基于改进 DQN 的最优联盟结构生成策略优化

赵庶旭,周宏泽,王小龙

(兰州交通大学电子与信息工程学院,甘肃 兰州 730070)

摘　要: 边缘服务器往往需要在资源有限的情况下通过组建联盟的方式协同执行任务,考虑服务器的资源利用率随任务执行而动态变化的特性,如何确保任务能够尽快完成的同时减少重构联盟耗费的成本是一大难点。针对上述问题,提出一种基于双重深度 Q 网络(DDQN)优化的联盟结构优化策略。首先,以最大化任务完成效率以及最小化联盟构建成本为优化目标,通过定义状态空间、动作空间和奖励函数,将问题建模为引入成本的马尔可夫决策过程(CT-MDP)。其次,针对 CT-MDP 中高维状态空间下易出现 Q 值过高估计的问题,提出一种基于 DDQN 的轻量化最优联盟结构搜索算法,通过两套独立的 Q 网络减少更新过程中的正向累计误差。为满足边缘设备在训练过程中对资源占用率的严格要求,对激活函数进行优化以降低训练模型对存储资源的需求。最后,通过仿真实验将所提算法与 Q-learning、DQN、Dueling DQN 等算法进行比较分析。实验结果表明,所提方法具有良好的收敛性和稳定性,且在联盟构建成本与资源占用率方面分别降低了 20.36%与 12.12%,证明了该方法的有效性。

关键词: 人工智能;移动边缘计算;计算资源受限;资源调度;联盟结构生成;深度强化学习

中图分类号: TP391　　**文献标志码:** A　　**DOI:** 10.19678/j.issn.1000-3428.0070326

Optimization of Optimal Coalition Structure Generation Strategy Based on Improved DQN

ZHAO Shuxu, ZHOU Hongze, WANG Xiaolong

(School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, Gansu, China)

【Abstract】 Edge servers often need to collaborate to execute tasks by forming alliances when resources are limited. Ensuring that tasks can be completed as quickly as possible while reducing the cost of restructuring alliances is a major challenge considering the dynamic changes in server resource utilization for task execution. A coalition structure optimization strategy based on dual Deep Q-Network (DDQN) optimization is proposed to address these issues. First, with the optimization objective of maximizing task completion efficiency and minimizing alliance-building costs, the problem is modeled as a Cost Introduced Markov Decision Process (CT-MDP) by defining the state space, action space, and reward function. Second, in response to the problem of overestimating Q-values in high-dimensional state spaces in the CT-MDP, a lightweight optimal alliance structure search algorithm based on DDQN is proposed. Two independent Q-networks are used to reduce the forward cumulative error during the update process. To satisfy the strict requirements of the edge devices for resource utilization during training, the activation function is optimized to reduce the storage resource requirements of the training model. Finally, the proposed algorithm is compared with Q-learning, DQN, Dueling DQN, and other algorithms using simulation experiments. The results show that the proposed method has good convergence and stability, and it reduces alliance construction costs and resource utilization by 20.36% and 12.12%, respectively, demonstrating the effectiveness of the method.

【Key words】 Artificial Intelligence (AI); Mobile Edge Computing (MEC); limited computing resources; resource scheduling; coalition structure generation; Deep Reinforcement Learning (DRL)

0　引言

随着智能设备的普及和移动应用的爆炸式增长,移动边缘计算(MEC)已成为支撑现代物联网服务的关键技术[1]。在边缘计算架构中,边缘服务器(边缘节点)扮演着至关重要的作用,其往往被边缘服务提供商(ESP)部署在靠近数据源的一侧,来为用户提供快速、低延迟的服务[2]。然而,由于资源的局限性,单个服务器往往难以独立应对日益增长的计算需求和复杂的任务负载[3]。

基金项目: 甘肃省重点研发计划(20YF8GA123)。

作者简介: 赵庶旭,男,教授、博士,主研方向为智能交通、边缘计算;周宏泽(通信作者),硕士研究生;王小龙,博士。

收稿日期: 2024-09-05　**修回日期:** 2024-10-15　**E-mail:** 1443756217@qq.com

计算机工程 www.ecice06.com

在这种资源受限的环境下，联盟结构模型为解决该问题提供了可能[4]。联盟结构作为博弈论中常用于解决合作问题的理论模型，用于描述一组参与者(智能体)如何通过不同形式的合作来最大化社会福利(效用)[5]。在边缘计算环境中，联盟结构可表示为边缘节点集合的一种有效划分，其由一组平等的边缘节点构成，节点间可通过协作来完成分配任务。构建边缘联盟的目的是通过优化资源分配和任务调度来提高整个系统的效率和性能，即寻求一个最优的联盟结构。因此，资源受限的边缘服务器动态任务分配问题可被转化为一个联盟结构生成问题[6]，求解联盟结构生成问题的关键在于设计一种能够适应环境变化并动态调整合作策略的算法，以实现资源受限边缘计算环境中任务处理的高效性和系统的稳定性。

目前，求解多智能体下联盟结构生成问题的方法大致可分为动态规划算法、合作博弈方法以及启发式近似算法。文献[7]考虑到智能体数量增加会导致联盟数量指数型增长，提出了快速动态生成算法，其通过对联盟结构图进行剪枝，再利用动态规划方法来搜索最优联盟结构。然而，剪枝预处理对于缩减搜索空间的作用并不显著。文献[8]将动态规划算法与任意时间方法相结合，提出了改进动态规划算法，其实际内存使用率仅为传统算法的1/3～2/3，但该算法在减少实际内存使用率以及提升求解速度的同时忽略了求解质量。文献[9]尝试利用合作博弈的方法求解联盟结构生成问题，着重研究了加权匹配和加权分配博弈，其中的联盟值由匹配问题的最优值决定。在此基础上，文献[10]提出了配分函数博弈，以模拟联盟值受其他智能体划分方式影响下的最优联盟结构生成情况。在启发式算法研究方面，文献[11]提出了一种基于离散粒子群优化算法的最优联盟结构搜索方法，该方法在解决资源调度问题方面优于其他启发式方法。文献[12]在上述研究基础上，提出了一种结合合作博弈和启发式算法的边缘联盟结构生成方法，其引入讨价还价集的概念消除不满足条件的联盟结构，从而缩小策略空间，并利用M进制离散粒子群算法搜索最优联盟结构解。尽管上述方法在求解联盟结构生成问题时取得了一定成效，但在MEC环境下，边缘节点状态会随时间的推移和任务的处理过程不断变化，因此要求联盟结构具备一定的灵活性来实时调整以适应这种动态性。

综上所述，目前多数基于MEC的联盟结构生成问题研究都是基于静态环境开展的，即一旦找到最优的联盟结构，问题就得到解决。基于合作博弈的方法能够有效地处理静态问题，而实际的MEC应用场景中计算资源和任务需求会随时间发生变化，因此需要考虑动态的联盟结构生成方法。文献[13]考虑到联盟组建过程是一个动态变化的过程，将该过程建模为一个马尔可夫决策过程(MDP)，并利用Q-learning算法进行求解，验证了其在动态环境下的有效性。文献[14]则从服务器联盟的角度，提出了一种动态联盟组建算法CR-learning，以实现多个边缘服务器动态地组建联盟。该算法在减小求解空间的同时，显著降低了用户成本，为MEC环境下的联盟结构生成问题提供了新的解决方案。文献[13-14]在求解模型时均采用传统的强化学习方法[15]，在任务数量和边缘节点数量较多的情况下，该类方法难以应对维度爆炸以及收敛速度慢的问题，从而导致任务完成效率大幅降低。此外，上述研究仅将联盟的构建过程用一个简单的MDP表示，无法很好地反映联盟构建的动态特点，且会忽略构建过程中边缘节点自身的状态(占用率)、不同联盟构成所需的成本以及能量损耗等因素的影响。

针对上述问题，本文将能量损耗和联盟构建成本作为影响因素纳入模型构建中，提出了基于改进深度Q网络(DQN)的求解方法，具体工作如下：

1)考虑边缘服务器在联盟中的状态会随环境而发生变化，以及联盟结构变化可能引入的成本(能量损耗)，将服务器间的动态协作过程转化为引入成本的马尔可夫动态决策过程(CT-MDP)。

2)考虑到在MEC环境下，边缘联盟执行任务的效率会受到处于联盟中的每一个边缘服务器的最大计算能力和最大存储空间等资源约束，将神经网络中的激活函数替换为ReLU6，使神经网络在保持较高计算效率的同时，减少对存储资源的需求。

3)在利用DQN算法对模型进行求解的过程中，出现了Q值过高估计的现象。针对该问题，提出一种基于双重DQN(DDQN)框架的深度强化学习(DRL)算法，旨在优化设备计算资源利用率的同时高效地搜索最优联盟结构。

4)通过仿真实验将所提出的算法与Q-learning算法[16]、DQN算法、Dueling DQN的求解方法进行了对比，结果表明，本文方法能够避免训练过程中Q值过高，同时，随着任务数量的逐渐增加，本文方法仍具备较好的收敛性和资源利用率，在减少构建联盟成本开销的同时能够确保任务高效完成。

1 边缘联盟模型构建

1.1 系统架构分析

边缘联盟的系统架构如图1所示。在完整的

计算机工程 www.ecice06.com

MEC 系统中,网络环境主要分为 3 层:用户层,边缘层,云层。用户层由大量异构的智能设备组成,这些设备动态地向边缘层中的 ESP 提出高性能服务的请求;边缘层主要由一系列的边缘服务器构成,主要为 ESP 提供计算和存储资源;云层为用户提供同样类型的服务以及更庞大的计算资源。首先,在边缘层中,每个边缘节点往往被独立部署在网络的边缘用于处理不同的业务。其次,由于边缘节点在地理位置上分布较为分散,因此在现有的边缘计算中,分属不同边缘基础设备供应商(EIP)管理的边缘节点之间不存在合作关系。与云相比,单个边缘节点的计算容量和服务范围远小得多,因而当需要处理更大资源规模需求的任务时,在新的网络位置部署更多的边缘节点成为 EIP 的主流选择[17]。而通过这种方法会造成诸如基础设施成本、维护成本、能源成本等一系列额外的开销和资源浪费,而边缘联盟概念的提出能够有效解决上述问题。如图 1 所示,在边缘层中,通过引入一个集中协调器来对散落的边缘节点进行协调部署,当接收到用户层发来的任务请求时,协调器根据收集到的用户信息,例如所在位置、时间、服务类型等,来计算调度所需的流量,并据此对适配的服务器进行调度。为直观地表示边缘联盟工作的过程,本文为任务列表中的任务增加了不同的工作负载,并且需要相对应数量的服务器来执行任务。例如在图 1 中,任务 1 需要 3 个负载,相对应地,该任务需要 3 台服务器协同进行处理;而当需要处理任务 2 时,集中协调器还会根据服务器的占用率来重构边缘联盟,以此来减少能量的消耗。

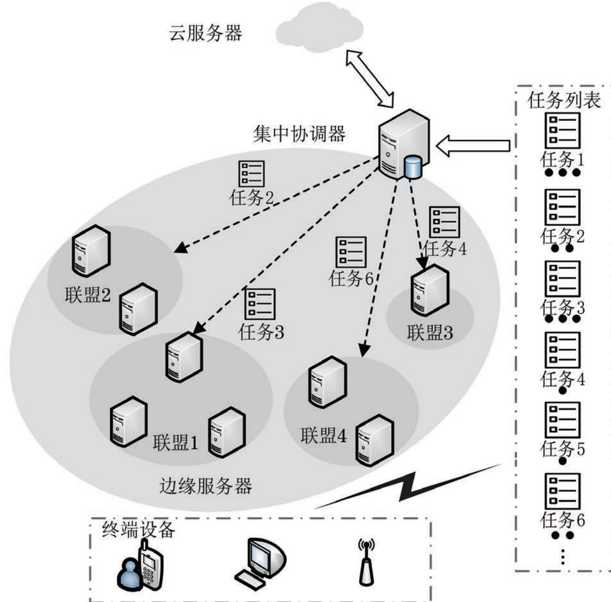

图 1 系统架构

Fig. 1 System architecture

1.2 引入成本的马尔可夫动态决策过程

本文拟将边缘联盟结构生成策略的问题建模为 CT-MDP,CT-MDP 中的符号定义如表 1 所示。

表 1 符号定义

Table 1 Symbol definition

符号	含义
S	状态空间
$\boldsymbol{S}_n^{\mathrm{ser}}$	服务器的状态
ty_n	可执行的任务
$\mathrm{S}^{\mathrm{ty}_n}$	任务集的状态
s'	转移到下一状态
$o_n^{\mathrm{or}}[t]$	服务器 n 的 CPU 占用率
l_n	每个 CPU 周期可执行的工作负载
k_n	能量成本系数
$A^{\mathrm{c}}(a^{\mathrm{c}})$	动作集合
T_s	概率转移函数
T_n	任务集状态的概率转移函数
$R(s,a,s')$	奖励函数
E	组建联盟和执行任务的能耗
N_G^{cs}	同一联盟中的成员
$c^{\mathrm{cost}}(C_t)$	成本函数
π_{cs}	形成联盟结构 cs 的策略
$\pi_a(\mathrm{cs},a)$	联盟采取动作的策略
R	即时奖励
γ	折扣因子
G_t	回报
J_t	目标函数
ω	用于评估成本的权重
π^{e}	等效策略
$Q^{\pi^{\mathrm{e}}}$	状态-价值函数
Q^{c}	成本-价值函数

本文将参与任务处理的每个边缘服务器视作一个智能体,并用一个集合 $N=\{1,2,\cdots,n\}$ 表示,其中,n 表示服务器的数量。每个智能体可以自主选择构建联盟的方式,既可以独立形成一个联盟,亦可以加入其他智能体的联盟。值得注意的是,每个智能体只能加入一个联盟,直至该联盟解散,由此形成的联盟结构记为 cs,如式(1)所示:

$$\mathrm{cs}=\left\{c_i \mid (\forall i\neq j, c_i=\varnothing)\wedge\left(\bigcup_{i=1}^{|\mathrm{cs}|} c_i=N\right)\right\} \tag{1}$$

而所有可能构成的联盟结构可用 P^N 表示:$P^N=\{\mathrm{cs}_1,\mathrm{cs}_2,\cdots,\mathrm{cs}_{|P^N|}\}$。

MDP 是描述离散时间决策过程的传统模型,常用于模拟智能体的动态行为[18]。在 MDP 中,做出决策的主体被称为智能体,而智能体需要在环境中通过观察获取相应的状态 S(其中 s 属于可观察状

计算机工程 www.ecice06.com

态的集合)，并从动作集合中选择行动 A(其中 a 属于动作的集合)。然后，根据概率转移函数，环境会以此概率转移到下一个状态 s'。随着状态的转换，智能体根据奖励函数 r 得到奖励 $r(s, a, s')$。而智能体的最终目标是学习到一个最优策略，通过在一段时间内累积的奖励值来衡量策略的优良性。

1.2.1 状态空间与动作空间的定义

状态空间 S 由两个部分组成：服务器的状态 S_n^{ser} 和任务集的状态 S^{ty_m}。服务器的状态反映了在每个时间步 t 的一系列参数，由一个三元向量进行表示，如式(2)所示：

$$\boldsymbol{S}_n^{ser} = [r_n^{or}[t], l_n, k_n] \tag{2}$$

式中：$r_n^{or}[t]$ 表示服务器 n 的 CPU 占用率[19]，可用服务器 n 上分配的任务数量来进行近似计算，如式(3)所示；l_n 表示服务器在每个 CPU 周期所能执行的工作负载；k_n 代表服务器的能量成本系数。任务集的状态描述了任务集合中每种任务类型的剩余数量，可以表示为 S^{ty_m}，其中，m 为任务类型，表示处理该任务所需负载的数量为 m。

$$r_n^{or} = \frac{N_{max}}{N'_{max}} \tag{3}$$

式中：N_{max} 表示服务器 n 上已分配的任务数量；N'_{max} 表示服务器 n 可处理的最大任务数量。

完整的状态空间定义如式(4)所示：

$$S = \{\boldsymbol{S}_n^{ser}, S^{ty_m}\} \tag{4}$$

CT-MDP 的动作空间是一个离散的动作空间，定义为一个动作集合 A^c，其中包括了每个边缘联盟 c 能够执行的任务或选择不执行任务的动作。可执行的任务用 ty_n 进行表示，其中动作集合 $a^c = \{ty_1, ty_2, \cdots, ty_n\}$ 表示不同的任务类型；而用 $a^c = 0$ 来表示不执行任何任务。因此，联盟的动作集合 A^c 可定义为：

$$A^c = \{ty_n \mid n = |c|\} \cup \{0\} \tag{5}$$

式中：动作集中的动作数量取决于联盟中的成员数量和任务类型的数量。

概率转移函数定义了在执行动作后系统状态的变化规律。考虑到状态空间的定义，本文更需要关注服务器的状态和任务集的状态在执行动作后的演变。

首先，所处联盟中的服务器状态会受到动作的影响。执行任务时，CPU 占用率和能量成本系数都可能发生变化。假设当前时刻的服务器状态为 s_t，执行动作后，可以通过概率转移函数来计算下一时刻的服务器状态。概率转移函数的定义如式(6)所示：

$$s_{t+1} = T_s(s_t, a_t) \tag{6}$$

概率转移函数 T_s 的具体形式可以根据系统的具体特性而定。

其次，任务集的状态也会发生改变，特别是任务被执行后，任务类型的剩余数量减少。假设当前时刻的任务集状态为 $n_t = \{ty_{n,t}\}$，执行动作 a_t 后，可以通过概率转移函数 T_n 来计算下一时刻的任务集状态：

$$n_{t+1} = T_n(n_t, a_t) \tag{7}$$

式中：T_n 表示任务集状态的概率转移函数。

综上，概率转移函数考虑了动作的执行对服务器状态和任务集状态的影响。在系统的每个决策时间步骤中，根据当前状态和执行的动作，利用概率转移函数可以预测下一时刻的状态。

1.2.2 奖励与成本函数的定义

在经典的 MDP 模型中，奖励函数 $R(s, a, s')$ 往往被定义为一个映射，其中，s 是当前状态，a 是智能体采取的动作，s' 是转移到的下一个状态，函数表示在某一状态下执行某一动作后获得的即时奖励，目的在于让智能体通过学习一个最优策略，使累计奖励最大化。

本文对奖励函数的设置旨在尽快完成所有任务的同时最小化能耗和联盟结构变动所产生的成本，因此，当所有任务都被完成时，会给予一个较大的正奖励(奖励为 100)。如果所有任务均已被完成，计算奖励时则需要减去组建联盟以及执行任务时的能耗 E，可表示为 $-E+100$。反之，当任务并未被全部完成时，奖励为 $-E$。关于能耗 E 的计算可用式(8)表示：

$$E = \sum_n k_n \times o_n^{or}[t] \tag{8}$$

奖励函数 r 的定义如式(9)所示：

$$R(s, a^c, s') = \begin{cases} -E+100, & s' = s_{term} \\ -E, & s' \neq s_{term} \end{cases} \tag{9}$$

区别于传统的 MDP，本文对成本函数进行了定义，用于表示联盟结构改变引入的成本[20]，其由一定时间内联盟结构中智能体的合作模式改变情况所决定。考虑到描述成本时需要关注联盟成员之间的相互关系和联盟的形成过程，而图的连通性可以有效地表示联盟中成员之间的交互，且通过可视化的方式使得联盟结构的理解和分析更加直观。本文将联盟结构表示为多个无向图。在无向图中，每个智能体(边缘服务器)都被表示为一个顶点，任意两个智能体可以共享一条边。而改变联盟结构意味着添加或删除无向图的边。因此，改变联盟结构的成本可以通过无向图中边变化的数量来进行计算。

计算机工程 www.ecice06.com

首先通过集合 N_G^{cs} 来定义无向图中每个智能体(顶点)的邻居，即同一联盟中的成员，如式(10)所示：

$$N_G^{\mathrm{cs}}=\{N_G^{\mathrm{cs}}(i)=\{k \mid k,i\in \mathrm{c}_j\in \mathrm{cs}\wedge k\neq i\}\mid \forall i\in N\} \tag{10}$$

设置集合 $N_G^{P^N}$ 表示所有可能的 N_G^{cs}，利用 $D(N_G^{\mathrm{cs}},N_G^{\mathrm{cs}'})$ 来量化联盟结构 N_G^{cs} 与 $N_G^{\mathrm{cs}'}$ 之间的差异，具体计算方式如式(11)所示：

$$D(N_G^{\mathrm{cs}},N_G^{\mathrm{cs}'})=\frac{1}{2}\sum_i \mid N_G^{\mathrm{cs}}(i)\oplus N_G^{\mathrm{cs}'}(i)\mid \tag{11}$$

该式通过对称差分运算，能够评估无向图中边变化的数量，从而获取联盟结构的具体变化。那么成本函数可定义为式(12)。通过最小化成本函数有助于选择更为经济高效的联盟结构。

$$c^{\mathrm{cost}}(s,\mathrm{cs},s',\mathrm{cs}')=-D(N_G^{\mathrm{cs}},N_G^{\mathrm{cs}'}) \tag{12}$$

传统 MDP 对于策略的描述是智能体在特定状态下采取不同动作的决策，而根据本文对动作空间的定义，采取行动的主体是智能体间形成的联盟，因此对于策略的定义需要包含两个部分，即智能体间形成联盟的决策过程和联盟采取动作的决策过程。

对于智能体间形成联盟的决策阶段，本文引入了策略 $\pi_{\mathrm{cs}}\rightarrow[0,1]$ 将状态空间 S 和联盟结构集合 P^N 映射至$[0,1]$，表示在状态 s 下形成联盟结构 cs 的概率。引入策略 $\pi_a(\mathrm{cs},a)=P(a\mid \mathrm{cs})\rightarrow[0,1]$ 表示联盟采取动作的概率。在每个联盟执行动作后，系统根据概率转移函数更新到下一个状态 s_{t+1}，在此过程中，每个智能体的目标是通过学习到一个最优的策略，来最大化在这一段时间内的累计奖励。通过不断调整策略，智能体可以在 MEC 这种复杂环境中构成高效的联盟结构与任务执行决策，以此提高整体系统的性能。

1.3 目标函数与价值函数

综上，CT-MDP 可以用一个六元组进行表示：$\langle N,S,A^{\mathrm{c}},T,R,c^{\mathrm{cost}}\rangle$。在 CT-MDP 中，需要利用价值函数来衡量联盟结构和任务执行策略在某一状态下的长期回报[21]。

1)目标函数。

基于上述动态联盟的形成过程以及对即时奖励 R 的定义，本文对时间间隔 t 内的回报 G_t 定义如式(13)所示：

$$G_t=R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\cdots=\sum_{k=1}^{T}\gamma^{k-1}R_{t+k}=\sum_{k=1}^{T}\gamma^{k-1}R(s_{[k]},a_{[k]}^{\mathrm{c}},s_{[k+1]}),\quad 0\leqslant\gamma\leqslant 1 \tag{13}$$

式中：$\gamma\in[0,1]$ 表示折扣因子，作用是体现未来奖励对当前决策的重要性。具体而言，将智能体在未来所获取的奖励折现到当前的时刻进行计算，且距离当前时刻越远的奖励，折现到当前时刻的奖励越小。在智能体进行学习的回合中，对于联盟结构改变的成本 C_t 计算方式如下：

$$C_t=c_{t+1}^{\mathrm{cost}}+\gamma c_{t+2}^{\mathrm{cost}}+\gamma^2 c_{t+3}^{\mathrm{cost}}+\cdots+\sum_{k=1}^{T}\gamma^{k-1}c_{t+k}^{\mathrm{cost}}=\sum_{k=1}^{T}\gamma^{k-1}c^{\mathrm{cost}}(s_{[k]},\mathrm{cs}_{[k]},s_{[k+1]},\mathrm{cs}_{[k+1]}),\quad 0\leqslant\gamma\leqslant 1 \tag{14}$$

根据预期回报和成本的计算方式，目标函数 J_t 的定义如式(15)所示：

$$J_t=G_t+\omega C_t \tag{15}$$

式中：ω 作为评估成本重要性的权重。在考虑最大化期望累计奖励的同时，应考虑改变联盟结构而引入的代价，因此，本文利用长期奖励和成本的加权和作为 CT-MDP 的目标函数。

而在联盟结构生成问题中，更希望在找到最优联盟结构的同时，任务能够在最短的时间内完成，即寻求最佳的任务执行策略，以此使系统性能最优。而根据本文对于策略的定义，两者分属于两种不同的策略，为了在两种策略之间找到平衡并实现共同优化，本文提出了一种等效策略 π^{e}。考虑到策略 π_a 是在确定联盟结构之后为执行任务做出的决策，因此可以将策略 π_{cs} 作为 π_a 的先决条件，等效策略定义如式(16)所示：

$$\pi^{\mathrm{e}}(s,a)=\pi_{\mathrm{cs}}(s,\mathrm{cs})\pi_a(\mathrm{cs},a_{\mathrm{cs}}) \tag{16}$$

在给定状态 s 和联盟结构 cs 的情况下，采取动作 a 的概率可以由两者乘积得到。π_{cs} 以及 π_a 由 π^{e} 表示，如式(17)和式(18)所示：

$$\pi_{\mathrm{cs}}(s,\mathrm{cs})=\sum_{a\in A_{\mathrm{cs}}}\pi^{\mathrm{e}}(s,a) \tag{17}$$

$$\pi_a(s,\mathrm{cs},a_{\mathrm{cs}})=\pi_{\mathrm{cs}(s,\mathrm{cs})}\frac{\pi^{\mathrm{e}}(s,a_{\mathrm{cs}})}{\sum_{a'\in A_{\mathrm{all}}}\pi^{\mathrm{e}}(s,a')} \tag{18}$$

式中：A_{cs} 和 A_{all} 分别表示构建联盟的动作集合以及所有可能的联盟结构能采取的动作集合。

综上，对于最优策略 $\pi^{\mathrm{e}*}=\underset{\pi^{\mathrm{e}}}{\operatorname{argmax}}\,E$ 的定义如式(19)所示：

$$\pi^{\mathrm{e}*}=\underset{\pi^{\mathrm{e}}}{\operatorname{argmax}}\,\underset{P^{\pi^{\mathrm{e}}}}{E}\,J \tag{19}$$

2)价值函数。

最优策略的确立往往需要利用价值函数来进行衡量。通过迭代更新价值函数，当价值函数最大化

计算机工程 www.ecice06.com

时，其所对应的即是最优策略。MDP中通常利用状态-动作价值函数 $Q^{\pi^e}(s,a)$ 来表示遵循策略 π 时当前状态 s 下执行动作 a 得到的期望回报，其定义过程如式(20)所示：

$$Q^{\pi^e}(s,a)=\underset{\pi^e}{E}[G_t|S_t=s,A_t=a]$$
$$V^{\pi^e}(s)=\sum_{a\in A_{\text{all}}}\pi(a|s)Q^{\pi^e}(s,a)$$
$$Q^{\pi^e}(s,a)=r(s,a)+\gamma\sum_{s'\in S}T(s'|s,a)V^{\pi^e}(s') \tag{20}$$

同样地，需要定义成本价值函数 Q^c 用于评估联盟结构改变的成本对于最优策略的影响，具体定义过程如式(21)所示：

$$Q^c(s,a)=\underset{\pi^e}{E}[c_t^{\text{cost}}|S_t=s,A_t=a]$$
$$V^c(s)=\sum_{a\in A_{\text{all}}}\pi(a|s)Q^c(s,a)$$
$$Q^c(s,a)=$$
$$c^{\text{cost}}(s,\text{cs},s',\text{cs}')+\gamma\sum_{s'\in S}T(s'|s,a)V^c(s') \tag{21}$$

基于上述定义过程，引入成本的状态-动作价值函数 Q^* 可定义为式(22)：

$$Q^*(s,a)=Q^{\pi^e}(s,a)+\omega Q^c(s,a)$$
$$Q^*(s,a)=r(s,a)+\omega c^{\text{cost}}(s,\text{cs},s',\text{cs}')+$$
$$\gamma\sum_{s'\in S}T(s'|s,a)[V^{\pi^e}(s')+\omega V^c(s')] \tag{22}$$

通过对 $Q^*(s,a)$ 直接求解能够对不同策略进行评估，求解结果中令 J_t 最大的策略即为最优策略。

2 算法设计

深度强化学习结合了深度学习的感知能力和强化学习的决策能力，使智能体能够在复杂环境中通过与环境的交互自主学习最优策略[22]。在DRL中，智能体通过探索和利用环境来最大化累计奖励。深度学习在此过程中用于处理高维输入数据，而强化学习则负责根据奖励信号更新智能体的策略。本文利用DDQN算法对上述联盟结构生成问题进行求解。

2.1 算法结构设计与激活函数的选取：

根据式(22)，传统的DQN算法优化的时序差分(TD)误差如式(23)所示：

$$L^{\text{DQN}}(\theta)=E[Q_{\max}^*(s,a;\theta)-Q^*(s,a;\theta^-)^2] \tag{23}$$

式中：$Q_{\max}^*(s,a;\theta)$ 表示为式(24)。

$$Q_{\max}^*(s,a;\theta)=r(s,a)+\omega c^{\text{cost}}(s,\text{cs},s',\text{cs}')+$$
$$\gamma\max Q^*(s',a';\theta^-) \tag{24}$$

在式(24)中，max的操作可拆解为两个部分：选取状态 s' 下的最优动作 $a^*=\text{argmax}\,Q^*(s',a')$，以及计算该动作对应的价值 $Q^*(s,a^*)$。当两个部分采用同一套Q网络进行计算时，每次得到神经网络中当前估算的所有动作价值的最大值，因而在DQN的更新过程中神经网络会正向累计误差，从而会产生对 Q 值的过高估计，影响训练效果。

DDQN算法的提出能够很好地解决这一问题。考虑在DQN算法中，存在两套独立的Q网络：目标网络和训练网络，因此本文将训练网络用于对动作进行选取，并将目标网络用来计算选取该动作下对应的 Q 值。DDQN算法框架如图2所示。

目标网络与训练网络都利用相同结构的深度神经网络来对 Q 值进行计算。如图2所示，在DNN中，输入层的维度与状态空间大小匹配，用于接收环境的状态；隐藏层由两个全连接层构成，每层包含128个神经元。在激活函数的选择上，考虑到在移动端设备和边缘设备中，计算资源往往受到限制，而低精度计算能够在加快处理速度的同时，满足实时性应用的需求。因而相比于传统DQN中所使用的ReLU函数，本文所引入的ReLU6[23]函数在正输入区域提供了一个梯度裁剪的机制，函数表达式如式(25)所示。将输出值限制在[0,6]的范围内，对输出范围进行约束，能够在一定程度上加快训练收敛速度，同时减少因精度损失带来的影响。

$$f(x)=\text{ReLU6}(x)=\min(\max(x,0),6) \tag{25}$$

误差函数 $L^{\text{DDQN}}(\theta)$ 定义过程如下：

$$Q_{\max}^{\text{DDQN}}=$$
$$r(s,a)+\gamma\max_{a'}(\omega c^{\text{cost}}+Q^*(s',a';\theta))=$$
$$r(s,a)+\gamma Q^*(s,\text{argmax}\,Q^*(s',a';\theta);\theta^-) \tag{26}$$

$$L^{\text{DDQN}}(\theta)=E[(Q_{\max}^{\text{DDQN}}-Q^*(s,a;\theta))^2] \tag{27}$$

2.2 基于DDQN的轻量化算法

DDQN算法包括训练和运行两个部分。算法1和算法2展示了DDQN的整体流程。在训练过程中，将智能体在环境中获取的状态作为当前Q网络的输入，选择动作则采用ε-greedy策略，以(1－ε)，ε∈[0,1]的概率选取 Q 值最大的动作，以ε的概率随机选取动作。在得到奖励后，将获取的系列经验 (s_t,a_t,r_t,s_{t+1}) 存放至经验回放池内，当达到最小经验回放缓冲区尺寸时，每次训练会从回放池中随机采样来对网络进行训练，并以步长 C 更新网络参数。区别于训练过程，在算法的运行过程中，选取动作使用贪心策略，即选取最大的 Q 值所对应的动作。

计算机工程 www.ecice06.com

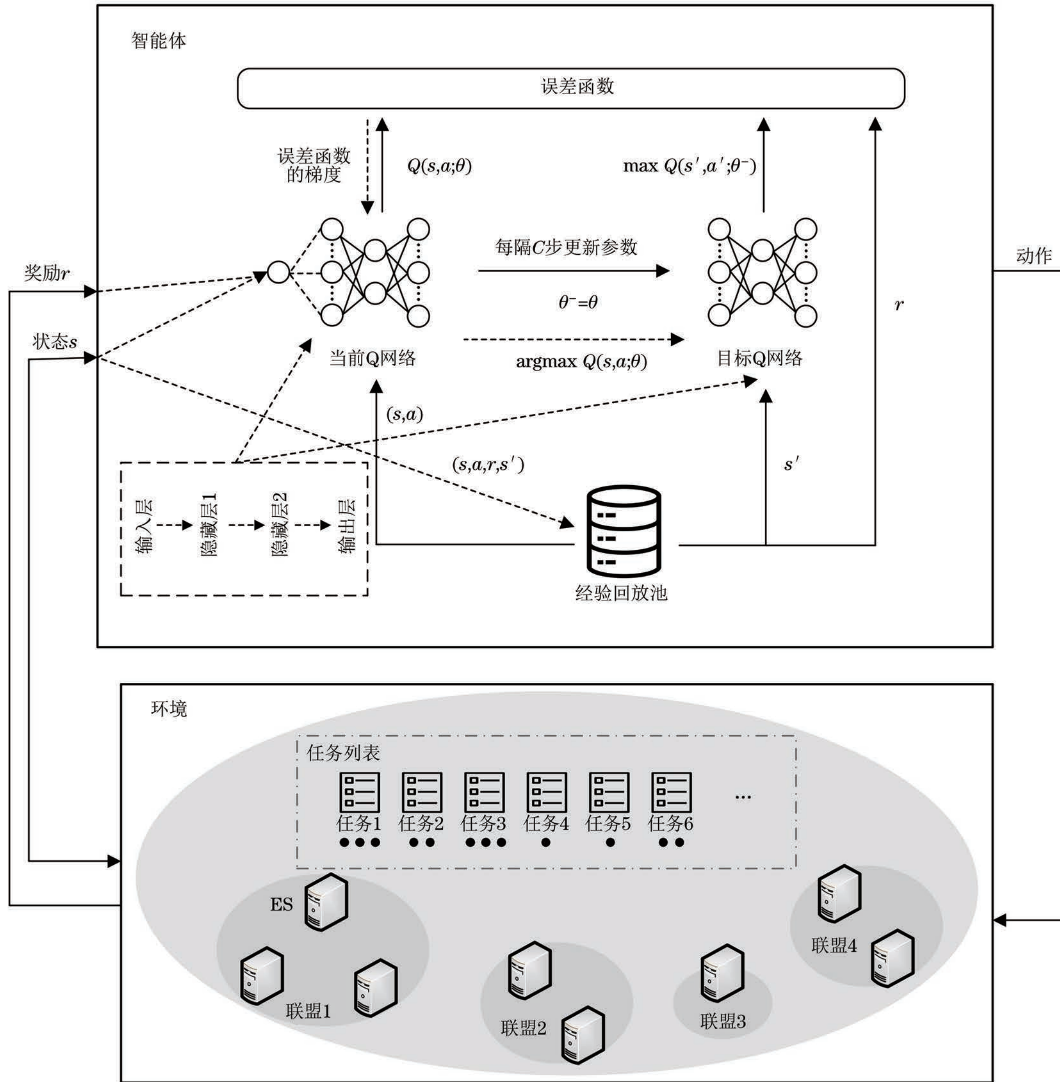

图 2　DDQN 算法框架

Fig.2　The framework of DDQN algorithm

算法 1　基于改进 DDQN 算法的训练流程

输入　系统环境参数和 DDQN 算法参数

输出　评估 Q 网络的参数 θ

初始化经验回放池；

利用随机参数 θ 初始化评估 Q 网络 ϕ，令 $\theta^{-}=\theta$ 初始化目标 Q 网络 $\hat{\phi}$；

for episode＝1：Maxepisode do

初始化状态 s_t，包括服务器的占用率以及任务列表；

for t＝1：Maxstep do

利用 ε-greedy 选择动作 a^c 并执行；

获取奖励 r 并计算成本 c^{cost}，并转移至下一个状态 s_{t+1}；

存储(s_t，a_t，r_t，s_{t+1})到经验回放池内；更新当前状态 $s_t=s_{t+1}$；

从回放池中随机取样数量为 batch 的(s_t，a_t，r_t，s_{t+1})；

根据式(22)对获取的(s_t，a_t，r_t，s_{t+1})计算目标 Q 值；根据式(26)、式(27)计算误差并更新训练网络中的参数 θ；

每隔 C 步更新目标 Q 网络：$\theta^{-}=\theta$；

end for

end for

算法 2　基于改进 DDQN 算法的运行流程

输入　系统环境参数、DDQN 算法参数、评估 Q 网络的参数 θ 和当前状态 s_t

输出　最终状态 $s_{\text{Maxstep}+1}$

for t＝1：Maxstep do

利用评估网络选取动作：$a_t=\underset{a^c}{\operatorname{argmax}}\, Q(s_t, a^c;\theta)$；

if 状态不为最终状态 then

计算奖励 r、成本 c^{cost}；

转移至下一个状态 s_{t+1}；

else

计算累计奖励 r、成本 c^{cost} 与目标函数 J_t；

end if

end for

3　实验与分析

为验证本文所提出的算法的有效性，本节对基

于轻量化 DDQN 的联盟结构策略优化方法进行了仿真实验。首先，对仿真场景进行详细说明；然后，通过仿真结果进行总结与分析，得出结论。

3.1 仿真场景与仿真参数设置

考虑本文在状态空间的表示中引入了 CPU 的占用率，相对应地，每个服务器的状态可通过 CPU 占用率的百分比划分进行表示，因而每个服务器分别对应着维度为 100 的状态空间。因此假设 2 台服务器构成的联盟，其对应 10^4 个状态，在不降低模型有效性的前提下，考虑到高维状态空间可能对模型性能和计算效率产生影响，本文对 MEC 系统参数的设置进行了简化[1]。考虑一个由 3 台服务器组成的边缘服务器，并在服务器的状态设置中令 $l_n=1$，$k_n=1$，即在每个 CPU 周期中每个服务器的工作负载为 1，服务器的能量成本系数为 1。

在任务集的设置中，考虑到边缘服务器在实际部署中遇到的计算需求，在确保任务集的规模和复杂性与边缘服务器的资源限制相匹配的前提下，将 1.0×10^8 个 CPU 周期作为一个工作负载，并设置了 3 种不同负载需求的任务，3 种类型的任务分别对应 1.0×10^8、2.0×10^8 和 3.0×10^8 个 CPU 周期的工作负载，与之对应则分别需要 1、2 和 3 台服务器参与联盟协同处理，为保证一般性，任务集中不同类型的任务数量将随机生成。

本文利用 Python 3.9 实现和训练 DDQN 的轻量化算法。在算法的参数设置中，每 10 次迭代后更新目标网络的参数（$C=10$），每次从经验池中采样数量（batch）为 64，折扣因子 $\gamma=0.98$，而学习率的设置很大程度上会影响模型的性能，在开展其他实验前需要确定一个合适的学习率以确保 DDQN 算法能够有效收敛并学习到高性能策略[24]。考虑到选取的学习率应尽可能覆盖一个数量级的范围，以考察模型在不同学习率下的表现，本文选取了 0.002、0.02、0.01、0.005 这 4 个强化学习中较为常用的学习率开展实验。

图 3 为学习率分别取值 0.002、0.02、0.01、0.005 的训练曲线。可见当选取的学习率较大时（$\alpha=0.01$，$\alpha=0.02$），对应的训练曲线均出现了不同幅度的波动，表明在 $\alpha=0.01$，$\alpha=0.02$ 的学习率下训练的稳定性较低；而当选取的学习率为 0.005 时，对应的训练曲线在步数为 100～200 之间均出现了不同幅度的波动，并且在达到收敛后仍出现了最优解振荡的情况。反观学习率为 0.002 的曲线相比于其他 3 种在训练过程中最为平稳，且以最快速度达到收敛水平，在收敛后更为平稳，无明显的大幅度波动。因此，在后续实验中确定学习率 $\alpha=0.002$。

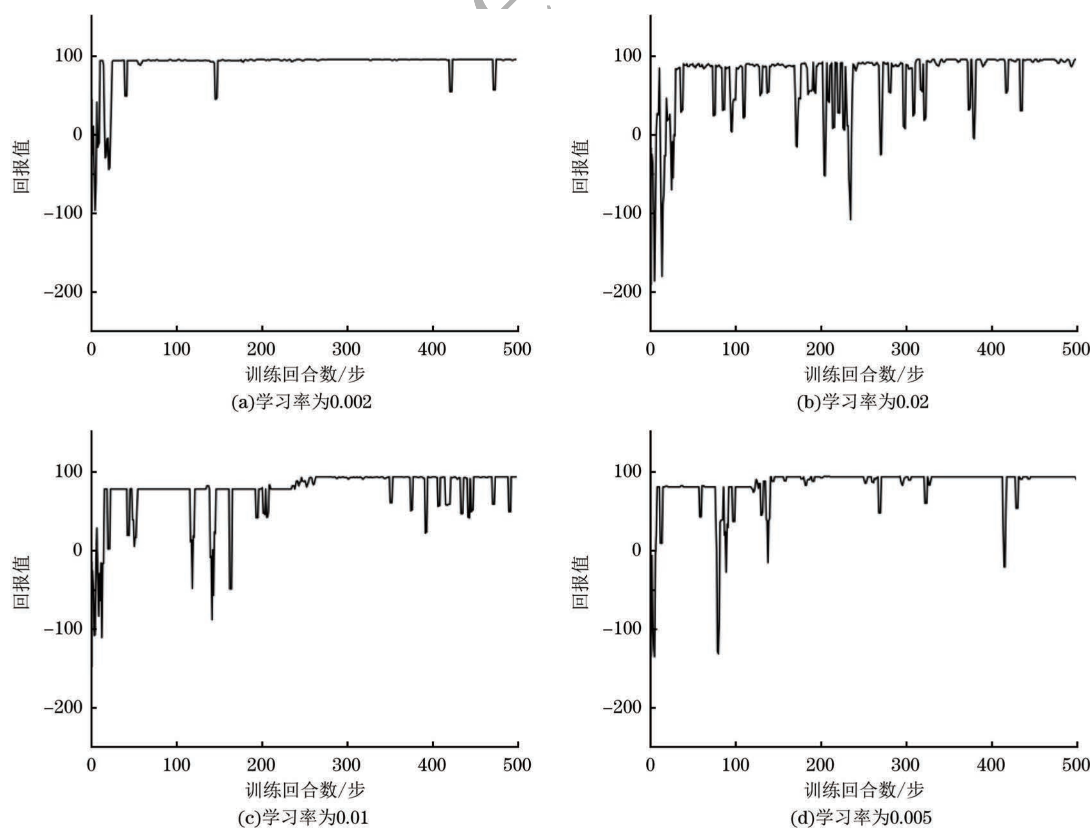

图 3 不同学习率下的 DDQN 训练表现

Fig.3 DDQN training performance at different learning rates

计算机工程 ecice06.com

此外，在训练开始前，经验回放缓冲区内必须积累到足够的样本数量，来确保智能体所学习的经验具备数据多样性和代表性。因此，还需要对算法中最小经验回放缓冲区尺寸 minimal size 进行确定。

图 4 为最小经验回放缓冲区尺寸分别取值为 400、450、475、500 的训练曲线。可以看出，取值为 400、475、500 的曲线在完整的训练过程中均出现了不同幅度的波动；取值为 400 和 450 的曲线以较快的速度达到了收敛水平，且回报值也会略高于其他取值，这是因为较小的缓冲区尺寸会加快训练速度，且样本多样性不足会导致学习过程陷入局部最优解，后续出现的不同幅度的波动表明学习质量的下降；相比之下，最小经验回放缓冲区尺寸为 450 的曲线在整体训练过程中较为平稳，且达到收敛水平后无较为明显的波动。因此，在后续实验中确定最小经验回放缓冲区尺寸为 475。

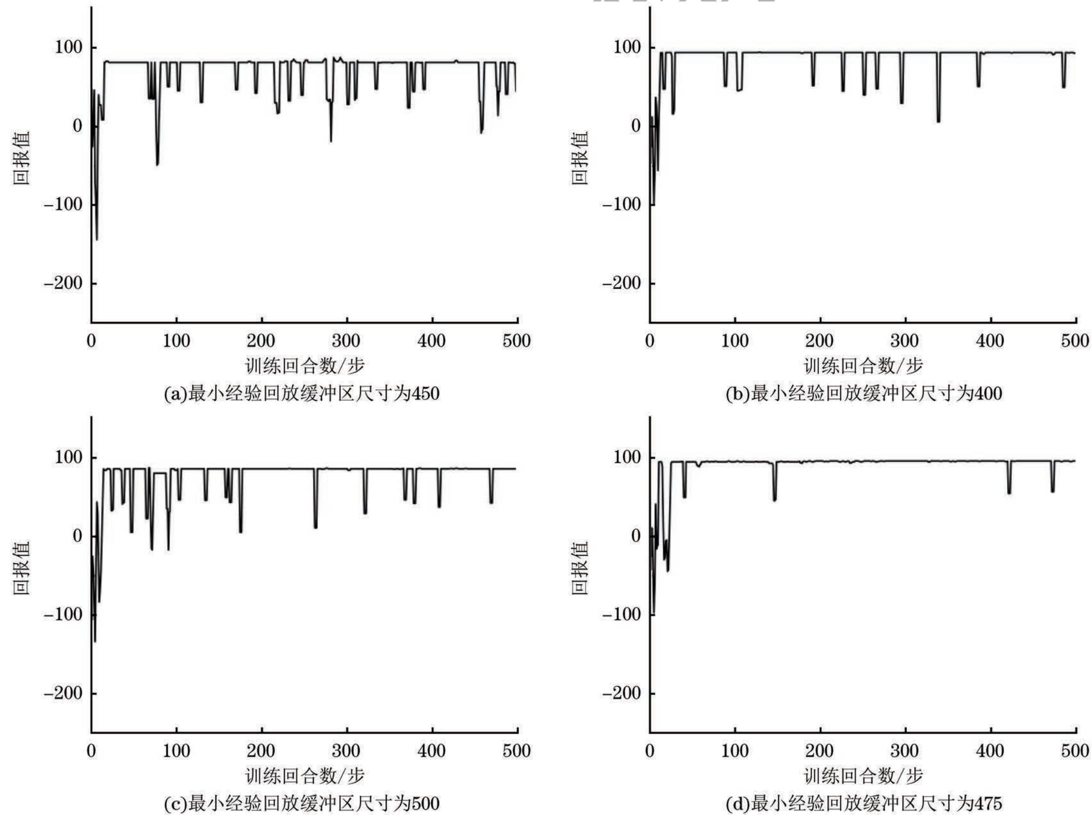

图 4 不同最小经验回放缓冲区尺寸下的 DDQN 训练表现

Fig. 4 DDQN training performance under different minimum experience replay buffer sizes

3.2 算法收敛性

本文将 QL 算法[12]、DQN 算法[13]以及 Dueling DQN 算法作为对比算法，来验证本文所提求解方法的有效性。

图 5 给出了本文算法与 DQN 两种算法在训练过程中 Q 值的变化情况。起初，两种算法的 Q 值都随着采取动作数量的增加而增加，而当采取了 5 000 次动作后，DQN 算法的 Q 值逐步呈现出较大幅度的波动，且 Q 值远大于采取同样动作次数的本文算法。最终两者在采取 25 000 次的动作后逐渐收敛。且在训练过程中，本文算法的 Q 值出现波动的幅度远小于 DQN，反映了在训练过程中 DQN 算法存在 Q 值过高估计的问题。

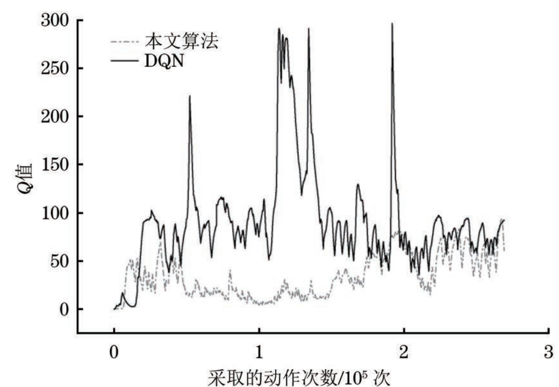

图 5 DDQN 算法和 DQN 算法训练中 Q 值的变化

Fig. 5 Changes in Q values during training of DDQN and DQN algorithms

图 6 展示了其他 4 种算法与本文算法收敛性的对比。在确立合适参数的情况下，相较于其他 4 种算法，Q-learning 需要较多的步数来搜索最优策略，

由此，当训练环境中存在庞大的状态空间，使用Q-learning算法的收敛性能远不如DRL中的其他算法。相比之下，DQN在训练初期能够得到正向的回报值，而考虑到训练中存在Q值过高估计的问题，其最终呈现出波动幅度较大的收敛趋势。本文算法和Dueling DQN同为改进的DQN算法，在状态空间维度较高的情况下，表现出了良好的收敛性。通过对Q值过高估计的改进，本文算法在训练回合数为75步左右达到收敛。而考虑到环境中动作空间维度较小的缘故，Dueling DQN算法中通过分离值函数和优势函数的设置所带来的性能提升有限，Dueling DQN算法收敛所需要的回合数略多于本文算法。A3C算法作为一种策略梯度算法，通过引入优势函数来减小策略梯度更新时出现的方差，表现为在训练初期能以很快速度达到收敛水平，且在训练后期抖动情况的出现大幅减少。但考虑A3C的多线程并行执行要求更多的计算资源，其性能优势在资源受限的环境中无法得到发挥。

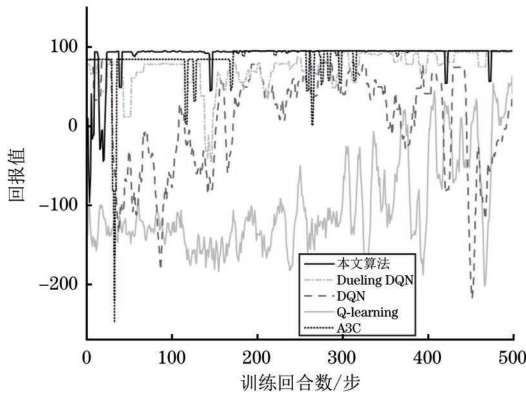

图6 不同算法的收敛性对比

Fig.6 Comparison of convergence of different algorithms

3.3 算法性能

图7给出了任务数量为5、10、15、20个时DQN算法和DDQN算法性能表现。由图7(a)可知，当需处理的任务数量较少时，DQN与DDQN均能在100个训练回合前达到收敛水平，表现出良好的算法性能。由图7(b)、图7(c)、图7(d)可知，随着任务数量的逐步增加，DQN算法与本文算法的训练曲线均出现了幅度不等的波动。当任务数量为5、10、20个时，本文算法和DQN算法的收敛步数逐渐趋于稳定，在状态与动作维度较高的情况下表现出了良好的稳定性。而在任务数量为15个时，DQN算法与本文算法分别在训练回合数250～350、100～200步之间出现了大幅度的波动，这可能是因为在随机生成任务列表中，需要频繁组建与解散联盟，导致在某些回合中会获得异常的奖励，从而影响回报值。

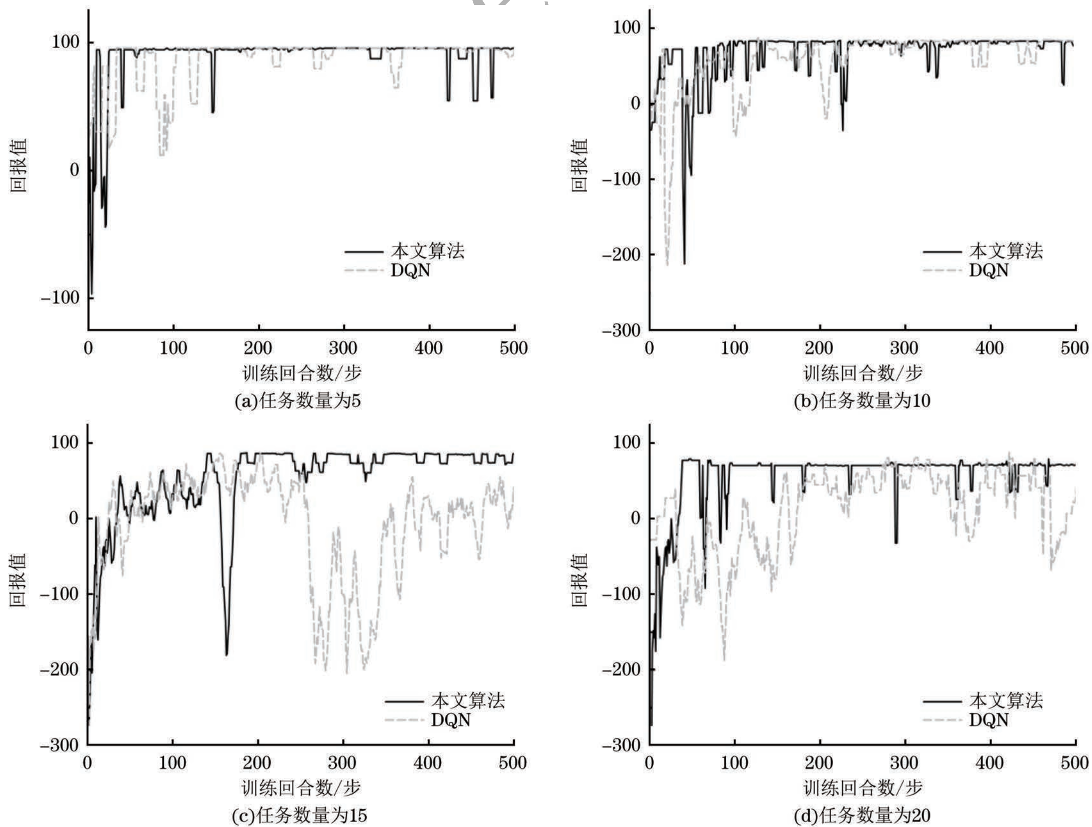

图7 不同任务数量下DQN和DDQN的算法性能

Fig.7 Performance of DQN and DDQN algorithms under different task quantities

www.ecice06.com 计算机工程

图 8 给出了对应任务数量下 DQN 算法和 DDQN 算法训练得到的最大回报值。在任务数量逐渐增加的情况下，由于联盟结构的成本也会随之增加，因此 DQN 算法与 DDQN 算法获取的回报值均逐步减少。而在任务数量较少的情况下 DQN 算法与本文算法均有良好的性能。当任务数量为 15 个时，本文算法获得 74.47 的最大回报值，高于 DQN 算法 15.8%；当任务数量为 20 个时，本文算法获得最大回报值为 71.55，高于 DQN 算法 24.91%。任务数量的变化对 DDQN 的回报值影响并不显著。综上所述，基于 DDQN 的联盟结构策略优化方法能够有效应对较多任务数量的情况。

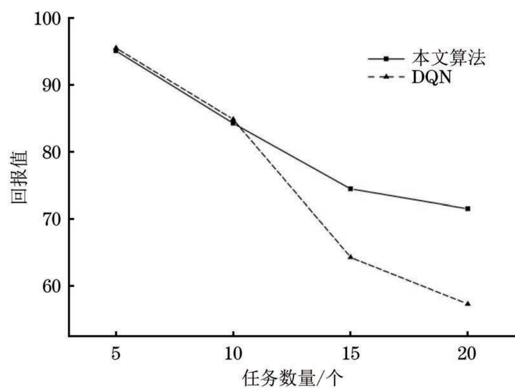

图 8 不同任务数量下 DQN 和 DDQN 的回报值对比

Fig. 8 Comparison of return values between DQN and DDQN under different task counts

图 9 所示为分别以 ReLU、ReLU6、Sigmoid 和 tanh 作为本文所提算法的激活函数在执行任务数量为 15 个情况下的训练表现。ReLU 和 ReLU6 由于其线性特性，在训练初期表现出较快的收敛速度，而 Sigmoid 和 tanh 因其饱和特性，在训练过程中的收敛速度相对较慢。由于 ReLU6 限制了输出值范围，减少了梯度爆炸的风险，加速了模型的收敛，具体表现在训练的初期，ReLU6 的收敛速度要略快于 ReLU。相比之下，Sigmoid 和 tanh 需要较多的步数才能达到收敛水平，尽管在训练后期 4 种函数均有较好的训练效果，但 ReLU 和 ReLU6 在有较好的表现水平的同时能够加快处理的速度，更适合本文所研究场景的需要。

为评估不同激活函数对计算资源需求的影响，本文在 PC 环境下综合比较了 ReLU、ReLU6、Sigmoid 和 tanh 这 4 种激活函数在 DDQN 网络中的表现，图 10 为训练过程中在 PC 上监测到的 CPU 占用率情况。在配置 8 核 CPU 的 PC 环境中，相较于其他 3 种激活函数，以 ReLU6 作为激活函数的 DDQN 算法训练所需内存占用率总体处于较低水

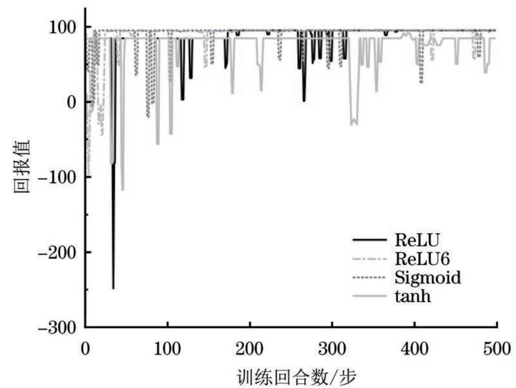

图 9 4 种激活函数下的训练曲线

Fig. 9 Training curves under four activation functions

平。可以发现在第 1 个和第 5 个核上，Sigmoid 和 tanh 函数分别出现了内存占用率低于其余 3 种激活函数的情况，但两者的平均内存占用率都要高于 ReLU6 函数，原因在于 Sigmoid 和 tanh 激活函数都涉及指数运算，而较高的计算复杂度会在资源受限的环境中引起不必要的计算开销，尤其是在边缘计算中可能会显著增加计算负担。在第 2～7 个核中，ReLU6 激活函数的内存占用率分别少于 ReLU 激活函数 27.52%、5.76%、15.43%、22.80%、36.31%、12.68%，最终 ReLU6 的平均内存占用率要少于 ReLU 12.12%。可见，在边缘设备这种资源受限的环境下部署联盟结构优化策略，使用 ReLU6 作为激活函数是一个更优的选择。

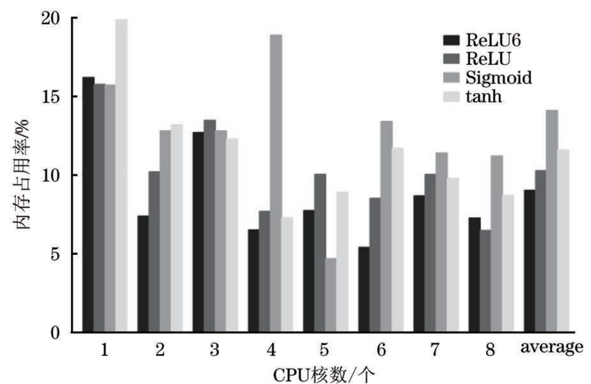

图 10 不同核数下 CPU 内存占用率情况

Fig. 10 CPU memory usage under different numbers of cores

综上，本文通过上述的仿真实验，验证了基于改进 DQN 的求解方法对于最优联盟策略优化问题求解的有效性，相比于 Q-learning、DQN 算法，轻量化 DDQN 算法在训练过程能够以较快速度达到收敛水平。当任务数量较多时，相比于 DQN 算法，DDQN 算法收敛所需步数优势明显，当任务数量发生变化时，DDQN 能够维持相对稳定的性能表现；且使用 ReLU6 作为 DDQN 算法的激活函数能够在保持算法性能的基础上有效降低部署设备的内存占用率。

www.ecice06.com 工程

4 结束语

本文对边缘计算系统这一动态环境下的联盟结构生成问题进行研究。将联盟结构生成过程建立为引入成本的马尔可夫决策过程,利用改进DQN的轻量化算法进行求解。实验结果表明,相较于传统强化学习,DDQN算法具有较好的收敛性且能够在一定程度上降低设备的资源占用率。

本文还存在一定的不足之处,需要进一步改进与优化。例如:根据所设置的场景,本文仅对数量较少的服务器进行了联盟结构生成问题的策略优化研究,后续可考虑在CT-MDP中改变状态空间的表示方法来应对服务器数量增加而导致的高维状态空间;目前在服务器的能量成本计算方式上是通过占用率与能量系数相乘得到,后续需要考虑引入边缘设备的实际参数,使智能体对奖励的获取更符合实际场景;在成本的计算方式中,本文只对构建边缘联盟所花费的成本进行了考虑,而服务器往往分布在不同地理位置,因此还需要将服务器之间的传输时延和通信成本加以考虑。

参考文献

[1] LIU X L, YU J D, WANG J, et al. Resource allocation with edge computing in IoT networks via machine learning[J]. IEEE Internet of Things Journal, 2020, 7(4): 3415-3426.

[2] SATYANARAYANAN M. The emergence of edge computing[J]. Computer, 2017, 50(1): 30-39.

[3] 刘亮,毛武平,李汶蔚,等. 空天地一体化边缘计算网络中基于博弈论的任务卸载策略[J]. 计算机工程, 2025, 51(2): 238-249.
LIANG L, MAO W P, LI W W, et al. Task offloading strategy based on game theory in air-ground-space integrated edge computing networks[J]. Computer Engineering, 2025, 51(2): 238-249. (in Chinese)

[4] WANG X L, DANG J W, ZHAO S X, et al. Coalition structure generation in edge computing environment with multitasking concurrency[J]. IEEE Internet of Things Journal, 2023, 10(5): 4324-4338.

[5] 赵庶旭,韦萍,王小龙. 多任务并发边缘计算环境中最优联盟结构生成策略[J]. 通信学报, 2023, 44(2): 172-184.
ZHAO S X, WEI P, WANG X L. Optimal coalition structure generation strategy in multi-task concurrent edge computing environment[J]. Journal on Communications, 2023, 44(2): 172-184. (in Chinese)

[6] LI X M, WAN J F, DAI H N, et al. A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing[J]. IEEE Transactions on Industrial Informatics, 2019, 15(7): 4225-4234.

[7] 刘惊雷,童向荣,张伟. 一种快速构建最优联盟结构的方法[J]. 计算机工程与应用, 2006, 42(4): 35-37,44.
LIU J L, TONG X R, ZHANG W. A kind of method for quick constructing optimal coalition structure[J]. Computer Engineering & Application, 2006, 42(4): 35-37, 44. (in Chinese)

[8] RAHWAN T, JENNINGS N R. An improved dynamic programming algorithm for coalition structure generation[C]// Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems. New York, USA: ACM Press, 2008: 1417-1420.

[9] GRECO G, LUPIA F, SCARCELLO F. Coalitional games induced by matching problems: complexity and islands of tractability for the Shapley value[J]. Artificial Intelligence, 2020, 278: 103180.

[10] THRALL R M, LUCAS W F. N-person games in partition function form[J]. Naval Research Logistics Quarterly, 1963, 10(1): 281-298.

[11] HU Y N, LI C S, ZHANG K J. A method of searching for optimal coalition structure for solving resource scheduling problem of overall load balancing in edge computing environments[J]. Journal of Physics: Conference Series, 2020, 1550(3): 032080.

[12] ZHANG K J, HU Y N, TIAN F, et al. A coalition-structure's generation method for solving cooperative computing problems in edge computing environments[J]. Information Sciences, 2020, 536: 372-390.

[13] DING S Y, LIN D H. A coalitional Markov decision process model for dynamic coalition formation among agents[C]// Proceedings of the IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT). Melbourne, Australia: IEEE Press, 2020: 308-315.

[14] DING S Y, LIN D H. Deep coalitional Q-learning for dynamic coalition formation in edge computing[J]. IEICE Transactions on Information and Systems, 2022, E105. D(5): 864-872.

[15] ZENG D Z, GU L, PAN S L, et al. Resource management at the network edge: a deep reinforcement learning approach[J]. IEEE Network, 2019, 33(3): 26-33.

[16] ALFAKIH T, HASSAN M M, GUMAEI A, et al. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA[J]. IEEE Access, 2020, 8: 54074-54084.

[17] LI J L, GU C H, XIANG Y, et al. Edge-cloud computing systems for smart grid: state-of-the-art, architecture, and applications[J]. Journal of Modern Power Systems and Clean Energy, 2022, 10(4): 805-817.

[18] SHANI G, HECKERMAN D, BRAFMAN R I, et al. An MDP-based recommender system[J]. Journal of Machine Learning Research, 2005, 6(9): 1265-1295.

[19] CHEN X, JIAO L, LI W Z, et al. Efficient multi-user computation offloading for mobile-edge cloud computing[J]. ACM Transactions on Networking, 2016, 24(5): 2795-2808.

[20] CAO X F, TANG G M, GUO D K, et al. Edge federation: towards an integrated service provisioning model[J]. ACM Transactions on Networking, 2020, 28(3): 1116-1129.

[21] WANG C, LEI S B, JU P, et al. MDP-based distribution network reconfiguration with renewable distributed generation: approximate dynamic programming approach[J]. IEEE Transactions on Smart Grid, 2020, 11(4): 3620-3631.

[22] ARULKUMARAN K, DEISENROTH M P, BRUNDAGE M, et al. Deep reinforcement learning: a brief survey[J]. IEEE Signal Processing Magazine, 2017, 34(6): 26-38.

[23] KRIZHEVSKY A, HINTON G. Convolutional deep belief networks on CIFAR-10[J]. Unpublished Manuscript, 2010, 40(7): 1-9.

[24] EVEN-DAR E, MANSOUR Y, BARTLETT P. Learning rates for Q-learning[J]. Journal of Machine Learning Research, 2003, 5(12): 1-25.

文字编辑 金胡考
栏目编辑 宋 圆

计算机工程 www.ecice06.com