文章编号:1671-4598(2021)11-0189-07 DOI:10.16526/j.cnki.11—4762/tp.2021.11.034 中图分类号:TP393 文献标识码:A

基于图神经网络的具有依赖关系任务的计算卸载方法

崔 硕,覃少华,谢志斌,张家豪,卞圣强
(广西师范大学 计算机科学与信息工程学院,广西 桂林 541004)

摘要:计算卸载作为移动边缘计算的关键技术之一,通过将任务就近迁移至边缘服务器上执行大幅降低了用户的等待时延;针对具有依赖关系任务的计算卸载问题,为了解决以往文献在将表示任务依赖关系的有向无环图输入深度强化学习算法的神经网络时存在的丢失结构信息的问题,提出了一种有向无环图神经网络(DAGNN,directed acyclic graph neural network),并将其与深度强化学习相结合,用以做卸载调度的决策;卸载决策的过程被描述为马尔科夫决策过程,用提出的DAGNN评估深度强化学习算法中每个卸载动作的Q值,进而做出卸载调度决策;仿真实验表明,所提出算法在各种条件下的表现均优于其它所有基线算法,并表现出较好的稳定性和通用性。

关键词:移动边缘计算;任务调度;计算卸载;图神经网络;依赖关系

Computational Offloading Method for Tasks with Dependency Based on Graph Neural Network

CUI Shuo, QIN Shaohua, XIE Zhibin, ZHANG Jiahao, BIAN Shengqiang
(College of Computer Science and Information Engineering, Guangxi Normal University, Guilin 541004, China)

Abstract: As one of the key technologies of mobile edge computing, computing offloading can greatly reduce the user's waiting time by moving the task to the edge server. For the problem of computing offloading for tasks with dependencies, in order to solve the problem of losing the structural information in the previous literature when the directed acyclic graph representing the task dependency is input into the neural network, came up with a directed acyclic graph neural network (DAGNN), and combine it with deep reinforcement learning to make decisions about offloading scheduling. The process of offloading decision is described as Markov decision process, and the Q value of each offloading action in the deep reinforcement learning algorithm is evaluated by the DAGNN proposed in this paper, and then the offloading scheduling decision is made. Simulation results show that the proposed algorithm performs better than all other baseline algorithms under various conditions, and shows good stability and versatility.

Keywords: mobile edge computing; task scheduling; computation offloading; graph neural network; dependency

0 引言

移动边缘计算(MEC,mobile edge computing)技术通过在网络边缘部署计算和存储资源,将云计算的部分能力下沉到无线接入网等靠近用户的网络边缘,就近为用户提供计算和存储服务,降低用户的等待时延,同时减轻了骨干网的负担,防止网络拥塞的发生[1-3]。计算卸载是MEC的关键技术[4],用户设备可以通过计算卸载技术将计算密集型的应用整体或部分地卸载到MEC服务器上运行,显著降低了处理时延和能耗。

细粒度的计算卸载将应用进一步分割为多个任务,使得任务间的并行卸载处理成为可能[5]。由于在应用程序中一个任务经常需要其它任务的输出数据作为自己的输入,因而不同任务间是具有依赖关系的。现有文献大多使用有向无环图(DAG,directed acyclic graph)对具有依赖关系的计算卸载问题进行建模,但是由于DAG结构的复杂性和多样性,基于DAG的计算卸载问题是困难的[5-7]。受制于问题的复杂性,文献[8-9]大多使用启发式算法对这一问题进行求解。然而,启发式算法难以满足MEC环境下的实时动态决策的要求[10],同时也缺乏通用性。针对这一问题,文献[10-12]提出通过深度强化学习(DRL,deep reinforcement learning)算法求解DAG任务卸载调度决策的方法,利用DRL算法较强的表示能力[13]对DAG卸载决策问题巨大的状态空间进行学习,与此同时DRL算法把最大化

收稿日期:2021-04-27; **修回日期**:2021-05-12。

作者简介:崔 硕(1991-),男,河南濮阳人,硕士研究生,主要从事移动边缘计算方向的研究。
覃少华(1969-),男,广西南宁人,硕士生导师,副教授,主要从事无线传感器网络路由与安全、计算机网络与多媒体通讯、嵌入式系统方向的研究。

引用格式:崔 硕,覃少华,谢志斌,等.基于图神经网络的具有依赖关系任务的计算卸载方法[J].计算机测量与控制,2021,29(11):189-195.

长期奖励作为目标，可以避免卸载决策算法陷入局部最优，更能求得全局最优解。

然而，使用 DRL 算法解决基于 DAG 的卸载调度问题时会遇到一个棘手的问题，那就是 DRL 算法所使用的神经网络只适用于处理欧氏空间数据[14]，无法处理非欧式空间的数据，因而对这一卸载调度问题中的 DAG 图无能为力。为了能将 DAG 图输入 DRL 算法的神经网络中，文献[15]采取嵌入的方法，把 DAG 图中的每个任务节点的本地执行时延、MEC 服务器执行时延和前驱后继任务的索引等信息构成一个定长的任务嵌入向量，然后输入到 DRL 算法的神经网络中。这种方法中每个任务的嵌入向量是定长的，同时由于嵌入向量的长度直接决定了 DRL 算法中神经网络的规模，因而嵌入向量的长度不能过长。但是对于很多 DAG 结构复杂的应用来说其中核心任务的前驱后继任务的数量可能是非常大的，这将导致 DAG 图转换为嵌入向量的过程会丢失很多 DAG 自身的结构信息，从而使制定的卸载策略不够准确。鉴于图神经网络（graph neural network，GNN）所表现出的对图结构数据出色的处理能力[16]，本文针对 DAG 任务卸载调度问题提出一种新颖的将 GNN 和 DRL 算法相结合的方案，收到了很好的效果。

本文提出的方案基于 DRL 中的 DQN 算法进行了修改以使其适用于处理 DAG 任务的卸载调度问题，具体的，本文使用 GNN 替代 DQN 算法中的标准神经网络来评估每个卸载动作的 Q 值。在 GNN 的结构方面，本文借鉴了消息传递网络（MPNN，message passing neural network）的结构[17]，由于 DAG 图是有向的和无环的，本文针对 DAG 图的结构特点设计了一种有向无环图神经网络（DAGNN，directed acyclic graph neural network）用来更有效地提取 DAG 图的信息。仿真实验表明，本文提出的方案具有良好的收敛性，并在不同的节点数和网络环境下收到了优于其它对比算法的调度效果。

1 系统结构及原理

下面首先给出整个系统的实现结构流程图，如图 1 所示，系统首先读取待调度应用的 DAG 结构图，以及应用中每个任务的本地执行时延和在 MEC 服务器上执行的传输时延和计算时延。随后系统初始化 DAGNN 的参数和每个节点的输入状态。在对应用进行卸载调度时，系统首先计算每个任务的最晚开始执行时间，并按照升序进行排列，从而得到任务的优先级序列，之后系统对所有任务按照优先级的顺序依次进行调度。在对每个任务进行调度时系统首先调用下文的算法 1 衡量每个动作的 q 值，之后根据贪心策略选择要采取的动作，并更新卸载动作序列。随后系统收到环境反馈的即时奖励 r，并进入下一状态 s'，与此同时系统将四元组（s，a，r，s'）存入经验回放缓存区，并定时从经验回放缓存区中取出一些元组数据通过随机梯度下降法更新 DAGNN 的参数。

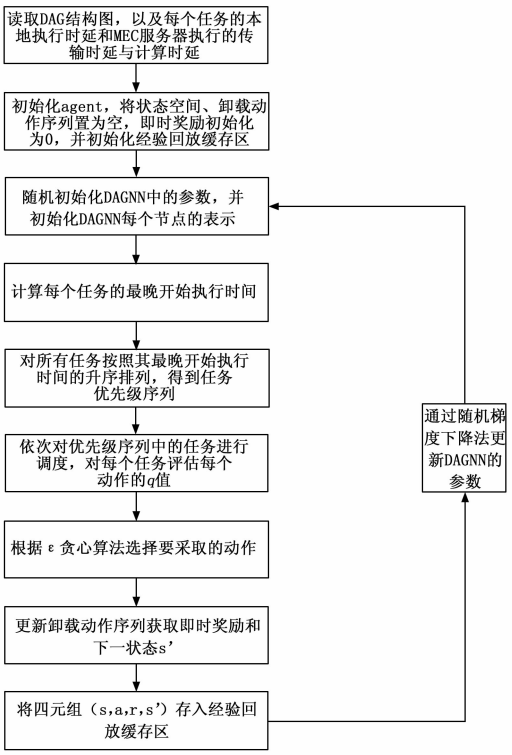

图 1 系统实现结构流程图

2 系统模型

系统模型如图 2 所示，本文提出的场景包括多个微基站（SBS）和单个用户设备（UE），其中每个微基站都配备有一台 MEC 服务器，可以进行计算任务的运行。计算任务既可以在用户设备 UE 本地运行也可以卸载至微基站的 MEC 服务器上运行，每台 MEC 服务器所能提供的计算资源各不相同，用户设备 UE 需要根据当前的信道状况和各台 MEC 服务器的计算能力对每个任务进行卸载调度决策。用集合 $N=\{0,1,2,\cdots,n,\cdots N\}$ 表示用户设备 UE 和 MEC 服务器的集合，其中 $N=\{0\}$ 表示用户设备 UE，N 为用户设备 UE 和 MEC 服务器数量的总和（由于用户设备 UE 和任一 MEC 服务器都可进行任务的计算，所以下文将 UE 和 MEC 服务器统称为处理单元）。用集合 $V=\{1,2,\cdots,i,\cdots,V\}$ 表示任务的集合，其中 i 代表第 i 个任务，V 代表任务数量的总和，对于每个任务 $i\in V$ 分别定义其被执行所需的 CPU 周期数为 C_i、输入数据量为 D_i^{in} 和输出数据量为 D_i^{out}。

2.1 通信模型

用户设备 UE 和每个微基站之间通过无线链路相连，我们假设在任务的执行过程中 UE 与 MEC 服务器之间的信道保持稳定，根据香农公式可得到用户设备 UE 到 MEC 服务器间链路的传输速率为：

$$R_n^{ul}=B\log_2\left(1+\frac{p_0 g_n}{N_0}\right) \tag{1}$$

其中：参数 B 代表用户设备 UE 可用的信道带宽，p_0

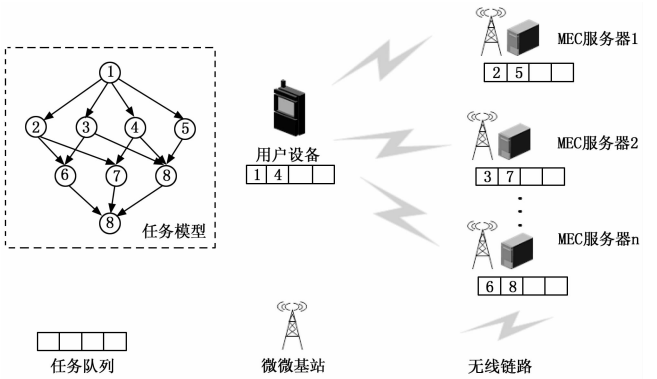

图 2　系统模型

代表用户设备 UE 的发送功率，g_n 表示 UE 与 MEC 服务器 n 之间的信道增益，N_0 表示高斯噪声功率。

2.2　计算模型

用户设备 UE 在同一时隙内只产生一个计算密集型的应用，它被进一步分割为多个相互之间具有依赖关系的任务，每个处理单元在同一时隙内只能处理一个任务，每个任务只能被一个处理单元所处理。下面分别讨论任务在本地计算和 MEC 服务器上计算的时延。

①本地计算：

用 f_0 表示用户设备 UE 自身的计算能力，则任务 i 在 UE 本地计算的计算时延可以表示为：

$$T_{0,i}^{com}=\frac{C_i}{f_0} \tag{2}$$

由于本地计算无需进行数据的传输所以不存在通信时延，所以任务 i 在本地计算的全部时延为：

$$T_{0,i}=T_{0,i}^{com} \tag{3}$$

②MEC 服务器计算：

当任务 i 被调度为 MEC 服务器 n 计算时，数据需要在用户设备 UE 和 MEC 服务器之间传输，因而 MEC 服务器计算情景下的时延由两部分构成：传输和 MEC 服务器执行。特别地，由于任务被执行所产生的计算结果的数据量往往很小，类似于现有大多数文献的假设[8,12,18]，本文不考虑计算结果的传输时延。由此可得任务 i 在 MEC 服务器 n 上计算的时延为：

$$T_{n,i}=T_{n,i}^{tra}+T_{n,i}^{com}=\frac{D^{in}}{R_n^{ul}}+\frac{C^i}{f_n} \tag{4}$$

其中：$T_{n,i}^{tra}$表示任务 i 的输入数据从用户设备 UE 传输到 MEC 服务器 n 的传输时延，$T_{n,i}^{com}$表示任务 i 在 MEC 服务器 n 上的执行时延，f_n 表示 MEC 服务器 n 的计算能力。

2.3　依赖关系模型

本文将应用中不同任务间的依赖关系建模为一个 DAG，表示为 $G=(V,E)$，其中 V 表示 DAG 中节点的集合，每个节点 $i\in V$ 表示 UE 运行的应用中的一个任务；E 表示 DAG 中边的集合，每条边 $e(i,j)\in E$ 表示任务 i 和 j 之间的依赖关系，即任务 j 需要任务 i 的输出数据作为自己的输入，并称任务 i 为 j 的前驱，任务 j 为 i 的后继，一个任务只有在其所有前驱任务完成后才能开始执行。在一个 DAG 中，没有前驱任务的任务被称为入口任务，没有后继任务的任务被称为出口任务，一个应用允许同时有多个入口任务和出口任务。

在应用开始卸载调度后，用 $AFT_{n,i}$表示任务 i 在处理单元 n 上执行时的实际完成时间。由于一个任务只有在其所有直接前驱任务都被执行完成后才能进入准备就绪状态，所以任务 i 的就绪时间可以表示为：

$$RT_i=\max_{h\in pre(i)}AFT_{n,h} \tag{5}$$

其中：$pre(i)$ 表示任务 i 所有直接前驱任务的集合。然而，当一个任务被分配至处理单元 n 上执行时该处理单元可能并未处于空闲状态，所以任务的就绪时间并不一定是它真正开始被执行的时间。我们用 $AT_{n,i}$表示处理单元 n 最早可以为任务 i 服务的时间，容易得到：

$$EST_{n,i}=\max\{RT_i,AT_{n,i}\} \tag{6}$$

用 $ET_{n,i}$表示任务 i 在处理单元 n 上执行所花费的时间，它可以表示为：

$$ET_{n,i}=\frac{C_i}{f_n} \tag{7}$$

由于用户设备 UE 的应用都是时延敏感型的，所以应用存在一个最大容忍时延。我们定义 $t^{\max}$为应用的最大容忍时延，它可以看作是应用的 DAG 图中出口任务的最晚执行完成时间。用 $LFT_{n,i}$表示 DAG 图中任务 i 的最晚执行完成时间，它可以从应用的出口任务逐级往上递归地求出，则 $LFT_{n,i}$可以表示为：

$$LFT_{n,i}=\min_{j\in suc(i)}(LFT_{n,j}-ET_{n,j}^{\min}) \tag{8}$$

其中：$j\in suc(i)$ 表示任务 i 所有直接后继任务的集合，$ET_{n,j}^{\min}=\min_{1\leqslant n\leqslant N}ET_{n,j}$，即 $ET_{n,j}^{\min}$为任务 j 在所有 N 个 MEC 服务器上最短的执行时间。值得注意的是，任务的最晚开始执行时间可以用来衡量该任务的紧急程度，这在下文的章节中将会用到。

在定义了任务的最晚完成时间后我们可以相应地得到任务的最晚开始执行时间，它可以表示为：

$$LST_{n,i}=LFT_{n,i}-ET_{n,i}^{\min} \tag{9}$$

2.4　优化目标制定

本文以使任务的时延最小为优化目标，首先定义两个 0－1 变量 $x_{n,i}$和 $y_{h,i}$，它们的取值分别为：

$$x_{n,i}=\begin{cases}0 & \text{否则，}\\ 1 & \text{任务 } i \text{ 在服务器 } n \text{ 上执行}\end{cases} \tag{10}$$

$$y_{h,i}=\begin{cases}0 & \text{否则，}\\ 1 & \text{任务 } h \text{ 执行顺序在 } i \text{ 之前}\end{cases} \tag{11}$$

基于上文的定义，本文的优化问题可表示为：

$$\min\sum_{i=1}^{V}\sum_{n=0}^{N}T_{n,i} \tag{12}$$

$$s.t.\ EST_{n,i}\geqslant x_{n,i}\cdot y_{n,i}\cdot AFT_h \tag{12a}$$

$$x_{n,i}\in\{0,1\},\forall n\in N,i\in V \tag{12b}$$

$$\sum_{n=0}^{N}\sum_{i=1}^{V}x_{n,i}=1 \tag{12c}$$

其中：约束式（12a）表示任务 i 的最早开始执行时间不能早于其直接前驱任务 h 的实际完成时间，约束式（12b）和（12c）表示同一任务只能在同一个处理单元上计算。

3 基于 DRL 和 GNN 的 DAG 任务卸载调度算法

由于 DAG 任务调度问题的复杂性，文献[11]等提出使用深度强化学习的方法进行卸载决策。但由于 DRL 算法无法处理诸如 DAG 图一类的图结构的信息，文献［22］采用节点嵌入的方式将 DAG 图的拓扑结构和节点信息转化为任务序列输入到 DRL 算法的神经网络中。但这种方式会损失很多 DAG 图的结构信息。鉴于图神经网络在处理图结构数据时的优异表现，本文提出将图神经网络（GNN，graph neural network）与 DRL 算法相结合用于解决 DAG 任务的卸载调度问题。与此同时由于 DAG 图是有向图，本文针对 DAG 图的结构特点设计了一种有向图神经网络（DAGNN）用于提取 DAG 图的信息，取得了较好的效果。

3.1 任务优先级

为了简化卸载调度决策过程同时也满足任务之间的依赖性，与其它基于 DAG 图的任务卸载调度问题文献一样[7,19]，本文首先确定任务的调度顺序。由于计算卸载的任务大多是时延敏感型的，所以在确定任务调度的优先级时应主要考虑任务的时延约束。如上文所述，任务的最晚完成时间可以看作是每个任务的时延约束，每个任务的最晚完成时间可以从出口任务开始逐层向上递归地求出，基于此我们可求得所有任务的优先级。

3.2 MDP 的构造

3.2.1 状态空间

MDP 的状态空间应该能够反映当前应用的 DAG 结构、已经完成调度决策的任务情况等，基于此状态空间应包括以下部分：①当前已完成调度的任务序列；②已完成调度任务的卸载动作的集合；③DAG 图。每当完成一个任务节点的调度，系统更新已完成调度任务序列及其动作集，并进入下一状态。

3.2.2 动作空间

动作空间中的动作表示的是 G 中每个任务的卸载调度决策，由于本文中每个任务都可以被分配给 N 个处理单元中的任意一个进行执行且分配给每个处理单元的概率是相同的，所以动作空间具体可以表示为：

$$A=\{0,1,2,\cdots,n,\cdots N\}$$

其中：动作 $a=\{0\}$ 表示任务被分配在本地执行，$a=\{n\}$ 表示任务被分配给了第 n 台 MEC 服务器执行。

3.2.3 即时奖励

智能体在执行选定的动作 a 后，环境将会反馈给智能体一个即时奖励 r，在本章中即时奖励被设定为任务节点调度前后应用执行时延的差值，特别地，如果任务的执行时间超过了其最大容忍时延，即时奖励被设定为 0。

3.3 有向无环图神经网络

由于 DAG 图是有向的、无环的，所以我们针对 DAG 的结构特性对标准的 GNN 做了改动使其能够处理 DAG 图。由于文献[16]归纳并抽象出各种 GNN 模型的共性，提出了一种通用的 GNN 处理框架信息传递网络（MPNN，message passing neural network），鉴于其在各领域的应用中展现出的良好的 GNN 信息提取能力[17]，本文的 DAGNN 借鉴了 MPNN 的基本结构，具体的结构如图 3 所示。

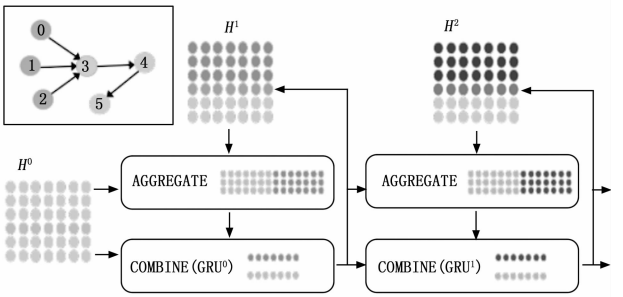

图 3 DAGNN 结构图

DAGNN 中每个节点代表应用的一个任务，共包含 L 层网络。DAGNN 通过聚合函数对每个节点的直接前驱节点的节点表示进行聚合，并把得到的聚合信息用连结函数与该节点在当前网络层的节点表示进行连结，用以更新节点表示。在所有节点间迭代进行这一过程，最后使用最大池化的方法将所有网络层的出口节点的节点表示进行聚合，通过读出函数将聚合后的信息读出，生成最终的图表示。DAGNN 总的计算过程如式（13）和式（14）所示：

$$h_v^l=COMBINE^l(h_v^{l-1},\ AGGREGATE^l(\{h_u^l\mid u\in p(v)\},h_v^{l-1})) \tag{13}$$

$$h_G=READOUT(h_v^L) \tag{14}$$

其中：$l=1,\cdots,L$，表示 DAGNN 的各个网络层，h_v^l 为节点 v 在第 l 层网络的隐藏状态，COMBINE^l、AGGREGATE^l 和 READOUT 分别是连结函数、聚合函数和读出函数，它们是带参数的神经网络，$P(v)$ 表示节点 v 所有直接前驱节点的集合。式（14）中 $v\in T$，T 表示 DAG 图中没有直接后继的节点（即出口节点）的集合。

由于每个任务的执行时延只受其前驱任务的影响，因此我们在进行信息聚合时只考虑节点的前驱任务。除此之外，本文在聚合函数计算过程中引入注意力机制以便更好地聚合相关节点的信息，具体的计算方式如式（15）所示：

$$\underbrace{m_v^l}_{\text{message}}=AGGREGATE^l(\{h_u^l\mid u\in P(v)\},h_v^{l-1})=\sum_{u\in P(v)}a_{vu}^l(\underbrace{h_v^{l-1}}_{\text{query}},\underbrace{h_u^l}_{\text{key}})\underbrace{h_u^l}_{\text{value}} \tag{15}$$

其中：h_v^{l-1}、h_u^l 分别充当注意力机制中的 query 和 key，带权系数 a_{vu}^l（h_v^{l-1}，h_u^l）的计算公式如下：

$$a_{w}^{l}(h_{v}^{l-1},h_{u}^{l}) = \underset{u \in P(v)}{softmax}(w'^{T}_{1}h_{v}^{l-1} + w'^{T}_{2}h_{u}^{l}) \tag{16}$$

其中：w_1^l 和 w_2^l 是模型的参数。在得到每个节点 v 的输出信息 m_v^l 后使用连结函数将节点 v 在上一网络层的节点表示 h_v^{l-1} 与当前网络层的输出信息 m_v^l 进行连结，得到更新后的节点表示 h_v^l。本文连结函数中使用了门控循环单元（GRU，gated recurrent units）机制，具体计算过程如式（17）：

$$h_v^l = COMBINE^l(h_v^{l-1},m_v^l) = GRU^l\left(\underbrace{h_v^{l-1}}_{\text{input}},\underbrace{\overbrace{m_v^l}^{\text{message}}}_{\text{state}}\right) \tag{17}$$

其中：h_v^{l-1}、m_v^l 和 h_v^l 分别是 GRU 机制的输入、过去状态和更新状态。

在进行图表示的读出时，类似于以往图表示学习文献，我们用最大池化的方法把同一节点在每个网络层的表示进行连结，之后用一个全连接层产生图表示的读出。但与以往文献不同的是，根据 DAGNN 特殊的结构我们只把出口节点在每个网络层中的节点表示进行连结，之后用一个全连接层得出最终的图表示。具体计算过程如式（18）所示：

$$h_G = FC(MaxPool_{v \in T}(\bigvee_{l=0}^{L} h_v^l)) \tag{18}$$

其中：$\vee_{l=0}^{L} h_v^l$ 表示将所有出口节点在所有网络层中的节点表示进行聚合。则 DAGNN 产生图表示的过程可以用算法 1 来表示。

算法 1：基于 DAGNN 的信息传递算法

1. 输入：节点 v 的输入特征 h_v^0，DAG 结构图
2. 输出：动作的 q 值
3. for $l=1$ to L do
4. for $v=1$ to V do
5. 通过式（3.22）计算每个节点 v 与其直接前驱节点聚合后的信息 m_v^l
6. 通过式（3.24）将 m_v^l 与节点 v 上一网络层的节点表示 h_v^{l-1} 进行连结
7. 得到节点 v 更新后的节点表示 h_v^l
8. end for
9. end for
10. for v in T do
11. 通过式（3.25）将出口节点在所有网络层中的节点表示进行连结
12. 用最大池化方法处理连结后的信息
13. 将上一步处理过的信息输入全连接层，生成最终的图表示
14. end for

3.4 卸载调度流程

卸载调度的具体过程如算法 2 所示。在算法开始时，DRL 算法的智能体收到要进行卸载调度决策的应用及其自身的 DAG 图，并获取每个任务节点的本地执行时延、MEC 服务器上传输和执行时延等信息，生成每个任务节点的节点表示，将累积奖励和动作空间分别初始化为 0 和 Ø，并创建一个经验回放缓存区。与此同时，用户设备产生一个待调度的应用，之后算法首先根据 3.1 节中所说的方式确定该应用中所有任务的调度优先级，然后按照任务优先级降序的顺序依次对任务进行调度决策。在对每个任务做决策时，智能体使用算法 1 评估每个动作的 q 值，得到每个动作的 q 值后根据 ε 贪心策略选择动作，在实施所选择的动作后系统进入下一状态 s'，并获得环境反馈的即时奖励 r，同时系统将当前状态 s，采取的动作 a，下一状态 s'，即时奖励 r 等信息存入经验回放缓存区，用于对图神经网络进行训练和更新参数。卸载动作序列即为卸载调度决策。

算法 2：基于 DAGNN 的卸载调度算法

1. 输入：DAG 结构图，每个任务的本地执行时延 $T_{0,i}^{\text{com}}$ 和 MEC 服务器执行的传输时延 $T_{n,i}^{\text{tra}}$ 和计算时延 $T_{n,i}^{\text{com}}$
2. 输出：卸载动作序列 β
3. 初始化 agent，状态空间 $S \leftarrow \{\emptyset\}$，卸载动作序列 $\beta \leftarrow \{\emptyset\}$，即时奖励 $r \leftarrow 0$，初始化经验回放缓存区 buffer
4. 随机初始化 DAGNN 中的参数，初始化 DAGNN 每个节点的表示
5. 根据 2.3 节中所述方式计算每个任务 i 的最晚开始执行时间 $LST_{n,i}$
6. 将所有任务按照其 $LST_{n,i}$ 升序排列，得到优先级序列 $\varphi = \{x_1, \cdots, x_v\}$
7. for x_1 to x_v in φ do
8. for $a=0$ to N do
9. 调用算法 1 计算每个动作 a 的 q 值
10. 根据 ε 贪心算法选择要采取的动作 a
11. $s \leftarrow s'$，更新动作序列 β，获取即时奖励 r
12. 将四元组（s，a，r，s'）存入经验回放缓存区 buffer
13. 随机从 buffer 中取出若干个元组
14. 令 $y_j = \begin{cases} r_j & \text{若 } s' \text{ 为终止状态} \\ r_j + \gamma \max\limits_{a'} Q(s', a'; \theta) & \text{否则} \end{cases}$
15. 使用随机梯度下降算法 SGD 更新 DAGNN 的参数
16. end for
17. end for

4 仿真实验

本节将对本文提出的算法进行实验验证，主要以应用的完成时延作为标准来衡量各算法的性能，除此之外还考虑了在改变某些环境因素的情况下验证本文算法的通用性和稳定性。

4.1 实验设计

本节的实验情景包括多个微微基站（均配备有 MEC 服务器）和一个用户设备 UE，具体的实验参数仿照文献[20]设置，如表 1 所示。

表 1 仿真实验参数

参数	数值
信道带宽/MHz	10
用户设备传输功率/mW	100
用户设备 CPU 计算能力/GHz	1～2
背景噪声功率/dBm	－100
微基站的传输功率/mW	1000
MEC 服务器的计算能力/GHz	2～4
任务的数据量大小/kB	200～400
任务的最大容忍时延/s	0.5～3
任务的平均计算密度/(cycles/bit)	400～600

为了模拟 DAG 应用的多样性，本节的实验使用文献[19－21]所采用的 DAG 生成器，本节实验中所用到的训练和测试 DAG 均通过该生成器生成。与此同时，本节设置了 4 个基线算法用以与本文提出的算法作对比，具体为：①本地执行（LE，local execution）：应用的所有任务均在 UE 本地执行；②卸载执行（OE，offloading execution）：DAG 图中的所有任务均被调度为卸载执行；③随机调度（RS，random scheduling）：对 DAG 图中所有任务均随机确定调度决策；④DRLOSM 算法（DRLOSM）：文献［22］提出的用来解决 DAG 应用卸载调度问题的算法，它采用图嵌入的方式将 DAG 图转化为节点信息向量的集合输入 DRL 算法的 DNN 网络中，并取得了同类文献中较好的调度效果。

图的密度是描述 DAG 中两个相邻的任务层之间依赖关系数量的参数，若该参数较大则表明 DAG 图中任务的前驱和后继节点较多。由于本文处理的主要是 DAG 结构复杂、模块的前驱和后继较多的应用，因而本节实验还将在改变 DAG 图的密度的条件下验证本文算法的性能。

4.2 实验结果分析

首先，对本章算法的收敛性进行评估。在上文所述的仿真参数下，随机生成 3 000 个 DAG 样本用于训练以得到对该批样本的卸载调度方案，然后将该调度方案在同样的仿真环境中对该批样本进行模拟调度，并记录该批样本所获得的平均累积奖励。如图 4 所示，该批样本的平均累积奖励在经历 700 此迭代后逐渐趋于稳定，证明了该算法具有良好的收敛性。

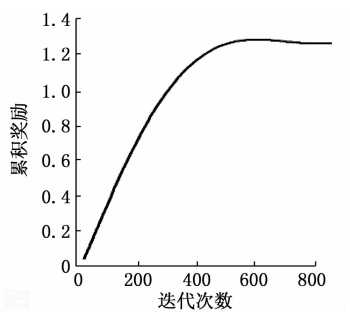

图 4 收敛性曲线

然后，基于使应用的总处理时延最小的优化目标，在默认环境下对任务节点数量不同的 DAG 应用的处理时延进行了比较，图 5 展示了实验结果。可以看出在不同节点数的条件下，完全本地执行（LE）的方式的时延都是最高的，本文提出的算法（TOSA－DAGNN）都要优于其它对比算法。值得注意的是，虽然文献［22］提出的 DRLOSM 算法在节点数较少时也体现出了较好的性能，但是在节点数增加至 50 个后本文的算法开始明显优于 DRLOSM 算法，这是由于 DRLOSM 算法将所有节点的信息转化为统一长度的向量来输入 DRL 算法的神经网络中，这种方式在处理节点数较多且 DAG 结构复杂的应用时会丢失很多 DAG 图的结构信息，导致做出的卸载决策不够准确。

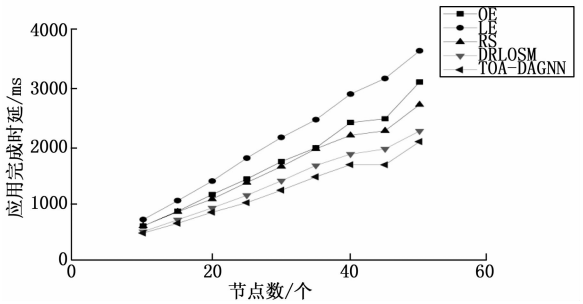

图 5 应用的完成时延随任务节点数的变化

下面将在改变 DAG 图的密度的条件下验证本文算法的性能，如图 6 所示，在节点数量均为 40 的条件下，本节实验分别在 DAG 图密度为 0.3、0.6、0.8 和 0.9 时进行验证。从图中可以看出，随着图密度的增加本地执行（LE）、卸载执行（OE）和随机调度（RS）3 种方式变化不大，这是由于在这 3 种方式中每个任务的执行方式是固定不变的，因而并不受到图密度变化的影响。但 DRLOSM 算法受图密度变化的影响较为明显，这是因为 DRLOSM 算法在处理过程中将任务的前驱和后继节点的信息转化为固定长度的向量，如果某个任务的前驱和后继节点过多就将超出的部分舍弃，而图的密度较大时 DAG 图中任务的前驱和后继节点往往是比较多的，因而此时 DRLOSM 算法的处理方式会舍弃大量的 DAG 图的结构信息，导致做出的卸载决策不准确。相较之下，本文的算法在各种图密度的条件下都能保持较好的卸载调度效果。

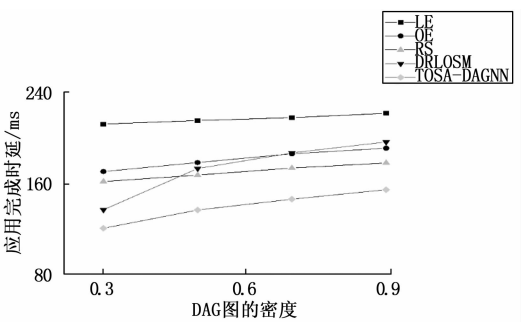

图 6 应用完成时延随图密度的变化

为了验证本文算法的通用性和稳定性，下面在节点数均为 40 的情况下改变网络的传输速率来验证各算法的性能，如图 7 所示。可以看出随着网络传输速率的增大，除

本地执行（LE）以外的各算法的完成时延都处于下降趋势，这是因为网络传输速率的加大降低了任务从用户设备 UE 到 MEC 服务器之间的传输时延，而本地执行（LE）的方式由于其所有任务都在本地执行，所以其完成时延不受网络传输速率的影响。卸载执行（OE）的方式由于其所有任务都被卸载至 MEC 服务器执行，所以随着网络传输速率的增加它的完成时延大幅降低，并在网络传输速率超过 8 Mbps 后其完成时延逐渐超过 DRLOSM 算法。同时可以看出，本文提出的算法（TOSA－DAGNN）始终在完成时延方面优于其它算法。

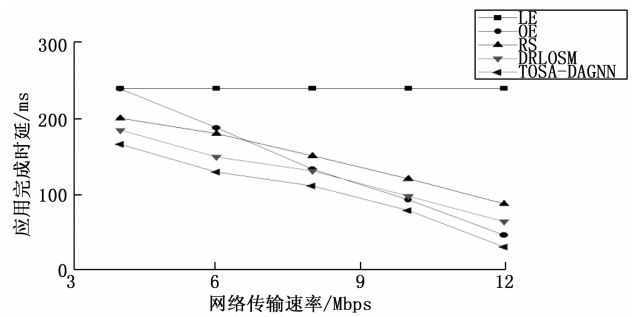

图 7　应用完成时延随网络传输速率的变化

5　结束语

本文研究了 MEC 中具有依赖关系任务的计算卸载问题，通过 DAG 对任务间的依赖关系进行建模，将卸载调度建模为 MDP 过程进而使用 DRL 算法进行求解。为了解决其它文献在将 DAG 图输入 DRL 算法的神经网络时存在的丢失 DAG 图的结构信息的问题，创新地将 GNN 与 DRL 算法相结合用于解决卸载调度的问题。同时为了适应 DAG 图的结构特点将 MPNN 进行了改进，提出有向无环图神经网络（DAGNN），最后的实验结果表明，与其它基线算法相比本文提出的方案收到了最好的调度效果，尤其是在处理 DAG 结构复杂、任务的前驱和后继节点较多的应用时本文的方案明显优于其它算法，并在网络环境改变时始终优于其它算法，体现了本文方案的通用性和稳定性。

参考文献：

[1] SATYANARAYANAN M，BAHL P，CA CERES R，et al. The Case for VM－Based Cloudlets in Mobile Computing [J]. IEEE Pervasive Computing，2009，8 (4)：14－23.

[2] SATYANARAYANAN M. Mobile computing [J]. Computer，1993，26 (9)：81－82.

[3] YU Y. Mobile edge computing towards 5G：Vision，recent progress，and open challenges [J]. China Communications，2016，13 (Supplement2)：89－99.

[4] SHI W，JIE C，QUAN Z，et al. Edge Computing：Vision and Challenges [J]. Internet of Things Journal，IEEE，2016，3 (5)：637－46.

[5] CHEN Z，CHENG S. Computation Offloading Algorithms in Mobile Edge Computing System：A Survey [M]. 2019.

[6] ULLMAN J D. NP－complete scheduling problems [J]. Journal of Computer & System Sciences，1975，10 (3)：384－93.

[7] 李金忠，夏洁武，曾劲涛，等. 网格工作流调度算法研究综述 [J]. 计算机应用研究，2009，26 (8)：2816－2820.

[8] LIU Y，WANG S，ZHAO Q，et al. Dependency－Aware Task Scheduling in Vehicular Edge Computing [J]. IEEE Internet of Things Journal，2020 (99)：1.

[9] SHU C，ZHAO Z，HAN Y，et al. Multi－User Offloading for Edge Computing Networks：A Dependency－Aware and Latency－Optimal Approach [J]. IEEE Internet of Things Journal，2020，7 (3)：1678－1689.

[10] Computation Offloading Toward Edge Computing [J]. Proceedings of the IEEE，2019，107 (8)：1584－1607.

[11] YAN J，BI S，ZHANG Y. Offloading and Resource Allocation with General Task Graph in Mobile Edge Computing：A Deep Reinforcement Learning Approach [J]. IEEE Transactions on Wireless Communications，2020，19 (8)：5404－5419.

[12] YAN J，BI S，ZHANG Y，et al. Optimal Task Offloading and Resource Allocation in Mobile－Edge Computing With Inter－User Task Dependency [J]. IEEE Transactions on Wireless Communications，2019，19 (1)：235－250.

[13] LI Y. Deep reinforcement learning：An overview [Z]. arXiv preprint arXiv：1701.07274，2017.

[14] SCARSELLI F，GORI M，TSOI A，et al. The Graph Neural Network Model [J]. IEEE Transactions on Neural Networks，2009，20 (1)：61.

[15] 卢海峰，顾春华，罗　飞，等. 基于深度强化学习的移动边缘计算任务卸载研究 [J]. 计算机研究与发展，2020，57 (7)：1539－1554.

[16] ZHOU J，CUI G，ZHANG Z，et al. Graph Neural Networks：A Review of Methods and Applications [Z]. arXiv e－prints，2018：arXiv：1812.08434.

[17] ASPURU－GUZIK A. Quantum Machine Learning and Quantum Computing for Chemistry [C] // proceedings of the APS March Meeting，2017.

[18] SUN J，GU Q，ZHENG T，et al. Joint Optimization of Computation Offloading and Task Scheduling in Vehicular Edge Computing Networks [J]. IEEE Access，2020 (99)：1.

[19] ARABNEJAD H，BARBOSA J G. List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table [J]. IEEE Transactions on Parallel & Distributed Systems，2014，25 (3)：682－694.

[20] CHEN W，HE Y，QIAO J. Cost Minimization for Cooperative Mobile Edge Computing Systems [C] // proceedings of the 2019 28th Wireless and Optical Communications Conference (WOCC)，F，2019.

[21] LEI，YANG，et al. A Framework for Partitioning and Execution of Data Stream Applications in Mobile Cloud Computing [J]. Performance Evaluation Review，2013，40 (4)：23－32.

[22] 詹文翰，王　瑾，朱清新，等. 移动边缘计算中基于深度强化学习的计算卸载调度方法 [J]. 计算机应用研究，2021，38 (1)：241－245.