哈爾濱工業大學

本科毕业论文（设计）中期报告

题　目：车辆驾驶安全场景下基于微服务的任务卸载优化方法

专　　业　软件工程

学　　生　张儒

学　　号　2021112678

指导教师　贺祥

日　　期　2025 年 3 月 3 日

哈尔滨工业大学教务处制

目　录

一、论文工作是否按开题报告预定的内容及进度安排进行

开题报告中工作进度安排如下：

工作安排	周数	起止时间
相关文献的研究和调研	2	2024.11.20-2024.12.04
系统模型建立	3	2024.12.05-2025.01.01
任务卸载算法设计	4	2025.01.01-2025.01.30
数据采集以及仿真实验	4	2025.02.01-2025.02.28
性能分析与优化验证	4	2025.03.01-2025.03.28
结题，论文编写	2	2025.03.29-2025.04.15

阶段目标安排如下：

（1）中期目标：完成系统模型（网络模型、任务模型、多计算层级协同的计算模型）的建立以及优化问题的形式化表达。

（2）结题目标：实现具体的任务卸载模型和算法，并进行仿真实验验证所设计的卸载策略性能以及经济效益。

目前仍在进行任务卸载算法设计阶段，但已完成中期目标。

二、已完成的研究工作及成果

2.1 已完成的研究工作

目前，已成功将现实问题进行了深入的数学建模以及精准的问题形式化。并采用基于遗传算法 (GA) 的方法，对问题进行求解，在不同的场景下对数据建模进行了验证以及与 Gubori 求解器进行对比。在实验支撑方面，开发了基于 TC 命令的网络带宽控制工具来控制驾驶环境下车辆与 RSU 之间的带宽等网络状况，为后续的实验验证奠定了坚实基础。

2.1.1 系统建模

在系统建模方面，建立了多级计算资源协同的三层系统数学模型。该模型全面且细致地刻画了车联网系统中数据传输、任务结构以及计算资源分配等关键要素之间的关系，为后续的算法设计和性能分析提供了模型支撑。关键部分如下：

为了模拟系统运行时状态的变化，将系统的运行时间划分为 h 个时隙 $\mathcal{T} = \{t_1, t_2, ..., t_h\}$，时隙的长度为 t。在同一个时隙内，系统的状态可以看作是不变的，但在多个时隙间是变化的。

系统中共有 i 辆车，记作 $L = \{L_1, L_2, \ldots, L_i\}$，其中 $L_l = (V_l, P_l, c_l)$，分别表示汽车的速度、位置以及计算设备的处理能力。边缘节点 RSU 配备有边缘计算服务器，表示为 $E = \{P_e, c_e, \beta_e\}$，分别表示 RSU 的位置、拥有的计算资源以及带宽资

源。云服务器 $C=\{P_c, c_c, \rho_c\}$，表示云服务器的位置、分配给每个任务的计算能力以及计算每比特的价格。

车载设备持续采集视频数据，被划分为固定时长为 κ 的视频块，每个视频块对应一个处理任务。也就是每隔 κ 的时间，车辆 l 就会产生一个任务 T_l，由多个含有依赖关系的子任务构成，表示为:$Q_l=\{T_l, \mathcal{E}_l\}$。其中任务 $T_l=\{A_l^1, A_l^2....A_l^k\}$，表示汽车 l 的 k 个子任务，每个子任务定义为

$$A_l^m=(x_l^m, d_l^m, w_l^m, c_l^m, t_{A_l^m}^{start}, t_{A_l^m}^{end}) \tag{2-1}$$

式中 x_l^m———任务的输入数据大小 (比特)；

d_l^m———任务的输出数据大小 (比特)；

w_l^m———任务的工作量 (CPU 周期)；

c_l^m———分配给任务 A_l^m 的计算资源 (CPU 周期/秒)；

$t_{A_l^m}^{start}$———任务 A_l^m 的开始时隙；

$t_{A_l^m}^{end}$———任务 A_l^m 的结束时隙；

任务 T_l 的最后截止时间为 t_l，为防止任务积压，$t_l=\kappa$，在下一个视频块任务到来之前，当前任务要处理完成。S_l 表示任务 T_l 的卸载策略:

$$S_l=\{s_l^1, s_l^2, s_l^3, ..., s_l^k\} \tag{2-2}$$

其中 $s_l^m \in \{0,1,2\}$，分别表示本地、边缘服务器、云服务器处理任务。子任务之间的依赖关系可以表示为:

$$\mathcal{E}_l=\{e_{A_l^m, A_l^n} | (m,n) \in \{1,2,3,...,k\} \times \{1,2,3,...,k\}\} \tag{2-3}$$

表示汽车 l 的子任务之间的依赖关系。若 $\mathcal{E}_l$ 包含依赖关系 $e_{A_l^m, A_l^n}$，那么表示只有 A_l^m 完成之后，任务 A_l^n 才能开始执行。

任务时延与任务处理方式密切相关，如果子任务在本地进行处理，子任务的时延只包括处理时延:

$$t_{A_l^i}^{loc}=\frac{w_l^m}{c_l^m \cdot \Delta t} \tag{2-4}$$

如果被卸载到边缘服务器，那么子任务卸载时延包括：任务上传时延以及计算时延，其中上传数据延迟:

$$t_{up}^{edg}=\min\{t: \sum_{\tau=t_{A_{l,t}^m}^{start}}^{t} r_{l,e}(\tau) \times \Delta t \geq x_{l,t}^m\} \tag{2-5}$$

其中 $r_{l,e}(\tau)$ 表示车辆 l 与边缘服务器 e 在时隙 τ 的传输速率，具体表示为：

$$r_{l,e}(\tau) = \beta_{l,e} \log_2 \left(1 + \frac{S_l \cdot \lambda_{l,e}(\tau)}{\sigma^2}\right) \tag{2-6}$$

式中 $\beta_{l,e}$———车辆 l 和边缘服务器 e 之间的带宽；

S_l———车辆 l 的发生功率；

$\lambda_{l,e}(\tau)$———车辆 l 与边缘服务器 e 在时隙 τ 的信道增益；

σ^2———周围噪声功率；

计算延迟与本地处理计算方式相同。卸载到云服务器时，时延包括任务上传时延、计算时延以及传播时延。上传数据时延和处理时延与本地处理、边缘服务器上传时延类似，考虑到云服务器距离车辆较远，因此传播时延不能被忽略。根据上述这些公式可以计算出各个子任务的完成时间，那么任务 T_l 的完成时间即为最后一个子任务完成的时间：

$$T_l^{\text{delay}} = \max_m \left(t_{A_l^m}^{\text{end}}\right) \tag{2-7}$$

任务的总成本 T_l^{cost} 表示为各个子任务成本之和：

$$T_l^{cost} = \gamma + \sum_{m=1}^{k} f_l^m \tag{2-8}$$

其中，γ 表示租赁边缘服务器的固定成本。f_l^m 表示子任务的成本，子任务的成本与其处理方式密切相关，本地处理时，企业不消耗成本；在边缘服务器处理时，企业按期租赁边缘服务器，不含有额外成本；在云服务器处理时，采用按需计费，根据任务的任务量以及数据量大小付费，具体表示为:

$$f_l^m = \begin{cases} 0, & \text{if } s_l^m = 0, 1 \\ \epsilon \times x_l^i + \rho_c \times w_l^i, & \text{if } s_l^m = 2 \end{cases} \tag{2-9}$$

式中 ϵ———光纤通信每 bit 的费用；

ρ_c———云服务器计算每 bit 的费用；

优化问题的目标就是在任务按时完成的前提下，最小化企业成本，因此优化问题

可以形式化表达为：

$$
\begin{aligned}
& \min T_l^{cost} \\
\text{s.t.} \quad & t_{A_l^m}^{end} + 1 \le t_{A_l^n}^{start}, \quad \forall e_{A_l^m, A_l^n} \in \mathcal{E}_{\updownarrow} \\
& \max_{m \in \{1,2,\ldots,k\}} \left(t_{A_l^m}^{\text{end}} \right) \le \kappa \\
& s_l^m \in \{0, 1, 2\}, \quad \forall m \in \{1, 2, ..., k\} \\
& \sum_{l=1}^{i} \sum_{m=1}^{k} I(s_l^m = 1) \cdot c_l^m \le c_e, \\
& \sum_{l=1}^{i} \sum_{m=1}^{k} [I(s_l^m = 1) + I(s_l^m = 2)] \cdot \beta_{l,e} \le \beta_e
\end{aligned}
$$

2.1.2 问题求解

针对上述优化问题，本研究采用基于遗传算法 (GA) 的方法进行求解。GA 算法在求解 NP-hard 混合整数非线性规划 (MINLP) 问题中表现出显著优势，其核心价值体现在全局优化机制与适应性处理能力上。MINLP 问题通常融合整数变量、连续变量、非线性目标函数及复杂约束条件，传统梯度类算法往往局限于局部最优解且难以有效处理离散变量。而基于 GA 的算法通过其基于种群的并行搜索策略巧妙规避了这一局限：

（1）多方向探索：通过模拟生物进化中的选择、交叉和变异操作，算法能够在解空间中实现多点并行探索，有效避免对初始解或梯度信息的依赖，增强了在复杂任务调度场景中找到全局最优解的能力。

（2）混合变量处理能力：遗传算法天然支持整数与连续变量并存的混合类型处理，并能通过自适应惩罚函数、约束松弛等机制高效应对非光滑、非凸及离散约束，这对于处理车联网环境中计算资源与带宽分配的多类型变量尤为重要。

（3）高效近似优化特性：针对 NP-hard 问题中的指数级计算复杂度挑战，遗传算法以启发式近似优化为核心，能在有限计算时间内提供高质量可行解，实现了计算效率与工程实用性的平衡，满足车联网环境下对实时性的严格要求。

以 GA 算法为基础，结合启发式的思想，可以有效处理本研究中的车联网任务调度优化问题。通过融合问题领域的专业知识与遗传算法的自适应搜索能力，在处理高维解空间和多目标优化方面展现出独特优势。特别是针对车联网环境下任务间的复杂依赖关系、异构计算资源的动态分配以及时间约束的严格要求，本研究设计的 GA 框架不仅保持了经典遗传算法的全局寻优特性，还通过引入关键路

径分析和依赖感知机制增强了算法的问题特定导向性。保证了在搜索效率的同时提高解的质量，尤其是在有限时间内为复杂的 NP-hard 问题提供高质量的近似最优解，实现了理论优化与工程实用之间的有效平衡。遗传算法的基本流程如图所示，本研究在此基础上进行了针对性的改进与优化。

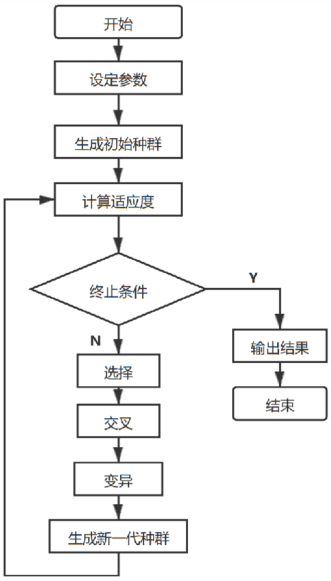

图 2-1: 遗传算法基本流程

在编码设计阶段，每个个体的编码结构包含所有车辆的全部子任务决策信息，其中每个子任务由三个连续基因位精确表示：决策位置、计算资源分配量和带宽分配量。这种编码方式直观映射了任务的调度方案全貌。本研究创新性地引入关键路径识别机制，通过计算最早开始时间 (EST) 与最晚开始时间 (LST) 精确识别对总完成时间影响最大的关键任务链。在初始种群生成过程中，算法综合考量依赖深度和关键路径属性，对任务进行优先级排序，使关键路径上的任务获得更为迅速的处理资源。通过依赖感知的动态绑定策略，系统赋予 70% 的倾向性使连续依赖的任务分配到相同处理节点，显著减少中间数据传输开销。当监测到 RSU 计算资源相对紧张时，系统会通过设定权重系数来精确控制任务分配到本地、RSU 和云端的概率分布，有效避免初始阶段 RSU 资源的过度分配。

遗传阶段的精细设计充分考虑了问题的特殊性和实际约束。选择操作采用锦标赛选择机制，每轮从种群中随机抽取多个候选个体进行适应度比较，选取最优个体进入下一代，这种机制既维持了适当的选择压力，又有效预防了遗传多样性的过早丧失；同时还结合了精英保存策略，确保历史最优解不会在迭代过程中丢失，为后续的交叉与变异操作奠定多样化的基因基础。

交叉操作不仅以车辆为单位进行整体信息交换，更融入了依赖感知机制，通过识别任务间的依赖关系优化交叉点的选择。对于存在依赖关系的任务链，系统倾向于作为整体单元进行交换，保持依赖任务决策的一致性与完整性。在交叉概率控制方面，算法根据父代个体的适应度差异动态调整交叉强度，当父代个体质量相近时增加交叉探索力度，反之则更多保留优质个体特征，这种自适应机制显著提高了交叉操作的有效性与算法收敛效率。

变异操作实现了差异化处理策略，为不同类型的任务设计专属变异机制。对关键路径上的任务，算法偏向选择计算能力更强的处理节点，并给予更高的资源分配优先级；对依赖链上的任务，保持较高的位置一致性概率，减少不必要的数据传输开销。此外，当决策位置发生变异时，系统联动更新资源分配策略，例如任务从 RSU 迁移至云端处理时，计算资源分配自动置零而保留必要带宽，这种多层次的智能变异机制确保了每一次基因变化都能产生实际可行且有意义的解，有效提升了算法的探索效率。

适应度评估引入一致性奖励与关键路径优化双重激励机制，通过精确衡量依赖任务处理位置的一致性比率以及关键路径的优化程度，为算法提供更精准的进化导向，显著提升了解决方案的质量与执行效率。下面为基于 GA 的任务卸载算法的伪代码：

Algorithm 1 基于遗传算法的任务分配

Require: 种群大小 P, 迭代次数 G, 用户集合 U, 任务集合 T, 资源限制 R

Ensure: 最优决策矩阵 X, 资源分配矩阵 R_{alloc}, 总成本 $Cost$

$DG \leftarrow$ 构建任务依赖图 (U, T); $population \leftarrow$ 初始化种群 (P, U, T, DG)
$bestIndividual \leftarrow null$; $bestFitness \leftarrow -\infty$
for $g = 1$ **to** G **do**
  $fitness \leftarrow$ 评估种群适应度 $(population)$; $bestIdx \leftarrow \arg\max(fitness)$
  if $fitness[bestIdx] > bestFitness$ **then**
    $bestIndividual \leftarrow population[bestIdx]$; $bestFitness \leftarrow fitness[bestIdx]$
  end if
  $selected \leftarrow$ 锦标赛选择 $(population, fitness)$; $newPopulation \leftarrow []$
  $i \leftarrow 0$
  while $i < |selected|$ **do**
    if $i + 1 < |selected|$ **then**
      $child1, child2 \leftarrow$ 依赖感知交叉 $(selected[i], selected[i+1])$
      $child1 \leftarrow$ 依赖感知变异 $(child1)$; $child2 \leftarrow$ 依赖感知变异 $(child2)$
      $newPopulation \leftarrow newPopulation \cup \{child1, child2\}$
    else
      $newPopulation \leftarrow newPopulation \cup \{selected[i]\}$
    end if
    $i \leftarrow i + 2$
  end while
  if $bestIndividual \neq null$ **then**
    $newPopulation[0] \leftarrow bestIndividual$ {精英保留}
  end if
  $population \leftarrow newPopulation[0 : P]$
end for
$X, R_{alloc}, B \leftarrow$ 解码个体 $(bestIndividual)$
$Cost \leftarrow$ 计算总成本 (X, R_{alloc}, B);
return $X, R_{alloc}, Cost$

Algorithm 2 依赖感知交叉

Require: 父代个体 $parent1$, $parent2$, 交叉概率 p_c, 任务依赖图 DG

Ensure: 子代个体 $child1$, $child2$

if $random() > p_c$ **then**
  return $parent1.copy(), parent2.copy()$
end if
$criticalTasks \leftarrow$ 识别关键路径 ()
$child1 \leftarrow parent1.copy(); child2 \leftarrow parent2.copy()$
$criticalVehicles \leftarrow$ 获取关键任务所属车辆 ()
for $car \in$ 所有车辆 **do**
  if $car \notin criticalVehicles$ **and** $random() < 0.5$ **then**
    交换 car 的所有任务决策和资源分配
  end if
end for
for $(car, task) \in criticalTasks$ **do**
  $deps \leftarrow DG.getDependencies(car, task)$
  if $deps \neq \emptyset$ **then**
    $depDecisions1 \leftarrow$ 获取 $child1$ 中依赖任务的决策
    $depDecisions2 \leftarrow$ 获取 $child2$ 中依赖任务的决策
    if $random() < 0.8$ **and** $depDecisions1 \neq \emptyset$ **then**
      $commonDecision \leftarrow$ 获取 $depDecisions1$ 中出现最多的决策
      更新 $child1$ 中 $(car, task)$ 的决策为 $commonDecision$ 并调整资源
    end if
    if $random() < 0.8$ **and** $depDecisions2 \neq \emptyset$ **then**
      $commonDecision \leftarrow$ 获取 $depDecisions2$ 中出现最多的决策
      更新 $child2$ 中 $(car, task)$ 的决策为 $commonDecision$ 并调整资源
    end if
  end if
end for
return $child1, child2$

Algorithm 3 依赖感知变异

Require: 个体 $individual$, 变异率 p_m, 任务依赖图 DG

Ensure: 变异后个体 $mutated$

$mutated \leftarrow individual.copy()$
$criticalTasks \leftarrow$ 识别关键路径 ()
$depthMap \leftarrow$ 计算任务依赖深度 ()
for $car \in$ 所有车辆 **do**
  for $task \in$ 所有任务 **do**
    if $random() < p_m$ **then**
      $currentDecision \leftarrow mutated[car, task].decision$
      $deps \leftarrow DG.getDependencies(car, task)$
      $depDecisions \leftarrow$ 获取依赖任务的决策
      if $(car, task) \in criticalTasks$ **then**
        $weights \leftarrow [0.1, 0.6, 0.3]$ {降低本地处理概率，提高边缘处理}
        $newDecision \leftarrow$ 基于权重随机选择新决策
      else if $depDecisions \neq \emptyset$ **and** $random() < 0.8$ **then**
        $newDecision \leftarrow$ 获取依赖任务中出现最多的决策
      else
        $newDecision \leftarrow$ 随机选择一个不同于当前的决策
      end if
      更新 $mutated[car, task]$ 的决策为 $newDecision$
      根据新决策调整计算资源和带宽分配
    end if
  end for
end for
return $mutated$

2.1.3 实验工具

在车联网场景中，V2I 通信的带宽是动态变化的。为验证通信模型对动态带宽的适应性，本研究开发了基于 TC 命令的带宽控制工具。TC 命令是一个强大的流量控制工具，可以实现对指定网络包的速度、顺序以及出入网卡的精细调控。我们可以轻松地完成诸如按流量类型限速、限制特定应用程序或网络接口的带宽使用等任务。此外，它还能模拟网络延迟和丢包等故障情况，为系统开发提供有效的测试手段，本研究中可以用于模拟车联网场景中动态变化的网络环境并验证任务卸载策略的适应性。

TC 命令以物理网卡（如 eth0）为流量管控起点，通过绑定层次化令牌桶（HTB）类型的主队列规则（QDisc）建立带宽管理框架，首先在主分类中设定全局带宽阈值，随后将其细分为多个子分类，每个子分类可以独立配置带宽限制和延迟参数，最终通过过滤器（Filter）将不同子分类与特定目标 IP 的路由规则动态绑定，从而实现对网络流量的多级分层管控。

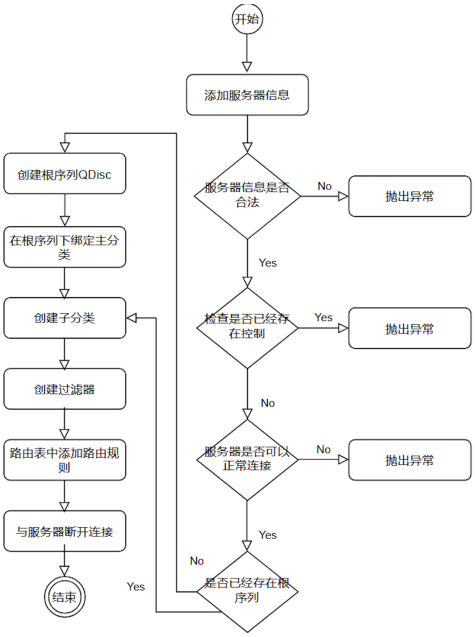

图 2-2: TC 网络控制流程图

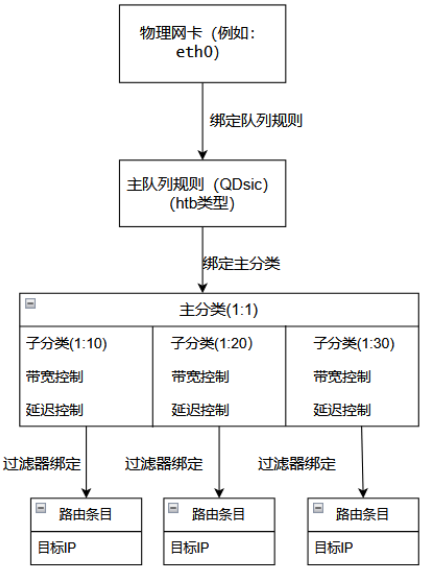

图 2-3: TC 命令实现网络控制架构

根据上述具体流程图以及原理架构图，结合软件工程的思想以及模块化分层思想，设计出类图，构建了一个完整的网络带宽控制系统架构。整体设计分为六个关键组件，每个组件承担特定职责并通过明确的关系相互协作。

服务器基础模块 (Server) 作为系统基础，提供网络服务器的基本信息如名称、IP 地址和端口配置。控制模块 (Control) 作为核心协调层，管理本地 IP、目标 IP、带宽参数和延迟策略，通过” 设置控制” 关系与服务器连接，并负责初始化系统配

置。队列调度模块 (QDisc) 处理网络流量的排队与调度逻辑，被控制模块” 绑定” 后，负责实现具体的带宽控制策略。流量分类模块 (Clazz) 实现细粒度的带宽管理，包含类型、名称、传输速率和单元参数，并通过” 绑定子分类” 关系形成层次化的分类体系。路由模块 (Route) 维护网络路径信息，连接目标 IP 与本地 IP，并通过” 设置” 关系与流量分类模块协同工作。过滤模块 (Filter) 则负责流量筛选，基于优先级、链条和处理规则来识别和分类数据包。

这种设计形成了从全局配置到细粒度控制的完整链路：服务器配置驱动控制参数，控制参数初始化队列策略，队列策略通过过滤规则和分类方案实现精确的流量管理。各模块间的关系（一对多、一对一、绑定、设置等）清晰定义了系统的交互边界，确保了架构的灵活性和可扩展性。

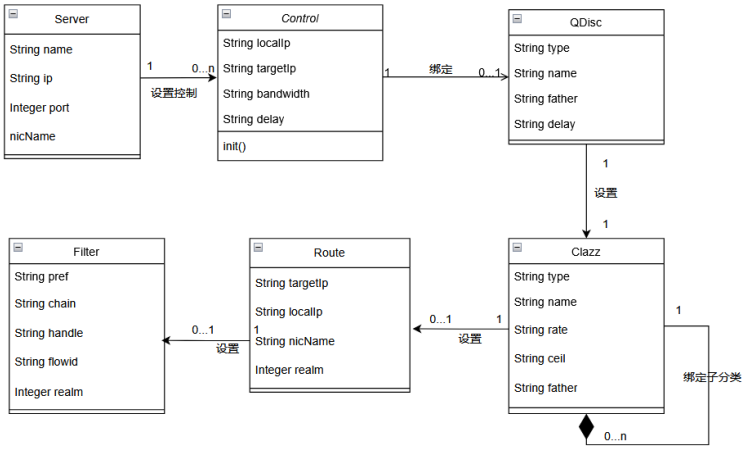

图 2-4: TC 网络控制系统类图

在技术栈选型上，后端使用 SpringBoot 框架提供稳定高效的 REST API 服务，通过 Jsch 安全 shell 库实现与目标服务器的安全连接与命令执行。后端将用户配置的带宽与延迟参数转换为标准 TC 命令，由 Jsch 执行并返回结果，实现了对网络接口的队列规则、流量分类和带宽限制的精确控制。最终采用 Docker 与 k8s 相结合的部署方案，实现了系统的弹性部署与集中化管理。

2.2 成果

2.2.1 算法求解

在测试场景中，考虑单个 RSU 覆盖范围内的 3 辆货车，每辆货车会持续采集视频数据，并将视频数据划分为固定时长 2 秒的视频块进行处理。将系统单次任务处理的运行时间划分为 20 个时隙，每个时隙长度为 0.1 秒。每个视频块对应一个处理任务，该任务可以进一步分解为 4 个具有依赖关系的子任务，使用有向无

环图 (DAG) 来表示，不同车辆的子任务可以具有不同的 DAG 结构，比如车辆 0 采用链式结构，车辆 1 采用分支合并结构，车辆 2 则采用较为复杂的混合结构：

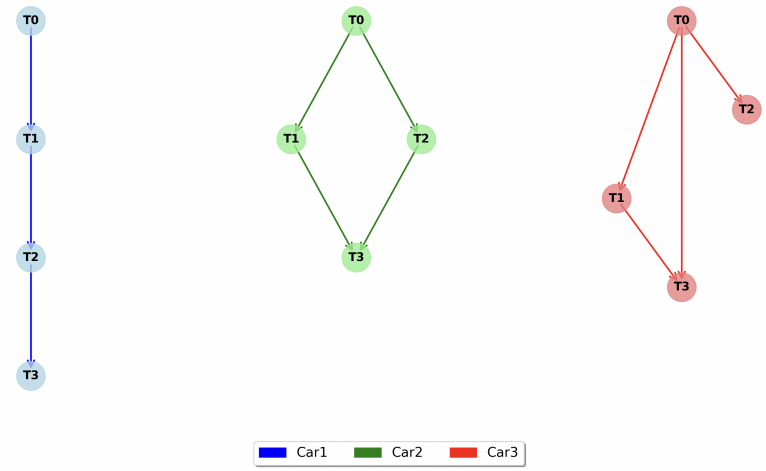

图 2-5: 子任务依赖关系图

每个子任务都有各自的的输入、输出数据大小和计算工作量。在计算资源方面，场景中部署了一个 RSU，位于道路两侧的位置，具有特定的计算能力和带宽资源。此外，还有一个位于远端的云服务器，提供丰富的计算能力。三辆车的初始位置和运动轨迹各不相同，但均处于 RSU 的覆盖范围内。每隔 2s，车辆生成一个视频块处理任务，向 RSU 的决策设备发送请求。决策设备需要在考虑任务依赖关系、资源容量约束、通信延迟等因素的基础上，为每个子任务确定最优的执行位置（本地、RSU 或云端）和资源分配方案，确保所有任务能在 2 秒的截止时间内完成（防止任务积压），同时最小化系统成本。采用 GA 算法对各个任务进行卸载决策，运行结果如下：

车辆	子任务	执行位置	计算资源	带宽	开始时间	结束时间	执行延迟
汽车1	T0	RSU	2376.73	2089.81	0.00	0.46	0.46
汽车1	T1	RSU	1746.1	568.01	0.46	1.18	0.72
汽车1	T2	RSU	1795.41	2006.27	1.18	1.78	0.60
汽车1	T3	云	-	1629.09	1.78	1.89	0.11
汽车2	T0	RSU	2870.46	2234.65	0.00	0.40	0.40
汽车2	T1	RSU	1432.2	1311.7	0.40	1.18	0.78
汽车2	T2	RSU	1813.32	944.83	0.40	1.07	0.67
汽车2	T3	RSU	2947.2	2298.97	1.18	1.57	0.39
汽车3	T0	RSU	2150.71	855.56	0.00	0.60	0.60
汽车3	T1	RSU	1812.47	1006.67	0.60	1.26	0.66
汽车3	T2	RSU	1847.53	289.51	0.60	1.52	0.93
汽车3	T3	RSU	2088.13	1066.94	1.26	1.84	0.58

图 2-6: GA 任务分配与详细执行时间

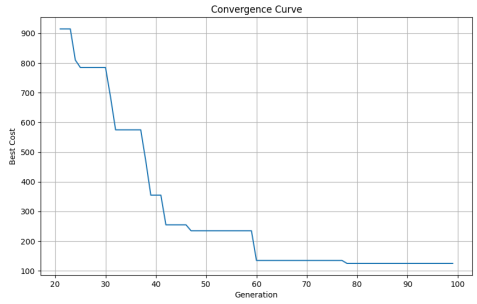

图 2-7: 适应度变化曲线

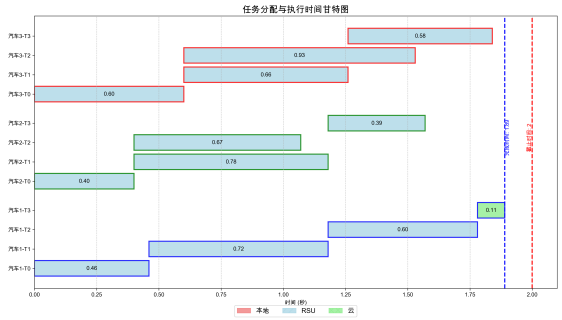

图 2-8: 任务运行时间甘特图

相同的场景下，选择 Gurobi 求解器作为对比基准。Gurobi 凭借其高度优化的单纯形法、内点法和分支定界算法，能够为复杂优化问题提供数学上严格的全局最优解，这使其成为评估启发式算法性能的理想参照系。可以量化评估基于 GA 方法的解与理论最优解之间的差距，还能验证启发式方法在计算效率和扩展性方面的优势。使用 Gubori 求解器进行求解，运行结果如下：

车辆	子任务	执行位置	计算资源	带宽	开始时间	结束时间	执行延迟
汽车1	T0	RSU	2501.2383	2000.0	0.00	0.53	0.53
汽车1	T1	RSU	10000.0	8000.0	0.95	1.26	0.32
汽车1	T2	RSU	5932.5711	8000.0	1.26	1.46	0.20
汽车1	T3	RSU	5938.2765	8000.0	1.58	1.78	0.20
汽车2	T0	RSU	3554.3544	3333.3333	0.00	0.32	0.32
汽车2	T1	RSU	5818.3843	5000.0	0.32	0.52	0.20
汽车2	T2	RSU	5884.1796	8000.0	0.43	0.63	0.20
汽车2	T3	RSU	5971.6783	8000.0	1.16	1.36	0.20
汽车3	T0	RSU	3201.2351	2500.0	0.00	0.40	0.40
汽车3	T1	RSU	5830.1813	8000.0	1.38	1.58	0.20
汽车3	T2	RSU	5232.4497	8000.0	0.53	0.74	0.21
汽车3	T3	RSU	5832.3873	8000.0	1.68	1.88	0.20

图 2-9: 求解器任务分配与详细执行时间

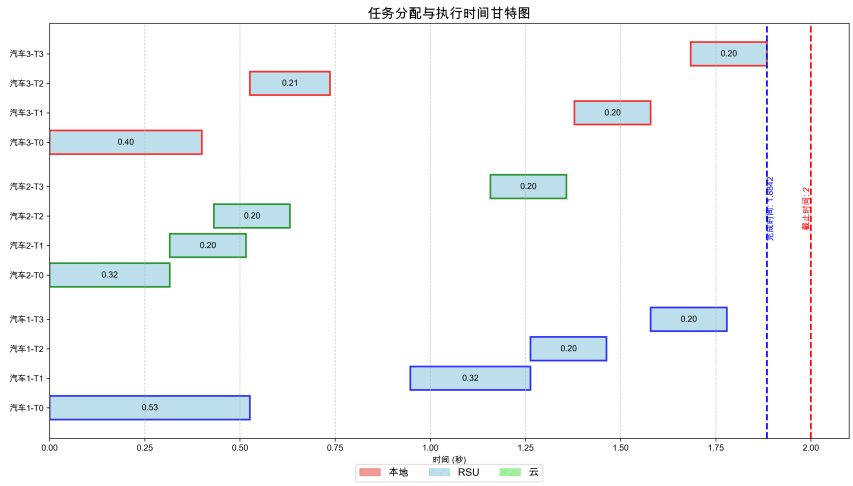

图 2-10: 求解器任务分配与执行时间甘特图

在上述场景下，多次运行我们所设计的算法以及 Gubori 求解器，记录每次的执行时间以及任务成本。之后，在保持系统其余参数不变的情况下，将车辆数目扩充到 5 辆，再次运行两者并统计出运行时间以及处理成本的平均值。整理得到两种场景下任务结果的响应时间和任务成本：

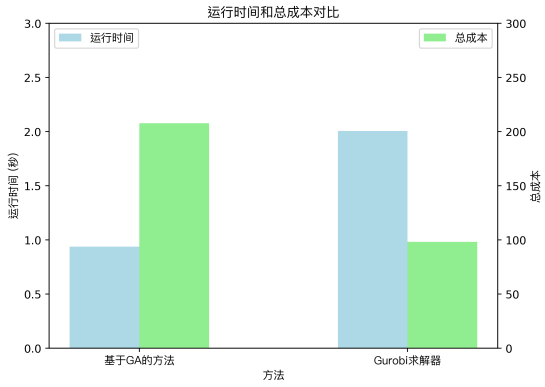

图 2-11: 3 辆车场景下数据对比

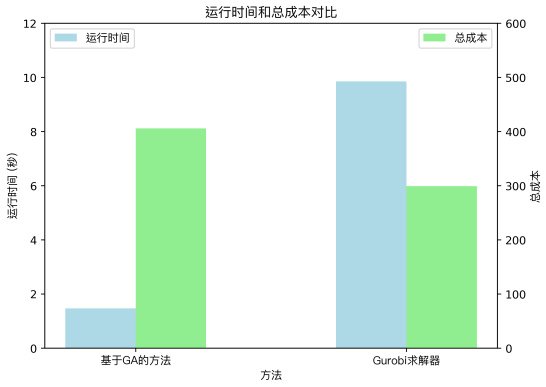

图 2-12: 5 辆车场景下数据对比

可以看到 Gurobi 求解器和基于 GA 的方法两者各有优劣。Gurobi 求解器作为精确求解方法，能够找到全局最优解，从结果来看，3 辆车和 5 辆车的场景下，其得到的总成本均比基于 GA 的方法要少，通过合理分配资源，能够最大限度的利用边缘服务器的计算资源。然而，求解器的计算时间会随着问题规模的增长呈指数级增长，在 3 辆车的场景下，求解器的求解时间为 2s 左右，但是在 5 辆车的场景下，求解时间直接增长到了 10s，在实际应用中是远远无法满足实时性需求。相比之下，基于 GA 算法的启发式方法虽然任务成本要高于求解器，但其具有明显的计算效率优势，可以在较短的时间内求解。特别是在实际应用场景中，当车辆数量增加、任务依赖关系复杂化时，GA 算法仍能在可接受的时间内给出可行解，这种快速响应能力更符合车联网环境下对实时性的要求。总的来说，求解器适合离线规划或小规模问题的精确求解，而遗传算法则更适合在线实时调度场景，特别是在大规模任务调度问题中，其计算效率的优势将更加明显。

2.2.2 工具演示

下图为基于 TC 命令的带宽控制工具相关前端页面：

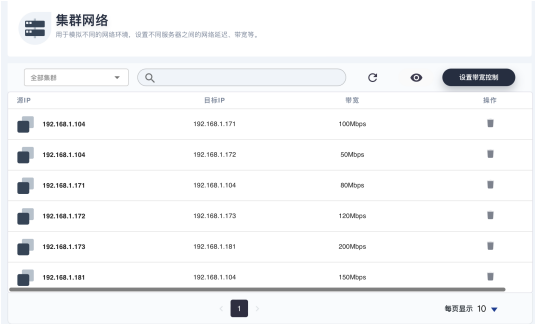

图 2-13: 查看网络控制信息

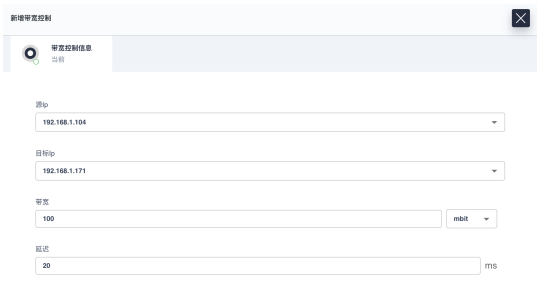

图 2-14: 添加网络控制

要模拟研究环境中车辆和边缘服务器之间的网络变化，可以按照以下步骤模拟：每隔 Δt 的时间，根据当前时隙下车辆和 RSU 之间的带宽、信噪比等参数，计算出车辆和 RSU 之间的传输速率，并向设置带宽的接口发出请求，模拟出车辆移动时，车辆与边缘服务器之间的传输速率。关键的部分代码如下：

```
try:
    while True:
        # 信噪比随机变化
        snr = random.uniform(min_snr, max_snr)
        # 计算带宽
        bandwidth_mbps = calculate_bandwidth_shannon(snr)
        # 请求体
        payload = {
            "bandwidth": f"{bandwidth_mbps}mbit",
            "delay": "0ms",
            "local": "192.168.1.104",
            "target": "192.168.1.173"
        }
        try:
            response = requests.post(url, json=payload, headers=headers)
            count += 1
            print(f"请求 #{count} - SNR: {snr:.2f}dB, 带宽: {bandwidth_mbps}mbit")

            if response.status_code == 200:
                print(f"请求成功，状态码: {response.status_code}")
            else:
                print(f"请求失败，状态码: {response.status_code}")
                print(f"响应内容: {response.text}")
        except requests.RequestException as e:
            print(f"请求 #{count} 发送出错: {e}")
        # 等待指定的时间间隔
        time.sleep(interval)
```

图 2-15: 模拟车辆运行时传速速率变化代码

三、后期拟完成的研究工作及进度安排

3.1 后期工作

目前算法仍然存在若干局限性。首先，算法中采用的固定概率参数（如 70% 依赖任务绑定概率、80% 位置一致性概率）缺乏动态适应性，无法根据系统负载状态、网络条件和任务特性进行自适应调整。其次，现有关键路径识别机制仅考虑时间依赖关系，忽略了任务自身的计算复杂度与数据传输量差异。针对上述局限性，未来研究将重点发展自适应参数调控与多因素综合决策机制。具体而言，将根据实时网络状况、计算资源分布和任务特性动态调整，取代当前的固定概率设置。

在关键路径分析方面，将引入多维加权评估模型，综合考量任务的计算密集度、数据传输量、能耗特性以及路径关键程度，形成更为精准的任务优先级排序机制。

若时间充足，将在上述研究的基础上，进一步探索云边端分层混合式动态调度框架，通过引入任务优先级分类机制与多级资源协同分配策略，优化异构计算环境下的任务执行效率。具体而言，首先基于任务特性将任务划分为不同类型（比如实时型、带宽敏感型 和弹性计算型），并为每类任务赋予不同的优先级权重。其次，结合云边端三层的资源状态（如边缘节点算力、云端资源负载、终端设备能耗），设计动态分层调度引擎，优先将高优先级任务调度至满足其服务质量（QoS）的最优层级——例如，将关键路径上的实时任务绑定至低延迟边缘节点，而将非关键的大规模计算任务迁移至云端。

后期会继续完善仿真实验工具，模拟场景中的复杂任务链和动态网络环境。在数据采集过程中，广泛收集不同场景下的车辆运行数据、网络状态数据以及任务执行数据，确保数据的多样性和完整性。利用数据挖掘技术，对采集到的数据进行深度分析，提取关键特征和规律，为后续的算法验证和性能评估提供坚实的数据基础。在仿真实验中，严格控制实验变量，全面模拟车联网实际运行中的各种复杂情况，包括车辆的高速移动、信号的干扰、任务的突发变化等，对设计的任务卸载算法进行全方位、多角度的验证，确保算法在实际应用中的可靠性和有效性。

性能分析与优化验证阶段，对任务卸载算法在不同场景下的性能进行全面、深入的分析。不仅关注算法的执行效率和资源利用率，还将重点评估算法在保障任务实时性、稳定性和可靠性方面的表现。并与现有主流算法进行对比分析，明确算法的优势和不足之处，为进一步的优化提供方向。针对性能分析中发现的问题，深入研究优化策略，从算法结构、参数设置、资源分配等多个方面进行优化，提高算法的整体性能。

结题，论文编写阶段，梳理整个研究过程，将研究成果进行全面、准确的总结和提炼。在完成初稿后，广泛征求导师和同学意见，对论文进行修改和完善，确保论文的质量达到较高水平，能够按时、高质量地完成结题和论文编写任务。

3.2 进度安排

工作安排	周数	起止时间
任务卸载算法设计	3	2025.03.01-2025.03.22
数据采集以及仿真实验	4	2025.03.23-2025.04.21
性能分析与优化验证	3	2025.04.22-2025.05.14
结题，论文编写	1	2025.05.14-2025.05.21

四、存在的问题与困难

此优化问题归属于 NP-hard 的混合整数非线性规划 (MINLP)，求解复杂度高。随着车辆规模的扩大、时隙的细化，会导致计算复杂度的大大增加。目前的基于 GA 的启发式方法，在三辆车，每辆车三个子任务的规模下，收敛时间约为 1s。然而，实际场景可能需要在极短时间内（如 500ms）做出响应，以关键任务的顺利执行。因此，如何在如此有限的时间窗口内获取较优的可行解，成为当前算法设计工作中有待研究和解决的问题。

在车联网的实际运行环境中，车辆处于高速移动状态，这使得信道状态处于持续且快速的改变之中。而现有的系统模型在构建时，虽考虑了部分静态网络因素，但对于这种动态变化的网络环境适应性不足。这就导致在实际应用中，模型所预测的任务卸载性能与实际情况存在较大偏差，无法准确指导任务卸载决策，进而影响整个车联网系统的稳定性和可靠性。如何使系统模型能够实时、准确地适配动态网络环境的变化，是亟需解决的重要问题。

在多车多任务的复杂场景下，面临着双重耦合约束的严峻挑战。一方面，任务内部存在严格的时序依赖关系。另一方面，边缘服务器的资源有限，众多车辆的任务同时竞争这些资源。当多个任务同时请求边缘服务器的计算资源时，会出现资源争抢导致部分任务等待时间过长，进而影响任务的整体执行效率。如何在这两种耦合约束条件下，实现 DAG 任务依赖与资源竞争的协同优化，确保任务按照正确的顺序高效执行，同时合理分配有限的资源，是当前研究工作中的又一难点问题。

五、论文按时完成的可能性

综合以上已完成的核心建模工作、算法验证以及实验工具搭建等方面的成果，论文按时完成具有很大的可能性。后续研究工作将在现有基础上，针对存在的问题与困难，有针对性地开展算法优化和实验验证工作，确保能够按照预定计划完成论文的撰写和相关研究任务。