软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(3):1466−1484 [doi: 10.13328/j.cnki.jos.006805]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

移动边缘计算不确定性任务持续卸载及资源分配方法*

许 斌[1], 赵云凯[2], 朱剑鸣[1], 刘一川[1], 李烜焘[1], 孙雁飞[1], 季一木[2]

[1](南京邮电大学 物联网学院, 江苏 南京 210003)

[2](南京邮电大学 计算机学院, 江苏 南京 210003)

通信作者: 许斌, E-mail: xubin2013@njupt.edu.cn

摘 要: 移动边缘计算场景中任务的不确定性增加了任务卸载及资源分配的复杂性和难度. 鉴于此, 提出一种移动边缘计算不确定性任务持续卸载及资源分配方法. 首先, 构建一种移动边缘计算不确定性任务持续卸载模型, 通过基于持续时间片划分的任务多批次处理技术应对任务的不确定性, 并设计多设备计算资源协同机制提升对计算密集型任务的承载能力. 其次, 提出一种基于负载均衡的自适应策略选择算法, 避免计算资源过度分配导致信道拥堵进而产生额外能耗. 最后, 基于泊松分布实现了对不确定任务场景模型的仿真, 大量实验结果表明时间片长度减小能够降低系统总能耗. 此外, 所提算法能够更有效地实现任务卸载及资源分配, 相较于对比算法, 最大可降低能耗 11.8%.

关键词: 移动边缘计算; 不确定性任务; 任务卸载; 负载均衡; 自适应

中图法分类号: TP393

中文引用格式: 许斌, 赵云凯, 朱剑鸣, 刘一川, 李烜焘, 孙雁飞, 季一木. 移动边缘计算不确定性任务持续卸载及资源分配方法. 软件学报, 2024, 35(3): 1466–1484. http://www.jos.org.cn/1000-9825/6805.htm

英文引用格式: Xu B, Zhao YK, Zhu JM, Liu YC, Li XT, Sun YF, Ji YM. Continuous Offloading and Resource Allocation Method of Uncertain Tasks in Mobile Edge Computing. Ruan Jian Xue Bao/Journal of Software, 2024, 35(3): 1466–1484 (in Chinese). http://www.jos.org.cn/1000-9825/6805.htm

Continuous Offloading and Resource Allocation Method of Uncertain Tasks in Mobile Edge Computing

XU Bin[1], ZHAO Yun-Kai[2], ZHU Jian-Ming[1], LIU Yi-Chuan[1], LI Xuan-Tao[1], SUN Yan-Fei[1], JI Yi-Mu[2]

[1](School of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

[2](School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: The uncertainty of tasks in mobile edge computing scenarios makes task offloading and resource allocation more complex and difficult. Therefore, a continuous offloading and resource allocation method of uncertain tasks in mobile edge computing is proposed. Firstly, a continuous offloading model of uncertain tasks in mobile edge computing is built, and the multi-batch processing technology based on duration slice partition is employed to address task uncertainty. A multi-device computing resource coordination mechanism is designed to improve the carrying capacity of computation-intensive tasks. Secondly, an adaptive strategy selection algorithm based on load balancing is put forward to avoid channel congestion and additional energy consumption caused by the over-allocation of computing resources. Finally, the uncertain task scenario model is simulated based on Poisson distribution, and experimental results show that the reduction of time slice length can reduce the total energy consumption of the system. In addition, the proposed algorithm can achieve task offloading and resource allocation more effectively and can reduce energy consumption by up to 11.8% compared with comparison algorithms.

Key words: mobile edge computing (MEC); uncertain task; task offloading; load balancing; adaptive

---

* 基金项目: 国家自然科学基金 (62172235); 江苏省研究生实践创新计划 (SJCX21_0301); 中国博士后科学基金 (2020M671552)
收稿时间: 2022-03-31; 修改时间: 2022-06-14, 2022-08-11; 采用时间: 2022-09-17; jos 在线出版时间: 2023-04-26
CNKI 网络首发时间: 2023-04-27

随着互联网的普及, 智能移动设备已广泛应用于人们的生活, 同时催生了大量新颖应用程序的发展, 诸如人脸识别、自然语言处理等[1]. 这些应用往往需要大量的计算资源, 虽然移动设备已得到长足的发展, 但受限于自身电池容量, CPU 性能等物理属性, 仍然难以满足计算密集型任务的需求[2,3].

移动云计算为解决上述问题提供了思路, 通过将任务卸载至云端进行处理, 一定程度上突破了移动设备的硬件限制[4]. 但是, 此方法在面临密集任务时会导致信道的拥堵, 极大地增加网络负荷, 对服务质量造成负面影响[5]; 此外, 云端服务器的部署位置往往距移动终端较远, 因此会产生额外的无线电回程, 使得任务的延迟增加, 难以满足任务的时延需求[6].

在此背景下, 移动边缘计算 (mobile edge computing, MEC) 应际而生. 通过在网络边缘端部署 MEC 服务器, 为用户提供算力资源的同时降低了任务的无线电回程, 提升了服务质量, 更好地满足用户的需求[7]. 相较于在本地执行计算任务, 移动边缘计算有选择地将任务卸载至 MEC 服务器, 克服了移动设备自身计算能力不足的问题. 对比于传统的移动云计算, MEC 服务器的计算资源一般弱于云服务器, 但其降低了任务的处理时延, 减小了网络的负担, 提升了服务质量, 能够在任务资源需求与时延之间取得平衡[8]. 在移动边缘计算中, 越来越多移动设备的接入, 使得设备间相互干扰, 降低了传输速率, 进一步造成了能源的不必要消耗以及传输时间的增加, 甚至使得 MEC 服务器过载[9]. 通过制定合理的任务卸载以及资源分配方案[10,11]可以有效地降低任务的延迟以及能源消耗, 从而显著地提升移动边缘计算中用户的服务体验[12]. 另一方面, 移动边缘计算任务具有不确定性, 如计算任务是否发布的不确定性, 发布时间的不确定性, 任务数量的不确定性以及任务所需的计算资源的不确定性. 因此, 在综合考虑不确定性的情况下, 研究移动边缘计算任务卸载及资源分配问题具有重要意义.

基于上述问题, 本文构建了一种移动边缘计算不确定性任务持续卸载模型. 模型基于持续时间片划分的任务多批次处理技术, 将不同时刻发布的任务分置于不同批次实现对任务的持续处理, 并引入了多设备资源协同机制, 通过允许设备之间共享计算资源以提升对密集任务的承载能力. 针对该模型, 设计一种基于负载均衡算子的自适应策略选择算法 (adaptive selection strategy algorithm based on load balancing, ASSLB) 对模型进行求解, 更有效地实现任务的卸载及资源分配. 基于泊松分布实现了对不确定任务场景模型的仿真, 仿真结果显示了移动用户在时延约束下的总能耗随系统对任务的刷新频率增长呈现下降趋势, 且相较于对照算法, ASSLB 可以有效降低系统能耗.

1 相关工作

移动边缘计算任务卸载与资源分配是紧密耦合的两个问题, 任务卸载问题本身为离散变量问题, 而资源分配问题的变量为连续变量, 因此, 这是一项具有挑战性的混合变量优化问题[13]. 从问题复杂度来看, 任务卸载与资源分配问题为 NP 难问题[14], 随着任务规模的增大, 求解难度剧增.

目前已有诸多相关研究工作, 文献 [15] 基于单基站场景提出了将系统能耗作为优化目标的任务卸载及资源分配模型, 在模型中引入一种协作模式, 即每个移动设备可以与其他移动设备共享计算资源, 同时设计了一种基于改进蚁群算法的 BiJOR 算法, 并在移动边缘计算任务卸载及资源分配问题上, 对比由蚁群算法 (ACS)[16]与综合学习粒子群算法 (CLSPO)[17]所结合的 ACS-CLSPO 算法, 表明了其优越性; 文献 [18] 针对计算密集型和时延敏感型应用场景, 考虑任务间的依赖关系构建了移动边缘计算任务卸载及资源分配的联合优化模型; 文献 [19] 在多用户-单 MEC 场景下提出一种轻量级的请求和允许框架, 更大限度地降低了设备在延迟需求下的能耗; 文献 [20] 建立了多用户-单 MEC 场景下的二进制卸载模型, 并降低了任务卸载及资源分配问题的耦合度, 以加权和计算率最大化为目标进行卸载方案和资源分配方案的制定; 文献 [21] 提出了一种具有数据安全性的多用户资源分配和计算卸载模型, 在多用户场景下联合考虑计算资源和无线资源, 以保证共享资源的有效利用; 文献 [22] 面向资源受限场景, 提出了一种面向多用户的串行任务动态卸载模型, 遵循先来先服务的原则, 充分考虑多用户请求对服务器资源的竞争关系, 动态调整选择策略; 文献 [23] 将求解移动边缘计算任务卸载及资源分配问题分解为两个子问题, 并分别采用分布式势能博弈算法以及云和无线资源分配算法予以求解; 文献 [24] 将计算卸载问题转化为考虑完

工时间和能量的系统能效成本最小化问题, 提出一种由时钟频率配置, 发射功率分配, 信道速率调度和卸载策略选择组成的分布式算法进行求解.

然而上述研究中均未考虑到在移动边缘计算场景下任务所具有的不确定性, 包括任务是否发布的不确定性, 任务发布时间不确定性, 任务数量的不确定性. 文献 [25] 构建了一种任务卸载模型, 将动态发布的任务分置于不同时间片内进行卸载, 并根据不同时间片内任务的到达情况将其建模为 0-1 模型, 一定程度上模拟了任务的发布数量的不确定性特征. 然而, 文献 [25] 在固定时间片内发布固定数量的任务, 忽略了任务的数量不确定性, 且将任务发布固定在时间片起始时刻, 忽略了任务发布时间上的不确定性; 文献 [26] 研究了在由多个时隙组成的有限时间范围内动态任务到达的联合计算和无线资源分配优化, 但是其将任务持续到达的时间划分成多个时隙, 并且规定了任务在每个时隙开始时到达, 不符合实际情况; 文献 [27] 的不确定性体现在任务所需资源的不确定性, 即在任务计算之前不确定其计算量, 但是在其他不确定性方面并未考虑.

本文构建了一种移动边缘计算不确定性任务持续卸载模型, 该模型中任务持续发布且在时间上具有不确定性, 采用基于持续时间片划分的任务多批次处理技术进行处理, 将不同时刻发布的任务分置于不同时间片内分批处理, 在每个时间片内, 任务是否发布, 任务发布数量以及任务本身所需计算资源也是具有不确定性的. 同时, 该模型中又引入多设备计算资源协同机制, 允许多设备之间共享计算资源, 区别于文献 [15] 中每个移动设备只允许计算一个任务的协同机制, 本文允许设备在自身计算资源允许的情况下最大化执行任务数, 以提升协同效果.

为了更好地在该场景下提升求解任务卸载及资源分配问题的能力, 本文提出了一种基于负载均衡的自适应策略选择算法, 算法中融合了不同的搜索策略, 通过自适应的方式进行策略的选择. 考虑到场景中多 MEC 服务器的特点, 本文设计了一种负载均衡算子进行局部搜索, 避免任务在 MEC 服务器上过度卸载造成信道拥堵进而造成能耗增加, 进一步提升算法跳出局部最优的能力. 仿真结果验证了自适应策略选择机制以及负载均衡算子的有效性.

2 移动边缘计算不确定性任务持续卸载模型

本文设计场景中的任务由设备持续发布, 即设备会在不同时刻发布任务, 对于系统而言, 任务发布时间是不确定的, 系统将任务划分在不同批次内进行处理, 每批次中的任务量也是不确定的, 本文将该场景称为不确定性任务场景. 该模型是对不确定性任务场景下移动边缘计算任务卸载问题的建模, 包括了系统模型和优化模型两部分.

2.1 系统模型

系统模型是对不确定性任务场景下移动边缘计算系统的描述. 该部分分别从系统所处场景, 任务传输, 任务计算, 任务时延等几个角度对系统进行描述.

(1) 不确定性任务场景模型

由于任务的发布在时间上具有先后顺序, 且对系统而言, 任务的发布时间是不确定的, 难以同时对所有任务进行卸载及资源分配, 因此每隔一定的时隙, 系统对该时隙产生的任务进行处理. 图 1 描述了不确定性任务场景示例. 场景中设备发布任务的时间是无法确定的, 以设备 1 为例, 来自设备的 3 个任务在发布前, 其任务的发布时间对系统而言是无法确定的. 这导致了在任务发布前, 任务被划分的处理批次是不确定的. 这使得在每一批次中, 任务是否发布具有了不确定性, 以设备 2 为例, 其只在第 $k+1$ 批次内发布了任务, 而在其他批次中未发布任务. 同时, 每批次内任务数量是不确定的, 以 k 批次为例, 该批次中仅仅处理了 1 个任务, 而 $k+1$ 批次中则处理了 5 个任务.

设定系统任务的刷新间隔为 ΔT, 每次会得到一批任务, 若要得到整个时间段内的所有任务, 需要 l 个批次, 批次的集合为 $K=\{1,2,\ldots,k,\ldots,l\}$, 其计算如公式 (1) 所示:

$$l=\left\lceil\frac{T}{\Delta T}\right\rceil \tag{1}$$

其中, k 表示任务批次, 可知 $k\in K$, T 为整个时间段的时长.

T_k 代表第 k 批任务的处理时刻, 关于 T_k 的递推公式见公式 (2):

$$T_k = T_{k-1} + \Delta T = T_{k-2} + 2\Delta T = \ldots = T_1 + (k-1)\Delta T \tag{2}$$

设场景中共有 m 个设备, n 个基站, 则每个任务具有 $n+m$ 种分配模式, 基站集合表示为 $N=\{1,2,\ldots,n\}$, 设备集合表示为 $M=\{n+1,n+2,\ldots,n+m\}$.

设 q^k 代表了第 k 批次的任务数量, U_i^k 表示在第 k 批任务中的第 i 个任务, 则 $i\in\{1,2,\ldots,q^k\}$, 五元组 $U_i^k=\left(C_i^k,D_i^k,B_i^k,T_{\max i}^k,e_i^k\right)$ 表示任务 U_i^k 的属性, C_i^k 是任务 U_i^k 所需的 CPU 周期数, D_i^k 是任务 U_i^k 的上行数据量, B_i^k 表示任务 U_i^k 的下行数据量, $T_{\max i}^k$ 是任务 U_i^k 所允许的最大时延.

对于任务 U_i^k, e_i^k 表示发布该任务的设备编号, 可知 $e_i^k\in M$, 使用 j 表示该任务的卸载模式, 可知 $j\in M\cup N$, 在该系统中任务有 3 种卸载模式.

- $j\in N$ 表示任务卸载至基站模式执行.
- $j\in M\backslash\{e_i^k\}$ 表示任务卸载至协作设备上进行执行.
- $j=e_i^k$ 表示任务卸载至本地进行执行.

O 为 $q^k\times(m+n)$ 维的矩阵, o_{ij}^k 为其中元素, 表示任务 U_i^k 的卸载决策, 当任务采用模式 j 进行卸载时 $o_{ij}^k=1$, 否则 o_{ij}^k=0.

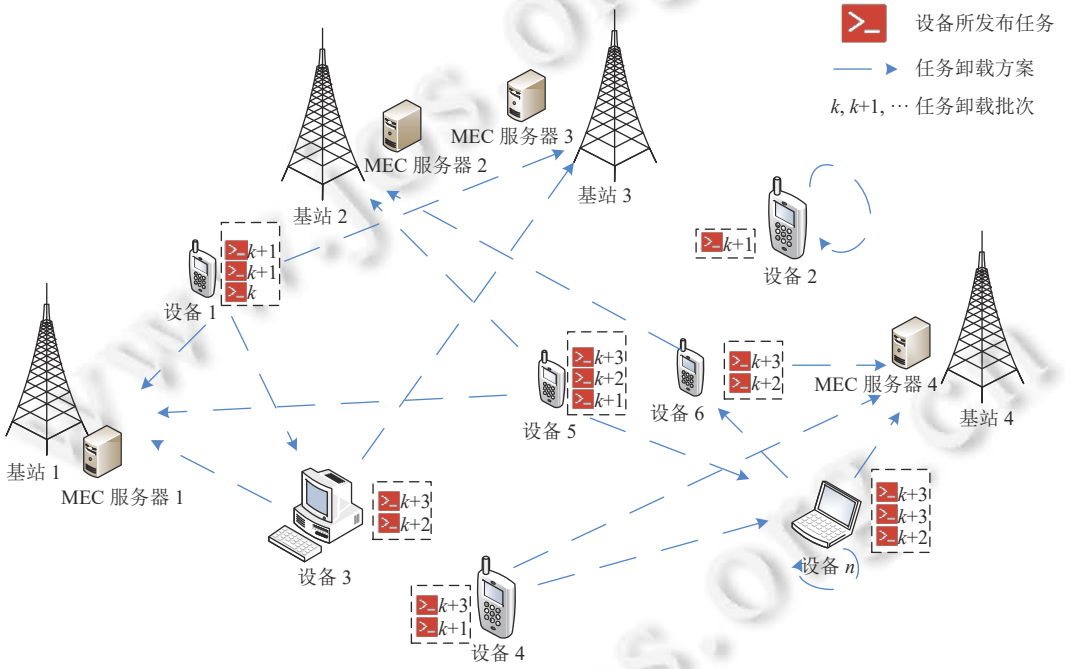


图 1 系统场景图

(2) 传输模型

系统中数据传输速率分为上行速率和下行速率, 当任务 U_i^k 采用模式 j 进行卸载时, 上行速率为任务进行卸载时的速率, 在本地模式时, 上行速率为∞, 而在非本地模式时, 遵循香农公式, 如公式 (3) 所示[15]:

$$R_{ij}(o)=\begin{cases}\infty, & j=e_i^k\\ B\log_2\left(1+\dfrac{p_i^{\mathrm{tr}}H_{ij}}{\omega+\sum_{g=1,e_g^k\neq e_i^k}^{q^k}o_{gj}^k p_g^{\mathrm{tr}}H_{gj}}\right), & j\in M\cup N\backslash\{e_i^k\}\end{cases} \tag{3}$$

其中, B 为信道带宽, H_{ij} 为任务 U_i^k 选择模式 j 时的信道增益, p_i^{tr} 为发布任务 U_i^k 的设备 e_i^k 发送任务数据时的功率, ω 为背景噪声功率.

任务 U_i^k 在模式 j 下的传输时间 $T_{\mathrm{tr}ij}^k$ 见公式 (4)[28]:

$$T_{\mathrm{tr}ij}^k(o)=\frac{D_i^k}{R_{ij}(o)}+\frac{B_i^k}{R_{ji}(o)},j\in M\cup N \tag{4}$$

任务 U_i^k 在模式 j 下的传输能耗 $E_{\mathrm{tr}ij}^k$ 见公式 (5)[28]:

$$E_{\mathrm{tr}ij}^k(o)=\begin{cases}\dfrac{p_i^{\mathrm{tr}}D_i^k}{R_{ij}(o)}+\dfrac{p_i^{\mathrm{re}}B_i^k}{R_{ji}(o)},j\in N\\ \dfrac{p_i^{\mathrm{tr}}D_i^k}{R_{ij}(o)}+\dfrac{p_i^{\mathrm{re}}B_i^k}{R_{ji}(o)}+\dfrac{p_j^{\mathrm{tr}}B_i^k}{R_{ji}(o)}+\dfrac{p_j^{\mathrm{re}}D_i^k}{R_{ij}(o)},j\in M\backslash\{e_i^k\}\\ 0,j=e_i^k\end{cases} \tag{5}$$

其中, p_j^{re} 为卸载模式 j 所对应设备的接收功率[15]. 在该场景下, 数据上行和下行过程中的信道条件以及环境噪声是相同的, 因此 $R_{ji}(o)$ 和 $R_{ij}(o)$ 是对称相等的.

(3) 计算模型

任务 U_i^k 在模式 j 下的计算时间 $T_{\mathrm{ca}ij}^k$ 见公式 (6)[29]:

$$T_{\mathrm{ca}ij}^k\left(r_{ij}^k\right)=\frac{C_i^k}{r_{ij}^k},j\in M\cup N \tag{6}$$

其中, r_{ij}^k 表示任务 U_i^k 在模式 j 下所分配的计算资源.

当任务卸载至基站模式时, 不考虑其计算能耗, 而在设备模式下需考虑计算能耗, 则任务 U_i^k 在模式 j 下的计算能耗 $E_{\mathrm{ca}ij}^k$ 如公式 (7) 所示[29]:

$$E_{\mathrm{ca}ij}^k\left(r_{ij}^k\right)=\alpha(r_{ij}^k)^2C_i^k,j\in M \tag{7}$$

其中, 计算任务能耗与移动设备自身硬件电路结构有关, 本文用常数系数 α 表示.

(4) 时延模型

任务 U_i^k 所需的总时间采用 $T_{\mathrm{tot}ij}^k$ 表示, 其计算公式见公式 (8):

$$T_{\mathrm{tot}ij}^k(o,r_{ij}^k)=\begin{cases}T_{\mathrm{ca}ij}^k\left(r_{ij}^k\right)+T_{\mathrm{wa}ij}^k,j=e_i^k\\ T_{\mathrm{ca}ij}^k\left(r_{ij}^k\right)+T_{\mathrm{tr}ij}^k(o)+T_{\mathrm{wa}ij}^k,j\in M\cup N\backslash\{e_i^k\}\end{cases} \tag{8}$$

其中, $T_{\mathrm{wa}ij}^k$ 表示任务 U_i^k 在模式 j 下从产生到批处理之间的时间差.

(5) 计算资源模型

定义 x_i^k 表示第 k 批次的任务 U_i^k 是否完成计算, 如公式 (9) 所示:

$$x_i^k=\begin{cases}1,\text{任务}U_i^k\text{仍占用计算资源}\\ 0,\text{任务}U_i^k\text{已释放计算资源}\end{cases} \tag{9}$$

其中, 当 x_i^k 为 1 时, 表示任务 U_i^k 仍在计算中, 将占用其对应卸载模式的计算资源. 当 x_i^k 为 0 时, 表示该任务已完成计算, 其占用的资源得到释放并提供给接下来的任务使用.

截至 k 批次, 模式 j 对应的设备 (或 MEC 服务器) 的当前剩余计算资源为 $r_{\mathrm{rest}j}^k$, $r_{\mathrm{rest}j}^k$ 计算公式如公式 (10) 所示:

$$r_{\mathrm{rest}j}^k=r_{\mathrm{all}j}-\sum_{v=1}^{k-1}\sum_{i=1}^{q^v}x_i^vr_{ij}^v,\ j\in M\cup N \tag{10}$$

其中, $r_{\mathrm{all}j}$ 为模式对应的设备的总计算资源, 在该系统中, 由于任务对计算资源的占用会影响系统资源状态, 对任务的持续处理产生影响, 因此系统需要动态的调整任务卸载策略, 以最大化利用当前系统中剩余计算资源.

2.2 优化模型

优化模型是对系统中任务卸载与资源分配问题的描述. 在该场景下, 以最小化系统总能耗为目标的移动边缘计算任务卸载及资源分配模型如公式 (11) 所示. 其中, $C1$ 表示对于任意批次的每一个任务都得到了执行; $C2$ 表

示对于任意批次, 分配至模式 j 的所有任务所需的计算资源小于等于模式 j 当前剩余计算资源 $r^k_{\text{rest}\,j}$; $C3$ 和 $C4$ 表示当任务 U^k_i 分配至模式 j 时, 任务在该模式下所分配的计算资源 r^k_{ij} 大于 0, 否则所分配的计算资源 r^k_{ij} 为 0; $C5$ 表示对于任意批次 k 中的任务, 必须在其最大时延内完成. 另外, 在下文的求解过程中, 我们定义 $Fit=\sum_{i=1}^{q^k}\sum_{j=1}^{n+m}o^k_{ij}\left(E^k_{\text{ca}ij}\left(r^k_{ij}\right)+E^k_{\text{tr}ij}(o)\right)$ 作为适应度值函数, 其决定一个解的好坏程度, 对解个体进行优胜劣汰, 实现对优化问题的寻优.

$$\begin{cases} P:\min\limits_{o,r^k_{ij}}\left(\sum\limits_{i=1}^{q^k}\sum\limits_{j=1}^{n+m}o^k_{ij}\left(E^k_{\text{ca}ij}\left(r^k_{ij}\right)+E^k_{\text{tr}ij}(o)\right)\right) \\ \text{s. t.} \\ C1:\sum\limits_{j=1}^{n+m}o^k_{ij}=1,\forall i\in\{1,2,\ldots,q^k\},\forall k\in K \\ C2:\sum\limits_{i=1}^{q^k}o^k_{ij}r^k_{ij}\leqslant r^k_{\text{rest}\,j},\forall j\in M\cup N,\forall k\in K \\ C3:r^k_{ij}>0,\forall o^k_{ij}=1,\forall k\in K \\ C4:r^k_{ij}=0,\forall o^k_{ij}=0,\forall k\in K \\ C5:0<\sum\limits_{j=1}^{m+n}o^k_{ij}T^k_{\text{tot}ij}\leqslant T^k_{\text{max}ij},\forall i\in\{1,2,\ldots,q^k\},\forall k\in K \end{cases} \tag{11}$$

3 求解方法

第 2 节构建了一种面向不确定性任务场景下的移动边缘计算持续卸载模型, 在模型中每批次任务的卸载决策及资源分配问题是一个 NP 难问题[15], 传统的数学方法难以解决, 故考虑采用智能优化算法予以求解, 本文设计了一种基于负载均衡算子的自适应策略选择算法 (ASSLB) 求解此问题. 算法中融合了邻域搜索策略以及个体重置算子进行个体的搜索, 其中邻域搜索策略由不同的破坏和修复算子组成, 并引入一种负载均衡算子来提升算法跳出局部最优的能力.

3.1 自适应策略选择机制

算法中融合了邻域搜索策略以及个体重置策略进行个体的搜索, 其中邻域搜索策略由不同的破坏和修复策略组成. 在搜索过程中 ASSLB 通过自适应机制动态的调整算法对策略的选择.

(1) 邻域搜索策略

本文依据模型特点设计了多种不同的破坏, 修复策略, 实现对解邻域的搜索, 得到更优个体. 图 2 展示了这一过程.

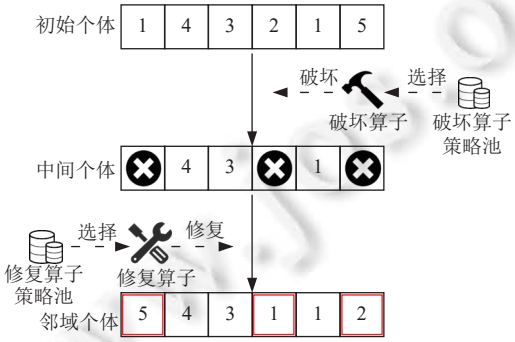


图 2 邻域搜索过程

图 2 中初始个体经过破坏后, 形成了在某些维度上空白的中间个体, 然后通过修复策略对中间解的空白维度进行修复得到新个体, 该个体被认为是初始个体的邻域个体. ASSLB 中融合了两种破坏与修复策略, 分别为随机破坏策略 (random destroy strategy, RDS) 和依据卸载优先级破坏策略 (destroy strategy based on offloading priority,

DSOP), 随机修复策略 (random repair strategy, RRS) 和依据卸载优先级修复策略 (repair strategy based on offloading priority, RSOP). 这两种破坏与修复策略共同组成了策略池.

DSOP 与 RSOP 是基于任务卸载优先级的, 对任务卸载优先级评估如图 3 所示.

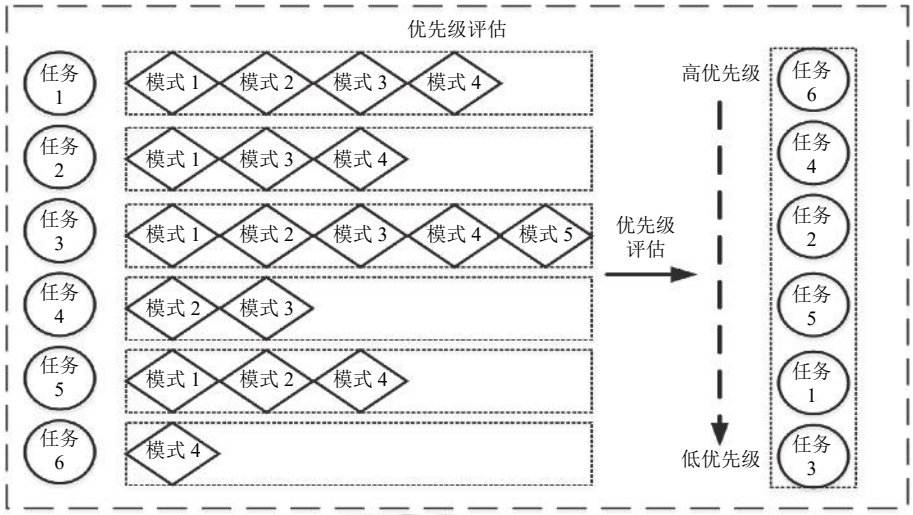


图 3 任务卸载优先级评估

图 3 描述了任务卸载优先级的评估过程. 如图 3 所示, 在得到任务的可行模式后, 任务 1 到任务 6 按照可行模式数量进行排序, 可行模式越少的任务其卸载优先级越高. 其中任务的可行模式需要通过可行模式评估算法获得, 具体描述如下.

首先依据公式 (11) 中 $C1-C5$, 评估得到每个任务的可行模式数量. 通过公式 (11) 中 $C5$ 可知, 任务 U_i^k 在模式 j 下计算资源的最小需求如公式 (12) 所示:

$$r_{\min ij}^k=\begin{cases}\dfrac{C_i^k}{T_{\max i}^k}, & j=e_i^k, j\in M\cup N\\ \dfrac{C_i^k}{T_{\max i}^k-T_{\mathrm{tr}ij}^k}, & j\neq e_i^k, j\in M\cup N\end{cases} \tag{12}$$

通过公式 (11) 中 $C2-C4$ 可知, 对于任务, 其可行模式需满足公式 (13).

$$0\leqslant r_{\min ij}^k\leqslant r_{\mathrm{rest}\,j}^k,\ j\in M\cup N \tag{13}$$

对于任务 U_i^k, 算法 1 描述了其可行模式的评估过程.

算法 1. 任务可行模式评估算法.

输入: 任务 U_i^k 的数据, 基站以及用户设备数据;

输出: 任务 U_i^k 的可行模式集合 Q_i^k.

Step 1. 初始化任务 U_i^k 的可行模式集合 $Q_i^k=\varnothing$.

Step 2. 依据公式 (12), 得到任务 U_i^k 在各个模式下的最小所需资源.

Step 3. 取模式 $j\in M\cup N$, 判断其对应的设备或 MEC 服务器的剩余计算资源 $r_{\mathrm{rest}\,j}^k$ 是否满足公式 (13), 如果满足则将模式放入集合 Q_i^k 中. 重复 Step 2, 直至完成对模式的判断.

Step 4. 返回可行模式集合 Q_i^k.

完成上述步骤后, 算法对每个任务的可行模式数量进行统计, 评估得到各个任务的卸载优先级. DSOP 与

RSOP 将通过任务的卸载优先级决定破坏以及修复顺序. DSOP 按照卸载优先级从低到高的顺序破坏个体, 因为低卸载优先级的任务拥有更多的可行模式可以选择, 为接下来的个体的修复提供了更多的可能性. RSOP 将优先修复任务卸载优先级高的任务, 即优先对可行模式较少的任务进行卸载, 因为高卸载优先级的任务可行模式更少, 计算资源或计算时延的要求往往更加苛刻, 所以需优先得到卸载及资源分配.

RRS 和 RSOP 在修复过程中采用轮盘赌策略. 采用集合 L_i^k 表示任务 U_i^k 的可行模式集合, 任务 U_i^k 采用模式 j 进行卸载的概率 p_{ij}^k 如公式 (14) 所示:

$$p_{ij}^k = \frac{E_{\mathrm{add}ij}^k}{\sum_{w\in L_i^k} E_{\mathrm{add}iw}^k} \tag{14}$$

其中, $E_{\mathrm{add}ij}^k$ 为任务 U_i^k 采用模式 j 进行卸载时总能耗的增量, 其计算如公式 (15) 所示:

$$E_{\mathrm{add}ij}^k = \begin{cases} \sum_{z\in R\cup\{i\}} E_{\mathrm{tr}zj}^k(o) - \sum_{z\in R} E_{\mathrm{tr}zj}^k(o), j\in N \\ \left[\sum_{z\in R\cup\{i\}} E_{\mathrm{tr}zj}^k(o) - \sum_{z\in R} E_{\mathrm{tr}zj}^k(o)\right] + E_{\mathrm{ca}ij}^k\left(r_{ij}^k\right), j\in M\backslash\{e_i^k\} \\ E_{\mathrm{ca}ij}^k\left(r_{ij}^k\right), j = e_i^k \end{cases} \tag{15}$$

其中, R 是已经通过模式 j 进行卸载的任务集合.

(2) 个体重置策略

单一地采用上述的破坏以及修复策略可能会使搜索陷入局部最优, 为了避免这一现象, ASSLB 在策略池中加入了一种基于轮盘赌机制的个体重置策略, 生成一个全新个体.

该策略与 RSOP 类似, 按照任务卸载优先级由高到低的顺序逐一对任务进行卸载, 并采用公式 (14), 公式 (15) 计算卸载至不同模式的概率.

(3) 自适应选择机制

ASSLB 为策略池中的破坏与修复策略以及个体重置策略分配了权重, 在搜索过程中通过动态的调整权重来控制对策略的选择. 具体流程如下.

设 Ω^- 和 Ω^+ 分别为破坏策略与修复策略的集合, 个体重置策略被视作一种特殊的修复策略加入至集合 Ω^+ 中. ρ_i^- 为破坏策略 i 的权重, ρ_j^+ 为修复策略 j 的权重, φ_i^- 表示破坏策略 i 被选择的概率, φ_j^+ 表示修复策略 j 被选择的概率, 其计算由公式 (16) 所示[30]:

$$\begin{cases} \varphi_i^- = \dfrac{\rho_i^-}{\sum_{a=1}^{|\Omega^-|}\rho_a^-} \\ \varphi_j^+ = \dfrac{\rho_j^+}{\sum_{a=1}^{|\Omega^+|}\rho_a^+} \end{cases} \tag{16}$$

公式 (16) 显示了权重对策略选择概率的影响, ASSLB 为了实现在搜索过程中对破坏策略与修复策略的自适应选择, 在搜索过程中, 每生成一个个体后会动态的调整策略的权重. 调整公式如公式 (17) 所示[30]:

$$\begin{cases} \rho_i^- = \lambda\rho_i^- + (1-\lambda)\psi \\ \rho_j^+ = \lambda\rho_j^+ + (1-\lambda)\psi \end{cases} \tag{17}$$

其中, $\lambda\in(0,1)$, ψ 表示了对新生成个体的评估标准, 如公式 (18)[30]所示:

$$\psi = \begin{cases} \omega_1, \text{如果新个体的适应度值小于初始个体} \\ \omega_2, \text{如果新个体的适应度值大于初始个体,但可以完成所有任务的卸载} \\ \omega_3, \text{如果新个体无法完成所有任务的卸载} \end{cases} \tag{18}$$

其中, $\omega_1 > \omega_2 > \omega_3 > 0$.

3.2 负载均衡算子

考虑到场景中存在多个 MEC 服务器, ASSLB 引入了一种负载均衡算子. 采用该算子的动机是: 从理论方面, 由第 2.1 节系统模型中传输模型提出的公式 (3) 与公式 (5) 得知, 负载均衡算子可以减小传输能耗, 能产生较好的解, 利于增强算法跳出局部最优的能力, 提升算法的搜索效果, 从实践方面, 将负载均衡算子应用于算法中进行后续实验时取得了较好的效果.

场景中存在多个 MEC 服务器为设备群提供计算资源, 虽然在协同模式下设备会互相提供计算资源, 但是由于设备自身计算能力的限制, 设备不能负载过多计算任务, 而 MEC 服务器则拥有可以负载更多任务的能力, 同时可以负载一些计算资源要求更高的任务, 因此 MEC 服务器的任务负载状态会成为影响解的质量的重要因素. 当多个任务在同一 MEC 服务器上进行卸载时, 会产生信道干扰造成能耗增加. 基于以上考虑, 本文设计了一种负载均衡算子, 通过评估各 MEC 服务器中的负载情况, 将负载过大的 MEC 服务器中的任务卸载至其他负载较小的 MEC 服务器. 负载均衡算子作用过程如算法 2 所示.

算法 2. 负载均衡算子.

输入: 当前种群中最优卸载决策与资源分配方案;
输出: 负载均衡处理后的最优卸载决策与资源分配方案.

Step 1. 从当前最优个体中统计每个 MEC 服务器所负载的任务数量.
Step 2. 依据公式 (19) 评估 MEC 服务器的均衡等级.

$$grade_\theta = \begin{cases} grade1, & num_\theta - num\_avg < \alpha_1 \\ grade2, & \alpha_1 \leqslant num_\theta - num\_avg < \alpha_2 \\ grade3, & \alpha_2 \leqslant num_\theta - num\_avg \end{cases} \tag{19}$$

其中, θ 代表了 MEC 服务器的编号, $grade_\theta$ 为 MEC 服务器 θ 的均衡等级, num_θ 为 MEC 服务器 θ 所负载任务数量, num_avg 为当前 MEC 服务器平均负载任务, α_1, α_2 ($\alpha_1 < \alpha_2$) 为均衡等级阈值.
Step 3. 根据 MEC 服务器均衡等级确定其所需迁移任务数量, 如公式 (20) 所示:

$$remove\_num_\theta = \begin{cases} 0, & grade_\theta = grade1 \\ \eta_1, & grade_\theta = grade2 \\ \eta_2, & grade_\theta = grade3 \end{cases} \tag{20}$$

其中, $remove\_num_\theta$ 为 MEC 服务器 θ 的迁移任务数量, 在算法中设定了 2 种不同均衡等级下的迁移数量, 分别为 η_1, η_2, 其中 $0 < \eta_1 < \eta_2 < num_\theta$, 所需迁移数量为 η_1, η_2 的 MEC 服务器加入集合 S.
Step 4. 在集合 S 中随机选择一个 MEC 服务器, 评估该 MEC 服务器上所负载的任务的卸载优先级.
Step 5. 选择卸载优先级最高的任务, 迁移至其他 MEC 服务器. 对于 MEC 服务器 θ, 其被选择迁移的概率如公式 (21) 所示:

$$selected\_p_\theta = \begin{cases} 0, & grade_\theta = grade3 \\ \mu_1, & grade_\theta = grade2 \\ \mu_2, & grade_\theta = grade1 \end{cases} \tag{21}$$

其中, $0 < \mu_1 < \mu_2 < 1$, 重复此步骤, 直至达到所需迁移数量.
Step 6. 返回 Step 4, 直至对集合 S 中所有 MEC 服务器完成任务迁移, 得到新的最优解.
Step 7. 对比新的最优解与初始最优解, 保留其中适应度值 Fit 更优者为负载均衡处理后的最优卸载决策与资源分配方案.

图 4 展示了负载均衡算子的作用过程. 如图 4 所示, 有 4 个 MEC 服务器, 分别为 MEC 服务器 1、MEC 服务

器 2、MEC 服务器 3、MEC 服务器 4, 它们负载不同数量的任务. 首先根据公式 (19) 计算出 MEC 服务器的负载均衡等级, 得出 MEC 服务器 1 的均衡等级为 *grade*1, MEC 服务器 2 的均衡等级为 *grade*2, MEC 服务器 3 的均衡等级为 *grade*3, MEC 服务器 4 的均衡等级为 *grade*1. 之后根据公式 (20) 确认 MEC 服务器所需迁移任务数量. 以 MEC 服务器 1 为例, 将选择卸载优先级最高的任务卸载至其他 MEC 服务器上, 其中 MEC 服务器被选择的概率计算如公式 (21) 所示, MEC 服务器 1 将重复这一过程, 直至达到所需迁移的任务数量.

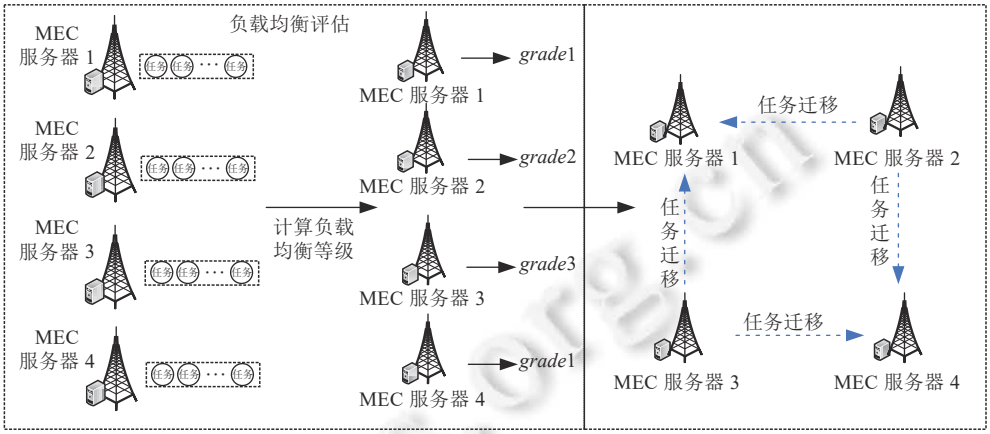


图 4 负载均衡算子作用过程

3.3 基于负载均衡算子的自适应策略选择算法

本文采用融合了自适应策略选择机制与负载均衡算子的 ASSLB 求解任务卸载及资源分配策略问题. ASSLB 与该问题的相关参数的映射关系如表 1 所示.

表 1 ASSLB 与问题的映射关系

ASSLB	任务卸载及资源分配问题
个体维度	用户任务数量
个体	单一卸载决策
种群	卸载决策与资源分配方案集合
适应度值	系统总能耗

ASSLB 具体求解过程如算法 3 所示.

算法 3. 基于负载均衡算子的自适应策略选择算法.

输入: 用户任务数据, 基站以及用户设备数据;
输出: 任务总能耗最优值, 最优卸载决策与资源分配方案.

Step 1. 初始化, 包括策略选择权重, 种群迭代次数等.
Step 2. 采用个体重置策略生成两个初始解, 将适应度值 *Fit* 较小的解作为全局最优解 *Global_best* , 适应度值较大的解作为全局最差解 *Global_worst* .
Step 3. 依据公式 (16) 计算所得概率进行策略选择, 然后分别使用 *Global_best* , *Global_worst* 按照所选策略生成个体, 由 *Global_best* 与 *Global_worst* 生成的个体将共同组成新种群. 根据公式 (17) 和公式 (18) 调整策略权重, 同时按照当前种群代数采用模拟退火思想控制由 *Global_worst* 生成的个体数量.
Step 4. 将新种群中适应度值最小的个体与 *Global_best* 的适应度值比较, 从中选择适应度值较小的个体更新为 *Global_best* , 将新种群中适应度值最大的个体与 *Global_worst* 的适应度值相比较, 从中选择适应度值较大的个体更新为 *Global_worst* .

Step 5. 使用负载均衡算子处理 *Global_best*, 生成新的个体, 若新个体适应度值小于 *Global_best* 的适应度值则更新 *Global_best*.

Step 6. 若当前种群迭代次数小于最大种群迭代次数, 则返回 Step 3, 否则输出 *Global_best*.

算法的时间复杂度分析: 在初始化阶段, 我们对每个任务检查其每个模式的可行性. 假设有 n 个任务, 每个任务都可以在本地以及基站执行, 每个任务需要检查 n 次. 因此, 候选模式的计算时间复杂度为 $O(n^2)$. 在卸载决策构建阶段, 破坏多少个解就需要生成多少个修复解, 设其数量为 $k \leqslant n$, 从可行候选模式集合中为每个任务选择一个模式, 需要 $k\cdot|Q_i|$ ($|Q_i|$为任务 i 的可行模式数量) 次计算, 设个体数量为 R, 由于$|Q_i| \leqslant n+1$ ($n+1$ 为总可行模式数量) 而需要 $R\cdot k\cdot|Q_i| \leqslant R\cdot n\cdot|Q_i| \leqslant R\cdot n\cdot(n+1)$ 次计算. 在资源卸载时, 对于每个任务, 需要计算给定卸载决策的计算资源的最优分配, 从而需要 $R\cdot k<R\cdot n$ 次计算. 因此, 算法的总体计算时间复杂度为 $O(R\cdot n^2)$.

4 实验分析

4.1 实验数据

为了使不确定性任务模型中任务的发布更贴近实际的情况, 模型中设定任务的发布时刻服从参数为 λ 泊松分布, 即同一设备所发布任务的间隔时间服从指数分布[31]. 本文采用生成发布任务的间隔时间来模拟任务发布, 则发布任务的间隔时间 t 的计算如公式 (22) 所示:

$$t=-\frac{\ln A}{\lambda},\ \lambda>0 \tag{22}$$

其中, A 是属于 (0,1) 的随机数. 由于不同设备发布任务的速度不同, 因此 λ 非固定值. 图 5 展示了任务发布的示例场景.

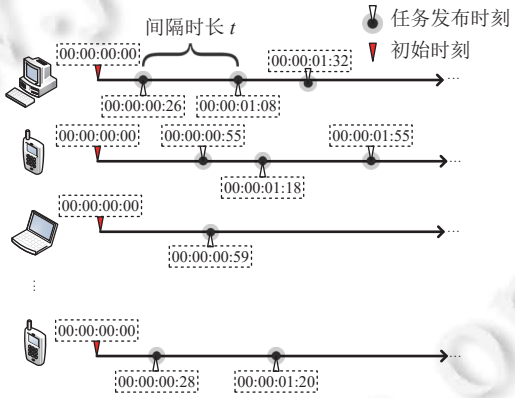


图 5 任务发布示例场景

我们假设所有用户随机分布在 2000 m×2000 m 的区域内, 基站位于中心区域. MEC 服务器的计算能力设为 200 GHz, 其数量共有 4 个, 用户设备的计算能力设置在 [0.5, 1.5] GHz 范围内, 其数量共 100 个. 任务所需计算资源 C_i^k 在 [10, 2500] MHz 内随机分布, 任务上行数据量 B_i^k 在 [1, 600] KB 内随机分布, 任务下行数据量 D_i^k 在 [0.1, 100] KB 内随机分布. 此外, 信道带宽 W 为 20 MHz, 数据传输功率 P^{tr} 和数据接收功率 P^{re} 设置在 [0.8, 1.3] W 范围内. 本文采用总能耗 (EC) 作为性能指标.

实验的仿真参数如后文表 2 所示.

4.2 实验结果与分析

本文利用 Matlab 对卸载决策与资源分配模型进行仿真实验, 采用 ASSLB 对模型进行求解, 与 BiJOR 算法相对照. 同时, 为了验证负载均衡算子的有效性, 本文将 ASSLB 与 ASS (ASS 采用与 ASSLB 完全相同的自适应策略选择搜索机制但无负载均衡算子) 进行对照, 此对照实验旨在验证在相同搜索机制下负载均衡算子的有效性.

4.2.1 单批次任务分析

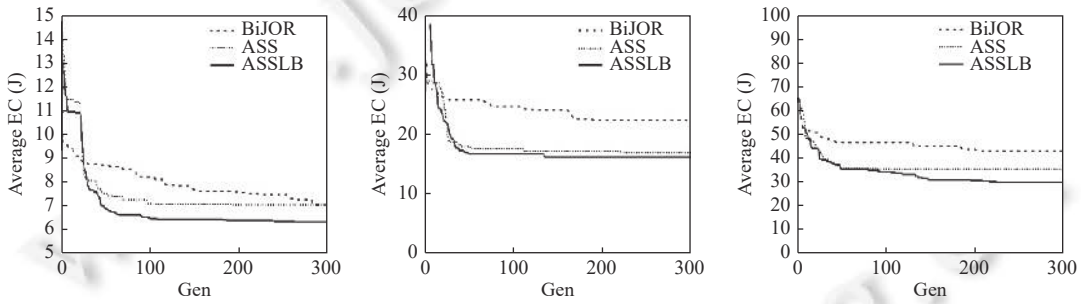
为了细致地分析算法性能, 本文设置了单批次任务下的实验. 该场景下实验的仿真参数与不确定性任务场景基本一致, 但不需要模拟任务的持续发布.

表 2 实验参数

实验参数	设置范围
MEC服务器计算能力	200 GHz
MEC服务器数量	4
用户设备计算能力	[0.5, 1.5] GHz
任务所需计算资源 C_i^k	[10, 2500] MHz
任务上行数据量 B_i^k	[1, 600] KB
任务下行数据量 D_i^k	[0.1, 100] KB
信道带宽 W	20 MHz
数据传输功率 P^{tr}	[0.8, 1.3] W
数据接收功率 P^{re}	[0.8, 1.3] W
用户设备数量 m	100
泊松分布参数 λ	[0.5, 5]

(1) 算法的收敛过程分析

图 6 展示了单一批次内, ASSLB 和 ASS 以及 BiJOR 的收敛情况, 设置批次内任务数量为 100, 150, 200.



(a) 任务规模 100 (b) 任务规模 150 (c) 任务规模 200

图 6 算法迭代收敛曲线

从图 6 可以看出, 随着迭代次数的增加, 系统能耗不断下降, 最后趋于稳定. 可以看出, 相较于采用自适应策略选择机制的 ASSLB 和 ASS, BiJOR 的收敛速度较慢, 跳出局部最优能力弱于 ASSLB 和 ASS, 导致其在趋于稳定后系统能耗高于 ASS 与 ASSLB, 这也反映出自适应选择机制的优越性. 由于 ASSLB 中加入了负载均衡算子, 使得算法具有更强的跳出局部最优的能力, 在算法迭代的初始几代, ASSLB 的收敛速度便超过了 ASS, 且当 ASS 的迭代趋于平稳后, ASSLB 仍然可以跳出局部最优, 进一步收敛.

相对于 BiJOR, ASSLB 迭代时收敛更为迅速. 事实上, 其时间效率高的原因还在于: 在 BiJOR 中, 会针对个体的每个维度进行更新. 在 ASSLB 中, 仅会针对解个体的部分维度 (本文中, 个体维度的破坏比例为 20% 或 40%) 进行破坏和修复.

(2) 负载均衡算子效果分析

为了验证在种群迭代过程中负载均衡算子的作用, 本文在单批次下对种群进化过程进行观察, 并统计了负载均衡算子的作用次数. 图 7 为负载均衡算子效果统计图.

算法在生成每代种群时都会调用负载均衡算子, 共有 300 代种群, 图片中依照从上往下, 从左至右的顺序排列有 300 个方格, 每个方格依顺序对应一代种群. 方格为白色, 代表该方格所对应的种群中负载均衡算子发挥了作

用, 成功搜索到了更优解.

图 7 显示了在算法迭代前期, 负载均衡算子更容易发挥作用, 这加速了算法的收敛, 在迭代的中后期趋于稳定时, 此时算法已经进入局部最优, 而负载均衡算子依然有一定概率使得算法跳出局部最优, 进一步收敛.

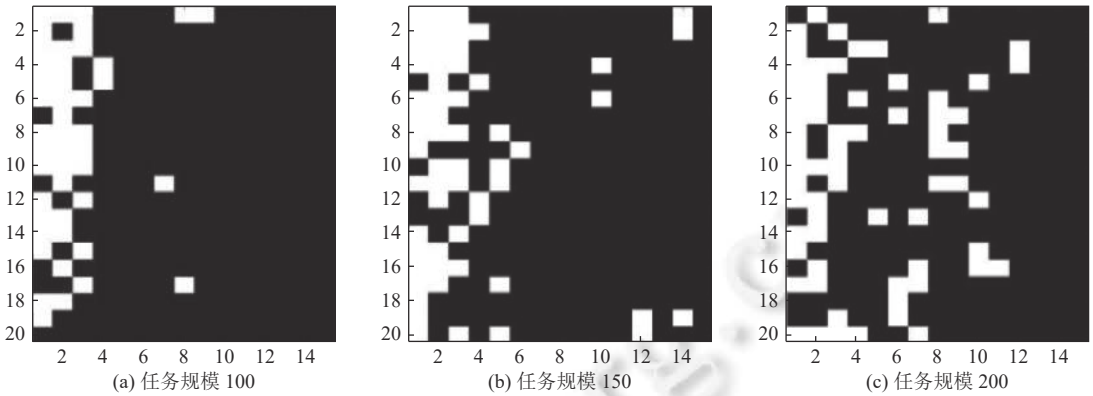


图 7 负载均衡算子效果

同时, 图像显示了在任务规模为 200 时, 负载均衡算子发挥作用的次数占比明显高于任务规模为 100 以及 150 时的比例, 这是因为随着任务规模增加, 系统负荷增大, 设备协同即本地模式一般难以承受较多任务, 此时 MEC 服务器的负荷显著增加, 担负更多任务, 负载均衡算子能够平衡每个 MEC 服务器的负担, 避免造成信道的拥堵使得能耗增加, 使算法跳出局部最优, 所以在此时更容易发挥其作用.

(3) 不同任务规模下算法效果分析

图 8 展示了在单批次内不同任务规模下, ASSLB, BiJOR 和 ASS 这 3 种算法的优化结果. 表 3 记录了单批次内不同任务规模下 ASSLB, BiJOR 和 ASS 这 3 种算法的能耗数据.

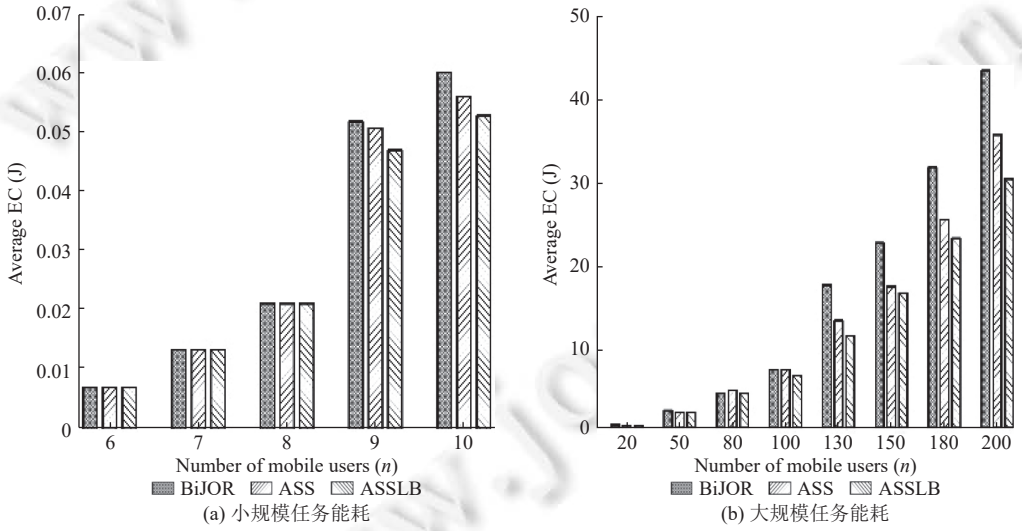


图 8 单批次任务处理能耗

图 8 中分别展示了单批次内不同任务规模下的 3 种算法的对照实验, 由图 8(a) 中可以看出, 在任务规模为 6, 7, 8 时, 3 种算法在求解效果上难以表现出差异, 由于任务规模较低, 算法很容易在低维解空间内搜索到接近最优解的任务卸载及资源分配方式. 而随着任务规模增加至 9, 10 时, ASSLB 逐渐展现出优势, 结合表 3 数据分析得知, 分别领先 BiJOR 9.62%, 13.11%.

表 3 任务处理能耗数据表

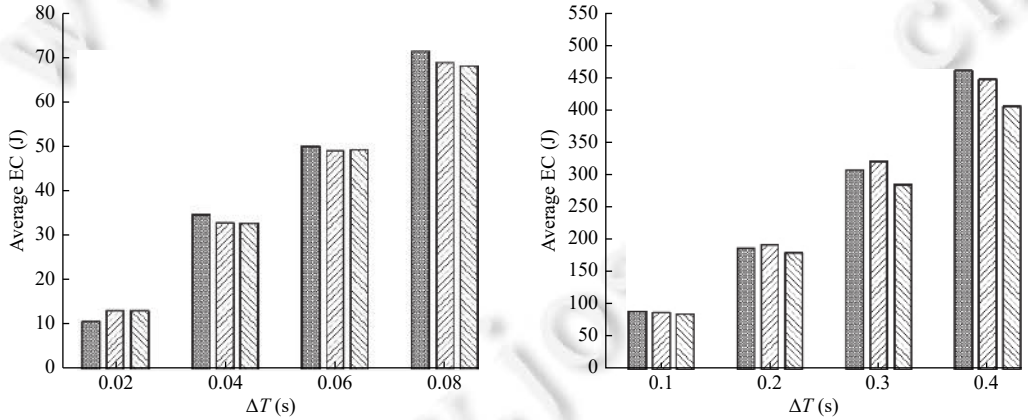
任务规模 (个)	BiJOR (J)	ASS (J)	ASSLB (J)	任务规模 (个)	BiJOR (J)	ASS (J)	ASSLB (J)
6	0.007	0.007	0.007	80	4.269	4.627	4.268
7	0.013	0.013	0.013	100	7.166	7.156	6.419
8	0.021	0.021	0.021	130	17.527	13.171	11.269
9	0.052	0.051	0.047	150	22.696	17.306	16.548
10	0.061	0.056	0.053	180	31.940	25.553	23.213
20	0.407	0.348	0.355	200	43.756	35.881	30.498
50	2.170	1.965	1.895				

而在图 8(b) 中展示了较大任务规模下 3 种算法的对照实验, 由图 8(b) 结合表 3 数据可以看出, 在任务规模从 20 至 200 的过程中, ASSLB 较 BiJOR 一直保持优势, 在任务规模为 20, 50, 80, 100 下, ASSLB 相较于 BiJOR 分别领先 12.78%, 12.67%, 5.57%, 10.42%, 当任务规模增至 130, 150, 180, 200 时, ASSLB 相较于 BiJOR 领先 35.70%, 27.09%, 27.32%, 30.30%, 可以看出, 在处理单一批次任务上, ASSLB 对于 BiJOR 的优势随着任务规模的扩大而扩大, 表明该算法在高维解空间展示了更好的搜索能力.

ASS 算法与 BiJOR 算法相比, 仍然展现出了一定优势, 在任务规模为 6, 7, 8 时与 BiJOR 结果基本持平, 而在任务规模为 9, 10, 20, 50, 100 时分别领先 1.92%, 8.20%, 14.50%, 9.45%, 0.14%, 而在任务规模为 130, 150, 180, 200 时分别领先 24.85%, 23.75, 20%, 18%, 没有负载均衡算子的 ASS 对比 BiJOR 依然保持了一定优势, 表明了自适应策略选择机制具有良好的性能.

4.2.2 不确定性任务持续卸载分析

本文通过模拟任务的持续发布构建不确定性任务场景, 通过移动边缘计算持续卸载模型刷新任务并分置于不同批次进行处理. 在不确定性任务模型下任务由移动设备发布之后, 通常不会被立即处理, 需要等待系统对任务的刷新, 不同的刷新间隔对任务的时延, 处理任务的数量等诸多方面均有影响, 为了验证在相同时间段内刷新间隔 ΔT 对系统能耗的影响. 图 9 展示了在相同时间段内不同 ΔT 对应的系统能耗, 图中横坐标为不同的 ΔT 所对应的单位时间, 而纵坐标代表了系统能耗.



(a) 较短刷新间隔下系统能耗 (b) 较长刷新间隔下系统能耗

图 9 不同刷新间隔下的系统能耗

由图 9 中可以看出, 在该时间段内, 随着 ΔT 增加, 3 种算法求解所得最终的系统能耗均呈上升趋势. 造成该现象原因来自于两方面, 一方面, 随着刷新间隔增大, 导致了任务发布后可能面临更久的等待时长, 这些等待造成的时延压缩了任务处理的时延, 系统不得不给予更多算力资源, 造成计算能耗的增加. 另一方面, 随着 ΔT 的增大, 在整个时间段内处理的任务数量虽然相同, 但是当 ΔT 较大时, 每一批次所处理的任务数量显著增加, 多个任务同时

传输造成了信道的拥堵, 增加了系统的传输能耗. 由此可知, 系统能耗随着 ΔT 增大而增大. 如果要降低移动边缘计算的系统能耗, 需要在服务器可以承载的范围内尽可能减小对任务刷新的时间间隔. 此外, 实验结果显示, 随着 ΔT 不断增大, 会逐渐超出系统的承载能力, 因此仿真将 ΔT 控制在 0.4 s 之内.

同时, 图 9 中还反映出了在 ΔT 取较小值时 (以 0.02 s 至 0.08 s 为例), 3 个算法的求解结果基本持平, 这说明在 ΔT 较小时, 任务处理呈现多批次少任务量的特点, 即通过较高的刷新频率, 使得每刷新一批次所处理任务数量减少, 降低了解空间的维度, 3 种算法都有能力搜索到接近最优的任务卸载及资源分配方案. 但是随着 ΔT 的增加, 每批次的任务数量不断上升, 导致解空间维度增加, 且任务的等待时延随之增加, 增大了算法搜索解的难度, 此时 ASSLB 对于 ASS 与 BiJOR 的优势也随之扩大, 结合表 4 (该表记录了不同刷新间隔下的系统能耗数据) 可以得知, 当 ΔT 分别为 0.1 s, 0.2 s, 0.3 s, 0.4 s 时, ASSLB 分别优于 BiJOR 3.84%, 3.59%, 7.53%, 11.82%, 优于 ASS 2.04%, 6.01%, 11.21%, 9.25%. ΔT 增加至 0.4 时, ASSLB 所得系统能耗显著优于 ASS, 而 ASS 的求解结果又显著优于 BiJOR, 这样的结果反映了在任务密集且时延敏感的场景下, 自适应策略选择机制处理任务的效果优于 BiJOR 中蚁群搜索机制. 同时, ASSLB 在 ASS 的基础上引入了负载均衡算子, 进一步提升了算法性能. 在 ΔT 为 0.2 s 以及 0.3 s 时, ASS 的能耗略高于 BiJOR 2.58%, 4.15%, 这样的结果反映出了自适应策略选择机制在中等任务规模下可能会出现陷入局部最优的情况.

表 4 系统能耗数据表

间隔 (s)	BiJOR (J)	ASS (J)	ASSLB (J)	间隔 (s)	BiJOR (J)	ASS (J)	ASSLB (J)
0.02	10.835	13.273	13.258	0.1	90.138	88.485	86.680
0.04	34.840	32.971	32.851	0.2	188.759	193.622	181.985
0.06	50.147	49.164	49.396	0.3	311.021	323.916	287.604
0.08	71.638	69.028	68.350	0.4	464.638	451.514	409.734

3 种算法在不同 ΔT 下截止不同批次的系统能耗如图 10 所示, 图中横坐标为系 统刷新截止批次, 纵坐标为截止该批次时的系统能耗. 图 10 显示在固定刷新间隔下, 截至不同刷新批次系统的当前能耗. 随着任务不断发布, 刷新批次增加, 系统能耗也不断增加, 且实验结果与图 9 相吻合.

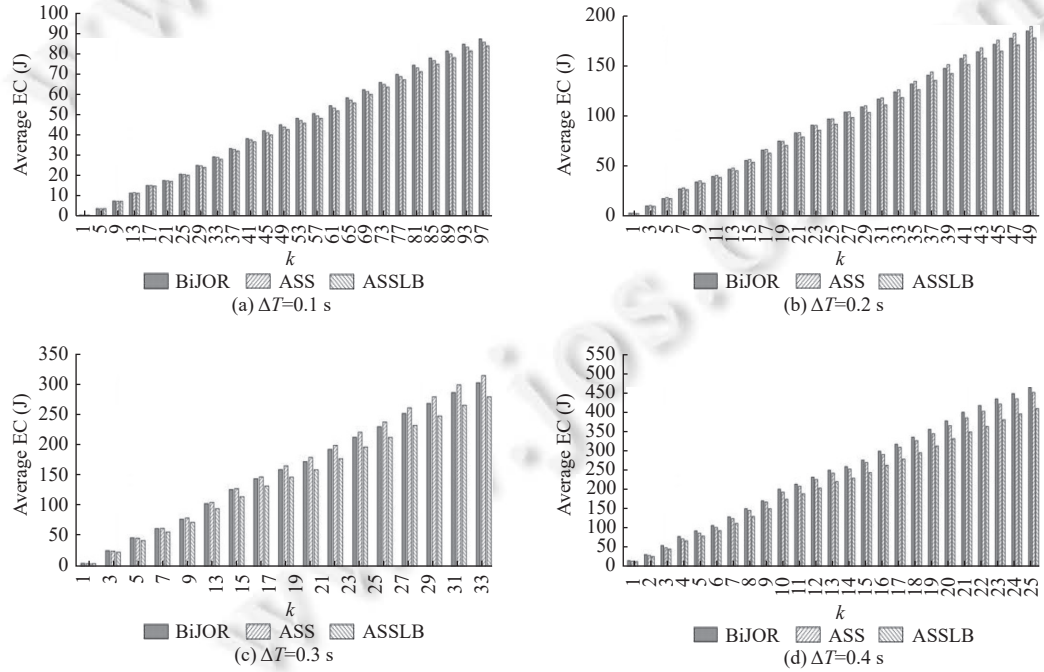
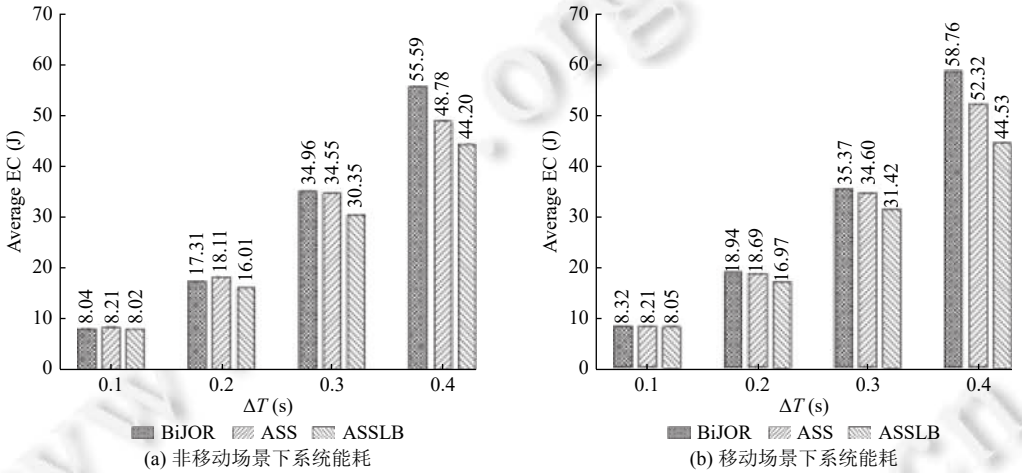


图 10 不同 ΔT 截止到不同批次下的系统能耗

4.2.3 移动场景下模型鲁棒性分析

在现实场景中, MEC 系统需要考虑用户设备的移动性. 在本文提出的移动边缘计算不确定性任务持续卸载模型中, 移动性会导致节点位置随时间改变, 从而使得部分参数动态变化, 这种场景称为移动场景. 为了验证模型在移动场景下的鲁棒性, 本文设计了基于移动场景下的实验.

实验通过模拟设备移动情况构建移动场景. 由于设备在处理过程中是移动的, 任务数据上行和下行时设备会产生位置信息的变化, 因此需要考虑移动距离的影响, 将任务数据上行开始时的位置作为设备的初始位置, 任务数据下行结束时的位置作为设备终点位置. 为了验证模型在移动场景下具有良好的鲁棒性, 该实验在已知设备移动轨迹的前提下截取了一段时间来对任务卸载与资源分配进行仿真, 通过更改刷新间隔 ΔT 得出不同情况下的系统总能耗, 并将仿真结果与非移动场景下的数据进行对比. 在本次实验中, 设置设备数量为 100. 由于随着刷新间隔 ΔT 不断增大, 系统的承载能力会逐渐超过上限, 因此实验将 ΔT 控制在 0.4 s 之内, 分别设置 ΔT 为 0.1 s, 0.2 s, 0.3 s, 0.4 s, 不同的刷新间隔会使实验所需的时间片变化, 但所有实验均在相同时间下进行. 图 11 展示了不同 ΔT 对应的系统能耗, 图中横坐标为刷新间隔 ΔT , 纵坐标代表了系统能耗.



(a) 非移动场景下系统能耗 (b) 移动场景下系统能耗

图 11 不同场景系统能耗

由图 11 可以看出, 不管是在非移动场景还是移动场景下, 3 种算法最终的系统能耗都随着 ΔT 的增加逐渐上升, 原因与第 4.2.2 节中的描述相同.

随着 ΔT 的增加, 算法搜索解的难度逐渐增大, 此时 ASSLB 相较于 ASS 与 BiJOR 的优势也逐渐扩大. 由图 11(a) 可知, 在非移动场景下, 当 ΔT 分别为 0.1 s, 0.2 s, 0.3 s, 0.4 s 时, ASSLB 分别优于 BiJOR 0.25%, 7.51%, 13.19%, 20.49%, 优于 ASS 2.44%, 11.60%, 12.16%, 9.39%. ΔT 较大时, ASSLB 所得系统能耗优势显著. 该结果再次表明了自适应策略选择机制以及负载均衡算子处理任务的优异效果.

在图 11(b) 中, 我们可以看出, 移动场景下的系统总能耗要高于非移动场景下的系统总能耗. 该现象的原因在于移动场景下需要考虑位置信息以及距离因素, 受距离的影响会增加处理任务的时间, 即移动性增加了卸载决策的难度, 进而导致系统总能耗增加. 当 ΔT 分别为 0.1 s, 0.2 s, 0.3 s, 0.4 s 时, ASSLB 分别优于 BiJOR 3.25%, 10.40%, 11.17%, 24.22%, 优于 ASS 1.95%, 9.20%, 9.19%, 14.89%. 通过两组实验数据对比可以得出, 在移动场景下 3 种算法应用于本文的模型仍有较好的效果. 即使因为考虑设备移动的情况而致使任务卸载决策的难度提升, 但依旧可以得出可行的卸载决策方案, 从而说明了在移动场景下, 本文提出的模型具有较好的鲁棒性.

5 总 结

本文对移动边缘计算任务卸载及资源优化问题进行了研究. 考虑任务的不确定性, 构建了移动边缘计算不确

定性任务持续卸载模型, 实现对不确定性任务的算力资源分配的优化. 同时设计了基于负载均衡的自适应策略选择算法以求解模型, 算法通过自适应的选择搜索策略获得良好的求解效果. 在仿真过程中, 设计任务的发布时刻符合泊松分布, 以模拟现实场景. 对照实验的结果表明了 ASSLB 可以有效地搜索更优个体, 且在规模较大的任务处理上具有显著优势.

ASSLB 采用自适应策略选择机制, 可以融合不同的搜索策略, 未来将会进一步丰富算法策略池中搜索策略以提高算法搜索能力. 此外, 实验表明系统能耗随着任务处理批次的增加呈现下降趋势, 本文对此进行了分析. 在实际情况中, 受限于系统整体的性能和任务的刷新成本, 系统在应用时需考虑上下文情景调整刷新任务的时间间隔, 这将是未来研究的方向之一.

References:

[1] Wang R, Cao Y, Noor A, Alamoudi TA, Nour R. Agent-enabled task offloading in UAV-aided mobile edge computing. Computer Communications, 2020, 149: 324–331. [doi: 10.1016/j.comcom.2019.10.021]

[2] Zhang GL, Ni SF, Zhao P. Learning-based joint optimization of energy delay and privacy in multiple-user edge-cloud collaboration MEC systems. IEEE Internet of Things Journal, 2022, 9(2): 1491–1502. [doi: 10.1109/JIOT.2021.3088607]

[3] Xu YQ, Zhang HL, Ji H, Yang LC, Li X, Leung VCM. Transaction throughput optimization for integrated blockchain and MEC system in IoT. IEEE Trans. on Wireless Communications, 2022, 21(2): 1022–1036. [doi: 10.1109/TWC.2021.3100985]

[4] Zhang KY, Gui XL, Ren DW, Li J, Wu J, Ren DS. Survey on computation offloading and content caching in mobile edge networks. Ruan Jian Xue Bao/Journal of Software, 2019, 30(8): 2491−2516 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5861.htm [doi: 10.13328/j.cnki.jos.005861]

[5] Peng HX, Shen XM. Multi-agent reinforcement learning based resource management in MEC- and UAV-assisted vehicular networks. IEEE Journal on Selected Areas in Communications, 2021, 39(1): 131–141. [doi: 10.1109/JSAC.2020.3036962]

[6] Zhang HB, Jing KL, Lui KJ, He XF. An offloading mechanism based on software defined network and mobile edge computing in vehicular networks. Journal of Electronics & Information Technology, 2020, 42(3): 645–652 (in Chinese with English abstract). [doi: 10.11999/JEIT190304]

[7] Etsi M. Mobile-edge computing—Introductory technical white paper. 2014. https://max.book118.com/html/2018/1006/80130221 0300 1125.shtm

[8] Zarandi S, Tabassum H. Delay minimization in sliced multi-cell mobile edge computing (MEC) systems. IEEE Communications Letters, 2021, 25(6): 1964–1968. [doi: 10.1109/LCOMM.2021.3051558]

[9] Li SC, Wang QY, Wang YF, Xie JL, Li CR, Tan DT, Kou WG, Li WJ. Joint congestion control and resource allocation for delay-aware tasks in mobile edge computing. Wireless Communications and Mobile Computing, 2021, 2021: 8897814. [doi: 10.1155/2021/8897814]

[10] Tong Z, Deng XM, Ye F, Basodi S, Xiao XL, Pan Y. Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. Information Sciences, 2020, 537: 116–131. [doi: 10.1016/j.ins.2020.05.057]

[11] Xiao Z, Dai XX, Jiang HB, Wang D, Chen HY, Yang L, Zeng FZ. Vehicular task offloading via heat-aware MEC cooperation using game-theoretic method. IEEE Internet of Things Journal, 2020, 7(3): 2038–2052. [doi: 10.1109/JIOT.2019.2960631]

[12] Fan WH, Liu Y, Tang BH, Wu F, Zhang HG. TerminalBooster: Collaborative computation offloading and data caching via smart basestations. IEEE Wireless Communications Letters, 2016, 5(6): 612–615. [doi: 10.1109/LWC.2016.2605694]

[13] Wang F, Zhang H, Zhou AM. A particle swarm optimization algorithm for mixed-variable optimization problems. Swarm and Evolutionary Computation, 2021, 60: 100808. [doi: 10.1016/j.swevo.2020.100808]

[14] Guo JF, Song ZZ, Cui Y, Liu Z, Ji YS. Energy-efficient resource allocation for multi-user mobile edge computing. In: Proc. of the 2017 IEEE Global Communications Conf. Singapore: IEEE, 2017. 1–7. [doi: 10.1109/GLOCOM.2017.8254044]

[15] Huang PQ, Wang Y, Wang KZ, Liu ZZ. A bilevel optimization approach for joint offloading decision and resource allocation in cooperative mobile edge computing. IEEE Trans. on Cybernetics, 2020, 50(10): 4228–4241. [doi: 10.1109/TCYB.2019.2916728]

[16] De Oliveira SM, Bezerra LCT, Stützle T, Dorigo M, Wanner EF, De Souza SR. A computational study on ant colony optimization for the traveling salesman problem with dynamic demands. Computers & Operations Research, 2021, 135: 105359. [doi: 10.1016/j.cor.2021.105359]

[17] Liang JJ, Qin AK, Suganthan PN, Baskar S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans. on Evolutionary Computation, 2006, 10(3): 281–295. [doi: 10.1109/TEVC.2005.857610]

[18] Dai YY, Xu D, Maharjan S, Zhang Y. Joint computation offloading and user association in multi-task mobile edge computing. IEEE

Trans. on Vehicular Technology, 2018, 67(12): 12313–12325. [doi: 10.1109/TVT.2018.2876804]

[19] Lyu XC, Tian H, Jiang L, Vinel A, Maharjan S, Gjessing S, Zhang Y. Selective offloading in mobile edge computing for the green Internet of Things. IEEE Network, 2018, 32(1): 54–60. [doi: 10.1109/MNET.2018.1700101]

[20] Huang L, Bi SZ, Zhang YJA. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans. on Mobile Computing, 2020, 19(11): 2581–2593. [doi: 10.1109/TMC.2019.2928811]

[21] Elgendy IA, Zhang WZ, Tian YC, Li KQ. Resource allocation and computation offloading with data security for mobile edge computing. Future Generation Computer Systems, 2019, 100: 531–541. [doi: 10.1016/j.future.2019.05.037]

[22] Liu W, Huang YC, Du W, Wang W. Resource-constrained serial task offload strategy in mobile edge computing. Ruan Jian Xue Bao/Journal of Software, 2020, 31(6): 1889–1908 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5705.htm [doi: 10.13328/j.cnki.jos.005705]

[23] Zhang J, Xia WW, Yan F, Shen LF. Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing. IEEE Access, 2018, 6: 19324–19337. [doi: 10.1109/ACCESS.2018.2819690]

[24] Wang QY, Guo ST, Liu JD, Yang YY. Energy-efficient computation offloading and resource allocation for delay-sensitive mobile edge computing. Sustainable Computing: Informatics and Systems, 2019, 21: 154–164. [doi: 10.1016/j.suscom.2019.01.007]

[25] Ma HR, Chen X, Zhou Z, Yu S. Dynamic task offloading for mobile edge computing with green energy. Journal of Computer Research and Development, 2020, 57(9): 1823–1838 (in Chinese with English abstract). [doi: 10.7544/issn1000-1239.2020.20200184]

[26] Wang F, Xing H, Xu J. Optimal resource allocation for wireless powered mobile edge computing with dynamic task arrivals. In: Proc. of the 2019 IEEE Int'l Conf. on Communications (ICC). Shanghai: IEEE, 2019. 1–7. [doi: 10.1109/ICC.2019.8761143]

[27] Eshraghi N, Liang B. Joint offloading decision and resource allocation with uncertain task computing requirement. In: Proc. of the 2019 IEEE Conf. on Computer Communications. Paris: IEEE, 2019. 1414–1422. [doi: 10.1109/INFOCOM.2019.8737559]

[28] Pu LJ, Chen X, Xu JD, Fu XM. D2D fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted D2D collaboration. IEEE Journal on Selected Areas in Communications, 2016, 34(12): 3887–3901. [doi: 10.1109/JSAC.2016.2624118]

[29] Wang KZ, Yang K, Magurawalage CS. Joint energy minimization and resource allocation in C-RAN with mobile cloud. IEEE Trans. on Cloud Computing, 2018, 6(3): 760–770. [doi: 10.1109/TCC.2016.2522439]

[30] Ropke S, Pisinger D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science, 2006, 40(4): 455–472. [doi: 10.1287/trsc.1050.0135]

[31] Yang B, Li JG, Kang YJ. Simulation of non-homogeneous Poisson process of customer arrival in stereo garage. Journal of Shenzhen University Science and Engineering, 2021, 38(2): 121–127 (in Chinese with English abstract). [doi: 10.3724/SP.J.1249.2021.02121]

附中文参考文献:

[4] 张开元, 桂小林, 任德旺, 李敬, 吴杰, 任东胜. 移动边缘网络中计算迁移与内容缓存研究综述. 软件学报, 2019, 30(8): 2491–2516. http://www.jos.org.cn/1000-9825/5861.htm [doi: 10.13328/j.cnki.jos.005861]

[6] 张海波, 荆昆仑, 刘开健, 贺晓帆. 车联网中一种基于软件定义网络与移动边缘计算的卸载策略. 电子与信息学报, 2020, 42(3): 645–652. [doi: 10.11999/JEIT190304]

[22] 刘伟, 黄宇成, 杜薇, 王伟. 移动边缘计算中资源受限的串行任务卸载策略. 软件学报, 2020, 31(6): 1889–1908. http://www.jos.org.cn/1000-9825/5705.htm [doi: 10.13328/j.cnki.jos.005705]

[25] 马惠荣, 陈旭, 周知, 于帅. 绿色能源驱动的移动边缘计算动态任务卸载. 计算机研究与发展, 2020, 57(9): 1823–1838. [doi: 10.7544/issn1000-1239.2020.20200184]

[31] 杨波, 李建国, 康耀军. 立体车库顾客到达的非齐次泊松过程模拟仿真. 深圳大学学报(理工版), 2021, 38(2): 121–127. [doi: 10.3724/SP.J.1249.2021.02121]

许斌(1981－), 男, 博士, 副教授, 主要研究领域为智能优化, 服务组合, 边缘计算.

赵云凯(1997－), 男, 硕士生, 主要研究领域为边缘计算, 智能优化算法.

朱剑鸣(1996－), 男, 硕士生, 主要研究领域为智能优化, 机器学习.

刘一川(2000－), 男, 硕士生, 主要研究领域为边缘计算, 智能优化算法.

李烜焘(2001－), 男, 本科生, 主要研究领域为边缘计算, 智能优化算法.

孙雁飞(1976－), 男, 博士, 研究员, 博士生导师, 主要研究领域为工业互联网, 能源互联网, 大数据管理与分析, 智能优化与控制.

季一木(1978－), 男, 博士, 教授, 博士生导师, 主要研究领域为云计算, 大数据, 人工智能及应用.