文章编号:2095-6134(2022)02-0267-08

移动边缘计算中的在线任务卸载方法*

刘婷[1,2,3†],罗喜良[1]

(1 上海科技大学 信息科学与技术学院, 上海 201210; 2 中国科学院上海微系统与信息技术研究所, 上海 200050;
3 中国科学院大学, 北京 100049)

(2020 年 1 月 15 日收稿; 2020 年 5 月 5 日收修改稿)

Liu T, Luo X L. Online task offloading in mobile edge computing[J]. Journal of University of Chinese Academy of Sciences, 2022, 39(2): 267-274. DOI: 10.7523/j.ucas.2020.0012.

摘 要 为减少移动边缘计算(mobile edge computing, MEC)网络中移动用户的长期任务开销,利用强化学习的马尔科夫决策过程,将用户的移动性与系统的动态信息建模为随机优化问题。依据系统信息的状态,将问题分为系统信息已知、系统信息未知 2 种情况。在系统信息已知时,提供了问题的最优解;系统信息未知时,基于在线学习提出 2 个任务卸载策略。一个策略能够收敛到系统最优解,但收敛速度较慢;另一个策略能以更快的收敛速度,达到接近最优解的表现,可用于更复杂的系统。最后在仿真中展示算法的有效性。

关键词 移动边缘计算;任务卸载;在线学习;马尔科夫决策过程

中图分类号:TP393 **文献标志码**:A **DOI**:10.7523/j.ucas.2020.0012

Online task offloading in mobile edge computing

LIU Ting[1,2,3], LUO Xiliang[1]

(1 *School of Information Science and Technology, ShanghaiTech University, Shanghai* 201210, *China*;
2 *Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences, Shanghai* 200050, *China*;
3 *University of Chinese Academy of Sciences, Beijing* 100049, *China*)

Abstract To reduce the costs of task processing, task offloading is put forward as a promising technology in mobile edge computing. In this paper, in order to lessen the burden of long-term tasks of the moving user, we utilize Markov decision process to formulate the task offloading problem as a stochastic programming problem, when taking the user mobility and system dynamics into account. According to the system information state, the problem can be categorized as the one with fully known system information and the other one with limited system information. We provide the optimal and learning-based task offloading algorithms under these two kinds of systems respectively. Furthermore, two learning-based algorithms, one with optimality and another with faster convergence rate, are proposed. The performance is verified in simulations.

Keywords mobile edge computing (MEC); task offloading; online learning; Markov decision process (MDP)

* 国家自然科学基金面上项目(61971286)资助

† 通信作者,E-mail:liuting@shanghaitech.edu.cn

随着物联网时代信息的快速增长,用户对数据处理速率、服务质量(quality of service, QoS)与体验质量(quality of experience, QoE)的要求在不断提高。然而大批提升用户体验的智能应用服务,如增强现实(augmented reality, AR)、虚拟现实(virtual reality, VR)、交互游戏等往往伴随着高计算复杂度和低时延的要求。因此即使移动设备的处理能力在不断提高,但智能手机作为便携性设备,由于其固有的缺点,如有限的计算资源、存储资源,仍然无法满足此类任务的要求。移动边缘计算能够有效平衡设备能力和用户体验的困境,同时,移动边缘计算的安全性能够得到保障,如文献[1]中作者从服务器安全、用户隐私等多方面来保证系统的稳定。因此移动边缘计算技术被广泛研究。在移动边缘计算(mobile edge computing, MEC)网络中[2-4],将高计算复杂度的任务卸载至网络边缘端,利用分布式的计算资源和存储资源,能够有效减少任务的处理时延。因此,如何实现更高效的任务卸载吸引了大量学者的关注。

在许多文献中,任务卸载被建模为确定性优化问题。You 等[5]在任务计算时延的约束下,通过任务卸载实现最小化能量消耗,并将该任务卸载问题定义为确定性优化问题。然而一个实用的任务卸载策略应该能够根据系统的实时状态进行自主调整,例如用户设备的队列信息、帮助节点的计算能力等。基于该考虑,Mao 等[6]将任务卸载建模为随机规划问题,利用李雅普诺夫优化方法,将复杂的随机规划问题转换为一系列简单的顺序决策问题。上述文献提出的任务卸载策略都是基于系统参数信息已知的假设,但在实际场景中用户难以获得系统信息,或者需要大量开销来获取信息,因此需要一个能够自主在线学习的策略实现任务卸载。此外,在文献[5-6]中,作者只根据系统的短期利益更新任务卸载策略,而忽略了系统的长远利益。本文将针对这 2 个因素来建立任务卸载模型。

强化学习作为一种在线学习方法,能够从系统历史反馈中学习系统信息,从而处理系统的未知性。近期许多文献利用强化学习技术在任务卸载方面取得进展。Chen 等[7]利用强化学习得到更优的任务卸载策略,从而实现系统长期效用的最大化。Min 等[8]将能量收集技术应用到 MEC 网络,通过强化学习来选择卸载节点和速率以提高用户体验。Huang 等[9]将无线供电 MEC 网络的任务卸载建模为组合优化问题,利用强化学习得到近似最优解。然而,上述文献都是基于静止的用户设备,忽视了移动用户的需求,无法应用于移动场景,如现在的车联网(vehicle-to-everything, V2X)应用、AR 的场景导览、移动机器人[10]等,因此针对于移动用户的任务卸载的研究具有重要意义。

相较于移动的用户,静止用户所处的 MEC 网络环境中的信道环境、周围节点的拓扑结构等比较稳定,这也是之前大部分研究任务卸载的文献中考虑的场景[4-8]。而相比于传统有线网络,如今很多使用无线网络、蜂窝网络的用户通常处于移动状态,因此不仅用户周围的帮助节点会随着变化,信道状态也会发生改变。基于此考虑,本文参照文献[11]中 MEC 网络的拓扑结构和用户移动模式,假设用户按照马尔科夫性质在网络中移动,利用强化学习技术对任务卸载进行研究。文献[11]中,在一个蜂窝网络完全覆盖、Wi-Fi 局部覆盖的场景中,作者探究在昂贵的蜂窝网络和便宜的 Wi-Fi 下,移动用户通过 Wi-Fi 卸载来减少开销从而完成长时间的文件传输。他们将卸载模型定义为马尔科夫决策过程(Markov decision process, MDP),并在用户移动模式已知时提供系统最优解。MDP 被广泛应用于随机规划问题中,它能够有效地刻画系统的动态变化,并将系统的长期表现作为目标。

本文将移动用户的任务卸载问题建模为 MDP,并在系统信息为先验知识时,提供系统的最优解。同时,在系统信息未知,即用户移动模式未知时,通过基于 Q-learning 和 DQN 的在线学习方法,得到收敛速度快,且效果逼近最优解的算法。

符号说明:指示函数 $1\{A\}$ 表示事件 A 发生(不发生)时取值为 1(0)。$\mathbb{E}[\cdot]$ 为期望。$[x]^+=\max\{0,x\}$。$\mathcal{A}\setminus\{0\}$ 代表集合 $\mathcal{A}$ 中除 $\{0\}$ 以外的所有元素的集合。

1 系统模型

1.1 网络模型

在移动边缘计算网络中(参见图 1),存在移动用户(本地节点)和一些固定节点。固定节点可以是宏基站、微基站和家庭基站等,它们能够为用户设备提供计算资源和存储资源等,后文将其称为帮助节点。

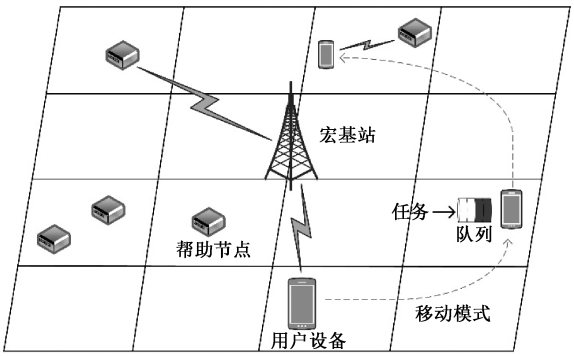

图 1　移动边缘计算的网络拓扑结构图

Fig. 1　Topology of the MEC network

参照文献[11]的场景设置,将移动边缘计算的网络场景离散化为 L 个方格(位置),每个方格用 l 表示,$l \in \mathcal{L} = \{1,2,\cdots,L\}$。在网络中存在 $(N+2)$ 个节点,用 $\mathcal{N} = \{0,1,2,\cdots,N,N+1\}$ 来表示这些节点。其中 0 和 $N+1$ 分别表示本地节点和宏基站,$\{1,2,\cdots,N\}$ 为其他的帮助节点。宏基站位于网络中心,能覆盖网络中所有位置。其他帮助节点可能存在于各个位置。本地节点按照马尔科夫性质在网络中移动。虽然在任何时刻用户都可以将任务卸载给宏基站,然而当用户离宏基站较远时,路径损耗会造成较大的传输延迟,此时若将任务卸载给本地节点周围的帮助节点可能会实现更低的时延①。

由于单一节点的计算和存储资源有限,在本地处理的任务通常会经历较高延迟,影响用户服务质量。为实现低延迟处理,本地节点可以将部分计算任务卸载到周围的帮助节点,这些帮助节点通常拥有更多的计算和存储资源,并且可以按需部署以实现更高要求。在这个任务卸载模型中,考虑 T 个时隙 $\mathcal{T} = \{1,2,\cdots T\}$,用户设备在每个时隙 t 的开始产生大小为 μ 的任务,称之为任务 t,该任务可以由本地节点计算,或者卸载给帮助节点。若在本地计算,任务将缓存在本地节点的先入先出(first-input-first-output, FIFO)队列中。若卸载到帮助节点,则本地节点与帮助节点建立通信链路,任务传输给帮助节点并在帮助节点端进行处理。为保证用户在移动到其他位置前完成传输,假设任务在单个时隙内传输完成。在单个时隙内用户只能和一个帮助节点通信。图 2 展示了任务卸载的时间线,假设任务 t 在本地节点处理,任务 $t+1$ 卸载到帮助节点。

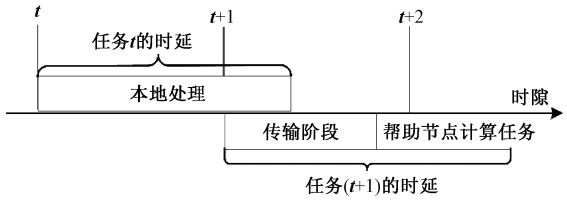

图 2　任务卸载模型时间线

Fig. 2　Timeline example of task offloading

1.2　问题建模

本文的目标是最小化移动用户的长期任务时延。为刻画用户移动性带来的系统随机性,将问题定义为一个马尔科夫决策过程,利用系统转移概率来刻画用户的移动模式。该问题可以由 4 个元素完全表征,下面将分别介绍。

1)系统状态 s:$s = (l,b)$,$s \in \mathcal{S}$。其中 $l \in \mathcal{L}$ 是用户位置(方格)的索引。$b \in \mathcal{B}$ 为本地节点的队列长度。$\mathcal{S}$ 为状态空间,包含所有可能的系统状态。定义 s_t 为时隙 t 的系统状态。由于本地节点的计算和存储资源有限,存在最大允许队列长度 B,因此 $\mathcal{B}$ 为

$$\mathcal{B} = \{0,\mu,\mu - f^0,\cdots,m\cdot\mu - n\cdot f^0,\cdots,B\}.$$

其中:m,n 为正整数,f^0 为本地节点计算能力,即每时隙处理的任务大小。当任务在本地计算时,队列会增加大小为 μ 的任务,同时本地节点以 f^0 的速度处理任务。当任务卸载到其他帮助节点时,本地节点不增加任务,同时以 f^0 的速度处理任务。

2)动作 a:$a \in \mathcal{A}$。其中 $\mathcal{A}$ 为动作空间,包含所有可能的动作,$\mathcal{A} = \{0,1,2,\cdots,N,N+1\}$. $a=0$ 表示任务在本地节点处理,否则为卸载到帮助节点 a,$a = N+1$ 表示宏基站。定义 a_t 为用户对任务 t 的卸载决策。由于用户设备的发送功率有限,因此本地节点能够通信的帮助节点也受限。同时,用户的移动性导致不同时隙内用户能够进行通信的帮助节点也不一样。当队列长度超过本地节点最大允许队列,即 $b > B$ 时,本地节点由于计算、存储资源限制,不能够再继续接收任务,此时用户设备只能够选择将任务卸载给帮助节点,即 $a \in \mathcal{A}\backslash\{0\}$。

3)转移概率 $\mathbb{P}(s' \mid s,a)$:$s,s' \in \mathcal{S}, a \in \mathcal{A}$。它

① 以图 1 为例,当用户移动至场景中的左上角时,相比于卸载至计算能力强但通信距离较远的宏基站,若卸载到离用户设备更近的帮助节点,可能会实现更低的时延。

用来刻画系统的动态变化,表示当系统处于状态 s,用户执行动作 a 后,系统转移到状态 s' 的概率。由于用户的移动模式服从马尔科夫性质,l_{t+1} 只依赖于 l_t,因此转移概率 $\mathbb{P}(s_{t+1} \mid s_t, a_t)$ 可以分解为

$$\mathbb{P}(s_{t+1} \mid s_t, a_t) = \mathbb{P}(l_{t+1}, b_{t+1} \mid l_t, b_t, a_t) = \mathbb{P}(l_{t+1} \mid l_t)\, \mathbb{P}(b_{t+1} \mid l_t, b_t, a_t), \tag{1}$$

其中 b_{t+1} 的转移概率服从

$$\mathbb{P}(b_{t+1} \mid l_t, b_t, a_t) = \begin{cases} 1, & b_{t+1} = [b_t + \mu \cdot 1\{a_t = 0\} - f^0]^+, b_{t+1} \leqslant B, \\ 0, & \text{else}, \end{cases} \tag{2}$$

$\mathbb{P}(l_{t+1} \mid l_t)$ 通过定义用户在 2 个连续时隙内,从位置 l_t 移动到 l_{t+1} 的概率,来表征用户的移动模式,可以通过历史移动模式估计得到[12-13]。

4)开销 $c(s,a)$:系统的开销定义为完成任务需要的时延,$c(s_t, a_t)$ 为完成任务 t 所需要的时延

$$c(s_t, a_t) = \frac{\mu + b}{f(l_t, a_t)} + \frac{\mu \cdot 1\{a_t \in \mathcal{A} \setminus \{0\}\}}{r(l_t, a_t)}, \tag{3}$$

等式右边的第 1 项为任务 t 的处理时延,第 2 项为传输时延,类似大多数文献的考虑[5-6],本文也忽略任务结果传输回本地节点的时间。$f(l_t, a_t)$ 为用户位于 l_t 时卸载到的节点 a_t 的计算能力,即节点 a_t 的计算速度。注意当 $a_t = 0$ 时,$f(l_t, a_t) = f^0$;当 $a_t \neq 0$ 时 $b = 0$。$r(l_t, a_t)$ 为传输速率,只有将任务卸载给帮助节点,即 $a_t \in \mathcal{A} \setminus \{0\}$ 时传输时延才非零。

为最小化移动用户的长期任务的时延,将问题定义为

$$\min_{\pi \in \Pi} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} c(s_t, \pi(s_t))\right]. \tag{4}$$

其中:π 为 $\mathcal{S}$ 到 $\mathcal{A}$ 的映射,$\pi(s_t) = a_t$,是系统状态 s_t 时的卸载决策。γ 为折扣因子,满足 $\gamma < 1$。Π 为所有可行解的集合。本文要求解能够实现最小化长期任务时延的最优策略 π^*。

2 任务卸载策略

2.1 系统已知时的最优任务卸载策略

当系统信息已知,即转移概率 $\mathbb{P}(s' \mid s, a)$ 为先验知识时,能够得到任务卸载的最优策略。定义策略 π 的状态值函数 $V_\pi(s)$ 为当系统处于状态 s 时,按照策略 π 实现的未来任务的折扣开销的期望

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=i+1}^{T} \gamma^{t-1} c(s_t, a_t) \mid s_i = s\right], \forall s \in \mathcal{S}, \tag{5}$$

其中:s_i 代表初始状态,$\mathbb{E}[\cdot]$ 为对策略 π 和转移概率 $\mathbb{P}(s' \mid s, a)$ 的期望。

最优策略 π^* 的状态值函数 $V_{\pi^*}(s)$ 称为最优状态值函数,下文简写为 $V^*(s)$。最优状态值函数满足贝尔曼最优性条件[14],以系统状态 s_t 为例:

$$V^*(s_t) = \min_{a \in \mathcal{A}} Q^*(s_t, a), \tag{6}$$

其中 $Q^*(s_t, a)$ 为状态-动作对 (s_t, a) 的动作值函数,满足

$$\begin{aligned} Q^*(s_t, a) &= \mathbb{E}[c(s_t, a) + \gamma V^*(s') \mid s_t, a] \\ &= c(s_t, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s' \mid s_t, a) V^*(s') \\ &= \frac{\mu + b}{f(l_t, a)} + \frac{\mu \cdot 1\{a \in \mathcal{A} \setminus \{0\}\}}{r(l_t, a)} + \\ &\quad \gamma \sum_{l' \in \mathcal{L}, b' \in \mathcal{B}} \mathbb{P}(l' \mid l)\, \mathbb{P}(b' \mid l_t, b_t, a) V^*(s'), \end{aligned} \tag{7}$$

其中 s' 为系统在状态 s_t 执行动作 a 后,系统可能转移的下一状态。式 (7) 中第 2 个等式的第 1 项为任务 t 的即时开销 $c(s_t, a)$,第 2 项为对未来所有任务的折扣开销的期望的估计,γ 是 1.2 节中的折扣因子,满足 $\gamma < 1$。在贝尔曼最优性方程中,即时开销的比重为 1,高于对未来开销的估计的比重 γ。

动作值函数 $Q^*(s,a)$ 的定义与状态值函数 $V^*(s)$ 相似,为系统在状态 s 执行动作 a 后,按照策略 π^* 实现的长期折扣开销的期望。从式(6)可以看出,若已知最优动作值函数 $Q^*(s,a)$,采用贪婪算法执行任务卸载,即最低动作值函数对应的动作为最优卸载决策

$$\pi^*(s) = \arg\min_{a \in \mathcal{A}} Q^*(s, a), \forall s \in \mathcal{S}. \tag{8}$$

基于以上研究,当系统信息已知时,通过数值迭代能够求解所有状态-动作对的最优动作值函数,再基于贪婪算法获得最优策略。具体流程见算法 1。

算法 1 最优任务卸载算法

输入:$\mathbb{P}(s' \mid s, a), \forall s, s' \in \mathcal{S}, a \in \mathcal{A}$.

步骤 1) 初始化策略 $\pi^*(s)$ 和状态值函数 $V^*(s), \forall s \in \mathcal{S}$;

步骤 2)　对 $\forall l \in \mathcal{L}, b \in \mathcal{B}$, 执行步骤 3)或 4);

步骤 3)　若 $b \leqslant B$, 对 $\forall a \in \mathcal{A}$, 执行步骤 5);

步骤 4)　若 $b > B$, 对 $\forall a \in \mathcal{A} \setminus \{0\}$, 执行步骤 5);

步骤 5)　根据式 (7)更新 $Q^*(s,a)$, 根据式 (8)更新 $\pi^*(s)$, 按照式(6)设置 $V^*(s)$。

步骤 6)　若 $V^*(s)$ 收敛,算法结束;否则跳转至步骤 2。

输出:　最优任务卸载策略 $\pi^*(s), \forall s \in \mathcal{S}$.

算法 1 通过离线地对贝尔曼最优性条件,式(6)和式(7),进行数值迭代,从而得到最优卸载策略 π^*。当系统需要进行在线任务卸载时,观测到状态 s 后,可以直接通过 $\pi^*(s)$ 获得最优卸载决策。

2.2　在线任务卸载算法

然而系统信息一般难以获得,虽然能够根据用户的历史移动轨迹,对用户移动模式进行预测[12-13],但会引入大量额外开销。为解决用户移动性带来的系统未知性,基于在线学习,提出能够应对用户移动性的任务卸载算法。

2.2.1　基于 Q-learning 的任务卸载

由于用户移动模式为后验知识,无法通过 2.1 节中的算法得到最优任务卸载策略,因此需要在线学习的算法来处理未知信息。Q-learning 作为一种无模型方法,可以根据系统的历史反馈,对动作值函数 $Q(s,a)$ 进行预测。

在系统状态 s_t 时执行动作 a_t 后,观测系统开销 $c(s_t,a_t)$, 同时系统转移到下一状态 s_{t+1}, 可以将这些信息称为一组经验值 $e_t = (s_t,a_t,c(s_t,a_t), s_{t+1})$。之后动作值函数 $Q(s_t,a_t)$ 利用这一组经验值,按照下式更新:

$$Q_t(s_t,a_t) = (1-\alpha_t)Q_{t-1}(s_t,a_t) + \alpha_t(c(s_t,a_t) + \min_{a\in\mathcal{A}} Q_{t-1}(s_{t+1},a)), \quad (9)$$

式中: Q_t 为动作值函数的第 t 次迭代, α_t 为第 t 次迭代的学习率,也代表此次经验值 e_t 的重要性。在每个时隙,系统利用经验值更新动作值函数,获得新的动作值函数后,在下一时隙根据新的动作值函数按照式(8)执行卸载决策。

根据文献[14-15]中的研究,若每个状态-动作对 $(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$ 都能够被访问无限多次,那么基于 Q-learning 获得的动作值函数能够以概率 1 收敛于最优值函数。为保证每个状态-动作对都能够被访问,采用 ε -greedy 方法(算法 2 的步骤 2),它是强化学习中能够有效平衡"探索"和"开发"困境的方法。用户按照一定概率,在开发(执行预估开销最小的动作)和探索(执行随机的动作)之间抉择,从而每个 (s,a) 得到多次访问。实际上,只要 Q-learning 满足以下 2 个条件,就可以保证动作值函数的收敛:

1)即时开销有界, $|c(s,a)| \leqslant C$;

2)学习率 α_t 满足随机逼近条件

$$\sum_{t=0}^{\infty}\alpha_t = \infty, \sum_{t=0}^{\infty}\alpha_t^2 < \infty. \quad (10)$$

我们的任务卸载模型符合条件 1)。同时通过对学习率的设定,如设置 $\alpha_t = 1/t$, 条件 2)也可以得到满足。因此本文的任务卸载模型能够通过 Q-learning 方法收敛到最优解。基于 Q-learning 的在线任务卸载策略的流程见算法 2。

算法 2　基于 Q-learning 的任务卸载

步骤 1)　设置 $t = 1$, 初始化动作值函数 $Q_0(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$;

步骤 2)　观测系统状态 s_t, 按照概率 ε 执行随机卸载决策;或按照 $(1-\varepsilon)$ 的概率,执行卸载决策 $a_t = \arg\min_{a\in\mathcal{A}} Q_{t-1}(s_t,a)$;

步骤 3)　观测系统开销 $c(s_t,a_t)$ 以及新状态 s_{t+1};

步骤 4)　存储经验值 $e_t = (s_t,a_t,c(s_t,a_t), s_{t+1})$;

步骤 5)　按照式 (9)更新 $Q_t(s_t,a_t)$;

步骤 6)　设置 $t = t + 1$;

步骤 7)　若 $t > T$, 算法结束;否则跳转步骤 2)。

Q-learning 是一个基于表格的策略,表格横轴为状态空间,纵轴为动作空间,表格内为所有状态-动作对的 Q 值。为达到收敛,表格中每一个数据都需要得到多次更新。但在算法 2 中,每个时隙只能更新表格中的一个数据,如果状态空间和动作空间非常大,将面临维度灾难并难以收敛。为应对这种情况,加快收敛速度,将采用基于拟合的方式,实现在单个时隙中批量更新 Q 值。

2.2.2　基于 DQN 的任务卸载

为避免对每个 $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$ 分别更新,采用深度神经网络对 Q^* 进行拟合,从而实现所有状态-动作对的更新:

$$Q(s,a;\boldsymbol{\theta}) \approx Q^*(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}, \quad (11)$$

其中：$Q(s,a;\boldsymbol{\theta})$ 为深度神经网络的输出，$\boldsymbol{\theta}$ 为神经网络的参数，将深度神经网络与 Q-learning 的结合称为 DQN[16]。

相比于 Q-learning 中每次只更新单个状态-动作对的动作值函数，DQN 中每次网络参数 $\boldsymbol{\theta}$ 的更新，使所有状态-动作对的动作值函数都能得到更新。同时，还将经验回放利用于神经网络的更新，加快 DQN 的收敛速度。首先设置容量为 P 的经验池 $\mathcal{P}_t$，用于存放历史经验值，即 $\mathcal{P}_t=\{e_i, t-P<i\leqslant t\}$，在时隙 t 更新 $Q(s,a;\boldsymbol{\theta})$ 时，从经验池 $\mathcal{P}_t$ 中随机采样 $\mathring{\mathcal{P}}_t$ 作为神经网络的训练数据。因此，该神经网络的损失函数 $L(\boldsymbol{\theta}_t)$ 定义为

$$L(\boldsymbol{\theta}_t)=\mathbb{E}_{\mathring{\mathcal{P}}_t\sim U(\mathcal{P}_t)}[(y_t-Q(s_t,a_t;\boldsymbol{\theta}_{t-1}))^2], \tag{12}$$

$$y_t=c(s_t,a_t)+\gamma\min_{a'}Q(s_{t+1},a';\boldsymbol{\theta}_{t-1}). \tag{13}$$

$\mathring{\mathcal{P}}_t\sim U(\mathcal{P}_t)$ 表示从经验池 $\mathcal{P}_t$ 中均匀抽取样本集合 $\mathring{\mathcal{P}}_t$，对于任意样本中任意经验值，利用贝尔曼最优性条件，即式(12)和式(13)，对动作值函数进行估计得到 y_t。DQN 通过将误差定义为满足贝尔曼最优性条件的动作值函数 y_t 和神经网络拟合得到的动作值函数 $Q(s,a;\boldsymbol{\theta})$ 的均方误差，完成在线任务卸载。基于 DQN 的任务卸载流程见算法 3。

算法 3 基于 DQN 的任务卸载

步骤 1) 初始化神经网络参数 $\boldsymbol{\theta}_0$，设置 $t=1$；

步骤 2) 观测系统状态 s_t，按照概率 ε 执行随机卸载决策；或按照 $(1-\varepsilon)$ 的概率，执行卸载决策 $a_t=\arg\min_{a\in\mathcal{A}}Q(s_t,a;\boldsymbol{\theta}_{t-1})$；

步骤 3) 观测系统开销 $c(s_t,a_t)$ 及新状态 s_{t+1}、经验值 $e_t=(s_t,a_t,c(s_t,a_t),s_{t+1})$，更新经验池 $\mathcal{P}_t$；

步骤 4) 从经验池 $\mathcal{P}_t$ 中均匀采样 $\mathring{\mathcal{P}}_t$，对每个采样数据执行步骤 5)；

步骤 5) 按照式(13)更新 y_t，在 $\boldsymbol{\theta}$ 方向对损失函数执行梯度下降，更新 $\boldsymbol{\theta}_t$；

步骤 6) 设置 $t=t+1$；

步骤 7) 若 $t>T$，算法结束；否则跳转步骤 2)。

3 仿真实验

3.1 参数设置

这一章节通过仿真实验验证本文的任务卸载策略。假设用户在 4×4 的正方形，共 $L=16$ 个方格(位置)的网络中移动，网络场景大小为 2 km×2 km。用户的移动模式参照文献[11]中的设置，用户以 0.6 的概率保持不移动，即 $\mathbb{P}(l\mid l)=0.6$，共 0.4 的概率往与方格相邻的上下左右 4 个位置移动。

移动边缘计算网络的参数设定主要参照文献[17]。除位于场景中心的宏基站外，网络中还有 $N=20$ 个帮助节点。本地节点的传输功率为 20 dBm，到宏基站和其他帮助节点的路径损耗分别为 $(35.7+\lg d+33.4)$ dB 和 $(35.7+\lg d+18.4)$ dB，d 为本地节点到帮助节点的距离，单位为 m。系统带宽为 20 MHz，功率频谱密度为 −174 dBm/Hz。本地节点和宏基站的处理速度分别为 10 和 35 Mbps，其他帮助节点的处理速度均匀分布于 (10,40) Mbps。

在对 DQN 任务卸载策略进行仿真时，采用 TensorFlow 框架来搭建深度神经网络[18]。在神经网络中包含 2 层隐藏层，分别含有 128 和 64 个神经元，采用 ReLU 作为激活函数[19]。其他参数列在表 1 中。

表 1 DQN 参数设置

Table 1 DQN parameters

参数	取值
经验池容量	500
经验池采样数量	32
折扣因子	0.9
ε-greedy	0.1

3.2 仿真结论

这一节验证基于 Q-learning 和 DQN 的任务卸载策略的表现，并与最优卸载方案进行比较。由于在实际中系统信息难以获得，最优卸载方案无法应用，因此本文将最优任务卸载算法作为基准来验证其他 2 个算法的效果。

1)收敛性：图 3 通过展示动作值函数的累计平均值来验证基于 Q-learning 和 DQN 的任务卸载策略的收敛性。不失一般性地，选择系统状态 $s=(7,30)$ 和动作 $a=1$ 的动作值函数作为示例，即 $Q((7,30),1)$。

从图 3 可以看出，与基于 Q-learning 的卸载策略相比，基于 DQN 的算法能够更快速地收敛到接近最优解。正如在 2.3.2 节中的讨论，基于 Q-learning 的任务卸载算法只有在系统处于状态 $s=(7,30)$ 且执行 $a=1$ 时，对应的动作值函数 $Q((7,30),1)$ 才能够得到单次更新。而基于

DQN 的任务卸载算法,由于深度神经网络参数 $\boldsymbol{\theta}$ 在每个时隙都进行更新,因此对于任意 $(s,a) \in \mathcal{S} \times \mathcal{A}$, 动作值函数在每个时隙都能够得到更新。因此在 DQN 算法中 $Q((7,30),1)$ 的更新次数远远多于 Q-learning 算法,基于 DQN 的任务卸载的收敛速度明显高于 Q-learning 的方法。

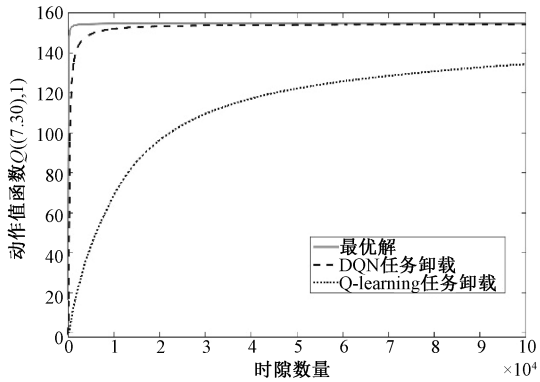

图 3　算法收敛性比较

Fig. 3　Convergence of the algorithms

2)近优性:在 2.2.1 节中提到,本文中的任务卸载模型采用的 Q-learning 算法能够以概率 1 收敛到最优解。为验证该结论,将通过长期任务的时延来验证算法的近优性。图 4 和图 5 的纵坐标为系统回报 R_t, 定义为 $R_t = C - c(s_t, a_t)$, C 为开销的上界。

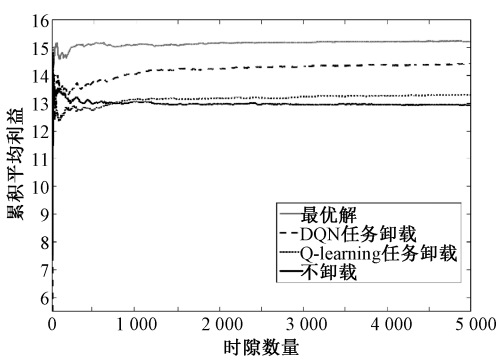

图 4　算法性能比较

Fig. 4　Performance of the algorithms

同样的,图中的最优解是基于系统信息已知的假设,因此只作为其他算法的表现基准,而基于 Q-learning 和 DQN 的任务卸载策略中,系统信息为后验知识,系统利用历史反馈以实现自主在线学习。同时,我们还将算法同“不卸载”进行比较,它忽略了用户的移动性,对所有任务都不执行任务卸载,只在本地节点进行处理。我们将观察不考虑用户移动性时的系统表现。

在图 4 中,基于 Q-learning 和 DQN 的算法对

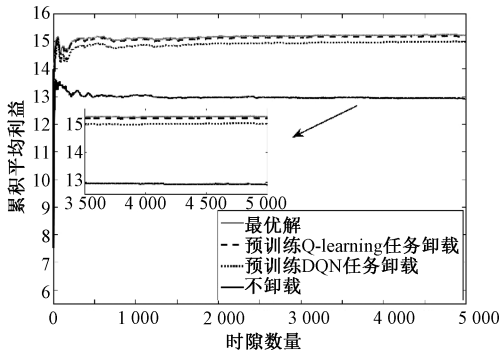

图 5　预训练后算法性能比较

Fig. 5　Performance of the algorithms after training

系统信息没有先验知识,从时隙 $t = 0$ 开始在线学习。因此在仿真的前期时隙中,Q-learning 由于学习数据的不足,导致算法对系统信息掌握较少,表现较差。此时基于 Q-learning 的任务卸载策略实现的系统回报,甚至低于不执行任务卸载的策略。但值得注意的是,基于 Q-learning 策略的回报始终保持着上升的趋势,并且从第 800 个时隙开始,表现超过不执行任务卸载的策略,且一直维持着上升的趋势。而基于 DQN 的任务卸载有着明显的优势,不论是仿真前期还是后期,获得的利益始终高于 Q-learning 和不执行卸载的策略,同时也维持着最为明显的系统回报上升的趋势。在短时间内,不考虑用户移动性的策略表现似乎与提出的算法相差不多,但从长时间尺度来看,考虑移动性的任务卸载策略,在后期表现更为优异。下面将进一步展示算法对长期任务的表现。

图 5 将经过预训练的 Q-learning 和 DQN 的算法与最优解进行比较,从而验证 2 个算法的近优性。其中基于 Q-learning 和 DQN 的任务卸载分别经过 200 000 和 2 000 个时隙的预训练,已经接近收敛。从时隙 $t = 0$ 开始进行在线的任务卸载。可以看出,与图 4 的未经预训练相比,2 个算法经过预先学习后已经对系统掌握大量信息,Q-learning 的表现已经非常接近最优解,证明它的确能够在系统信息未知的情况下,通过在线学习达到接近最优解,并高效地完成任务卸载。而 DQN 的表现虽然略逊于 Q-learning,但相差不大。然而值得注意的是,相比于 Q-learning,基于 DQN 的任务卸载只需要 1% 的预训练时间就能够达到接近 Q-learning 的效果,这也再一次验证了基于 DQN 的任务卸载策略快速的收敛速度。可以看出,在长期任务中,提出的算法在表现上一直明显优于不考虑用户移动性的不卸载策略,这也验证了上

一段的讨论,考虑用户的移动性在长时间尺度上可以实现更高的系统利益。

4 结束语

本文研究 MEC 网络中高效的在线任务卸载策略。为最小化移动用户在系统中的长期任务时延,利用马尔科夫决策过程建立任务卸载模型。在假设系统信息已知的前提下,提供了系统的最优解。在系统信息未知时,提出 2 个在线学习的算法,基于 Q-learning 和 DQN 的任务卸载。其中基于 Q-learning 的任务卸载在本文的模型中能够收敛到最优解,而基于 DQN 的算法能够快速收敛,并且达到接近最优解的表现。

参考文献

[1] Li Y T, Cheng Q F, Liu X M, et al. A secure anonymous identity-based scheme in new authentication architecture for mobile edge computing[J]. IEEE Systems Journal, 2021, DOI: 10.1109/JSYST.2020.2979006.

[2] Dinh T Q, Tang J H, La Q D, et al. Offloading in mobile edge computing: task allocation and computational frequency scaling[J]. IEEE Transactions on Communications, 2017, 65(8): 3571-3584. DOI:10.1109/TCOMM.2017.2699660.

[3] Mao Y Y, You C S, Zhang J, et al. A survey on mobile edge computing: the communication perspective [J]. IEEE Communications Surveys & Tutorials, 2017, 19(4): 2322-2358. DOI:10.1109/COMST.2017.2745201.

[4] Mach P, Becvar Z. Mobile edge computing: a survey on architecture and computation offloading [J]. IEEE Communications Surveys & Tutorials, 2017, 19(3): 1628-1656. DOI:10.1109/COMST.2017.26823.

[5] You C S, Huang K B, Chae H, et al. Energy-efficient resource allocation for mobile-edge computation offloading [J]. IEEE Transactions on Wireless Communications, 2017, 16(3): 1397-1411. DOI:10.1109/TWC.2016.2633522.

[6] Mao Y Y, Zhang J, Song S H, et al. Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems[J]. IEEE Transactions on Wireless Communications, 2017, 16(9): 5994-6009. DOI:10.1109/TWC.2017.2717986.

[7] Chen X F, Zhang H G, Wu C, et al. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning[J]. IEEE Internet of Things Journal, 2019, 6(3): 4005-4018. DOI: 10.1109/JIOT.2018.2876279.

[8] Min M H, Xiao L, Chen Y, et al. Learning-based computation offloading for IoT devices with energy harvesting [J]. IEEE Transactions on Vehicular Technology, 2019, 68(2): 1930-1941. DOI:10.1109/TVT.2018.2890685.

[9] Huang L, Bi S Z, Zhang Y J A. Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks[J]. IEEE Transactions on Mobile Computing, 2020, 19(11): 2581-2593. DOI: 10.1109/TMC.2019.2928811.

[10] 余翀,邱其文. 基于栅格地图的分层式机器人路径规划算法[J]. 中国科学院大学学报, 2013, 30(4): 528-538, 546. DOI:10.7523/j.issn.2095-6134.2013.04.015.

[11] Cheung M H, Huang J W. DAWN: delay-aware Wi-Fi offloading and network selection [J]. IEEE Journal on Selected Areas in Communications, 2015, 33(6): 1214-1223. DOI:10.1109/JSAC.2015.2416989.

[12] Nicholson A J, Noble B D. BreadCrumbs: forecasting mobile connectivity[C]//MobiCom '08:Proceedings of the 14th ACM International Conference on Mobile Computing and Networking. 2008: 46-57. DOI: 10.1145/1409944.1409952.

[13] Gambs S, Killijian M O, del Prado Cortez M N. Next place prediction using mobility Markov chains [C]//MPM '12: Proceedings of the First Workshop on Measurement, Privacy, and Mobility. 2012: 1-6. DOI:10.1145/2181196.2181199.

[14] Sutton R S, Barto A G. Reinforcement learning: an introduction[M]. 2nd ed. Cambridge, MA:The MIT Press, 2018.

[15] Watkins C J C H, Dayan P. Q-learning[J]. Machine Learning, 1992, 8(3/4): 279-292. DOI: 10.1007/BF00992698.

[16] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529-533. DOI:10.1038/nature14236.

[17] Zhu Z W, Jin S D, Yang Y, et al. Time reusing in D2D-enabled cooperative networks [J]. IEEE Transactions on Wireless Communications, 2018, 17(5): 3185-3200. DOI: 10.1109/twc.2018.2808259.

[18] Abadi M, Barham P, Chen J M, et al. Tensorflow: a system for large-scale machine learning[J]. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, 2016,abs/1605.08695.

[19] Glorot X, Bordes A, Bengio Y. Deep sparse rectifier neural networks [C] // Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. 2011: 315-323.