计算、量子态叠加和量子干涉等优势，快速搜索问题的解空间，且其使用稀疏矩阵存储问题数据，可以有效地处理大规模问题的数据结构，减少了内存的使用和计算时间的复杂度。因此其求解效果较好，收敛速度较快。求解复杂的大规模问题时，模拟退火算法可能需要迭代很多次才能找到最优解，而迭代次数的增加会增加算法的时间复杂度，因此其收敛速度较慢、迭代时间较长。为进一步展示三种求解思路的效果，绘制如图所示的迭代曲线对比图。图中蓝线表示 PyQUBO 求解结果，红线表示模拟退火算法迭代曲线。由图也能看出，对于大规模优化问题，运用 PyQUB 工具能更快的求解出结果。

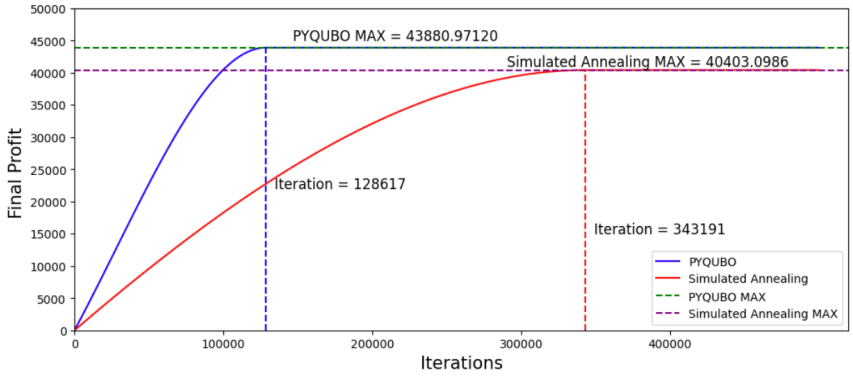

图 5. 11 三种求解迭代曲线

由图 5.1 可知，调用 PYQUBO 工具箱求解效率和求解效果远远优于模拟退火算法。因此，问题三调用 PYQUBO 工具箱求得的最优目标函数值为：银行总收入为 43880.9712；借助启发式算法—模拟退火算法求得的最优目标函数值为：银行总收入为 40403.0986。两种求解结果相差 3477.8752，即相比于模拟退火算法，调用 PYQUBO 工具箱求解效果提升 8.61%，求解时间提升 75%。由此可见，针对大规模优化问题，采用 QUBO 模型及其适用的算法更优于其他的智能优化算法。

六、模型的评价与推广

6.1 模型的优点

(1) 本文问题 1，2 建模过程中，分析了选用信用评分卡及其阈值可能出现的所有情况，推到每种情况出现的条件，建立了 0-1 整数规划模型，模型严谨准确；可直接使用 Gurobi 求解器进行求解，Gurobi 是一种高校的求解器，具有高精度的求解能力，对于小规模优化问题可以得到非常精确的解决方案；由于模型较简便，采用模拟退火算法也能高效求解；PyQUBO 工具对于小规模优化问题也可以直接快速求解；

(2) 对于问题 3 这种大规模 0-1 整数规划模型，PyQUBO 工具由于使用了量子计算的概念和技术且其使用稀疏矩阵存储问题数据，因此比模拟退火算法、Gurobi 求解器能更快地找到全局最优解；模拟退火算法其优点是能够在解空间内进行全面搜索，从而避免局部最优解的问题，它通过接受劣解的策略来保证全局最优解的概率。

6.2 模型的缺点

(1) QUBO 模型只能解决二进制变量的问题，如果问题需要使用连续变量或离散变量，需要将其转换为二进制形式，这可能会导致问题变得更加复杂。。

(2) 对于问题 3 的大规模计算，模拟退火算法可能需要大量的计算时间来达到最优解，特别是对于复杂的问题，可能需要进行数百万次迭代才能找到最优解；且模拟退火算法的解是随机的，因此在不同的运行中可能会得到不同的结果，这可能会使其在某些情况下不可靠。

(3) Gurobi 求解器对于大规模优化问题，可能存在内存限制，导致求不出结果，且其非线性规划问题处理能力有限，尤其对于某些复杂的非线性规划问题不一定能求解。

6.3 模型的推广

本文建立的 0-1 整数规划模型，约束条件全部是线性表达式，可以在高效的求解器上进行求解，为今后模型求解提供良好的模型基础。建立的 QUBO 模型，形式简单，易于理解和实现，对于 QUBO 模型的推广有以下几个方面进行：

(1) 扩展到整数优化问题：QUBO 模型只考虑了变量取值为二进制值的情况，但是有些问题需要考虑变量取值为整数的情况。为了解决这种问题，可以将 QUBO 模型扩展到整数优化问题，即将变量限制为整数取值，同时将目标函数进行相应的修改。

(2) 扩展到连续优化问题：QUBO 模型只能处理离散问题，而有些问题需要考虑变量取值为连续值的情况。为了解决这种问题，可以将 QUBO 模型扩展到连续优化问题，即将变量限制为连续取值，同时将目标函数进行相应的修改。

(3) 扩展到多目标优化问题：QUBO 模型只能处理单一目标的优化问题，而有些问题需要考虑多个目标的优化问题。为了解决这种问题，可以将 QUBO 模型扩展到多目标优化问题，即将目标函数进行相应的修改，同时考虑多个目标函数的权重关系。

(4) 扩展到多约束优化问题：QUBO 模型只能处理无约束或简单约束的优化问题，而有些问题需要考虑多个复杂约束的优化问题。为了解决这种问题，可以将 QUBO 模型扩展到多约束优化问题，即将目标函数进行相应的修改，同时考虑多个复杂约束条件的限制。

参考文献

[1] 杨绘宏. 我国信用卡市场现状与发展举措[J]. 中国信用卡(专业),2010(8):46-49.
[2] 陈粘. 高维金融市场的时变投资组合优化研究[D].成都理工大学,2021.
[3] 马孝先. 若干投资组合优化问题模型及算法的研究[D].山东师范大学,2007.
[4] Burer S, Vandenbussche D. Solving lift-and-project relaxations of binary integer programs[J]. SIAM Journal on Optimization, 2006, 16(3): 726-750.
[5] 王宝楠,水恒华,王苏敏,胡风,王潮.量子退火理论及其应用综述[J].中国科学:物理学 力学 天文学,2021,51(08):5-17.
[6] Papalitsas C, Andronikos T, Giannakis K, et al. A QUBO model for the traveling salesman problem with time windows[J]. Algorithms, 2019, 12(11): 224.
[7] Glover F, Kochenberger G, Hennig R, et al. Quantum bridge analytics I: a tutorial on formulating and using QUBO models[J]. Annals of Operations Research, 2022, 314(1): 141-183.
[8] Morstyn T. Annealing-based Quantum Computing for Combinatorial Optimal Power Flow[J]. IEEE Transactions on Smart Grid, 2022.
[9] Delahaye D, Chaimatanan S, Mongeau M. Simulated annealing: From basics to applications[J]. Handbook of metaheuristics, 2019: 1-35.
[10] Wang K, Li X, Gao L, et al. A genetic simulated annealing algorithm for parallel partial disassembly line balancing problem[J]. Applied Soft Computing, 2021, 107: 107404.
[11] Ghannadi P, Kourehli S S, Mirjalili S. A review of the application of the simulated annealing algorithm in structural health monitoring (1995-2021)[J]. Frattura ed Integrità Strutturale, 2023, 17(64): 51-76.
[12] Wang K, Li X, Gao L, et al. A genetic simulated annealing algorithm for parallel partial disassembly line balancing problem[J]. Applied Soft Computing, 2021, 107: 107404.
[13] 汪勇,孟香君,沈维萍.量子计算在经济与金融领域中的应用[J].经济学动态,2023(01):126-143.
[14] 严炜,龙长江,李善军.基于差分量子退火算法的农用无人机路径规划方法[J].华中农业大学学报,2020,39(01):180-186.

附录

第一部分

数据分析、问题 1 问题 2 结果验证、可视化展示 Python 代码：

1：数据可视化分析

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
data_0 = pd.read_csv("附件 1：data_100.csv")
data_t = data_0.filter(like='t_')
data_h = data_0.filter(like='h_')
fig, ax = plt.subplots(figsize=(30, 10))
heatmap = ax.imshow(data_h, cmap='coolwarm')
cbar = ax.figure.colorbar(heatmap, ax=ax)
cbar.ax.set_ylabel('Bad Debt Rate', rotation=-90, va="bottom",fontsize=30)
cbar.ax.tick_params(labelsize=20)
ax.set_xlabel('Credit Scorecard Number Index',fontsize=30)
ax.set_ylabel('Threshold Number Index',fontsize=30)
ax.set_title('Trend of Bad Debt Rate Changes',fontsize=40)
# 设置横坐标和纵坐标的刻度
x_ticks = np.arange(0, 101, 2)
y_ticks = np.arange(0, 11, 1)
ax.set_xticks(x_ticks)
ax.set_yticks(y_ticks)
# 将纵坐标进行翻转
ax.invert_yaxis()
plt.xticks(rotation=-20)
plt.savefig('数据分析_100 种信用评分卡坏账率变化趋势.png')
plt.show()
fig, ax = plt.subplots(figsize=(30, 10))
heatmap = ax.imshow(data_t, cmap='coolwarm')
cbar = ax.figure.colorbar(heatmap, ax=ax)
cbar.ax.set_ylabel(' Approval ', rotation=-90, va="bottom",fontsize=30)
cbar.ax.tick_params(labelsize=20)
ax.set_xlabel('Credit Scorecard Number Index',fontsize=30)
ax.set_ylabel('Threshold Number Index',fontsize=30)
ax.set_title('Trend of Approval Rate Changes',fontsize=40)
# 设置横坐标和纵坐标的刻度
x_ticks = np.arange(0, 101, 2)
y_ticks = np.arange(0, 11, 1)
ax.set_xticks(x_ticks)
ax.set_yticks(y_ticks)
# 将纵坐标进行翻转
ax.invert_yaxis()
plt.xticks(rotation=-20)
plt.savefig('数据分析_100 种信用评分卡通过率变化趋势.png')
plt.show()
```

2：问题 1 结果验证

```
import pandas as pd
import time
# 记录开始时间
start_time = time.time()
data = pd.read_csv(" 附件 1：data_100.csv")
grouped_data = pd.DataFrame()
for i in range(0, 200, 2):
    group = [data.iloc[:,i], data.iloc[:,i+1], data.iloc[:,i]*1000000*0.08*(1-data.iloc[:,i+1]) -
```

```
data.iloc[:,i]*1000000*data.iloc[:,i+1]]
        group = pd.DataFrame(group)
        group = group.T
        group = group.rename(columns={'Unnamed 0': 'Result'})
        grouped_data = pd.concat([grouped_data, group], axis=1)
# 查找 dataframe 中所有值中的最大值
max_value = grouped_data.values.max()
# 查找 dataframe 中最大值的位置
max_position = divmod(grouped_data.values.argmax(), grouped_data.shape[1])
# 显示结果
print('Max value:', max_value)
print('Position of max value:', max_position)
# 记录结束时间
end_time = time.time()
# 计算代码运行时间
duration = end_time - start_time
print(duration)
```

Output:

```
Max value: 61172.0
Position of max value: (0, 146)
0.2648329734802246
```

3：问题二结果验证

```
import pandas as pd
import time
# 记录开始时间
start_time = time.time()
data = pd.read_csv("附件 1：data_100.csv")
data_1 = data.iloc[:,:2]
data_2 = data.iloc[:,2:4]
data_3 = data.iloc[:,4:6]
Q2_result = pd.DataFrame()
for i in range(0,10,1):
        a_1 = data_1.iloc[i,0]
        b_1 = data_1.iloc[i,1]
        for j in range(0,10,1):
                a_2 = data_2.iloc[j,0]
                b_2 = data_2.iloc[j,1]
                for k in range(0,10,1):
                        a_3 = data_3.iloc[k,0]
                        b_3 = data_3.iloc[k,1]
                        Interest_Income = 1000000*0.08*(a_1*a_2*a_3) * (1-1/3*(b_1+b_2+b_3))
                        Bad_Debt_Loss = 1000000*(a_1*a_2*a_3) * (1/3*(b_1+b_2+b_3))
                        Net_Income = Interest_Income - Bad_Debt_Loss
                        r = [i+1, j+1, k+1, Interest_Income, Bad_Debt_Loss, Net_Income]
                        r = pd.DataFrame(r)
                        r = r.T
                        Q2_result = pd.concat([Q2_result, r], axis=0)

Q2_result = Q2_result.rename(columns={0: '信用评分卡 1 阈值', 1: '信用评分卡 2 阈值',2: '信用评分卡 3
阈值', 3: '利息收入',4: '损失', 5: '总收入'})
# 按数字顺序重新排序行索引
Q2_result = Q2_result.reset_index(drop=True)
# 找到总收入列中的最大值及其索引
max_value = Q2_result['总收入'].max()
max_value_index = Q2_result['总收入'].idxmax()
# 输出最大值及其对应的那一行
print("最大收入是：", max_value)
```

```
print( "最大收入的阈值组合是:")
print(Q2_result.loc[max_value_index])
# 记录结束时间
end_time = time.time()
# 计算代码运行时间
duration = end_time - start_time
print("计算时间：",duration)
```
Output：
```
最大收入是： 27914.820479999995
最大收入的阈值组合是:
信用评分卡 1 阈值          8.00000
信用评分卡 2 阈值          1.00000
信用评分卡 3 阈值          2.00000
利息收入           42739.76448
损失             14824.94400
总收入            27914.82048
Name: 701, dtype: float64
计算时间： 0.5260212421417236
```
4：问题一结果可视化
```
#热力图
r1 = pd.read_csv(r"E:\2023mathorcup\Q1_result.csv")
r1_1 = r1.iloc[:, 2::3].values
r1_1 = pd.DataFrame(r1_1)
# 将列索引从 1 开始排序
r1_1.columns = range(1, len(r1_1.columns) + 1)
fig, ax = plt.subplots(figsize=(30, 10))
heatmap = ax.imshow(r1_1, cmap='YlOrRd')
cbar = ax.figure.colorbar(heatmap, ax=ax)
cbar.ax.set_ylabel('Value', rotation=-90, va="bottom",fontsize=30)
cbar.ax.tick_params(labelsize=20)
ax.set_xlabel('Credit Scorecard Number Index',fontsize=30)
ax.set_ylabel('Threshold Number Index',fontsize=30)
ax.set_title('Data Trends',fontsize=40)
# 设置横坐标和纵坐标的刻度
x_ticks = np.arange(0, 101, 2)
y_ticks = np.arange(0, 11, 1)
ax.set_xticks(x_ticks)
ax.set_yticks(y_ticks)
# 将纵坐标进行翻转
ax.invert_yaxis()
plt.xticks(rotation=-20)
#plt.savefig('Q1_可视化 1.png')
plt.show()
#散点图
r1_1 = r1.iloc[:, 2::3].values
# 获取数据的行列数
num_rows, num_cols = r1_1.shape
# 生成每个点的 x，y 坐标和对应的数值
x = np.tile(np.arange(1,num_cols+1,1), (num_rows, 1)).ravel()
y = np.tile(np.arange(1,num_rows+1,1), (num_cols, 1)).T.ravel()
c = r1_1.ravel()  # 将二维数组平展为一维数组
# 设置图形大小
plt.figure(figsize=(20, 10))
# 绘制散点图
plt.scatter(x, y, c=c, cmap='viridis')
plt.colorbar()  # 添加颜色条
```

```
# 设置 x，y 轴标签和标题
plt.xlabel('Credit Scorecard Number Index',fontsize=20)
plt.ylabel('Threshold Number Index',fontsize=20)
plt.title('Data Trends',fontsize=30)
# 设置横坐标刻度和数字大小，以及间隔为 1
plt.xticks(np.arange(0,101,5), fontsize=15)
# 设置纵坐标刻度和数字大小，以及间隔为 1
plt.yticks(np.arange(0,11,1), fontsize=15)
#plt.savefig('Q1_可视化 2.png')
# 显示图形
plt.show()
#通过率与总收入的关系曲线
fig, axs = plt.subplots(10, 10, figsize=(12, 12))
for i in range(1,101,1):
    x = r1.iloc[:,i*3-3]
    y = r1.iloc[:,i*3-1]
    row = (i-1) // 10
    col = (i-1) % 10
    axs[row, col].plot(x, y)
    axs[row, col].set_title(f"Card {i}", fontdict={'fontsize': 9}) # 设置子图标题
    axs[row, col].tick_params(axis='both', which='major', labelsize=5)
plt.suptitle("Relationship between Approval Rate and Total Revenue", fontsize=16)
plt.tight_layout()
plt.show()
fig.savefig("通过率与总收入的关系曲线.png", dpi=300, bbox_inches='tight')
#坏账率与总收入的关系曲线
fig, axs = plt.subplots(10, 10, figsize=(12, 12))
for i in range(1,101,1):
    x = r1.iloc[:,i*3-2]
    y = r1.iloc[:,i*3-1]
    row = (i-1) // 10
    col = (i-1) % 10
    axs[row, col].plot(x, y)
    axs[row, col].set_title(f"Card {i}", fontdict={'fontsize': 9}) # 设置子图标题
    axs[row, col].tick_params(axis='both', which='major', labelsize=5)
plt.suptitle("Relationship between Bad Debt Rate and Total Revenue", fontsize=16)
plt.tight_layout()
plt.show()
fig.savefig("坏账率与总收入的关系曲线.png", dpi=300, bbox_inches='tight')
# 阈值与总收入的关系曲线
fig, axs = plt.subplots(10, 10, figsize=(12, 12))
for i in range(1,101,1):
    x = range(1,11,1)
    y = r1.iloc[:,i*3-1]
    row = (i-1) // 10
    col = (i-1) % 10
    axs[row, col].plot(x, y)
    axs[row, col].set_title(f"Card {i}", fontdict={'fontsize': 9}) # 设置子图标题
    axs[row, col].tick_params(axis='both', which='major', labelsize=5)
plt.suptitle("Relationship between Threshold and Total Revenue", fontsize=16)
plt.tight_layout()
plt.show()
fig.savefig("阈值与总收入的关系曲线.png", dpi=300, bbox_inches='tight')
5:问题 2 结果可视化
r2 = pd.read_csv(r"E:\2023mathorcup\Q2_result.csv")
r2.drop('Unnamed: 0',axis = 1,inplace = True)
r2_1 = r2[['信用评分卡 1 阈值','信用评分卡 2 阈值','信用评分卡 3 阈值','总收入']]
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# 分离列数据
x = r2_1.iloc[:, 0]
y = r2_1.iloc[:, 1]
z = r2_1.iloc[:, 2]
c = r2_1.iloc[:, 3]
# 创建散点图
fig = plt.figure(figsize=(15, 12))
ax = fig.add_subplot(111, projection='3d')
sc = ax.scatter(x, y, z, c=c, cmap='coolwarm')
# 添加颜色映射
cbar = plt.colorbar(sc)
cbar.ax.set_ylabel('Total Income')
# 设置坐标轴标签
ax.set_xlabel('card1 ')
ax.set_ylabel('card2 ')
ax.set_zlabel('card3 ')
plt.savefig('Q2_可视化.png')
plt.show()
```

第二部分

问题求解程序

(1) 问题 1

Gurobi 求解

```
# -*- coding: utf-8 -*-
import time
import csv
import random
import gurobipy as gp
from gurobipy import GRB
import numpy as np
import copy
import math
from itertools import islice
print('程序开始时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
start = time.time()
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"""基础数据"""
I = np.arange(0,100,1)
J = np.arange(0,10,1)

T = []
H = []
for i in range(100):
    t_l = 't_' + str(i+1)
    T.append(t_l)
    h_l = 'h_' + str(i+1)
    H.append(h_l)
tij = {}
hij = {}

data_file ='C:/Users/ASUS/Desktop/question/data.csv'
with open(data_file,'r') as f:  #读写“data”文件
    data_reader=csv.DictReader(f)
    j = 0
    for row in data_reader:
```

```
        for i in range(100):
            tij[(i,j)] = float(row[T[i]])
            hij[(i,j)] = float(row[H[i]])
        j = j + 1

current_obj = []
def mycallback(m, where):
    if where == GRB.Callback.MIP:
        current_obj.append(m.cbGet(GRB.Callback.MIP_OBJBST))
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
m = gp.Model('integer linear programming model')

x = m.addVars(I, J, vtype = gp.GRB.BINARY, name="x")
m.update()
#-----------------------------添加选取一种约束-----------------------
xij_sum = 0
for i in I:
    for j in J:
        xij_sum = xij_sum + x[(i,j)]
m.addConstr(xij_sum == 1)
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'''目标函数'''
obj = 0
for i in I:
    for j in J:
        obj = obj + (1000000*(0.08*tij[(i,j)]*(1-hij[(i,j)])-tij[(i,j)]*hij[(i,j)]))*x[(i,j)]
m.setObjective(obj , GRB.MAXIMIZE)

# m.optimize
m.optimize(mycallback)
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"""结果输出"""
print()
if m.status == GRB.OPTIMAL:
    for i in I:
        for j in J:
            if x[(i,j)].x != 0:
                print("选择的信用评分卡序号为：",i+1)
                print("对应阈值为：",j+1)
    print('最大运营成本：', m.objVal)
    # print(current_obj)
else:
      print('No solution')

end = time.time()
# print('程序结束时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
print('程序用时:',end - start,'s')
```

PyQUBO

```
from pyqubo import Array, Placeholder, Constraint
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import time
import csv
import random
print('程序开始时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
start = time.time()
I = np.arange(1,101,1)
```

```
J = np.arange(1,11,1)
n_I = len(I)
n_J = len(J)
x = Array.create('x', (n_I, n_J), 'BINARY')
print(x)
xij_sum = 0.0
xij_sum += Constraint((sum(x[i, j] for i in range(n_I) for j in range(n_J)) - 1)**2, label="xij{}".format(1))
T = []
H = []
for i in range(100):
    t_l = 't_' + str(i+1)
    T.append(t_l)
    h_l = 'h_' + str(i+1)
    H.append(h_l)
tij = {}
hij = {}
data_file ='C:/Users/ASUS/Desktop/1/data.csv'
with open(data_file,'r') as f:  #读写“data”文件
    data_reader=csv.DictReader(f)
    j = 0
    for row in data_reader:
        for i in range(100):
            tij[(i,j)] = float(row[T[i]])
            hij[(i,j)] = float(row[H[i]])
        j = j + 1
w = 0.0
for i in range(n_I):
    for j in range(n_J):
        w += ((1000000*(0.08*tij[(i,j)]*(1-hij[(i,j)])-tij[(i,j)]*hij[(i,j)])) * x[i,j] * x[i,j])
A = Placeholder("A")
H = w + A * (xij_sum)
# Compile model
model = H.compile()
# Generate QUBO
feed_dict = {'A': 100000.0}
bqm = model.to_bqm(feed_dict=feed_dict)
import neal
sa = neal.SimulatedAnnealingSampler()
sampleset = sa.sample(bqm, num_reads=100, num_sweeps=100)

# Decode solution
decoded_samples = model.decode_sampleset(sampleset, feed_dict=feed_dict)
best_sample = max(decoded_samples, key=lambda x: x.energy)
num_broken = len(best_sample.constraints(only_broken=True))
print("number of broken constarint = {}".format(num_broken))
if num_broken == 0:
    print(best_sample.sample)
end = time.time()
print('程序结束时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
print('程序用时:',end - start,'s')
```

模拟退火算法求解问题一：

```
import numpy as np
from scipy.optimize import minimize
import random
%% 参数初始化
T0 = 100;     % 初始温度
T = T0;
```

```
maxgen = 100;   % 最大迭代次数
Lk = 200;   % 每个温度下的迭代次数
alfa = 0.95;   % 温度衰减系数
def simulated_annealing(cost, initial_state, T, max_iter, cool_rate):
    state = initial_state
    cost_cur = cost(state)
    T_cur = T
    for i in range(max_iter):
        T_cur *= cool_rate
        state_new = flip(state)
        cost_new = cost(state_new)
        cost_delta = cost_new - cost_cur
        if cost_delta <= 0 or np.exp(-cost_delta / T_cur) > random.random():
            state = state_new
            cost_cur = cost_new
    return state
def cost_function(z):
    return sum(-w[i] * z[i] for i in range(100))
def flip(z):
    z_new = z.copy()
    i = random.randint(0, 99)
    z_new[i] = 1 - z_new[i]
    return z_new
w = np.array([np.mean(data[:, i]) for i in range(100)])
initial_state = np.zeros(100)
T = 1.0
max_iter = 1000
cool_rate = 0.99
z = simulated_annealing(cost_function, initial_state, T, max_iter, cool_rate)
for iter = 1 : maxgen   % 外循环开始
    for i = 1 : Lk   %   内循环
        way1 = gen_new_way(way0,s,b);
        money1 = calculate_money(way1,freight,M,b); % 计算新方案的花费
        if money1 < money0       % 如果新方案的花费小于当前方案的花费
            way0 = way1; % 更新当前方案为新方案
            money0 = money1;
        else
p = exp(-(money1 - money0)/T)
            if rand(1) < p
                way0 = way1;
                money0 = money1;
        % 判断是否要更新找到的最佳的解
        if money0 < min_money   % 如果当前解更好，则对其进行更新
            min_money = money0;   % 更新最小的花费
            best_way = way0;   % 更新找到的最佳方案
    MONEY(iter) = min_money; % 保存本轮外循环结束后找到的最小花费
    T = alfa*T;     % 温度下降
# 输出结果
idx = np.where(z == 1)[0][0]
print("选择的信用评分卡编号为:", idx+1)
print("对应的收入为:", w[idx])
```

(2) 问题 2

Gurobi

```
# -*- coding: utf-8 -*-
import time
import csv
import random
```

```
import gurobipy as gp
from gurobipy import GRB
import numpy as np
import copy
import math
from itertools import islice
print('程序开始时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
start = time.time()
"""基础数据"""
M = [1,2,3]
N = [1,2,3,4,5,6,7,8,9,10]
T = []
H = []
for i in range(100):
    t_l = 't_' + str(i+1)
    T.append(t_l)
    h_l = 'h_' + str(i+1)
    H.append(h_l)
tij = {}
hij = {}
data_file ='C:/Users/ASUS/Desktop/question/data.csv'
with open(data_file,'r') as f:  #读写“data”文件
    data_reader=csv.DictReader(f)
    j = 1
    for row in data_reader:
        for i in range(1,101):
            tij[(i,j)] = float(row[T[i-1]])
            hij[(i,j)] = float(row[H[i-1]])
        j = j + 1
current_obj = []
def mycallback(m, where):
    if where == GRB.Callback.MIPSOL:
        current_obj.append(m.cbGet(GRB.Callback.MIPSOL_OBJBST))
m = gp.Model('integer linear programming model')
x =m.addVars(N, M, vtype = gp.GRB.BINARY, name="x")
y = m.addVars(N, N, N, vtype = gp.GRB.BINARY, name="y")

m.update()
#-----------------------------添加选取一种约束-----------------------
# for m in M:
# xi1_sum = 0
# for i in N:
#         xi1_sum = xi1_sum + x[(i,1)]
# m.addConstr(xi1_sum == 1)

# xi2_sum = 0
# for i in N:
#         xi2_sum = xi2_sum + x[(i,2)]
# m.addConstr(xi2_sum == 1)

# xi3_sum = 0
# for i in N:
#         xi3_sum = xi3_sum + x[(i,3)]
# m.addConstr(xi3_sum == 1)
yijk_sum = 0
for i in N:
    for j in N:
        for k in N:
            yijk_sum = yijk_sum + y[(i,j,k)]
```

```
m.addConstr(yijk_sum == 1)
# for i in N:
#         for j in N:
#                 for k in N:
#                         m.addConstr(y[(i,j,k)]+2 >= x[(i,1)]+x[(j,2)]+x[(k,3)])
'''目标函数'''
total_t = {}
for i in N:
        for j in N:
                for k in N:
                        total_t[(i,j,k)] = tij[(1,i)] * tij[(2,j)] * tij[(3,k)]
total_h = {}
for i in N:
        for j in N:
                for k in N:
                        total_h[(i,j,k)] = (hij[(1,i)] + hij[(2,j)] + hij[(3,k)]) / 3
obj = 0
for i in N:
        for j in N:
                for k in N:
                        obj = obj + (1000000*(0.08*total_t[(i,j,k)]*(1-total_h[(i,j,k)])-total_t[(i,j,k)]*total_h[(i,j,k)]))*y[(i,j,k)]
m.setObjective(obj , GRB.MAXIMIZE)

# m.optimize
m.optimize(mycallback)
"""结果输出"""
print()
if m.status == GRB.OPTIMAL:
        for i in N:
                # print(x[(i,3)])
                for j in N:
                        for k in N:
                                if y[(i,j,k)].x != 0:
                                        print("信用评分卡 1 对应阈值为：",i)
                                        print("信用评分卡 2 对应阈值为：",j)
                                        print("信用评分卡 3 对应阈值为：",k)
        print('最大运营成本：', m.objVal)
        # print(current_obj)
else:
        print('No solution')
end = time.time()
# print('程序结束时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
print('程序用时:',end - start,'s')
```

PyQUBO

```
# -*- coding: utf-8 -*-
from pyqubo import Array, Placeholder, Constraint
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import time
import csv
import random
print('程序开始时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
start = time.time()
I = np.arange(1,101,1)
```

```
J = np.arange(1,11,1)
n = 10
x = Array.create('x', (n, n, n), 'BINARY')
# print(x)
xij_sum = 0.0
xij_sum += Constraint((sum(x[i, j, k] for i in range(n) for j in range(n) for k in range(n)) - 1)**2,
label="xijk{}".format(1))
T = []
H = []
for i in range(100):
    t_l = 't_' + str(i+1)
    T.append(t_l)
    h_l = 'h_' + str(i+1)
    H.append(h_l)
tij = {}
hij = {}
data_file ='C:/Users/ASUS/Desktop/1/data.csv'
with open(data_file,'r') as f:  #读写“data”文件
    data_reader=csv.DictReader(f)
    j = 0
    for row in data_reader:
        for i in range(100):
            tij[(i,j)] = float(row[T[i]])
            hij[(i,j)] = float(row[H[i]])
        j = j + 1
total_t = {}
for i in range(n):
    for j in range(n):
        for k in range(n):
            total_t[(i,j,k)] = tij[(1,i)] * tij[(2,j)] * tij[(3,k)]
total_h = {}
for i in range(n):
    for j in range(n):
        for k in range(n):
            total_h[(i,j,k)] = (hij[(1,i)] + hij[(2,j)] + hij[(3,k)]) / 3
w = 0.0
for i in range(n):
    for j in range(n):
        for k in range(n):
            w = w +
(1000000*(0.08*total_t[(i,j,k)]*(1-total_h[(i,j,k)])-total_t[(i,j,k)]*total_h[(i,j,k)]))*x[(i,j,k)]*x[(i,j,k)]
# w = 0.0
# for i in range(n):
#     for j in range(n):
#         for k in n:
#             w += -((1000000*(0.08*tij[(i,j)]*(1-hij[(i,j)])-tij[(i,j)]*hij[(i,j)])) * x[i,j] * x[i,j])

A = Placeholder("A")
H = w + A * (xij_sum)
# Compile model
model = H.compile()
# Generate QUBO
feed_dict = {'A': 100000000.0}
bqm = model.to_bqm(feed_dict=feed_dict)
import neal
sa = neal.SimulatedAnnealingSampler()
sampleset = sa.sample(bqm, num_reads=1000, num_sweeps=1000)
# Decode solution
decoded_samples = model.decode_sampleset(sampleset, feed_dict=feed_dict)
```

```
best_sample = min(decoded_samples, key=lambda x: x.energy)
num_broken = len(best_sample.constraints(only_broken=True))
print("number of broken constarint = {}".format(num_broken))
if num_broken == 0:
    print(best_sample.sample)
end = time.time()
print('程序结束时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
print('程序用时:',end - start,'s')
```

模拟退火算法求解问二：

```
import dwavebinarycsp
from dwave.system import LeapHybridSampler
T0 = 100;    % 初始温度
T = T0;
maxgen = 1000;   % 最大迭代次数
Lk = 200;   % 每个温度下的迭代次数
alfa = 0.95;   % 温度衰减系数
# 输入数据，包括评分卡 1、评分卡 2、评分卡 3 的通过率和坏账率
t1 = [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55]
h1 = [0.1, 0.12, 0.15, 0.18, 0.22, 0.27, 0.32, 0.38, 0.44, 0.5]
t2 = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
h2 = [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55]
t3 = [0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65]
h3 = [0.1, 0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19]
csp = dwavebinarycsp.ConstraintSatisfactionProblem(dwavebinarycsp.BINARY)
z1 = ['z1_'+str(i) for i in range(10)]
z2 = ['z2_'+str(i) for i in range(10)]
z3 = ['z3_'+str(i) for i in range(10)]
# 添加约束
for i in range(10):
    csp.add_constraint(lambda z1i, z2i, z3i: z1i + z2i + z3i == 1, [z1[i], z2[i], z3[i]])
    csp.add_constraint(lambda z1i, z2i, z3i: z1i + z2i + z3i >= 1, [z1[i], z2[i], z3[i]])
# 定义能量函数
H = 0
for i in range(10):
    H += -1 * (h1[i] + h2[i] + h3[i]) * (z1[i] + z2[i] + z3[i]) + (t1[i] * z1[i] + t2[i] * z2[i] + t3[i] * z3[i])
# 转化为二进制问题
bqm = dwavebinarycsp.stitch(csp, max_graph_size=10000)
# 使用 D-Wave Leap API 进行量子求解
sampler = LeapHybridSampler()
response = sampler.sample(bqm, num_reads=100)
# 处理求解结果
best_solution = response.first.sample
best_energy = response.first.energy
for var, value in best_solution.items():
if value:
print(f"信用评分卡{var//10+1}设置为第{var%10+1}项闯值")
print(f"最大总收入为{-(best_energy-bias)/2}")
```

(3) 问题三

Gurobi

```
# -*- coding: utf-8 -*-
from pyqubo import Array, Placeholder, Constraint
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import time
```

```
import csv
import random
print('程序开始时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
start = time.time()
I = 100
J = 10
x = Array.create('x', (I, I, I, J, J, J), 'BINARY')
# print(x)
xij_sum = 0.0
xij_sum += Constraint((sum(x[l, m, n, i, j, k] for l in range(I) for m in range(I) for n in range(I) for i in range(J) for j in range(J) for k in range(J)) - 1)**2, label="xlmnijk{}".format(1))
T = []
H = []
for i in range(100):
    t_l = 't_' + str(i+1)
    T.append(t_l)
    h_l = 'h_' + str(i+1)
    H.append(h_l)
tij = {}
hij = {}

data_file ='C:/Users/ASUS/Desktop/1/data.csv'
with open(data_file,'r') as f:  #读写“data”文件
    data_reader=csv.DictReader(f)
    j = 0
    for row in data_reader:
        for i in range(100):
            tij[(i,j)] = float(row[T[i]])
            hij[(i,j)] = float(row[H[i]])
        j = j + 1

total_t = {}
for l in range(I):
    for m in range(I):
        for n in range(I):
            for i in range(J):
                for j in range(J):
                    for k in range(J):
                        total_t[(l,m,n,i,j,k)] = tij[(l,i)] * tij[(m,j)] * tij[(n,k)]
total_h = {}
for l in range(I):
    for m in range(I):
        for n in range(I):
            for i in range(J):
                for j in range(J):
                    for k in range(J):
                        total_h[(l,m,n,i,j,k)] = (hij[(l,i)] + hij[(m,j)] + hij[(n,k)]) / 3
w = 0.0
for l in range(I):
    for m in range(I):
        for n in range(I):
            for i in range(J):
                for j in range(J):
                    for k in range(J):
                        w = w + (1000000*(0.08*total_t[(l,m,n,i,j,k)]*(1-total_h[(l,m,n,i,j,k)])-total_t[(l,m,n,i,j,k)]*total_h[(l,m,n,i,j,k)]))*x[(l,m,n,i,j,k)]*x[(l,m,n,i,j,k)]
```

```
# w = 0.0
# for i in range(n):
#         for j in range(n):
#                 for k in n:
#                         w += -((1000000*(0.08*tij[(i,j)]*(1-hij[(i,j)])-tij[(i,j)]*hij[(i,j)])) * x[i,j] * x[i,j])
A = Placeholder("A")
H = w + A * (xij_sum)
# Compile model
model = H.compile()
# Generate QUBO
feed_dict = {'A': 100000000.0}
bqm = model.to_bqm(feed_dict=feed_dict)
import neal
sa = neal.SimulatedAnnealingSampler()
sampleset = sa.sample(bqm, num_reads=100, num_sweeps=100)
# Decode solution
decoded_samples = model.decode_sampleset(sampleset, feed_dict=feed_dict)
best_sample = max(decoded_samples, key=lambda x: x.energy)
num_broken = len(best_sample.constraints(only_broken=True))
print("number of broken constarint = {}".format(num_broken))
if num_broken == 0:
    print(best_sample.sample)
end = time.time()
print('程序结束时间：',time.strftime("%Y-%m-%d %H:%M:%S",time.localtime(int(time.time()))))
print('程序用时:',end - start,'s')
```

模拟退火算法求解问题三：

```
import dwavebinarycsp
from dwave.system import LeapHybridSampler
T0 = 100;     % 初始温度
T = T0;
maxgen = 500000;   % 最大迭代次数
Lk = 200;   % 每个温度下的迭代次数
alfa = 0.95;   % 温度衰减系数
# 定义信用评分卡的价格和收益
data = pd.read_csv(r"C:\Users\Tan\data_100.csv)
variables = ['z1', 'z2', 'z3']
csp = dwavebinarycsp.ConstraintSatisfactionProblem(dwavebinarycsp.BINARY)
for variable in variables:
    csp.add_variable(variable, [0, 1])
for i in range(len(variables)):
    for j in range(i+1, len(variables)):
H = 0
for i in range(100):
for j in range(i+1, 100):
for k in range(j+1, 100):
H += -1 * (c1[i] + c2[j] + c3[k]) * (z1[i] + z2[j] + z3[k]) + (p1[i] * z1[i] + p2[j] * z2[j] + p3[k] * z3[k])
bqm = dwavebinarycsp.stitch(csp, max_graph_size=10000)
sampler = LeapHybridSampler()
response = sampler.sample(bqm, num_reads=100)
for sample, energy, num_occurrences, _ in response.data(['sample', 'energy', 'num_occurrences']):
z1_soln = [int(sample[f'z1_{i}']) for i in range(100)]
z2_soln = [int(sample[f'z2_{i}']) for i in range(100)]
z3_soln = [int(sample[f'z3_{i}']) for i in range(100)]
total_profit = sum([p1[i] * z1_soln[i] + p2[i] * z2_soln[i] + p3[i] * z3_soln[i] for i in range(100)])
print(f"Total profit: {total_profit}")
```