第一章　绪论

本章主要探讨了车联网中能耗优先调度算法的重要性，介绍了相关研究现状和取得的成果，提出了本文的研究任务，并阐述了主要研究内容，同时介绍了本文的结构组织。

1.1　研究背景及研究意义

未来将迎来“万物互联”的时代。随着射频识别技术、网络协议、数据存储以及传感器技术的不断发展，越来越多的物理设备与互联网连接起来，实现了物体之间的信息交互，赋予机器以“智能”。这将深刻改变人们的生产和生活方式，为人们提供更便利、智能和高效的生活和工作体验。[1]。

车联网是“万物互联”时代的典型应用，汽车的功能已经超出了传统的交通工具范畴，不再只是交通出行的工具，而变成了一个智能互联的计算系统，车辆将承担起对车辆自身的调度与管理的任务，以及满足影音娱乐等用户额外的需求。

车联网是指通过无线通信技术将车辆、路边单元与互联网连接起来的系统，实现了车辆之间、车辆与基础设施之间以及车辆与互联网之间的全面互联互通。通过车联网，用户可以获得丰富的实时信息，帮助驾驶员更好地应对路况变化。例如通过获取拥堵情况、事故警报，避开高拥挤路段，减少交通拥堵，提升驾驶安全性。通过车联网，用户可以实时监测车辆周围的情况，一方面有利于知晓车辆的问题避免产生安全问题。另一方面，当车辆与其他车辆距离过近时，车联网可以及时发出警报，并辅助驾驶员采取相应的应对措施。通过车联网，车辆还可以实现自动泊车、自动驾驶等功能，减轻驾驶员的驾驶负担，提高驾驶的便利性和舒适性。

在车联网对车辆自身进行调度与管理的任务中，有些任务对计算资源的要求较高，且对时延敏感，因此需要大量的稳定的计算资源来保证其服务质量。显而易见，车载处理器的计算资源受限于汽车电池[2]，计算资源有限，无法满足对互联的计算任务的要求。

已经提出了许多方法来解决计算资源不足问题，例如通过提高能量的利用率，减少能耗侧面的增加计算资源。在文献[3]中通过考虑完整的车辆能量管理来减少能耗，在文献[4]中通过车辆的滑行来减少能量消耗。

为了解决计算资源受限的问题，基于云的解决方案通过将任务上传到云端（远端、远程云）来进行处理（如图 1-1），云端本身就是一个计算中心，通常具有超大的规模，能提供给用户超强的计算能力，对大规模数据进行聚合、挖掘、分析、

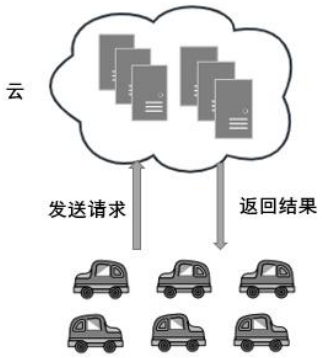

图 1-1　云计算

优化，可以在短的时间内计算得到结果。

车辆应用产生的计算任务可以通过网络传输到远程云，满足任务对运算和储存资源的需求，在云端进行计算并将结果返回给车辆。由于云端超强的计算能力，因此能够计算和分析复杂的计算任务提供高性能计算资源。

但是因为云端位于网络的远端，在物理位置上往往部署在偏远地区，将车辆计算任务传输到远程云并得到相应的返回结果需要很大的传输时间开销，在数据传输过程中会产生较高的时延。因此对于数据密集型和延迟敏感型的计算任务，即使计算延迟较小（数据量较小），较大的传输延迟仍会使任务总卸载延迟较大。同时，为了将车辆产生的大量数据全部上传到云端，这将会导致网络中的数据量大幅增加，占用大量的网络带宽，使得网络设施负担加重，从而造成网络性能和稳定性的下降，这就是云计算中链路过载的问题。

为了解决云计算的时延以及流量问题，促进数据处理能力向数据源更接近的方向发展，边缘计算（edge computing，EC）[5] [6] [7]应运而生。EC 选择了在网络边缘中（边缘端）处理数据，这里将“边缘”定义为数据源和云数据中心之间通道中的任何计算和网络资源。边缘计算。与传统的云计算相比，边缘计算将服务部署在距离用户更近的地方，放置到网络的边缘，可以实现更低的时延和更高的带宽，减少了核心网络的负载，解决了云计算中链路过载的问题。同时，这还有利于数据安全以及隐私保护[8]。

边缘计算也可以通过与物联网技术结合，实现一些与智能车联网的应用，如自动驾驶、人类行为识别、智慧城市等[9]。美国自然基金委和英特尔在 2016 年开始合作讨论建设无线边缘网络。同一年，中国也兴起了移动边缘计算的研究，在

科研院所、高等院校、企业、行业协会等各方单位的共同努力下，边缘计算产业联盟成立。

边缘计算并不是云计算的反面，边缘计算是云计算的延伸，未来将与云计算协同配合，为用户提供更高质量的服务。

为了使车联网支持云计算和边缘计算，科研组织和联盟实现了车到万物通信 (V2X) 技术[10]，V2X 主要包括车辆对车辆通信 (V2V)、车辆对基础设施通信 (V2I)。通过 V2X 技术，车辆可以与其他车辆、交通信号灯、路边停车位、行人信号、路况监测设备等进行数据交换和信息共享。

最近流行的边缘计算技术中，不仅可以将任务分配给附近的边缘服务器，还可以分配给附近资源丰富的用户[11]。文献[12]中关注车载边缘云中的虚拟机资源分配。文献[13]中考虑了动态需求和资源约束，并将所有任务分类为四种待处理列表。然后，根据它们的特性，每个列表中的任务将被卸载到不同的节点。

1.2 国内外研究历史和现状

1.2.1 任务卸载与车辆任务分类

任务卸载[14]是指将移动设备（如智能手机、车载终端等）上的计算任务分解成若干个子任务，其中一部分在本地完成，另一部分则通过网络卸载到云端或者其他设备上进行计算，最终将结果返回到本地设备，从而提高移动设备的计算性能和能耗效率的一种技术。

车联网中智能车辆的任务并不是都可以卸载的，按关键程度来分，车辆任务可以分为三类：关键任务（Crucial Tasks，CTs）、高优先级任务（High-Priority Tasks，HPTs）和低优先级任务（Low-Priority Tasks，LPTs）。CTs 是车辆安全应用，是保证车辆和乘客安全的关键应用程序，如车辆控制、系统监控和事故预防等。由于 CTs 和安全紧密相关，因此享有除操作系统的系统应用程序外最高的优先级，必须保留充足的计算资源给它，不能因为 HPTs 和 LPTs 的存在而影响 CTs 的正常运行，因此这类任务也不允许卸载，只允许在本地执行，不属于可卸载任务的范畴。该类任务的实例是：车辆控制、车辆预警、红绿灯提醒、道路情况检测（例如道路湿滑程度检测）等。

HPTs 包括与驾驶相关的应用和可选的安全增强应用，这类应用程序对车辆而言是重要但不是必须的，拥有较高的执行优先级，例如实时路径规划和路况提醒等。所以这类应用可以卸载，且这类应用允许出现延迟或卸载失败的情况，但最重要的一点是，该类任务不能影响 CTs。该类任务的实例有：地图路线规划导航、

视野增强、车辆传感等。

LPTs 是一类与影音娱乐服务相关的应用，它的优先级较低。例如语音识别，它允许驾驶员发出各种声音命令，通过语音识别命令计算机做一些响应，而不会使驾驶员分心。该类任务的实例是：虚拟现实、语音识别、影视音乐、在线游戏等。

HPTs 和 IPTs 已经被部署到越来越多的车辆上，由于 HPTs 和 LPTs 不会涉及到安全，因此可以将这两类任务进行卸载，来提高车辆资源的利用率。

1.2.2 任务卸载的最优化目标

（1）最小化卸载时延。车辆所产生的任务可以选择在本地执行或者卸载到边缘服务器上进行执行，前者会产生本地时延和卸载时延，本地时延是指数据在本地运行所消耗的全部时间：卸载时延是指在边缘计算环境中，数据从生成到处理的时间间隔（一般忽略回传时延）。文献[15]最优化目标为处理时延。文献[16] [17]设计了一种基于多臂老虎机理论的自适应学习任务卸载（ALTO）算法，以最小化平均卸载延迟。

（2）最小能量消耗。通常而言，实际的能耗也是必不可缺的指标。对于车辆而言，可以提高车辆的行驶里程，环节“里程焦虑”。对于边缘服务器而言，可以最大限度地提高网络运营商和服务提供商的收入。[18] [19]的目标为最小化能量消耗。

1.2.3 车联网任务卸载研究现状

车联网中任务卸载的常用算法包括基于强化学习的车联网任务调度、基于博弈论的车联网任务调度和基于启发式算法的车联网任务调度。

（1）基于强化学习的车辆边缘计算任务调度算法研究。

随着深度强化学习（DRL）算法理论的不断完善，DRL 已广泛应用于车联网中智能控制与资源管理与优化等方面[20]。DRL 算法可以感知环境的变化并做出决策，能满足车联网场景的需求，被认为是解决车辆边缘计算的有效方案。

DRL 算法主要包括智能体、环境、动作和奖励等部分。一般而言，模型通常把执行决策的主体定义为智能体，并把能够影响智能体进行决策的因素称为环境。智能体与环境进行交互，不断地学习积累经验，直至解决问题。

目前已有众多学者运用 DRL 方法解决车联网中任务卸载问题。如文献[21]提出了一种新的层次框架来解决云计算系统中的整体资源分配和电源管理问题，分层包括用于将虚拟机资源分配给服务器的全局层和用于本地服务器的分布式电源管理的本地层，以应对算法的高维问题，实现了服务器集群中实现延迟和能耗之间的最佳权衡。

文献[22]提出了一种基于深度强化学习的智能云资源管理架构，使云能够直接从复杂的云环境中自动高效地协商最合适的配置。文献[23]提出了一种基于 DRL 的、为云服务商提供资源配置和任务调度的方案，用于最大限度地降低具有大量服务器的云服务提供商的能源成本。文献[24]将问题分为两个子优化问题。分别发展了双侧匹配方案和深度强化学习方法，分别用于调度卸载请求和分配网络资源。绩效评估说明了我们构建的系统的有效性和优越性。文献[25]将任务卸载问题被分解为两部分流重定向和卸载决策并提出了一种基于深度强化学习的方案来解决这个优化问题。

（2）基于博弈论的车辆边缘计算任务调度算法研究。

博弈论是研究决策制定者在冲突或合作情境中进行策略选择的数学理论。当使用博弈论方法对车联网中的任务卸载进行研究时，首先要把目标最优问题转换为博弈论的基础问题，然后对其进行数学建模。建模的目的是利用博弈论来确定均衡状态。均衡状态是指所有的参与者都得到最佳收益的状态，当一个系统的博弈达到均衡时，每个参与者都无法仅通过改变自己的策略而在系统中得到更好的结果。博弈的本质就是多个相互影响的参与者在选择各自的决策时最大化自己的收益。

博弈论的方法主要包括以下几种：

（1）纳什均衡方法。纳什均衡是博弈论最重要的概念之一。它指的是在一个博弈中，每个玩家都采取了最优的策略，而没有人想改变自己的策略，因为改变策略会导致收益降低。这种情况下，所有玩家都处于理性选择的状态，即达到了稳定状态。

（2）最小化最大损失方法。也被称为“最小保证赢法”。这种方法的基本思想是：在不确定对手的行动时，尽量减少自己可能遭受的最大损失。

（3）博弈树方法。博弈树是博弈论中常用的一种分析方法。它是一种图形，用于显示博弈的各种可能情况和策略选择，以及每种情况下玩家的收益或损失。通过分析博弈树，可以找到纳什均衡点和最优策略。

文献[26]将任务卸载子问题建模为精确势博弈，并提出了一种多智能体分布式深度确定性策略梯度来实现纳什均衡。将资源分配子问题划分为两个独立的凸优化问题，并利用基于梯度的迭代方法和 KKT 条件提出了最优解。

文献[27]将问题表述为潜在的博弈并证明它存在纳什均衡，设计了一种去中心化算法，用于在博弈中找到纳什均衡。

文献[28]将多用户卸载决策问题建模为非合作博弈，证明该问题存在纳什均衡，

表 1-1　启发式算法的主要思想

算法名称	主要思想
粒子群算法	鸟群或鱼群等生物群体的行为
模拟退火算法	固体材料退火过程中的原子结构调整
遗传算法	生物遗传和进化理论
蚁群算法	蚂蚁搜寻食物时的行为

并提出了一种基于机器学习技术的全分布式计算卸载算法来寻找该纳什均衡点。

（3）基于启发式算法的车辆边缘计算任务调度算法研究

启发式算法是一种解决复杂问题的计算方法，它通过利用经验或启示来指导搜索过程，以找到可能的解决方案。与精确算法不同，启发式算法可能无法保证找到最优解，但通常能够在合理的时间内找到接近最优解的解决方案[29]。启发式算法是基于直观自然现象或者结构自然背后的经验构造的算法（如表1-1），在可接受的时空开销内给出问题的一个可行解（次优解）。贪婪算法是一种直观的启发式算法，它每一步选择当前最好的选择（局部最优），而不考虑长远影响（整体最优）。贪婪算法通常易于实现和理解，但可能会得到次优解。模拟退火算法受到固体退火过程的启发，通过接受一定概率的劣解，以避免陷入局部最优解，并逐渐趋向全局最优解。模拟退火算法适用于求解复杂问题，但需要合适的参数设置和调整[30]。

文献[31]通过考虑通用型任务计算时间分布来分析排队延迟，通过考虑给定概率满足延迟约束，以最小化受延迟和计算资源约束的 ITS 算子费用提出了一个非线性优化问题。针对该问题，该文提出一种线性化方法，并设计了一种基于整数线性规划的算法同时引入了一种称为任务卸载的成本效益启发式算法的启发式算法来解决该问题。

文献[32]研究了一个由多用户、一个边缘节点以及一个云服务器组成的移动云计算系统，提出了一个优化目标为用户之间的能耗、计算和最大延迟的总体成本的非凸二次约束二次规划问题，并提出了一种高效的启发式算法来解决这个问题。

Huynh 等人[33]开发了一种基于粒子群的启发式算法，用来解决异构网络中多用户多服务器的两层计算卸载策略问题，该问题的优化目标为总计算开销，包括完成时间和能耗。

文献[34]提出了一种用于共享交通流信息的联网车辆之间的通信框架，将联网车辆视为人工蚂蚁，能够根据交通流量的动态进行自我计算以做出自适应决策，使

车辆能计算出到达目的地的最佳路径。

1.2.4　车辆边缘计算面临的挑战

目前，基于车联网的车辆边缘计算面临着诸多挑战。

（1）实时性问题，车辆位置、速度、路况等方面的影响会导致车辆任务的实时变化，进而导致车辆任务卸载环境的不稳定性，上一时刻的最优分配方案可能下一时刻就是最劣方案，导致任务卸载方案的可靠性降低。

（2）资源动态变化问题，车辆位置、速度等方面的影响，车辆的计算等资源会发生相应改变。目前，大多数车联网任务卸载的研究都假设车联网中计算资源在较长时间内保持相对稳定。而在现实情况下，各种因素会导致资源状态发生明显的变化，导致车联网任务卸载算法的适应性降低。

（3）卸载环境复杂问题，在车联网中，由于任务关键程度的不同，车联网任务卸载环境变得更加复杂。不是所有的任务都是可卸载的，对于那些不适合在边缘设备上进行处理的任务，应当优先本地来完成。这样可以保证车辆的行车安全。

根据以上的分析，车联网中任务卸载面临的问题对求解精确解的传统优化算法提出了挑战，传统优化算法只能求解特定的问题，以及高复杂度的特点，使得难以解决车辆边缘计算任务卸载。通过将任务卸载给附近的车辆是一种可能的解决方案，但是又带来了求解的方案难以兼顾公平的问题。

因此，在本文中，车联网存在不可卸载的任务情况下，以提高能源利用效率为目标，通过将最优化目标设置为能耗的平方来来兼顾公平性，通过启发式算法来生成任务卸载方案。

1.3　论文的研究内容和组织结构

本文主要研究了车联网任务卸载算法，在车辆任务不能完全卸载情况下，即有部分任务必须车辆在本地执行的情况下，如何利用能耗与 CPU 频率之间的非线性关系进行任务卸载用来提高整体的能量利用率，使得整体能耗下降。

论文共分为五章，论文的具体内容安排如下：第 1 章对全文内容进行概述和章节安排，首先介绍了车联网和移动边缘计算研究背景和意义，并简述了边缘计算中任务卸载的国内外研究现状，对全文内容进行概述，最后给出论文组织章节安排。

第 2 章介绍了相关的知识背景，如数学规划的定义以及与整数规划形式和 0-1 线性规划形式两种不同的数学规划形式，其次综述了三种车联网任务卸载现有的相关算法研究。

第 3 章首先将问题进行建模为数学规划的形式，该问题分为任务可分情况和任务不可分情况，因为该问题是一个非线性整数规划问题，没有多项式时间范围内的解决方法，因此提出使用遗传算法解决该问题，并通过仿真实验验证了算法的性能实验。

第 4 章首先说明了贪心法调整的策略，然后重新定义了运算符，重新定义的运算符有双目加法运算符、双目减法运算符、双目乘法运算符、单目调整运算符。通过重新定义运算符的方式，将侏儒猫鼬算法修改为解决离散问题的算法，解决本文提出的离散形式的问题。

第 5 章全文总结与展望。总结全文的研究内容和主要贡献，以及需要后续深入研究的问题。

第二章　任务卸载技术研究

本章给介绍了相关的知识背景，如数学规划的定义以及与本文相关的数学规划问题的不同描述形式（整数规划形式和 0-1 线性规划形式），然后介绍了本文中使用的两种算法，基因算法和侏儒猫鼬算法，最后进行了章节小结。

2.1　相关知识背景

本节介绍了想要将算法修改为离散算法的相关知识。如数学规划的定义、数学规划问题的不同描述形式（整数规划和 0-1 线性规划）。

2.1.1　数学规划

数学规划是一种数学建模技术，旨在找到最优解决方案以满足特定的约束条件。它在现代科学、工程和经济领域中具有广泛的应用。常见的数学规划包括线性规划、整数规划、非线性规划和混合整数规划等类型。

线性规划（Linear Programming，LP）：线性规划是数学规划中最常见的类型。线性规划的目标函数和约束条件都是线性的，这使得它具有许多良好的数学特性，例如可解析性、单调性和凸性。线性规划可以用单纯形算法、内点算法[35]等方法来求解。下面是一个线性规划的实例：

$$\begin{aligned} \max \quad & z = 40x_1 + 45x_2 + 24x_3 \\ \text{s.t.} \quad & 2x_1 + 3x_2 + x_3 \leq 100 \\ & 3x_1 + 3x_2 + 2x_3 \leq 120 \\ & x_1 \geq 0, x_2 \geq 0, x_3 \geq 0 \end{aligned}$$

整数规划（Integer Programming，IP）：整数规划是线性规划的扩展，其决策变量被限制为整数。这种类型的规划适用于需要做出离散决策的问题，比如资源分配、项目排程等。常用的方法有分支定界法[36]、割平面法[37] [38]等。下面是一个整形规划的实例：

$$\begin{aligned} \max \quad & f = 5x_1 + 8x_2 \\ \text{s.t.} \quad & x_1 + x_2 \leqslant 6 \\ & 5x_1 + 9x_2 \leqslant 45 \\ & x_1, x_2 \in \mathbb{Z} \end{aligned}$$

非线性规划（Nonlinear Programming，NLP）：非线性规划是指目标函数或约束条件中存在非线性项的优化问题。非线性规划可以包括凸优化、非凸优化等问题，通常需要使用不同的算法进行求解。常用的方法包括梯度下降法、牛顿法、拟牛顿法等。

混合整数规划（Mixed Integer Programming，MIP）：混合整数规划是整数规划的进一步扩展，允许部分决策变量为连续变量，部分为整数变量。混合整数规划在实际问题中很常见，例如生产调度、设施选址等。求解方法包括分支定界法、启发式算法等。

数学规划是一种强大的工具，可以帮助我们在复杂的环境下寻找最优解决方案。通过使用数学规划，我们可以优化资源分配、提高效率、降低成本，从为我们的生活和经济发展做出更大的贡献。

2.1.2 数学规划问题描述形式

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

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

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

最小最大完工时间的混合整数规划公式如下:

$$\min \max_{1 \le j \le m} \sum_{i=1}^{n} p_i x_{ij} \tag{2.1}$$

图 2-1　解的表示

$$\text{s.t.} \sum_{j=1}^{m} x_{ij} = 1 \quad i = 1, \ldots, n \tag{2.2}$$

$$x_{ij} \in \{0, 1\} \tag{2.3}$$

公式2.1表明，本论文应该耗时最久的那台机器尽可能地小，因为所有的作业完成才算是完成了任务。公式2.2表明每个作业必须分配到一台机器上。如果 $x_{ij} = 1$，意味着作业 i 被分配到 j 的机器上。

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

$$\min \max_{1 \le j \le m} \sum_{1 \le i \le m, x_i = j} p_i \tag{2.4}$$

$$\text{s.t.} \quad x_i \in \{0, 1, \ldots, m\}, \quad i = 1, \ldots, n \tag{2.5}$$

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

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

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

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

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

2.2 基因算法

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

基因算法是一种典型的启发式优化算法。算法鲁棒性来源主要有以下三点：（1）个体多样性，通过引进交叉、变异等操作，使得不同个体之间进行信息的交流，有助于保持种群的多样性，从而不易陷入问题的局部最优解。

（2）选择策略。基因算法中的选择策略来自于达尔文自然选择学说中的“适者生存”原则，通过对适应度高的个体进行选择和繁殖，逐步提高整个种群的适应性，这提高了算法的鲁棒性。

（3）最后来源于参数设置。基因算法中的参数设置，例如交叉的概率、变异的概率，这对算法的表现具有较大影响。通过对参数的合理设置，可以提高算法的鲁棒性，并且使其更容易达到全局最优解。

基因算法已被应用于各个领域。在文献[39] 中，Sun 等人使用基因算法设计 CNN 架构。张等人[40] 使用基因算法来最小化能量消耗。Yoon 等人[41] 提出了一种高效的基因算法，使用新颖的归一化方法来解决无线传感器网络中的最大覆盖部署问题。在复杂的三维环境中，Roberge 等人[42] 使用它计算固定翼无人机的可行和准最优轨迹。

以下是基因算法的实现步骤：

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

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

基因算法二进制解码公式如下：

$$\delta = \frac{U_x - L_x}{2^l - 1} \tag{2.6}$$

$$x = L_x + \delta \cdot \sum_{i=1}^{l} A_i 2^{i-1} \tag{2.7}$$

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

基因算法二进制编码二进制长度公式如下：

$$l = \left\lceil \log_2 \left(\frac{U_x - L_x}{\text{SearchAccuracy}} \right) \right\rceil \tag{2.8}$$

其中，U_x 和 L x 分别是解的最大值和最小值，SearchrAccuracy 是求解的精度。例如，给定 $U_x = 5$ 和 $|\mathscr{L}_x = -5$，精度是 0.01，那么 $l = \left\lceil \log_2 \left(\frac{5-(-5)}{0.01} \right) \right\rceil = 10$，也就是说，需要十个二进制位来表示解。

适应度计算公式如下：

$$fit_i = e^{-\frac{y_i}{mean(y)}}. \tag{2.9}$$

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

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

$$p_i = \frac{\hat{f}it_i}{\sum_{i=1}^{n} \hat{f}it_i} \tag{2.10}$$

其中，n 是种群中的个体数量。

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

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

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

2.3　侏儒猫鼬算法

2.3.1　算法简介

和粒子群、基因算法类似，侏儒猫鼬算法（Dwarf Mongoose Optimization Algorithm，DMOA）也是一种基于动物行为的启发式优化算法，能够用来解决线性以及非线性的优化问题。该算法是由 Jeffrey O. Agushaka[43] 等科学家于 2022 年提出的，算法的灵感源于侏儒猫鼬的捕猎行为，通过模拟侏儒猫鼬在觅食和捕猎时的行为特征[44] 来进行对问题的求解。

图 2-2　侏儒猫鼬

DMOA 在理论上比文献中的一些算法更能找到不同优化问题的全局最优解，因为它具有以下独特的属性: DMOA 特殊的探索和开发能力，这模仿了侏儒猫鼬的半游牧行为和补偿适应。DMOA 只有一个可调参数，可以提供更高的效率和可靠性。简化调优过程：开发人员只需关注一个参数的调整，而无需考虑其他参数的影响。提高效率：更容易地找到最佳参数配置。提高可靠性：减少了参数之间的相互影响，使算法更加稳定和可靠。当参数之间存在复杂的交互作用时，调优过程可能会变得复杂且困难，同时也增加了算法出现错误或不稳定性的风险。

侏儒猫鼬算法被应用于机器学习，建模等方面。文献[45]提出了一种改进的侏儒猫鼬优化器的智能元启发式形式，用于太阳能光伏系统的最佳建模和参数提取。文献[46]提出了一种二元侏儒猫鼬优化算法，以解决机器学习模型在数据挖掘种高维特征选择问题。文献[47]使用经过侏儒猫鼬优化算法（DMOA）调整的 SqueezeNet 轻量级的卷积神经网络来预测心脏病。文献[48]提出了一种基于矮猫鼬优化的多跳路由方案（DMOSC-MHRS）的安全集群来生成到达目的地的最佳路线。

2.3.2　侏儒猫鼬

侏儒猫鼬（英文名：Dwarf mongoose）是一种小型的食肉动物，属于猫鼬科。它们主要分布在非洲撒哈拉以南地区的草原、稀树草原和山地森林中。

侏儒猫鼬的体型相对较小（如图 2-2），平均身长约为 25-30 厘米，尾部长约为 15-20 厘米。其身体呈灰褐色，背部有黑色纵纹，腹部呈浅黄色。它们的体重通常在 0.2kg 到 0.35kg 之间。侏儒猫鼬是杂食性动物，他们的除了捕食昆虫和小型动

物外，它们还会食用水果、种子和其他植物物质。它们的食性多样性让它们能够适应不同的环境条件。

侏儒猫鼬通常生活在母系社会的家庭群体中，由一对终身结合的阿尔法配偶领导。在家庭群体中，雌性猫鼬的地位高于雄性[49]。根据年龄和性别，不同的猫鼬扮演着守卫、保姆等不同的角色[50]。

2.3.3　侏儒猫鼬算法种群分组

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

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

$$X_i^{t+1} = X_i^t + \text{phi} \times \text{peep} \times (\text{X}_i^t\text{-X}_{peep}) \tag{2.11}$$

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

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

$$\alpha = \frac{fit_i}{\sum_{i=1}^{n} fit_i} \tag{2.12}$$

其中, fit_i 是第 i 个个体的适应度,n 是种群数量。

适应度的计算方式为：

$$fit = e^{-\frac{y}{y}} \tag{2.13}$$

其中，y 为目标函数的值。

计算适应度使用公式2.13的好处有：首先，考虑了每个个体的表现与平均表现的比值，因此能够根据整体表现水平动态地调整适应度值。这意味着在不同阶段，个体的适应度能够更好地反映其在当前群体中的相对表现。其次，数函数的性质能够有效地抑制过大的适应度值，使得在优化过程中，个别极端值对整体优化的影响被降低，从而增强了算法的鲁棒性。最后，体现相对优势：通过将个体的表现与整体平均水平进行比较，适应度算法能够更好地体现个体的相对优势，从而促进种群中优秀个体的选择和保存。

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

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

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

睡眠丘的计算公式如下：

$$sm_i = \frac{fit_{i+1} - fit_i}{\max\left\{\left|fit_{i+1}, fit_i\right|\right\}} \tag{2.14}$$

睡眠丘的平均值反映了该轮迭代的移动范围，φ 计算公式如下：

$$\varphi = \frac{\sum_{i=1}^{n} sm_i}{n} \tag{2.15}$$

CF 是表示侦察兵小组移动能力的参数，其计算公式如下，$iter$ 为当前迭代次数，T 为最大迭代次数。

$$CF = \left(1 - \frac{iter}{T}\right)^{\left(2\frac{iter}{T}\right)} \tag{2.16}$$

在图 2-3中，$x = \frac{iter}{T}$，$x \in [0,1]$，当迭代次数增加时，其值会减小，也就是说，随着迭代次数增加，侦察兵小组移动能力会减小。

侦察兵小组的更新方式如下：

$$X_{i+1} = \begin{cases} X_i - CF * \text{phi} * \text{rand} * \left[X_i - \overline{\mathbf{M}}\right] & \text{if } \varphi_{i+1} > \varphi_i \\ X_i + CF * \text{phi} * \text{rand}^* \left[X_i - \overline{\mathbf{M}}\right] & \text{else} \end{cases} \tag{2.17}$$

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

2.3.4 算法流程

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

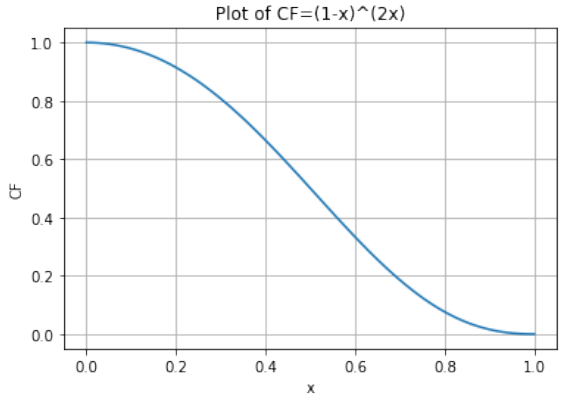

图 2-3　CF 变化图

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

侦察兵会发现一个新的觅食地点，并通知雌性首领，带领家族到达新的地点。

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

显而易见，侏儒猫鼬算法在进行每一步的更新时，使用了类似于梯度（如公式2.18）的更新方式，因此该算法只能解决连续形式的问题，不能解决离散形式的问题，需要本论文将其改造为像基因算法一样的离散型算法。

$$X_i^{t+1} = X_i^t + \text{phi} \times \text{peep} \times (X_i^t - X_{peep}) \tag{2.18}$$

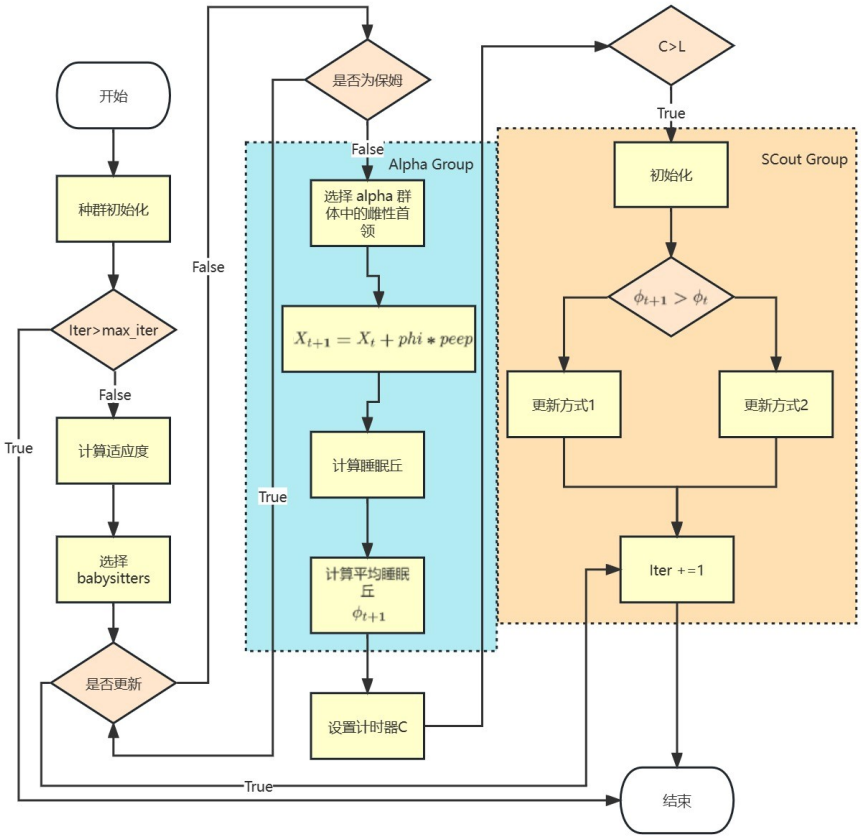

图 2-4　侏儒猫鼬算法流程图

2.4　本章小结

本章详细介绍了将算法修改为离散形式的相关知识，包括数学规划和数学规划描述形式本章所介绍的内容将为后续离散算法的提出提供借鉴的基础。

第三章　基于基因算法的车辆任务卸载算法

本章首先将问题进行建模为数学规划的形式，该问题分为任务可分情况和任务不可分情况，因为该问题是一个非线性整数规划问题，没有多项式时间范围内的解决方法，因此提出使用遗传算法解决该问题，并通过仿真实验验证了算法的性能实验。实验结果表明，算法可以达到节能的目的的同时很好的兼顾公平性。

3.1　系统模型

本节主要介绍了系统模型，分为三个部分，系统场景，车辆计算能力模型，以及车辆能耗模型。

3.1.1　系统场景

在本文中，如图 3-1所示，层次结构由车辆云、边缘云和中心云组成。中心云负责全局调度，它执行的任务包括执行复杂的计算任务和进行全局的决策。边缘云是车辆云的控制器，负责创建、维护和删除车辆云。车辆云由一定范围内的、参与共享其计算资源的智能车辆组成。车辆可以通过将任务卸载到其他车辆的方式来节省能源。这样可以提高整体资源利用率。

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

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

3.1.2　车辆计算能力模型

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

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

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

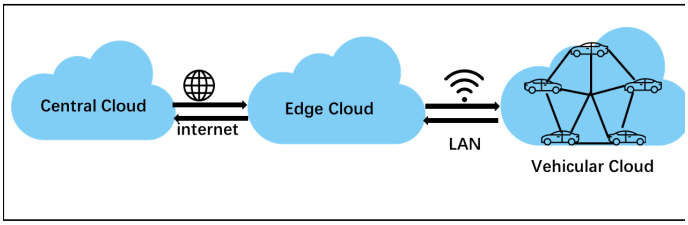

图 3-1　系统场景图

行。

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

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

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

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

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

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

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

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

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

其中，υ_i 和 θ_i 是待估计的参数。

最后，公式3.2中的 CPU 的容量 C_{it} 在能被下面的公式计算出来：

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

3.1.3　车辆能耗模型

当本论文考虑车辆的能源消耗时，本论文将它分为两个部分（1）计算所需要的能量以及（2）传输所需要的能量。

根据[51] [52] [53] [54]，计算所需要的能耗能够被下列公式计算：

$$E = \lambda_i \cdot f_{it}^3 \cdot \Delta T \tag{3.5}$$

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

$$E_i^{save} = (\lambda_i \cdot f_{it}^3 - \lambda_i \cdot f'^3_i) \times \Delta T, \forall i \in R \tag{3.6}$$

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

$$E = P_0 \cdot \frac{d_{it}}{b_{ij}} \tag{3.7}$$

其中，P_0 是传输率。尽管功率可以变化[55] [56] [57]，但为了简单起见，本论文在本文中使用固定值。

最大的传输速率（b_{ij}）可以通过香农公式计算：

$$b_{ij} = W \log(1 + \text{SNR}) \tag{3.8}$$

其中，SNR 是信噪比，W 是频道的带宽。因为它和每一个场景相关，本论文将它考虑为一个常数。

对于每一辆车，接收信息所消耗的能量是：

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

对于每一辆车，发送信息所需要的能量是：

$$E_i^{send} = \sum_{i=1, i\neq j}^{N} x_{ij}^t \cdot P_0 \cdot \frac{d_{it}}{W \log(1 + \text{SNR})}, \quad j = 1, \ldots, N \tag{3.10}$$

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

$$\begin{aligned} E_{it}^{blnc} = & \sum_{j=1,j\neq i}^{N} x_{ij}^{t} \cdot P_0 \cdot \frac{d_{jt}}{W \log(1+\mathrm{SNR})} \\ & + \sum_{j=1,,j\neq i}^{N} x_{ji}^{t} \cdot P_0 \cdot \frac{d_{it}}{W \log(1+\mathrm{SNR})} \\ & + \lambda_i \cdot f_{it}^{3} \cdot \Delta T, \quad i = 1, \ldots, N \end{aligned} \tag{3.11}$$

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

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

$$\min \sum_{t=1}^{T} \sum_{i=1}^{N} (E_{it}^{blnc})^2 \tag{3.12}$$

3.1.4 分配矩阵例子说明

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

X 是一个分配矩阵的例子。车辆集合为 {A，B，C，D，E}，车辆 A、B 和 D 没有分配给其他车辆任务，因此它们不会产生传输能量消耗。而车辆 C 执行车辆 E 的卸载任务。每列的总和大于或等于 1。

表 3-1　本文符号说明

参数	解释
T	资源共享时间
C_{it}	CPU 容量
r_i	车辆 i 的工作负载
r'_{jt}	资源请求者 j 需要的资源
M_{it}	每秒百万次指令数（MIPS）
θ_i	估计参数
E_{it}^{blnc}	车辆 i 消耗的能量
E_i^{rec}，E_i^{send}	车辆 i 接收或发送的能量

3.2　问题建模

在本节中，本论文将问题分为两种情况来建模，一种是任务可分情况（3.3.1 节），另一种是任务不可分情况（3.3.2 节）。当任务可分的情况下，问题能够直接求解数学规划的方式解决；当任务不可分时，不能通过求解数学规划的方式来解决。

3.2.1　任务可分情况

当车辆在进行某些同质任务时，可以将大任务任意的拆成一些小任务，因为任务的粒度太小，因此可以近似看成任何一个任务可以随意拆分然后分配到任意的车辆上，也就是任务可分的情况。

在任务可分时，根据本地留存的高优先级任务的大小又分为：高于平均任务（大任务），和低于平均任务大小（小任务）这两种情况。

（1）小任务情况

当本地任务低于平均任务大小（小任务）时，该问题较为简单，直接将所有任务平均分配到车辆上就能解决。可以看出，该问题等价于：首先将固定值 C 切制成 n 个数字，使得的他们的三次方之和最小。设切分的数分别为 $a_1, a_2, \cdots, a_n$，根据均值不等式：$\sqrt[r]{\frac{\sum_{i=1}^{n} a_i^r}{n}} \geq \frac{\sum_{i=1}^{n} a_i}{n}$，可得：$\frac{\sum_{i=1}^{n} a_i^3}{n} \geq \left(\frac{\sum_{i=1}^{n} a_i}{n}\right)^3$，其中当 $a_1 = a_2 = ... = a_n = \frac{C}{n}$ 时，等号成立。

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

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

（2）大任务情况

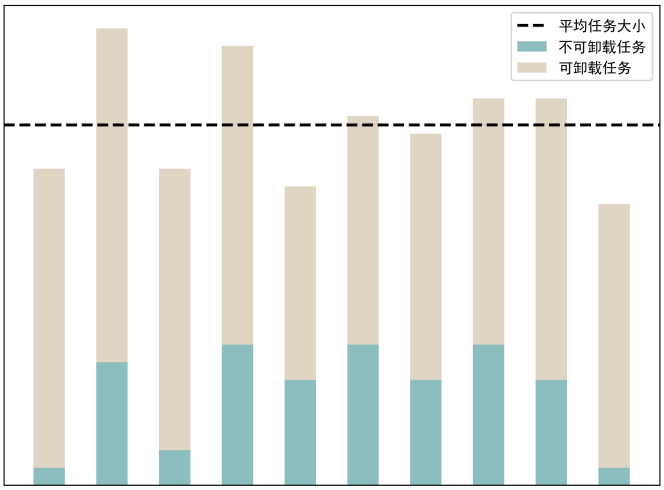

图 3-2　小任务情况

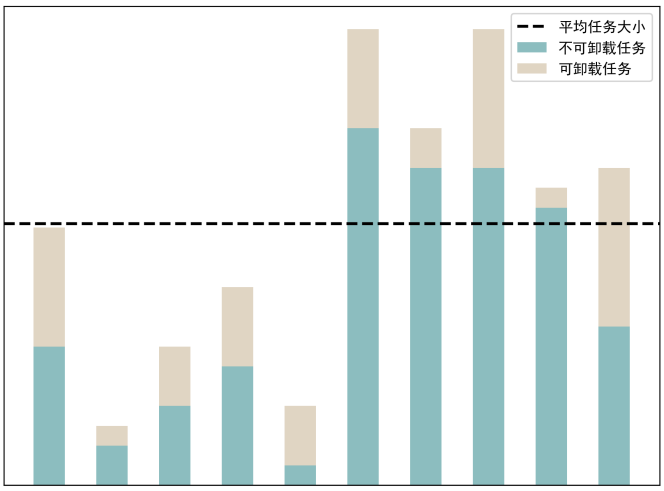

图 3-3　大任务情况

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

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

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

有以下约束条件：

$$\begin{cases} \sum_{i=1}^{n} a_i = C \\ x_i - a_i \leq 0, i = 1,,,m \end{cases} \tag{3.13}$$

目标函数为：

$$\min \sum_{i=1}^{n} a_i^3 \tag{3.14}$$

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

$$\begin{aligned} &L(a_1, a_2, ..., a_n, \lambda_1, \lambda_2, ..., \lambda_m, \mu) \\ &= \sum_{i=1}^{n} a_i^3 + \sum_{i=1}^{m} \lambda_i \left(x_i - a_i\right) + \mu \left(\sum_{i=1}^{n} a_i - C \right) \end{aligned} \tag{3.15}$$

其中，λ_i ,μ 是拉格朗日乘子。

如公式3.16所示，分别对 L 关于 a_i、λ_i 、μ 求偏导，并令其等于零。

$$\begin{cases} \frac{\partial L}{\partial a_i} = 0 \\ \frac{\partial L}{\partial \lambda_i} = 0, i = 1, ...m \\ \frac{\partial L}{\partial \mu} = 0 \end{cases} \tag{3.16}$$

得到以下结果：

$$\begin{cases} \frac{\partial L}{\partial a_i} = 3a_i^2 - \lambda_i + \mu = 0, \quad i = 1, \dots m \\ \frac{\partial L}{\partial a_i} = 3a_i^2 + \mu = 0, \qquad i = m+1, \dots n \\ \frac{\partial L}{\partial \lambda_i} = a_i - x_i = 0, \qquad i = 1, \dots m \\ \frac{\partial L}{\partial \mu} = \sum_{i=1}^{n} a_i - C = 0 \end{cases} \tag{3.17}$$

根据 $\frac{\partial L}{\partial \lambda_i} = 0$ 可得对于所有必须大于平均值的 a_i，$a_i = x_i$。根据 $\frac{\partial L}{\partial a_i} = 0$，可得 $a_i = \sqrt{\frac{-\mu}{3}}$，即对于没有限制的 a_i，其值都相等。

可得最优的分配方案是：（1）对于本地任务是大任务的车辆，将他们所有可分得任务分出去。（2）其余车辆再均分剩余的任务。

3.2.2　任务不可分情况

在任务不可分时，该问题没有多项式时间内的解决方法，因此，在下一节中，本文打算通过智能算法来获得一个近似解，来满足靠近最优解和卸载方案计算时

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

$$\min \quad \sum_{t=1}^{T}\sum_{i=1}^{N}(E_{it}^{blnc})^2 \tag{3.18}$$

$$\begin{aligned}\text{s. t.} \quad E_{it}^{blnc} = &\sum_{j=1,j\neq i}^{N} x_{ij}^{t}\cdot P_0\cdot\frac{d_{jt}}{W\log(1+\mathrm{SNR})} \\ &+\sum_{j=1,j\neq i}^{N} x_{ji}^{t}\cdot P_0\cdot\frac{d_{it}}{W\log(1+\mathrm{SNR})} \\ &+\lambda_i\cdot f_{it}^{3}\cdot\Delta T, \quad i=1,\ldots,N\end{aligned} \tag{3.19}$$

$$\sum_{j=1}^{N} x_{ij}^{t}\cdot r'_{jt}\leq C_{it} \tag{3.20}$$

$$\sum_{i=1}^{N} x_{ij}^{t}\geq 1 \tag{3.21}$$

$$f_{it}\geq 0 \tag{3.22}$$

$$x_{it}^{t}\in\{0,1\} \tag{3.23}$$

正如前文所提到的，公式 (3.20) 说明了对于每一辆车辆 i，需求的总和不能超过他的容量。公式 (3.21) 说明了车辆 i 的任务，必须要被执行，无论是本地执行还是分配给其他车辆。

3.3 仿真实验

3.3.1 实验环境与仿真参数设计

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

本文参数设置如下，车辆 CPU 使用 Cortex-A57，Cortex-A57 处理器是 ARM 性能最高的处理器，旨在推动移动和企业计算应用。Cortex-A57 的频率从 700 MHz 到 1900 MHz。频率集为 700, 800, 900, , , 1900，因此，只有 13 个频率级别。当车辆在 700MHz 处理不了任务时，频率升为 800MHz 来处理任务。

在表3-2中，$U[x,y]$ 是 x 和 y 之间的均匀分布。$N(x,y)$ 是正态分布，均值为 x，方差为 y。