全国大学生数学建模竞赛论文模板

摘要

随着物联网设备的指数级增长，移动通信需求呈现爆发式增长，推动了网络架构的不断演进。本文针对无线网络切片资源管理的一系列复杂优化问题，设计并实现了一套从静态到动态、从同构到异构、从性能优先到兼顾能耗的渐进式求解方案。

对于问题一，我们针对单基站静态场景，建立了服务质量（QoS）最大化模型。通过对所有可能的资源块（RB）分配方案进行穷举搜索，得到了全局最优解：为 URLLC、eMBB、mMTC 切片分别分配 20、10、20 个 RB 时，可达到最大 QoS 得分 15.78。此结果为后续更复杂的问题提供了性能基准。

对于问题二，面对动态任务到达和时变信道，我们引入了**模型预测控制（MPC）**框架。将总时长划分为 10 个 100ms 的决策窗口，在每个窗口开始时，根据当前系统状态（如用户队列）进行枚举寻优。该方法有效应对了系统的动态性，实现了 337.42 的累计 QoS，验证了 MPC 框架处理时变问题的有效性。

对于问题三，在存在同频干扰的多微基站场景下，决策维度急剧增加。我们建立了用户接入、RB 分配和功率控制的联合优化模型，并提出一种 **MPC 结合遗传算法（GA）**的两层求解框架。外层 MPC 负责时域滚动，内层 GA 通过混合编码方案，高效求解每个窗口内非凸、高维的静态资源分配问题，最终在有效抑制干扰的同时，获得了 853.37 的累计 QoS。

对于问题四，针对宏基站（MBS）与微基站（SBS）共存的异构网络，我们对 MPC+GA 框架进行了扩展。模型引入异构网络下的 MBS 与 SBS 资源分配和功率控制，以及用户只能接入 MBS 或最近 SBS 的约束。求解结果显示出智能的**网络功能协同策略：**无干扰、资源丰富的 MBS 主要承载海量 mMTC 连接，而靠近用户的 SBS 则重点保障 URLLC 等高性能业务，最终将累计 QoS 提升至 1041.28。

对于问题五，为在保障 QoS 的同时最小化网络能耗，我们设计了一种新颖的**两阶段优化算法**。在 MPC 框架的每个窗口内，第一阶段采用 GA 优化发射功率以最小化能耗；第二阶段在给定功率下，通过枚举优化 RB 分配以最大化 QoS。该分层解耦策略成功平衡了性能与能耗的矛盾，在将总能耗控制在 183.68 焦耳的同时，取得了 381.17 的累计 QoS。

综上，本文通过对一系列问题的层层递进的建模与求解，系统地展示了如何运用枚举、MPC、遗传算法及分层优化等方法，解决不同复杂度下的网络切片资源管理难题，为设计高效、智能的无线资源分配方案提供了有价值的思路与验证。

关键字： 模型预测控制 (MPC)　遗传算法 (GA)　异构网络　两阶段优化算法

目录

1 问题重述

1.1 问题背景

随着移动通信需求的激增和物联网（IoT）的快速发展，网络架构正向异构化和虚拟化演进。异构蜂窝网络（HetNet）通过混合部署宏基站与微基站，有效提升了网络容量与覆盖。在此基础上，5G 网络切片技术利用网络功能虚拟化（NFV），将单一物理网络划分为多个逻辑切片，以满足超高可靠低时延（URLLC）、增强移动宽带（eMBB）和大规模机器通信（mMTC）等多样化服务需求。

无线资源的管理依赖于正交频分多址接入（OFDMA）技术，它将频谱划分为时频资源块（RB）进行灵活分配。因此，在异构网络与多切片共存的复杂场景下，如何设计高效的资源块和功率分配策略，以最大化用户服务质量并优化能耗，成为无线资源管理领域的核心挑战。

1.2 问题要求

本赛题旨在研究异构蜂窝网络中基于网络切片的无线资源管理问题。核心任务是设计一套优化方案，在满足不同用户多样化服务质量（QoS）需求的同时，实现系统资源的高效利用。具体来说，需要解决以下几个层层递进的问题：

- **问题一：**针对单个微基站和单一用户任务的场景,研究如何将有限的资源块在 URLLC、eMBB、mMTC 三类切片间进行静态分配，以实现用户服务质量的最大化。
- **问题二：**在动态场景下，考虑用户任务的随机到达和用户移动性，设计一个多周期的资源分配策略。该策略需要在 10 个决策点上对资源进行重新分配，不仅要服务新到达的任务，还要处理队列中积压的任务，目标是最大化整个时间窗口内的总体用户服务质量。
- **问题三：**将场景扩展到多个微基站，引入了基站间的同频干扰问题。要求在进行资源块分配的同时，对每个基站各切片的发射功率进行协同优化，以抑制干扰，最大化全系统的用户服务质量。
- **问题四：**构建一个包含宏基站和多个微基站的异构网络模型。在此模型中，需要为每个用户决策其接入基站（宏基站或微基站），并为所有基站进行切片划分和功率控制，以应对更大规模的用户需求和更复杂的网络环境，最终目标仍是最大化整体服务质量。
- **问题五：**在问题四的基础上，引入基站能耗模型，探讨在保证最大化用户服务质量的同时，如何通过优化资源分配策略来实现网络总能耗的最低化，从而在服务性能和绿色节能之间取得平衡。

1.3 我们的工作

2 模型假设

为简化问题，本文做出以下假设：

- 假设 1
- 假设 2
- 假设 3

3 符号说明

符号	含义	单位
R_n	基站 n 的 RB 总数（MBS：100；SBS：50）	RB
b	单个 RB 带宽（360 kHz）	kHz
v_s	切片 s 内每用户占用的 RB 数（U/e/m 为 $10/5/2$）	RB
$x_{n,s}(t)$	基站 n 在窗口 t 为切片 s 分配的 RB 数	RB
$p_{n,s}(t)$	基站 n 在窗口 t 对切片 s 的发射功率	dBm
$\gamma_k(\tau)$	用户 k 在时刻 τ 的信干噪比（SINR）	-
$r_k(\tau)$	用户 k 在时刻 τ 的传输速率	bps
D_k	用户 k 的任务数据量	Mbit
L_k^s	用户 k 在切片 s 的总时延	s
$Q_k(t)$	用户 k 在时刻 t 的队列积压量	Mbit
L_s^{SLA}	切片 s 的时延 SLA 要求	ms
M_s	切片 s 的任务丢失惩罚系数（$s \in \{U, e, m\}$）	-
$E_{total}(t)$	窗口 t 内全网总能耗	J

注：未列出的以及重复的符号均以首次出现处为准。

4 问题一的模型的建立和求解

4.1 问题一的描述与分析

问题一考虑单个微基站的资源分配场景。该基站拥有 50 个资源块（Resource Block, RB），需要为三类网络切片——URLLC（高可靠低时延）、eMBB（增强移动宽带）和 mMTC（大规模机器通信）进行资源分配，以最大化用户服务质量。这是一个静态资源分配优化问题，需要在满足资源约束的条件下，找到最优的资源块分配方案。

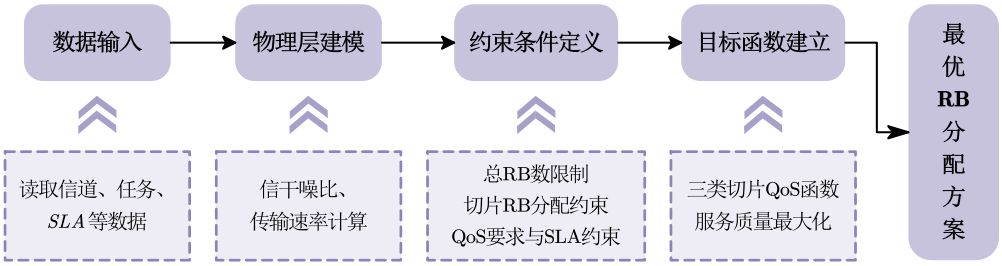

图 **1**　问题一场景与资源分配分析示意图

4.2 预备工作

4.2.1 关键参数补充

基站与信号参数：基站发射功率 $p_{\text{tx}} = 30$ dBm，单个资源块（RB）带宽 $b = 360$ kHz，噪声系数 $NF = 7$ dB。

决策周期：系统每 $T_{\text{window}} = 100$ ms 进行一次资源分配决策。

切片资源占用与服务等级协议（SLA）：URLLC 切片：每个用户占用 10 个资源块，要求时延 $L_k^U \leq 5$ ms，速率 $r_k \geq 10$ Mbps；eMBB 切片：每个用户占用 5 个资源块，要求时延 $L_k^e \leq 100$ ms 且传输速率 $r_k \geq 50$ Mbps；mMTC 切片：每个用户占用 2 个资源块，要求时延 $L_k^m \leq 500$ ms，速率 $r_k \geq 1$ Mbps。

QoS 评估参数：U 切片的效益折扣系数 $\alpha = 0.95$，任务丢失惩罚 $M_U = 5$；e 切片的任务丢失惩罚 $M_e = 3$；m 切片的任务丢失惩罚 $M_m = 1$。

4.3 模型建立

问题一为单时刻静态场景，不引入显式时间索引。每位用户仅有一个任务，服务在一个决策窗口内完成；窗口内允许在各切片内进行“编号靠前优先”的非抢占串并行调度。

4.3.1 传输速率计算

根据附录中的信号传输模型，用户 k 占用 i_k 个资源块时的接收功率为：

$$p_{\text{rx},k} = 10^{\frac{p_{\text{tx}} - \phi_k}{10}} \cdot |h_k|^2 \quad \text{(mW)} \tag{1}$$

其中，p_{tx} 为基站发射功率（dBm），ϕ_k 为大规模衰减（dB），h_k 为小规模瑞利衰落幅度，$p_{\text{rx},k}$ 为接收功率（mW）。

考虑噪声功率的影响，噪声功率谱密度为：

$$N_0(i_k) = -174 + 10\log_{10}(i_k \cdot b) + 7 \quad \text{(dBm)} \tag{2}$$

其中，i_k 为用户 k 占用的 RB 数量，b 为单 RB 带宽（Hz），-174 dBm/Hz 为热噪声谱密度，7 dB 为噪声系数。

信噪比（SNR）为：

$$\gamma_k = \frac{p_{\mathrm{rx},k}}{10^{\frac{N_0(i_k)}{10}}} \tag{3}$$

其中，$N_0(\cdot)$ 以 dBm 计，$10^{\frac{N_0(i_k)}{10}}$ 为噪声功率（mW）。

根据香农公式，用户 k 的传输速率为：

$$r_k = i_k \cdot b \cdot \log_2\left(1+\gamma_k\right) \quad \text{(bps)} \tag{4}$$

在本问的调度中，同一切片 $s \in \{U, e, m\}$ 内的每位用户占用固定 RB 数量 v_s，故 $i_k \equiv v_s$。

4.3.2 时延计算模型

根据附录中描述的用户任务服务流程，用户的总时延由排队时延和传输时延两部分构成。

用户 k 的传输时延 T_k 为完成其任务数据量 D_k 所需的传输时间：

$$T_k = \frac{D_k \times 10^6}{r_k} \quad \text{(s)} \tag{5}$$

其中，D_k 为任务数据量（Mbit），r_k 为用户的传输速率（bps）。

给定切片 s 被分配的 RB 数量 n_s 与每用户占用 v_s，并发能力为 $C_s = \lfloor n_s/v_s \rfloor$。在同一调度窗口内，切片 s 中的用户按“编号靠前优先”顺序：初始分配给前 C_s 位用户，其余用户在有会话完成后接续开始服务。由此得到每位用户的等待时延 Q_k（其开始服务时刻）与传输时延 T_k，总时延

$$L_k^s = Q_k + T_k, \quad s \in \{U, e, m\}. \tag{6}$$

4.3.3 服务质量评估函数

根据附录中的用户服务质量定义，并结合计算出的总时延，不同切片的 QoS 评估函数如下：

(1) U 切片（URLLC）

服务质量函数为：

$$y_k^U = \begin{cases} \alpha^{L_k^U} & \text{若} L_k^U \le L_U^{\mathrm{SLA}} \\ -M_U & \text{若} L_k^U > L_U^{\mathrm{SLA}} \end{cases} \tag{7}$$

其中，$\alpha \in (0,1)$ 为效益折扣系数（本题取 $\alpha = 0.95$），M_U 为 U 切片任务丢失惩罚系数，L_U^{SLA} 为 U 切片时延 SLA。

(2) e 切片（eMBB）

e 切片用户采用三段式 QoS 函数：

$$y_k^e = \begin{cases} 1 & \text{若} r_k \geq r_e^{\text{SLA}} \text{ 且} L_k^e \leq L_e^{\text{SLA}} \\ \frac{r_k}{r_e^{\text{SLA}}} & \text{若} r_k < r_e^{\text{SLA}} \text{ 且} L_k^e \leq L_e^{\text{SLA}} \\ -M_e & \text{若} L_k^e > L_e^{\text{SLA}} \end{cases} \tag{8}$$

其中，r_e^{SLA}、L_e^{SLA} 分别为 e 切片的速率与时延 SLA，M_e 为惩罚系数。

(3) m 切片（mMTC）

每个 mMTC 用户 k 的 QoS 评估为：

$$y_k^m = \begin{cases} \dfrac{\sum_{i \in \mathcal{U}_m} c_i'}{\sum_{i \in \mathcal{U}_m} c_i} & \text{若} L_k^m \leq L_m^{\text{SLA}} \\ -M_m & \text{若} L_k^m > L_m^{\text{SLA}} \end{cases} \tag{9}$$

其中，$\mathcal{U}_m$ 为 m 切片用户集合，c_i 表示用户 i 是否有任务（本问为“有数据量”），c_i' 表示该用户是否成功在 SLA 内完成任务。实现上先计算比例 ratio $= \frac{\sum c_i'}{\sum c_i}$，再对每个有任务的 m 用户按“成功得 ratio，失败得 $-M_m$”计分并加总。

4.3.4 优化模型

基于上述分析，建立如下单时刻优化模型：

$$\max_{n_U, n_e, n_m} \quad Q = \sum_{k \in \mathcal{U}_U} y_k^U + \sum_{k \in \mathcal{U}_e} y_k^e + \sum_{k \in \mathcal{U}_m} y_k^m$$

$$\text{s.t.} \quad \begin{cases} n_U + n_e + n_m = N \\ n_U \bmod 10 = 0 \\ n_e \bmod 5 = 0 \\ n_m \bmod 2 = 0 \\ n_s \in \mathbb{Z}_{\geq 0}, \quad \forall s \in \mathcal{S} \end{cases} \tag{10}$$

其中，n_U, n_e, n_m 分别为分配给 U、e、m 切片的 RB 个数，$\mathcal{S} = \{U, e, m\}$，$\mathcal{U}_s$ 为切片 s 的用户集合。

4.4 模型求解

该优化问题属于整数规划问题，考虑到总资源块数量有限（$N = 50$），且各切片用户占用 RB 数量固定，使得分配给各切片的 RB 数量 $n_s(t)$ 的可行组合是有限的。因此，我们采用枚举法结合调度仿真的策略进行求解，以确保找到全局最优解。算法流程如下（本题固定单个决策时刻 $t = t_0$）：

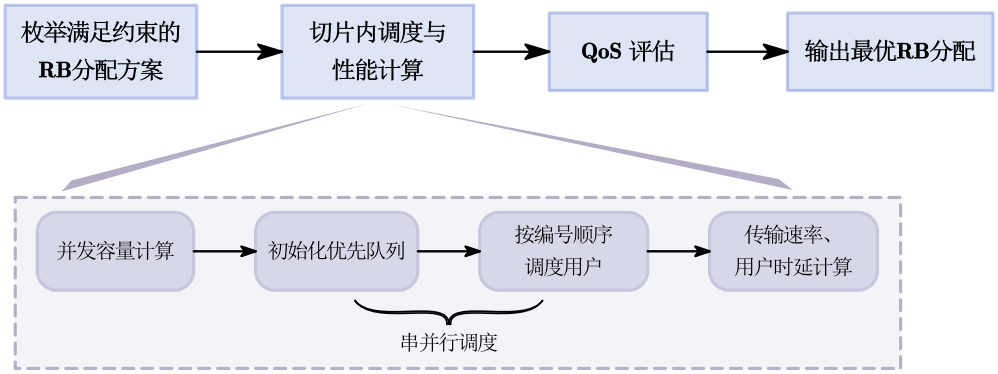

图 **2**　第一问资源分配与调度算法流程图

Step1：生成 RB 分配方案

我们枚举所有满足约束条件的 RB 分配方案 $(n_U(t), n_e(t), n_m(t))$。为避免资源浪费，分配给各切片的 RB 数量应为其用户占用量的整数倍。具体地：

- $n_U(t)$ 在 $\{0, 10, 20, \ldots, 50\}$ 中取值。
- $n_e(t)$ 在 $\{0, 5, 10, \ldots, 50 - n_U(t)\}$ 中取值。
- $n_m(t) = 50 - n_U(t) - n_e(t)$，并检验 $n_m(t)$ 是否为 2 的倍数。若否则舍弃该方案。

Step2：切片内调度与性能计算

对于每一个有效的 RB 分配方案，我们在各切片内部独立进行调度仿真，以计算每个用户的性能指标。

- **并发容量计算：**对于切片 $s \in \{U, e, m\}$，其并发服务能力为 $C_s = \lfloor n_s(t)/v_s \rfloor$。
- **串并行调度：**在 100ms 决策周期内，我们采用一种串并行的服务策略。初始时，将前 C_s 个用户（按用户编号排序）分配至并发信道进行传输。当某个用户完成传输后，其占用的信道立即释放，并分配给队列中的下一个用户。
- **性能计算：**通过该调度过程，我们可以计算出每个用户 k 的传输时延 $T_k(t)$ 和等待时延 $Q_k(t)$，从而得到总时延 $L_k(t) = Q_k(t) + T_k(t)$。用户的传输速率 $r_k(t)$ 也一并计算得出。

Step3：服务质量评估

根据步骤 2 计算出的性能指标 (L_k, r_k)，我们依据模型中定义的服务质量评估函数计算每个用户的 QoS 得分 y_k^s，并汇总得到当前 RB 分配方案下的总服务质量 $Q = \sum y_k^s$。

Step4：寻找最优方案

遍历所有 RB 分配方案后，总服务质量 Q 最高的方案即为问题的最优解。我们记录下最优方案对应的 (n_U, n_e, n_m) 组合、各用户的详细性能指标以及最终的总 QoS 值。

4.5 结果分析

通过执行上述算法，我们得到的最优资源分配方案及对应的性能指标如下：

最优资源分配方案

经枚举计算，我们找到了 3 个并列的最优资源分配方案，它们均能使系统总服务质量达到最大值 15.7823。这三个方案的具体 RB 分配如下表所示。

表 **1**　并列最优资源分配方案

方案	URLLC RB 数 (n_U)	eMBB RB 数 (n_e)	mMTC RB 数 (n_m)	总 QoS
1	20	10	20	15.7823
2	20	20	10	15.7823
3	30	10	10	15.7823

在以上所有方案中，各切片获得的 QoS 合计分数均相同，分别为：URLLC QoS 合计 1.9870，eMBB QoS 合计 3.7953，mMTC QoS 合计 10.0000。

这三个方案均实现了资源的高效利用。在实际部署中，可以根据网络运营商的偏好进行选择。例如，方案 1（20, 10, 20）为 mMTC 分配了最多的资源，适合未来 mMTC 连接数可能增加的场景；方案 2（20, 20, 10）则向 eMBB 倾斜，适合视频流量大的场景；方案 3（30, 10, 10）则最优先保障 URLLC 业务。这些方案共同构成了问题的最优解集。

综上所述，我们提出的资源分配方案能够有效满足各类切片用户的服务需求，实现了系统整体服务质量的最优。

5　问题二的模型的建立和求解

5.1 问题二的描述与分析

问题二考虑动态环境下的多时间段资源分配场景。与问题一的静态单时刻分配不同，问题二面临用户移动性、信道时变性以及任务队列动态变化的复杂问题。在 T_{tot} 的观测窗口内，系统需要每 T_w 进行一次资源分配决策，既要处理新到达的任务，又要考虑积压在排队队列中的历史任务。这是一个多阶段动态优化问题，需要在时间维度上综合考虑任务到达、信道变化和排队延迟的耦合影响。

5.2 预备工作

5.2.1 模型预测控制（MPC）简介

模型预测控制（Model Predictive Control, MPC）是一种先进的控制策略，广泛应用于工业控制、自动驾驶、机器人等领域。它通过预测未来系统的行为，优化当前的控制

输入，从而实现对复杂系统的高效控制。

核心思想:

基于模型的预测: MPC 利用系统的数学模型（如状态空间模型、传递函数等）预测未来一段时间内系统的输出行为。

滚动优化: 在每个时刻，MPC 通过求解一个优化问题，找到未来一段时间的最优控制输入序列，使系统输出尽可能接近目标，同时满足约束条件。

反馈调整: 只执行当前时刻的控制输入，随后根据新的测量值更新预测，重复上述过程。这种滚动优化和反馈机制使 MPC 能够应对系统的不确定性和扰动。

5.3 模型建立

问题二模型大部分与问题一相同，可直接复用问题一中的模型，不再重复说明。不同之处在于问题二允许 t 在决策窗口内按 $\mathcal{T} = \{0, 100, \ldots, 900\}$ 演化，并引入任务到达与队列动态，下面将具体阐述。

5.3.1 任务队列动态演化模型

为精确描述任务的动态变化，我们引入两个时间尺度：

- **决策时刻** t：每 100ms 进行一次资源分配决策，对应 $t \in \{0, 100, 200, \ldots, 900\}$。
- **仿真时刻** τ：以 1ms 为步长，用于模拟任务到达和信道变化，τ 表示具体的毫秒时刻。

定义用户 k 在时刻 t 的任务队列状态：

- $A_{k,\tau}(t)$：用户 k 在时刻 τ 到达且在时刻 t 仍在队列中的任务数据量（Mbit）
- $Q_k(t) = \sum_{\tau=0}^{t} A_{k,\tau}(t)$：用户 k 在时刻 t 的总排队任务量

任务队列的动态演化遵循以下规律：

(1) 任务到达：在每个 1ms 时刻 τ，根据数据文件读取新到达任务：

$$A_{k,\tau}(\tau) = \text{TaskFlow}_k(\tau) \tag{11}$$

(2) 任务服务：在决策时刻 t，若用户 k 被分配 $i_k(t)$ 个 RB，则在接下来的 100ms 内可传输的数据量为：

$$S_k(t) = r_k(t) \times 0.1 \times 10^{-6} \quad \text{(Mbit)} \tag{12}$$

(3) 队列更新：任务按 FIFO（先进先出）顺序服务，队列更新规则为：

$$Q_k(t+100) = \max\left(0, Q_k(t) + \sum_{\tau=t+1}^{t+100} \text{TaskFlow}_k(\tau) - S_k(t)\right) \tag{13}$$

5.3.2 时延计算模型（任务级）

由于存在到达过程，问题二按任务到达时刻进行时延度量。对于用户 k 在时刻 τ 到达的任务，其总时延包含排队时延和传输时延：

(1) 排队时延：任务在时刻 τ 到达，在时刻 t_{start} 开始服务，则排队时延为：

$$Q_{k,\tau} = t_{\text{start}} - \tau \tag{14}$$

(2) 传输时延：假设任务从时刻 t_{start} 开始传输，数据量为 $D_{k,\tau}$，则传输时延为：

$$T_{k,\tau} = \frac{D_{k,\tau} \times 10^6}{r_k(t_{\text{start}})} \tag{15}$$

(3) 总时延：

$$L^s_{k,\tau} = Q_{k,\tau} + T_{k,\tau}, \quad s \in \{U, e, m\} \tag{16}$$

5.3.3 窗口内 QoS 计分规则

仅对“**在当前窗口内完成**”的 URLLC/eMBB 任务计分；mMTC 采用“**本窗到达用户的比例计分**”。令窗口 t 的结束时刻为 $t + 100$，定义集合：

$$\mathcal{F}_U(t) = \big\{(k, \tau_0) \mid k \in \mathcal{U}_U,\ \tau_f \in [t,\, t + T_w)\big\}, \tag{17}$$

$$\mathcal{F}_e(t) = \big\{(k, \tau_0) \mid k \in \mathcal{U}_e,\ \tau_f \in [t,\, t + T_w)\big\}. \tag{18}$$

$$Y_U(t) = \sum_{(k,\tau_0)\in\mathcal{F}_U(t)} y^U_{k,\tau_0}, \quad y^U_{k,\tau_0} = \begin{cases} \alpha^{L_{k,\tau_0}} & \text{若} L_{k,\tau_0} \le L^{\text{SLA}}_U \\ -M_U & \text{若} L_{k,\tau_0} > L^{\text{SLA}}_U \end{cases} \tag{19}$$

$$Y_e(t) = \sum_{(k,\tau_0)\in\mathcal{F}_e(t)} y^e_{k,\tau_0}, \quad y^e_{k,\tau_0} = \begin{cases} 1 & \text{若} L_{k,\tau_0} \le L^{\text{SLA}}_e \text{ 且} r_{\text{avg}} \ge r^{\text{SLA}}_e \\ \dfrac{r_{\text{avg}}}{r^{\text{SLA}}_e} & \text{若} L_{k,\tau_0} \le L^{\text{SLA}}_e \text{ 且} r_{\text{avg}} < r^{\text{SLA}}_e \\ -M_e & \text{若} L_{k,\tau_0} > L^{\text{SLA}}_e \end{cases} \tag{20}$$

$$Y_m(t) = \sum_{k\in\mathcal{U}_m} y^m_k(t), \quad y^m_k = \begin{cases} \dfrac{\sum_{i\in\mathcal{U}_m} c'_i}{\sum_{i\in\mathcal{U}_m} c_i} & \text{若} L^m_k \le L^{\text{SLA}}_m \\ -M_m & \text{若} L^m_k > L^{\text{SLA}}_m \end{cases} \tag{21}$$

5.3.4 总体优化模型

在上述定义下，问题二的动态优化模型写作：

$$\max_{\{n_U(t),n_e(t),n_m(t)\}_{t\in\mathcal{T}}} \quad Q=\sum_{t\in\mathcal{T}}\Big(Y_U(t)+Y_e(t)+Y_m(t)\Big)$$

$$\text{s.t.} \quad \begin{cases} n_U(t)+n_e(t)+n_m(t)=50 \\ n_U(t)\bmod 10=0,\ n_e(t)\bmod 5=0,\ n_m(t)\bmod 2=0 \\ Q_k(t+100)=\max(0,Q_k(t)+\Delta A_k(t)-S_k(t)) \\ n_s(t)\in\mathbb{Z}_{\geq 0} \\ \forall t\in\mathcal{T},\ s\in\mathcal{S},\ \forall k \end{cases} \tag{22}$$

该模型的核心挑战在于：(1) 状态空间（用户队列）的动态演化导致各阶段决策相互耦合；(2) 任务到达的随机性与信道的时变性增加了预测难度；(3) 排队时延与传输时延的耦合优化关系复杂。需要设计高效的求解算法来处理这一复杂的多阶段动态优化问题。

5.4 模型求解

通过以上对模型的分析，我们决定采用模型预测控制（Model Predictive Control, MPC）算法求解，该算法已在预备工作中简要介绍。

算法的核心思想是：在每个决策时刻 $t\in\{0,100,\ldots,900\}$，我们面对当前的系统状态（主要是各用户的任务队列），通过枚举所有可能的 RB 分配方案，并对每种方案进行精细化的仿真，来预测未来 100ms 内的系统服务质量。然后，我们选择能使当前窗口 QoS 最大化的方案进行实施，并将演化后的系统状态作为下一个决策时刻的初始条件。具体算法流程如下：

Step1：初始化

在仿真开始时刻 $t=0$，初始化所有用户的任务队列为空。

Step2：逐窗口迭代决策

对于每个决策窗口 w（对应时间段 $[t,t+100)$，其中 $t=w\times 100$）：

1. **生成 RB 分配方案**:与问题一类似,枚举所有满足约束条件的 RB 分配方案 $(n_U(t),n_e(t),n_m(t))$，确保资源总量为 50，且各切片 RB 数量分别为 10、5、2 的倍数。
2. **窗口内调度仿真与评估**：对于每个分配方案，进行 100ms 窗口的详细仿真。仿真从当前系统状态出发，1ms 步长推进，动态加入新到达任务，按切片并发能力和“编号靠前优先”原则调度用户，任务完成后立即释放资源并接续新用户。全过程记录任务的到达、服务、完成时刻，计算端到端时延，并据各切片 QoS 评估函数统计窗口内所有已完成任务的服务质量得分。

3. **选择最优方案并更新状态：**遍历所有 RB 分配方案后，我们选择在当前窗口内获得累计 QoS 总分最高的方案作为本次决策的结果。然后，我们将该最优方案对应的仿真结束时刻（$t+100$）的用户队列状态，作为下一个决策窗口的初始状态。

Step3：汇总结果

重复 Step2，直到完成所有 10 个决策窗口（$t=0$ 至 $t=900$）的决策。最后，将每个窗口获得的最优 QoS 得分进行累加，得到整个 1000ms 内的总服务质量。通过这种方式，我们得到了一系列动态的 RB 分配决策，以及最终的系统整体性能评估。

5.5 结果分析

通过执行上述基于 MPC 的贪心算法，我们得到了 1000ms 内的动态资源分配策略，其总服务质量达到了 **352.1029**。

5.5.1 最优资源分配序列

算法在 10 个决策窗口中选择的 RB 分配序列如下表所示。该序列是在每个窗口选择瞬时最优解（若有多个则选择第一个）的结果。

表 **2**　问题二动态资源分配序列

决策时刻 (ms)	URLLC RB 数 (n_U)	eMBB RB 数 (n_e)	mMTC RB 数 (n_m)	窗口 QoS
0	10	20	20	65.490
100	10	20	20	40.028
200	10	20	20	38.835
300	10	0	40	32.800
400	20	0	30	31.850
500	10	0	40	24.250
600	10	0	40	27.100
700	20	0	30	32.800
800	20	0	30	31.850
900	20	0	30	27.100
总计	-	-	-	**352.103**

在整个仿真周期内，各切片累计获得的 QoS 分数分别为：URLLC QoS 合计 **205.8175**，eMBB QoS 合计 **46.2854**，mMTC QoS 合计 **100.0000**。

从资源分配序列（表 2）可以看出，算法展现了良好的动态适应性。在仿真初期（0-300ms），eMBB 业务有大量任务到达，算法明智地为其分配了 20 个 RB 以快速处理，

获得了较高的 QoS。在 300ms 后，eMBB 任务队列清空，算法果断地将其 RB 资源完全回收，转而分配给 URLLC 和 mMTC，以应对后续的 URLLC 任务并保证 mMTC 的稳定接入。

综上所述，我们提出的动态资源分配模型与求解算法，能够在时变的信道和任务到达条件下，做出快速有效的决策，实现了系统在整个时间窗口内总服务质量的最优。

6 问题三的模型的建立和求解

6.1 问题三的描述与分析

6.2 预备工作

为适配附件三的多微基站、同频复用且存在小区间干扰的场景，定义如下集合与索引：

- 基站集合：$\mathcal{N} = \{1, 2, 3\}$（分别对应 BS1、BS2、BS3）。
- 切片集合：$\mathcal{S} = \{U, e, m\}$，分别对应 URLLC、eMBB、mMTC。
- 用户集合：$\mathcal{K} = \mathcal{K}_U \cup \mathcal{K}_e \cup \mathcal{K}_m$，其中 $\mathcal{K}_U = \{\text{U1}, \text{U2}\}$，$\mathcal{K}_e = \{\text{e1}, \ldots, \text{e12}\}$，$\mathcal{K}_m = \{\text{m1}, \ldots, \text{m30}\}$。
- 决策时刻集合：$\mathcal{T} = \{0, 100, \ldots, 900\}$（单位 ms），每个决策窗口长度为 100 ms；窗口内以 1 ms 步长进行链路与队列仿真，记窗口内细粒度时刻集合为 $\mathcal{F}(t) = \{t, t+1, \ldots, t+99\}$。

其余关键系统参数与问题一、二保持一致。

6.3 模型建立

问题三的基本模型与问题二一致，但是需要考虑多个微基站之间的信号干扰，故需要对信道与干扰模型进行重新建模，其余复用模型不再重复。

6.3.1 信道与干扰模型

附件三提供了每 1 ms 的大规模损耗 $\phi_{n,k}(\tau)$（dB）与小规模瑞利衰落 $h_{n,k}(\tau)$。设窗口 $t \in \mathcal{T}$ 内细粒度时刻为 $\tau \in \mathcal{F}(t)$，若用户 $k \in \mathcal{K}_s$ 在窗口 t 由基站 n 服务且被分配切片 s 的 RB，则其接收功率（mW）为

$$p_{\text{rx},n\to k}(\tau) = 10^{\frac{p_{n,s}(t) - \phi_{n,k}(\tau)}{10}} \cdot |h_{n,k}(\tau)|^2. \tag{23}$$

噪声功率与占用 RB 数 i 成正比，换算为线性功率（mW）：

$$N_0(i) = 10^{\frac{-174 + 10\log_{10}(i \cdot b) + NF - 30}{10}}. \tag{24}$$

微基站同频复用引入同信道干扰。为保持“同一 RB 索引才互扰”的规则，我们令每站在窗口 t 内将其 50 个 RB 在频域上按切片连续划分且次序固定（例如 U-e-m），每个切片获得一段连续 RB 区间，跨站的相同 RB 索引构成同信道。于是用户 k 的瞬时信干噪比为

$$\gamma_k(\tau) = \frac{p_{\mathrm{rx},n\to k}(\tau)}{\sum\limits_{u\in\mathcal{N},\ u\neq n} I_{u\to k}(\tau) + N_0(i_s)}, \quad s \in \mathcal{S}, \tag{25}$$

其中 $I_{u\to k}(\tau)$ 表示来自他站 u、在与 k 所占 RB 索引重叠的切片 RB 上的干扰功率，按与上式相同的接收功率表达（由 $p_{u,s'}(t), \phi_{u,k}(\tau), h_{u,k}(\tau)$ 决定）。基于香农公式，窗口内瞬时速率为

$$r_k(\tau) = i_s \cdot b \cdot \log_2\big(1 + \gamma_k(\tau)\big) \quad (\text{bps}). \tag{26}$$

6.3.2 任务到达与队列演化

任务队列的动态演化过程与问题二完全一致，队列状态 $Q_k(t)$ 仍以每个用户为中心进行追踪，其演化遵循公式 (13)。

关键区别在于，服务量 $S_k(t)$ 的计算变得更为复杂，因为它取决于用户 k 在当前窗口 t 接入了哪个基站 n，以及所有基站的功率决策 $\{p_{n,s}(t)\}$。这将在后续的服务质量评估函数中具体体现。

6.3.3 决策变量与优化模型

决策变量：

- RB 切片分配：$x_{n,s}(t) \in \mathbb{Z}_{\geq 0}$
- 发射功率：$p_{n,s}(t) \in [10, 30]$ (dBm)
- 接入关联：$a_{n,k}(t) \in \{0, 1\}$, 当 $a_{n,k}(t) = 1$ 则 k 在窗口 t 仅由站 n 调度

综合上述要素，第三问的动态联合优化模型可表述为（跨 10 个窗口聚合）：

$$\max_{\{x,p,a\}} \quad Q_{\text{total}} = \sum_{t\in\mathcal{T}} \left[\sum_{k\in\mathcal{K}_U} \sum_{\tau\in\mathcal{A}_k(t)} y^U_{k,\tau} + \sum_{k\in\mathcal{K}_e} \sum_{\tau\in\mathcal{A}_k(t)} y^e_{k,\tau} + \sum_{k\in\mathcal{K}_m} \sum_{\tau\in\mathcal{A}_k(t)} y^m_{k,\tau} \right]$$

$$\text{s.t.} \quad \begin{cases} \sum_{s\in\mathcal{S}} x_{n,s}(t) = 50 \\ x_{n,U}(t) \bmod 10 = 0, x_{n,e}(t) \bmod 5 = 0, x_{n,m}(t) \bmod 2 = 0 \\ x_{n,s}(t) \in \mathbb{Z}_{\geq 0} \\ 10 \leq p_{n,s}(t) \leq 30 \\ Q_k(t+100) = \max\left\{0,\ Q_k(t) + \sum_{\tau\in\mathcal{F}(t)} D_k(\tau) - S_k(t)\right\} \\ r_k(\tau),\ \gamma_k(\tau) \text{ 由 } (x,p,a) \text{ 与 } (\phi,h) \text{ 及调度生成} \\ \sum_{n\in\mathcal{N}} a_{n,k}(t) \leq 1 \\ \forall n \in \mathcal{N},\ t \in \mathcal{T},\ \forall n,k,s,t \end{cases} \tag{27}$$

其中 $\mathcal{A}_k(t) \subseteq \mathcal{F}(t)$ 为窗口 t 内属于用户 k 且在 SLA 内完成的任务到达时刻集合；$r_k(\tau)$ 与 $S_k(t)$ 均受干扰耦合与调度影响，是 (x,p,a) 的非线性函数。

该模型体现了“多站同频干扰 + 切片化 RB 分配 + 切片级功率控制 + 任务队列”的耦合特性，属于带整数约束与非凸干扰项的时变 MINLP 问题。

6.4 模型求解

针对第三问的模型，我们分析得到：1) 决策维度急剧扩张，包含了 9 个连续的功率变量 $\{p_{n,s}\}$ 与 48 个离散的用户接入基站变量 $\{a_{n,k}\}$，构成了庞大的混合搜索空间。2) 小区间干扰项使目标函数呈现强非凸、非线性特性，枚举或传统凸优化方法难以求解，计算量将呈指数级爆炸。故，前两问的“切片 RB 枚举 + 预测-控制 (MPC)”框架无法直接扩展到第三问。

为兼顾求解质量与计算效率，本文不再使用枚举仿真的方法，而是引入启发式算法，快速找到近似最优解。本问采用**“滚动时窗预测控制 (MPC)+ 混合编码遗传算法 (GA)”**的两层求解方法。

6.4.1 外层：滚动时窗预测控制 (MPC)

我们将 1000 ms 的总时长离散为 10 个长度为 $T_w = 100$ ms 的独立决策窗口。在每个窗口 t 的开始，我们求解一个静态优化问题，以确定该窗口内恒定不变的资源分配策略（包括用户接入、RB 切片、功率等级）。这种滚动优化的方式使得模型可以应对时变的信道和业务。

6.4.2 内层：混合编码遗传算法 (GA)

针对每个窗口内的静态资源分配问题，我们设计了遗传算法进行启发式搜索。GA能够有效处理高维、非凸、含混合变量的复杂优化问题。

- **个体编码方案：**每个个体（染色体）代表一个完整的资源分配策略，采用混合编码，由三部分组成：
  1. **用户接入决策：**长度为 48 的整数向量，每个基因位代表一个用户，其值（0, 1, 2）表示该用户接入的基站（BS1, BS2, BS3）。
  2. **RB 切片分配：**长度为 6 的整数向量，每两位表示一个基站为 URLLC 和 eMBB 切片分配的 RB 数量。mMTC 切片的比例则由剩余资源确定，并根据切片粒度约束进行解码。
  3. **切片功率控制：**长度为 9 的浮点数向量，每个基因位表示一个基站上特定切片（U/E/M）的发射功率（dBm），范围在 $[10, 30]$。
- **适应度函数：**个体的适应度由一个精细的仿真器评估。该仿真器将个体解码后的策略作为输入，以 1 ms 为步长模拟整个 100 ms 窗口的动态过程。仿真完整地计算了用户速率（包含所有基站的同频干扰）、任务队列的演化以及最终产生的 QoS 总分。该 QoS 总分即为个体的适应度。
- **遗传算子与参数：**GA 采用精英保留策略，并使用锦标赛选择、算术交叉（针对连续变量）/单点交叉（针对离散变量）和高斯/随机扰动变异（针对不同类型变量）来产生新一代种群。关键参数设置如下：种群大小为 40，最大进化代数为 200，精英个体保留 5 个，交叉概率 0.8，变异概率 0.3。

6.4.3 求解流程

本算法采用“滚动时窗预测控制”框架，将 1000ms 划分为 10 个独立的 100ms 优化窗口。每个窗口内，利用遗传算法（GA）进行静态资源分配优化，简要流程如图 3所示。

图 **3**　问题三“滚动 **MPC + GA**”求解流程图

6.5 结果分析

我们采用上一节提出的“滚动 MPC+GA”方法对问题三进行了求解，得到了 10 个决策窗口内每个基站的资源块（RB）分配、切片功率以及用户接入策略。详细的数值结果记录在代码附录的 `q3_user_bs_mapping.csv` 和 `q3_window_results.csv` 文件中。本节将对这些结果进行分析。

6.5.1 总体性能与动态适应性

在 1000ms 的仿真周期内，我们所提出的算法实现了 **871.34** 的总服务质量（QoS）得分。每个窗口的详细 QoS 得分、资源和功率分配决策汇总于表 3。

表 **3**　问题三各窗口资源分配决策与 **QoS** 结果

Win	BS1 (RB/P(dBm))			BS2 (RB/P(dBm))			BS3 (RB/P(dBm))			QoS			Obj.
	U	E	M	U	E	M	U	E	M	U	E	M	
0	0/24.4	40/22.8	10/24.8	20/21.6	0/22.9	30/23.4	20/21.6	0/22.9	30/21.7	38.16	8.40	30.00	76.56
1	10/21.1	0/21.0	40/25.6	10/20.2	20/24.2	20/20.9	20/21.7	20/24.3	10/23.2	51.03	6.81	30.00	87.84
2	10/21.9	0/20.1	40/25.1	0/18.9	40/25.4	10/22.7	20/20.5	0/24.8	30/21.0	38.35	9.87	30.00	78.22
3	30/19.9	0/22.5	20/18.0	20/20.4	0/20.3	30/22.4	0/19.4	30/23.7	20/20.2	49.26	7.79	30.00	87.05
4	10/23.2	20/26.2	20/24.0	30/21.6	0/20.7	20/21.6	10/20.5	0/21.8	40/22.2	46.43	5.90	30.00	82.33
5	0/19.2	30/24.1	20/21.4	20/21.3	0/21.3	30/20.2	30/22.4	0/21.3	20/22.8	52.02	7.28	30.00	89.30
6	10/21.9	0/21.3	40/20.6	10/22.5	0/21.5	40/22.7	20/21.3	20/25.5	10/21.0	52.08	6.25	30.00	88.33
7	10/21.6	0/22.9	40/22.0	10/19.2	20/26.2	20/17.5	20/24.7	0/21.5	30/20.6	48.68	6.40	30.00	85.08
8	20/22.5	0/18.6	30/22.9	20/18.7	0/17.9	30/23.3	10/22.3	20/29.8	20/23.0	51.45	7.59	30.00	89.04
9	0/21.4	30/27.3	20/22.7	30/24.5	0/22.0	20/20.7	10/21.6	0/23.2	40/21.1	56.80	7.88	30.00	94.69

结果显示，算法能够根据每个窗口变化的信道条件和任务到达情况，动态调整资源分配策略。

6.5.2 切片资源分配与干扰管理策略

遗传算法在每个窗口内搜索“用户接入 +RB 分配 + 功率控制”的联合最优解，其分配策略体现了对不同切片 QoS 需求的深刻理解和对干扰的精细化控制。

- **URLLC (U-slice)**：作为最高优先级的业务，其 QoS 得分（sum_U）在各个窗口都获得了较高的满足度。算法倾向于为其分配较高的发射功率（如窗口 8 中 BS3 的 29.8dBm），以克服路径损耗和干扰，确保其超低时延和高可靠性要求。当 URLLC 任务负载较重时（如窗口 9），算法会聚合多个基站的资源（BS2 分配 30 个 RB，BS3 分配 10 个 RB）来共同满足需求。
- **eMBB (E-slice)**：该切片追求高吞吐量，对 SINR 敏感。算法的决策体现了机会主义调度思想：当某个基站与某些 eMBB 用户之间的信道条件极好，且来自邻近基站的干扰较小时，会大胆地为其分配大量 RB 和较高的功率（如窗口 9 中 BS1 为 eMBB 分配 30 个 RB，功率高达 27.3dBm）。反之，若信道或干扰环境恶劣，则会减少资源分配，避免资源浪费。

- **mMTC (M-slice)**：该切片服务质量目标是“尽力而为”地满足所有用户的基本连接需求。从结果看，几乎所有窗口的 mMTC QoS 得分都达到了其上限 30 分，表明算法总能找到足够的“边角料”资源来满足海量但低速率的连接请求，实现了对网络资源的充分利用。

7 问题四的模型的建立和求解

7.1 问题四的描述与分析

本问引入宏基站（Macro BS, 记作 MBS）与多个微基站（Small BS, 记作 SBS）的异构蜂窝网络：MBS 具备更充裕的频谱资源且覆盖广；SBS 负责边缘热点的增强覆盖。题设指出 MBS 与所有 SBS 采用不重叠频谱，因此跨层无干扰；但各 SBS 之间同频复用，存在相互干扰。本问是一个“跨层接入 + 多站切片 + 切片级功率控制 + 队列与 SLA 约束 + SBS 间干扰耦合”的时变混合整数非凸优化问题（MINLP）。经过分析，上一问的模型框架在本问中得以延续，但需针对异构网络的特性进行调整。

7.2 模型建立

本问的模型延续之前的内容，但需要对信道与干扰模型、接入与调度规则、决策变量等进行调整，以适应宏基站与微基站的异构网络结构，相同的模型不再重复。

7.2.1 信道与干扰模型

本问的信道模型核心要素与前文一致，用户的瞬时接收功率 $p_{\text{rx},n\to k}(\tau)$ 与热噪声功率 $N_0(i_s)$ 的计算公式保持不变。我们在此基础上，重点阐述第四问独特的异构网络干扰模型。

根据题目设定，宏基站（MBS, $n=0$）与所有微基站（SBS, $n\in\mathcal{N}$）工作在不同频段，因此它们之间不存在跨层干扰。干扰仅存在于同频部署的各个 SBS 之间。用户的信干噪比（SINR）$\gamma_k(\tau)$ 因此取决于其所接入的基站类型：

- **当用户接入 MBS ($n=0$) 时**，不存在干扰，其接收质量由信噪比（SNR）决定：

$$\gamma_k(\tau)=\frac{p_{\text{rx},0\to k}(\tau)}{N_0(i_s)},\quad \text{若} a_{0,k}(t)=1 \tag{28}$$

- **当用户接入 SBS ($n\in\mathcal{N}$) 时**，会受到来自其他 SBS 的同频干扰。干扰功率 $I_{u\to k}(\tau)$ 来自于其他 SBS u ($u\in\mathcal{N}, u\neq n$) 在相同 RB 上的发射。此时，信干噪比为：

$$\gamma_k(\tau)=\frac{p_{\text{rx},n\to k}(\tau)}{\sum_{u\in\mathcal{N},u\neq n} I_{u\to k}(\tau)+N_0(i_s)},\quad \text{若} a_{n,k}(t)=1, n\in\mathcal{N} \tag{29}$$

其中，干扰项 $I_{u\to k}(\tau)$ 的计算方式与信号功率 p_{rx} 类似。

综合以上两种情况，用户 k 在时刻 τ 的瞬时数据传输速率（bps）可由香农公式计算得出：

$$r_k(\tau) = i_s \cdot b \cdot \log_2\left(1 + \gamma_k(\tau)\right) \tag{30}$$

7.2.2 接入与调度规则

本问的接入与调度在继承前问规则的基础上，引入了针对异构网络的特定限制：

- **接入限制：**这是本问的核心特征。每个用户 k 的接入选择被严格限定在宏基站（MBS）与地理位置最近的微基站（SBS）之间，即 $a_{n,k}(t) = 1$ 仅在 $n = 0$ 或 $n = n^*(k)$ 时才可能成立。
- **并发与调度：**各基站（MBS 与 SBS）的切片并发容量 $C_{n,s}(t)$ 的计算方式，以及窗口内“编号优先、任务完成即补位、URLLC 按紧迫度优先”的调度机制，与前文保持一致。

7.2.3 决策变量与优化模型

决策变量：

- RB 切片分配：$x_{n,s}(t) \in \mathbb{Z}_{\geq 0}$
- 发射功率：$p_{0,s}(t) \in [10, 40]$ (dBm)，$p_{n,s}(t) \in [10, 30]$ (dBm)
- 接入关联：$a_{n,k}(t) \in \{0, 1\}$, 当 $a_{n,k}(t) = 1$, 则 k 在窗口 t 仅由站 n 调度；若 $n \notin \{0, n^*(k)\}$, $a_{n,k}(t) = 0$

综合上述要素，第四问的动态联合优化模型可表述为（跨 10 个窗口聚合）：

$$\max_{\{x,p,a\}} \quad Q_{\text{total}} = \sum_{t\in\mathcal{T}} \left[\sum_{k\in\mathcal{K}_U} \sum_{\tau\in\mathcal{A}_k(t)} y^U_{k,\tau} + \sum_{k\in\mathcal{K}_e} \sum_{\tau\in\mathcal{A}_k(t)} y^e_{k,\tau} + \sum_{k\in\mathcal{K}_m} \sum_{\tau\in\mathcal{A}_k(t)} y^m_{k,\tau} \right] \tag{31}$$

$$\text{s.t.} \quad \begin{cases} \displaystyle\sum_{s\in\mathcal{S}} x_{n,s}(t) = R_n \\ x_{n,U}(t) \bmod 10 = 0,\ x_{n,e}(t) \bmod 5 = 0,\ x_{n,m}(t) \bmod 2 = 0 \\ x_{n,s}(t) \in \mathbb{Z}_{\geq 0} \\ 10 \leq p_{0,s}(t) \leq 40,\ 10 \leq p_{n,s}(t) \leq 30\ (\forall n \in \mathcal{N}) \\ Q_k(t+100) = \max\left\{0,\ Q_k(t) + \displaystyle\sum_{\tau\in\mathcal{F}(t)} D_k(\tau) - S_k(t)\right\} \\ r_k(\tau),\ \gamma_k(\tau) \text{ 由 } (x, p, a) \text{ 与 } (\phi, h) \text{ 及调度生成} \\ \displaystyle\sum_{n\in\bar{\mathcal{N}}} a_{n,k}(t) \leq 1 \\ \forall n \in \bar{\mathcal{N}}, s \in \mathcal{S}, t \in \mathcal{T},\ \forall k, t \end{cases} \tag{32}$$

其中 $\mathcal{A}_k(t) \subseteq \mathcal{F}(t)$ 为窗口 t 内属于用户 k 且在 SLA 内完成的任务到达时刻集合。
该模型体现了“跨层接入选择 + 多站切片 + 切片级功率 + SBS 间互扰 + 任务队列”的耦合，属于时变 MINLP。

7.3 模型求解

第四问的优化模型在第三问的基础上引入了宏基站 (MBS) 与微基站 (SBS) 的异构结构，并增加了用户接入选择的约束。模型的决策变量维度、干扰关系和资源边界均发生了显著变化，使其成为一个更复杂的混合整数非线性规划 (MINLP) 问题。直接求解该问题在计算上是不可行的。

我们延续并拓展了问题三中**“滚动时窗预测控制 (MPC) + 混合编码遗传算法 (GA)”**的求解框架。该框架的总体思想和流程与问题三 (见图 3) 保持一致，即通过 MPC 将长时域问题分解为一系列独立的百毫秒窗口优化问题，再利用 GA 对每个窗口内的“用户接入 + 资源切片 + 功率控制”联合优化问题进行高效搜索。本节将重点阐述为适配第四问场景而对 GA 内核所做的关键修改，与问题三重复之处不再赘述。

7.3.1 内层：混合编码遗传算法 (GA) 的适配

针对每个决策窗口内的静态资源分配问题，我们对遗传算法的编码、适应度评估和算子进行了如下调整：

- **个体编码方案**：每个个体 (染色体) 代表一个完整的资源分配策略，其混合编码结构调整如下：
  1. **用户接入决策**：根据模型约束，每个用户在每个窗口只能选择接入宏基站 (MBS) 或距离其最近的一个微基站 (SBS)。因此，该部分编码为一个长度为 48 的**二进制向量**，其中基因位 i 的值为 0 代表用户 i 接入 MBS，为 1 则代表其接入该窗口起始时刻的最近 SBS。这相比问题三的接入决策空间有了大幅简化。
  2. **RB 切片分配**：编码为一个长度为 $2 \times 4 = 8$ 的整数向量。由于存在 1 个 MBS 和 3 个 SBS，共 4 个基站，我们为每个基站的 URLLC 和 eMBB 切片分配 RB 数量。MBS 的总 RB 数为 100，每个 SBS 为 50。mMTC 切片的 RB 数仍由总数减去 U/E 切片后剩余的资源确定，并向下对齐到切片粒度。
  3. **切片功率控制**：编码为一个长度为 $4 \times 3 = 12$ 的浮点数向量，分别表示 4 个基站上 3 个切片的发射功率 (dBm)。MBS 的功率范围为 $[10, 40]$ dBm，而 SBS 的功率范围为 $[10, 30]$ dBm。
- **适应度函数**：个体的适应度评估依然通过一个精细的、步长为 1 ms 的窗口仿真器来完成。该仿真器根据解码后的策略进行模拟，但其核心的速率计算模块进行了关键更新：

  - **异构干扰模型**：仿真器精确地实现了第四问的干扰关系。MBS 与 SBS 工作在不同频段，因此 MBS 上的用户不受任何来自 SBS 的干扰。反之，所有 SBS 在同频段工作，因此一个 SBS 上的用户会受到其他所有**正在服务的、占用同类切片**的 SBS 的同道干扰。
  - **动态接入点**：在每个窗口开始时，会根据所有用户当时的坐标预先计算出各自的“最近 SBS”。GA 在解码接入决策时，将基于这一预计算结果来确定用户的具体接入基站。

  最终，仿真器输出的 QoS 总分作为该个体的适应度值。
- **遗传算子与参数**：我们沿用了问题三中的精英保留策略、锦标赛选择、混合交叉 (单点交叉用于离散的接入决策，算术交叉用于连续的功率和 RB 变量) 以及多模式变异 (随机扰动或高斯扰动)。为应对更复杂的问题，参数调整为：种群大小为 50，最大进化代数为 500，交叉概率 0.8，变异概率 0.3。

7.4 结果分析

我们采用上一节提出的“滚动 MPC+GA”方法对问题四进行了求解，得到了 10 个决策窗口内每个基站的资源块（RB）分配、切片功率以及用户接入策略。详细的数值结果记录在代码附录的 `q4_user_bs_mapping.csv` 和 `q4_window_results.csv` 文件中。本节将对这些结果进行分析。

7.4.1 总体性能与动态适应性

在 1000ms 的仿真周期内，我们所提出的算法实现了 **834.02** 的总服务质量（QoS）得分。每个窗口的详细 QoS 得分、资源和功率分配决策汇总于表 4。

从表 4可以看出，算法在不同时间窗口展现出高度的动态适应性。

7.4.2 宏基站与微基站的协同策略

- **宏基站（MBS）的角色**：MBS 拥有 100 个 RB 和更高的功率上限（40dBm），且无干扰，是网络中的“定海神针”。算法倾向于利用 MBS 来处理大规模、可容忍一定时延的业务。从窗口 1 到 9，MBS 几乎将全部资源投入到 mMTC 切片，确保了 mMTC 业务在大多数窗口都能拿到满分（40 分），这体现了利用 MBS 广覆盖、大容量优势来满足海量连接的策略。
- **微基站（SBS）的角色**：SBS 资源有限（50 个 RB）且存在同频干扰，但其部署更靠近用户，能提供更好的信道条件。算法主要利用 SBS 来保障对时延和速率要求苛刻的业务。例如，在窗口 3，当 MBS 全力服务 mMTC 时，三个 SBS 协同为 URLLC 用户提供了总计 80 个 RB 的资源，有效保障了高优先级业务的 QoS。此外，SBS 的功

表 **4**　问题四各窗口资源分配决策与 **QoS** 结果（部分）

Win	MBS_1 (RB/P(dBm))			SBS_1 (RB/P(dBm))			SBS_2 (RB/P(dBm))			QoS			Obj.
	U	E	M	U	E	M	U	E	M	U	E	M	
0	30/25.1	40/23.8	30/24.6	20/24.6	20/25.8	10/25.0	20/25.0	0/27.8	30/26.2	93.42	14.26	40.00	147.68
1	30/25.7	0/25.2	70/26.9	0/25.5	0/27.5	50/24.1	30/25.9	0/22.3	20/24.0	85.68	0.00	40.00	125.68
2	50/25.8	0/24.1	50/22.9	0/28.1	0/26.0	50/23.3	0/19.9	0/26.8	50/25.1	64.61	0.00	40.00	104.61
3	0/22.5	0/22.7	100/23.3	30/21.0	0/24.7	20/21.1	30/26.4	0/21.9	20/25.7	50.85	0.00	40.00	90.85
4	0/24.9	0/24.8	100/27.8	0/25.5	0/29.2	50/21.0	30/23.7	0/25.9	20/24.1	43.16	0.00	40.00	83.16
5	0/21.2	0/26.5	100/25.9	30/30.0	0/25.9	20/23.1	30/25.6	0/24.7	20/27.0	50.27	0.00	40.00	90.27
6	0/22.0	0/29.6	100/22.4	20/25.8	0/24.9	30/24.1	30/27.6	0/22.2	20/25.5	47.51	0.00	40.00	87.51
7	0/27.4	0/24.9	100/29.0	20/26.3	0/28.5	30/20.9	30/25.1	0/29.4	20/26.5	48.32	0.00	34.10	82.42
8	0/23.2	0/22.8	100/26.9	10/24.2	0/21.9	40/29.4	20/23.8	0/26.8	30/23.9	19.48	0.00	10.03	29.50
9	0/22.3	0/23.9	100/29.8	30/22.3	0/25.6	20/26.0	50/23.9	0/28.4	0/29.3	11.74	0.00	-19.38	-7.64

率控制也十分灵活，例如在窗口 5，SBS_1 为 URLLC 切片输出了 30dBm 的满功率，以对抗干扰、确保可靠性。

8　问题五的模型的建立和求解

8.1 问题五的描述与分析

本问在第四问的异构网络场景和 QoS 评估框架基础上，引入了基站的能耗模型，旨在探索网络能耗的最小化的基础上，保障用户服务质量。这是一个典型的两阶段多目标优化问题。为此，我们设计了一个两阶段的优化求解框架。

8.2 预备工作

本问的集合、索引与大部分参数均与第四问保持一致。

- 基站集合：$\bar{\mathcal{N}} = \{0\} \cup \mathcal{N}$，其中 0 表示 MBS，$\mathcal{N} = \{1, 2, 3\}$ 表示 SBS 集合。
- 切片集合：$\mathcal{S} = \{U, e, m\}$。
- 用户集合：$\mathcal{K} = \mathcal{K}_U \cup \mathcal{K}_e \cup \mathcal{K}_m$。
- 决策时刻集合：$\mathcal{T} = \{0, 100, \ldots, 900\}$ ms。

新增的能耗模型相关参数（基于附录 6）如下：

- 固定功耗：$P_{static} = 28\,\mathrm{W}$。
- 每 RB 激活功耗系数：$\delta = 0.75\,\mathrm{W/RB}$。
- 功率放大器效率：$\eta = 0.35$。

8.3 模型建立

8.3.1 能耗模型

基于附录 6，每个基站 $n \in \bar{\mathcal{N}}$ 在任意时刻 τ 的总功耗 $P_n(\tau)$ 由三部分组成：

$$P_n(\tau) = P_{static} + P_{RB,n}(\tau) + P_{tx,n}(\tau) \tag{33}$$

其中：

- **固定功耗** P_{static}：基站基础运行所需功耗。
- **RB 激活功耗** $P_{RB,n}(\tau) = \delta \cdot N_{active,n}(\tau)$，与该时刻激活的 RB 总数 $N_{active,n}(\tau)$ 成正比。$N_{active,n}(\tau)$ 是基站 n 在时刻 τ 上服务所有用户的 RB 数量之和。
- **发射功耗** $P_{tx,n}(\tau) = \frac{1}{\eta} P_{transmit,n}(\tau)$，与总发射功率 $P_{transmit,n}(\tau)$ 成正比。$P_{transmit,n}(\tau)$ 是基站 n 在时刻 τ 上服务所有用户的发射功率之和，单位为瓦特 (W)。

全网在时刻 τ 的总功耗为 $P_{total}(\tau) = \sum_{n \in \bar{\mathcal{N}}} P_n(\tau)$。在一个决策窗口 t（时长为 $T_w = 100$ms）内的总能耗（单位：焦耳）为：

$$E_{total}(t) = \int_{t}^{t+T_w} P_{total}(\tau) d\tau \tag{34}$$

8.3.2 两阶段优化模型

为实现“优先最小化能耗，再最大化 QoS”的目标，我们将原问题分解为两个独立的子问题，在每个决策窗口 $t \in \mathcal{T}$ 内依次进行优化。该方法的核心思想是，首先在满足基本通信需求的前提下，确定最优的节能功率配置；然后在此功率配置下，通过精细化的资源块（RB）分配来最大化用户服务质量（QoS）。

第一阶段：能耗最小化 此阶段的核心目标是，在用户接入和 RB 分配策略固定的情况下，通过调整各基站各切片的发射功率，找到使得窗口总能耗 $E_{total}(t)$ 最小的功率分配方案 $\{p_{n,s}(t)\}$。该阶段的优化模型可表述为：

$$\begin{aligned} \min_{\{p_{n,s}(t)\}} \quad & E_{total}\left(t; \{p_{n,s}(t)\}\right) \\ \text{s.t.} \quad & \begin{cases} 10 \le p_{0,s}(t) \le 40, & \forall s \in \mathcal{S} \\ 10 \le p_{n,s}(t) \le 30, & \forall n \in \mathcal{N}, s \in \mathcal{S} \end{cases} \end{aligned} \tag{35}$$

决策变量仅为各基站各切片的发射功率 $p_{n,s}(t)$。

第二阶段：QoS 最大化 在第一阶段得到最优功率分配方案 $p^*_{n,s}(t)$ 后，此阶段的目标是在此节能功率配置下，通过优化 RB 的切片划分方案 $\{x_{n,s}(t)\}$，来最大化全网用户的总

服务质量 $Q_{total}(t)$。该阶段的优化模型表述为：

$$\max_{\{x_{n,s}(t)\}} \quad Q_{total}\left(t;\{x_{n,s}(t)\},\{p^*_{n,s}(t)\}\right)$$
$$\text{s.t.} \quad \begin{cases} \sum_{s\in\mathcal{S}} x_{n,s}(t) = R_n, \quad \forall n \in \bar{\mathcal{N}} \\ x_{n,U}(t) \bmod 10 = 0, \quad \forall n \in \bar{\mathcal{N}} \\ x_{n,e}(t) \bmod 5 = 0, \quad \forall n \in \bar{\mathcal{N}} \\ x_{n,m}(t) \bmod 2 = 0, \quad \forall n \in \bar{\mathcal{N}} \\ x_{n,s}(t) \in \mathbb{Z}_{\geq 0}, \quad \forall n \in \bar{\mathcal{N}}, s \in \mathcal{S} \end{cases} \tag{36}$$

决策变量为各基站为不同业务切片分配的 RB 数量 $x_{n,s}(t)$。目标函数 Q_{total} 的评估方式与第四问一致，均通过动态仿真得出，但此时是在给定的节能功率模式下进行。

8.4 模型求解

我们延续“滚动时域控制 (MPC)”框架，结合第二问的枚举仿真和第三四问的遗传算法，在每个 100ms 的窗口内，依次执行上述两阶段优化。

第一阶段求解：遗传算法 (GA) 由于能耗与功率之间存在复杂的非线性关系，且其评估依赖于精细的动态仿真，我们采用遗传算法（GA）求解第一阶段的功率优化问题。

- **编码**：个体仅编码各基站各切片的功率值，为一个长度为 $4 \times 3 = 12$ 的浮点数数组。
- **适应度函数**：适应度被定义为窗口内总能耗的负值，即 $f = -E_{total}$。该能耗值通过调用一个 1ms 步长的仿真器，在固定的接入和 RB 分配下模拟整个窗口得出。
- **遗传算子**：采用锦标赛选择、算术交叉和高斯扰动变异。

第二阶段求解：枚举仿真 该阶段方法与第二问大致相同，区别在于第四问开始增加了宏基站，在枚举时需要同时考虑宏基站和微基站的 RB 分配。

8.5 结果分析

我们采用上一节提出的两阶段优化方法对问题五进行了求解。在 1000ms 的总仿真时长内，实现了 **183.68 J** 的总能耗，以及 **428.60** 的总服务质量（QoS）得分。每个决策窗口的详细结果汇总于表 5。

8.5.1 节能策略分析

第一阶段的 GA 优化展现了智能的功率控制策略以实现节能：

表 **5**　问题五各窗口资源分配决策与性能结果

Win	MBS_1 (RB/P(dBm))			SBS_1 (RB/P(dBm))			QoS	Energy (J)
	U	E	M	U	E	M		
0	0/23.9	0/27.1	100/25.1	10/14.9	10/10.0	30/10.0	95.75	19.68
1	0/27.8	0/21.1	100/17.6	0/21.8	0/10.0	50/10.0	72.13	18.10
2	0/27.9	0/24.5	100/25.6	0/18.7	0/10.0	50/10.0	40.37	18.12
3	0/29.1	0/25.6	100/27.0	0/17.8	0/10.0	50/10.0	92.88	18.25
4	0/23.8	0/20.5	100/32.0	0/10.0	0/10.0	50/10.0	61.77	17.97
5	0/25.2	0/30.8	100/18.2	0/10.0	0/10.0	50/10.0	83.54	18.10
6	0/26.8	0/25.6	100/23.2	0/10.0	10/10.0	40/10.0	13.49	19.48
7	0/21.0	0/24.9	100/34.7	0/10.0	0/10.0	50/10.0	3.64	18.09
8	0/26.6	0/24.4	100/25.0	0/10.0	0/10.0	50/10.0	-11.53	17.84
9	0/29.2	0/31.5	100/35.9	0/10.0	0/10.0	50/10.0	-23.45	18.05

注：表格仅展示了 MBS_1 和 SBS_1 的决策，完整决策见附录代码输出。

- **功率的精细化控制：**与问题四中功率普遍较高的设定不同，本问中的功率分配显示出极大的差异性。例如，在窗口 0，SBS_1 为 eMBB 和 mMTC 切片仅分配了 10.0dBm 的最低功率，而在窗口 5，MBS 为 eMBB 切片分配了高达 30.8dBm 的功率。这表明算法并非简单地“一刀切”式降低所有功率，而是根据不同切片、不同基站的信道和业务情况，有选择地、精细地调整功率，以最小的能耗代价满足基本通信需求。
- **向低功耗基站/切片倾斜：**在固定 RB 分配下，GA 倾向于为信道条件好的用户（通常离基站近）分配较低的功率，而为信道条件差的用户分配较高的功率，以保证所有用户的基本连通性。同时，对于非关键业务或负载较轻的切片，算法会果断采用最低功率策略，从而最大化节能效果。

综上，本问提出的两阶段优化框架成功地实现了在保障基本服务的前提下最小化网络能耗的目标。结果清晰地量化了能耗与服务质量之间的权衡关系，并验证了通过精细化功率控制实现节能的有效性。

9 模型的评价

9.1 模型的优点

- 优点 1
- 优点 2
- 优点 3

9.2 模型的缺点

- 缺点 1
- 缺点 2

参考文献

[1] 司守奎, 孙玺菁. 数学建模算法与应用[M]. 北京: 国防工业出版社, 2011.

[2] 卓金武. MATLAB 在数学建模中的应用[M]. 北京: 北京航空航天大学出版社, 2011.

附录 A　文件列表

文件名	功能描述
q1.py	问题一：单基站枚举法求解器
q2.py	问题二：单微基站 MPC 滚动窗口求解器
q3.py	问题三：多基站 GA-MPC 求解器（同频干扰 + 功率控制）
q4.py	问题四：异构网络 GA-MPC 求解器（MBS+SBS，接入受限）
q5.py	问题五：能耗优化 GA-MPC 求解器（二阶段：功率优先 +RB 枚举）
$q1_enum_results.csv$	问题一求解结果
$q2_enum_results_all.csv$	问题二求解结果
$q3_window_results.csv$	问题三求解结果
$q4_window_results.csv$	问题四求解结果
$q5_window_results.csv$	问题五求解结果

附录 B　代码

q1.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import csv
import math
import os
from dataclasses import dataclass
from typing import Dict, List

# 系统常量
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
ROOT_DIR = os.path.abspath(os.path.join(SCRIPT_DIR, "..", ".."
    ))
ATTACH_DIR = os.path.join(ROOT_DIR, "题目", "附件", "附件1")

CSV_TASK = os.path.join(ATTACH_DIR, "q1_任务流.csv")
```

```
CSV_PL = os.path.join(ATTACH_DIR, "q1_大规模衰减.csv")
CSV_RAY = os.path.join(ATTACH_DIR, "q1_小规模瑞丽衰减.csv")

P_TX_DBM = 20.0
B_HZ = 360_000.0
NF_DB = 7.0
V_U, V_E, V_M = 10, 5, 2
ALPHA, M_U, M_E, M_M = 0.95, 5.0, 3.0, 1.0
SLA_L_U_MS, SLA_L_E_MS, SLA_L_M_MS = 5.0, 100.0, 500.0
SLA_R_E_MBPS = 50.0

@dataclass
class User:
    name: str
    category: str
    data_mbit: float
    pl_db: float
    h_abs: float

def load_csv_users() -> List[User]:
    """加载CSV并构建用户最小所需信息"""
    with open(CSV_TASK, "r", encoding="utf-8") as f:
        task_reader = csv.DictReader(f)
        tasks = {row["用户"]: float(row["任务数据量(Mbit)"])
   for row in task_reader}

    phi_db: Dict[str, float] = {}
    with open(CSV_PL, "r", encoding="utf-8") as f:
        pl_reader = csv.DictReader(f)
        for row in pl_reader:
            phi_db[row["用户"]] = float(row["衰减(dB)"])

    h_raw: Dict[str, float] = {}
    with open(CSV_RAY, "r", encoding="utf-8") as f:
        ray_reader = csv.DictReader(f)
```

```
        for row in ray_reader:
            h_raw[row["用户"]] = float(row["衰减(dB)"])

    users: List[User] = []
    for name, data_mbit in tasks.items():
        category = "U" if name.startswith("U") else ("E" if name.startswith("e") else "M")
        pl = phi_db.get(name, 100.0)
        h_val = h_raw.get(name, 0.0)
        # CSV中瑞丽一般以dB给出（负值），取功率/幅度绝对值的近似
        h_abs = math.sqrt(10 ** (h_val / 10.0)) if h_val <= 0 else h_val
        users.append(User(name, category, data_mbit, pl, h_abs))
    return users

def dbm_to_mw(dbm: float) -> float:
    return 10 ** (dbm / 10.0)

def noise_power_mw(num_rbs: int) -> float:
    if num_rbs <= 0:
        return 1e-30
    n0_dbm = -174.0 + 10.0 * math.log10(num_rbs * B_HZ) + NF_DB
    return dbm_to_mw(n0_dbm)

def user_rate_bps(user: User, num_rbs: int) -> float:
    p_rx = 10 ** ((P_TX_DBM - user.pl_db) / 10.0) * (user.h_abs * user.h_abs)
    n0 = noise_power_mw(num_rbs)
    snr = p_rx / n0
    return num_rbs * B_HZ * math.log2(1.0 + snr)

def delay_ms(user: User, num_rbs: int) -> float:
```

```
    r_bps = user_rate_bps(user, num_rbs)
    if r_bps <= 0.0:
        return float("inf")
    return (user.data_mbit * 1e6) / r_bps * 1e3

def urllc_qos_from_L(L_ms: float) -> float:
    return ALPHA ** L_ms if L_ms <= SLA_L_U_MS else -M_U

def embb_qos_from_L_and_r(L_ms: float, r_bps: float) -> float:
    r_mbps = r_bps / 1e6
    if L_ms <= SLA_L_E_MS and r_mbps >= SLA_R_E_MBPS:
        return 1.0
    elif L_ms <= SLA_L_E_MS and r_mbps < SLA_R_E_MBPS:
        return max(0.0, r_mbps / SLA_R_E_MBPS)
    return -M_E

def schedule_slice(users: List[User], num_rbs_slice: int,
   per_user_rbs: int) -> Dict[str, Dict[str, float]]:
    """切片内串并行调度"""
    import heapq
    results = {}
    cap = num_rbs_slice // per_user_rbs if per_user_rbs > 0
   else 0
    if cap <= 0:
        for u in users:
            r_bps = user_rate_bps(u, per_user_rbs) if
   per_user_rbs > 0 else 0.0
            results[u.name] = {"Q_ms": float("inf"), "T_ms":
   delay_ms(u, per_user_rbs) if per_user_rbs > 0 else float("
   inf"), "L_ms": float("inf"), "r_bps": r_bps}
        return results

    active_heap = []
    counter = 0
    idx = 0
```

```
    n = len(users)

    # 填满初始并发槽位
    while idx < min(cap, n):
        u = users[idx]
        T_ms = delay_ms(u, per_user_rbs)
        Q_ms = 0.0
        L_ms = Q_ms + T_ms
        r_bps = user_rate_bps(u, per_user_rbs)
        results[u.name] = {"Q_ms": Q_ms, "T_ms": T_ms, "L_ms": L_ms, "r_bps": r_bps}
        heapq.heappush(active_heap, (L_ms, counter))
        counter += 1
        idx += 1

    # 其余用户按完成最早者释放后接续
    while idx < n:
        u = users[idx]
        earliest_finish, _ = heapq.heappop(active_heap)
        T_ms = delay_ms(u, per_user_rbs)
        start_ms = earliest_finish
        Q_ms = start_ms
        L_ms = Q_ms + T_ms
        r_bps = user_rate_bps(u, per_user_rbs)
        results[u.name] = {"Q_ms": Q_ms, "T_ms": T_ms, "L_ms": L_ms, "r_bps": r_bps}
        heapq.heappush(active_heap, (L_ms, counter))
        counter += 1
        idx += 1
    return results

def enumerate_solution(users: List[User]) -> Dict[str, object]:
    """核心：枚举RB切片分配并计算目标值，返回最优方案"""
    def sort_key(u: User):
```

```
        pr = 0 if u.category == "U" else (1 if u.category == "E" else 2)
        num = int(u.name[1:]) if u.name[1:].isdigit() else 0
        return (pr, num)

    U = sorted([u for u in users if u.category == "U"], key=sort_key)
    E = sorted([u for u in users if u.category == "E"], key=sort_key)
    M = sorted([u for u in users if u.category == "M"], key=sort_key)

    best = {"obj": -1e18, "R": (0, 0, 0)}

    for R_U in range(0, 51, V_U):
        for R_E in range(0, 51 - R_U, V_E):
            R_M = 50 - R_U - R_E
            if R_M < 0 or R_M % V_M != 0:
                continue

            U_sched = schedule_slice(U, R_U, V_U)
            E_sched = schedule_slice(E, R_E, V_E)
            M_sched = schedule_slice(M, R_M, V_M)

            sum_U = sum(urllc_qos_from_L(U_sched[u.name]["L_ms"]) for u in U)
            sum_E = sum(embb_qos_from_L_and_r(E_sched[e.name]["L_ms"], E_sched[e.name]["r_bps"]) for e in E)

            denom_M = sum(1 for m in M if m.data_mbit > 0.0)
            num_success = sum(1 for m in M if m.data_mbit > 0.0 and M_sched[m.name]["L_ms"] <= SLA_L_M_MS)
            ratio = (num_success / denom_M) if denom_M > 0 else 0.0
            sum_M = sum((ratio if M_sched[m.name]["L_ms"] <=
```

```
    SLA_L_M_MS else -M_M) for m in M if m.data_mbit > 0.0)

            obj = sum_U + sum_E + sum_M
            if obj > best["obj"]:
                best["obj"] = obj
                best["R"] = (R_U, R_E, R_M)
    return best

def main():
    users = load_csv_users()
    result = enumerate_solution(users)
    R_U, R_E, R_M = result["R"]
    print(f"最优RB分配: R_U={R_U}, R_E={R_E}, R_M={R_M}")
    print(f"目标函数: {result['obj']:.4f}")

if __name__ == "__main__":
    main()
```

q2.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""问题二: 单微基站MPC滚动窗口求解器"""

import csv
import math
import os
from dataclasses import dataclass, field
from typing import Dict, List, Tuple, Deque
from collections import deque

# 系统常量
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
ROOT_DIR = os.path.abspath(os.path.join(SCRIPT_DIR, "..", ".."
    ))
ATTACH_DIR = os.path.join(ROOT_DIR, "题目", "附件", "附件2")
```

```
16
17  P_TX_DBM = 30.0
18  B_HZ = 360_000.0
19  NF_DB = 7.0
20  V_U, V_E, V_M = 10, 5, 2
21  ALPHA, M_U, M_E, M_M = 0.95, 5.0, 3.0, 1.0
22  SLA_L_U_MS, SLA_L_E_MS, SLA_L_M_MS = 5.0, 100.0, 500.0
23  SLA_R_E_MBPS = 50.0
24  WINDOW_MS, TOTAL_MS = 100, 1000
25
26  @dataclass
27  class Chunk:
28      arrival_ms: int
29      size_mbit: float
30      remain_mbit: float
31      start_ms: int | None = None
32      finish_ms: int | None = None
33
34  @dataclass
35  class UserState:
36      name: str
37      category: str
38      queue: Deque[Chunk] = field(default_factory=deque)
39
40      def has_backlog(self) -> bool:
41          return len(self.queue) > 0 and self.queue[0].
    remain_mbit > 1e-15
42
43  @dataclass
44  class Env:
45      time_list: List[float]
46      arrivals_mbit: Dict[str, List[float]]
47      pl_db: Dict[str, List[float]]
48      ray_raw: Dict[str, List[float]]
49      small_scale_is_db: bool
```

```
50
51      def get_phi_db(self, name: str, t_ms: int) -> float:
52          arr = self.pl_db.get(name)
53          if arr is None:
54              return 100.0
55          t_ms = max(0, min(t_ms, len(arr) - 1))
56          return float(arr[t_ms])
57
58      def get_h_pow(self, name: str, t_ms: int) -> float:
59          arr = self.ray_raw.get(name)
60          if arr is None or len(arr) == 0:
61              return 1.0
62          t_ms = max(0, min(t_ms, len(arr) - 1))
63          val = float(arr[t_ms])
64          if self.small_scale_is_db:
65              return 10 ** (val / 10.0)
66          return val * val if val >= 0 else 1e-6
67
68  def load_time_series_csv(path: str) -> Tuple[List[float], Dict
    [str, List[float]]]:
69      """读取时间序列CSV"""
70      time_list = []
71      series = {}
72      with open(path, "r", encoding="utf-8-sig") as f:
73          reader = csv.DictReader(f)
74          for row in reader:
75              t_str = row.get("Time", row.get("time", row.get("
    TIME")))
76              if t_str is None:
77                  continue
78              try:
79                  t_val = float(t_str)
80              except ValueError:
81                  continue
82              time_list.append(t_val)
```

```
            for key, val in row.items():
                k = key.strip()
                if k.lower() == "time":
                    continue
                if val is None or val == "":
                    series.setdefault(k, []).append(0.0)
                    continue
                try:
                    v = float(val)
                except ValueError:
                    v = 0.0
                series.setdefault(k, []).append(v)
    # 对齐长度
    max_len = len(time_list)
    for k, arr in series.items():
        if len(arr) < max_len:
            arr.extend([0.0] * (max_len - len(arr)))
    return time_list, series

def build_env() -> Tuple[Env, List[str], List[str], List[str]]:
    """构建环境和用户列表"""
    time_list, arrivals = load_time_series_csv(os.path.join(ATTACH_DIR, "q2_用户任务流.csv"))
    _, pl = load_time_series_csv(os.path.join(ATTACH_DIR, "q2_大规模衰减.csv"))
    _, ray = load_time_series_csv(os.path.join(ATTACH_DIR, "q2_小规模瑞丽衰减.csv"))

    # 检测小规模衰减是否为dB
    small_is_db = any(v < 0 for arr in ray.values() for v in arr[:200])

    def sort_key(name: str) -> Tuple[int, int]:
        prefix_rank = 0 if name.startswith("U") else (1 if
```

```
    name.startswith("e") else 2)
113         num = int(name[1:]) if name[1:].isdigit() else 0
114         return (prefix_rank, num)
115
116     all_names = list(arrivals.keys())
117     U_names = sorted([n for n in all_names if n.startswith("U"
    )], key=sort_key)
118     E_names = sorted([n for n in all_names if n.startswith("e"
    )], key=sort_key)
119     M_names = sorted([n for n in all_names if n.startswith("m"
    )], key=sort_key)
120
121     env = Env(time_list=time_list, arrivals_mbit=arrivals,
    pl_db=pl, ray_raw=ray, small_scale_is_db=small_is_db)
122     return env, U_names, E_names, M_names
123
124 def user_rate_bps(env: Env, name: str, t_ms: int, num_rbs: int
    ) -> float:
125     """计算用户传输速率"""
126     if num_rbs <= 0:
127         return 0.0
128     phi_db = env.get_phi_db(name, t_ms)
129     h_pow = env.get_h_pow(name, t_ms)
130     p_rx_mw = 10 ** ((P_TX_DBM - phi_db) / 10.0) * h_pow
131     n0_mw = 10 ** ((-174.0 + 10.0 * math.log10(num_rbs * B_HZ)
    + NF_DB) / 10.0)
132     snr = p_rx_mw / max(n0_mw, 1e-30)
133     return num_rbs * B_HZ * math.log(1.0 + snr, 2)
134
135 def simulate_window(env: Env, users: Dict[str, UserState],
    U_order: List[str], E_order: List[str], M_order: List[str],
    t0: int, nU: int, nE: int, nM: int):
136     """仿真100ms窗口"""
137     from copy import deepcopy
138     st = {name: UserState(name=u.name, category=u.category,
```

```
    queue=deque(deepcopy(list(u.queue))) for name, u in users.
    items()}
139
140     capU = nU // V_U if V_U > 0 else 0
141     capE = nE // V_E if V_E > 0 else 0
142     capM = nM // V_M if V_M > 0 else 0
143
144     activeU, activeE, activeM = [], [], []
145     m_had_arrival = set()
146     m_success_users = set()
147
148     sum_U = sum_E = sum_m_score = 0.0
149
150     t1 = min(t0 + WINDOW_MS, TOTAL_MS)
151     for t in range(t0, t1):
152         # 1) 到达处理
153         for name, arr_series in env.arrivals_mbit.items():
154             if t < len(arr_series):
155                 vol = arr_series[t]
156                 if vol > 0.0:
157                     st[name].queue.append(Chunk(arrival_ms=t,
    size_mbit=vol, remain_mbit=vol))
158                     if name.startswith("m") and t0 <= t < t1:
159                         m_had_arrival.add(name)
160
161         # 2) 填充并发槽位
162         def fill_active(order: List[str], active: List[str],
    cap: int):
163             active[:] = [nm for nm in active if st[nm].
    has_backlog()]
164             for nm in order:
165                 if len(active) >= cap or nm in active:
166                     continue
167                 if st[nm].has_backlog():
168                     active.append(nm)
```

```
169
170         fill_active(U_order, activeU, capU)
171         fill_active(E_order, activeE, capE)
172         fill_active(M_order, activeM, capM)
173
174         # 3) 服务处理
175         def serve_one(name: str, per_user_rbs: int):
176             if not st[name].has_backlog():
177                 return
178             head = st[name].queue[0]
179             if head.start_ms is None:
180                 head.start_ms = t
181             r_bps = user_rate_bps(env, name, t, per_user_rbs)
182             served_mbit = (r_bps * 0.001) / 1e6
183             head.remain_mbit -= served_mbit
184             if head.remain_mbit <= 1e-12:
185                 head.remain_mbit = 0.0
186                 head.finish_ms = t + 1
187
188         for nm in activeU:
189             serve_one(nm, V_U)
190         for nm in activeE:
191             serve_one(nm, V_E)
192         for nm in activeM:
193             serve_one(nm, V_M)
194
195         # 4) QoS统计
196         def collect_finished(order: List[str], slice_type: str
    ):
197             nonlocal sum_U, sum_E, m_success_users
198             for nm in order:
199                 while st[nm].queue and st[nm].queue[0].
    finish_ms == t + 1:
200                     ch = st[nm].queue.popleft()
201                     L_ms = (ch.finish_ms - ch.arrival_ms)
```

```
                    if slice_type == 'U':
                        sum_U += ALPHA ** L_ms if L_ms <= SLA_L_U_MS else -M_U
                    elif slice_type == 'E':
                        r_mbps = (ch.size_mbit * 1000.0) / L_ms
                        if L_ms <= SLA_L_E_MS and r_mbps >= SLA_R_E_MBPS:
                            sum_E += 1.0
                        elif L_ms <= SLA_L_E_MS:
                            sum_E += max(0.0, r_mbps / SLA_R_E_MBPS)
                        else:
                            sum_E += -M_E
                    else:  # mMTC
                        if t0 <= ch.arrival_ms < t1 and L_ms <= SLA_L_M_MS:
                            m_success_users.add(nm)

        collect_finished(U_order, 'U')
        collect_finished(E_order, 'E')
        collect_finished(M_order, 'M')

    # 5) mMTC最终计分
    ratio = len(m_success_users) / len(m_had_arrival) if m_had_arrival else 0.0
    for u in m_had_arrival:
        sum_m_score += ratio if u in m_success_users else -M_M

    obj = sum_U + sum_E + sum_m_score
    return st, type('SimResult', (), {'sum_U': sum_U, 'sum_E': sum_E, 'sum_m_score': sum_m_score, 'obj': obj})()

def enumerate_splits() -> List[Tuple[int, int, int]]:
    """枚举所有合法RB分配"""
```

```
    splits = []
    for nU in range(0, 51, V_U):
        for nE in range(0, 51 - nU, V_E):
            nM = 50 - nU - nE
            if nM >= 0 and nM % V_M == 0:
                splits.append((nU, nE, nM))
    return splits

def main():
    env, U_names, E_names, M_names = build_env()
    users = {}
    for nm in U_names:
        users[nm] = UserState(name=nm, category='U')
    for nm in E_names:
        users[nm] = UserState(name=nm, category='E')
    for nm in M_names:
        users[nm] = UserState(name=nm, category='M')

    splits = enumerate_splits()
    total_obj = 0.0

    # 逐窗口MPC决策
    for w in range(0, TOTAL_MS, WINDOW_MS):
        best_split, best_state_after, best_res = (0, 0, 50), None, None
        best_score = -1e18
        for (nU, nE, nM) in splits:
            new_state, res = simulate_window(env, users, U_names, E_names, M_names, w, nU, nE, nM)
            if res.obj > best_score:
                best_split, best_state_after, best_res = (nU, nE, nM), new_state, res
                best_score = res.obj
        users = best_state_after
        total_obj += best_res.obj
```

```
262
263         # 简化输出：仅在结尾汇总
264
265     print(f"累计目标函数: {total_obj:.4f}")
266
267 if __name__ == "__main__":
268     main()
```

q3.py

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """问题三：多基站GA-MPC求解器（同频干扰+功率控制）"""
4
5  import csv
6  import math
7  import os
8  import random
9  from dataclasses import dataclass
10 from typing import Dict, List, Tuple
11 import numpy as np
12 from collections import deque
13 from copy import deepcopy
14
15 # 基础路径
16 SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
17 ROOT_DIR = os.path.abspath(os.path.join(SCRIPT_DIR, "..", ".."
   ))
18 ATTACH3_DIR = os.path.join(ROOT_DIR, "题目", "附件", "附件3")
19
20 B_HZ = 360_000.0
21 NF_DB = 7.0
22 RB_PER_SLICE = {"U": 10, "E": 5, "M": 2}
23 ALPHA, M_U, M_E, M_M = 0.95, 5.0, 3.0, 1.0
24 SLA_L = {"U": 5.0, "E": 100.0, "M": 500.0}
25 SLA_R_E_MBPS = 50.0
```

```python
POWER_MIN, POWER_MAX = 10.0, 30.0
WINDOW_MS, TOTAL_MS = 100, 1000

# GA参数（适度精简）
POP_SIZE, MAX_GEN = 30, 200
TOURN_K, CROSS_RATE, MUTATE_RATE, ELITE_NUM = 3, 0.8, 0.3, 5
BS_LIST = ["BS1", "BS2", "BS3"]
SLICE_LIST = ["U", "E", "M"]

@dataclass
class Env:
    time_list: List[float]
    arrivals: Dict[str, List[float]]
    phi: Dict[str, Dict[str, List[float]]]
    h_abs: Dict[str, Dict[str, List[float]]]

    def phi_db(self, bs: str, user: str, t: int) -> float:
        arr = self.phi.get(bs, {}).get(user)
        if arr is None or len(arr) == 0:
            return 120.0
        idx = min(max(t, 0), len(arr) - 1)
        return float(arr[idx])

    def h_pow(self, bs: str, user: str, t: int) -> float:
        arr = self.h_abs.get(bs, {}).get(user)
        if arr is None or len(arr) == 0:
            return 1.0
        idx = min(max(t, 0), len(arr) - 1)
        val = float(arr[idx])
        return val * val if val >= 0 else 0.0

@dataclass
class Chunk:
    arrival_ms: int
    size_mbit: float
```

```
    remain_mbit: float
    start_ms: int | None = None
    finish_ms: int | None = None

@dataclass
class UserState:
    name: str
    category: str
    queue: deque[Chunk]

    def has_backlog(self) -> bool:
        return len(self.queue) > 0 and self.queue[0].
   remain_mbit > 1e-12

def load_time_series_csv(path: str) -> Tuple[List[float], Dict
   [str, List[float]]]:
    """加载时间序列CSV数据"""
    time_list = []
    series = {}
    with open(path, "r", encoding="utf-8-sig") as f:
        reader = csv.DictReader(f)
        for row in reader:
            t_val_str = row.get("Time", row.get("time", row.
   get("TIME")))
            if t_val_str is None:
                continue
            try:
                t_val = float(t_val_str)
            except ValueError:
                continue
            time_list.append(t_val)
            for key, val in row.items():
                k = key.strip()
                if k.lower() == "time":
                    continue
```

```
                if val is None or val == "":
                    series.setdefault(k, []).append(0.0)
                    continue
                try:
                    v = float(val)
                except ValueError:
                    v = 0.0
                series.setdefault(k, []).append(v)
    # 填齐列长度
    max_len = len(time_list)
    for arr in series.values():
        if len(arr) < max_len:
            arr.extend([0.0] * (max_len - len(arr)))
    return time_list, series

def build_env() -> Tuple[Env, List[str]]:
    """构建环境和用户列表"""
    time_list, arrivals = load_time_series_csv(os.path.join(ATTACH3_DIR, "taskflow_用户任务流.csv"))
    phi = {}
    h_abs = {}

    CSV_PL = {
        "BS1": os.path.join(ATTACH3_DIR, "BS1_大规模衰减.csv"),
        "BS2": os.path.join(ATTACH3_DIR, "BS2_大规模衰减.csv"),
        "BS3": os.path.join(ATTACH3_DIR, "BS3_大规模衰减.csv"),
    }
    CSV_RAY = {
        "BS1": os.path.join(ATTACH3_DIR, "BS1_小规模瑞丽衰减.csv"),
        "BS2": os.path.join(ATTACH3_DIR, "BS2_小规模瑞丽衰减.csv"),
```

```
        "BS3": os.path.join(ATTACH3_DIR, "BS3_小规模瑞丽衰减.csv"),
    }

    for bs in BS_LIST:
        _, phi_bs = load_time_series_csv(CSV_PL[bs])
        _, ray_bs = load_time_series_csv(CSV_RAY[bs])
        phi[bs] = phi_bs
        h_abs[bs] = ray_bs

    all_users = sorted(arrivals.keys(), key=lambda x: (x[0], int(x[1:]) if x[1:].isdigit() else 0))
    return Env(time_list=time_list, arrivals=arrivals, phi=phi, h_abs=h_abs), all_users

def user_category(name: str) -> str:
    """确定用户类别"""
    if name.startswith("U"):
        return "U"
    if name.startswith("e"):
        return "E"
    return "M"

def user_rate_bps(env: Env, bs: str, name: str, cat: str, power_alloc: Dict[str, Dict[str, float]],
                  active_bs_same_slice: List[str], t_ms: int) -> float:
    """计算考虑干扰的下行速率"""
    p_tx_mw = 10 ** (power_alloc[bs][cat] / 10.0)
    phi_db = env.phi_db(bs, name, t_ms)
    h_pow = env.h_pow(bs, name, t_ms)
    recv_mw = p_tx_mw * 10 ** (-phi_db / 10.0) * h_pow

    # 计算同切片干扰
    interf_mw = 0.0
    for b2 in active_bs_same_slice:
```

```
152         if b2 == bs:
153             continue
154         p_int_mw = 10 ** (power_alloc[b2][cat] / 10.0)
155         phi_db_i = env.phi_db(b2, name, t_ms)
156         h_pow_i = env.h_pow(b2, name, t_ms)
157         interf_mw += p_int_mw * 10 ** (-phi_db_i / 10.0) *
    h_pow_i
158
159     n0_mw = 10 ** ((-174.0 + 10.0 * math.log10(RB_PER_SLICE[
    cat] * B_HZ) + NF_DB) / 10.0)
160     sinr = recv_mw / (interf_mw + n0_mw + 1e-30)
161     return RB_PER_SLICE[cat] * B_HZ * math.log2(1.0 + sinr)
162
163 def simulate_window_multibs(env: Env, init_states: Dict[str,
    UserState], mapping: Dict[str, str], rb_alloc: Dict[str,
    Dict[str, int]], power_alloc: Dict[str, Dict[str, float]],
    t0: int, copy_state: bool = True):
164     """多基站窗口仿真"""
165     states = {}
166     if copy_state:
167         for name, st in init_states.items():
168             states[name] = UserState(name=st.name, category=st
    .category, queue=deque(deepcopy(list(st.queue))))
169     else:
170         states = init_states
171
172     # 计算并发容量
173     cap = {bs: {s: rb_alloc[bs][s] // RB_PER_SLICE[s] for s in
     SLICE_LIST} for bs in BS_LIST}
174     active = {bs: {s: [] for s in SLICE_LIST} for bs in
    BS_LIST}
175
176     # 用户顺序（编号优先）
177     def sort_key(u: str):
178         return (u[0], int(u[1:]) if u[1:].isdigit() else 0)
```

```
179
180     order = {bs: {s: [] for s in SLICE_LIST} for bs in BS_LIST
    }
181     for nm, bs in mapping.items():
182         cat = user_category(nm)
183         order[bs][cat].append(nm)
184     for bs in BS_LIST:
185         for s in SLICE_LIST:
186             order[bs][s].sort(key=sort_key)
187
188     sum_U = sum_E = sum_M = 0.0
189     had_m_users = set()
190     success_m_users = set()
191
192     t1 = min(t0 + WINDOW_MS, TOTAL_MS)
193     for t in range(t0, t1):
194         # 到达处理
195         for nm in states.keys():
196             arr_series = env.arrivals.get(nm, [])
197             if t < len(arr_series):
198                 vol = arr_series[t]
199                 if vol > 0.0:
200                     states[nm].queue.append(Chunk(arrival_ms=t
    , size_mbit=vol, remain_mbit=vol))
201                     if states[nm].category == 'M':
202                         had_m_users.add(nm)
203
204         # 填充并发槽位
205         for bs in BS_LIST:
206             for s in SLICE_LIST:
207                 active[bs][s] = [u for u in active[bs][s] if
    states[u].has_backlog()]
208                 while len(active[bs][s]) < cap[bs][s]:
209                     for cand in order[bs][s]:
210                         if cand in active[bs][s]:
```

```
                            continue
                        if states[cand].has_backlog():
                            active[bs][s].append(cand)
                            break
                    else:
                        break

        # 服务处理
        for bs in BS_LIST:
            for s in SLICE_LIST:
                if len(active[bs][s]) == 0:
                    continue
                active_bs_same_slice = [b for b in BS_LIST if
len(active[b][s]) > 0]
                for u in list(active[bs][s]):
                    rate_bps = user_rate_bps(env, bs, u, s,
power_alloc, active_bs_same_slice, t)
                    served_mbit = (rate_bps * 0.001) / 1e6
                    head = states[u].queue[0]
                    if head.start_ms is None:
                        head.start_ms = t
                    head.remain_mbit -= served_mbit
                    if head.remain_mbit <= 1e-12:
                        head.remain_mbit = 0.0
                        head.finish_ms = t + 1
                        states[u].queue.popleft()
                        # QoS统计
                        L_ms = (head.finish_ms - head.
arrival_ms)
                        if s == 'U':
                            sum_U += ALPHA ** L_ms if L_ms <=
SLA_L['U'] else -M_U
                        elif s == 'E':
                            r_mbps = (head.size_mbit * 1000.0)
 / max(L_ms, 1e-12)
```

```
                            if L_ms <= SLA_L['E'] and r_mbps >= SLA_R_E_MBPS:
                                sum_E += 1.0
                            elif L_ms <= SLA_L['E']:
                                sum_E += max(0.0, r_mbps / SLA_R_E_MBPS)
                            else:
                                sum_E += -M_E
                        else:
                            if L_ms <= SLA_L['M']:
                                success_m_users.add(u)

    # mMTC评分
    if len(had_m_users) > 0:
        ratio = len(success_m_users) / len(had_m_users)
        for u in had_m_users:
            sum_M += ratio if u in success_m_users else -M_M

    obj = sum_U + sum_E + sum_M
    result = type('SimResult', (), {'sum_U': sum_U, 'sum_E': sum_E, 'sum_M': sum_M, 'obj': obj})()
    return states, result

# 遗传算法实现
def random_individual(num_users: int):
    """生成随机个体: 用户-基站映射 + RB分配 + 功率设置"""
    user_bs = np.random.randint(0, 3, size=num_users, dtype=np.int8)
    rb_counts = np.zeros(6, dtype=np.int16)  # 每个BS的U_RB, E_RB
    idx = 0
    for _bs in BS_LIST:
        total = 50
        u_units_max = total // RB_PER_SLICE['U']
        u_units = np.random.randint(0, u_units_max + 1)
```

```
271         u_rb = int(u_units * RB_PER_SLICE['U'])
272         e_units_max = (total - u_rb) // RB_PER_SLICE['E']
273         e_units = np.random.randint(0, e_units_max + 1)
274         e_rb = int(e_units * RB_PER_SLICE['E'])
275         rb_counts[idx] = u_rb
276         rb_counts[idx + 1] = e_rb
277         idx += 2
278     power = np.random.uniform(18.0, 26.0, size=9).astype(np.
    float32)
279     return [user_bs, rb_counts, power]
280 
281 def decode_rb(rb_counts: np.ndarray) -> Dict[str, Dict[str,
    int]]:
282     """解码RB分配"""
283     out = {}
284     idx = 0
285     for bs in BS_LIST:
286         total = 50
287         u_req = int(rb_counts[idx])
288         e_req = int(rb_counts[idx + 1])
289         idx += 2
290         u_rb = max(0, min(total, (u_req // RB_PER_SLICE['U'])
    * RB_PER_SLICE['U']))
291         e_rb = max(0, min(total - u_rb, (e_req // RB_PER_SLICE
    ['E']) * RB_PER_SLICE['E']))
292         if u_rb + e_rb > total:
293             e_rb = max(0, (total - u_rb) // RB_PER_SLICE['E']
    * RB_PER_SLICE['E'])
294         m_rb = total - (u_rb + e_rb)
295         if m_rb % RB_PER_SLICE['M'] != 0:
296             if e_rb >= RB_PER_SLICE['E']:
297                 e_rb -= RB_PER_SLICE['E']
298             elif e_rb + RB_PER_SLICE['E'] <= total - u_rb:
299                 e_rb += RB_PER_SLICE['E']
300             elif u_rb >= RB_PER_SLICE['U']:
```

```
                u_rb -= RB_PER_SLICE['U']
            m_rb = total - (u_rb + e_rb)
            if m_rb % RB_PER_SLICE['M'] != 0 and u_rb >=
    RB_PER_SLICE['U']:
                u_rb -= RB_PER_SLICE['U']
                m_rb = total - (u_rb + e_rb)
        out[bs] = {"U": u_rb, "E": e_rb, "M": m_rb}
    return out

def decode_power(power_arr: np.ndarray) -> Dict[str, Dict[str,
    float]]:
    """解码功率分配"""
    out = {}
    idx = 0
    for bs in BS_LIST:
        sub = {}
        for s in SLICE_LIST:
            p = float(power_arr[idx])
            p = max(POWER_MIN, min(POWER_MAX, p))
            sub[s] = p
            idx += 1
        out[bs] = sub
    return out

def evaluate_individual(indiv, env: Env, users: List[str],
    user_states: Dict[str, UserState], t0: int) -> float:
    """评价个体适应度"""
    user_bs, rb_ratio, power_arr = indiv
    rb_alloc = decode_rb(rb_ratio)
    power_alloc = decode_power(power_arr)
    mapping = {users[i]: BS_LIST[int(user_bs[i])] for i in
    range(len(users))}
    _, res = simulate_window_multibs(env, user_states, mapping
    , rb_alloc, power_alloc, t0, copy_state=True)
    return res.obj
```

```
331
332 def tournament_select(pop_scores: List[float]) -> int:
333     """锦标赛选择"""
334     cand = random.sample(range(len(pop_scores)), TOURN_K)
335     cand.sort(key=lambda idx: pop_scores[idx], reverse=True)
336     return cand[0]
337
338 def crossover(parent1, parent2):
339     """交叉操作"""
340     if random.random() > CROSS_RATE:
341         return parent1, parent2
342     u1, rb1, p1 = parent1
343     u2, rb2, p2 = parent2
344     num_users = len(u1)
345     pt = random.randint(1, num_users - 1)
346     child_u1 = np.concatenate([u1[:pt], u2[pt:]]).astype(np.
    int8)
347     child_u2 = np.concatenate([u2[:pt], u1[pt:]]).astype(np.
    int8)
348     child_rb1 = ((rb1.astype(np.float32) + rb2) / 2.0).astype(
    np.int16)
349     child_rb2 = child_rb1.copy()
350     child_p1 = (p1 + p2) / 2.0
351     child_p2 = child_p1.copy()
352     return [child_u1, child_rb1, child_p1], [child_u2,
    child_rb2, child_p2]
353
354 def mutate(indiv):
355     """变异操作"""
356     user_bs, rb_counts, power_arr = indiv
357     num_users = len(user_bs)
358     # 用户基站变异
359     for i in range(num_users):
360         if random.random() < MUTATE_RATE:
361             user_bs[i] = np.random.randint(0, 3)
```

```
    # RB变异
    for b in range(len(BS_LIST)):
        iu = 2 * b
        ie = iu + 1
        total = 50
        if random.random() < MUTATE_RATE:
            u_units_max = total // RB_PER_SLICE['U']
            u_units = np.random.randint(0, u_units_max + 1)
            rb_counts[iu] = int(u_units * RB_PER_SLICE['U'])
        if random.random() < MUTATE_RATE:
            u_rb = int(rb_counts[iu])
            e_units_max = (total - u_rb) // RB_PER_SLICE['E']
            e_units = np.random.randint(0, e_units_max + 1)
            rb_counts[ie] = int(e_units * RB_PER_SLICE['E'])
    # 功率高斯噪声
    for i in range(len(power_arr)):
        if random.random() < MUTATE_RATE:
            power_arr[i] += np.random.normal(0.0, 1.0)
            power_arr[i] = max(POWER_MIN, min(POWER_MAX,
  power_arr[i]))

def main():
    """附录精简：第三问省略与第二问相同的MPC仿真，仅保留关键函
  数与接口。"""
    print("[附录] 问题三：已省略滚动窗口MPC仿真及完整GA循环，
  仅保留核心组件与接口。")

if __name__ == "__main__":
    main()
```

q4.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""问题四：异构网络GA-MPC求解器（MBS+SBS，接入受限）"""

```

```python
import csv
import math
import os
import random
from dataclasses import dataclass
from typing import Dict, List, Tuple
import numpy as np
from collections import deque
from copy import deepcopy

# 路径和常量配置
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
ROOT_DIR = os.path.abspath(os.path.join(SCRIPT_DIR, "..", ".."))
ATTACH4_DIR = os.path.join(ROOT_DIR, "题目", "附件", "附件4")

B_HZ = 360_000.0
NF_DB = 7.0
RB_PER_SLICE = {"U": 10, "E": 5, "M": 2}
SLICE_LIST = ["U", "E", "M"]
ALPHA, M_U, M_E, M_M = 0.95, 5.0, 3.0, 1.0
SLA_L_MS = {"U": 5.0, "E": 100.0, "M": 500.0}
SLA_R_E_MBPS = 50.0
WINDOW_MS, TOTAL_MS = 100, 1000

# 基站配置
BS_LIST = ["MBS_1", "SBS_1", "SBS_2", "SBS_3"]
SBS_ONLY = [bs for bs in BS_LIST if bs.startswith("SBS_")]
RB_TOTAL = {"MBS_1": 100, "SBS_1": 50, "SBS_2": 50, "SBS_3": 50}
P_MIN_DBM = {"MBS_1": 10.0, "SBS_1": 10.0, "SBS_2": 10.0, "SBS_3": 10.0}
P_MAX_DBM = {"MBS_1": 40.0, "SBS_1": 30.0, "SBS_2": 30.0, "SBS_3": 30.0}

```

```python
# 基站坐标
BS_COORD = {
    "MBS_1": (0.0, 0.0),
    "SBS_1": (0.0, 500.0),
    "SBS_2": (-433.0127, -250.0),
    "SBS_3": (433.0127, -250.0),
}

# GA参数（适度精简）
POP_SIZE, MAX_GEN = 30, 200
TOURN_K, CROSS_RATE, MUTATE_RATE, ELITE_NUM = 3, 0.8, 0.3, 5

@dataclass
class Env:
    time_list: List[float]
    arrivals: Dict[str, List[float]]
    phi: Dict[str, Dict[str, List[float]]]
    h_abs: Dict[str, Dict[str, List[float]]]
    pos_x: Dict[str, List[float]]
    pos_y: Dict[str, List[float]]

    def phi_db(self, bs: str, user: str, t: int) -> float:
        arr = self.phi.get(bs, {}).get(user)
        if arr is None or len(arr) == 0:
            return 120.0
        idx = min(max(t, 0), len(arr) - 1)
        return float(arr[idx])

    def h_pow(self, bs: str, user: str, t: int) -> float:
        arr = self.h_abs.get(bs, {}).get(user)
        if arr is None or len(arr) == 0:
            return 1.0
        idx = min(max(t, 0), len(arr) - 1)
        val = float(arr[idx])
        return val * val if val >= 0 else 0.0
```

```
71
72      def user_xy(self, user: str, t: int) -> Tuple[float, float
    ]:
73          xs = self.pos_x.get(f"{user}_X") or self.pos_x.get(f"{
    user}X") or self.pos_x.get(user)
74          ys = self.pos_y.get(f"{user}_Y") or self.pos_y.get(f"{
    user}Y") or self.pos_y.get(user)
75          if not xs or not ys:
76              return 0.0, 0.0
77          idx = min(max(t, 0), min(len(xs), len(ys)) - 1)
78          return float(xs[idx]), float(ys[idx])
79
80  @dataclass
81  class Chunk:
82      arrival_ms: int
83      size_mbit: float
84      remain_mbit: float
85      start_ms: int | None = None
86      finish_ms: int | None = None
87
88  @dataclass
89  class UserState:
90      name: str
91      category: str
92      queue: deque[Chunk]
93
94      def has_backlog(self) -> bool:
95          return len(self.queue) > 0 and self.queue[0].
    remain_mbit > 1e-12
96
97  def load_time_series_csv(path: str) -> Tuple[List[float], Dict
    [str, List[float]]]:
98      """读取时间序列CSV"""
99      time_list = []
100     series = {}
```

```
    with open(path, "r", encoding="utf-8-sig") as f:
        reader = csv.DictReader(f)
        for row in reader:
            t_val_str = row.get("Time", row.get("time", row.get("TIME")))
            if t_val_str is None:
                continue
            try:
                t_val = float(t_val_str)
            except ValueError:
                continue
            time_list.append(t_val)
            for key, val in row.items():
                k = key.strip()
                if k.lower() == "time":
                    continue
                if val is None or val == "":
                    series.setdefault(k, []).append(0.0)
                    continue
                try:
                    v = float(val)
                except ValueError:
                    v = 0.0
                series.setdefault(k, []).append(v)
    max_len = len(time_list)
    for arr in series.values():
        if len(arr) < max_len:
            arr.extend([0.0] * (max_len - len(arr)))
    return time_list, series

def build_env() -> Tuple[Env, List[str]]:
    """构建环境"""
    time_list, arrivals = load_time_series_csv(os.path.join(ATTACH4_DIR, "taskflow_用户任务流.csv"))

```

```
    # 信道数据
    phi = {}
    h_abs = {}
    CSV_PL = {
        "MBS_1": os.path.join(ATTACH4_DIR, "MBS_1_大规模衰减.csv"),
        "SBS_1": os.path.join(ATTACH4_DIR, "SBS_1_大规模衰减.csv"),
        "SBS_2": os.path.join(ATTACH4_DIR, "SBS_2_大规模衰减.csv"),
        "SBS_3": os.path.join(ATTACH4_DIR, "SBS_3_大规模衰减.csv"),
    }
    CSV_RAY = {
        "MBS_1": os.path.join(ATTACH4_DIR, "MBS_1_小规模瑞丽衰减.csv"),
        "SBS_1": os.path.join(ATTACH4_DIR, "SBS_1_小规模瑞丽衰减.csv"),
        "SBS_2": os.path.join(ATTACH4_DIR, "SBS_2_小规模瑞丽衰减.csv"),
        "SBS_3": os.path.join(ATTACH4_DIR, "SBS_3_小规模瑞丽衰减.csv"),
    }

    for bs in BS_LIST:
        _, phi_bs = load_time_series_csv(CSV_PL[bs])
        _, ray_bs = load_time_series_csv(CSV_RAY[bs])
        phi[bs] = phi_bs
        h_abs[bs] = ray_bs

    # 位置数据
    _, pos_series = load_time_series_csv(os.path.join(ATTACH4_DIR, "taskflow_用户位置.csv"))
    pos_x = {}
    pos_y = {}
```

```
    for key, arr in pos_series.items():
        if key.endswith("_X"):
            pos_x[key] = arr
        elif key.endswith("_Y"):
            pos_y[key] = arr

    all_users = sorted(arrivals.keys(), key=lambda x: (x[0], int(x[1:]) if x[1:].isdigit() else 0))
    return Env(time_list=time_list, arrivals=arrivals, phi=phi, h_abs=h_abs, pos_x=pos_x, pos_y=pos_y), all_users

def user_category(name: str) -> str:
    """确定用户类别"""
    if name.startswith("U"):
        return "U"
    if name.startswith("e"):
        return "E"
    return "M"

def user_rate_bps(env: Env, bs: str, name: str, cat: str, power_alloc: Dict[str, Dict[str, float]],
                  active_sbs_same_slice: List[str], t_ms: int) -> float:
    """计算下行速率，MBS无干扰，SBS受其他SBS干扰"""
    p_tx_mw = 10 ** (power_alloc[bs][cat] / 10.0)
    phi_db = env.phi_db(bs, name, t_ms)
    h_pow = env.h_pow(bs, name, t_ms)
    recv_mw = p_tx_mw * 10 ** (-phi_db / 10.0) * h_pow

    interf_mw = 0.0
    if bs in SBS_ONLY:
        for b2 in active_sbs_same_slice:
            if b2 == bs:
                continue
            p_int_mw = 10 ** (power_alloc[b2][cat] / 10.0)
            phi_db_i = env.phi_db(b2, name, t_ms)
```

```
            h_pow_i = env.h_pow(b2, name, t_ms)
            interf_mw += p_int_mw * 10 ** (-phi_db_i / 10.0) *
    h_pow_i

    n0_mw = 10 ** ((-174.0 + 10.0 * math.log10(RB_PER_SLICE[
    cat] * B_HZ) + NF_DB) / 10.0)
    sinr = recv_mw / (interf_mw + n0_mw + 1e-30)
    return RB_PER_SLICE[cat] * B_HZ * math.log2(1.0 + sinr)

def simulate_window(env: Env, init_states: Dict[str, UserState
    ], mapping: Dict[str, str], rb_alloc: Dict[str, Dict[str,
    int]], power_alloc: Dict[str, Dict[str, float]], t0: int,
    copy_state: bool = True):
    """异构网络窗口仿真"""
    states = {}
    if copy_state:
        for name, st in init_states.items():
            states[name] = UserState(name=st.name, category=st
    .category, queue=deque(deepcopy(list(st.queue))))
    else:
        states = init_states

    cap = {bs: {s: rb_alloc[bs][s] // RB_PER_SLICE[s] for s in
    SLICE_LIST} for bs in BS_LIST}
    active = {bs: {s: [] for s in SLICE_LIST} for bs in
    BS_LIST}

    def sort_key(u: str):
        return (u[0], int(u[1:]) if u[1:].isdigit() else 0)

    order = {bs: {s: [] for s in SLICE_LIST} for bs in BS_LIST
    }
    for nm, bs in mapping.items():
        cat = user_category(nm)
        order[bs][cat].append(nm)
```

```
    for bs in BS_LIST:
        for s in SLICE_LIST:
            order[bs][s].sort(key=sort_key)

    sum_U = sum_E = sum_M = 0.0
    had_m_users = set()
    success_m_users = set()

    t1 = min(t0 + WINDOW_MS, TOTAL_MS)
    for t in range(t0, t1):
        # 到达
        for nm in states.keys():
            arr_series = env.arrivals.get(nm, [])
            if t < len(arr_series):
                vol = arr_series[t]
                if vol > 0.0:
                    states[nm].queue.append(Chunk(arrival_ms=t
, size_mbit=vol, remain_mbit=vol))
                    if states[nm].category == 'M':
                        had_m_users.add(nm)

        # 填充并发槽位
        for bs in BS_LIST:
            for s in SLICE_LIST:
                active[bs][s] = [u for u in active[bs][s] if
states[u].has_backlog()]
                while len(active[bs][s]) < cap[bs][s]:
                    for cand in order[bs][s]:
                        if cand in active[bs][s]:
                            continue
                        if states[cand].has_backlog():
                            active[bs][s].append(cand)
                            break
                    else:
                        break
```

```
250
251         # 服务
252         for bs in BS_LIST:
253             for s in SLICE_LIST:
254                 if len(active[bs][s]) == 0:
255                     continue
256                 active_sbs_same_slice = [b for b in SBS_ONLY
    if len(active[b][s]) > 0]
257                 for u in list(active[bs][s]):
258                     rate_bps = user_rate_bps(env, bs, u, s,
    power_alloc, active_sbs_same_slice, t)
259                     served_mbit = (rate_bps * 0.001) / 1e6
260                     head = states[u].queue[0]
261                     if head.start_ms is None:
262                         head.start_ms = t
263                     head.remain_mbit -= served_mbit
264                     if head.remain_mbit <= 1e-12:
265                         head.remain_mbit = 0.0
266                         head.finish_ms = t + 1
267                         states[u].queue.popleft()
268                         # QoS统计
269                         L_ms = (head.finish_ms - head.
    arrival_ms)
270                         if s == 'U':
271                             sum_U += ALPHA ** L_ms if L_ms <=
    SLA_L_MS['U'] else -M_U
272                         elif s == 'E':
273                             r_mbps = (head.size_mbit * 1000.0)
     / max(L_ms, 1e-12)
274                             if L_ms <= SLA_L_MS['E'] and
    r_mbps >= SLA_R_E_MBPS:
275                                 sum_E += 1.0
276                             elif L_ms <= SLA_L_MS['E']:
277                                 sum_E += max(0.0, r_mbps /
    SLA_R_E_MBPS)
```

```
                                else:
                                    sum_E += -M_E
                            else:
                                if L_ms <= SLA_L_MS['M']:
                                    success_m_users.add(u)

    # mMTC评分
    if len(had_m_users) > 0:
        ratio = len(success_m_users) / len(had_m_users)
        for u in had_m_users:
            sum_M += ratio if u in success_m_users else -M_M

    obj = sum_U + sum_E + sum_M
    result = type('SimResult', (), {'sum_U': sum_U, 'sum_E': sum_E, 'sum_M': sum_M, 'obj': obj})()
    return states, result

def nearest_sbs_per_user(env: Env, users: List[str], t_ms: int) -> Dict[str, str]:
    """计算每个用户的最近SBS"""
    out = {}
    for u in users:
        x, y = env.user_xy(u, t_ms)
        best_bs = None
        best_d2 = 1e99
        for sbs in SBS_ONLY:
            bx, by = BS_COORD[sbs]
            d2 = (x - bx) ** 2 + (y - by) ** 2
            if d2 < best_d2:
                best_d2 = d2
                best_bs = sbs
        out[u] = best_bs or "SBS_1"
    return out

# 遗传算法实现
```

```python
def random_individual(num_users: int):
    """生成随机个体：用户选择(0=MBS,1=nearest SBS) + RB分配 + 功率"""
    user_choice = np.random.randint(0, 2, size=num_users, dtype=np.int8)
    rb_counts = np.zeros(2 * len(BS_LIST), dtype=np.int16)
    idx = 0
    for bs in BS_LIST:
        total = RB_TOTAL[bs]
        u_units_max = total // RB_PER_SLICE['U']
        u_units = np.random.randint(0, u_units_max + 1)
        u_rb = int(u_units * RB_PER_SLICE['U'])
        e_units_max = (total - u_rb) // RB_PER_SLICE['E']
        e_units = np.random.randint(0, e_units_max + 1)
        e_rb = int(e_units * RB_PER_SLICE['E'])
        rb_counts[idx] = u_rb
        rb_counts[idx + 1] = e_rb
        idx += 2
    power = np.random.uniform(20.0, 30.0, size=3 * len(BS_LIST)).astype(np.float32)
    return [user_choice, rb_counts, power]

def decode_rb(rb_counts: np.ndarray) -> Dict[str, Dict[str, int]]:
    """解码RB分配"""
    out = {}
    idx = 0
    for bs in BS_LIST:
        total = RB_TOTAL[bs]
        u_req = int(rb_counts[idx])
        e_req = int(rb_counts[idx + 1])
        idx += 2
        u_rb = max(0, min(total, (u_req // RB_PER_SLICE['U']) * RB_PER_SLICE['U']))
        e_rb = max(0, min(total - u_rb, (e_req // RB_PER_SLICE
```

```
    ['E']) * RB_PER_SLICE['E']))
        if u_rb + e_rb > total:
            e_rb = max(0, (total - u_rb) // RB_PER_SLICE['E']
    * RB_PER_SLICE['E'])
        m_rb = total - (u_rb + e_rb)
        if m_rb % RB_PER_SLICE['M'] != 0:
            if e_rb >= RB_PER_SLICE['E']:
                e_rb -= RB_PER_SLICE['E']
            elif e_rb + RB_PER_SLICE['E'] <= total - u_rb:
                e_rb += RB_PER_SLICE['E']
            elif u_rb >= RB_PER_SLICE['U']:
                u_rb -= RB_PER_SLICE['U']
            m_rb = total - (u_rb + e_rb)
            if m_rb % RB_PER_SLICE['M'] != 0 and u_rb >=
    RB_PER_SLICE['U']:
                u_rb -= RB_PER_SLICE['U']
                m_rb = total - (u_rb + e_rb)
        out[bs] = {"U": u_rb, "E": e_rb, "M": m_rb}
    return out

def decode_power(power_arr: np.ndarray) -> Dict[str, Dict[str,
    float]]:
    """解码功率分配"""
    out = {}
    idx = 0
    for bs in BS_LIST:
        sub = {}
        for s in SLICE_LIST:
            p = float(power_arr[idx])
            p = max(P_MIN_DBM[bs], min(P_MAX_DBM[bs], p))
            sub[s] = p
            idx += 1
        out[bs] = sub
    return out

```

```python
def evaluate_individual(indiv, env: Env, users: List[str], user_states: Dict[str, UserState], t0: int, nearest_sbs_map: Dict[str, str]) -> float:
    """评价个体适应度"""
    user_choice, rb_counts, power_arr = indiv
    rb_alloc = decode_rb(rb_counts)
    power_alloc = decode_power(power_arr)
    mapping = {users[i]: ("MBS_1" if int(user_choice[i]) == 0 else nearest_sbs_map[users[i]]) for i in range(len(users))}
    _, res = simulate_window(env, user_states, mapping, rb_alloc, power_alloc, t0, copy_state=True)
    return res.obj

def tournament_select(pop_scores: List[float]) -> int:
    """锦标赛选择"""
    cand = random.sample(range(len(pop_scores)), TOURN_K)
    cand.sort(key=lambda idx: pop_scores[idx], reverse=True)
    return cand[0]

def crossover(parent1, parent2):
    """交叉操作"""
    if random.random() > CROSS_RATE:
        return parent1, parent2
    u1, rb1, p1 = parent1
    u2, rb2, p2 = parent2
    num_users = len(u1)
    if num_users >= 2:
        pt = random.randint(1, num_users - 1)
        child_u1 = np.concatenate([u1[:pt], u2[pt:]]).astype(np.int8)
        child_u2 = np.concatenate([u2[:pt], u1[pt:]]).astype(np.int8)
    else:
        child_u1 = u1.copy()
        child_u2 = u2.copy()
```

```
    child_rb1 = ((rb1.astype(np.float32) + rb2) / 2.0).astype(np.int16)
    child_rb2 = child_rb1.copy()
    child_p1 = (p1 + p2) / 2.0
    child_p2 = child_p1.copy()
    return [child_u1, child_rb1, child_p1], [child_u2, child_rb2, child_p2]

def mutate(indiv):
    """变异操作"""
    user_choice, rb_counts, power_arr = indiv
    num_users = len(user_choice)
    # 接入变异
    for i in range(num_users):
        if random.random() < MUTATE_RATE:
            user_choice[i] = 1 - int(user_choice[i])
    # RB变异
    for b in range(len(BS_LIST)):
        iu = 2 * b
        ie = iu + 1
        total = RB_TOTAL[BS_LIST[b]]
        if random.random() < MUTATE_RATE:
            u_units_max = total // RB_PER_SLICE['U']
            u_units = np.random.randint(0, u_units_max + 1)
            rb_counts[iu] = int(u_units * RB_PER_SLICE['U'])
        if random.random() < MUTATE_RATE:
            u_rb = int(rb_counts[iu])
            e_units_max = (total - u_rb) // RB_PER_SLICE['E']
            e_units = np.random.randint(0, e_units_max + 1)
            rb_counts[ie] = int(e_units * RB_PER_SLICE['E'])
    # 功率高斯扰动
    for i in range(len(power_arr)):
        if random.random() < MUTATE_RATE:
            power_arr[i] += np.random.normal(0.0, 1.0)

```

```
def main():
    """附录精简：第四问省略与第二问相同的MPC滚动仿真，仅保留关键函数/接口。"""
    print("[附录] 问题四：已省略MPC滚动仿真与完整GA循环，仅保留核心组件与接口。")

if __name__ == "__main__":
    main()
```

q5.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""问题五：能耗优化GA-MPC求解器（二阶段：功率优先+RB枚举）"""

import csv
import math
import os
import random
from dataclasses import dataclass
from typing import Dict, List, Tuple
import numpy as np
from collections import deque
from copy import deepcopy

# 路径和常量配置
SCRIPT_DIR = os.path.dirname(os.path.abspath(__file__))
ROOT_DIR = os.path.abspath(os.path.join(SCRIPT_DIR, "..", ".."))
ATTACH4_DIR = os.path.join(ROOT_DIR, "题目", "附件", "附件4")

B_HZ = 360_000.0
NF_DB = 7.0
RB_PER_SLICE = {"U": 10, "E": 5, "M": 2}
SLICE_LIST = ["U", "E", "M"]
BS_LIST = ["MBS_1", "SBS_1", "SBS_2", "SBS_3"]
```

```python
SBS_ONLY = [bs for bs in BS_LIST if bs.startswith("SBS_")]
RB_TOTAL = {"MBS_1": 100, "SBS_1": 50, "SBS_2": 50, "SBS_3": 50}
P_MIN_DBM = {"MBS_1": 10.0, "SBS_1": 10.0, "SBS_2": 10.0, "SBS_3": 10.0}
P_MAX_DBM = {"MBS_1": 40.0, "SBS_1": 30.0, "SBS_2": 30.0, "SBS_3": 30.0}

ALPHA, M_U, M_E, M_M = 0.95, 5.0, 3.0, 1.0
SLA_L_MS = {"U": 5.0, "E": 100.0, "M": 500.0}
SLA_R_E_MBPS = 50.0
WINDOW_MS, TOTAL_MS = 100, 1000

# 能耗模型参数
P_STATIC_W = 28.0
DELTA_W_PER_RB = 0.75
ETA = 0.35

# GA参数（阶段一功率优化，适度精简）
POP_SIZE, MAX_GEN = 16, 120
TOURN_K, CROSS_RATE, MUTATE_RATE, ELITE_NUM = 3, 0.8, 0.3, 5

@dataclass
class Env:
    time_list: List[float]
    arrivals: Dict[str, List[float]]
    phi: Dict[str, Dict[str, List[float]]]
    h_abs: Dict[str, Dict[str, List[float]]]
    pos_x: Dict[str, List[float]]
    pos_y: Dict[str, List[float]]

    def phi_db(self, bs: str, user: str, t: int) -> float:
        arr = self.phi.get(bs, {}).get(user)
        if arr is None or len(arr) == 0:
            return 120.0
```

```
        idx = min(max(t, 0), len(arr) - 1)
        return float(arr[idx])

    def h_pow(self, bs: str, user: str, t: int) -> float:
        arr = self.h_abs.get(bs, {}).get(user)
        if arr is None or len(arr) == 0:
            return 1.0
        idx = min(max(t, 0), len(arr) - 1)
        val = float(arr[idx])
        return val * val if val >= 0 else 0.0

    def user_xy(self, user: str, t: int) -> Tuple[float, float]:
        xs = self.pos_x.get(f"{user}_X") or self.pos_x.get(f"{user}X") or self.pos_x.get(user)
        ys = self.pos_y.get(f"{user}_Y") or self.pos_y.get(f"{user}Y") or self.pos_y.get(user)
        if not xs or not ys:
            return 0.0, 0.0
        idx = min(max(t, 0), min(len(xs), len(ys)) - 1)
        return float(xs[idx]), float(ys[idx])

@dataclass
class Chunk:
    arrival_ms: int
    size_mbit: float
    remain_mbit: float
    start_ms: int | None = None
    finish_ms: int | None = None

@dataclass
class UserState:
    name: str
    category: str
    queue: deque[Chunk]
```

```
89
90      def has_backlog(self) -> bool:
91          return len(self.queue) > 0 and self.queue[0].
    remain_mbit > 1e-12
92
93  def load_time_series_csv(path: str) -> Tuple[List[float], Dict
    [str, List[float]]]:
94      """读取时间序列CSV"""
95      time_list = []
96      series = {}
97      with open(path, "r", encoding="utf-8-sig") as f:
98          reader = csv.DictReader(f)
99          for row in reader:
100             t_val_str = row.get("Time", row.get("time", row.
    get("TIME")))
101             if t_val_str is None:
102                 continue
103             try:
104                 t_val = float(t_val_str)
105             except ValueError:
106                 continue
107             time_list.append(t_val)
108             for key, val in row.items():
109                 k = key.strip()
110                 if k.lower() == "time":
111                     continue
112                 if val is None or val == "":
113                     series.setdefault(k, []).append(0.0)
114                     continue
115                 try:
116                     v = float(val)
117                 except ValueError:
118                     v = 0.0
119                 series.setdefault(k, []).append(v)
120     max_len = len(time_list)
```

```
    for arr in series.values():
        if len(arr) < max_len:
            arr.extend([0.0] * (max_len - len(arr)))
    return time_list, series

def build_env() -> Tuple[Env, List[str]]:
    """构建环境"""
    time_list, arrivals = load_time_series_csv(os.path.join(ATTACH4_DIR, "taskflow_用户任务流.csv"))
    phi = {}
    h_abs = {}

    CSV_PL = {
        "MBS_1": os.path.join(ATTACH4_DIR, "MBS_1_大规模衰减.csv"),
        "SBS_1": os.path.join(ATTACH4_DIR, "SBS_1_大规模衰减.csv"),
        "SBS_2": os.path.join(ATTACH4_DIR, "SBS_2_大规模衰减.csv"),
        "SBS_3": os.path.join(ATTACH4_DIR, "SBS_3_小规模瑞丽衰减.csv"),
    }
    CSV_RAY = {
        "MBS_1": os.path.join(ATTACH4_DIR, "MBS_1_小规模瑞丽衰减.csv"),
        "SBS_1": os.path.join(ATTACH4_DIR, "SBS_1_小规模瑞丽衰减.csv"),
        "SBS_2": os.path.join(ATTACH4_DIR, "SBS_2_小规模瑞丽衰减.csv"),
        "SBS_3": os.path.join(ATTACH4_DIR, "SBS_3_小规模瑞丽衰减.csv"),
    }

    for bs in BS_LIST:
        _, phi_bs = load_time_series_csv(CSV_PL[bs])
```

```
        _, ray_bs = load_time_series_csv(CSV_RAY[bs])
        phi[bs] = phi_bs
        h_abs[bs] = ray_bs

    _, pos_series = load_time_series_csv(os.path.join(ATTACH4_DIR, "taskflow_用户位置.csv"))
    pos_x = {}
    pos_y = {}
    for key, arr in pos_series.items():
        if key.endswith("_X"):
            pos_x[key] = arr
        elif key.endswith("_Y"):
            pos_y[key] = arr

    users = sorted(arrivals.keys(), key=lambda x: (x[0], int(x[1:]) if x[1:].isdigit() else 0))
    return Env(time_list=time_list, arrivals=arrivals, phi=phi, h_abs=h_abs, pos_x=pos_x, pos_y=pos_y), users

def user_category(name: str) -> str:
    """确定用户类别"""
    if name.startswith("U"):
        return "U"
    if name.startswith("e"):
        return "E"
    return "M"

def user_rate_bps(env: Env, bs: str, name: str, cat: str, power_alloc: Dict[str, Dict[str, float]],
                  active_sbs_same_slice: List[str], t_ms: int) -> float:
    """计算下行速率"""
    p_tx_mw = 10 ** (power_alloc[bs][cat] / 10.0)
    phi_db = env.phi_db(bs, name, t_ms)
    h_pow = env.h_pow(bs, name, t_ms)
    recv_mw = p_tx_mw * 10 ** (-phi_db / 10.0) * h_pow
```

```
177
178     interf_mw = 0.0
179     if bs in SBS_ONLY:
180         for b2 in active_sbs_same_slice:
181             if b2 == bs:
182                 continue
183             p_int_mw = 10 ** (power_alloc[b2][cat] / 10.0)
184             phi_db_i = env.phi_db(b2, name, t_ms)
185             h_pow_i = env.h_pow(b2, name, t_ms)
186             interf_mw += p_int_mw * 10 ** (-phi_db_i / 10.0) *
    h_pow_i
187
188     n0_mw = 10 ** ((-174.0 + 10.0 * math.log10(RB_PER_SLICE[
    cat] * B_HZ) + NF_DB) / 10.0)
189     sinr = recv_mw / (interf_mw + n0_mw + 1e-30)
190     return RB_PER_SLICE[cat] * B_HZ * math.log2(1.0 + sinr)
191
192 def simulate_window(env: Env, init_states: Dict[str, UserState
    ], mapping: Dict[str, str], rb_alloc: Dict[str, Dict[str,
    int]], power_alloc: Dict[str, Dict[str, float]], t0: int,
    copy_state: bool = True):
193     """带能耗计算的窗口仿真"""
194     states = {}
195     if copy_state:
196         for name, st in init_states.items():
197             states[name] = UserState(name=st.name, category=st
    .category, queue=deque(deepcopy(list(st.queue))))
198     else:
199         states = init_states
200
201     cap = {bs: {s: rb_alloc[bs][s] // RB_PER_SLICE[s] for s in
     SLICE_LIST} for bs in BS_LIST}
202     active = {bs: {s: [] for s in SLICE_LIST} for bs in
    BS_LIST}
203
```

```
    def sort_key(u: str):
        return (u[0], int(u[1:]) if u[1:].isdigit() else 0)

    order = {bs: {s: [] for s in SLICE_LIST} for bs in BS_LIST}
    for nm, bs in mapping.items():
        cat = user_category(nm)
        order[bs][cat].append(nm)
    for bs in BS_LIST:
        for s in SLICE_LIST:
            order[bs][s].sort(key=sort_key)

    sum_U = sum_E = sum_M = energy_J = 0.0
    had_m_users = set()
    success_m_users = set()

    t1 = min(t0 + WINDOW_MS, TOTAL_MS)
    for t in range(t0, t1):
        # 到达
        for nm in states.keys():
            arr_series = env.arrivals.get(nm, [])
            if t < len(arr_series):
                vol = arr_series[t]
                if vol > 0.0:
                    states[nm].queue.append(Chunk(arrival_ms=t, size_mbit=vol, remain_mbit=vol))
                    if states[nm].category == 'M':
                        had_m_users.add(nm)

        # 填充并发槽位
        for bs in BS_LIST:
            for s in SLICE_LIST:
                active[bs][s] = [u for u in active[bs][s] if states[u].has_backlog()]
                while len(active[bs][s]) < cap[bs][s]:
```

```
                    for cand in order[bs][s]:
                        if cand in active[bs][s]:
                            continue
                        if states[cand].has_backlog():
                            active[bs][s].append(cand)
                            break
                    else:
                        break

        # 服务
        for bs in BS_LIST:
            for s in SLICE_LIST:
                if len(active[bs][s]) == 0:
                    continue
                active_sbs_same_slice = [b for b in SBS_ONLY
if len(active[b][s]) > 0]
                for u in list(active[bs][s]):
                    rate_bps = user_rate_bps(env, bs, u, s,
power_alloc, active_sbs_same_slice, t)
                    served_mbit = (rate_bps * 0.001) / 1e6
                    head = states[u].queue[0]
                    if head.start_ms is None:
                        head.start_ms = t
                    head.remain_mbit -= served_mbit
                    if head.remain_mbit <= 1e-12:
                        head.remain_mbit = 0.0
                        head.finish_ms = t + 1
                        states[u].queue.popleft()
                        L_ms = (head.finish_ms - head.
arrival_ms)
                        if s == 'U':
                            sum_U += ALPHA ** L_ms if L_ms <=
SLA_L_MS['U'] else -M_U
                        elif s == 'E':
                            r_mbps = (head.size_mbit * 1000.0)
```

```
 / max(L_ms, 1e-12)
                            if L_ms <= SLA_L_MS['E'] and
r_mbps >= SLA_R_E_MBPS:
                                sum_E += 1.0
                            elif L_ms <= SLA_L_MS['E']:
                                sum_E += max(0.0, r_mbps /
SLA_R_E_MBPS)
                            else:
                                sum_E += -M_E
                        else:
                            if L_ms <= SLA_L_MS['M']:
                                success_m_users.add(u)

        # 能耗累计
        p_total_w_all_bs = 0.0
        for bs in BS_LIST:
            n_active_rb = 0
            p_tx_w_bs = 0.0
            for s in SLICE_LIST:
                n_users_act = len(active[bs][s])
                n_active_rb += n_users_act * RB_PER_SLICE[s]
                p_tx_w_bs += n_users_act * (10 ** (power_alloc
[bs][s] / 10.0) / 1000.0)  # mW to W
            p_bs = P_STATIC_W + DELTA_W_PER_RB * n_active_rb +
 (1.0 / ETA) * p_tx_w_bs
            p_total_w_all_bs += p_bs
        energy_J += p_total_w_all_bs * 0.001  # 1 ms

    # mMTC评分
    if len(had_m_users) > 0:
        ratio = len(success_m_users) / len(had_m_users)
        for u in had_m_users:
            sum_M += ratio if u in success_m_users else -M_M

    obj = sum_U + sum_E + sum_M
```

```
    result = type('SimResult', (), {'sum_U': sum_U, 'sum_E': sum_E, 'sum_M': sum_M, 'obj': obj, 'energy_J': energy_J})()
    return states, result

def nearest_sbs_per_user(env: Env, users: List[str], t_ms: int) -> Dict[str, str]:
    """计算最近SBS"""
    BS_COORD = {
        "SBS_1": (0.0, 500.0),
        "SBS_2": (-433.0127, -250.0),
        "SBS_3": (433.0127, -250.0),
    }
    out = {}
    for u in users:
        x, y = env.user_xy(u, t_ms)
        best_bs = None
        best_d2 = 1e99
        for sbs in ["SBS_1", "SBS_2", "SBS_3"]:
            bx, by = BS_COORD[sbs]
            d2 = (x - bx) ** 2 + (y - by) ** 2
            if d2 < best_d2:
                best_d2 = d2
                best_bs = sbs
        out[u] = best_bs or "SBS_1"
    return out

# 阶段一：功率优化GA
def random_individual(num_users: int):
    """阶段一个体：仅编码功率"""
    power = np.random.uniform(20.0, 30.0, size=3 * len(BS_LIST)).astype(np.float32)
    return [power]

def make_equal_rb_alloc() -> Dict[str, Dict[str, int]]:
    """生成均衡RB分配"""
```

```
    out = {}
    for bs in BS_LIST:
        remain = RB_TOTAL[bs]
        x = {"U": 0, "E": 0, "M": 0}
        order = ["U", "E", "M"]
        idx = 0
        while remain >= 2:
            s = order[idx % 3]
            need = RB_PER_SLICE[s]
            if remain >= need:
                x[s] += need
                remain -= need
            idx += 1
        out[bs] = x
    return out

def decode_power(power_arr: np.ndarray) -> Dict[str, Dict[str, float]]:
    """解码功率分配"""
    out = {}
    idx = 0
    for bs in BS_LIST:
        sub = {}
        for s in SLICE_LIST:
            p = float(power_arr[idx])
            p = max(P_MIN_DBM[bs], min(P_MAX_DBM[bs], p))
            sub[s] = p
            idx += 1
        out[bs] = sub
    return out

def evaluate_power_individual(indiv, env: Env, users: List[str], user_states: Dict[str, UserState], t0: int, nearest_map: Dict[str, str]) -> Tuple[float, float, float]:
    """阶段一评价：以能耗最小为目标"""
```

```
    power_arr = indiv[0]
    power_alloc = decode_power(power_arr)
    rb_alloc = make_equal_rb_alloc()
    mapping = {u: ("MBS_1" if nearest_map.get(u) is None else
   nearest_map[u]) for u in users}
    _, res = simulate_window(env, user_states, mapping,
   rb_alloc, power_alloc, t0, copy_state=True)
    fitness = -res.energy_J  # 最小化能耗
    return fitness, res.obj, res.energy_J

def tournament_select(pop_scores: List[float]) -> int:
    cand = random.sample(range(len(pop_scores)), TOURN_K)
    cand.sort(key=lambda idx: pop_scores[idx], reverse=True)
    return cand[0]

def crossover(parent1, parent2):
    if random.random() > CROSS_RATE:
        return parent1, parent2
    p1 = parent1[0]
    p2 = parent2[0]
    child_p = (p1 + p2) / 2.0
    return [child_p.copy()], [child_p.copy()]

def mutate(indiv):
    power_arr = indiv[0]
    for i in range(len(power_arr)):
        if random.random() < MUTATE_RATE:
            power_arr[i] += np.random.normal(0.0, 1.0)

# 阶段二：RB枚举优化
def gen_splits_for_total(total: int) -> List[Tuple[int, int,
   int]]:
    """生成合法的(U,E,M) RB分配"""
    splits = []
    for nU in range(0, total + 1, RB_PER_SLICE['U']):
```

```
393         for nE in range(0, total - nU + 1, RB_PER_SLICE['E']):
394             nM = total - nU - nE
395             if nM >= 0 and nM % RB_PER_SLICE['M'] == 0:
396                 splits.append((nU, nE, nM))
397     return splits
398 
399 def coordinate_enumeration_rb(env: Env, users: List[str],
    user_states: Dict[str, UserState], mapping: Dict[str, str],
    power_alloc: Dict[str, Dict[str, float]], t0: int, rounds:
    int = 2) -> Dict[str, Dict[str, int]]:
400     """坐标枚举RB分配"""
401     rb_alloc = make_equal_rb_alloc()
402     bs_splits = {bs: gen_splits_for_total(RB_TOTAL[bs]) for bs
     in BS_LIST}
403 
404     for _ in range(rounds):
405         for bs in BS_LIST:
406             best_obj = -1e30
407             best_split = rb_alloc[bs]['U'], rb_alloc[bs]['E'],
     rb_alloc[bs]['M']
408             for (nU, nE, nM) in bs_splits[bs]:
409                 rb_alloc[bs]['U'] = nU
410                 rb_alloc[bs]['E'] = nE
411                 rb_alloc[bs]['M'] = nM
412                 _, res = simulate_window(env, user_states,
    mapping, rb_alloc, power_alloc, t0, copy_state=True)
413                 if res.obj > best_obj:
414                     best_obj = res.obj
415                     best_split = (nU, nE, nM)
416             rb_alloc[bs]['U'], rb_alloc[bs]['E'], rb_alloc[bs
    ]['M'] = best_split
417     return rb_alloc
418 
419 def main():
420     """附录精简：第五问省略与第二问相同的MPC/仿真循环与二阶段
```

```
    细节，仅保留核心函数与接口。"""
421     print("[附录] 问题五：已省略MPC滚动仿真与二阶段完整流程，
    仅保留能耗模型/评价与接口。")
422
423 if __name__ == "__main__":
424     main()
```

附录 C　求解结果

3.1　问题一：单基站枚举结果

表 6　问题一最优 **RB** 分配方案

R_U	R_E	R_M	URLLC 得分	eMBB 得分	mMTC 得分	目标函数
10	10	30	1.98	3.80	10.00	15.77
10	20	20	1.98	3.80	10.00	15.77
10	30	10	1.98	3.80	10.00	15.77
20	10	20	1.99	3.80	10.00	15.78
20	20	10	1.99	3.80	10.00	15.78
30	10	10	1.99	3.80	10.00	15.78

3.2 问题二：单微基站 MPC 滚动窗口最优决策

表 7　问题二各窗口最优 **RB** 分配决策

窗口	R_U	R_E	R_M	URLLC 得分	eMBB 得分	mMTC 得分	目标函数
0	10	20	20	22.75	32.74	10.00	65.49
1	10	20	20	21.71	8.32	10.00	40.03
2	10	20	20	23.61	5.23	10.00	38.83
3	10	0	40	22.80	0.00	10.00	32.80
4	40	0	10	21.85	0.00	10.00	31.85
5	40	0	10	14.25	0.00	4.40	18.65
6	40	0	10	17.10	0.00	7.10	24.20
7	40	0	10	22.80	0.00	7.10	29.90
8	40	0	10	21.85	0.00	10.00	31.85
9	40	0	10	17.10	0.00	7.10	24.20
累计目标函数				337.42			

3.3 问题三：多基站 GA-MPC 优化结果

表 8　问题三各窗口多基站资源分配结果

窗口	BS1-RB 分配	BS1-功率 (dBm)	BS2-RB 分配	BS2-功率 (dBm)	BS3-RB 分配	BS3-功率 (dBm)	URLLC	eMBB	mMTC	目标
0	(0,40,10)	(24.4,22.8,24.8)	(20,0,30)	(21.6,22.9,23.4)	(20,0,30)	(21.6,22.9,21.7)	38.16	8.40	30.00	76.56
1	(10,0,40)	(21.1,21.0,25.6)	(10,20,20)	(20.2,24.2,20.9)	(20,20,10)	(21.7,24.3,23.2)	51.03	6.81	30.00	87.84
2	(10,0,40)	(21.9,20.1,25.1)	(0,40,10)	(18.9,25.4,22.7)	(20,0,30)	(20.5,24.8,21.0)	38.35	9.87	30.00	78.22
3	(30,0,20)	(19.9,22.5,18.0)	(20,0,30)	(20.4,20.3,22.4)	(0,30,20)	(19.4,23.7,20.2)	49.26	7.79	30.00	87.05
4	(10,20,20)	(23.2,26.2,24.0)	(30,0,20)	(21.6,20.7,21.6)	(10,0,40)	(20.5,21.8,22.2)	46.43	5.90	30.00	82.33
5	(0,30,20)	(19.2,24.1,21.4)	(20,0,30)	(21.3,21.3,20.2)	(30,0,20)	(22.4,21.3,22.8)	52.02	7.28	30.00	89.30
6	(10,0,40)	(21.9,21.3,20.6)	(10,0,40)	(22.5,21.5,22.7)	(20,20,10)	(21.3,25.5,21.0)	52.08	6.25	30.00	88.33
7	(10,0,40)	(21.6,22.9,22.0)	(10,20,20)	(19.2,26.2,17.5)	(20,0,30)	(24.7,21.5,20.6)	48.68	6.40	30.00	85.08
8	(20,0,30)	(22.5,18.6,22.9)	(20,0,30)	(18.7,17.9,23.3)	(10,20,20)	(22.3,29.8,23.0)	51.45	7.59	30.00	89.04
9	(0,30,20)	(21.4,27.3,22.7)	(30,0,20)	(24.5,22.0,20.7)	(10,0,40)	(21.6,23.2,21.1)	56.80	7.88	30.00	94.69
累计目标函数							853.37			

3.4 问题四：异构网络 GA-MPC 优化结果

表 **9** 问题四异构网络前 **5** 窗口资源分配结果

窗口	MBS_1-RB 分配	MBS_1-功率 (dBm)	SBS_1-RB 分配	SBS_2-RB 分配	SBS_3-RB 分配	URLLC	eMBB	目标
0	(30,40,30)	(25.1,23.8,24.6)	(20,20,10)	(20,0,30)	(20,0,30)	93.42	14.26	147.68
1	(30,0,70)	(25.7,25.2,26.9)	(0,0,50)	(30,0,20)	(10,0,40)	85.68	0.00	125.68
2	(50,0,50)	(25.8,24.1,22.9)	(0,0,50)	(0,0,50)	(10,0,40)	64.61	0.00	104.61
3	(0,0,100)	(22.5,22.7,23.3)	(30,0,20)	(30,0,20)	(20,0,30)	50.85	0.00	90.85
4	(0,0,100)	(24.9,24.8,27.8)	(0,0,50)	(30,0,20)	(20,0,30)	43.16	0.00	83.16
5	(0,0,100)	(21.2,26.5,25.9)	(30,0,20)	(30,0,20)	(30,0,20)	50.27	0.00	90.27
6	(0,0,100)	(22.0,29.6,22.4)	(20,0,30)	(30,0,20)	(20,0,30)	47.51	0.00	87.51
7	(0,0,100)	(27.4,24.9,29.0)	(20,0,30)	(30,0,20)	(30,0,20)	48.32	0.00	82.42
8	(0,0,100)	(23.2,22.8,26.9)	(10,0,40)	(20,0,30)	(0,0,50)	19.48	0.00	29.50
9	(0,0,100)	(22.3,23.9,29.8)	(30,0,20)	(50,0,0)	(0,0,50)	11.74	0.00	-7.64
累计目标函数						1041.28		

3.5 问题五：能耗优化 GA-MPC 结果

表 **10** 问题五能耗优化结果（前 **5** 窗口）

窗口	MBS_1-RB	SBS_1-RB	SBS_2-RB	SBS_3-RB	URLLC	eMBB	mMTC	目标	能耗 (J)
0	(0,0,100)	(10,10,30)	(30,0,20)	(0,10,40)	54.25	1.50	40.00	95.75	19.68
1	(0,0,100)	(0,0,50)	(30,0,20)	(0,0,50)	35.11	0.00	37.03	72.13	18.10
2	(0,0,100)	(0,0,50)	(30,0,20)	(0,0,50)	3.34	0.00	37.03	40.37	18.12
3	(0,0,100)	(0,0,50)	(20,0,30)	(0,0,50)	52.88	0.00	40.00	92.88	18.25
4	(0,0,100)	(0,0,50)	(30,0,20)	(0,0,50)	24.74	0.00	37.03	61.77	17.97
5	(0,0,100)	(0,0,50)	(30,0,20)	(0,0,50)	46.51	0.00	37.03	83.54	18.10
6	(0,0,100)	(0,10,40)	(30,10,10)	(0,10,40)	47.89	-42.00	7.60	13.49	19.48
7	(0,0,100)	(0,0,50)	(30,0,20)	(0,0,50)	36.74	0.00	-33.10	3.64	18.09
8	(0,0,100)	(0,0,50)	(30,0,20)	(0,0,50)	28.47	0.00	-40.00	-11.53	17.84
9	(0,0,100)	(0,0,50)	(30,0,20)	(0,0,50)	16.55	0.00	-40.00	-23.45	18.05
累计目标函数					381.17			总能耗:183.68J	