第 3 章基于基因算法的车辆任务卸载算法

3.1 引言

作为下一代智慧交通的电动智能汽车有许多的问题需要解决，其中电池问题尤为重要，这关系着汽车的行驶里程。而任务卸载是其中较有希望的一个减少能源消耗，提高能源利用率的一种方式。

在最近流行的边缘计算技术中，不仅可以将任务分配给附近的边缘服务器，还可以分配给附近资源丰富的用户。但是其中很多问题需要解决，例如，如何衡量计算资源，如何实时的分配任务，如何分配任务更加公平。

在该问题中，将任务分配给附近的车辆，来提高资源的利用率以及最小化系统的能耗。我们将移动边缘计算场景车联网中任务卸载的问题抽象为一个数学规划的形式，其中，目标是最小化所有能源消耗的平方，这可以兼顾能耗最小化与公平。

约束包括：所有的任务必须有车辆来做，所有的车辆能够在规定时间内完成任务，以及为了保证通讯的质量，参与资源共享的智能汽车的距离要在一定的距离内。

该问题是一个非线性整数规划问题，没有多项式时间范围内的解决办法。

在本文中，我们提出了基于用于电动智能汽车的遗传算法。我们评估通过仿真实验验证了算法的性能实验。实验结果表明，我们的算法可以达到节能的目的。在未来的研究中，我们计划建立在数据大小和指令数量之间函数关系并考虑最后期限限制，同时，也将考虑卸载过程的真实性。

3.2 系统模型

本节主要介绍了系统模型，分为两个部分，系统场景，车辆计算能力模型，以及车辆能耗模型。

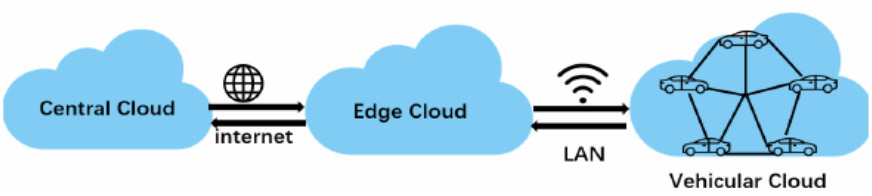
3.2.1 系统场景

在本文中，如图 4 所示，层次结构由车辆云、边缘云和中心云组成。中心云负责全局调度，它执行的任务包括执行复杂的计算任务和进行全局的决策。边缘云是车辆云的控制器，负责创建、维护和删除车辆云。车辆云由一定范围内的、参与共享其计算资源的智能车辆组成。基站、边缘服务器和远程的中心云服器通过有线来连接，基站和边缘服务器物理上距离十分接近，因此可以看作一体，后文中以边缘云来称呼这两者的总和。

每辆车都可以访问边缘云并与其进行通讯。车辆可以通过将任务卸载到其他车辆的方式来节省能源。这样可以提高整体资源利用率。

任务卸载过程如下：首先，实时流量信息被传输到云服务器进行分析。实时信息包括目的地、当前位置、时间、车速等。云服务器在汇总数据后，进行大数据分析以获取道路拥堵情况，然后推断未来某个时间段内车辆的位置信息，并将结果返回到车辆附近的边缘云。

这些车辆在未来一段时间内将会参与资源共享。然后，边缘云根据这些信息分配任务，并将车辆分为提供者和请求者，他们在 T 时间段共享资源。



图：系统场景层次结构

3.2.2 车辆计算能力模型

如何定义车辆的计算资源是建立起问题描述非常重要的一步，在此，我使用计算每秒执行的指令条数来刻画计算资源，我们将资源共享时间定义为 $\mathcal{T}=\{1,\ldots,T\}$.。车辆数量定义为 N。

我们使用元组 $J_{it}=\{r_{it},r'_{it},d_{it}\}$ 来表示时间为 t 时，车辆 i 的任务大小，d_{it} 是任务的数据大小。

任务分为必须在本地执行的任务，以及可以分配出去的任务。这一点并不抽象， 例如有些涉及隐私的任务，或者是要求实时性的任务，就必须在本地执行。

在时间 t 时，r_{it} 是车辆 i 必须要在本地执行的任务，r'_{it} 是车辆 i 能够分配给别人的任务，并且，它们的值是由需要的指令条数来表示的。

为了更好的描述这个模型，我们定义 $\mathbf{X}_t=\left\{x_{ij}\right\}\in\{0,1\}^{N\times N}$ 为分配矩阵，如果 $x_{ij}=1$，代表车辆 i 执行车辆 j 所卸载的任务。注意，如果 $x_{ii}=1$，车辆 i 所有的任务都在本地执行。

我们将车辆 i 的容量定义为 C_{it} 在时间为 t 时。在时间 t，当车辆 i 被选为提供者（P），所有需求者需要的资源需要少于这辆车的容量。公式化表达如下：

$$\sum_{j=1}^{N}x_{ij}^{t}\cdot r'_{jt}\le C_{it},i=1,\ldots,N \tag{3.1}$$

一个关键的步骤是怎样衡量车辆的计算资源，在本篇文章中，计算资源定义在每秒能够执行的指令条数上。

公式化表达如下：

$$C_{it}=M_{it}\cdot\Delta T-r_{it} \tag{3.2}$$

其中，ΔT 是资源共享的持续时间，并且它是一个比建立车辆边缘网络的更小的时间粒度，在 ΔT 秒后，分配矩阵会发生变化。

对于一个单核的 CPU，每秒能够执行的指令条数（m_{it}） 和 CPU 的频率（f_{it}）有如下的关系:

$$M_{it} = v_i \cdot f_{it} + \theta_i \tag{3.3}$$

其中，v_i和θ_i是待估计的参数。

最后，公式（2）中的CPU的容量C_{it}在能被下面的公式计算出来：

$$C_{it} = (v_i \cdot f_{it} + \theta_i) \times \Delta T - r_{it} \tag{3.4}$$

3.2.3 车辆能耗模型

当我们考虑车辆的能源消耗时，我们将它分为两个部分（1）计算所需要的能量以及（2）传输所需要的能量。

根据[9][10][11][12]，计算所需要的能耗能够被下列公式计算：

$$E = \lambda_i \cdot f_{it}^3 \cdot \Delta T \tag{3.}$$

其中f_{it}是CPU在t时刻的频率。如果一辆车被选为了需求者（R），这辆车的频率会下降$f_{it}^{'}$。因为任务被分配出去了，所以他消耗的能量会减少。消耗的能量能够被以下公式计算：

$$E_i^{save} = (\lambda_i \cdot f_{it}^3 - \lambda_i \cdot f_i^{'3}) \times \Delta T, \forall i \in R \tag{3.}$$

传输能耗和传输的时间有线性关系，传输的时间取决于数据大小和传输速率的比值(b_{ij})：

$$E = P_0 \cdot \frac{d_{it}}{b_{ij}} \tag{3.}$$

其中，P_0是传输率。

最大的传输速率（b）可以通过香农公式计算：

$$b_{ij} = W \log(1 + \mathrm{SNR}) \tag{3.}$$

其中，SNR是信噪比，W是频道的带宽。因为它和每一个场景相关，我们将它考

虑为一个常数。

对于每一辆车，接收信息所消耗的能量是：

$$E_i^{rec} = \sum_{j=1, j\neq i}^{N} x_{ij}^t \cdot P_0 \cdot \frac{d_{jt}}{W \log(1+\mathrm{SNR})}, \quad i = 1, \ldots, N \tag{3.9}$$

对于每一辆车，发送信息所需要的能量是：

$$E_i^{send} = \sum_{i=1, i\neq j}^{N} x_{ij}^t \cdot P_0 \cdot \frac{d_{it}}{W \log(1+SNR)}, \quad j = 1, \ldots, N \tag{3.}$$

定义 E_t^{blnc} 是车辆 i 在时刻 t 所消耗的能量总和：可以被这么计算

$$E_{it}^{blnc} = \sum_{j=1, j\neq i}^{N} x_{ij}^t \cdot P_0 \cdot \frac{d_{jt}}{W \log(1+\mathrm{SNR})} + \sum_{j=1,, j\neq i}^{N} x_{ji}^t \cdot P_0 \cdot \frac{d_{it}}{W \log(1+\mathrm{SNR})} + \lambda_i \cdot f_{it}^3 \cdot \Delta T, \quad i = 1, \ldots, N$$

(1)

我们算法的设计目标是:最小化所有车辆能量消耗的平方。平方和和普通的和相比，平方和更容易受到极端值的影响，因为平方项会使得较大或较小的值对平方和的贡献更加显著。

这个目标既能够考虑最小化能量消耗，又能考虑公平，也就是说平衡了能量消耗和公平。公式表示如下文：

$$\min \sum_{t=1}^{T} \sum_{i=1}^{N} (E_{it}^{blnc})^2$$

3.3 问题建模

在本节中，我们将问题分为两种情况来建模，一种是任务可分情况，另一种是任务不可分情况。当任务可分的情况下，问题能够直接通过规划的方式解决；

当任务不可分时，不能通过规划的方式来解决。

3.3.1 任务可分情况

当车辆在进行某些同质任务时，可以将大任务任意的拆成一些小任务，因为任务的粒度太小，因此可以近似看成任务可以随意分配，也就是任务可分的情况。

在任务可分时，根据本地留存的高优先级任务的大小又分为：高于平均任务大小，和低于平均任务大小。

当本地任务低于平均任务大小（小任务）时，该问题较为简单，直接将所有任务平均分配到车辆上就能解决。

该问题等价于：首先将固定值 C 切制成 n 个数，是的他们的三次方之和最小。设切分的数分别为 $a_1, a_2, \cdots, a_n$，根据均值不等式：

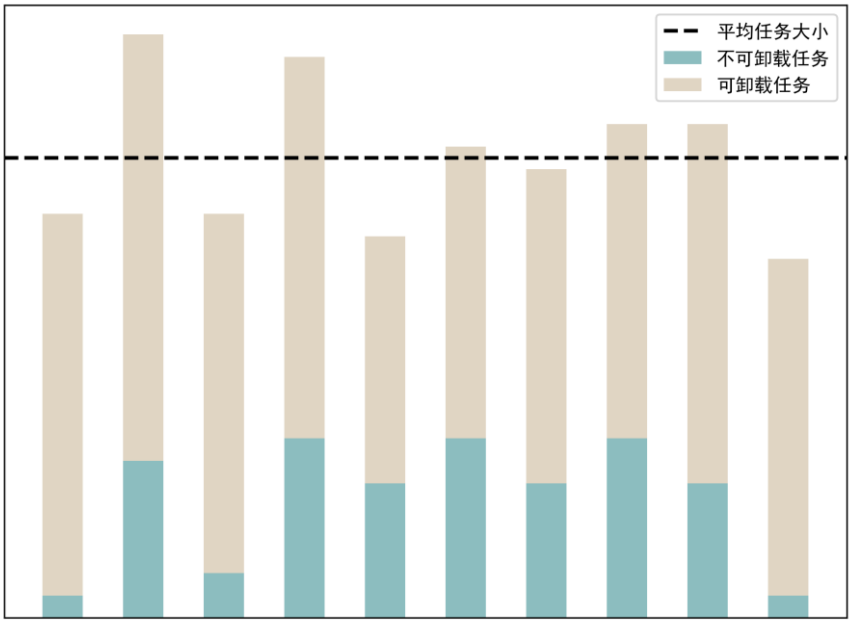
$$\sqrt[3]{\frac{\sum_{i=1}^{n} a_i^3}{n}} \geq \frac{\sum_{i=1}^{n} a_i}{n}$$

即

$$\sum_{i=1}^{n} a_i^3 \geq \frac{n(\sum_{i=1}^{n} a_i)^3}{n^3} = \frac{C^3}{n^3}$$

因此，要使 $\sum_{i=1}^{n} a_i^3$ 最小，需要让 $\sum_{i=1}^{n} a_i$ 的值尽可能平均分配给 n 个数，即令 $a_1 = a_2 = \cdots = a_n = \frac{C}{n}$。

当本地任务是小任务时，能搞保证参与资源共享的车辆，任务的大小可以卸载为平均值。



图：

但是当任务高于平均任务大小（大任务）时，该问题需要进行分类的讨论，是否还需要将再给有大任务的汽车分配任务。

该问题等价于：有一个固定值 C，将其切割为 n 个数，要求这 n 个数的三次方最小，知道有 m 个数必须大于 $\frac{C}{n}$，如何切割。

在有限制条件的情况下，该问题可以使用拉格朗日乘数法来求解。设切分后的数为 $a_1, a_2, \cdots, a_n$，且有 m 个数大于 $\frac{C}{n}$。

有以下约束条件：

$$\begin{cases} \sum_{i=1}^{n} a_i = C, \\ \sum_{i=1}^{n} \text{count}(a_i, \frac{C}{n}) = m, \end{cases}$$

其中，count 表示指示函数，当 $a_i > \frac{C}{n}$ 时，取值为 1，否则为 0。

目标为：

$$\min \sum_{i=1}^{n} a_i^3 \tag{3.}$$

根据拉格朗日乘数法，构造带拉格朗日乘数 $\lambda_1, \lambda_2, \cdots, \lambda_n$ 的拉格朗日函数：

$$L\left(\{a_i\}_{i=1}^{n}, \{\lambda_i\}_{i=1}^{n}\right) = \sum_{i=1}^{n} a_i^3 + \sum_{i=1}^{n} \lambda_i \left(a_i - \frac{C}{n}\right) + \mu \sum_{i=1}^{n} \text{count}\left(a_i, \frac{C}{n}\right) \tag{3.}$$

其中，μ 是 Lagrange 乘子。

对 L 求导并令其等于 0，可得：

$$3a_i^2 + \lambda_i - \mu \text{count}\left(a_i, \frac{C}{n}\right) = 0$$

将上式分为两种情况：

（1） 当 $a_i \le \frac{C}{n}$ 时，μ=0，则有 $a_i = -\frac{1}{3}\lambda_i$。

（2） 当 $a_i > \frac{C}{n}$ 时，则有 $\lambda_i = -3a_i^2$， μ 取任意值。

将 a_i 代入约束条件中可得：

$$m = \sum_{i=1}^{n} \text{count}(a_i, \frac{C}{n}) = \sum_{i=1}^{n} \text{count}(\lambda_i, 3(\frac{C}{n})^2) \tag{3.}$$

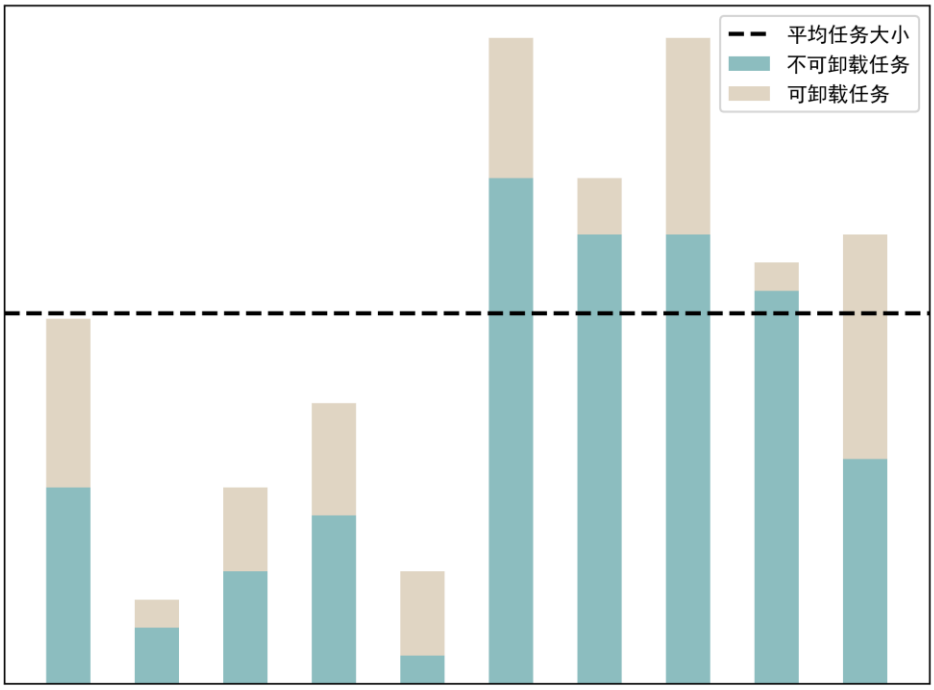
即有 m 个 λ_i 满足 $\lambda_i > 3(\frac{C}{n})^2$。假设它们的下标 $i_1, i_2, \cdots, i_m$，则有：

$$\sum_{j=1, j\neq i_k}^{n} a_j = C - \frac{\sum_{k=1}^{m} \lambda_{i_k}}{n}, \quad k = 1, 2, \cdots, m \tag{3.}$$

进一步代入目标函数 $\sum_{i=1}^{n} a_i^3$ 中，可以得到：

$$\sum_{i=1}^{n} a_i^3 = (\frac{C - \sum_{k=1}^{m} \lambda_{i_k}}{n})^3 \cdot (n-m) + \sum_{k=1}^{m} \lambda_{i_k}^3 \tag{3.}$$

要求 $\sum_{i=1}^{n} a_i^3$ 最小，就要使 $\sum_{k=1}^{m} \lambda_{i_k}^3$ 最小。该问题是一个无约束优化问题，可以使用导数来求解。可得，最优的分配方案是，将那几个本地任务是大任务的车辆将他们所有可分得任务分出去，然后再均分剩下的任务。



图：

3.3.2 任务不可分情况

在任务不可分时，这个问题不可能在多项式时间内解决，因此，在下一章中，本文打算通过智能算法来获得一个近似解，来满足靠近最优解和卸载方案计算时间的平衡。

本文的优化目标是通过调度各个车辆的任务，最小化所有能耗的平方，该问题最终被建模为如下形式：

$$\min \quad \sum_{t=1}^{T}\sum_{i=1}^{N}(E_{it}^{blnc})^2 \tag{2}$$

$$\text{s.t.} \quad \sum_{j=1,j\neq i}^{N} x_{ij}^{t}\cdot P_0\cdot\frac{d_{jt}}{W\log(1+\mathrm{SNR})} + \sum_{j=1,j\neq i}^{N} x_{ji}^{t}\cdot P_0\cdot\frac{d_{it}}{W\log(1+\mathrm{SNR})} + \lambda_i\cdot f_{it}^{3}\cdot\Delta T,\quad i=1,\ldots,N \tag{3}$$

$$\sum_{j=1}^{N} x_{ij}^{t}\cdot r'_{jt}\le C_{it} \tag{4}$$

$$\sum_{i=1}^{N} x_{ij}^{t}\ge 1 \tag{5}$$

$$f_{it}\ge 0 \tag{6}$$

$$x_{it}^{t}\in\{0,1\} \tag{7}$$

正如前面所提到的，公式（14）说明了对于每一辆车辆 i，需求的总和不能超过他的容量。公式（15）说明了车辆 i 的任务，必须要被执行，无论是本地执行还是分配给其他车辆。

3.4 基于基因算法的问题求解

因为[公式]这是一个 NP 难（NP-Hard） 问题，多项式时间内求解不出该问题的精确解，直接计算法求解是不现实的，因此采用了基因算法求解该问题

基因算法是一种典型的启发式智能优化算法，其鲁棒性来源主要有以下三点：首先来源于个体多样性，通过引进交叉、变异等操作，使得不同个体之间进行信

息的交流，有助于保持种群的多样性，从而不易陷入问题的局部最优解。其次来源于选择策略。基因算法中的选择策略来自于达尔文自然选择学说中的“适者生存”原则，通过对适应度高的个体进行选择和繁殖，逐步提高整个种群的适应性，这提高了算法的鲁棒性。最后来源于参数设置。基因算法中的参数设置，例如交叉的概率、变异的概率，这对算法的表现具有较大影响。通过对参数的合理设置，可以提高算法的鲁棒性，并且使其更容易达到全局最优解。因此本文选择了能适用于离散问题的基因算法来求解该问题。

以下是基因算法的实现步骤：

(1) 使用初始化参数来初始化种群：设置种群的大小，每个个体的基因编码方式和以及初值范围，设置循环最大次数为 $it_{\max}$，然后根据这些参数生成初始种群。同时设置目标函数 $f(x)$ 以及约束函数族 $G(x)$。

(2) 评估适应度：将每个个体的基因编码字符串解码成相应的解，然后计算其适应度值。适应度值是个体在解空间中执行任务的优劣程度的量化指标。

基因算法二进制解码公式如下：

$$\delta = \frac{U_x - L_x}{2^l - 1} \tag{3.}$$

$$x = L_x + \delta \cdot \sum_{i=1}^{l} A_i 2^{i-1}$$

其中，δ 是偏移量，l 是二进制码的长度，A_i 是第 i 位二进制位的值，x 是解码后的值。例如，给定 $\mathrm{U_x} = 5$，$\mathrm{L_x} = -5$ 以及 $l = 10$，那么 $\delta = \frac{5-(-5)}{2^{10}-1} \approx 0.009775$。

基因算法二进制编码二进制长度公式如下：

$$l = \left\lceil \log_2 \left(\frac{U_x - L_x}{\mathrm{SearchAccuracy}} \right) \right\rceil \tag{3.}$$

其中，$\mathrm{U_x}$ 和 $\mathrm{L_x}$ 分别是解的最大值和最小值，SearchrAccuracy 是求解的精度。例如，给定 $\mathrm{U_x} = 5$ 和 $\mathrm{L_x} = -5$，精度是 0.01，那么 $\mathrm{l} = \left\lceil \log_2 \left(\frac{5-(-5)}{0.01} \right) \right\rceil = 10$

适应度计算公式如下：

$$fit_i = e^{-\frac{y_i}{mean(y)}} \tag{3.}$$

其中，$y_i = f(x_i)$，$mean(y)$是种群中所有的个体的平均值。

(3) 选择操作：按照轮盘赌选择的规则，从种群中选择一部分较好的个体进行进化操作，以期望得到更加优秀的后代个体。在进行轮盘赌选择时，标准化处理的公式为：

$$p_i = \frac{fit_i}{\sum_{i=1}^{n} fit_i} \tag{3.}$$

其中，n是种群中的个体数量。

(4) 进行遗传操作：包括交叉操作和变异操作。交叉操作是将两个个体的染色体进行配对，然后按照一定的概率产生新的后代染色体；变异操作是随机改变个体染色体的一个或多个基因，以增加种群的多样性。

(5) 更新种群：这是“适者生存”法则在基因算法中的应用。将新生成的后代个体插入到种群中，也就是按照一定的比例，某些适应度高的个体替换掉适应度低的个体。

(6) 判断遗传算法是否满足停止准则：停止的准则可以设置最大迭代次数或者在种群中出现一定数量的相似个体等条件，决定是否终止遗传算法。如果不满足条件则跳转到第（2）步。

3.5 仿真实验

3.5.1 实验环境

本章实验环境为：处理器为：Intel(R) Core(TM) i7-7700HQ；显卡的型号为：NVIDIA GeForce RTX 1050；机带 RAM 为 8GB；操作系统为 Windows10 专业版；Python 版本为 3.8，Numpy 版本为 1.19.2，matplotlib 版本为 3.3.2。

3.5.2 仿真参数设计

表 1：仿真参数

参数名称	值/分布	参数名称	值/分布
f_i	[700, 1900]	ΔT	1
v_i	7.683	r_{it}	U[820, 10000]
θ	-4558.52	r'_{it}	U[200, r_{it}]
λ	0.00125	P_0	0.2
W	10 MHz	SNR	677

本文将变异概率设置为 0.01，参数用于控制每个个体发生变异的概率。该概率越高，个体发生变异的可能性就越大。一般情况下，变异概率会设置为较小的值，以保持种群的多样性，并防止算法陷入局部最优解。

本文选择了 7 种种类的参与资源共享的车辆数目，分别是{10，15，20，25，30，35，40}。在算法中，种群数量被设置为 40。而当车辆数量为 35 和 40 时，种群数量被设置为 150。因为当车辆数量为 35 和 40 时，种群的规模太小，算法不会收敛或者不能很好的收敛到最优值附近。

3.5.3 评价标准

我们设置了个指标来衡量我们的实验，分别是，性能指标，公平指标，种群离散程度。

在实验中，我们的性能指标是指节能的百分比。节能的百分比，其定义如下：

$$P = \min\left(100 \cdot \left(1 - \frac{\sum_{i=1}^{N} E_{it}^{blnc}}{E_{loc}}\right)\right), t = 1, \ldots, T \tag{3.}$$

其中，E_{loc} 是本地执行任务所需要消耗的能量，

为了反映我们任务分配的公平性，我们在标准差的基础上定义了公平系数（FC）。与标准差一样，数值越小，公平性越高。标准差定义如下：

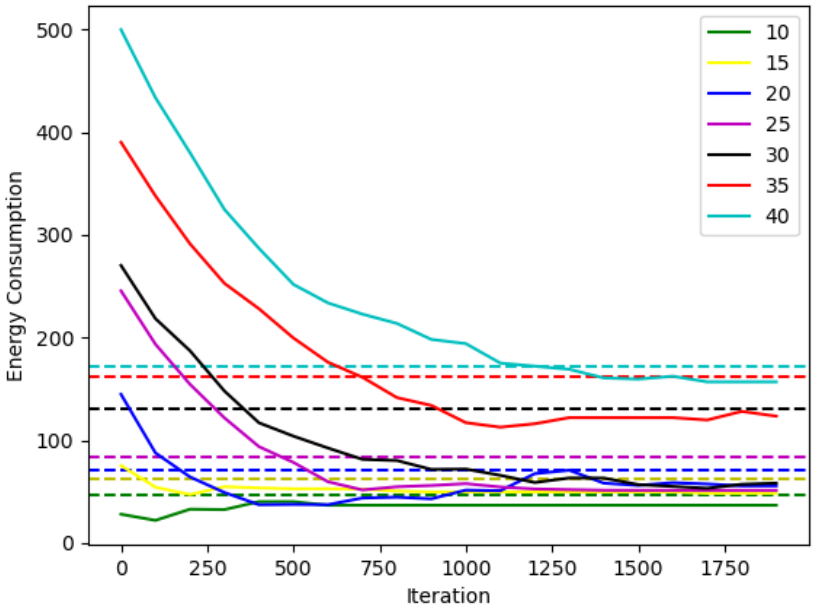
$$\sigma=\sqrt{\frac{\sum_{i=1}^{n}\left(x_i-\mu\right)^2}{n}}$$

FC 的定义如下。

$$FC=\max\frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(E_{it}^{\text{blnc}}-\bar{E}\right)^2}}{\bar{E}},t=1,\ldots,T$$

其中，t 是迭代的次数，$\bar{E}$ 是所有车辆能耗的平均值。

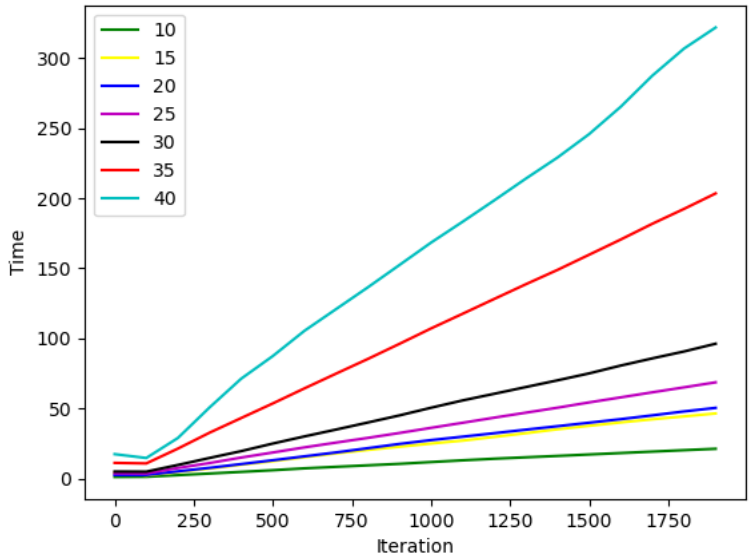
3.5.4 实验结果



在图 2 中，虚线是任务没有负载出去而在本地执行所需要消耗的能量。我们可以看到，当车辆数为 10 时，该算法迭代几十轮就可以得到很好的结果。当车辆数为 20 时，需要 350 次迭代才能得到一个较好的结果。而当车辆数量为 30 时，能耗在 750 次迭代之前变化较大，而在 750 次迭代之后变化就会明显变小。也就

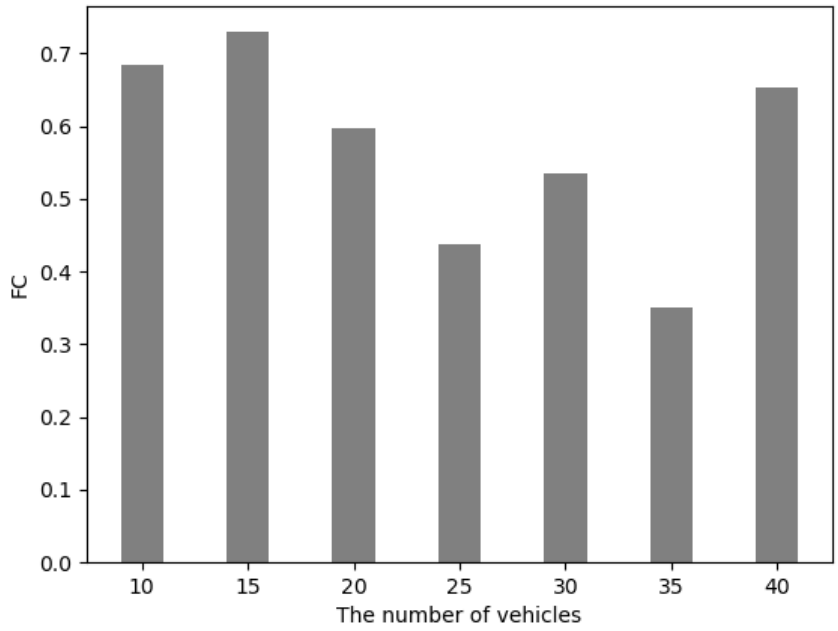
是说，能源消耗的减少率在 750 次迭代前较快，而在 750 次迭代后，能源消耗的减少率就会降低。当车辆数为 40 时，几乎没有节能效果。

我们可以得出结论，随着车辆数量的增加，迭代的次数也在增加。因此，当我们在边缘服务器上运行该算法时，我们需要合理地调整迭代次数，因为车辆数量的变化会显著的影响算法的运行时间。



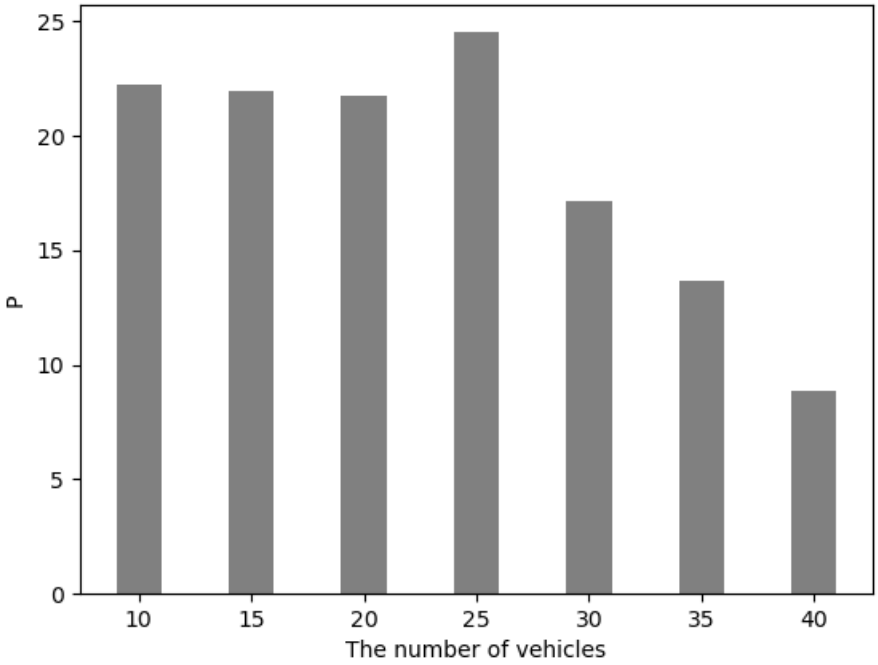
图：

在图 3 中，经过 100 次迭代以后，所有数目运行时间和迭代次数呈线性关系。然而，当迭代次数固定时，算法的运行时间和车辆之间的关系不是线性的，而是非线性的，参与资源共享的车辆的数越多，运行时间增加的越多。



图：平衡因子 FC 的值

如图 4(a)所示，随着车辆的增加，车辆能量消耗的平衡首先增加，然后减少。当车辆数目为 30 和 40 时，能耗的平衡系数反而增加。



性能度量 P 的值

如图 4(b)所示，当车辆数量在 10 到 25 辆之间时，我们的系统可以节省 20%以上的能量消耗，但是当车辆数量为 30 辆时，只有 17%的能量消耗。但当车辆数为 30 时，只能节省 17%，而当车辆数为 40 时，几乎可以节省 20%。当车辆数量为 40 辆时，几乎没有节省能量。

综合考虑性能测量、算法运行时间以及平衡因子，选择 25 辆汽车参与资源共享是非常合适的。

3.6 本章小结

本章所研究的是车联网领域的任务卸载和分配的方案，通过能耗优先的原则设计出一套卸载算法。首先，本章详细介绍了车联网中任务卸载的系统模型，并对其中的要素进行了说明，包括系统场景、车辆计算模型及车辆能耗模型。接着，作者将问题建模为数学规划形式，提出了一种基于粒子群算法的卸载与分配策略，然后设计了相关评价标准，并对实验结果进行了分析。评价标准表明，该算法能够很好地提高资源利用率，同时还兼顾了公平性。

然而，在研究过程中也发现了一些缺陷，比如，基因算法存在着编码和解码的过程，需要进行交叉和变异操作，这导致了计算量太大和计算时间过长等问题。因此，在下一步的研究中，我们将引入新的智能算法，并将其改进为适用于离散问题的算法，以加快算法的运行时间。

本章研究了一种车联网环境下能耗优先的计算卸载和分配的方案。本章首先介绍了车联网中任务卸载的系统模型，包括，系统场景，车辆计算模型、车辆能耗模型，然后将问题建模为了数学规划形式，提出了一种基于基因算法的任务卸载分配算法，然后设计了相关评价标准及分析了实验结果。评价标准显示，该算

法能够很好的提高资源的利用率，同时也很好的兼顾公平。

但在本章的研究过程中发现了一些不足：基因算法存在着编码和解码的过程，以便于进行交叉和变异操作，这带来了计算量太大以及计算时间过长的问题。因此，下一张会引出较新的智能算法并将其修改为能解决离散问题的算法，加快算法的运行时间。

第 4 章基于贪心法调整的离散侏儒猫鼬算法的任务卸载算法

通过上一章节的研究，我们完成了车联网中对任务卸载算法研究的任务，但也指出了存在计算量太大以及计算时间过长等问题。因此本章将通过使用新的智能算法并将其修改为能解决离散问题的算法来解决该问题。

4.1 侏儒猫鼬算法

和粒子群、基因算法类似，侏儒猫鼬算法（Dwarf Mongoose Optimization Algorithm，DMOA）也是一种基于动物行为的启发式优化算法，能够用来解决线性以及非线性的优化问题。该算法是由 Jeffrey O. Agushaka 等科学家于 2022 年提出的，算法的灵感源于侏儒猫鼬的捕猎行为，通过模拟侏儒猫鼬在觅食和捕猎时的行为特征来进行对问题的求解。

侏儒猫鼬算法在搜索过程中，不断尝试各种不同的位置和状态组合，计算适

应度，并据此进行调整，最终找到最优解或次优解。

像其他智能算法一样，侏儒猫鼬算法借鉴了侏儒猫鼬在觅食和捕猎时的行为特征，同时也吸收了其他算法的特征。

表 4.2 肺部 CT 图像数据集

算法名称	主要思想
粒子群算法	
训练集	1600
验证集	400
测试集	500

侏儒猫鼬是一种小型哺乳动物，以体型较小昆虫以及蚂蚁，蚁蛇等为食，它们具有机敏的行动能力和快速适应环境的能力。

侏儒猫鼬通常生活在在一个母权制的家庭群体中，由一对阿尔法夫妻（leader）领导种群生活在一起。无论在每个年龄组中，雌性都比雄性地位高，而幼崽比它们的哥哥姐姐地位高。这些群体中的劳动分工和利他主义是哺乳动物中最高的，他们根据年龄和性别，不同的猫鼬充当警卫、保姆、攻击捕食者和攻击同种入侵者。

为了模仿它们的觅食行为，优化过程建立了三个社会结构：阿尔法小组，侦察兵小组，保姆小组。

4.1.1 阿尔法小组

阿尔法小组是又进行出发去觅食（探索新的解）的行为的种群，食物的位置由下面的公式给出：

$$X_i^{t+1} = X_i^t + \text{phi} \times \text{peep} \times (X_i^t - X_{peep})$$

其中，X_i^{t+1} 是新产生的解，phi 服从于在[-1,1]之间的均匀分布。peep 是雌性首领的位置数量，X_{peep} 是选出的雌性首领的位置。

雌性首领在阿尔法小组中产生，每个个体都有可能成为雌性首领，适应度高的个体成为雌性首领的可能性更大，适应度差的个体成为雌性首领的可能性小，每个个体成为雌性首领的概率计算方式如下：

$$\alpha = \frac{fit_i}{\sum_{i=1}^{n} fit_i}$$

其中， fit_i 是第 i 个个体的适应度，n 是种群数量。

4.1.2 保姆小组

保姆通常是附属的群体成员，和幼仔呆在一起，定期轮换，让母亲带领其他成员进行日常觅食。她通常在中午和晚上回来给幼崽喂奶。也就是说，保姆小组的成员不进行觅食行为，同时，保姆的数量取决于种群规模，种群越大，保姆数量越多。

在算法中，当一个个体很久没有进行觅食（更新）时，就说明不是陷入了局部最优，就是这个解太差了，更新的可能比较小，将其放入保姆小组中，并重新进行初始化。

4.1.3 侦察兵小组

侦察兵寻找下一个睡觉的土堆（睡眠丘），因为猫鼬不会回到先前睡觉的土堆，这保证了探险的有效性。在该算法中，侦察兵小组和阿尔法小组是一个小组，首先会进行阿尔法小组的跟新行为，然后再进行侦察兵小组的更新行为。

睡眠丘的计算公式如下：

$$sm_i = \frac{fit_{i+1} - fit_i}{\max\left\{\left| fit_{i+1}, fit_i \right|\right\}}$$

睡眠丘的平均值反映了该轮迭代的移动范围，φ 计算公式如下：

$$\varphi = \frac{\sum_{i=1}^{n} sm_i}{n}$$

CF 是表示侦察兵小组移动能力的参数：

$$CF = \left(1 - \frac{iter}{T}\right)^{\left(2\frac{iter}{T}\right)}$$

其中，*iter* 为当前迭代次数，*T* 为最大迭代次数，当迭代次数增加时，其值会减小，也就是说，随着迭代次数增加，侦察兵小组移动能力会减小。

侦察兵小组的更新方式如下：

$$X_{i+1} = \begin{cases} X_i - CF * \text{ phi } * \text{ rand } * \left[X_i - \overrightarrow{\mathbf{M}}\right] & \text{if } \varphi_{i+1} > \varphi_i \\ X_i + CF * \text{ phi } * \text{ rand} * \left[X_i - \overrightarrow{\mathbf{M}}\right] & \text{else} \end{cases}$$

其中，phi 服从于[0,1]之间的均匀分布，$\overrightarrow{\mathbf{M}} = \sum_{i=1}^{n} \frac{X_i \times sm_i}{X_i}$

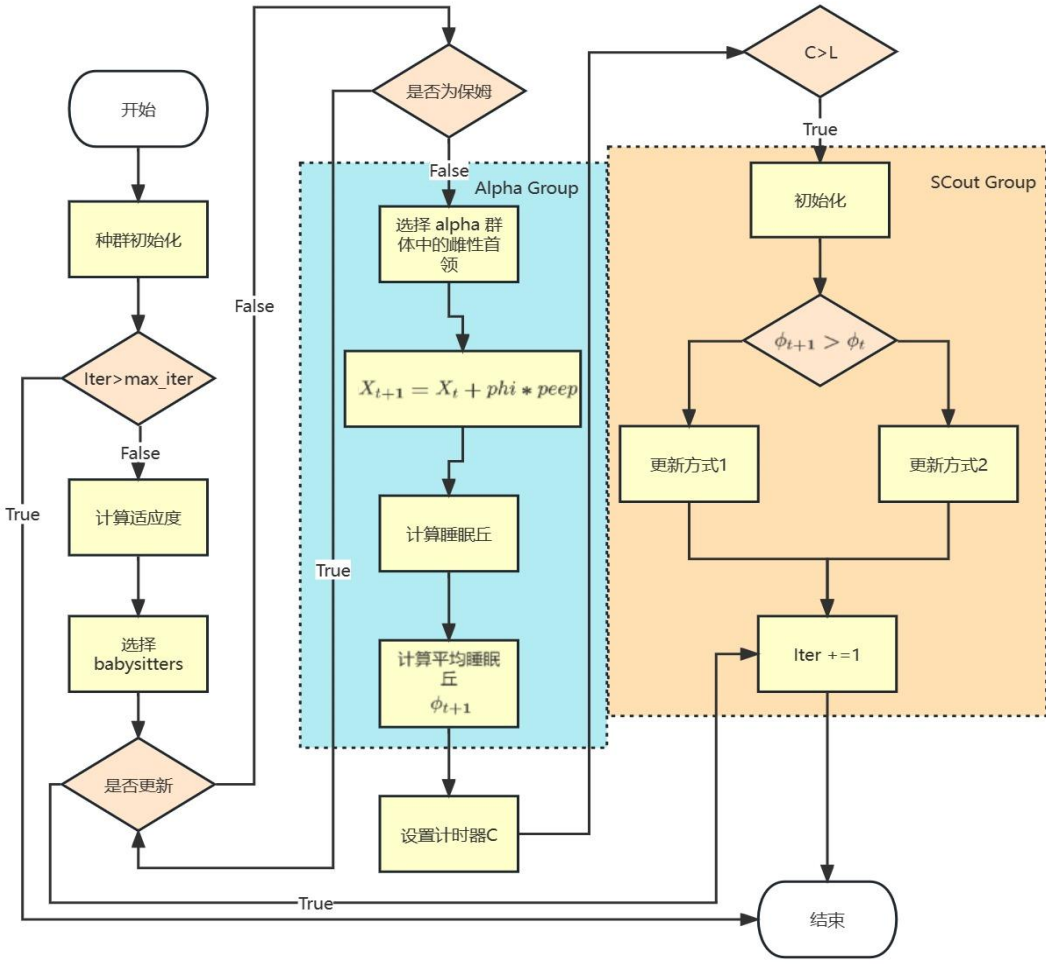
4.1.4 算法的流程

优化从阿尔法小组出发(探索空间)觅食开始，将保姆小组留在巢穴中。一旦找到觅食地点，阿尔法小组就一直觅食到中午（保留更新值）。

当他们返回交换保姆时，按照保姆交换标准，很久没觅食的成员会成为保姆。一旦保姆被交换，他们就不会回到先前觅食的地方，以避免过度放牧。

侦察兵会发现一个新的觅食地点，并通知雌性首领，带领家族到达新的地点。

可以看出，侏儒猫鼬算法中种群的主体部分（阿尔法小组或者侦察兵小组）一次迭代中会进行两次更新行为。



图：

显而易见，侏儒猫鼬算法在进行每一步的更新时，使用了类似于梯度的更新方式：$X_i^{t+1} = X_i^t + \text{phi} \times \text{peep} \times (X_i^t - X_{peep})$，因此该算法只能解决连续形式的问题，不能解决离散形式的问题，需要我们将其改造为像基因算法一样的离散型算法。

4.2 离散算法相关的知识

本节介绍了想要将算法修改为离散算法的相关知识。

4.2.1 优化问题的描述形式

我们的问题和并行机器调度问题（parallel machines scheduling problem）有同质性，并行机器调度问题是指在一些并行处理的机器上调度执行一组作业的问题。该问题与日常生产生活紧密联系，通常出现在工业车间生产、物流快递配送、计算机任务分配等领域，可以帮助企业有效地安排生产计划、降低成本、提高效率，也可以在计算机系统中动态地调度任务，提高系统总体性能。

该问题的形式化描述如下：有一个工件（任务）集合 $J=\{J_1, J_2,\ldots J_n\}$，其中，任务共有 n 个。还有一个机器的集合即 $M=\{M_1, M_2,\ldots,M_m\}$，共有 m 个机器，所有的机器都是一样的，并且每台机器一次只能处理一个任务。每一项任务应该必须在其中一台机器上进行，在所有机器上运行的时间不会改变。任何一台机器处理任务 i 所需的时间由 p_i 给出。调度中分配给机器 M_j 的作业的子集用 S_j 表示。作业一旦开始处理，就必须在不中断作业的情况下完成，即，任务一旦开始做那就必须做完。

此外，作业之间没有优先关系，也就是说，在任何时间，作业可以不受限制地随意分配到任意一台机器上。研究了以最小化最后一个工件离开系统的时间为目标的最优工件分配问题，即最大完工时间准则。

最小最大完工时间的混合整数规划公式如下:

$$\min \max_{1\le j\le m} \sum_{i=1}^{n} p_i x_{ij}$$

$$\text{s.t.} \sum_{j=1}^{m} x_{ij} = 1 \quad i = 1,\ldots,n$$

$$x_{ij} \in \{0,1\}$$

公式~(ref{pmpmin1})表明，我们应该耗时最久的那台机器尽可能地小，因为所有的作业完成才算是完成了任务。公式~(ref{pmpst1})表明每个作业必须分配

到一台机器上。如果 $x_{ij}=1$，意味着作业 i 被分配到 j 的机器上。

然而，工件问题这个形式的空间复杂度是 $O(mn)$，而且因为每个变量都是二进制变量，所以时间复杂度是 $O(2^{mn})$。我们将并行机器的调度表述为另一种形式：

$$\min \max_{1\le j\le m} \sum_{1\le i\le n, x_i=j} p_i$$

$$\text{s.t.} \quad x_i \in \{0,1,\ldots,m\}, \quad i=1,\ldots,n$$

其中，工件问题该形式的空间复杂度为 $O(n)$，时间复杂度为 $O(m^n)$。

工件问题每个变量之间的取值有相关性，如，一个两个变量的值交换，很大的概率就会导致最优解变成次优解甚至是劣解。

后文中，称呼每一个 x_i 为变量，所有变量的解集合即 $X=(x_1,x_2,...x_n)$ 称为该问题的一个解。

在后文设计离散形式的算法时，解的表示是一个关键的步骤，解具有与问题域相关的必要的信息。例如上文中使用基因算法解决问题，基因算法为了表示解，将解重新进行了编码，用二进制字符串的形式表示解。为了建立算法和问题之间的联系，对于有 n 个任务的问题，对应的解的维度也是 n。因此，在后文进行离散算法表示时，并行机器调度问题的解是用一个长度等于作业数量的数组表示的。

图：解的表示

如图所示，一共有六个任务，第 1，2，5 号任务放在第一台机器上处理，第 3，4，6 号任务放在编号为 2 的机器上进行处理

4.2.2 各变量之间的相关性

本节通过两个实验来验证各个变量之间的相关性。第一个实验是验证解中某

些变量的取值发生改变对目标函数的影响，第二个实验是验证两个解进行交叉行为对目标函数的影响。第二个实验实际是第一个实验的更进阶的版本，第一个实验交换的是同一个解中的某些变量，第二个实验交换的是不同解中的连续的变量。

这两个实验都说明了解变量之间的相关性，为了消除变量之间的相关性，我们生成了解以后，还需要对解的变量进行调整。

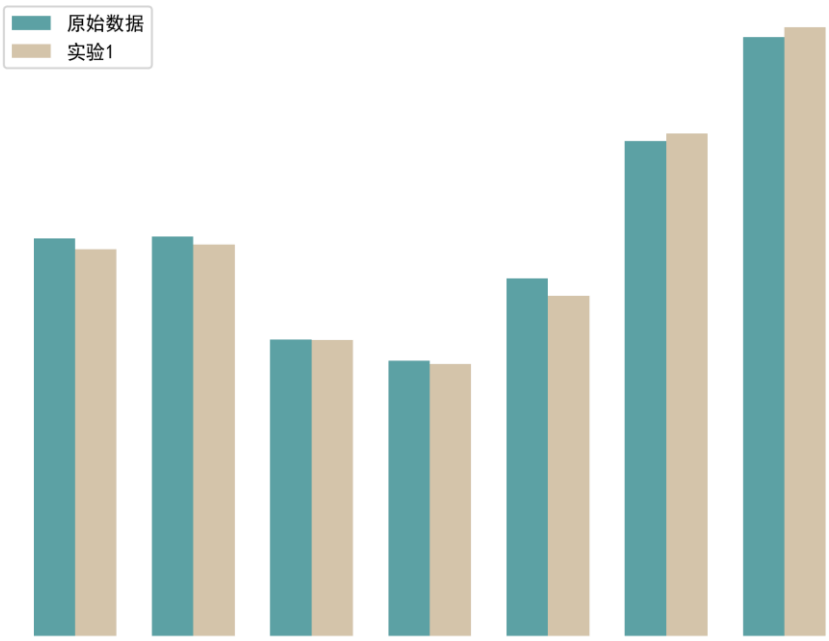
4.2.3 实验一及其结果

实验一：验证解中某些变量的取值发生改变对目标函数的影响。

（1） 随机生成初始解，且保证解的合法性，计算目标函数。

（2） 对每个解的某些位置的变量，交换他们的取值。

（3） 计算完成操作后的解所对应的目标函数，生成图像。



图：

在图中，绿色是没有更新的解所对应的函数值，黄色柱状图是更新解以后所对应的变量值，可以看出，如果仅仅只是交换两个变量值的更新方式，更新和不更新没有什么显著的区别。

4.2.4 实验二及其结果

实验二：两个解进行交叉行为对目标函数的影响。

（1） 随机生成初始解，且保证解的合法性，计算目标函数。

（2） 解两两配对，进行交叉操作。

（3） 计算完成操作后的解所对应的目标函数，生成图像。

乘法运算法($\overset{+}{O}$)，解中每一个等于变量进行，值为另一个解相对应变量的值，然后进行交叉操作。

下面将解释什么是交叉操作。交叉操作是基因算法的一个解更新行为，是指在两个父代个体（解）之间交换染色体上的一部分基因信息（变量）并生成新的子代个体（解）。作为进化算法的重要步骤之一，用于产生更好的后代解，并增加群体的多样性。

交叉操作通常需要满足如下两个条件：首先，确定交叉方式，即交叉方式有多种形式，例如单点交叉、多点交叉、均匀交叉和线性交叉等，本文选择了双点交叉的方式。其次，保证基因信息的完整性和正确性，即交叉操作需要保证新生成的解是有效的解，因此需要保证每个基因在子代中都不缺失或重复。

下面讲解双点交叉的操作，在双点交叉（Two-point Crossover）中，需要随机选择两个交叉点的位置，然后将两个父代个体的这两个交叉点之间的基因序列进行互换，从而产生新的子代个体。

下面以一个长度为 6 的染色体为例，说明双点交叉的具体操作过程：假设有两个父代个体分别是：

$$S_1 = [0,1,0,1,0,1]$$

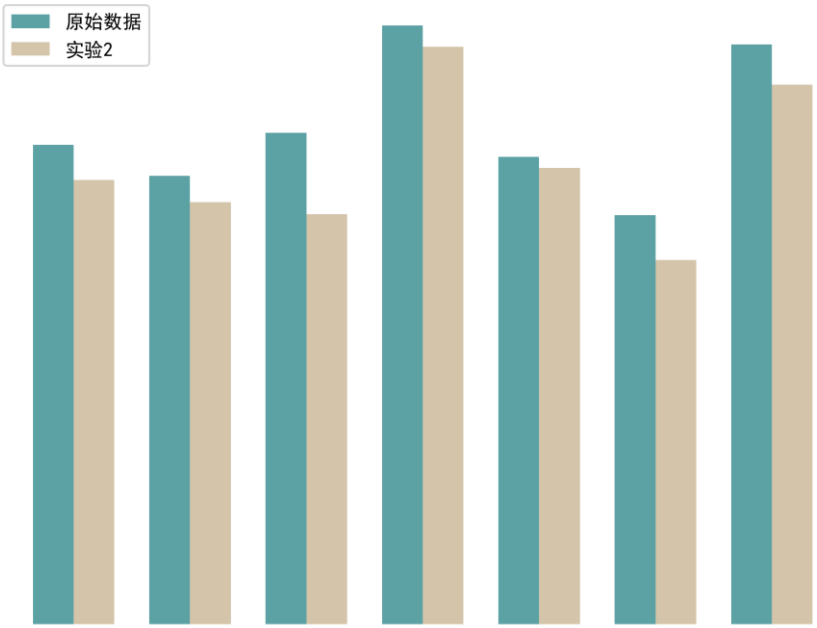
$$S_2 = [1,0,1,0,1,0]$$

我们随机选择两个交叉点位置 k1=2 和 k2=5，然后将两个父代个体在这两个位置之间的基因信息进行互换，得到新的子代个体 S_3 和 S_4：

$$S_3 = [0,0,1,0,1,1]$$

$$S_4 = [1,1,0,1,0,0]$$

子代个体中来自父代个体的基因信息（变量）在交叉点之前或之后是完全一致的，但在交叉点之间则发生了互换。



图：

在图中，绿色是没有本地执行所对应的函数值，黄色柱状图是进行交叉操作以后所对应的变量值，可以看出，虽然比实验一效果要好，但是这种交叉这种更新策略效果并不明显，且这几次实验之间的差值比较大，更加说明了变量之间有相关性。

4.3 基于贪心法调整的离散侏儒猫鼬算法

本节提出了修改后的基于贪心法调整的离散侏儒猫鼬算法。

4.3.1 基于贪心法的调整策略

为了克服变量之间存在的相关性，基因算法有着解码和编码两个过程，来配合交叉操作，使得交叉操作更具有随机性，克服了变量之间的相关性。

我们不使用基因算法，而是想将侏儒猫鼬算法改造为能够求解离散形式问题