计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(11):309−319 [doi: 10.15888/j.cnki.csa.008792]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

Wi-Fi 网络下多 AP 协作边缘计算资源分配策略①

韩启福, 方旭明

(西南交通大学 信息科学与技术学院, 成都 611756)
通信作者: 方旭明, E-mail: xmfang@swjtu.edu.cn

摘　要: 近年来, AR/VR、在线游戏、4K/8K 超高清视频等计算密集且时延敏感型应用不断涌现, 而部分移动设备受自身硬件条件的限制, 无法在时延要求内完成此类应用的计算, 且运行此类应用会带来巨大的能耗, 降低移动设备的续航能力. 为了解决这一问题, 本文提出了一种 Wi-Fi 网络多 AP (access point) 协作场景下边缘计算卸载和资源分配方案. 首先, 通过遗传算法确定用户的任务卸载决策. 随后, 利用匈牙利算法为进行任务卸载的用户分配通信资源. 最后, 根据任务处理时延限制, 为进行任务卸载的用户分配边缘服务器计算资源, 使其满足任务处理时延限制要求. 仿真结果表明, 所提出的任务卸载与资源分配方案能够在满足任务处理时延限制的前提下有效降低移动设备的能耗.

关键词: 移动边缘计算; Wi-Fi; 多 AP 协作; 资源分配; 遗传算法; 匈牙利算法

引用格式: 韩启福,方旭明.Wi-Fi 网络下多 AP 协作边缘计算资源分配策略.计算机系统应用,2022,31(11):309–319. http://www.c-s-a.org.cn/1003-3254/8792.html

Resource Allocation Schemes of Edge Computing in Wi-Fi Network Supporting Multi-AP Coordination

HAN Qi-Fu, FANG Xu-Ming

(School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China)

Abstract: In recent years, compute-intensive and time delay-sensitive applications such as AR/VR, online games, and 4K/8K ultra-high-resolution videos have been emerging. Due to the limitations of their hardware conditions, some mobile devices are unable to calculate such applications under the time-delay requirements, and running such applications will consume huge energy and reduce the endurance of mobile devices. To solve this problem, this study proposes an edge computing offloading and resource allocation scheme in a Wi-Fi network with the coordination of multiple access points (APs). Firstly, the genetic algorithm is utilized to determine the task offloading decision of users. Then, the Hungarian algorithm is used to allocate communication resources to users with task offloading. Finally, according to the time-delay limit of task processing, the computing resources of mobile edge computing (MEC) servers are allocated to the users with task offloading. The simulations reveal that the proposed task offloading and resource allocation scheme can effectively reduce the energy consumption of mobile devices on the premise of meeting the time-delay limit of task processing.

Key words: mobile edge computing (MEC); Wi-Fi; multi-AP coordination; resource allocation; genetic algorithm; Hungarian algorithm

1　引言

近年来, 日益丰富化、智能化的移动设备应用也不断涌现, 例如虚拟现实、增强现实、在线游戏、4K/8K 超高清视频等等. 此类应用通常具有计算数据量大、时延敏感等特点. 尽管新的智能移动设备具备的计算能力越来越强大, 但是受自身硬件条件的限制,

① 基金项目: 四川省应用基础研究重点项目 (2020YJ0218)
收稿时间: 2022-02-24; 修改时间: 2022-03-28; 采用时间: 2022-04-12; csa 在线出版时间: 2022-07-07

仍然无法在短时间内处理此类计算密集型应用, 影响用户的使用体验[1]. 同时, 运行此类应用会带来巨大的电量消耗, 从而降低移动设备的续航能力.

为了解决上述问题, 移动边缘计算 (mobile edge computing, MEC) 技术应运而生[2]. MEC 技术通过在用户附近部署具有强大计算、存储能力的 MEC 服务器, 使用户可以将移动设备本地无法处理的计算任务卸载到 MEC 服务器上计算, 借助服务器强大的计算能力在短时间内完成任务计算, 再将计算结果回传给移动设备, 从而有效减少任务处理时延并降低移动设备能耗.

与此同时, 随着无线局域网的快速发展, Wi-Fi 成为社会数字化的关键基础网络设备, 承载了 60% 的移动通信业务流量[3]. 而 Wi-Fi 网络中的 AP (access point) 与移动设备的相比, 可以配备更加强大的计算能力, 包括集成具有更强计算能力或者固化了 AI 算法的处理器. 同时, 与蜂窝网络相比, Wi-Fi 网络具有大带宽、低时延、高速率以及无需考虑运营商资费问题等优势. 因此, 研究将 MEC 技术应用到 Wi-Fi 网络中, 以克服移动设备计算、能耗以及存储方面的限制, 是未来通信、计算、存储一体化发展的一个重要趋势.

针对边缘计算中的任务卸载和资源分配, 许多学者做了相关研究. 现有研究中根据研究场景进行分类, 可分为单 MEC 服务器场景[4] 与多 MEC 服务器场景. 而多 MEC 服务器场景又可分为不支持基站 (base station, BS) 间协作[5] 与支持 BS 间协作[6] 两类. 在单 MEC 服务器场景中, 只存在一个 BS, 且在该 BS 中部署了一个 MEC 服务器. 在多 MEC 服务器场景中, 存在多个 BS, 且每个 BS 中部署了一个 MEC 服务器. 在进行数据传输时, 不同 BS 间可能复用相同的频谱资源, 从而产生干扰, 影响数据传输速率. 而支不支持 BS 间协作的区别在于, 若支持 BS 间协作, 一个 BS 可通过 BS 间通信链路将无法处理的任务进一步卸载到有剩余计算资源的 BS 中进行计算.

根据研究方向进行分类, 大致可分为任务卸载策略算法研究、任务卸载策略与资源分配联合优化算法研究以及 MEC 网络架构设计[7] 3 类. 文献 [8] 中, 针对蜂窝网络单 MEC 服务器场景, 以任务执行时间限制下最小化能耗作为优化目标, 利用非协作博弈算法解决了任务卸载决策的问题, 但并未考虑通信资源分配的问题. 文献 [9] 则针对蜂窝网络中 MEC 任务卸载决策、通信和计算资源分配联合优化问题, 以最小化用户任务处理时延与能耗加权值作为优化目标, 将 Q 学习和凸优化理论整合到一起, 压缩动作空间的维度, 降低计算复杂度.

根据优化目标进行分类, 大致可分为最小化时延、最小化能耗、权衡时延和能耗 3 类. 文献 [10] 以在满足能耗限制的条件下最小化时延作为优化目标, 将 SDN 网络与 MEC 结合, 利用 Q 学习和协作 Q 学习的强化学习算法联合优化计算任务卸载和资源分配. 文献 [11] 以在时延限制条件下最小化移动设备能耗作为优化目标, 联合优化蜂窝网络下 MEC 卸载策略、通信和计算资源分配, 将优化问题建模为混合整数非线性规划问题, 利用变量融合的分支限界算法求解.

综上所述, 尽管当前业界对 MEC 系统下任务计算卸载策略和资源分配问题已经有了多方面的研究, 但大多针对蜂窝网或车联网, 而 Wi-Fi 网络下的 MEC 系统研究几乎没有. 而 Wi-Fi 网络与蜂窝网、车联网之间在网络资源特征、设备移动性等方面存在较大差异. 例如, 在蜂窝网络中, 基站在为用户分配通信资源时, 以固定大小的 RB (resource block) 作为单位, 一个用户可分配得到多个 RB; 而在 Wi-Fi 网络中, AP 在为用户分配通信资源时, 以 RU (resource unit) 作为单位, 一个用户只能分配得到一个 RU. 所以目前针对蜂窝网以及车联网的 MEC 方案并不完全适用于 Wi-Fi 网络. 同时, 现有大多数研究中仅对任务卸载决策以及计算资源分配进行研究, 而并未考虑移动设备在进行任务卸载时所使用的通信资源分配. 但系统中通信资源也是有限的, 如果移动设备分配得到的通信资源较少, 且卸载任务量较大时, 也会产生较大的数据传输时延. 由此可见, 在 MEC 系统中, 通信资源与计算资源互为瓶颈. 基于此, 本文对 Wi-Fi 网络中的 MEC 系统进行研究, 针对多 MEC 服务器多 AP 协作场景, 提出了一种任务卸载和资源分配方案, 通过对任务卸载决策、通信资源分配以及计算资源分配的联合优化, 实现在满足任务处理时延限制的前提下, 最小化移动设备端能耗的目标.

2 系统模型

假设多 MEC 服务器多 AP 协作场景中存在的多个 AP, 并且网络中的 AP 通过某种方式形成了相应的多 AP 协作集, 本文针对一个多 AP 协作集下的计算卸载与资源分配问题进行研究. 如图 1 所示, 一个多 AP 协作集中包含了一个主 AP, 其余为从 AP. 每个

AP 关联着多个 STA (station), 且每个 AP 部署了一个 MEC 服务器. 同时, AP 可以通过 AP 间的高速通信链路与其他 AP 进行信息的交互[12]. 因此, 在进行卸载决策时, STA 除了可以将任务卸载到关联 AP 的 MEC 服务器上计算之外, 还可以通过 AP 间协作, 将无法处理的任务进一步卸载到有剩余计算资源的邻居服务器上计算.

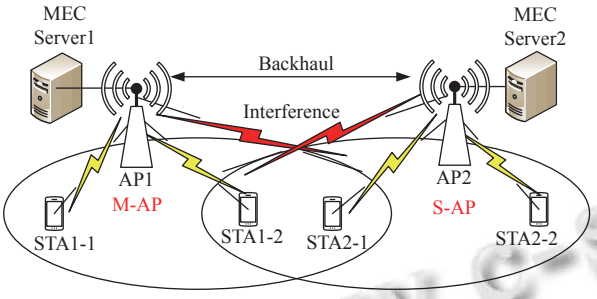

图 1 多 AP 协作集

Wi-Fi 网络下的 MEC 任务卸载流程可分为以下 4 个阶段.

(1) 信息交互及算法决策阶段: 各 AP 收集关联 STA 的计算资源、任务量大小与任务时延限制等信息. 随后各从 AP 将收集到的本基本服务集 (basic service set, BSS) 内相关信息上报给主 AP. 主 AP 在收集到各个从 AP 的 BSS 信息后, 根据本文所设计的算法, 对所有 AP 的计算卸载、通信资源分配和计算资源分配进行统一决策. 随后, 主 AP 将决策结果告知各个从 AP.

(2) 任务卸载阶段: 根据上一阶段获得的任务卸载决策与通信资源分配结果, STA 首先将计算任务卸载到关联 AP 上, 如果需要执行协作卸载, 则关联 AP 再通过 AP 间的高速回传链路将任务卸载到其他 AP 的 MEC 服务器上执行.

(3) 任务计算阶段: MEC 服务器根据第一阶段执行任务卸载及资源分配联合优化算法得到的计算资源分配结果对 STA 卸载的任务进行计算.

(4) 计算结果回传阶段: MEC 服务器完成 STA 卸载任务的计算后, 将卸载任务计算结果回传给 STA.

2.1 网络模型

假设网络中存在 M 个 AP, 每个 AP 下关联了 N 个 STA, $\mathrm{STA}_{m,n}$ 表示 AP_m 下关联的第 n 个 STA, 每个 AP 部署有一个 MEC 服务器, MEC 服务器可以并行对多个任务进行计算. 假设每个 STA 都有一个计算密集、时延敏感型任务 $CT_{m,n}=\{D_{m,n},X_{m,n},\tau_{m,n}\}$ 需要计算, 且任务不可分割, 其中, $D_{m,n}$ 表示任务的数据量, $X_{m,n}$ 表示任务的计算量, 即计算该任务所需要的 CPU 周期, 本文中假设任务在 AP 和 STA 上计算单位相同, 通常 $X_n=g\times D_n$, 其中 g 表示任务的平均计算密度, $\tau_{m,n}$ 表示任务处理能容忍的最大时延, 即时延限制. 定义 $a_{m,n}\in\{0,1,2,\cdots,M\}$ 为 $\mathrm{STA}_{m,n}$ 的决策, 其中, $a_{m,n}=0$ 表示 $\mathrm{STA}_{m,n}$ 的计算任务在本地执行, $a_{m,n}=k$ 表示将 $\mathrm{STA}_{m,n}$ 的计算任务卸载到 AP_k 的 MEC 服务器上执行. 定义 $h_{m,n}$ 为任务 $CT_{m,n}$ 卸载所经过的 AP 间跳数, 为了避免任务无限制地卸载, 造成网络拥塞等问题, 本文规定, AP 间协作卸载只允许卸载到相邻 AP, 即 $h_{m,n}\leqslant 1$. 所有 STA 的任务卸载决策向量表示为 $A_{M\times N}=\{a_{1,1},a_{1,2},\cdots,a_{1,N};\cdots;a_{M,1},a_{M,2},\cdots,a_{M,N}\}$.

2.2 通信传输模型

对于 802.11ax 设备, 既可以通过 CSMA/CA 方式独立竞争进行上行传输, 也可以通过上行 OFDMA 方式, 在 AP 调度下进行上行传输. 为了避免出现用户向 AP 上报任务卸载请求, 请求 AP 为其分配通信资源后, 该用户再通过 CSMA/CA 竞争信道进行上行任务卸载, 导致通信资源浪费的情况, 本研究中假设, STA 如果向 AP 上报任务卸载请求, 请求 AP 为其分配通信资源, 则该用户不再通过 CSMA/CA 的方式竞争信道进行上行任务卸载. 同时本文主要针对 STA 采用上行 OFDMA 方式的情况进行研究. 即 AP 利用剩余的通信资源通过 OFDMA 的方式为进行任务卸载的 STA 分配一个 RU, 调度 STA 在对应的 RU 上进行上行传输, 完成任务卸载. 对于既存在通信任务又存在计算任务的 STA, 本文中假设计算任务优先级高于通信任务优先级, 优先调度计算任务, 且通信任务与计算任务不进行混合调度.

在进行任务卸载时, 假设用 B_m 表示 AP_m 所使用的信道带宽, 其中包含了 S_m 个子载波. 同时, 假设 AP_m 所关联的 STA 中有 O_m 个 STA 需要进行任务卸载, 则在通信资源充足的情况下, AP_m 需要将带宽 B_m 划分 O_m 个 RU, 并分配给需要进行任务卸载的 STA 用于数据传输. 定义 $r_{m,n}$ 表示 AP_m 为 $\mathrm{STA}_{m,n}$ 分配的子载波个数, 即 RU 的大小. 在 IEEE 802.11ax 协议中规定, RU 的取值包括: (26, 52, 106, 242, 484, 996) 个子载波. 因此 $r_{m,n}\in\{0,26,52,106,242,484,996\}$, 其中 $r_{m,n}=0$ 表示 $\mathrm{STA}_{m,n}$ 的计算任务在本地执行, 不为其分配通信资源. 则包含所有 STA 的通信资源分配向量

表示为: $R_{M\times N}=\{r_{1,1},r_{1,2},\cdots,r_{1,N};\cdots;r_{M,1},r_{M,2},\cdots,r_{M,N}\}$.

根据香农公式, 任务 $CT_{m,n}$ 进行任务卸载时的最大数据传输速率为:

$$v_{m,n}=B_m\times\frac{r_{m,n}}{S_m}\times\log_2(1+SINR_{m,n}) \tag{1}$$

其中, $SINR_{m,n}$ 表示 $STA_{m,n}$ 在 $RU_{m,n}$ 上的信干噪比.

由此, 可计算出 $STA_{m,n}$ 将任务卸载到 MEC 服务器的传输时延和传输能耗分别为:

$$T_{m,n}^{\text{trans}}=\frac{D_{m,n}}{v_{m,n}} \tag{2}$$

$$E_{m,n}^{\text{trans}}=p_{m,n}\times T_{m,n}^{\text{trans}} \tag{3}$$

其中, $p_{m,n}$ 表示 $STA_{m,n}$ 的发送功率.

本文中假设 AP 间通过高速回传链路进行通信, 因此, 在进行 AP 间协作卸载时, 任务在 AP 间的传输时延可忽略不计. 同时, 根据文献 [13,14], 无线网络中下行传输速率一般远高于 STA 上传速率, 且 MEC 服务器计算完成后的回传数据大小通常远小于原始计算任务的数据大小, 因此, 在本文中, 任务计算结果回传所产生时延开销也忽略不计.

2.3 计算模型

当任务在 $STA_{m,n}$ 本地进行计算, 所产生的计算时延表示为:

$$T_{m,n}^{\text{local}}=\frac{X_{m,n}}{f_{m,n}^{\text{local}}} \tag{4}$$

其中, $f_{m,n}^{\text{local}}$表示 $STA_{m,n}$ 的计算频率.

本地计算能耗为:

$$E_{m,n}^{\text{local}}=\kappa(f_{m,n}^{\text{local}})^2X_{m,n} \tag{5}$$

其中, κ为功耗系数[15].

假设 $C_{M\times N}=\{c_{1,1},c_{1,2},\cdots,c_{1,N};\cdots;c_{M,1},c_{M,2},\cdots,c_{M,N}\}$表示 MEC 服务器分配给 STA 的计算资源的比例, $c_{m,n}\in[0,1]$. 当$c_{m,n}=0$表示 $STA_{m,n}$ 的计算任务在本地执行, 则 $STA_{m,n}$ 将任务卸载到 MEC 服务器上计算所需计算时延为:

$$T_{m,n}^{\text{server}}=\frac{X_{m,n}}{c_{m,n}\times f_m^{\text{server}}} \tag{6}$$

其中, f_m^{server}表示 AP_m 上 MEC 服务器的计算频率.

2.4 优化问题模型

定义 $STA_{m,n}$ 完成任务 $CT_{m,n}$ 处理的能耗开销为:

$$E_{m,n}=\begin{cases}E_{m,n}^{\text{local}},\ \text{if } a_{m,n}=0\\ E_{m,n}^{\text{trans}},\ \text{if } a_{m,n}\neq 0\end{cases} \tag{7}$$

同时, 定义完成任务 $CT_{m,n}$ 处理的时延开销为:

$$T_{m,n}=\begin{cases}T_{m,n}^{\text{local}},\ \text{if } a_{m,n}=0\\ T_{m,n}^{\text{trans}}+T_{m,n}^{\text{server}},\ \text{if } a_{m,n}\neq 0\end{cases} \tag{8}$$

本文主要研究问题是在多 MEC 服务器 AP 协作场景中, 在 AP 通信资源和 MEC 服务器剩余计算资源有限的条件下, 联合优化任务卸载决策、通信资源分配以及计算资源分配, 实现在满足任务处理时延限制的前提下, 最小化 STA 端能耗的目的. 问题可归纳为下列表达式:

$$P:\min_{A,R,C}\sum_{m=1}^{M}\sum_{n=1}^{N}E_{m,n} \tag{9}$$

s.t.

$$C1:\ h_{m,n}\leqslant 1,\ \forall m,\forall n \tag{10}$$

$$C2:\ \sum_{n=1}^{N}r_{m,n}\leqslant S_m,\ \forall m \tag{11}$$

$$C3:C_m^{sum}\leqslant 1,\ \forall m \tag{12}$$

$$C4:\mathrm{T}_{m,n}\leqslant\tau_{m,n},\ \forall m,\forall n \tag{13}$$

其中, $C1$ 表示 AP 间协作卸载决策中只允许在相邻 AP 间进行; $C2$ 表示各 AP 分配给关联 STA 的子载波总数不能超过该 AP 竞争到的子载波总数; $C3$ 表示各 AP 分配给 STA 的计算资源不能超过 MEC 服务器的总资源; $C4$ 表示任务需要在时延限制内完成计算.

3 边缘计算卸载和资源分配方案

为了解决第 2 节中提出的问题, 本节设计了一种基于遗传算法与匈牙利算法任务卸载及资源分配联合优化方案, 方案总体流程如图 2 所示.

3.1 任务卸载决策

当场景中 MEC 服务器数量为 M, STA 总数量为 N 时, 任务卸载决策存在 $(M+1)^N$ 种组合, 当 MEC 服务器数量或 STA 总数较多时, 卸载决策组合数也会指数级增长, 因此采用遍历的方式获得最优任务卸载决策是不现实的. 而遗传算法, 将问题的求解过程转换成类似生物进化中的染色体基因的交叉、变异等过程, 按照优胜劣汰的原理, 在每一代, 根据问题域中个体的适应度大小选择“优质”的个体, 并模拟遗传学中的交叉、变异过程, 产生出代表新的解集的种群, 逐代演化产生出越来越好的近似解. 在求解较为复杂的优化问题时, 遗传算法相对一些常规的优化算法, 通常能够较快地获得较好的优化结果. 因此, 本文选择了遗传算法

对卸载决策进行求解. 首先, 介绍遗传算法中较为重要的几个概念.

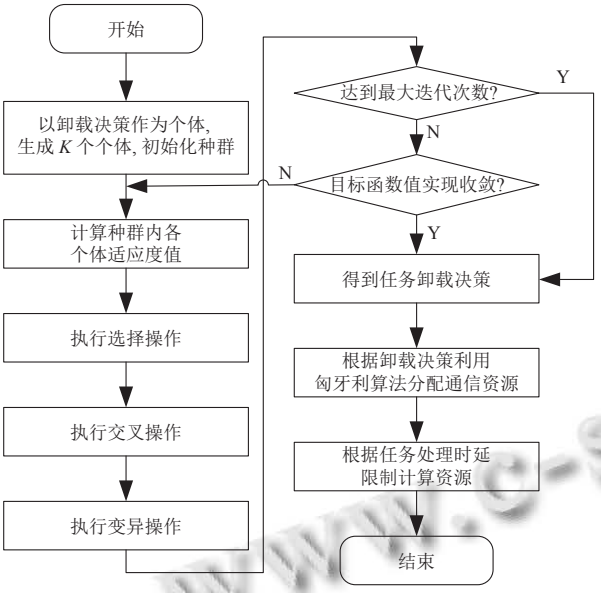

图 2 方案总体流程

个体: 优化问题中所需优化的变量, 在本文中即为卸载决策 $A_{M\times N}$, 而多个个体组成一个种群.

适应度: 每个个体都有其对应的适应度, 用于衡量个体是否“优质”, 适应度越大, 个体越“优质”. 适应度函数通常利用优化问题的目标函数进行构建, 本文的优化目标是在满足任务处理时延的限制下, 最小化 STA 总能耗, 因此将适应度函数表示为:

$$fitvalue=\begin{cases}\dfrac{1}{e_{\text{sum}}}, \text{ if } N_{OD}=0\\ \dfrac{1}{pf\times e_{\text{sum}}}, \text{ if } N_{OD}\neq 0\end{cases} \tag{14}$$

其中, e_{sum} 表示 STA 总能耗, N_{OD} 表示超出任务处理时延限制的 STA 数量, pf 表示惩罚因子, 表示为:

$$pf=\alpha\times N_{OD} \tag{15}$$

其中, α表示惩罚影响因子, α>1, α值越大, 惩罚因子对适应度值的影响越大.

选择: 根据种群中个体的适应度, 选择若一定数量的个体遗传到下一代, 适应度越大, 被选择遗传到下一代的概率越高. 本文采用保留前 5 个适应度最高的个体, 其余个体采用轮盘赌的方式进行选择.

交叉: 指的是两个父代个体的部分结构以一定的概率进行替换重组而生成新个体的操作, 本文中即对应两个卸载决策向量 $A_{M\times N}$ 中的部分结构以一定的概率进行替换重组而生成新的卸载决策向量.

变异: 个体中某个位置以很小的概率变异出新的个体. 本文中即对应卸载决策向量 $A_{M\times N}$ 中的某个 STA 的卸载决策发生了突变, 生成新的卸载决策向量.

任务卸载决策的算法流程如算法 1 所示.

算法 1. 基于遗传算法的任务卸载决策算法

```
1. pop = Initial_pop(popsize); % 初始化种群
2. While 未达到最大迭代次数且最优个体的适应度未收敛
3. fitvalue = Cal_fitness(pop); % 计算种群适应度
4. pop_sel = selection(pop, fitvalue); % 执行选择操作
5. pop_cros = crossover(pop_sel); % 执行交叉操作
6. pop_mut=mutation(pop_cros); % 执行变异操作
7. pop = pop_mutation; % 更新种群
8. loop = loop + 1; % 更新迭代次数
9. End while
10. fitvalue = Cal_fitness(pop); % 计算完成迭代后种群适应度
11. [~, index] = max(fitvalue); % 找到种群中适应度最大的个体
12. A = pop(index); % 输出任务卸载决策
```

3.2 通信资源分配

完成任务卸载决策后, 则需为进行任务卸载的 STA 分配通信资源. 而通信资源分配包括两个步骤: 一是根据执行任务卸载的 STA 个数, 确定所使用的 RU 规格组合; 二是根据确定的 RU 规格组合, 为各个进行任务卸载的 STA 分配 RU.

对于第 1 步, 在执行任务卸载的 STA 数量 N_o 确定后, 由于需要为每个 STA 分配一个 RU, 因此 AP 需要将带宽划分为 N_o 个 RU 分配, 而将一定带宽划分为 N_o 个 RU 具有不同的划分方式, 不过由于 RU 规格种类有限, 且为了减小传输时延, 应尽量将所有子载波用完, 因此将一定带宽划分为一定数量的 RU 的划分方式也有限的, 所以, 本文中采取遍历的方式, 找寻出使得 STA 传输能耗最小的 RU 规格组合.

对于第 2 步, 则是需要考虑如何将 N_o 个 RU 分配给 N_o 个 STA, 且一个 RU 只能分配给一个 STA, 一个 STA 也只能得到一个 RU, 这是一个典型的指派问题, 而匈牙利算法可以通过较低的复杂度, 快速求解出指派问题的最优解. 因此, 本文中采用了匈牙利算法求解 RU 分配问题.

要利用匈牙利算法求解 RU 分配问题, 首先需要构造一个效率矩阵 $Cost$, $Cost$ 是一个维度为 $N_o\times N_o$ 的方阵, 其中的元素 c_{ij} 表示 STA_i 在 RU_j 上传输的能耗. 同时, 利用矩阵 X 表示 RU 分配方式, X 是一个维度为 $N_o\times N_o$ 的方阵, 其中的元素 x_{ij} 表示是否将 RU_j 分配给

STA_i, 1 表示是, 0 表示否.

由于本节的优化目标为: 求解出一种 RU 分配方式, 使得所有进行任务卸载的 STA 传输总能耗最小, 因此, 可将问题归纳为下列表达式:

$$P:\ \min_{x}\sum_{i=1}^{N_o}\sum_{j=1}^{N_o}c_{ij}x_{ij} \tag{16}$$

s.t.

$$C1:\ \sum_{i=1}^{N_o}x_{ij}=1,\ j=1,2,\cdots,N_o \tag{17}$$

$$C2:\ \sum_{j=1}^{N_o}x_{ij}=1,\ i=1,2,\cdots,N_o \tag{18}$$

$$C3:\ x_{ij}=0\text{ or }1,\ i,j=1,2,\cdots,N_o \tag{19}$$

其中, $C1$ 表示一个 RU 只能分配给一个 STA. $C2$ 表示一个 STA 只能分配得到一个 RU. $C3$ 表示 RU 分配矩阵的取值范围.

在利用匈牙利算法求解上述问题前, 首先需要了解匈牙利算法中关键的一个定义与两个定理.

定义 1. 在效率矩阵 C 中, 若存在一组 0 元素, 其中任意两个 0 元素均处在不同行不同列, 则该组 0 元素称为独立 0 元素组, 其中每个元素称为独立 0 元素.

定理 1. 将效率矩阵中的某一行或列同时减去一个常数 t, 得到新的效率矩阵 C', 则以 C'为效率矩阵的新指派问题与原指派问题的最优解相同, 只是其最优值比原最优值减少了 t.

定理 2. 效率矩阵 C 中独立 0 元素的最多个数等于能覆盖效率矩阵 C 中所有 0 元素的最少直线数.

通过定理 1, 可以推导出: 若将指派问题的效率矩阵每行及每列分别减去各行及各列的最小元素, 则得到的新指派问题与原指派的最优解相同. 且生成的新效率矩阵 C'必然会出现一些零元素, 元素$c'_{i,i}=0$表示若将 RU_j 分配给 STA_i, 产生的传输能耗最低. 而利用匈牙利算法求解指派问题的本质是通过变换效率矩阵, 将效率矩阵的部分元素化为 0, 如果存在一个独立 0 元素组中 0 元素个数等于矩阵的阶数 (即等于 RU 个数), 则该组独立 0 元素组所对应的分配方式即为最优解. 而具体算法流程分为以下 4 步:

Step 1. 对效率矩阵进行初等行列变换, 每行减去该行的最小数, 每列减去该列的最小数, 得到新的效率矩阵 C', 使各行各列都出现 0 元素.

Step 2. 执行试指派. 1) 找出只有一个 0 元素的行, 将该 0 元素标记为 T, 同时将该 0 元素所在列的其他 0 元素标记为 F; 2) 找出只有一个 0 元素的列, 将该 0 元素标记为 T, 同时将该 0 元素所在行的其他 0 元素标记为 F; 3) 若存在还没有被标记的 0 元素, 则从剩余的 0 元素最少的行 (列) 开始, 选 0 元素标记为 T, 将同行同列的其它 0 元素标记为 F, 反复进行, 直到所有 0 元素均被标记, 被标记为 T 的 0 元素即为一组独立 0 元素组; 4) 若被标记为 T 的 0 元素个数等于矩阵维度 N_o, 则找到最优解, 根据 0 元素的位置, 确定 RU 分配结果, 随后结束程序; 否则, 执行 Step 3.

Step 3. 找出覆盖所有 0 元素的最少直线. 1) 标记不存在被标记为 T 的 0 元素的行; 2) 对已被标记行中存在被标记为 F 的 0 元素所在的列进行标记; 3) 对已被标记列中存在被标记为 T 的 0 元素所在的行进行标记; 4) 重复 2)–3), 直至不存在需要被标记的行或列; 5) 对没有被标记的行画横线, 对被标记的列画竖线, 能覆盖效率矩阵 C 中所有 0 元素的最少直线数 L; 6) 如 L 等于矩阵维度 N_o, 则执行 Step 2, 重新试指派, 若 L 小于 N_o, 根据定理 2 可知, 效率矩阵 C'中独立 0 元素的最多个数小于 N_o, 这种情况下, 无法求解出问题的最优解, 需要执行 Step 4.

Step 4. 增加 0 元素. 寻找效率矩阵 C'中未被直线覆盖部分的最小元素, 对未被划线的行减去最小元素, 并对被划线的列加上最小元素, 随后执行 Step 2.

通信资源分配的算法流程如算法 2 所示.

算法 2. 基于匈牙利算法的通信资源分配算法

```
N_off=length(find(A~=0)); % 统计执行卸载的 STA 数量
If N_off > N_RU % 如果卸载 STA 数大于最大可划分的 RU 数量
将卸载 STA 按时延限制升序排序
取前 N_RU 个 STA 执行任务卸载, 其余 STA 回退到本地执行
更新卸载决策 A 与卸载 STA 数 N_off
End if
RU_group=get_RU_group(B, N_off);% 根据卸载 STA 数量及带宽获取 RU 规格组合
N_group = size(RU_group, 1); % RU 规格组合数
For i = 1:N_group
Cost = Cal_cost(N_group(i, :)); :% 计算 RU 规格组合 i 的效率矩阵
[X{i}, E(i)] = Hungarian(Cost); % 利用匈牙利算法得到 RU 规格组合 i 的分配结果以及对应的 STA 传输总能耗
End for
[~, index] = min(E); % 找出使 STA 传输总能耗最小的 RU 规格组合
N = RU _group(index, :);% 得到 RU 分配结果
```

3.3 计算资源分配

假设任务处理时延恰好等于时延限制, 即式 (13)

左右两边相等时, 可推导出计算资源分配比例:

$$c_{m,n}=\frac{X_{m,n}}{\left(\tau_{m,n}-\dfrac{S_m\times D_{m,n}}{r_{m,n}\times B_m\times\log_2(1+SINR_{m,n})\times f_m^{\text{server}}}\right)} \tag{20}$$

计算资源分配的算法流程如算法 3 所示.

算法 3. 计算资源分配算法

1. c_{sum}=0; % 初始化已分配的计算资源比例
2. For i = 1:N % 依次为 STA 分配计算资源
3. If r_i == 0 % 如果 STA 本地计算
4. c_i = 0;
5. else % 如果为 STA 分配了通信资源
6. 利用式 (20) 计算 c_{temp}
7. If 1 – c_{sum} < c_{temp} % 如果剩余计算资源不足
8. r_i = 0; c_i = 0; % STA 回退到本地计算
9. else % 存在剩余计算资源
10. c_i = c_{temp}; % 分配计算资源
11. c_{sum} = c_{sum} + c_i % 更新已分配计算资源
12. End if
13. End if

4 仿真结果及分析

4.1 仿真场景与参数设置

为了验证本文所提出的基于遗传算法与匈牙利算法任务卸载及资源分配联合优化算法对能耗、时延性能的影响, 本文利用 Matlab 对其进行了仿真分析. 同时, 为了对比所提方案的性能提升, 本文还提出了 4 种基线方案. 具体如下.

基线 1: 全部本地执行, 即所有 STA 均不进行任务卸载, 而是在本地执行.

基线 2: 全部卸载到 MEC 服务器执行, 即所有 STA 将任务卸载到 MEC 服务器上执行, 且各个 STA 均分 AP 的通信资源与 MEC 服务器的计算资源.

基线 3: 采用了在资源管理领域比较常用的匈牙利算法[16] 进行通信资源分配, 随机选择 STA 将任务卸载到 MEC 服务器执行, 且利用匈牙利算法为执行任务卸载的 STA 分配通信资源, 并为采用均分的方式分配计算资源, 其余 STA 任务在本地执行.

基线 4: 利用在求解 MEC 卸载决策领域比较常用的遗传算法[17] 设置了基线 4, 通过遗传算法决定各 STA 的卸载决策, 对于卸载到 MEC 服务器执行的 STA, 采用均分的方式分配通信资源与计算资源, 其余 STA 任务在本地执行.

4 种基线方案与本文所设计方案所采用的卸载决策、通信资源分配、计算资源分配方式如表 1 所示.

表 1 方案比较

分配方式	基线1	基线2	基线3	基线4	本文方案
卸载决策	全部本地计算	全部卸载计算	随机卸载	遗传算法	遗传算法
通信资源分配	无	均分	匈牙利算法	均分	匈牙利算法
计算资源分配	无	均分	均分	均分	根据时延限制计算

仿真与算法参数如表 2 所示.

表 2 仿真与算法参数设置

仿真参数	设定值
总带宽	80 MHz
协作集AP数量	2个
每个AP关联的STA数	9个
STA发送功率	15 dBm
AP发送功率	23 dBm
工作频率f	5.8 GHz
STA剩余计算能力	0.5 GHz
AP剩余计算能力	5 GHz
任务平均计算密度	1 800 cycle/bit
功率系数κ	10^{-26}
遗传算法种群大小	200
交叉率	60%
变异率	1%
惩罚影响因子α	5

4.2 仿真结果分析

本文在同一网络场景下, 分别对 4 种基线方案与本文所设计方案的性能进行了仿真, 对比了用不同方案情况下, 能耗、时延性能与任务量大小之间的关系, 对于同一任务量大小, 采用了进行 200 次实验, 然后将 200 次实验结果取平均值的方式得到最终性能结果.

图 3 比较了采用不同方案情况下, STA 总能耗与任务量大小之间的关系 (图中横坐标的范围表示各 STA 从该范围中随机选择一个数作为任务量大小).

可以看出, 随着 STA 任务量的增加, STA 总能耗也随之增加, 但对于同一任务量大小, 相比于 4 种基线方案, 采用所设计的基于遗传算法与匈牙利算法任务卸载及资源分配联合优化方案所消耗的 STA 总能耗最小. 其中, 当任务量大小范围为 [450, 500] Kb 时, 与基线 1 相比, 方案的 STA 总能耗下降了 58%, 这是因为基线 1 中所有任务由 STA 本地执行, 会产生巨大的计算能耗. 与基线 2 相比, 方案的任务处理总能耗下降

了 44%, 这是因为系统中通信资源是有限的, 当 STA 数量较多时, 若全部 STA 均将任务卸载到 MEC 服务器上计算, 则每个 STA 分配得到的通信资源较少, 导致传输时延增大, 从而产生较大的传输能耗. 由此可以看出, 相比于基线 1 和基线 2, 合理地将部分 STA 的任务卸载 MEC 服务器上计算能够有效降低 STA 的任务处理能耗. 与基线 3 相比, 方案的 STA 总能耗下降了 36%, 这是因为在进行通信资源分配时, 虽然基线 3 与方案都采用了匈牙利算法, 但在进行卸载决策时, 基线 3 采用的是随机卸载, 并没有根据 STA 的任务量与时延限制等因素合理地安排卸载决策, 而本文所设计的方案通过遗传算法对可以满足时延限制前提下最小化能耗的卸载决策进行搜索, 从而实现更低的能耗, 这也验证了方案中采取遗传算法进行卸载决策对降低任务处理能耗的增益. 类似地, 与基线 4 相比, 方案的 STA 总能耗下降了 21%, 这是因为在进行通卸载决策时, 虽然基线 4 与方案都采用了遗传算法, 但在进行通信资源时, 基线 4 采用的是均分通信资源, 并没有根据 STA 的任务量与信道条件合理地进行通信资源分配, 而本文所设计的方案通过匈牙利算法, 对最小化 STA 传输能耗的通信资源分配方式进行求解, 从而实现更低的能耗, 这也验证了方案中采取匈牙利算法进行通信资源分配对降低任务处理能耗的增益.

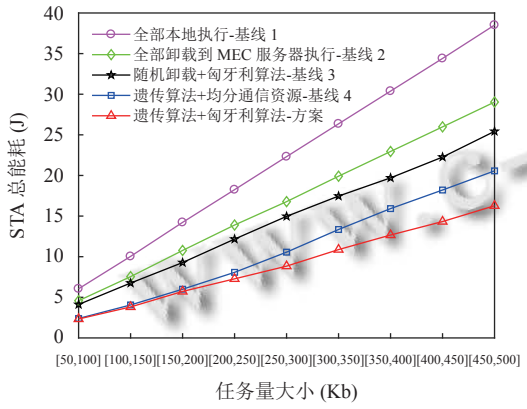

图 3 STA 总能耗与任务量关系

图 4 比较了采用不同方案情况下, STA 在时延限制下完成任务计算的比例与任务量大小之间的关系.

可以看出, 当任务量较小时, 所有方案时延限制下完成任务计算的比例均能保持在一个较高的水平, 但随着任务量的增大, 逐渐超出了 STA 或 MEC 服务器的处理能力, 时延限制下完成任务计算的比例会随之下降. 其中基线 1 下降速度最快, 这是因为 STA 算力较弱, 当任务量范围达到 [150, 200] Kb 时, 已经超出了 STA 的处理能力范围, 导致部分 STA 无法在时延限制下完成任务计算. 而基线 2、基线 3、基线 4 都采用均分的方式分配计算资源, 未考虑到 STA 的任务量与时延限制等因素, 因此时延限制下完成任务计算的比例均低于本文所设计方案. 而采取本文所设计的方案, 在任务量范围达到 [450, 500] Kb, 时延限制下完成任务计算的比例依然能保持在 90% 以上. 这是因为在利用遗传算法对卸载决策进行搜索时, 将超出任务处理时延的 STA 数量作为适应度函数的惩罚因子, 因此在搜索卸载决策时, 除了考虑 STA 总能耗外, 会优先选择超出任务处理时延的 STA 数量小的卸载决策.

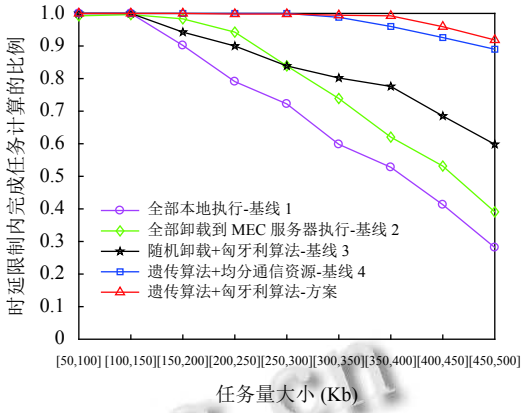

图 4 时延限制下完成任务计算的比例与任务量关系

图 5 比较了支持多 AP 协作与不支持多 AP 协作两种情况下, 任务处理能耗与 AP1 所关联 STA 任务量大小之间的关系.

可以看出, 当不支持多 AP 协作时, AP1 与 AP2 独立完成所关联 STA 任务的处理, 因此, 随着 AP1 所关联 STA 任务量的增大, AP1 所关联 STA 的任务处理总能耗不断增大, 而由于仿真场景中假设 AP2 所关联的 STA 任务量始终保持在一个较低的水平, 因此, AP2 所关联 STA 的任务处理总能耗并不随 AP1 所关联 STA 任务量的变化而变化. 而在支持多 AP 协作情况下, 随着 AP1 所关联 STA 任务量的增大, 逐渐超出了 AP1 所关联 STA 或 MEC 服务器的处理能力, AP1 通过协作将部分 STA 的任务进一步卸载到有剩余计算资源的 AP2 上进行处理. 因此所产生的能耗要小于不支持多 AP 协作情况下所产生的能耗. 而由于 AP2 需

要将部分计算资源分配给来自 AP1 的任务, 因此 AP2 所关联的部分 STA 在保证其能在时延限制内完成任务计算的情况下, 选择将任务本地执行, 所以 AP2 所关联 STA 的任务处理总能耗会随 AP1 所关联 STA 任务量增大而增大. 然而当 AP2 的计算任务也趋近于饱和, 不存在剩余计算资源时, AP2 也就不会再接受来自 AP1 的任务卸载, 因此 AP2 所关联 STA 的任务处理总能耗也就不再随之变化.

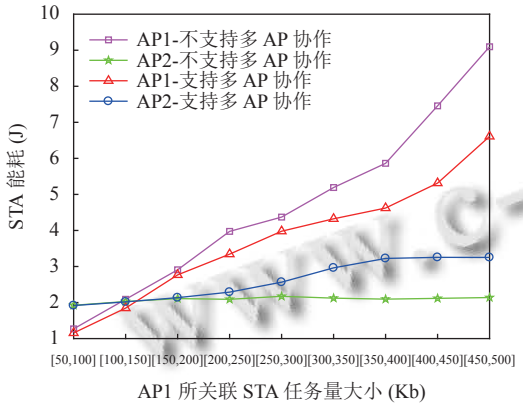

图 5 支持/不支持多 AP 协作场景下任务处理总能耗与任务量关系

图 6 比较了支持多 AP 协作与不支持多 AP 协作两种情况下, STA 在时延限制内完成任务计算的比例与 AP1 所关联 STA 任务量大小之间的关系.

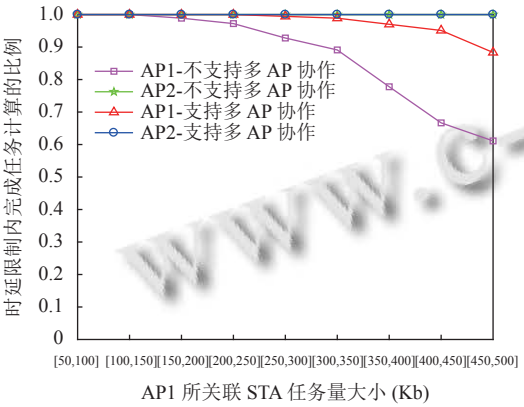

图 6 支持/不支持多 AP 协作场景下在时延限制下完成任务计算的比例与任务量关系

可以看出, 在不支持多 AP 协作情况下, 随着 AP1 所关联 STA 任务量的增大, 逐渐超出了 AP1 所关联 STA 或 MEC 服务器的处理能力, 因此在时延限制内完成任务计算的比例不断降低. 而由于 AP2 所关联的 STA 任务量较小, AP2 所关联 STA 在时延限制内完成任务计算的比例始终保持在 100%. 在支持多 AP 协作情况下, 当任务量超出 AP1 所关联 STA 与 MEC 服务器的处理能力时, AP1 通过协作将部分 STA 的任务进一步卸载到有剩余计算资源的 AP2 上进行处理, 因此在时延限制内完成任务计算的比例要高于不支持多 AP 协作的场景. 而 AP2 虽然将部分计算资源分配给来自 AP1 的任务, 但分配的前提是 AP2 拥有剩余计算资源, 因此并不影响 AP2 所关联 STA 在时延限制内完成任务计算的比例, 始终保持在 100%.

图 7 展示了采用所设计方案情况下, STA 总能耗随迭代次数的变化.

本文优化问题的决策变量包含了卸载决策向量、通信资源分配向量以及计算资源分配向量, 本文将上述 3 个决策变量转换为 3 个子问题进行求解. 对于计算资源分配向量, 是根据任务处理时延限制与通信资源分配结果利用公式直接计算获得, 求解复杂度可以忽略不计. 对于通信资源分配向量, 本文将其转换为一个匹配问题, 并利用匈牙利算法进行求解, 将算法时间复杂度从 $O(n!)$ 降低为 $O(n^3)$[18]. 而对于卸载决策向量的求解, 则是利用遗传算法进行求解, 求解结果的好坏与遗传算法中种群规模与迭代次数等因素有关, 种群规模越大、迭代次数越多, 越接近最优解. 而通过图 7 可看出, 随着迭代次数的增大, STA 总能耗逐渐减小后趋于平稳, 在经过 10–15 次左右的迭代后即可实现收敛, 且每次收敛时, 所得到的 STA 总能耗指标均比较稳定, 维持在 3.4 J 左右. 说明所设计方案能在较少的迭代次数内实现收敛, 通过较低的算法复杂度得到一个较为理想和稳定的性能指标.

5 总结

本文对 Wi-Fi 网络中 MEC 卸载决策、通信资源、计算资源分配联合优化问题进行研究. 提出了一种基于遗传算法与匈牙利算法任务卸载及资源分配联合优化方案. 首先, 通过遗传算法确定用户的任务卸载决策. 随后, 利用匈牙利算法为进行任务卸载的用户分配通信资源. 最后, 根据任务处理时延限制, 为进行任务卸载的用户分配 MEC 服务器计算资源, 使其满足任务处理时延限制要求. 虽然在极端情况下, 可能会出现部分需要执行任务卸载的 STA 由于通信或计算资源不足, 而回退到本地执行的情况, 但是仿真结果表明,

所提出的方案能够显著提高系统在时延限制下完成任务计算的比例, 并且有效降低系统中 STA 总能耗. 在后续工作中, 也会进一步考虑网络中计算任务与通信任务联合调度的问题.

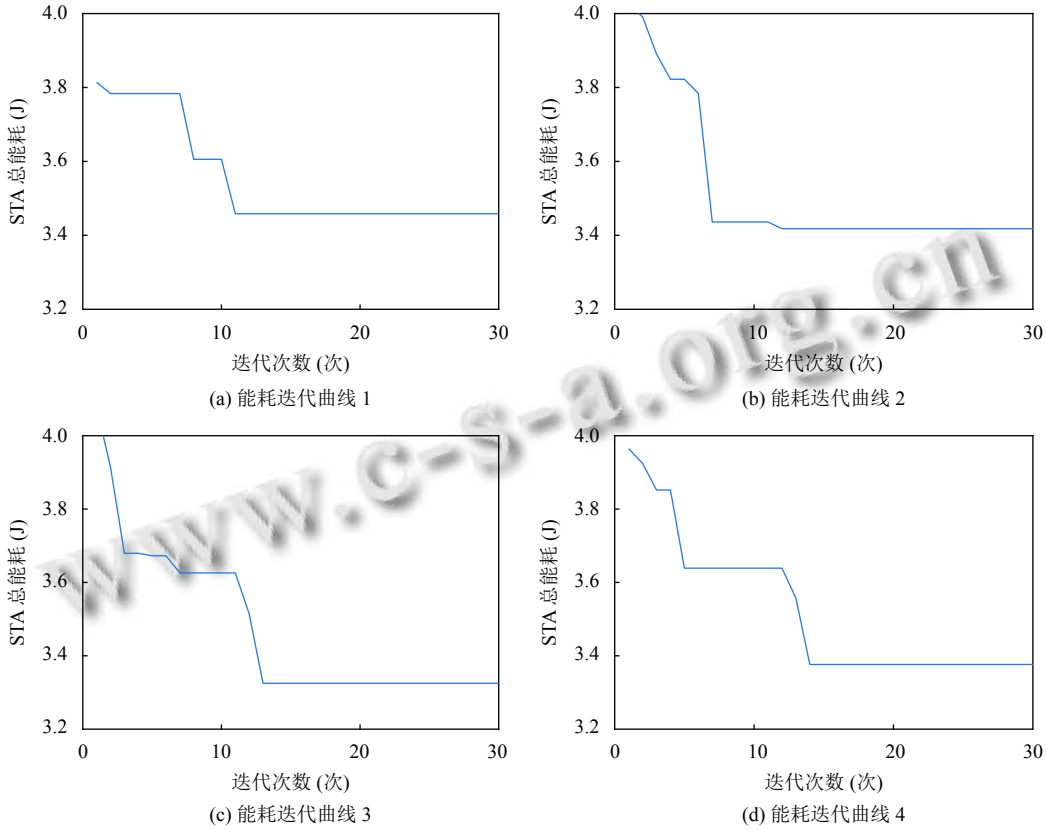

图 7 STA 总能耗与迭代次数的关系

参考文献

1 Wang YT, Chen IR, Wang DC. A survey of mobile cloud computing applications: Perspectives and challenges. Wireless Personal Communications, 2015, 80(4): 1607–1623. [doi: 10.1007/s11277-014-2102-7]

2 Mach P, Becvar Z. Mobile edge computing: A survey on architecture and computation offloading. IEEE Communications Surveys & Tutorials, 2017, 19(3): 1628–1656.

3 Bennis M, Simsek M, Czylwik A, *et al*. When cellular meets WiFi in wireless small cell networks. IEEE Communications Magazine, 2013, 51(6): 44–50. [doi: 10.1109/MCOM.2013.6525594]

4 Yan J, Bi SZ, Zhang YJ, *et al*. Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency. IEEE Transactions on Wireless Communications, 2020, 19(1): 235–250. [doi: 10.1109/TWC.2019.2943563]

5 Yang XT, Yu XY, Huang H, *et al*. Energy efficiency based joint computation offloading and resource allocation in multi-access MEC systems. IEEE Access, 2019, 7: 117054–117062. [doi: 10.1109/ACCESS.2019.2936435]

6 王忍, 王翊, 胡艳军, 等. 超密集异构网络中过载 MEC 服务器的协作卸载. 西安电子科技大学学报, 2020, 47(2): 126–134.

7 Passas V, Makris N, Nanis C, *et al*. MEC service placement over the Fronthaul of 5G Cloud-RANs. Proceedings of 2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN). Paris: IEEE, 2019. 1–2.

8 邓茂菲. 基于移动边缘计算的任务迁移策略研究 [硕士学位论文]. 北京: 北京邮电大学, 2017.

9 Wang DF, Zhao J. Computation offloading and resource allocation in mobile edge computing via reinforcement learning. Proceedings of 2019 11th International Conference on Wireless Communications and Signal Processing (WCSP). Xi'an: IEEE, 2019. 1–6.

10 Kiran N, Pan CY, Wang SH, *et al*. Joint resource allocation

and computation offloading in mobile edge computing for SDN based wireless networks. Journal of Communications and Networks, 2020, 22(1): 1–11.

11 Yang XT, Yu XY, Rao AQ. Efficient energy joint computation offloading and resource optimization in multi-access MEC systems. Proceedings of 2019 IEEE 2nd International Conference on Electronic Information and Communication Technology (ICEICT). Harbin: IEEE, 2019. 151–155.

12 Schelstraete S, Latif I, Dash D, *et al*. Multi-AP backhaul analysis. https://mentor.ieee.org/802.11/dcn/19/11-19-1588-00-00be-multi-ap-backhaul-analysis.pptx. (2019-09-13).

13 Zhao PT, Tian H, Qin C, *et al*. Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing. IEEE Access, 2017, 5: 11255–11268. [doi: 10.1109/ACCESS.2017.2710056]

14 Wang CM, Yu FR, Liang CC, *et al*. Joint computation offloading and interference management in wireless cellular networks with mobile edge computing. IEEE Transactions on Vehicular Technology, 2017, 66(8): 7432–7445. [doi: 10.1109/TVT.2017.2672701]

15 Mao YY, You CS, Zhang J, *et al*. A survey on mobile edge computing: The communication perspective. arXiv: 1701.01090, 2017.

16 张海波, 李虎, 陈善学, 等. 超密集网络中基于移动边缘计算的任务卸载和资源优化. 电子与信息学报, 2019, 41(5): 1194–1201. [doi: 10.11999/JEIT180592]

17 Li HL, Xu HT, Zhou CC, *et al*. Joint optimization strategy of computation offloading and resource allocation in multi-access edge computing environment. IEEE Transactions on Vehicular Technology, 2020, 69(9): 10214–10226. [doi: 10.1109/TVT.2020.3003898]

18 SimyHsu. Hungarian Algorithm 匈牙利算法. https://blog.csdn.net/u014754127/article/details/78086014. (2017-09-25).

(校对责编: 孙君艳)