中图分类号：TP371　　论文编号：1028716 22-S047
学科分类号：080605

硕士学位论文

复杂场景下基于 Rollout 思想的高效探索和学习方法研究

研究生姓名　　吴卿源
学科、专业　　计算机科学与技术
研 究 方 向　　模式分析与智能计算
指 导 教 师　　谭晓阳教授

南京航空航天大学
研究生院　计算机科学与技术学院/人工智能学院

二〇二二年三月

Nanjing University of Aeronautics and Astronautics
The Graduate School
College of Computer Science and Technology

Research on Efficient Exploration and Learning Method Based on Rollout in Complex Scenarios

A Thesis in

Computer Science and Technology

by

Qingyuan Wu

Advised by

Prof. Xiaoyang Tan

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Engineering

March, 2022

承诺书

本人声明所呈交的硕士学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京航空航天大学或其他教育机构的学位或证书而使用过的材料。

本人授权南京航空航天大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本承诺书）

作者签名：_____

日　　期：_____

摘　要

近些年来，强化学习已经在规划与调度问题、游戏人工智能、机器人控制等多个领域进行了广泛的应用，并取得了巨大的成功。在现有的强化学习算法中，Rollout 算法作为简单且高效的强化学习代表算法，已经成为研究强化学习问题的主流解决算法。本文将先对当前已有的强化学习算法进行综述，并对当前强化学习中尚未解决的问题进行归纳分析。并且针对当前 Rollout 算法中基于模型模拟的蒙特卡洛方法以及基于时序差分的状态价值估计方法所存在的挑战问题，在理论框架，探索策略以及价值函数更新算子进行了如下工作：

(1) 针对 Rollout 算法中基于模型模拟的蒙特卡洛方法在解决具有巨大空间的强化学习任务时所面临的探索效率低下问题提出了一种新的强化学习算法 GLVS。并结合现实场景中具有 NP 难属性的调度问题进行数值模拟实验，通过理论分析和实验结果表明 GLVS 具有比其他流行基准算法更好的性能。

(2) 针对 Rollout 算法中基于时序差分的状态估计方法在解决具有稀疏奖励的强化学习任务时所面临的探索策略低效问题，结合 UCB 算法提出了一种集成探索策略 UCB-Q 和强化学习算法 Alter DQN。通过在不同的强化学习实验环境上的实验结果可表明 Alter DQN 具有比其他流行基准算法更好的探索性能。

(3) 在具有稀疏奖励的强化学习任务中，Rollout 算法中基于时序差分的状态估计方法还面临着价值函数学习效率低下问题。对于该问题提出了一个高效探索和学习的集成强化学习框架 TEAM-Q。通过理论分析和实验结果表明所提出的 TEAM-Q 框架能够有效提升智能体的探索效率和学习效率，取得比流行基准算法更好的性能表现。

关键词：强化学习，Rollout 算法，深度强化学习，蒙特卡洛树搜索，Q 学习算法

ABSTRACT

Recently, reinforcement learning (RL) has been widely applied and achieved great success in many domains such as planning and scheduling problems, game artificial intelligence, and robotics. Among the existing RL algorithms, the Rollout algorithm, as a simple and efficient representative RL algorithm, has become the mainstream solution algorithm for the study of RL problems. This paper will firstly summarize the existing popular RL algorithms, and summarize and analyze the problems that have not been solved in the current RL. In view of the challenges existing in the current Rollout algorithm based on the Monte Carlo method based on model simulation and the state value estimation method based on temporal difference, the contribution of this thesis in the theoretical framework, the exploration strategy and the value function update operator are as follows:

(1) For addressing the low exploration efficiency when the Monte Carlo based on model simulation in Rollout algorithm applied in RL task with huge state space, we propose a new RL algorithm, GLVS. Combined with the NP-hard scheduling problem in real scenarios, numerical simulation experiments are conducted. The theoretical analysis and experimental results show that GLVS has better performance than other popular benchmark algorithms.

(2) For addressing the low exploration efficiency when the value estiation method based on temporal difference in Rollout algorithm applied in RL task with sparse rewards, we propose an ensemble exploration strategy UCB-Q and the RL algorithm Alter DQN. The experimental results on different RL experimental environments show that Alter DQN has better exploration performance than other popular benchmark algorithms.

(3) In the RL task with sparse rewards, the state estimation method based on temporal difference in the Rollout algorithm also faces the problem of low value learning efficiency. For this problem, we propose an ensemble reinforcement learning framework TEAM-Q for efficient exploration and learning. The theoretical analysis and experimental results show that the proposed TEAM-Q framework can effectively improve the exploration efficiency and learning efficiency of the agent, and achieve better performance than popular benchmark algorithms.

Keywords: reinforcement learning, rollout algorithm, deep reinforcement learning, monte carlo tree search, Q-learning

目　录

图表清单

注释表

A^{π}	优势函数	R_t	t 时刻后所获得的累积奖励
$\mathcal{A}$	单个智能体的动作空间	r	环境的奖励函数
a	智能体的动作	$\mathcal{S}$	环境的状态空间
B	经验回放池的容量	T	轨迹长度
UCB	UCB 值	V^{π}	状态价值函数
d^{π}	π 所对应的环境状态分布	V^{*}	最优状态值函数
$\mathbb{E}$	数学期望	y	时间差分目标
f, f_s	用于值函数变换的连续函数	α	学习率
$G_t^{(n)}, G_t^{\lambda}$	n 步和 λ 形式的 TD 目标	γ	奖励折扣因子
$\mathcal{H}$	算法的假设空间	ϵ	贪心策略选择的贪心常数
$J, J(\theta)$	强化学习的目标函数	$\theta, \boldsymbol{\theta}$	神经网络的参数
L	优化问题的损失函数	$\theta^{-}, \boldsymbol{\theta}^{-}$	目标神经网络的参数
K	集成大小	λ	融合多步值函数估计的常数
N	动作器个数	μ	行为策略
o	局部观测	π	策略函数
$\mathcal{P}$	环境的状态转移函数	π^{*}	最优策略函数
P	状态转移函数	ρ_0	环境的初始状态分布
Q^{π}	状态动作价值函数	η, η^{k}	无穷小量
Q^{*}	最优动作值函数	τ	历史轨迹
$\mathcal{B}$	贝尔曼算子	$\mathcal{B}^{*}$	贝尔曼最优算子
Q_k	第 k 个状态动作价值函数	$\mathbf{T}^{\mathrm{EM}}$	集成更新算子
$\mathbf{T}^{\mathrm{ME}}$	Expected-Max 集成更新算子	$\nabla J(\theta)$	策略 π 的策略梯度

缩略词

缩略词	英文全称
AC	Actor Critic
A2C	Advantage Actor Critic
A3C	Asynchronous Advantage Actor Critic
C51	Categorical51-atom agent
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q Network
D3QN	Dueling Double Deep Q Network
DPG	Deterministic Policy Gradient
DQN	Deep Q Network
GPI	Generalized Policy Iteration
I2A	Imagination-Augmented Agent
KL	Kullback Leibler
LSTM	Long Short Term Memory
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
PER	Prioritized Experience Replay
PG	Policy Gradient
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Units
RL	Reinforcement Learning
SAC	Soft Actor Critic
SimPLe	Simulated Policy Learning
TB	Tree Backup
TD	Temporal Difference
TD3	Twin Delayed Deep Deterministic Policy Gradient
TEAM-Q	Temporally-varying Expected-mAx ensemble-Q
TRPO	Trust Region Policy Optimization

第一章 绪论

1.1 本文研究背景

近些年来，随着计算机存储技术以及运算能力的迅速蓬勃发展，机器学习因此获得巨大的进步，并且其已经在多个领域进行了广泛且非常成功的应用。机器学习通常是指：通过一个算法对经验数据进行学习得到一个模型，并使用该模型去完成一些特定的任务。当前常见的机器学习研究领域有：视觉图像处理，自然语言处理，机器人自动控制。

机器学习算法可以主要分为：(1) 监督学习[1]。给定训练数据样本的特征以及对应的标签，算法目的是通过拟合这些样本得到一个模型，并且该模型能有效地泛化到测试数据样本上，如预测，分类。代表性的监督学习算法包括：K 近邻[2]，决策树[3]，朴素贝叶斯[4] 和逻辑回归[5]。(2) 无监督学习[6]。该算法的应用场景通常为没有标签的训练数据样本，算法需要通过主动挖掘这些数据内在隐含的结构特征来学习模型来完成聚类，生成等任务。代表性的无监督学习算法包括：K-Means[7]，典型相关性分析[8]。(3) 强化学习[9]。算法需要控制智能体与环境不断地进行交互，并对所产生的奖励信号进行学习，目的是最大化期望累计回报。常见的任务场景如棋盘游戏，电子游戏，机器人控制以及规划调度问题。代表性的强化学习算法包括：Q 学习[10]，SARSA[9]，Reinforce[9]。

强化学习是机器学习中一个是重要的分支，其通过智能体与环境交互来进行目标导向性的学习。在整个过程中，智能体不会被提前告知该如何决策，而是从其自身与环境之间交互所获得的经验数据进行学习。因此，强化学习本质上也可被视作是一个试错学习过程。这也意味着强化学习任务中存在的一个独有挑战，即探索与利用的权衡 (exploration-exploitation trade-off)[9]。智能体的最终目标是获得最大化的期望累计回报，因此其需要在那些已经尝试过并能够带来不错奖励的动作与那些没有选择过且有可能有更高奖励的动作之间进行抉择。因此智能体需要对不同的动作进行大量地尝试，并逐渐学习到那些最优动作。与另外两种机器学习范式不同。在强化学习场景中，并不会有一个具有完备知识的监督者给数据进行标记，也没有提前给定的特征标签数据对作为样本训练集。智能体只能通过与环境进行交互，并使用产生的数据进行学习。并且监督学习的主要目标是学到一个在测试数据集上具有良好泛化性的模型。而在无监督学习中，任务通常为使用模型对没有标签的数据进行内部隐式结构的挖掘。因此，强化学习与这两者学习范式有着本质的不同。并且强化学习有如下明显特征：(1) 其只能通过交互得到的奖励信号进行学习；(2) 奖励信号的反馈是具有延时性；(3) 其解决的是序列决策问题，所产生的数据不是序列产生的；(4) 智能体的交互方式会影响其自身接收到的奖励信号。随着深度学习的提出以及人工神经网络的蓬勃发展[11]，传统机器学习算法通过结合深度学习技术，其性能得到了巨

大进步，应用场景也变得前所未有的广阔。由于深度神经网络拥有强大的表达能力以及函数拟合近似能力，其已经在众多机器学习任务场景上进行了广泛的应用，如变分自编码器[12]，对抗生成网络[13]。随着深度学习与强化学习结合，深度强化学习的出现和广泛应用，使得其能够有效解决多个领域中的问题，甚至是突破性的进步。在如此多的成就中，深度学习技术最为著名的成果之一是结合了强化学习的 Alpha Go[14]，其也是深度强化学习中的基石算法之一。

虽然强化学习现在已经取得了十分瞩目的成就，但其仍然面临着诸多挑战。现有的强化学习算法大多都只能在虚拟的模拟仿真环境中应用，因为在现实场景中的试错成本过于昂贵。并且，现有的强化学习算法还面临探索策略及样本效率低下问题，即强化学习算法通常需要在模拟环境中交互进行千万甚至上亿帧才能够有效地学习。强化学习算法的性能还高度依赖于奖励函数，即奖励函数的设计决定了智能体能够学习的功能，但在现实场景中是很难去定义一个优质的价值函数。强化学习算法的泛化性有限，在一个的任务环境下训练的智能体往往不能迁移到另一个任务环境中去。由于任务环境中存在高度的不确定性，强化学习算法还面临这性能不稳定和复现性差等问题。尽管如此，强化学习作为机器学习中的重要分支之一，已经在多个领域上取得了卓越的成果，其也是实现通用人工智能的重要研究方向。

1.2 国内外研究现状

当前，强化学习已经广泛地应用到了规划和调度问题[15–17]，棋盘游戏[14] 以及游戏 AI[18]。现有的强化学习算法可以被分为三种主要类型：(1) 基于价值函数强化学习算法：智能体会学习状态或状态动作对价值函数，并通过其在每个状态下选择价值最优的动作与环境进行交互。代表性的算法有：Rollout 算法[9]，深度 Q 网络[18]，C51[19]，Rainbow[20]；(2) 基于策略强化学习算法：智能体会学习一个策略函数，其是状态直接到动作的映射，并直接使用该策略与环境交互。代表性的算法有：PPO[21]，TRPO[22]，A3C[23];(3) 基于模型强化学习算法：智能体会学习或被给定一个模型，并根据这个模型进行规划。代表性的算法有：MCTS[24]，Alpha Go[14]，Alpha Zero[25]，Dyna[26]，SimPLe[27]，I2A[28]，World Models[29]。除此之外，现有的主流工作还使用将价值函数和策略函数相结合的“行动器-评判器”强化学习算法。其有将价值函数和策略函数分别作为评判器和行动器，并使用价值函数来评价策略函数选择的动作。代表性的算法有：DDPG[30]，SAC[31]，TD3[32]。无论哪一类强化学习算法，其都是解决序列决策问题，且目标都是最大化期望未来回报。图1.1中总结了当前各种类型中的流行算法。深度学习技术的发展带来了具有强大表达能力的人工神经网络，这使得传统强化学习进入了新的纪元，深度强化学习。通过结合卷积神经网络与 Q 学习算法[10]，Deepmind 公司在 2013 年提出了深度 Q 学习算法[18]，将其应用到玩 Atari[33] 游戏上，取得了突破性的进展。这也是在广泛意义上，深度强化学习的第一次出现。Deepmind 公司在 2015 年提出了著名的围棋程序 Alpha Go[14]，其融合了深度神经网络和蒙特卡洛树搜索[24]，并达到超越了人类顶尖围棋选手的水平。

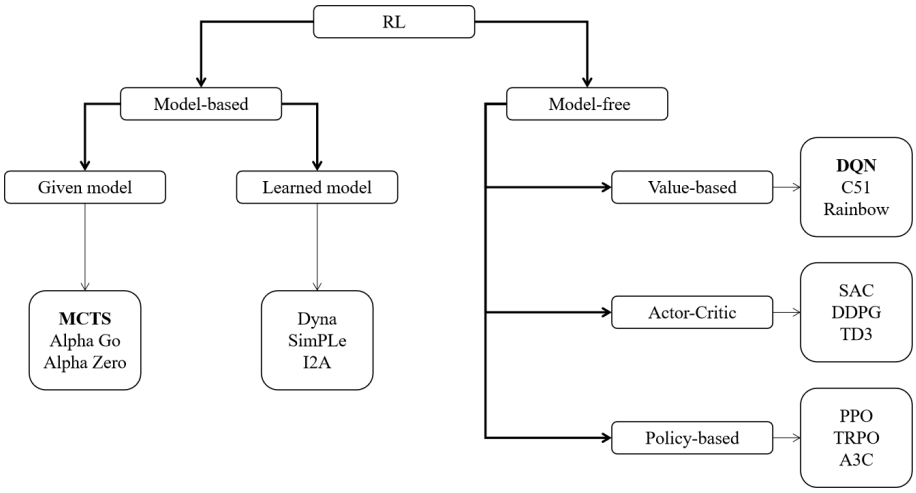


图 1.1　强化学习中不同类型的流行算法

1.3　本文主要贡献

本文的研究内容为复杂场景下的基于 Rollout 思想的强化学习算法研究。强化学习任务中的复杂场景通常指包括具有巨大状态空间或者奖励具有延迟和稀疏性的环境场景。在这样的复杂场景下，Rollout 算法通常面临着三个挑战，其分别为 (1)Rollout 算法中基于模型模拟的蒙特卡洛方法在具有巨大状态空间中探索效率不足问题，(2)Rollout 算法中基于时序差分的状态价值估计方法在具有稀疏奖励的强化学习问题中探索效率低下以及 (3) 价值函数学习效率低下问题。本文所要解决的研究问题为上述 Rollout 算法在复杂强化学习场景中面临的三个挑战。本文的主要贡献如下：

- 对当前流行的强化学习算法进行了综述。强化学习通常使用马尔科夫决策过程作为理论框架，其也是序列决策问题的经典形式化表达。现有的强化学习算法通常可以被分为基于价值函数和基于策略函数强化学习算法，以及可被分为无模型和基于模型强化学习算法。并总结当前强化学习中 Rollout 算法仍面临的挑战。
- 对于具有巨大空间的强化学习问题，Rollout 算法中基于模型模拟的蒙特卡洛方法所面临的探索效率低下和奖励延迟问题提出了一种新的强化学习算法，GLVS。并在现实中常见且具有巨大状态空间的调度问题上进行实验。通过理论分析和实验结果表明 GLVS 具有比其他流行基准算法更好的探索性能。
- Rollout 算法中基于时序差分的状态估计方法在解决具有稀疏奖励的强化学习问题时，其探索策略通常会变得非常低效。对此问题结合 UCB 算法提出了一种集成的探索策略 UCB-Q。通过实验结果表明，所提出的集成探索策略能够有效地使用不同价值函数进行交替探索而提升智能体的探索有效性。
- Rollout 算法中基于时序差分的状态估计方法在面对具有稀疏奖励的强化学习问题时，智

能体的探索策略有效性和价值函数的学习效率会由于奖励的稀疏而变得十分低效。对于该问题提出了一个高效探索和学习的集成强化学习框架，TEAM-Q。该框架中包含了时序变化集成搜索策略和 Expected-Max 集成算子。通过理论分析表明，所提出的探索方式和算子分别拥有很好的探索性能和价值函数收敛率。在多个不同规模的实验环境中的实验结果表明，对比当前的流行的基准算法，所提出的 TEAM-Q 框架能够取得显著的性能提升。

1.4 本文总体结构

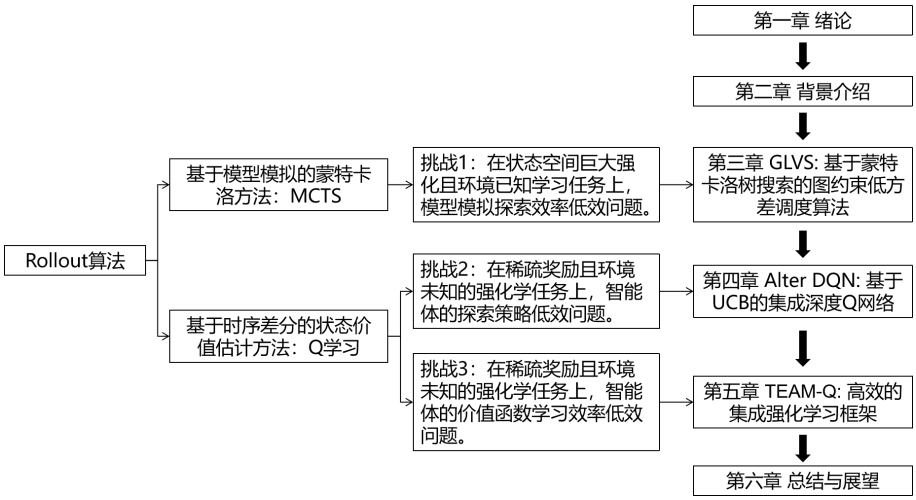


图 1.2　本文的组织结构图

本文的组织结构图如图1.2所示。具体的，本文结构安排如下：第一章为绪论部分，对强化学习的研究背景及意义，国内外研究现状，并对本文的主要贡献和总体结构进行了总结介绍；第二章对强化学习中的马尔科夫决策过程与及价值函数进行背景介绍，并介绍了 Rollout 算法中基于模型模拟的蒙特卡洛方法与基于时序差分的状态估计方法，并对 Rollout 算法所面临的挑战以及本文所要及解决的研究问题进行总结；第三章对具有巨大状态搜索空间的强化学习问题进行研究，如现实生活中的调度问题，针对当前基于模型模拟的蒙特卡洛方法中探索性能低下的问题提出了一种新的 Rollout 算法，GLVS；第四章对基于时序差分的状态价值方法在解决具有稀疏奖励的强化学习问题时出现的智能体探索性能不足问题，提出了一种新的集成强化学习探索策略，UCB-Q 探索策略；第五章为基于时序差分的状态价值估计算法在稀疏奖励环境中探索和学习效率低下问题提出了一个新的集成强化学习算法框架，其包含能够有效进行多样化探索的时序变化集成探索策略和 Expected-Max 集成更新算子；第六章为对本文工作的总结与未来工作的展望。

第二章　强化学习背景介绍

在强化学习任务中[9]，智能体的目标是对一个给定的 MDP 求解出最优策略函数 π^*。当前流行的强化学习问题求解方法主要为:(1) 基于价值迭代 (Value Iteration, VI) 的方法：不会直接地构造策略函数，而是通过求解和优化价值函数 $V^*(s)$ 或 $Q^*(s,a)$，并由此计算出策略函数 (如直接基于价值函数构造的贪婪策略)；(2) 基于策略迭代 (Policy Iteration, PI) 的方法：会直接显示构造一个参数化的策略函数，并在训练过程中使用经验数据优化其参数；(3) 基于“行动器-评判器”("Actor-Critic", AC) 的方法: 其会同时构造和优化价值函数和策略函数，其中价值函数作为“评判器”来评价由策略函数作为“行动器”所选择动作的好坏。除此之外，强化学习算法还可以分为：(1) 无模型的强化学习算法 (model-free)：不会学习或给定环境模型，而从经验数据中学习价值函数和 (或) 策略函数；(2) 基于模型的强化学习算法 (model-based): 给定或从经验数据中学习一个模型，并使用模型去规划价值函数和 (或) 策略函数。

2.1　马尔可夫决策过程

强化学习[9] 所要解决的机器学习问题为序列决策问题，目标为训练一个智能体，其通过不断地与环境进行交互，对所得到的经验数据进行学习以调整自己的交互方式。最终目标为取得最大的累计奖励。因此，强化学习也可以被视作在交互中学习。对于这样的交互式学习问题，智能体和环境的交互过程通常会使用序列决策的经典理论框架——马尔可夫决策过程 (Markov decision process, MDP)[9] 来进行形式化表达。通常，马尔可夫决策过程的定义如下所示：

定义 2.1（马尔可夫决策过程）：马尔可夫决策过程 (Markov decision process, MDP) 可以由一个五元组 $< \mathcal{S}, \mathcal{A}, r, \mathcal{P}, \gamma >$ 所定义，其中：

- $\mathcal{S}$: 状态空间，其可以是有限的或是无限的；
- $\mathcal{A}$: 动作空间，其可以离散或者连续的；
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: 奖励函数，当智能体在状态 s 下采取动作 a 后所获得的即时奖励 $r(s,a)$；
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$: 状态转移函数，$\Delta(\mathcal{S})$ 为一个在状态空间上的概率分布，$\mathcal{P}(s'|s,a)$ 为智能体在状态 s 下采取动作 a 后能够转移到状态 s' 的概率;
- $\gamma \in [0,1)$: 折扣因子，奖励随时间变化而进行折扣降低。

智能体在每个离散时刻 $t = 0, 1, \ldots$ 都与环境进行交互。具体来说，智能体在时刻 t 时观测环境得到状态表示 $s_t \in \mathcal{S}$，并根据状态从状态空间中选择一个动作 $a_t \in \mathcal{A}$。智能体会得到奖励信号 r_t，环境的状态也转移到了 $s_{t+1} \sim \mathcal{P}(\cdot \mid s_t, a_t)$。并且在时刻 t 时，智能体的历史交互过程可以表示为轨迹 $\tau_t = (s_0, a_0, r_0, s_1, \ldots, s_t)$。图2.1展示了智能体与环境之间的交互过程。在一

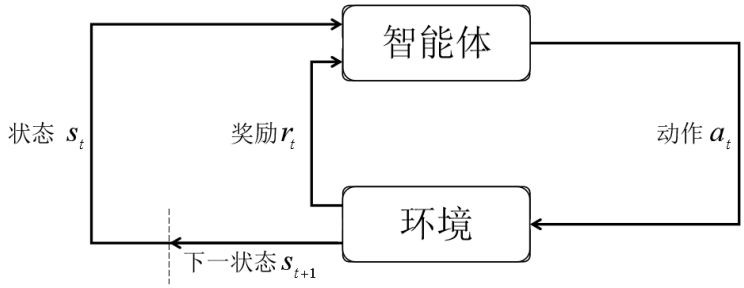


图 2.1　强化学习中智能体与环境的交互过程

些强化学习任务中，当智能体到达终止状态 s_T 是，其完成了一幕的交互过程，T 为终止时刻。而在另一些强化学习任务中，智能体会一致持续性地与环境交互，没有终止状态，即终止时刻 $T=\infty$。

在强化学习中，智能体的目标是最大化未来累计奖励。其不单要考虑现有的即时奖励，也要考虑未来的长期奖励。因此，在时刻 t，智能体后续采取的动作所得到的轨迹的折扣回报定义为:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = \sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1}, a_{t+k+1}) \tag{2.1}$$

对于智能体来说，其在当前状态下所做出的动作决策都根据一个策略函数 $\pi(a|s) : \mathcal{S} \times \mathcal{A}$，其为状态到动作的映射[9]。策略函数根据智能体所观察的状态 s，直接确定性地指导智能体的动作行为 $a = \pi(s)$ 或者给出动作的选择概率 $a \sim \pi(\cdot|s)$。

2.2　强化学习中的价值函数

在强化学习中，对智能体的策略函数 π 来说，其重要的评价标准为价值函数[9]。在状态 s 下，策略函数 π 的状态价值函数 $V^\pi(s)$ 通常被定义为:

$$V^\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid s_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right], \forall s \in \mathcal{S} \tag{2.2}$$

其表示的含义为对在状态 s 下，智能体执行策略 π 后续所能获得的期望回报的估计。相似地，在状态 s 下，采取动作 a 后，策略函数的状态动作价值函数 $Q^\pi(s,a)$ 通常被定义为:

$$Q^\pi(s,a) \doteq \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right], \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{2.3}$$

其表示的含义为对在状态 s 下，并执行动作 a 后，智能体执行策略 π 后续所能获得的期望回报的估计[9]。状态价值函数和状态动作价值函数之间是紧密相连的，两者区别为价值函数 $V^\pi(s)$ 估计的是在状态 s 下根据策略函数 $\pi(\cdot|s)$ 采取动作的期望折扣回报，而在状态动作价值函数

$Q^{\pi}(s,a)$ 中，在状态 s 下的动作 a 为提前指定的，和策略函数 π 无关。两者的关系可描述为[9]：

$$\begin{aligned} V^{\pi}(s) &= \int \pi(a|s)Q^{\pi}(s,a)\mathrm{d}a \\ Q^{\pi}(s,a) &= r(s,a) + \gamma \int \mathcal{P}(s'|s,a)V^{\pi}(s')\mathrm{d}s' \end{aligned} \tag{2.4}$$

状态价值函数可以用来比较两个策略函数 π^0 和 π^1。如果对状态空间中任意的状态 $s \in \mathcal{S}$，都满足 $V^{\pi^0}(s) \geq V^{\pi^1}(s)$，这时我们可以称策略函数 π^0 优于策略函数 π^1 时，即 $\pi^0 \leq \pi^1$。类似地，对于状态动作价值函数，也可以推导出：如果对状态空间中任意的状态 $s \in \mathcal{S}$ 和动作 $a \in \mathcal{A}$，都满足 $Q^{\pi^0}(s,a) \geq Q^{\pi^1}(s,a)$，这时 $\pi^0 \leq \pi^1$。当我们可以找到一个策略函数 π^*，对于其他任意的策略函数 π，其满足：$V^{\pi^*}(s) \geq V^{\pi}(s)$ 或 $Q^{\pi^*}(s,a) \geq Q^{\pi}(s,a)$，则 π^* 为最优策略函数。如果我们定义 π 为策略函数空间，则最优策略函数 π^* 可以基于状态价值函数被定义为 $\pi^* \leftarrow \arg\max_{\pi} V^{\pi}(s), \forall s \in \mathcal{S}$。或者基于状态动作价值函数被定义为 $\pi^* \leftarrow \arg\max_{\pi} Q^{\pi}(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}$。相反地，最优状态价值函数为 $V^{\pi^*}(s) = V^*(s)$，最优状态动作价值函数为 $Q^{\pi^*}(s,a) = Q^*(s,a)$。并且，通过最优价值函数，我们可以很容易地通过构造对应的贪心策略以获得最优策略函数[9]。对于一个策略函数 π，根据公式2.4，我们可以推导出如下价值函数关于策略函数 π 的贝尔曼等式 (Bellman Equation)[9]：

$$\begin{aligned} V^{\pi}(s) &= \int \pi(a|s)[r(s,a) + \gamma \int \mathcal{P}(s'|s,a)V^{\pi}(s')\mathrm{d}s']\mathrm{d}a \\ Q^{\pi}(s,a) &= r(s,a) + \gamma \int \mathcal{P}(s'|s,a)[\int \pi(a'|s')Q^{\pi}(s',a')\mathrm{d}a']\mathrm{d}s' \end{aligned} \tag{2.5}$$

最优状态价值函数为 $V^*(s)$ 和最优状态动作价值函数为 $Q^*(s,a)$ 则满足贝尔曼最优等式 (Bellman Optimality Equation)[9]：

$$\begin{aligned} V^*(s) &= \max_{a} \int \pi(a|s)[r(s,a) + \gamma \int \mathcal{P}(s'|s,a)V^*(s')\mathrm{d}s']\mathrm{d}a \\ Q^*(s,a) &= r(s,a) + \gamma \int \mathcal{P}(s'|s,a)[\int \pi(a'|s') \max_{a'} Q^*(s',a')\mathrm{d}a']\mathrm{d}s' \end{aligned} \tag{2.6}$$

基于状态价值函数 $V^{\pi}(s)$ 和状态动作价值函数 $Q^{\pi}(s,a)$，我们可以定义策略函数 π 的优势函数 (Advantage Function)$A^{\pi}(s,a)$

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.7}$$

其表示在状态 s 下，动作 a 相对于平均而言的相对优势。优势函数将状态动作价值函数进行归一化，以提升学习效率。

2.3 Rollout 算法

Rollout 算法是一种基于蒙特卡洛方法 (Monte Carlo, MC) 的决策时规划算法，其思想为通过仿真模拟部分轨迹来更新价值函数。从一个特定的状态开始，Rollout 算法会根据一个给定策略 π 来进行仿真模拟轨迹，并利用这些轨迹来估计策略的状态动作价值函数。不同于传统的蒙特卡洛方法需要将当前状态下的所有后续状态动作价值函数都求出来，再选择最优的动作，Rollout 算法可以通过不同的动作的仿真轨迹来提升策略。在处理状态空间巨大的强化学习问题时 Rollout 算法是非常有效的，因为其不需要智能体将所有的后续状态都进行访问，而是通过部分轨迹来估计状态价值函数。Rollout 算法的算法流程如图所示。Rollout 算法的核心思想是如何去估计状态动作价值函数，从而让智能体做出正确的决策。因此 Rollout 算法主要分为: (1) 基于模型模拟的蒙特卡洛方法，如蒙特卡洛树搜索 (Monte Carlo Tree Search, MCTS); (2) 基于时序差分方法的价值函数估计方法，如 Q 学习 (Q-learning)。

2.3.1 蒙特卡洛树搜索

当智能体给定或学习到了一个模型之后，蒙特卡洛树搜索[24] 是最流行的强化学习算法。特别是对那些任务环境容易构造，具有完美信息的环境，如围棋，象棋等。如图2.2所示，蒙特卡

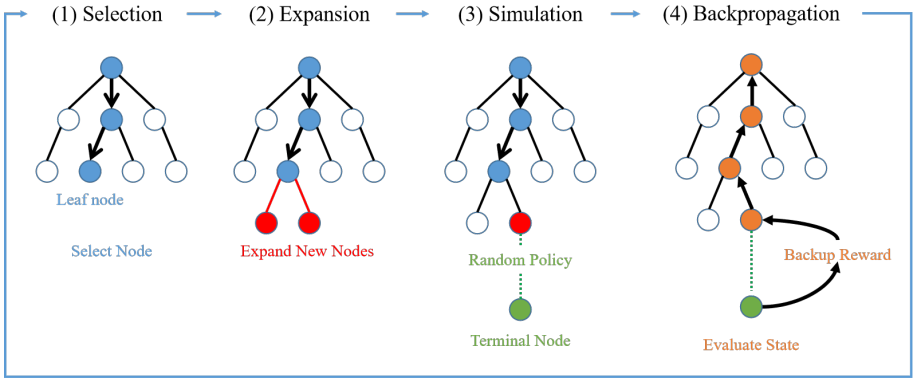


图 2.2 蒙特卡洛树搜索迭代中的四个主要步骤

洛树搜索主要分为如下四个主要步骤:

(1) 选择 (Selection): 如图2.2(1) 所示，从根节点 (root node) 开始，通过一直迭代地使用子节点选择策略选择当前节点的子节点直到非终止的叶子节点 (leaf node)，其拥有未展开的子节点。通常子节点选择策略将上置信界算法 (Upper Confidence Bounds Algorithm) 结合了树结构的 UCT 算法，其会选择具有最大 UCT 值的子节点。对于第 i 个子节点，其 UCT 值的定义如下:

$$UCT_i = V_i + C \times \sqrt{\frac{2ln(N)}{N_i}} \tag{2.8}$$

其中 V_i, N_i 分别为第 i 个子节点的平均价值和访问次数，N 为父节点的访问次数，$C > 0$ 为控制利用和探索的常数。

(2) 扩展 (Expansion): 如图2.2(2) 所示，当一个叶子节点被选择后，其一个或者多个子节点会加入树结构之中。

(3) 模拟 (Simulation): 如图2.2(3) 所示，从一个新扩展的节点开始，通过迭代地执行模拟策略选择子节点直到终止节点。通常使用的模拟策略为完全随机策略。

(4) 反传 (Backpropagation): 如图2.2(4) 所示，对终止节点进行评估得到价值 R，并将价值沿着选择路径进行方向传播到根节点。之前选择过节点 i 的价值 V_i 和访问次数 N_i 会进行更新:

$$V_i \longleftarrow= \frac{N_i * V_i + R}{N_i + 1}, N_i \leftarrow N_i + 1 \tag{2.9}$$

2.3.2 时序差分学习与 Q 学习算法

在强化学习中，结合了蒙特卡洛 (Monte Carlo，MC) 方法和动态规划 (Dynamic Planning，DP) 方法的思想的时序差分 (Temporal Difference，TD) 学习[9] 是最为主要的思想之一。其可以直接从交互的经验数据中学习。并且基于自举 (Bootstrapping) 的思想，其可以使用已有的状态来更新当前状态的价值函数，而不用等到交互到终止状态。时序差分学习算法中的代表算法为 Q 学习算法 (Q-Learning)[10]。Q 学习的目标为学习最优状态动作价值函数 $Q^*(s,a)$，对于任意给定的价值函数 $Q(s,a)$，该算法的更新形式如下：

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{2.10}$$

其中 s 为智能体所在的当前状态，a 为智能体采取的动作，r 为智能体得到的奖励，s' 为智能体到达的下一状态，$\gamma \in [0,1)$ 为奖励折扣因子，$\alpha \in (0,1)$ 为学习率。其中 $r + \gamma \max_{a'} Q(s',a')$ 也称作时序差分目标值 (Temporal Difference Target，TD Target)。Q 学习算法的核心为贝尔曼最优算子 (Bellman Optimality Operator)，其定义如下:

定义 2.2（贝尔曼最优算子）：贝尔曼最优算子 (Bellman Optimality Operator) $\mathcal{B}^*$ 定义为如下映射:

$$(\mathcal{B}^*Q)(s,a) \doteq r(s,a) + \gamma \int \mathcal{P}(s'|s,a) \max_{a'} Q(s',a') \mathrm{d}s', \forall (s,a) \in \mathcal{S} \times \mathcal{A} \tag{2.11}$$

并且贝尔曼最优算子具有如下性质：

引理 2.3（贝尔曼最优算子性质）：对于任意两个状态动作价值函数 $Q_1(s,a), Q_2(s,a)$，贝尔曼最优算子 $\mathcal{B}^*$ 有如下两个性质:

- 单调性 (Monotonicity): 如果 $Q_1 \leq Q_2$，则 $\mathcal{B}^*Q_1 \leq \mathcal{B}^*Q_2$；

- 收缩性 (Contraction): 贝尔曼最优算子 $\mathcal{B}^*$ 是 γ-收缩映射，即

$$||\mathcal{B}^*Q_1 - \mathcal{B}^*Q_2||_\infty \leq \gamma||Q_1 - Q_2||_\infty$$

可知贝尔曼最优算子和无穷范数可以形成一个完备的度量空间，并根据巴拿赫不动点定理 (Banach Fixed Point Theorem) 可知贝尔曼最优算子有唯一的不动点。可以得到如下定理证明贝尔曼最优算子 $\mathcal{B}^*$ 可以使任一状态动作价值函数收敛到最优状态动作价值函数 Q^*。

定理 2.1（贝尔曼最优算子不动点及收敛定理）：贝尔曼最优算子 $\mathcal{B}^*$ 拥有不动点 Q^*，

$$\mathcal{B}^*Q^* = Q^*$$

并且 $Q^*(s,a) = \sup_\pi Q^\pi(s,a), \forall(s,a) \in \mathcal{S} \times \mathcal{A}$ 为最优状态动作价值函数。给定任意初始的状态动作价值函数 Q_0，通过不断迭代计算

$$Q_{k+1} \leftarrow \mathcal{B}^*Q_k$$

其中 $k = 0, 1, \ldots$。可得 $Q_k \to Q^*$

综上所述，如果在 Q 学习更新时为其提供足够多的 “状态-动作对” 样本，其就能够将状态动作价值函数收敛至 Q^*。强化学习中，智能体需要在探索与利用之间进行有效地平衡。为此，Q 学习算法通常在智能体的探索阶段使用 ϵ-贪心策略 (ϵ-greedy policy)[9] 而不是完全贪心策略 (greedy policy) 与环境进行交互，以获取足够多样的数据和避免陷入局部最优。

定义 2.4（ϵ-贪心策略）：当智能体使用 ϵ-贪心策略 $\pi_\epsilon(Q)$ 与环境进行交互时，在状态 s_t 下，其动作 a_t 选择为

$$a_t = \begin{cases} \arg\max_a Q(s_t, a), \text{以概率} 1-\epsilon \\ uniform(\mathcal{A}), \text{以概率} \epsilon \end{cases} \tag{2.12}$$

其中 $\epsilon \in [0, 1]$ 为探索因子。

智能体会以 $1-\epsilon$ 的概率根据状态动作价值函数 $Q(s,a)$ 去选择在当前状态下估计价值最大的动作，并会以 ϵ 的概率去动作空间中均匀选取动作进行探索。因为在强化学习任务中会存在噪声等因素，这要求智能体要进行多次探索才能找到最优动作。因此整个 Q 学习可以被视作两个部分:(1) 价值函数更新: 使用经验数据对价值函数进行有效更新；(2) 环境交互策略: 在环境中交互时使用合适的策略进行探索和利用。这也是反应了强化学习任务的特点——智能体在交互中学习。

2.3.2.1 基于 Q 学习的改进算法

Q 学习是现有强化学习中最为简单且有效的算法之一，但其也面临着许多的问题。因此，现在已经有非常多针对 Q 学习改进的方法被提出且应用到各种不同的场景中。传统的 Q 学习通常会构造一个 Q 表格来作为状态动作价值函数 $Q(s,a)$，其中储存这每一个“状态——动作对”的估计价值。但当 MDP 的状态空间变得非常大时，这个方法就变得非常低效且不可行。因此，Q 学习当前最重要的一个改进就是深度 Q 网络 (Deep Q Network，DQN)[18,34]。其使用具有强大泛化能力的深度神经网络来拟合 Q 表格，这使得其在一些具有无限维状态空间的强化学习任务上取得成功，如 Atari 游戏[33] 等。其构造一个参数化的深度网络 $Q(s,a;\theta)$，其中 θ 为网络参数。为了稳定神经网络的训练过程，有两点改进是非常重要的。第一，神经网络在每次更新时都会从经验回访池 (Experience Replay Buffer) 中进行随机采样进行离线地训练。这样不仅能够保留历史的经验数据，提高其利用率，还能够打乱数据之间的关联性进行异策略训练。第二，使用一个固定的目标网络 $Q^-(s,a;\theta^-)$ 来计算目标值更新网络 $Q(s,a;\theta)$，其中参数 θ^- 为 $Q(s,a;\theta)$ 的历史参数，即每过一段固定的时间参数会进行复制 $\theta^- \leftarrow \theta$。具体来说，深度 Q 网络的损失函数为:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}[(r + \gamma \max_{a'} Q^-(s',a';\theta^-) - Q(s,a;\theta))^2] \tag{2.13}$$

其中 (s,a,r,s') 为从经验回访池中提取的批次样本。深度 Q 网络的 TD 目标值为:

$$y^{DQN} = r + \gamma \max_{a'} Q^-(s',a';\theta^-) \tag{2.14}$$

在与环境进行交互时，深度 Q 学习同样使用 ϵ-贪婪策略收集数据至经验回访池。深度 Q 网络是强化学习发展过程中最为重要的算法之一，但其还是面临着许多问题如探索有效性，样本有效性以及价值函数估计偏差等[35]。因此，现有许多基于深度 Q 网络算法的改进。在状态动作价值函数进行更新时，需要选择具有最大估计值的动作来计算 TD 目标值。这样的更新机制会给价值函数带来过估计偏差 (Overestimation Bias)。对于这个问题，Double DQN 算法[36,37] 最先提出了一个有效的缓解过估计方案，即使用一个双重估计器 (Double Estimation) 来计算 TD 目标值。具体来说，其会初始化两个独立的状态动作价值函数 Q_A, Q_B。在每次价值函数更新阶段，会随机选取一个状态动作价值函数 (如 Q^A)，并使用如下的方式计算 DoubleDQN 的 TD 目标值:

$$y_A^{DDQN} = r + \gamma Q_A(s', \arg\max_{a'} Q_B(s',a')) \tag{2.15}$$

在实际的 Double DQN 算法实现中，许多工作会直接使用学习网络 $Q(s,a;\theta)$ 和目标网络 $Q^-(s,a;\theta^-)$ 作为两个不同的估计器。Dueling DQN[38] 提出了一种对偶的网络结构: 其将网络的输出分解成了两个独立的部分，即状态价值函数 $V(s)$ 和优势函数 $A(s,a)$。这样的结构能够使得网络在训

练时能够同时训练多个动作，提升训练效率。其通过如下方式计算状态动作价值函数:

$$Q(s,a) = V(s) + (A(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}A(s,a')}) \tag{2.16}$$

其中优势函数做了中心化处理能够有效稳定训练过程。在 N-step 时序差分学习中[9]，多步回报 (Multi-step Return) 是能够有效提升学习效率的方法之一，其会使用轨迹中多步的奖励型号来估计 TD 目标值:

$$y^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max_{a_{t+n+1}} Q(s_{t+n+1}, a_{t+n+1}) \tag{2.17}$$

值分布式强化学习 (Distributional RL)[19] 指的是使用一个随机变量来表示定点的值，具体来说就是将价值函数估计值变为估计分布。Categorical DQN 基于值分布贝尔曼等式 (Distributional Bellman Equation) 进行更新:

$$Z(s,a) = R(s,a) + \gamma Z(s',a') \tag{2.18}$$

其中 $Z(s,a)R(s,a)Z(s',a')$ 为分别为当前“状态动作”对，奖励和下一“状态动作”对的近似分布。Noisy DQN[39] 通过在神经网络模型中加入参数化噪声来增加探索性，且参数会随着梯度下降而调整。

$$\boldsymbol{y} = (\boldsymbol{b} + \boldsymbol{W}x) + (\boldsymbol{b}_{noisy} \cdot \epsilon^b + (\boldsymbol{W}_{noisy} \cdot \epsilon^w)\boldsymbol{x}) \tag{2.19}$$

其中 ϵ^b, ϵ^w 为随机变量，$\boldsymbol{y} = \boldsymbol{b} + \boldsymbol{W}x$ 为普通的线性神经网络。优先经验回放池 (Prioritized Experience Replay, PER)[40] 是一个对于经验回放池的有效改进。通常情况下，智能体会从经验回放池中的队列中均匀采样进行训练。而 PER 通过时序差分误差 (Temporal Difference Error, TD-error)σ 来确定采样样本的优先顺序，通常具有更大的 $|\sigma|$ 会被优先采样到。

$$\sigma = r + \gamma \max_{a'} Q(s',a') - Q(s,a) \tag{2.20}$$

其能够有效提升强化学习算法的训练效率。Rainbow[20] 有效地集成了以上提到的所有技巧并在多个强化学习任务上取得了先进的性能，其也是当前强化学习算法中最为先进的算法之一。其 TD 目标值可以简化为:

$$y^{rainbow} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n Q^-(s_{t+n+1}, \arg\max_{a_{t+n+1}} Q(s_{t+n+1}, a_{t+n+1})) \tag{2.21}$$

基于此，Ape-X DQN[41] 对其进行改进得到了效率更高的的分布式训练版本。

2.3.2.2 集成强化学习算法

Double DQN 算法使用两个独立的价值函数，其可以归属于集成强化学习 (Ensemble Reinforcement Learning, Ensemble RL)。集成强化学习是重要研究方向，现在已经有多项工作证明了其能够有效地提升强化学习算法性能。对于估计偏差 (Estimation Bias) 问题，Ensemble RL 已经被广泛使用，且其是非常有效的一类方法。Ensemble RL 通常会初始化 K 个独立的状态动作价值函数 $\{Q_i(s,a)\}_{i=1}^{K}$。和 Double DQN 同样，Ensemble DQN[42] 在每次价值函数更新阶段，会随机选取一个状态动作价值函数 (如 Q_j)，并使用如下的方式计算 Ensemble DQN 的 TD 目标值:

$$y^{Ensemble} = r + \gamma \max_{a'}[\frac{1}{K}\sum_{i=1}^{K} Q_i(s',a')] \tag{2.22}$$

但 K 个价值函数需更多的计算量去进行更新。因此 Averaged DQN[42] 使用学习网络 $Q(s,a;\theta)$ 的 K 个不同历史版本，其 TD 目标值计算方式和 Ensemble DQN 相同。Maxmin DQN[43] 则使用多个另一种方式来控制估计偏差, 其用多个构造最小状态动作价值函数来计算 TD 目标值:

$$y^{Maxmin} = r + \gamma \max_{a'}[\min_{i} Q_i(s',a')] \tag{2.23}$$

Ensemble Bootstrapping DQN[44] 可以被视作是 Double DQN 的泛化版本，在更新价值函数阶段，会随机选取一个状态动作价值函数 (如 Q^j), 并使用如下的方式计算 Ensemble Bootstrapping DQN 的 TD 目标值:

$$y^{EB} = r + \gamma[\frac{1}{K-1}\sum_{i\neq j}^{K} Q_i(s', \arg\max_{a'} Q_j(s',a'))] \tag{2.24}$$

因为集成强化学习会同时拥有多个不同价值函数，因此这会自然而然地引入了对个函数直接的随机性。因此，集成强化学习还被广泛应用到提升智能体探索效率中。强化学习算法中常用的探索算法为 ϵ-贪婪策略[9]，玻尔兹曼探索策略 (Boltzmann Exploration Strategy)。但这样的探索方式是非常低效甚至无效的。在一些困难的强化学习任务中，奖励信号会是稀疏，具有欺骗性的。简单的探索策略并不能探索到这些珍贵的奖励信号，从而进行有效的学习。Bootstrapped DQN[45] 是基于汤普森采样 (Thompson Sampling) 的思想的强化学习算法，其构造一个多输出的神经网络作为集成状态动作价值函数。在智能体与环境交互每一幕的开始阶段，智能体会基于均匀分布采样一个价值函数 $Q_j(s,a)$

$$a_t = \arg\max_a Q_j(s_t,a) \tag{2.25}$$

其中 $Q_j \sim uniform(\{Q_i\}_{i=1}^{K})$ 当 $t=0$。并且在这一幕中，智能体的所有交互都是基于这个价值函

数的贪婪策略，以确保智能体能够进行深度探索 (Deep Exploration) 和时序持续探索 (Temporally-Extended Exploration)。Bootstrapped Prior DQN[46] 基于贝叶斯线性回归 (Bayesian Linear Regression) 的思想，在 Bootstrapped DQN 的基础上，对神经网络的每一个输出分支加上固定的随机先验参数进一步提升探索性。为了保持不同神经网络的多样性，在更新网络时会给第 j 个神经网络附加独特的二值权重 $m_j \in \{0, 1\}$:

$$L_j = m_j(Q_j(s,a) - r - \gamma \max_{a'} Q_j(s',a')) \tag{2.26}$$

UCB Exploration[47] 结合了置信度上界算法，其通过价值函数的均值和标准差构造一个 UCB 值，并更根据其大小选取动作:

$$a_t = \arg\max_a \mu(s_t,a) + \lambda\sigma(s_t,a) \tag{2.27}$$

其中 $\mu(s_t,a) = \frac{1}{K}\sum_{i=1}^{K} Q_i(s_t,a)$ 为集成价值函数的均值函数，$\sigma(s_t,a) = \sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_i(s_t,a) - \mu(s_t,a))^2}$ 为集成价值函数的均方差函数。Ensemble Voting[47] 根据各个神经网络的选择动作的结果，并通过集成投票的方式选择动作:

$$a_t = MajorityVote(\{\arg\max_a Q_i(s_t,a)\}_{i=1}^{K}) \tag{2.28}$$

SUNRISE DQN[48] 使用了 UCB Exploration 的探索交互方式，使用一种基于不确定度的加权更新方式:

$$L = w(s',a')(Q_j(s,a) - r - \gamma \max_{a'} Q_j(s',a')) \tag{2.29}$$

其中 $w(s',a') = sigmoid(-\sqrt{\frac{1}{K}\sum_{i=1}^{K}(Q_i(s',a') - \frac{1}{K}\sum_{i=1}^{K} Q_i(s',a'))^2})$。这种更新方式能够有效地减小误差传播。

2.3.3 Rollout 算法面临挑战

目前 Rollout 算法已经被广泛地应用在解决强化学习问题中，但其在复杂场景中仍然面临着许多挑战，如具有状态空间巨大或奖励稀疏等强化学习任务。当强化学习问题是具有完美信息的 MDP 时，即其环境的模型是已知 (Model-based RL) 的时候，通常会使用 MCTS 来求解。随着状态空间的变大，MCTS 的搜索树在进行模拟时会面临探索效率低下问题，因为搜索树需要用更多的计算资源进行更多次的搜索才能找到一个具有更高奖励的轨迹来更新状态价值函数从而当前状态下的最优动作。而当强化学习问题没有给定环境模型时 (Model-free RL) 时，通常会使用 Q 学习等算法来求解。这时智能体不能通过使用环境模型来进行模拟，而只能通过试错来学习找出当前状态下的最优动作。因此，智能体需要高效的探索策略在巨大的未知搜索空间中去寻找到潜在的高奖励。当状态空间变大时，智能体通常也会面临奖励稀疏和延迟等问题，因

此其需要高效的价值函数学习方式从交互得到的样本中高效地学习有用的知识。

2.4 本章小结

强化学习是一种最接近人类进行决策的机器学习范式，其已经广泛应用在多个领域中。马尔科夫决策过程是常用的强化学习问题的建模方式，其目标为求解出最优的策略函数。对于求解一个 MDP，强化学习算法可以分为多种类型，如基于价值迭代和基于策略迭代的强化学习算法，还有将两者结合的“行动器-评价器”强化学习算法。Rollout 算法是强化学习中广泛使用的一种基于蒙特卡洛的决策时规划算法。代表算法有基于模型模拟的蒙特卡洛树搜索以及基于时序差分的 Q 学习算法。但这两种 Rollout 算法都存在着各自的挑战。本文后续内容将具体解决 Rollout 算法在具有巨大状态空间的强化学习问题中的以下三个挑战:(1) 在给定环境模型下，MCTS 的探索模拟效率低下;(2) 在无模型强化学习问题下，Q 学习中智能体探索策略效率低下;(3) 智能体价值函数学习效率低下。

第三章　基于蒙特卡洛树搜索和图约束下的低方差调度算法

3.1　背景介绍

在强化学习问题中，当环境的模型已知，智能体通常通过基于模型模拟的 Rollout 算法进行而找出当前状态下的最优动作进行决策。但随着问题规模变大，其模型模拟的效率会随着状态空间的变大而变得十分低下。因为在这种情况下，智能体需要进行更多的模拟才能够发现具有高奖励的轨迹，并且其奖励的延迟性会随着状态的变多而加剧。在现实的生产生活中，调度问题 (Scheduling Problem) 就是这一类状态空间巨大应用广泛且具有重要研究意义的优化问题，如公交车司机调度问题 (Bus Driver Scheduling Problem, BDSP)[49], 车间作业调度问题 (Job Shop Scheduling Problem, JSSP)[50] 和学校课程表调度问题 (School Timetabler Scheduling Problem, STSP)[51]。本文所讨论的调度问题为静态调度问题 (Static Scheduling Problem)，通常这类调度问题中会有事先给定的资源或任务集合，任务目标是将这资源分配或调度给不同的对象，在满足约束的同时最小化目标函数，如成本，时间等。例如，BDSP 中通常包含了许多的班次 (trip)，其任务是将这些班次规划或者分配给不同的司机和公交车辆。多个班次形成的连续序列称之为轮班 (shift)，通常一个轮班就代表其需要一名司机或公交车去执行。这些轮班都需要满足各种约束如限制工作时间等。这些所有的轮班组成一个计划表 (schedule)，若其满足所有约束就称之为可行计划表 (feasible schedule)。对于 BDSP，公交车公司的目标为最小化计划表的成本花销，而司机们需要一个员工友好的计划表。在现实场景中，这样的需求通常是有冲突的，因此该调度问题是具有挑战性和研究意义的。

3.2　相关工作

大多数的调度问题通常是 NP-难问题[52]，这一类问题的解空间会随着问题规模的增加而呈指数级增长，且其没有最优精确算法。对于这类调度问题，常用的方法为基于集合覆盖的方法[53,54]。在实际中，其会先构造一个候选解 (Candidate Solution) 集合池 $S_{pool} = \{s_1, s_2, \ldots, s_M\}$，其中包含许多不同的预先构造的满足约束矩阵 C 的局部解。集合覆盖的方法目标为从 S_{pool} 中选择多个局部解来构造一个最终解。但构造一个候选解集合池是非常复杂和耗时的[55],[56]。并且候选解集合池中的局部解个数通常非常大[57]。因此，有许多启发式方法被提出，如状态空间 relaxation[58]，拉格朗日 relaxation[59]，分枝定界[60]。另外地，有许多方法都将一个问题规模很大的调度问题分解为多个子问题或多个阶段。例如，将 BDSP 分解为早上和下午两个子问题[61]，并分别解决两个子问题再最终结合两个解。还可以将 BDSP 建模为多重分配问题[62]，并且采取一个两阶段确定性启发式算法。基于组合优化的启发式算法，如 GRASP[63] 和 Hungarian 算

法[64]。但现有的启发式算法仍然有许多的缺点[57]: 其通常需要大量的人工先验知识，因此算法的有效性通常非常依赖于设计者的经验，因此会非常不稳定。而蒙特卡洛树搜索 (MCTS)[65] 已经被成功应用在许多的规划问题中[50,66–74]。然而，在解决具有巨大搜索解空间的调度问题上时，现有的基于蒙特卡洛树搜索算法仍然面临着模拟搜索效率低下等问题。

3.3 提出方法

3.3.1 图约束马尔科夫决策过程

为了方便在调度问题上使用强化学习算法，其可以视作一个具有完美信息的单人博弈问题，并可以将该问题转化为一个强化学习问题。具体的，先为调度问题构造一个马尔科夫决策过程 $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$，这样智能体就能够直接在的调度问题解空间中直接进行搜索求解。其中 $\mathcal{S}$ 为调度解空间/状态空间，$\mathcal{A}$ 为动作空间，$\mathcal{P}$ 为状态转移函数，r 为奖励函数。

本文将结合 BDSP 为例子来展示为调度问题构造图约束马尔科夫决策过程的过程。假设给定一共 m 个班次，智能体的初始状态 $S_0 \in \mathcal{S}$ 是为这 m 个班次都各分配一名公交车司机。对于一个状态 $S \in \mathcal{S}$，其可以表示为一个有向二分图 $G_S = (V, E)$，其包含了计划表 S 中班次的邻接信息。其中 $V = \{v_1, v_2, \ldots, v_n\}$ 和 $E \subseteq V \times V$ 分别为 G_S 的点集和边集。图 G_S 中的点 v_i 表示班次 t_i，边 $e_{i,j}$ 为连接点 v_i 和点 v_j 的边。图 G_S 的邻接矩阵表示为加权计划表邻接矩阵 $W = [w_{i,j}]_{m\times m}$，即如果在班次 t_i 和班次 t_j 之间有连接，其对应权重 $w_{i,j}$ of $e_{i,j}$ 为班次 t_i 和班次 t_j 之间的被分配的休息时间。图 G_S 的加权邻接矩阵 W 为:

$$w_{ij} = \begin{cases} st(t_j) - et(t_i) & \text{第}i\text{班次的下一班次为第}j\text{个班次} \\ 0 & \text{否则} \end{cases} \tag{3.1}$$

其中 st, et 分别为班次的开始时间和结束时间。并且根据具体的约束条件会构造一个约束矩阵 $C = [c_{i,j}]m \times m$，其中包含了班次之间的可连接信息，例如 $c_{i,j} = 1$ 表示第 i 和 j 个班次可以互相连接，若 $c_{i,j} = 0$ 则表示这两个班次不能互相连接。动作 $A_{i,j} \in \mathcal{A}$ 表示将第 i 和 j 个班次互相连接，同时这会将两名司机的任务结合，使得调度表减少一个司机。在具体的搜索过程中，智能体会根据当前状态 $S \in \mathcal{S}$ 和约束矩阵 C 计算当前状态的可行动作，若一个动作 $A_{i,j}$ 是可行的，其需要满足以下条件:

$$c_{i,j} \neq 0, \sum_{k=1}^{m} w_{i,k} = 0, \sum_{k=1}^{m} w_{k,j} = 0 \tag{3.2}$$

当智能体在当前状态上没有可执行动作时，该状态被称为终止状态 S_T，这也意味着不能够再减少司机。在本文的静态调度问题中，状态转移函数 $\mathcal{P} : \mathcal{S}\times\mathcal{A} \to \mathcal{S}$ 是已知的，即在一个状态 S 下采取确定动作 A 后，智能体转移到的下一状态是可观测并且是确定的。奖励函数 $r(S)$ 是用来评估智能体所到达状态 S 所获得的奖励，通常设置为对调度方案进行评价的目标函数 $f(S)$。为了

评估和对比算法的调度结果性能，BDSP 通常使用多目标优化问题 (Multi-objective Optimization Problem) 中常用的线性组合方式[75] 来构造目标函数 $f(S)$。具体来说，现实场景中会考虑以下子目标:

- 司机总数 $|S| = n$，公交车司机希望使用更少的司机来完成调度任务以节约成本;
- 司机工作时间的方差 $Var(S) = \frac{1}{n}\sum_{k=1}^{n}(g_k - g_{aver})^2$，其中 $g_{aver} = \frac{1}{n}\sum_{k=1}^{n} g_k$ 为司机的平均工作时长，司机们希望其工作量都与其同事接近;
- 司机的过度工作时间 $T(S) = \sum_{k=1}^{n} \max\{0, g_k - \hat{g}\}$，其中 $\hat{g}$ 为标准工作时间，g_k 为第 k 个司机的工时，司机希望其每天的工作时长不能超过标准工时;
- 单向轮班的个数 $D(S)$。司机的轮班需要在相同的地点出发和结束，否则司机需要在多执行一个没有乘客的班次返回出发地点，这时公交车司机需要增加成本来完成这样一班次。
- 总工作时长 $P(S) = \sum_{k=1}^{n} g_k$，公交车公司希望使用更短的总工时来完成给定的任务，以节约汽车消耗成本。

因此，目标函数 $f(S)$ 设计为这些子目标项的线性组合。

$$f(S) = \alpha_1|S| + \alpha_2 Var(S) + \alpha_3 T(S) + \alpha_4 D(S) + \alpha_5 P(S) \tag{3.3}$$

其中 $\alpha_i \le 0, i = 1, \ldots, 5$ 为对应子目标项的系数，公交车公司可以自由地通过控制其系数来调节各个子目标项的重要程度。因此，BDSP 问题是找到满足约束条件的解 S，并且最小化目标函数 $f(S)$，该问题可以表示为以下最优化问题:

$$\begin{aligned} \min \ & f(S) = \alpha_1|S| + \alpha_2 Var(S) + \alpha_3 T(S) + \alpha_4 D(S) + \alpha_5 P(S) \\ s.t. \ & \begin{cases} \sum_{i=1}^{m} w_{i,j} \in \{0,1\}, & (j = 1, 2, \cdots, m) \\ \sum_{j=1}^{m} w_{i,j} \in \{0,1\}, & (i = 1, 2, \cdots, m) \\ w_{i,j} = \begin{cases} w_{i,j} = 0 & c_{i,j} = 0 \\ w_{i,j} \le 0 & c_{i,j} = 1 \end{cases}, & (i, j = 1, 2, \cdots, m) \\ 0 < g_k \ge \hat{g}, & (k = 1, 2, \cdots, n) \end{cases} \end{aligned} \tag{3.4}$$

综上所述，通过使用该目标函数 $f(S)$ 作为马尔科夫决策过程中的奖励函数，就可以为一个调度问题构造一个马尔科夫决策过程，因此将其转化为一个强化学习问题。

3.3.2 基于蒙特卡洛树搜索的调度算法

当给定 MDP 的状态转移函数时，蒙特卡洛树搜索是一个基于模拟搜索的 Rollout 算法，其包括四个主要步骤: 选择，扩展，模拟和反传。蒙特卡洛树搜索通过在有限的计算量内迭代地构造一个搜索树，当搜索结束会选出并返回最好的叶子节点作为解。在蒙特卡洛树搜索中的每

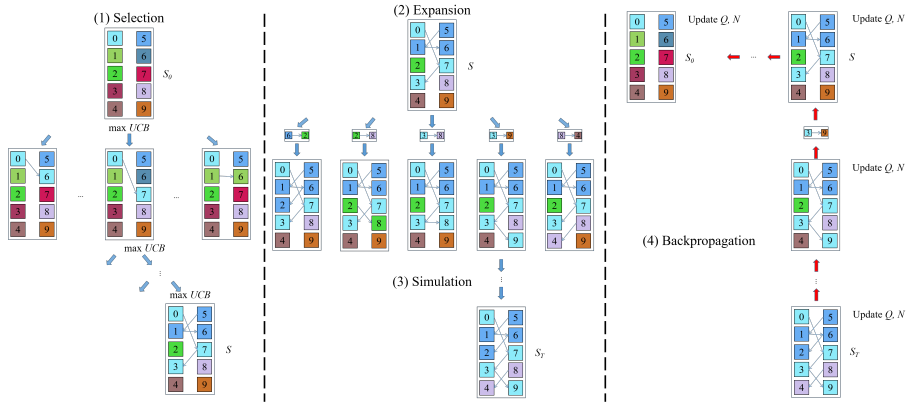


图 3.1　在 BDSP 上使用蒙特卡洛树搜索示例

一个节点表示一个状态 S，因此这些节点也为状态节点。搜索树的每一个边 $(S,A) \rightarrow S'$ 表示动作 A 连接的下一状态 S'。且边 $(S,A) \rightarrow S'$ 会保存状态的访问次数 $N(S')$ 和对应的动作价值 $Q(S,A) = \frac{1}{N(S')} \sum_{S'|S,A \rightarrow S'} V(S')$，其中 $V(S')$ 为从状态 S' 通过模拟得到终止状态的估计回报。本文以 BDSP 为例子展示如何将蒙特卡洛树搜索应用于调度问题，结合图3.1的示例，其四个主要步骤可以定义如下:

- 选择: 从当前的根节点 S_0 开始迭代地选择具有最大 UCB 值的子节点作为下一状态直到叶子节点。如果叶子节点为终止状态，智能体直接进行评估反传，否则进行扩展。

  UCB 值[76] 用来评估节点的价值:

$$UCB(S') = Q(S,A) + c\sqrt{\frac{2\ln N(S)}{1+N(S')}} \tag{3.5}$$

  其中状态 S 为状态 S' 的父状态节点，即通过在状态 S 下采取动作 $A((S,A) \rightarrow S')$。$Q(S,A)$ 为在状态 S 采取动作 A 的状态动作价值，$N(S)$ 为状态 S 的访问次数，c 为用来平衡利用和探索的常数项。如果一个状态节点从没被选择过，其 UCB 值将会变大而鼓励探索。如图3.1(1) 所示，智能体从根节点开始不断地选择具有最大 UCB 值的子节点作为下一状态直到当前节点的叶子节点。
- 扩展: 若叶子节点表示一个非终止节点，就基于当前可行动作，将其所有的子节点扩展加入搜索树中。在一个状态 S 的可行动作集合 $A(S,G_C)$ 需要满足公式3.2。因此，我们先通过约束图 G_C 推断出叶子节点的可行动作，并将其所有的可行动作都扩展进搜索树中，如图3.1(2) 所示。
- 模拟: 随机从扩展的子节点中选取一个节点，并使用随机策略快速模拟到终止状态 S_T。

$$S_T \leftarrow Rollout(S) \tag{3.6}$$

随机策略为在当前状态随机从可行动作集合中选择一个动作，并且这个策略会一直重复执行直到终止状态 S_T。在终止状态上，没有任何的可行动作能够被采取，如图3.1(3) 所示。

- 反传: 如图3.1(4) 所示，当智能体到达了一个终止状态 S_T，就对其使用奖励函数 $r(S_T)$ 进行评估并沿着轨迹反向传播其价值。实际中，对于轨迹上所有被选择过的节点，其访问次数 N 和状态动作价值 Q 会进行更新:

$$Q \leftarrow \frac{1}{N+1}(NQ + r(S_T)), N \leftarrow N + 1 \tag{3.7}$$

为了节约计算资源，我们使用了贪婪策略来生长搜索树。即当经过一定数量的搜索之后，基于收集到的信息使用一个贪婪策略 π_g:

$$\pi_g(S_0) = \arg \max_{A \in A(S_0, G_C)} Q(S_0, A) \tag{3.8}$$

使用该策略选择在状态 S_0 的最优动作 $\pi_g(S_0)$ 执行，并将其后续状态 $(S_0, \pi(S_0)) \rightarrow S_0^{new}$ 作为下一次搜索时新的根状态节点。当根节点状态为终止状态时，蒙特卡洛树搜索的过程就结束并返回最终的解。

3.3.3 GLVS: 基于蒙特卡洛树搜索和图约束下的低方差调度算法

随着问题规模和状态解空间变大，蒙特卡洛树搜索中的搜索树深度会变得很深，这时蒙特卡洛树搜索就会出现探索效率不足问题。因为直接对模拟结果得到的终止状态进行价值评估和价值反传会使得探索效率变得低下，因为随着状态的增多，智能体需要更多的模拟才能够探索到具有好的奖励的轨迹并学习。为了解决该问题，本节提出了 LVS，其是一种直接优化蒙特卡洛树模拟搜索得到的最终解 S_T 的算法，其能够有效地提升搜索树的探索效率。结合 BDSP 为例子，LVS 的输入为计划表 S_T，并输出一个更优的可行解 S_T^*，其拥有更低的方差，而不需要使用更多的计算量去执行多次模拟探索。本文将结合 BDSP 来展示 LVS 算法的执行过程，该算法可以应用于车间调度问题和学校课程表调度问题等其他调度问题中。最后，将 zuihouLVS 算法和 MCTS 结合得到本节提出算法 GLVS。

3.3.3.1 LVS: 低方差交换算法

LVS 的目标为降低当前解的方差并保持其可行性。因此，其核心思想为将当前解 $S_T = \{s_1, \ldots, s_i, \ldots, s_j, \ldots, s_n\}$ 中长的轮班 s_i 进行分解并用其和短的轮班 s_j 进行结合而形成新的可行解 $S_T^* = \{s_1, \ldots, s_i^*, \ldots, s_j^*, \ldots, s_n\}$，其有更低的工时方差 $Var(S_T^*)$。具体的，轮班的长度为轮班的总工作时长。在 LVS 中，需要根据约束图 C 进行优化来确保计划表的可行性。并且该过

算法 1 低方差交换算法 (LVS)

Input:

当前计划表 S，约束图 G_C;

Output:

新的计划表 S^*;

$S^* = S$

$(s_i, s_j) \leftarrow Select(S)$

if (s_i, s_j) 不为空值 **then**

    计算 $2l$ 个 $Decompose$ 和 $Swap\&Merge$ 结合的方差下降量，$\{\Delta_1^1, \Delta_2^1, \ldots, \Delta_1^l, \Delta_2^l\}$

    若没有组合能够满足约束，将对应的 Δ 设置为 0

    计算最优方差下降量

$$\Delta^* = \max_{sm=1,2,de=1,\ldots,l} \Delta_{sm}^{de}$$

;

    if $\Delta^* > 0$ **then**

        得到最优 $Decompose - Swap\&Merge$ 组合对应的方差下降量 Δ^*

        $(s_i^1, s_i^2) \leftarrow Decompose(s_i)$

        $(s_i^*, s_j^*) \leftarrow Swap\&Merge(s_i^1, s_i^2, s_j)$

        $S^* \leftarrow Update(S, s_i, s_j, s_i^*, s_j^*)$

        $S = S^*$，执行第二步

    else

        执行第二步

    end if

end if**return** schedule S^*

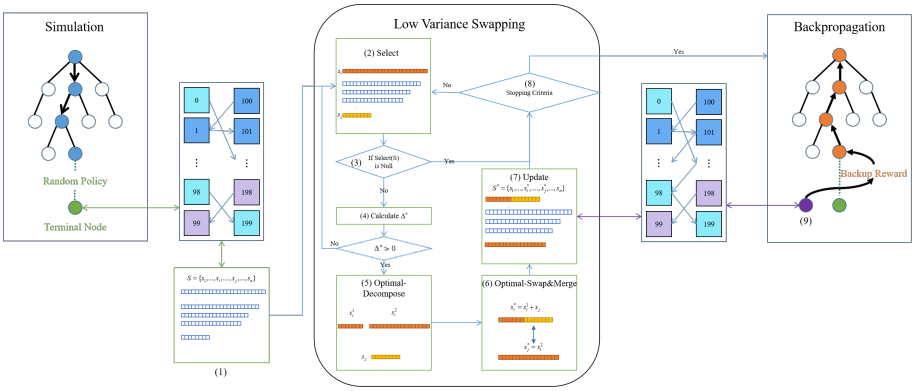


图 3.2　LVS 算法在 BDSP 中的示例

程会一致重复使用直到 $Var(S_T^*)$ 不能够在进一步地下降且满足可行约束。这是使用该优化的解作为最终结果返回。LVS 算法的具体步骤总结为算法1中，并且图3.2也显示了 LVS 在 BDSP 中的应用过程。具体的 LVS 可以分为以下四个步骤:

1. $(s_i, s_j) \leftarrow Select(S)$ 为了从给定计划表 S 中选择两个合适的轮班 s_i 和 s_j，我们规定所选择的轮班满足如下条件3.1，其要求轮班 s_j 的工作时间 $g(s_j)$ 要小于轮班 s_i 的工作时间 $g(s_i)$ 的一半:

条件 3.1：

$$g(s_j) < \frac{1}{2} g(s_i) \tag{3.9}$$

若在计划表 S 中没有两个轮班 s_i, s_j 满足条件3.1。$Select$ 函数返回一个空值来表示 S 不能被 LVS 继续优化。

2. $(s_i^1, s_i^2) \leftarrow Decompose(s_i)$ 在这一步中，将较长的轮班 s_i 分解为两个子轮班 s_i^1, s_i^2。为此需要分解的两个子班次 s_i^1 和 s_i^2 满足条件3.2。该条件分别给分解得到子班次 s_i^1 和 s_i^2 工作时长 $g(s_i^1)$ 和 $g(s_i^2)$ 的上界和下界:

条件 3.2：

$$\begin{aligned} g(s_j) \le g(s_i^1) \le g(s_i) - g(s_j) \\ g(s_j) \le g(s_i^2) \le g(s_i) - g(s_j) \end{aligned} \tag{3.10}$$

3. $(s_i^*, s_j^*) \leftarrow Swap\&Merge(s_i^1, s_i^2, s_j)$ 在这一步中，通过将第一步选择的轮班 s_j 和第二步分解得到的两个子轮班 s_i^1 和 s_i^2 之一进行交换和结合。若能够成功，这一步将获得两个新的轮班 s_i^* 和 s_j^*，并且其将更换原来计划表 S 中的轮班 s_i 和 s_j。因此生成一个新的且具有更低方差的轮班 S^*。在实际中，方差下降量 $\Delta = Var(S) - Var(S^*)$ 可以根据交换的是哪一个子轮班而进行如下计算。若轮班 s_j 和子轮班 s_i^1 进行交换并和子轮班 s_i^2 进行结合，其方差下降两可以进行如下计算:

$$\Delta_1 = Var(S) - Var(S^*) = \frac{2}{n} g(s_i^2)(g(s_i^1) - g(s_j)) \tag{3.11}$$

类似地，如果轮班 s_j 和子轮班 s_i^2 进行交换并和子轮班 s_i^1 进行结合，其方差下降两可以进行如下计算:

$$\Delta_2 = Var(S) - Var(S^*) = \frac{2}{n} g(s_i^1)(g(s_i^2) - g(s_j)) \tag{3.12}$$

这两个方差下降量的推导过程在定理3.1中给出。并且由于满足了条件3.1和条件3.2，方差下降量 Δ_1 和 Δ_2 都是非负的。并且所生成的两个新的轮班 s_i^*, s_j^* 也是可行的。即其满足约束图 G_C 中所有的约束。否则，会将方差下降量设置为 0，并直接进行下一轮的迭代优化。如图3.3所示，对被选择的两个轮班 s_i 和 s_j，假设存在 l 个可能的选择来进行 $decompose$ 并且同时满足条件3.2，每个选择又对应着两种 $Swap\&Merge$ 方式。因此，总共有 $2l$ $Decompose-Swap\&Merge$ 组合方式。对所有可能的结果进行分别评估 $\{\Delta_1^1, \Delta_2^1, \ldots, \Delta_1^l, \Delta_2^l\}$ 并选择其中最优的组合方式。$\Delta_{sm}^{de}(sm \in \{1,2\}; de \in \{1,\ldots,l\})$ 表示选择第 de 个 $Decompose$ 和第 sm 个 $Swap\&Decompose$ 所对应的方差下降量。其中最大的方差下降量 $\Delta^* = \max\limits_{sm=1,2,de=1,\ldots,l} \Delta_{sm}^{de}$，表示 $Decompose-Swap\&Merge$ 的最佳组合。若 Δ^* 为 0，这也表示没有可行的 $Decompose-Swap\&Merge$ 组合。因此会再重新进行 $Select$。否则就直接使用新生成的轮班更新计划表。

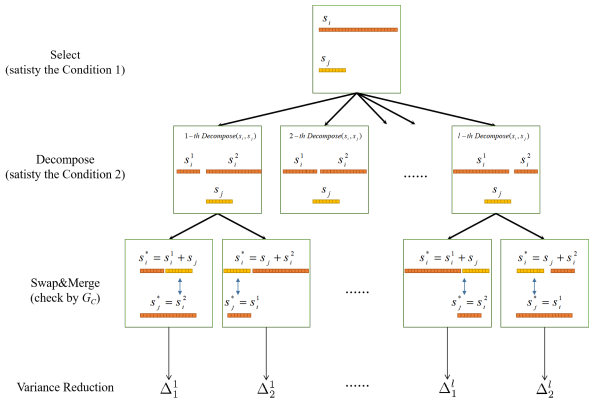


图 3.3　$2l$ 种 $Decompose$ 和 $Swap\&Merge$ 组合方式

4. $S^* \leftarrow Update(S, s_i, s_j, s_i^*, s_j^*)$ 在这一步中，直接将 S 中的 s_i 和 s_j 更新为 s_i^* 和 s_j^* 得到新的计划表 S^*。

3.3.3.2　算法理论分析

对于 LVS 算法可以通过有以下定理保证其性质。定理3.1保证 LVS 输出的新计划表比输入的计划表有更低的方差。

定理 3.1：LVS 算法能够将一个可行计划表 S 优化成一个新的可行计划表 S^*，其有着更低的方差。即 $Var(S^*) \leq Var(S)$，其中 $S^* \leftarrow LVS(S)$。

证明: 本定理的具体证明过程见A.1。 □

3.3.4　GLVS 复杂度分析

在本节中，我们结合 BDSP 对 GLVS 算法进行博弈复杂度和计算复杂度分析。

博弈复杂度　公交车司机调度问题被看做一个具有完美信息的单人序列博弈。对于一个具有完美信息的博弈，其相关的复杂度有状态空间复杂度和博弈树复杂度[77]。公交车司机调度问题的状态空间复杂度定义为从初始状态开始能够到达的合法状态数目。博弈树复杂度为从初始状态出发能够到达搜索树叶子节点的数量。对于一个特定的公交车司机调度问题，假设在最终计划表里有 m 个班次和 n 个司机。$m-n$ 为博弈的长度，b 为搜索树的平均分支因子。博弈树的复杂度为 $b^{(m-n)}$。公交车司机调度问题的实际状态空间复杂度是非常估计的。因此我们放宽约束限制，将公交车司机调度问题看做将 m 个班次分给 n 个司机的问题，因此可以使用第二类斯特林数 (Stirling Number) 对其进行估计:

$$StirN(m,n) = \frac{1}{n!}\sum_{q=0}^{n}(-1)^q\binom{n}{q}(n-q)^m \tag{3.13}$$

计算复杂度 对 GLVS 的时间复杂度进行粗略的估计。实际中，假设 GLVS 的迭代搜索次数为 I，每一次迭代搜索包含五步。D 表示树结构的深度，B 表示树结构的平均分支因子，L 表示 LVS 的平均迭代次数。GLVS 的时间复杂度可以表示为 $O(ID(selection + expansion + simulation + LVS + backpropogation))$。$Selection$ 的时间复杂度为 $O(DB)$。$Expansion$ 的时间复杂度为 $O(B)$。$Simulation$ 的时间复杂度为 $O(D)$。LVS 的时间复杂度为 $O(L)$.$Backpropogation$ 的时间复杂度为 $O(D)$。因此，GLVS 的时间复杂度为 $O(ID(DB+L))$。实际中 I, B, D 会根据具体解决的问题变化。

3.4 数值实验

本文为 Rollout 算法中基于模型模拟的蒙特卡洛方法在具有巨大状态搜索空间的强化学习任务时所面临的探索效率低下问题提出新的强化学习算法 GLVS。为了验证算法的有效性和对比其相对于流行算法的优势，本文选择同样具有巨大解搜索空间的 BDSP 上进行数值实验。

3.4.1 实验设定

数值实验所使用数据集南京公交车交通公司所提供的为六个数据集[1]。每个数据集中包含在一定时间范围内的所有班次的信息，包含每个班次的出发结束的时间地点，具体信息总结为表3.1。

表 3.1 数据集的简要信息

数据集名称	总班次数	时间范围
NK172	172	05:50-22:29
NK220	220	05:50-22:02
NK224	224	05:40-22:19
NK396	396	05:40-22:29
NK440	440	05:40-22:02
NK616	616	05:40-22:29

为了评估算法在不同评估目标下的有效性，本文使用了三组不同的目标函数系数来进行实验，其具体系数设置为表3.2所示。

表 3.2 不同目标函数的参数设计

objective function	$\alpha_1(\|S\|)$	$\alpha_2(Var(S))$	$\alpha_3(T(S))$	$\alpha_4(D(S))$	$\alpha_5(P(S))$
$f_1(S)$	100	360	1	25	0.5
$f_2(S)$	100	1800	1	25	0
$f_3(S)$	100	36	1	25	5

[1]该数据集的下载链接为:https://github.com/QINGYWuuu/GLVS

3.4.2 主要实验结果

本节实验为对比 GLVS 在解决具有巨大搜索空间问题上的算法性能，并在具体的 BDSP 问题上进行实验。主要对比的基准算法有蒙特卡洛树搜索和其他三种流行调度算法:Hungarian[64]，GRASP[63] 和 GrBDSP[62]。在每个数据集上进行 10 次独立的实验，并对比最优性能和平均性能。表3.3和表3.4分别给出了所有算法的最优性能和平均性能，即目标函数值。通过结果表明，在总体上 GLVS 取得最好的性能。例如，在问题规模最大的 NK616 数据集上取得最好的性能(20410)，而第二好的算法为 GrBDSP，其目标值为 21923，其相对与 GLVS 大约低了 7.4%。并且，在同样的计算量下，蒙特卡洛树搜索在规模大的调度问题上的性能不如其在规模较小的调度问题上好。但是，LVS 能够有效在不同的目标函数设定下提升性能。例如在表3.4中数据集 NK660 结果表明了，在 f_2 目标函数下，GLVS 能够提升蒙特卡洛树搜索约 42.6% 性能，从 30408 优化到 17444。表3.5展示了在 NK172 数据集上不同目标函数下各个子目标项的值。当使用 f_1 为目标函数时，蒙特卡洛树搜索和 GLVS 相比其他算法能够取得更好的性能。特别地，GLVS 在所有目标项都能取得最好的性能。并且在使用 f_2 为目标函数时，GLVS 能够得到最好性能的同时有最小的方差。当使用 f_3 为目标函数时，GLVS 能取得最好的性能，并且有最小的总工作时间。并且，通过比较蒙特卡洛树搜索和 GVLS 可以发现，LVS 不仅能够有效降低方差，还能够降低其他子目标的值。

表 3.3 GLVS 和 MCTS 与基准算法在不同目标函数下的最优性能对比

Best Performance	Hungarian[64]	GRASP[63]	GrBDSP[62]	MCTS(ours)	GLVS(ours)
f_1					
NK172	11803	8776	7868	8021	**6739**
NK220	13445	9792	9647	10163	**8444**
NK224	12031	10124	8107	8874	**6899**
NK396	20500	16063	14088	16793	**12832**
NK440	21609	17200	15713	19274	**15685**
NK616	29743	23198	21923	26231	**20410**
f_2					
NK172	27917	8882	9097	10112	**5845**
NK220	25144	10910	10683	11471	**7833**
NK224	26414	9666	9487	10072	**4815**
NK396	37278	13247	13079	21756	**10468**
NK440	35851	14533	14085	20777	**13966**
NK616	44053	18202	17324	30071	**17294**
f_3					
NK172	40165	39306	39281	38470	**38016**
NK220	49165	48438	48422	47094	**47033**
NK224	41253	40720	40662	**38669**	38985
NK396	81083	79313	79307	79740	**75554**
NK440	89661	87823	87857	88106	**84895**
NK616	129748	127077	127026	129585	**121823**

3.4.3 消融实验

本小节通过 GLVS 消融实验来验证 LVS 的有效性。通过图3.4(a) 对比蒙特卡洛树搜索和 GLVS 获得最终解的方差可知，在所有数据集上，LVS 能够有效地降低司机工作时间的方差。并且，图3.4(b) 表示了蒙特卡洛树搜索和 GLVS 的在 NK172 上使用 f_1 为目标函数的性能曲线，

表 3.4　GLVS 和 MCTS 与基准算法在不同目标函数下的平均性能对比

Mean Performance	Hungarian [64]	GRASP [63]	GrBDSP [62]	MCTS(ours)	GLVS(ours)
f_1					
NK172	13083	9424	8209	8443	**6973**
NK220	13658	10667	9839	10795	**8706**
NK224	12499	10702	8490	9341	**7077**
NK396	21169	16731	14508	17643	**14252**
NK440	22040	18317	**16271**	20008	17053
NK616	30087	24594	22333	27953	**21366**
f_2					
NK172	26054	9428	10026	10472	**5477**
NK220	25993	11227	11055	12674	**8456**
NK224	27842	10229	10061	10706	**4931**
NK396	37723	13801	13477	22360	**10995**
NK440	36200	14812	**14612**	21139	14700
NK616	44432	18494	17617	30408	**17444**
f_3					
NK172	41558	39510	39473	38771	**38259**
NK220	49278	48584	48527	47643	**47108**
NK224	41585	40940	40915	39160	**39073**
NK396	81193	79657	79428	80829	**76052**
NK440	89817	88111	88043	88631	**85015**
NK616	130152	127508	127174	130095	**121939**

表 3.5　GLVS 与基准算法在 NK172 上的最优性能和各个子目标项值

Method	Objective Values	$\|S\|$	$Var(S)$	$T(S)$	$D(S)$	$P(S)$
$f_1(S)$						
Hungarian	11802.70	32	10.92	584	18	7275
GRASP	8776.10	27	5.51	254	8	7277
GrBDSP	7868.40	28	3.29	74	6	7320
MCTS	8020.70	27	4.32	48	6	7135
GLVS	**6738.60**	**26**	**1.66**	**0**	**0**	**7082**
$f_2(S)$						
Hungarian	27916.84	31	13.06	854	18	7156
GRASP	8881.50	31	3.04	13	12	7182
GrBDSP	9097.16	30	3.17	189	8	7243
MCTS	10111.68	29	3.81	159	8	**7150**
GLVS	**5845.32**	**27**	**1.72**	**0**	**2**	7213
$f_3(S)$						
Hungarian	40164.47	31	4.01	160	12	7292
GRASP	39306.10	26	2.78	36	6	7284
GrBDSP	39281.22	31	**2.28**	24	8	7175
MCTS	38469.51	27	5.63	42	4	7085
GLVS	**38015.90**	**27**	2.39	**0**	**4**	**7026**

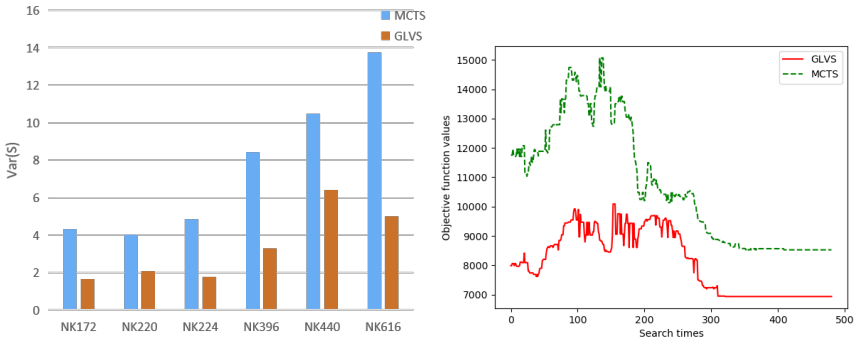
GLVS 能够取得比 MCTS 更好的性能。GLVS 能够比蒙特卡洛树搜索快速地搜索到不错的调度方案，以及快速地收敛。

3.4.4　参数敏感度实验

本小节将探究迭代次数对 GLVS 的影响。通过在 NK172 数据集上，采用 f_1 为目标函数。图3.5(a) 和 (b) 分别表示迭代次数对 GLVS 性能和时间的影响。随着迭代次数上升，GLVS 的性能进行有效地提升，并在迭代次数为 18 时取得最好的性能。但 GLVS 算法运行时间也会随迭代次数的上升增加。

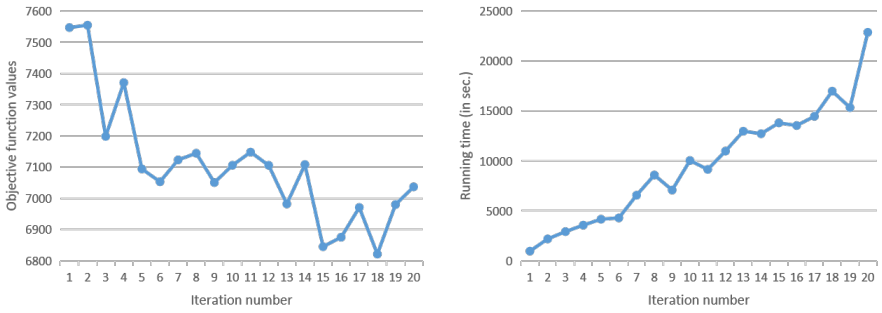
3.5　本章小结

本章提出了一种新的强化学习算法, GLVS。其有效地解决了 Rollout 算法使用模型模拟决策时在面对状态空间巨大的强化学习问题中探索性能低下问题。并且结合现实生活场景中常见和



(a) (b)

图 3.4 MCTS 和 GLVS(a) 在不同数据集上最终调度方案的方差对比和 (b) 随着探索次数的目标函数值曲线



(a) (b)

图 3.5 迭代次数对 GLVS(a) 目标函数值性能的影响和 (b) 算法执行时间的影响

状态空间巨大且为 NP-hard 的调度问题进行算法有效性的验证。通过理论分析和在具体的 BDSP 上进行的数值实验可知本文所提出的 GLVS 算法能够取得比基准算法更好的性能。本章所解决的是静态调度算法，而蒙特卡洛树搜索在解决动态调度问题时是具有优势的，因为其可以在原有搜索树的搜索结果上继续进行搜索而不需要完全重新搜索，这也是后续的研究方向。

第四章 Alter DQN: 基于 UCB 算法的集成深度 Q 网络

4.1 背景介绍

在强化学习中[9]，智能体需要与真实环境进行交互产生数据。并对这些数据进行有效的学习，即学习如何在当前状态下采取最优动作以最大化未来累积奖励。当智能体没有被给定环境模型时，其通常需要使用 Rollout 算法中基于时序差分的状态估计方法来估计当前状态下动作的价值来选择最优动作。这种直接对环境交互得到的真实数据进行学习的方式也称作无模型强化学习算法，这类算法中最为著名的算法为 Q 学习[10] 和 SARSA[9] 等，且其都已经在简单的强化学习任务上取得了不错的效果。深度 Q 网络[18,34] 是将人工神经网络和 Q 学习进行有效的结合，以利用神经网络强大的表示能力来处理图像等 Q 学习无法处理的高维数据。随着近年来深度学习的快速发展，在强化学习算法的研究中，对深度 Q 网络研究及改进是最为流行的方向之一。在实际的强化学习任务中，智能体常常处于未知的环境中，并且其所能够获得关于环境的信息非常少，其只能通过之前获得的交互经验来学习和提升自己的交互策略[9]。并且智能体还要面临强化学习中独有的挑战，即探索与利用的平衡。智能体需要对不同的环境状态进行充分的探索，以提升未来的奖励。智能体可以牺牲一些眼前即得的奖励，以使得最终获得的奖励最大化。因此，无模型强化学习算法通常需要大量的交互数据才能够获得有价值的交互数据。因此一个有效的探索策略能够让智能体进行高效的探索，以获取有价值的交互数据而提升智能体的训练效率。

4.2 相关工作

在强化学习中，深度 Q 网络是代表性的深度强化学习算法之一，并且当前的众多工作都是基于其基础而发展而来的。如为解决过估计偏差的 Double DQN[37]，其将 Double Q 学习算法[36] 应用到深度 Q 网络上。深度 Q 网络算法通常使用 ϵ-贪婪策略[9] 来进行探索。但在复杂环境中，该方法的探索有效性是非常低的。现有的方法都致力于解决此问题，如 Nosiy DQN[39] 使用了参数化噪声网络来增强探索性。集成强化学习使用了多个独立初始化的 Q 网络，多个网络之间的差异自然就给智能体带来了随机性，从而有效提升算法的探索性。如 Bootstrapped DQN[45] 使用了多头 Q 神经网络，其是基于汤普森采样的思想对深度 Q 网络的改进。UCB Exploration[47] 基于 UCB 算法通过对状态动作价值的均值和方差构造 UCB 值来提升探索性，其更加侧重于动作维度上的利用和探索。[78] 则是对网络更新样本的采样方式进行改进，其基于 UCB 算法对经验回访池的采样策略进行了改进。

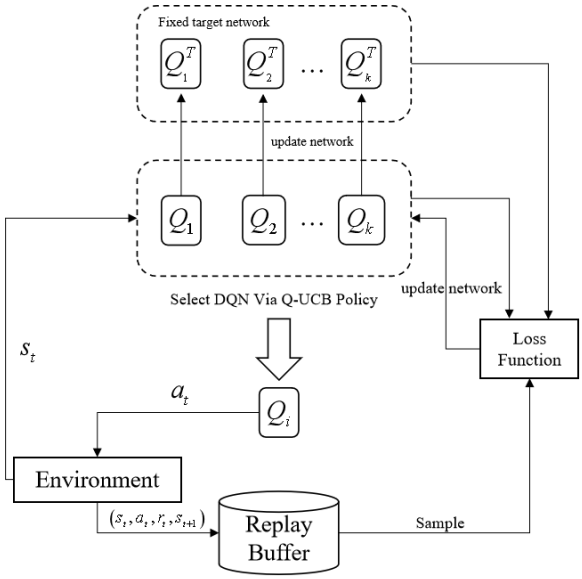


图 4.1　基于 UCB 算法的集成深度 Q 网络算法流程图

4.3　提出方法

在现有的集成强化学习方法中，其在对环境进行探索时，所采取的动作都是基于多个 Q 网络的共同决策。但这样的探索交互方式较为单一，当多个 Q 网络收敛时，其探索性会下降。因此，为了解决该问题，本章提出了一种新的探索策略，UCB-Q 探索策略。其通过让多个 Q 网络轮流进行单独决策并共同探索，而有效地提升探索效率。

4.3.1　UCB-Q 探索策略

先独立初始化 K 个深度 Q 网络 $\{Q_i(s,a;\theta_i)\}_{i=1}^{K}$。在每个时刻 t 时，智能体会根据当前经验回放池 B 中收集的样本情况和在状态 s_t 下 Q 网络的估计值来选择出一个深度 Q 网络进行决策。具体来说，所使用的策略为 UCB-Q 探索策略。对于第 i 个 Q 网络其拥有对应的 $UCB(Q^i)$ 值:

$$UCB(Q_i) = \max_a Q_i(s_t, a_t|\theta_i) + c\sqrt{\frac{2\ln N}{1+N_i}} \tag{4.1}$$

其中 N_i 为经验回放池中 Q^i 所产生样本的个数，N 为当前经验回放池中的样本总数，c 为平衡利用和探索的常数项。再根据计算得到的 K 个 UCB-Q 值中选择最大的深度 Q 网络进行动作的选取。

4.3.2 基于 UCB 算法的集成深度 Q 网络

基于所提出的 UCB-Q 探索策略，为了稳定集成 Q 网络的学习。对于 $\{Q_i(s,a;\theta_i)\}_{i=1}^K$ 还使用了目标集成网络 $\{Q_i^-(s,a;\theta_i^-)\}_{i=1}^K$。详细来说，第 j 个深度 Q 网络的损失函数为

$$L(\theta_j) = \mathbb{E}_{(s,a,r,s')}[(r + \gamma \max_{a'} Q_j^-(s',a';\theta_j^-) - Q_j(s,a;\theta_j))^2] \tag{4.2}$$

其中 (s,a,r,s') 为从经验回放池中采样的数据。因此提出的算法为 Alter DQN，其算法流程图如图4.1所示，且其算法步骤总结在算法2中。

算法 2 基于 UCB 算法的集成深度 Q 网络 (Alter DQN)

Input:

K 个独立初始化的深度 Q 网络 $\{Q_i(s,a;\theta_i)\}_{i=1}^K$，$K$ 个独立初始化的深度 Q 目标网络 $\{Q_i^-(s,a;\theta_i^-)\}_{i=1}^K$，经验回放池 B;
for 每 **do** 一幕的交互
    获取环境的初始状态 s_0
    for 每 **do** 一时刻 $t = 1, 2, \ldots$
        根据公式4.1计算出 K 个深度 Q 网络的 UCB-Q 值
        选择 UCB-Q 值最大的深度 Q 网络采取贪婪策略选取动作 a_t
        执行动作 a_t 获得奖励 r_t，并且智能体转移至状态 s_{t+1}
        将获得的数据 (s_t, a_t, r_t, s_{t+1}) 存储进经验回放池 B 中
        更新深度 Q 网络的选择次数

$$N \leftarrow N + 1, N_i \leftarrow N_i + 1$$

        从经验回放池中均匀采样并更新网络 $\{Q_i(s,a;\theta_i)\}_{i=1}^K$
    end for
end for

4.4 仿真实验

在本节中，将在不同的实验环境下对比现有算法，以证明本文提出算法的有效性。为了公平对比，所有算法所使用相同的深度学习框架，且参数等设置都一致，如历史经验回放池的大小、神经网络的结构、学习率、优化器等。

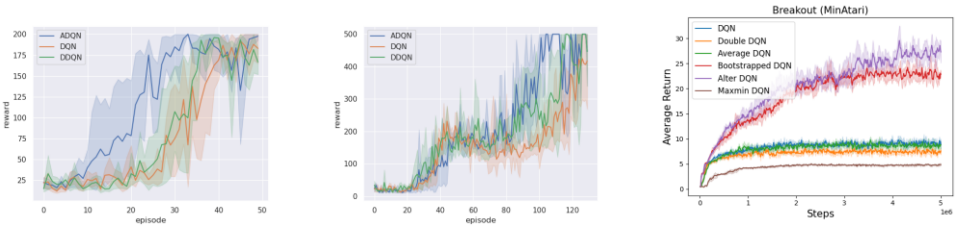
4.4.1 实验设置

本文选择在三个不同的标准强化学习实验环境验证算法的有效性，所选择的实验环境有经典强化学习问题环境 gym[79] 中的 CartPole-v0 和 CartPole-v1，以及 MinAtar[80] 中的 Breakout 环境。其中 CartPole-v0 和 CartPole-v1 是经典的强化学习实验环境，本文在此环境上选择的对比基准算法为 DQN[18] 与 DDQN[37]。在算法参数设置方面，使用激活函数为 ReLU 的三层全连接神经网络。其训练长度分别为 50 和 125 个 episode。具体的参数设置如表4.1所示。在 MinAtar

表 4.1　实验环境的参数设置

参数名	设置值	
	CartPole	MinAtar
经验回放池大小	10,000	100,000
Batch Size	32	32
目标网络更新间隔	100	1,000
γ	0.99	0.99
学习率	0.001	0.0025

的 Breakout 环境上选择的五个不同的基准算法分别为 DQN[18]，DDQN[37]，Maxmin DQN[43]，Averaged DQN[42] 和 Bootstrapped DQN[45]。在算法参数设置方面，使用激活函数为 ReLU 的五层卷积神经网络。其训练长度为 5e6 个 steps。所有使用 K 个网络的算法都设置为一致。具体的参数设置如表4.1所示。本文提出的算法在经典强化学习环境上 (CartPole-v0/v1) 使用 $K = 2$ 个独立随机初始化的深度 Q 网络，而在 MinAtar 强化学习环境 (Breakout) 上使用 $K = 5$ 个独立随机初始化的深度 Q 网络。



(a)　(b)　(c)

图 4.2　在 (a)CarPole-v0，(b)CarPole-v1 和 (c)MinAtar-Breakout 上的 Alter DQN 和基准算法的性能对比

4.4.2　实验结果及分析

在本文实验中，对每种算法在每个环境上进行了五次独立的实验，因此得到如下性能曲线。根据分析在经典强化学习问题上（CartPole-v0 与 CartPole-v1）的实验结果可知。由图4.2(a) 可知，相比 DQN 和 DDQN，Alter DQN 在初始阶段的探索能力更强，其样本效率更高，并在 30 个 episode 时就趋近于收敛；而 DQN 和 DDQN 其样本效率较低，分别在大约 35 和 40 个 episode 时趋近于收敛。分析图图4.2(b) 的结果可知，Alter DQN 在接近 100 个 episode 时就接近收敛到最高分；相比之下，DQN 和 DDQN 样本效率低，需要在 120 个 episode 才能收敛。由这两个经典的强化学习环境上的实验结果可知，ADQN 相比基准算法探索能力更强，且样本效率更高，性能更好。通过分析在图4.2(c) 中展示在 MinAtar-Breakout 环境上的实验结果可知，Alter DQN 和 Bootstrapped DQN 效果都明显优于其他基准算法。并且在大约 3e6 个 steps 的时候 Alter DQN 开始优于 Bootstrapped DQN。

4.5 本章小结

本章针对 Rollout 算法中基于时序差分的状态估计方法在解决具有稀疏奖励的强化学习问题时智能体探索效率低下问题提出了一种基于最大置信度上界的交替深度 Q 网络算法。不同于已有探索策略，本文希望通过多个独立初始化的 DQN，并通过新构造的最大 UCB 策略算法来控制选择单个智能体与环境进行交互。本文构造的 UCB 项希望能够平衡对当前网络的利用和探索，通过该策略能够提升智能体的探索效率以提高算法性能。通过在多个不同的标准强化学习环境上的实验结果证明了本文提出的算法的有效性。

第五章 TEAM-Q 学习: 一种高效探索与学习的集成强化学习框架

5.1 背景介绍

当智能体使用 Rollout 算法在解决环境模型未知的强化学习问题时，其通常使用基于时序差分的状态价值估计方法，如 Q 学习[10]。但由于复杂强化学习任务中巨大的状态空间会使得 Q 学习变得不可行。但随着近十年深度学习的成功发展，深度强化学习已经在许多复杂强化学习任务上取得了成功。深度神经网络具有强大的泛化能力能够解决那些阻挡传统强化学习算法的高维的状态空间。但是，利用与探索的平衡仍然是强化学习中最基础的挑战。探索可以被视作对于未知奖励的搜索和为已发现奖励搜索捷径。解决该问题最有效的一类方法是集成强化学习，其利用集成 Q 函数或者集成策略函数来进行有效探索。UCB Exploration[47] 结合了 UCB 算法和不确定度估计，并将其应用到探索中。并且基于此，SUNRISE[48] 结合了这种探索方式了加权贝尔曼更新来减少误差传播。集成强化学习中最著名的算法为 Bootstrapped DQN[45]，其利用集成 Q 函数来进行时序持续性探索 (Temporally-Consistent Exploration)。即在每一幕的开始就采样一个 Q 网络，并在这一幕中一直使用它与环境进行持续性交互产生数据。现有的许多工作[39,46,81–83] 也都采取了这种探索方式。但是，时序持续性策略的探索能力是相对有限的，且其一般与集成大小成线性关系。因此，其在指数级的状态空间中是很难进行多样性探索。集成强化学习还能够用来优化标准的贝尔曼更新算子。如降低估计偏差[35] 和误差传播[84]，其可以通过不同的方式[36,42–44,85] 来解决过估计偏差。

但是当智能体所解决的强化学习问题具有稀疏奖励时，智能体的探索效率以及价值函数学习效率会变得十分低下。为了解决以上问题，本章提出了一种高效探索和学习的集成强化学习框架，其中包含了时序变化性探索策略和 Expected-Max 集成更新算子。在具体实现时，基于提出的两点进一步地提出 TEAM Q 学习算法。并且对于高维的强化学习任务，结合深度神经网络提出了两种 TEAM Q 学习的深度算法，TEAM DQN 和 DP-TEAM DQN。并且通过在多个实验环境上的实验结果表明，提出的算法能够取得先进性能。

5.2 相关工作

5.2.1 集成探索策略

在强化学习中,集成方法常用作于提升探索有效性。一个代表性的例子为 Bootstrapped DQN[45]，其是汤普森采样到强化学习中的自然拓展。其从一个多头深度 Q 网络中使用一个输出来执行时序持续性探索。并且不同的输出头使用不同的数据进行训练。基于此和基于贝叶斯线性回归，

Bootstrapped Prior DQN[46] 引入一个随机先验网络参数来提升探索性能。UCB Exploration[47] 基于 UCB 算法提出一种新的探索机制，其融合了 Q 集成的均值和方差。基于此，SUNRISE[48] 使用不确定性[84] 提出加权贝尔曼更新算子来减小误差传播。Disagreement[86] 提出了一种基于不同的探索方式，其使用多个动态模型的不确定性。

5.2.2 集成更新算子

集成方法也常用作构造一个合适的更新方式，如减小过估计偏差和误差近似方差。Average DQN[42] 使用 Q 集成来构造一个 Max-Expected 集成差分目标值来稳定训练过程。Double DQN[36,37] 通过使用双重估计器来解决过估计误差。Ensemble Bootstrapping DQN[44] 提出了集成估计器，其可以看做双重估计器的泛化形式。Maxmin DQN[43] 和 REDQ[85] 基于 Q 集成来构造一个最小 Q 函数以减小过估计误差。

5.3 提出方法

5.3.1 时序变化集成探索策略

在现有的强化学习算法中，集成强化学习方法已经被证实其能够有效地提升探索效率。在集成强化学习中，通常会有一个集成策略 $\Pi = \{\pi_i\}_{i=1}^{K}$，其中包含 K 个不同且独立的策略函数。因此，其核心问题为如何将这些不同的策略函数有效的聚合使用以进行多样的探索。大多数现有的集成方法在对环境探索时采用了时序持续性策略[45,46]，即智能体在收集整条轨迹时会持续使用同一策略函数与环境进行交互。正式地，使用 Π_w 表示用一个分布 $w = \{w_i\}_{i=1}^{K}$ 为采样分布支撑的 Π，其中 w_i 为选择策略函数 π_i 的概率。时序持续性探索策略会在一幕的开始选择一个策略函数 $\pi_k \sim \Pi_w$，并使用该策略与环境进行整幕的交互:

$$s_1, a_1, s_2, a_2, \ldots, a_{T-1}, s_T \tag{5.1}$$

其中 $a_t \sim \pi_k(\cdot|s_t)$。不同策略函数能够访问的状态通常是互补的，因此其探索到更多样的状态。但时序持续性探索策略的探索能力是相对有限的，因为其探索能力通常与策略函数的个数呈线性关系。这也意味着，这种集成探索策略对于高维的强化学习任务是低效的，因为通常状态的数量随着状态的维度呈指数级上升。因此，本章先提出了一种简单且有效的集成探索方式，其探索能力能够随着策略函数的个数而呈指数级上升。新的探索方式以时序变化方式结合多个策略函数，其在每一步重新选择一个新的策略函数。正式地，在每一个时间步 t，一个新的策略 π_k^t 会从集成策略 Π_w 中采样出来。因此，所收集的轨迹策略:

$$s_1, a_1, s_2, a_2, \ldots, a_{T-1}, s_T \tag{5.2}$$

其中 $a_t \sim \pi_k^t(\cdot|s_t)$，$\pi_k^t \sim \Pi_w$。通过这样的探索方式，$K$ 个不同的策略函数能够基于之前探索到的状态，并进行更深和更多样的探索。图5.1展现了不同集成探索方式在一个格子世界上的探索性能，其中越能到达的状态会越蓝，而越不能到达的状态会越白。平均访问频率的计算方式为

$$\frac{1}{T}\sum_{t=0}^{T}\rho_t(s) \tag{5.3}$$

其中 T 为一幕中的步数，$\rho_t(s)$ 为智能体在时刻 t 能到达状态 s 的概率。如图5.1所示，使用时序变化集成探索策略所探索到的状态比时序持续集成探索策略更多样且更有效。初始状态为 11×11 格子世界的中心点。智能体和环境交互的幕数设计为 100，每一幕的长度 T 设置为 $1,000$。ϵ-贪婪策略的探索因子设置为 $\epsilon = 0.01$，策略集成大小为 $K = 5$。

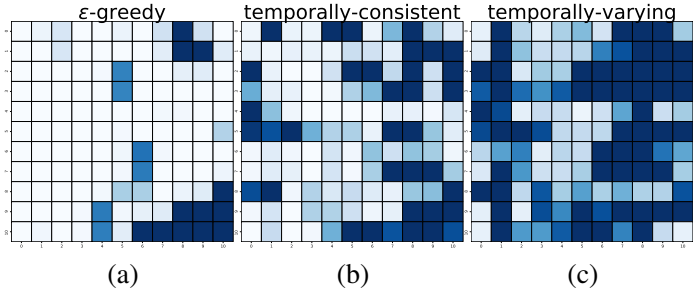


图 5.1　在格子世界问题上 (a)ϵ-贪心策略，(b) 时序持续性集成探索策略 ($K = 5$) 和 (c) 时序变化性集成探索策略 ($K = 5$) 的平均访问频率

5.3.2　Expected-Max 集成更新算子

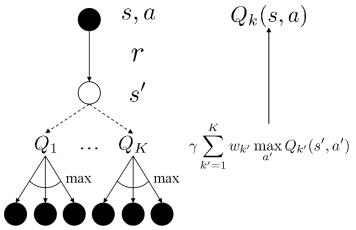


图 5.2　Expected-Max 集成更新算子的更新流程图

集成强化学习还被经常使用在优化标准贝尔曼更新算子。通常在集成强化学习中，通常会构造集成 K 个价值函数的 Q 集成 $\{Q_i\}_{i=1}^N$ 和策略函数的 Q 集成 $\Pi_w = \{\pi_i\}_{i=1}^K$，其中 π_k 为 Q_k 的导出策略。因此核心问题为如何利用包含在价值函数中包含的已知信息来提升学习有效性。强化学习的探索不仅被看作对未知奖励的搜索，以及对已发现奖励捷径的搜索。因此本章提出了 Expected-Max 集成算子。正式地，先从 Q 集成 $\{Q_i\}_{i=1}^K$ 中选择一个状态动作价值函数 Q_k。对于

更新样本 (s,a,r,s')，并且计算 Expected-Max 目标值 $y^{Expected-Max}$:

$$y_k^{Expected-Max} = r + \gamma \sum_{i=1}^{K} w_i \max_{a'} Q_i(s',a') \tag{5.4}$$

并且更新 Q_k:

$$Q_k(s,a) \leftarrow Q_k(s,a) + \alpha(y^{Expected-Max} - Q_k(s,a)) \tag{5.5}$$

Expected-Max 集成算子的更新图如图5.2所示。在对 Q 集成更新时，会依次选择每个 Q 函数，整个算子的更新算子总结于算法。Ensemble Q 学习[42] 算法提出了新的 Max-Expected 集成更新算子，其集成所有的状态价值函数来更新 Q_k:

$$Q_k(s,a) \leftarrow Q_k(s,a) + \alpha(r + \gamma \max_{a'}(\sum_{i=1}^{K} w_i Q_i(s',a')) - Q_k(s,a)) \tag{5.6}$$

对于此算子，所提出的算子能够保持每个状态动作价值函数的独立估计。因此其能够有效传输已知信息来提升学习效率。

5.3.3 TEAM-Q 框架

5.3.3.1 TEAM Q 学习

算法 3 TEAM Q 学习

输入： Q 集成 $\{Q_k(s,a)\}_{k=1}^K$; 固定分布 $\{w_k\}_{k=1}^K$; 学习率 α; 奖励折扣因子 γ; 探索因子 ϵ;

for 每一幕 **do**
  得到环境的初始状态 s_0
  for 对每一个时刻 $t = 1, 2, \cdots$ **do**
    选择 $Q_{k'}(s,a)$，其中 $k' \sim \{w_k\}_{k=1}^K$
    根据基于 $Q_{k'}$ 的 ϵ-贪婪策略采取动作 a_t，获得奖励 r_t 和下一状态 s_{t+1}
    计算 Expected-Max 目标值

$$Y^{\mathrm{EM}} = r_t + \gamma \sum_{k'=1}^{K} w_{k'} \max_{a'} Q_{k'}(s_{t+1},a')$$

    for 对每一个 Q 函数 $k = 1, 2, \cdots, K$ **do**
      更新 Q_k：

$$Q_k(s,a) \leftarrow Q_k(s,a) + \alpha(Y^{\mathrm{EM}} - Q_k(s,a))$$

    end for
  end for
end for

基于之前提出的时序变化集成探索方式和 Expected-Max 集成更新算子，将其结合 Q 学习算法提出 TEAM-Q 学习算法 (Temporally-varying Expected-mAx enseMble Q-learning, TEAM-Q)。

其可以看做是 Q 学习的泛化形式，即当 $K = 1$ 时 TEAM-Q 学习变为 Q 学习。算法3提供了 TEAM-Q 学习的伪代码。并且，基于 TEAM-Q 学习算法和深度神经网络和分布式学习，提出两中新的深度强化学习算法，TEAM DQN 和 DP-TEAM DQN。

5.3.3.2 TEAM 深度 Q 网络

算法 4 TEAM DQN

输入: K 个价值 Q 网络 $\{Q_k(s,a;\theta_k)\}_{k=1}^K$; K 个目标 Q 网络 $\{Q_k^-(s,a;\theta_k^-)\}_{k=1}^K$; 固定分布 $\{w_k\}_{k=1}^K$; 经验回放池 B; 学习率 α; 奖励折扣因子 γ; 探索因子 ϵ;

for 每一幕 **do**
  获得环境的初始状态 s_0
  for 对每一时刻 $t = 1, 2, \cdots$ **do**
    选择一个价值 Q 网络 $Q_{k'}(s,a;\theta_{k'})$，其中 $k' \sim \{w_k\}_{k=1}^K$
    根据基于 $Q_{k'}$ 的 ϵ-贪婪策略采取动作 a_t，获得奖励 r_t 和下一状态 s_{t+1}
    储存样本数据 (s_t, a_t, r_t, s_{t+1}) 到 B 中
    从 B 中采样一个批次的数据
    for 每一个价值 Q 网络 $k = 1, 2, \cdots, K$ **do**
      计算 Expected-Max 目标值

$$Y_k^{\mathrm{EM}} = r + \gamma\Big(\sum_{k' \neq k}^{K} w_{k'} \max_{a'} Q_{k'}^-(s', a'; \theta_{k'}^-) + w_k Q_k^-\big(s', \arg\max_{a'} Q_k(s', a'; \theta_k); \theta_k^-\big)\Big)$$

      通过最小化损失函数来更新 Q_k

$$L_k(\theta_k) = \mathbb{E}_{s,a,r,s'}[(Y_k^{\mathrm{EM}} - Q_k(s,a;\theta_k))^2]$$

    end for
  end for
  每更新 τ 次后更新目标 Q 网络

$$\{Q_k^-(s,a;\theta_k^-)\}_{k=1}^K \leftarrow \{Q_k(s,a;\theta_k)\}_{k=1}^K$$

end for

为解决高维的强化学习任务，我们提出 TEAM DQN。初始化 K 个 Q 网络 $\{Q_i(s,a;\theta_i)\}_{i=1}^K$，并且其中每一个 Q 网络 $Q_i(s,a;\theta_k)$ 是参数为 θ_k 的神经网络。为了稳定神经网络的训练过程，我们也使用了目标网络 $\{Q_i^-(s,a;\theta_i^-)\}$ 和经验回放池 B。并且使用双重估计器[36,37] 来避免过估计偏差。因此，对于 Q 网络 $Q_i(s,a;\theta_i)$，Expected-Max 目标值为:

$$\begin{aligned} y_k^{\mathrm{EM}} =& r + \gamma\Big(\sum_{k' \neq k}^{K} w_{k'} \max_{a'} Q_{k'}^-(s', a'; \theta_{k'}^-) \\ &+ w_k Q_k^-\big(s', \arg\max_{a'} Q_k(s', a'; \theta_k); \theta_k^-\big)\Big) \end{aligned} \tag{5.7}$$

并且其损失函数为

$$L_k(\theta_k) = \mathbb{E}_{s,a,r,s'}[(Y_k^{\mathrm{EM}} - Q_k(s,a;\theta_k))^2] \tag{5.8}$$

其中 (s, a, r, s') 为从经验回放池 B 采样的样本数据。TEAM DQN 的算法伪代码如算法4所示。Bootstrapped DQN[45] 使用了一个二值权值来保证 Q 集成的多样性，但通过实验发现该方法会降低样本有效性且独立随机初始化对于多样性已经足够[47,48]。因此，TEAM DQN 使用同样的数据来更新所有 Q 网络。

5.3.3.3 分布式 TEAM-深度 Q 网络

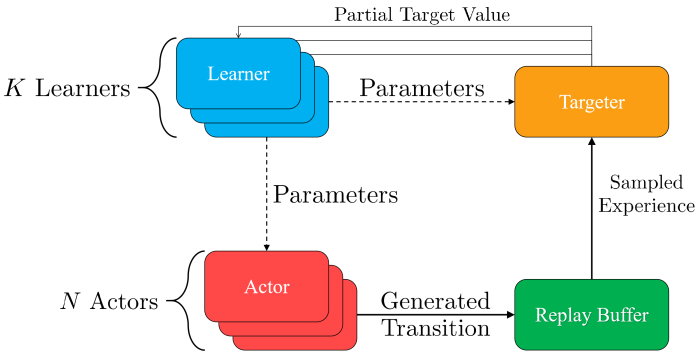


图 5.3 DP-TEAM DQN 的网络结构图

集成强化学习中一个重要的问题是其会带来 K 倍的网络更新计算量，其会使得智能体训练过程变得缓慢。为解决该问题，基于 Ape-X[41] 和 IMPALA[87] 的分布式结构，对 TEAM DQN 进行结构改进。因此提出了新的分布式集成强化学习算法，DP-TEAM DQN。其包含多个学习器 (learner) 和动作器 (actor)，能够有效地提升网络的学习效率。DP-TEAM DQN 的结构流程图如图5.3所示，其包含一个目标值计算器 (Targeter)，K 个学习器 (Learner)，N 个动作器 (Actor)，一个共享经验回放池 B。目标值计算器中包含了 K 个目标值计算网络 $\{Q_i^-(s, a; \theta_i^-)\}_{i=1}^K$，其目的是从共享经验回放池 B 中采样数据并为第 j 个学习器计算部分 Expected-Max 目标值:

$$y_j^{\mathrm{C}} = r + \gamma \sum_{i \neq j}^{K} w_i \max_{a'} Q_i^-(s', a'; \theta_i^-) \tag{5.9}$$

目标值计算网络的网络参数会在每当网络进行了 τ_l 次学习后对学习器中最新参数进行复制。对于 K 个学习器，其中每一个学习器对应着 Q 集成中一个 Q 网络。每第 k 个学习器中设置一个学习网络 $Q_k(s, a; \theta_k)$ 和一个学习目标网络 $Q_k(s, a; \theta_k^-)$。其会从目标价值计算器中获取 y_j^{C}，并根据以下损失函数更新对应的学习网络:

$$L_k(\theta_k) = \mathbb{E}_{s,a,r,s'}[(Y_k^{\mathrm{C}} + \gamma w_k Q_k^-(s', \arg\max_{a'} Q_k(s', a'; \theta_k); \theta_k^-) - Q_k(s, a; \theta_k))^2] \tag{5.10}$$

其中学习目标网络会每进行 τ_l 学习后进行更新。动作器的目标是通过不断地与环境进行交互，为共享经验回放池 B 高效地收集足够的交互数据。每一个动作器中包含 K 个行动网络 $\{Q_k^{\mathrm{act}}(s, a; \theta_k^{\mathrm{act}})\}_{k=1}^K$，

其参数会每当经过 τ_a 次学习后从 K 个学习器中直接复制。目标值计算器，学习器，动作器的算法伪代码分别如算法5，算法6和算法7所示。

算法 5 DP-TEAM DQN 的目标值计算器

输入： K 个目标值计算网络 $\{Q_k^{\mathrm{c}}(s,a;\theta_k^{\mathrm{c}})\}_{k=1}^K$; 固定分布 $\{w_k\}_{k=1}^K$; 经验回放池 B; 奖励折扣因子 γ;

1: **for** 每一个训练步 $t = 1, 2, \cdots$ **do**

2: 从 B 中采样一个批次的数据

3: 为学习器 ($k = 1, \cdots, K$) 计算部分 Expected-Max 目标值

$$Y_k^{\mathrm{c}} = r + \gamma \sum_{k' \neq k}^{K} w_{k'} \max_{a'} Q_{k'}^{\mathrm{c}}(s', a'; \theta_{k'}^{\mathrm{c}})$$

4: 每更新 τ_{c} 步后根据学习器更新目标值计算网络

$$\{Q_k^{\mathrm{c}}(s,a;\theta_k^{\mathrm{c}})\}_{k=1}^K \leftarrow \{Q_k(s,a;\theta_k)\}_{k=1}^K$$

5: **end for**

算法 6 DP-TEAM DQN 中第 k 个学习器

输入： 第 k 个学习网络 $Q_k(s,a;\theta_k)$; 第 k 个学习目标网络 $Q_k^-(s,a;\theta_k^-)$; 第 k 个学习网络的采样概率 w_k; 学习率 α; 奖励折扣因子 γ;

1: **for** 每一个训练步 $t = 1, 2, \cdots$ **do**

2: 从目标值计算器中得到部分 Expected-Max 目标值 Y_k^C

3: calculate Expected-Max target value

$$Y_k^{\mathrm{EM}} = Y_k^C + \gamma w_k Q_k^-(s', \arg\max_{a'} Q_k(s', a'; \theta_k); \theta_k^-)$$

4: 通过最小化损失函数更新 $Q_k(s,a;\theta_k)$

$$L_k(\theta_k) = \mathbb{E}_{s,a,r,s'}[(Y_k^{\mathrm{EM}} - Q_k(s,a;\theta_k))^2]$$

5: 每更新 τ_{l} 步后根据更新学习目标网络

$$\{Q_k^-(s,a;\theta_k^-)\}_{k=1}^K \leftarrow \{Q_k(s,a;\theta_k)\}_{k=1}^K$$

6: **end for**

5.4 理论分析

本节将对提出的时序变化集成探索策略和 Expected-Max 集成更新算子进行理论分析，并有以下结论:

(1) 时序变化集成探索策略有比时序持续集成探索策略更多的状态访问量。

(2) Expected-Max 集成更新算子拥有收缩性。

(3) 对于任意给定 Q 集成，其能够通过迭代使用 Expected-Max 集成更新算子来收敛到一个唯一的不动点，最优 Q 集成。

算法 7 DP-TEAM DQN 的动作器

输入： K 个行动网络 $\{Q_k^{\text{act}}(s,a;\theta_k^{\text{act}})\}_{k=1}^K$; 固定分布 $\{w_k\}_{k=1}^K$; 共享经验回放池 B; 探索因子 ϵ;

1: **for** 每一幕 **do**
2: 获得环境的初始状态 s_0
3: **for** 对每一时刻 $t = 1, 2, \cdots$ **do**
4: 选择一个行动网络 $Q_{k'}^{\text{act}}(s,a;\theta_{k'}^{\text{act}})$，其中 $k' \sim \{w_k\}_{k=1}^K$
5: 根据基于 $Q_{k'}^{\text{act}}$ 的 ϵ-贪婪策略采取动作 a_t，获得奖励 r_t 和下一状态 s_{t+1}
6: 储存样本数据 (s_t, a_t, r_t, s_{t+1}) 到 B 中
7: **end for**
8: 每更新 τ_{a} 步后根据学习器更新目标值计算网络

$$\{Q_k^{\text{act}}(s,a;\theta_k^{\text{act}})\}_{k=1}^K \leftarrow \{Q_k(s,a;\theta_k)\}_{k=1}^K$$

9: **end for**

(4) 通常情况下，Expected-Max 集成更新算子比其他集成更新算子能够更快地收敛。

5.4.1 时序变化集成探索策略分析

$\mathbb{S} \triangleq \left\{s \in \mathcal{S} | \frac{1}{T}\sum_{t=1}^{T} \rho_t(s) > 0\right\}$ 表示智能体在 T 个时间步内所能访问到状态的集合。$\mathbb{S}^{\text{vary}}$ and $\mathbb{S}^{\text{consitent}}$ 分别表示时序变化集成探索策略和时序持续集成探索策略的访问状态集合。$|\mathbb{S}|$ 表示集合中元素的个数。

假设 5.1：(马尔科夫决策过程条件) 马尔科夫决策过程 MDP 是一个有向无环图。

假设条件5.1是强化学习中常用的假设[81,88]，即马尔科夫决策过程在一幕中不能重复地访问同一状态。

以下定理表示了时序变化集成探索策略的状态访问量能够随着策略集成中策略个数 K 指数级上升，而时序持续集成探索策略的状态访问量能够随着策略集成中策略个数 K 线性上升。

定理 5.1：(多样的状态访问量) 为方便分析，假设马尔科夫决策过程和策略函数是确定型的。则有:

$$|\mathbb{S}^{\text{vary}}| \leq K^T, |\mathbb{S}^{\text{consitent}}| \leq KT \tag{5.11}$$

特别地，马尔科夫决策过程满足假设5.1，并且状态转移方程是单射，如 $\mathcal{T}(s,a) \neq \mathcal{T}(s',a')$ 对任意 $s \neq s'$ 或 $a \neq a'$。并且所有策略都各不相同 $\pi_k(s) \neq \pi_{k'}(s)$ 对任 $k \neq k'$, 则有:

$$|\mathbb{S}^{\text{vary}}| = K^T, |\mathbb{S}^{\text{consitent}}| = KT \tag{5.12}$$

证明: 本定理的具体证明过程见A.2。 □

5.4.2 Expected-Max 集成更新算子分析

$\mathbb{Q}$ 为状态动作价值函数空间。$\mathbf{Q} \triangleq \{Q_1, Q_2, \cdots, Q_K\} \in \mathbb{Q}^K$ 表示 Q 集成，其中 $\mathbb{Q}^K$ 为 Q 集成空间。对应地，可以定义 $\mathbf{Q}^* = \{\underbrace{Q^*, Q^*, \cdots, Q^*}_{K}\}$ 为最优 Q 集成，其中 Q^* 为最优 Q 函数。正式地，Expected-Max 集成更新算子的定义如下:

定义 5.2：(Expected-Max 集成更新算子) $\mathcal{T}_k : \mathbb{Q}^K \to \mathbb{Q}^K$ 表示一个在 Q 集成空间中的一个算子，其定义为:

$$\mathcal{T}_k \mathbf{Q}(s,a) = \left\{Q_1, Q_2, \cdots, Q_{k-1}, \hat{Q}_k, Q_{k+1}, \cdots, Q_K\right\} \tag{5.13}$$

其中

$$\hat{Q}_k = \mathbb{E}\left[r + \gamma \sum_{k'=1}^{K} w_{k'} \max_{a'} Q_{k'}(s', a')\right] \tag{5.14}$$

即只有第 k 个状态动作价值函数被更新了。并且定义 $\mathcal{T}_{k:1} \triangleq \mathcal{T}_k \cdots \mathcal{T}_2 \mathcal{T}_1$。Expected-Max 集成更新算子定义为

$$\mathbf{T} \triangleq \mathcal{T}_{K:1} \tag{5.15}$$

证明: 本定理的具体证明过程见A.2。 □

下面定义本章中对Q集成所使用的范数。对任意 $\mathbf{q} = \{q_1, \ldots, q_K\} \in \mathbb{Q}^K, ||\mathbf{q}||_k \triangleq \max_{s,a} |q_k(s,a)|$ and $||\mathbf{q}|| \triangleq \max_k ||\mathbf{q}||_k$.

以下定理证明了 Expected-Max 集成更新算子具有收缩性和其唯一不动点的存在性。

定理 5.2：(收缩性) 对于任意 $\mathbf{q}, \mathbf{q}' \in \mathbb{Q}^K$，则有

$$||\mathbf{T}\mathbf{q} - \mathbf{T}\mathbf{q}'|| \leq \gamma ||\mathbf{q} - \mathbf{q}'|| \tag{5.16}$$

证明: 本定理的具体证明过程见A.2。 □

定理 5.3：(唯一不动点) 任意 $\mathbf{q} \in \mathbb{Q}^K$ 通过迭代地计算 $\mathbf{q}^{i+1} = \mathbf{T}\mathbf{q}^i (i = 0, 1, \cdots)$ 收敛到一个唯一的不动点 $\mathbf{T}\mathbf{Q}^* = \mathbf{Q}^*$ 即有:

$$\lim_{i \to \infty} \mathbf{q}^i \to \mathbf{Q}^* \tag{5.17}$$

证明: 本定理的具体证明过程见A.2。 □

相同于之前的定义，定义集成 Max-Expected 更新算子 $\mathbf{T}^{\mathrm{ME}}$ [42]:

$$\mathcal{T}_k^{\mathrm{ME}} \mathbf{Q}(s,a) = \left\{Q_1, Q_2, \cdots, Q_{k-1}, \hat{Q}_k, Q_{k+1}, \cdots, Q_K\right\} \tag{5.18}$$

其中

$$\hat{Q}_k = \mathbb{E}\left[r + \gamma \max_{a'} \sum_{k'=1}^{K} w_{k'} Q_{k'}(s', a')\right] \tag{5.19}$$

$\mathcal{T}_{k:1}^{\mathrm{ME}} \triangleq \mathcal{T}_k^{\mathrm{ME}} \cdots \mathcal{T}_2^{\mathrm{ME}} \mathcal{T}_1^{\mathrm{ME}}$。Max-Expected 更新算子定义为:

$$\mathbf{T}^{\mathrm{ME}} \triangleq \mathcal{T}_{K:1}^{\mathrm{ME}}$$

假设 5.3：(Q 集成条件) 对于 $\mathbf{q} \in \mathbb{Q}^K$，其满足 $\mathbf{q} \leq \mathbb{Q}^*$

假设条件5.3表示 Q 集成小于最优价值函数集成 $\mathbb{Q}^*$。在实际中，价值函数通常初始化为 0，最优价值函数通常为正。若不满足此情况，我们也可以初始化一个 Q 集成满足此条件。因此，通过定理5.4表明，通常 Expected-Max 集成更新算子 $\mathbf{T}$ 比 Max-Expected 更新算子 $\mathbf{T}^{\mathrm{ME}}$ 能够收敛更快.

定理 5.4：(收敛速度) 对 $\mathbf{q} \in \mathbb{Q}^K$，若 $\mathbf{q}$ 满足条件5.3，则有:

$$||\mathbf{Tq} - \mathbf{Q}^*|| \leq ||\mathbf{T}^{\mathrm{ME}}\mathbf{q} - \mathbf{Q}^*|| \tag{5.20}$$

证明: 本定理的具体证明过程见A.2。 □

5.5 仿真实验

在本节中，通过在不同规模的强化学习任务中的实验分别验证 TEAM-Q 学习，TEAM DQN 和 DP-TEAM DQN 算法的有效性。因此设计一系列实验来验证以下问题:

(1) 提出算法相比于现有流行的基准算法的性能比较如何?

(2) Expected-Max 集成更新算子能否提升算法的学习有效性?

(3) 时序变化集成探索策略能够提升探索能力?

(4) 集成大小 K 如何影响性能?

5.5.1 DeepSea 上 TEAM Q 学习性能实验

我们先在一个小规模的稀疏奖励环境进行实验，DeepSea[46,83]。如图5.4(a) 所示，其是一个 22×22 的格子世界下三角。初始状态是左上角的红点，并且在每个状态，智能体有两个可选的动作: 下和右下。并且智能体在每一步会收到一个小的负奖励 (−0.0005)。唯一的一个正奖励 (+1) 能够在目标状态获得，其在右下角的黄点。并且其要采取一个单独的序列动作，即一直采取右下动作。实验结果以平均回报和训练幕数构造训练曲线作为对比。

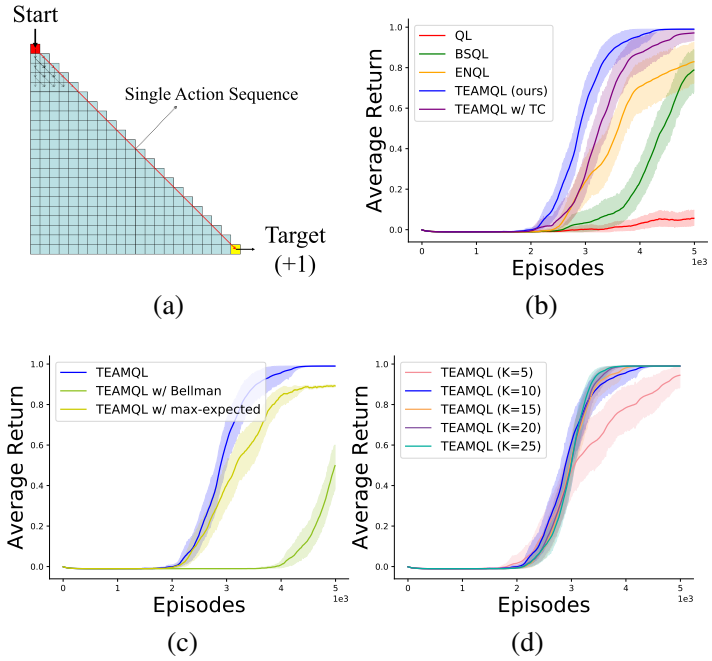


图 5.4　(a)DeepSea 环境示例，(b)TEAM Q 学习与基准算法的性能对比，(c)TEAM Q 学习不同的更新算子的性能对比和 (d)TEAM Q 学习使用不同集成大小的性能对比

5.5.1.1　实验参数设置

对于 TEAM-Q 学习算法，采样分布设置为 $\{w_k\}_{k=1}^{K} = \mathcal{U}\{1, \cdots, K\}$。对于所有的算法，折扣因子设置为 $\gamma = 1$，学习率设置为 $\alpha = 0.1$，Q 表格基于均匀分布 $\mathcal{U}(-0.01, 0.01)$ 进行初始化，Q 学习算法使用探索因子 $\epsilon = 0.01$ 的 ϵ-贪婪策略。所有算法的性能都是 50 个随机种子的平均结果。总幕数设置为 $5,000$。

5.5.1.2　主要实验结果

图5.4(b) 中展示了 Q 学习 (QL)[10]，Bootstrapped Q 学习 (BSQL)[45]，Ensemble Q 学习 (ENQL)[42] 和 TEAM-Q 学习 (TEAMQL) 的对比结果。通过分析可明显发现 TEAM-Q 学习相比其他基准算法拥有更为先进的性能。在 DeepSea 中，最优无折扣回报为 $R^{opt} = 0.99$。遗憾定

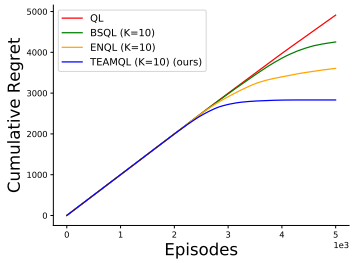


图 5.5　TEAM-Q 学习与基准算法在 DeepSea 上的累计遗憾对比

义为

$$\Delta(R^m) = R^{opt} - R^m \tag{5.21}$$

其中 R^m 是智能体在第 m 幕的无折扣回报。并且累计遗憾定义为:

$$\Delta^c = \sum_{m=1}^{M} \Delta(R^m) \tag{5.22}$$

其中 M 为智能体和环境交互的总幕数。图5.5展现了 Q 学习，Bootstrapped Q 学习，Ensemble Q 学习和 TEAM-Q 学习在 DeepSea 上的累计遗憾。并且可知 TEAM-Q 学习能够更有效地找到最优策略。

5.5.1.3 TEAM Q 学习集成大小参数敏感度实验

图中展现了 TEAM-Q 学习算法使用时序持续集成探索策略 (TEAMQL w/ TC)。并且图5.4(c) 展现了使用时序变化集成探索策略时，不同更新算子 (标准贝尔曼更新算子和 Max-Expected 更新算子) 的性能对比。基于之前的结果，对第二个问题的回答如下。无论是使用时序变化集成探索策略或时序持续集成探索策略，Expected-Max 集成更新算子相对于 Max-Expected 更新算子能更有效地提升学习有效性。通过对比图5.4(b) 和5.4(c) 可以知道两种探索策略对不同更新算法的影响。因此问题 (3) 的回答如下。对于集成更新算子，使用时序变化集成探索策略能够提升学习有效性。但是，对于标准贝尔曼更新算子，使用时序变化集成探索策略反而会使性能下降。因此可知，时序变化集成探索策略需要所有的状态动作价值函数互相合作，而标准贝尔曼更新算子没有让 Q 函数中的信息进行传递。这也意味着时序变化集成探索策略不适合该算子。对于问题 (4)，图5.4(d) 展示了 TEAM-Q 学习算法的性能随着集成大小增大而提升。

5.5.2 MinAtar 上 TEAM DQN 性能实验

我们在 MinAtar 上对比当前流行的集成基准算法来验证 TEAM DQN 的有效性。

5.5.2.1 实验参数设置

算法基于 MinAtar[80] 环境提供的官方代码进行实现。并且对所有的方法使用相同的神经网络结构和超参数设置。其中详细的参数设计总结为表5.1。对于 Bootstrapped DQN，UCB Exploration，Sunrise DQN 和 TEAM DQN，探索因子设置为 $\epsilon = 0$，除了 DQN 和 Ensemble DQN。对于 TEAM DQN，设置固定分布为 $\{w_k\}_{k=1}^K = \mathcal{U}\{1, \cdots, K\}$。根据原文献来设置基准算法的参数。对于 Bootstrapped DQN，更新权值为 $m \sim$ Bernoulli (1)。对于 UCB Exploration，设置参数 $\lambda = 0.1$. 对于 Sunrise DQN，设置参数 $\lambda = 0.1$，并且相同的更新权重设置方法。

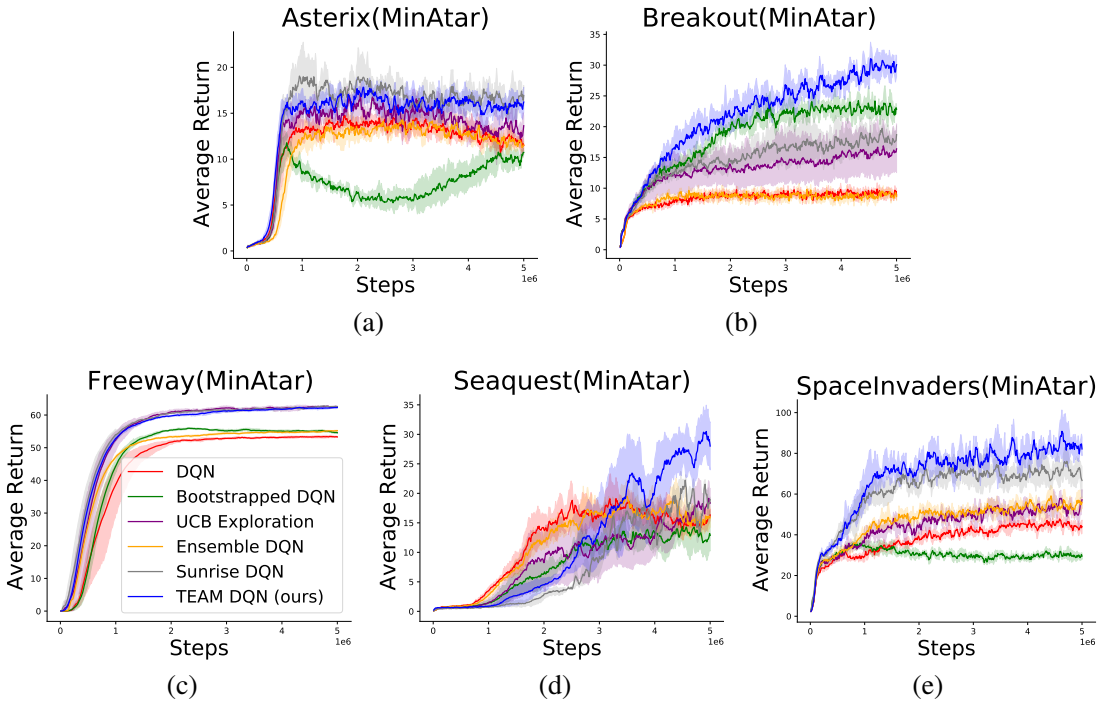


图 5.6　TEAM DQN 和基准算法在 MinAtar 五个环境上的性能对比

5.5.2.2　性能对比

图5.6(a)-(e) 分别展现了 TEAM DQN，DQN[18]，Bootstrapped DQN[45]，UCB Exploration[47]，Ensemble DQN[42]，SUNRISE DQN[48] 在 MinAtar 中五个环境上的性能。结果表明了 TEAM DQN 比其他的基准算法取得更先进的性能，特别是 Breakout，Seaquest 和 Space Invaders。并且值得注意的是，TEAM DQN 在两个需要更多探索的环境上取得先进的性能，即 Freeway 和 Seaquest 中奖励具有稀疏性。

5.5.2.3　消融实验

图5.7(a)-(e) 展现了 TEAM DQN 具有不同集成大小 ($K = 1, 3, 5$) 的性能。特别地，当 $K = 1$ 时，TEAM DQN 变为 Double DQN[37]。TEAM DQN 在所有环境上的性能都比 Double DQN 更好。特别是在 Breakout，Freeway 和 Space Invaders 上，性能随着集成大小的上升而提升。在 Seaquest 中，TEAM DQN($K = 5$) 的学习速度比 TEAM DQN($K = 3$) 要慢一些，但其有相似的最终性能。

5.5.3　Atari 上 DP-TEAM DQN 性能实验

我们在 Atari[33] 游戏环境上评价 DP-TEAM DQN，并且对比三个基准算法:SimPLe[27]，Rainbow[20] 和 Ape-X DQN[41]。其中 SimPLe 为一个高效的基于模型强化学习算法，智能体能够使用动态预测模型采样的数据进行训练。Rainbow 是一个结合了多个深度 Q 网络技巧的高质量基准

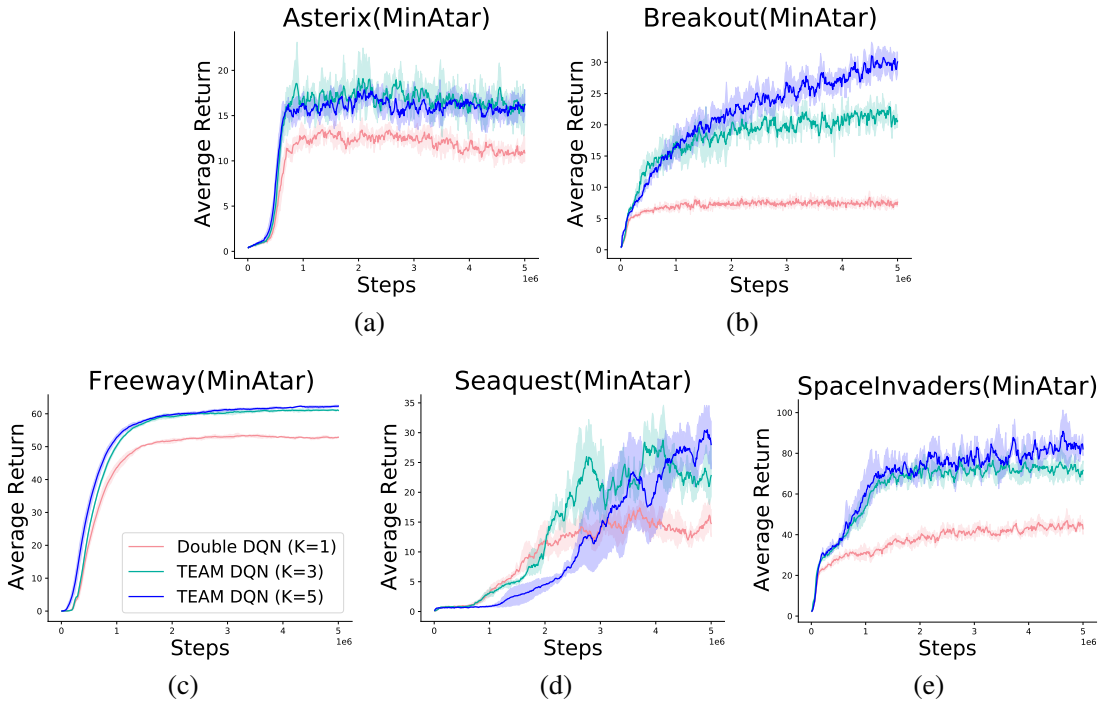


图 5.7　TEAM DQN 使用不同集成大小在 MinAtar 五个环境上的性能对比

算法。Ape-X DQN 是一个代表性的高效分布式强化学习算法。为了验证方法的学习有效性。本章使用先前工作的相似实验设置[27]。每种方法 36 个 Atari 游戏的 5 个随机种子上训练 1M 步。对于 DP-TEAM DQN，集成大小设置为 $K = 5$。

5.5.3.1　实验参数设置

在 Atari 环境中，对于每个算法每经过 10,000 训练步后，每个算法进行 30 幕的评估。对于 Rainbow 和 SimPLe，使用原先文献[27] 中报告的结果。本文采用和文献[34] 一致的网络结构。对于 DP-TEAM DQN，采样分布设置为 $\{w_k\}_{k=1}^{K} = \mathcal{U}\{1, \cdots, K\}$。Ape-X DQN 使用和 DP-TEAM DQN 一样的超参数设置如表5.2所示。

5.5.3.2　性能对比

本文使用三个 Atari 评估指标: 相对得分 (relative score)[38]，人类标准得分 (human normalized score)[37]，人类差距 (human gap)[89]。这三个评估指标分别定义为:

1. 相对得分，用于评估智能体对比特定基准算法:

$$Score_R = \frac{Score_a - Score_b}{\max\{Score_b, Score_h\} - Score_r} \times 100\% \tag{5.23}$$

表 5.1　在 MinAtar 环境上 TEAM DQN 的参数设置

超参数名	设置值
集成大小 K	5
初始探索因子	1
最终探索因子	0.1
批次大小	32
优化器	RMSProp
损失函数	Huber loss
学习率	0.00025
训练步数	5,000,000
经验回放池大小	100,000
奖励折扣因子 γ	0.99
学习开始步数	5,000
目标网络更新步数	1,000
Grad momentum	0.95
Squared grad momentum	0.95
Min squared grad	0.01
Q network channels	16
Q network stride	1
Q network hidden units	128

2. 人类标准得分，用于评估智能体对比人类水平:

$$Score_{HN} = \frac{Score_a - Score_r}{Score_h - Score_r} \times 100\% \tag{5.24}$$

3. 人类差距，用于评估智能体与人类水平的差距:

$$H_{gap} = 1 - \mathbb{E}_{game}[\min(1, Score_{HN})] \tag{5.25}$$

其中 $Score_a, Score_b, Score_h$ 和 $Score_r$ 分别表示智能体，基准算法，人类和随机玩家的得分水平。

图5.8具体展示了 DP-TEAM DQN 对其他算法在每个环境上的相对得分。表5.3总结了 DP-TEAM DQN 对其他算法的相对得分，并通过结果可知 DP-TEAM DQN 能够取得更为先进的平均水平。表5.4总结了 DP-TEAM DQN 的人类标准得分和人类差距，并通过结果可知 DP-TEAM DQN 比其他算法取得更为先进的性能。在 Atalantis 上，Rainbow 比其他算法取得非常大的差距，但其中间性能为 61.75% 而平均性能和均方差分别为 253.82% 和 8.54。但 DP-TEAM DQN 取得比其他的算法更高的人类标准得分和更小的人类差距。各算法的在 36 个 Atari 环境上的具体性能得分情况如表5.5所示，其中 DP-TEAM DQN 在 28 个环境上取得了最好的性能。

表 5.2　在 Atari 环境上 DP-TEAM DQN 的参数设置

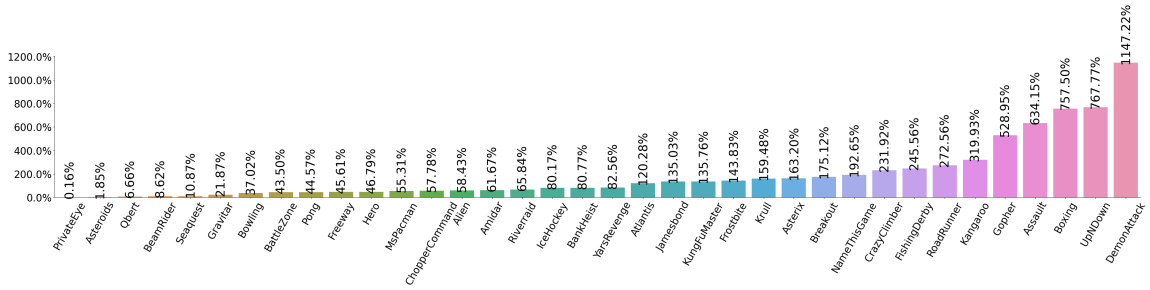
超参数名	设置值
学习器个数 (集成大小) K	5
动作器个数 N	6
第 i 个行动器的探索因子	$0.4^{(1+7\frac{i}{N-1})}$
批次大小	128
优化器	Adam
损失函数	MSE loss
学习率	0.0001
训练步数	1,000,000
经验回放池大小	50,000
奖励折扣因子 γ	0.99
学习开始步数	5,000
目标值计算器更新步数 τ_c	250
学习器学习目标网络更新步数 τ_l	250
动作器动作网络更新步数 τ_a	100
Q network channels	32, 64, 64
Q network stride	4, 2, 1
Q network hidden units	512
Grey-scaling	TRUE
Observation down-sampling	(84, 84)
Frames stacked	4

表 5.3　DP-TEAM DQN 对基准算法和人类水平的相对得分

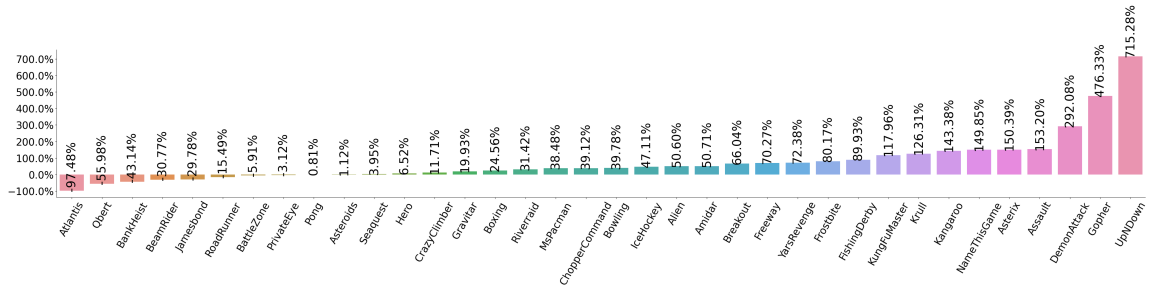
Relative Score (1M training steps over 5 seeds)		
Baseline	Mean(±Std)	Median
Human	128.53%(±2.79)	16.07%
SimPLe	192.80%(±2.59)	81.67%
Rainbow	77.44%(±1.50)	39.45%
Ape-X DQN	43.53%(±0.69)	21.58%

5.6　本章小结

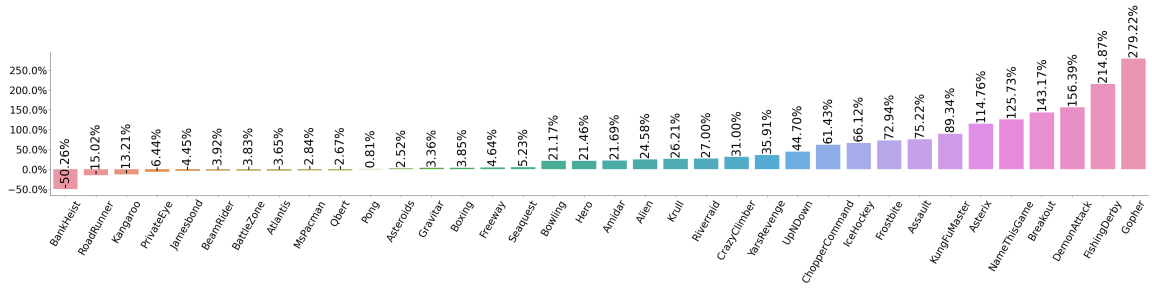
本章提出了 TEAM-Q 框架, 其包含了时序变化性集成探索策略和 Expected-Max 集成更新算子。为了解决具体的控制问题, 提出了三种具体的算法 TEAM-Q 学习, TEAM DQN 和 DP-TEAM DQN。通过理论分析表明, 时序变化性集成探索策略具有更强的探索能力, Expected-Max 集成更新算子能够有效地提升智能体的学习效率。在不同的强化学习任务上的实验结果表明 TEAM-Q 集成强化学习框架能够有效地提升智能体的探索和学习性能，特别是在稀疏奖励的环境中提升更为明显。



(a)



(b)



(c)

图 5.8　DP-TEAM DQN 相对于基准算法在 36 个 Atari 环境上的相对得分

表 5.4　DP-TEAM DQN 和基准算法的人类标准得分和人类差距得分

Human-Normalized Score (1M training steps over 5 seeds)			
Algorithm	Mean(±Std)	Median	Human Gap
SimPLe	26.93%(±0.52)	6.74%	0.78
Rainbow	**253.82%(±8.54)**	61.57%	0.42
Ape-X DQN	152.20%(±1.95)	63.34%	0.38
DP-TEAM DQN	228.53%(±2.79)	**116.07%**	**0.26**

表 5.5　DP-TEAM DQN 和基准算法在 36 个 Atari 游戏环境上的平均得分结果

Game	Human	Random	SimPLe	Rainbow	Ape-X DQN	DP-TEAM DQN
Alien	7127.8	227.8	405.2	945.0	2741.5	**4436.7**
Amidar	1719.5	5.8	88.0	275.8	773.1	**1144.8**
Assault	742.0	222.4	369.3	1581.8	2186.5	**3664.5**
Asterix	8503.0	210.0	1089.5	2151.6	5106.7	**14623.4**
Asteroids	47388.7	719.1	731.0	1071.5	420.0	**1596.0**
Atlantis	29028.1	12850.0	14481.6	**848800.0**	34740.0	40606.4
BankHeist	753.1	14.2	8.2	1053.3	**1202.0**	829.3
BattleZone	37187.5	2360.0	5184.4	22391.4	21666.8	**24200.0**
BeamRider	16926.5	363.9	422.7	**6945.3**	2499.5	6572.4
Bowling	160.7	23.1	34.4	30.6	56.6	**85.3**
Boxing	12.1	0.1	9.1	80.3	96.3	**100.0**
Breakout	30.5	1.7	12.7	38.7	21.9	**63.1**
ChopperCommand	7387.8	811.0	1246.9	2474.0	1006.6	**5046.7**
CrazyClimber	35829.4	10780.5	39827.8	97088.1	84380.0	**107193.3**
DemonAttack	1971.0	152.1	169.5	5478.6	8297.6	**21036.2**
FishingDerby	-38.8	-91.7	-91.5	-23.2	-75.2	**38.4**
Freeway	29.6	0.0	20.3	13.0	32.3	**33.8**
Frostbite	4334.7	65.2	254.7	2972.3	3281.3	**6395.3**
Gopher	2412.5	257.6	771.0	1905.0	3398.5	**12169.4**
Gravitar	3351.4	173.0	198.3	260.0	786.6	**893.3**
Hero	30826.4	1027.0	1295.1	13295.5	8842.6	**15237.0**
IceHockey	0.9	-11.2	-10.5	-6.5	-8.8	**-0.8**
Jamesbond	302.8	29.0	125.3	**692.6**	516.6	550.0
Kangaroo	3035.0	52.0	323.1	4084.6	**11360.0**	9866.6
Krull	2666.0	1598.0	4539.9	4971.1	7646.5	**9231.6**
KungFuMaster	22736.3	258.5	17257.2	21258.6	25353.2	**47773.6**
MsPacman	6951.6	307.3	762.8	1881.4	**4626.7**	4438.0
NameThisGame	8049.0	2292.3	1990.4	4454.2	5842.7	**13080.7**
Pong	14.6	-20.7	5.2	20.6	20.6	**20.9**
PrivateEye	69571.3	24.9	58.3	2336.7	**4650.5**	177.5
Qbert	13455.0	163.9	559.8	**8885.2**	1800.0	4655.0
Riverraid	17118.0	1338.5	1587.0	7018.9	7716.7	**11976.7**
RoadRunner	7845.0	11.5	5169.4	31379.7	31206.8	**36180.0**
Seaquest	42054.7	68.4	370.9	3279.9	2741.5	**5602.7**
UpNDown	11693.2	533.4	2152.6	8010.9	60866.4	**105202.7**
YarsRevenge	54576.9	3092.9	2980.2	8225.1	26997.4	**50901.6**

第六章　总结与展望

6.1　工作总结

强化学习作为一种重要的机器学习范式，其已经在智能决策，游戏人工智能等多个领域得到了成功广泛的应用。并且随着深度学习在近十年来的迅速发展，深度强化学习现在已经成为机器学习领域中最热门的研究领域之一。在强化学习中,Rollout 算法是经常被使用的一类基于蒙特卡洛的决策时规划算法。当智能体获得环境模型时, 其一般使用 Rollout 算法中基于模型模拟的蒙特卡洛方法, 如蒙特卡洛树搜索; 当智能体在面对未知环境是, 其可以使用 Rollout 算法中的基于时序差分的状态价值估计方法, 如 Q 学习。但这两种方式仍然存在着许多挑战, 本文的主要研究问题为 (1) 在状态空间巨大的强化学习任务中, 蒙特卡洛树搜索算法探索效率低下问题;(2) 在具有稀疏奖励的强化学习任务中, 智能体探索策略效率低下问题;(3) 智能体在稀疏奖励环境中价值函数学习效率低效问题。针对这三个研究问题, 分别在第三, 四和五章提出了各自的解决算法。在第三章中, 本文提出了 GLVS 算法, 并结合现实生活中 NP 难的调度问题进行算法有效性检验, 通过理论分析以及在 BDSP 上的数值模拟实验可表明所提出的 GLVS 算法有着比基准算法更好的性能。在第四章中结合 UCB 算法提出了 UCB-Q 探索策略和 Alter DQN 算法, 并通过在强化学习实验环境上的实验结果, 表明 Alter DQN 算法能够比其他算法拥有更好的探索性能和样本有效性。在第五章中, 本文提出了一个高效探索和学习的集成强化学习框架, 其由时序变化集成探索策略和 Expected-Max 集成更新算子所组成。通过理论分析可知时序变化集成探索策略能够有效地利用不同策略函数的差异性进行多样化探索。并且 Expected-Max 集成更新算子能够保证收敛性质以及相比其他集成算子更快的收敛速度，该算子能够充分传递价值函数中的价值信息以提升智能体学习效率。实验结果表明，TEAM-Q 集成强化学习框架能够在多个强化学习任务环境上取得比基准流行算法更优的性能。

6.2　未来展望

强化学习，尤其是结合了深度学习的深度强化学习是当前机器学习领域中最收欢迎的研究方向之一。本文对 Rollout 算法及其当前所存在的挑战问题进行了研究，特别是基于模型模拟的蒙特卡洛方法和基于时序差分的状态价值估计方法。因此，基于当前已有的研究内容，未来的研究方向主要包含以下几个方面:

1. 将提出 GLVS 算法，应用到更为复杂且实际的动态调度规划问题中。

2. 将所提出的 UCB-Q 探索策略结合其他先进的价值函数优化机制，如 Rainbow 中所使用的改进方式。

3. 将 TEAM-Q 集成强化学习框架结合到基于策略的强化学习算法和“行动器-评价器”强化学习算法中以处理具有连续动作空间的强化学习任务，如 DDPG[30]，PPO[21] 和 SAC[31] 等。

参考文献

[1] Muhammad I, Yan Z. SUPERVISED MACHINE LEARNING APPROACHES: A SURVEY[C]. Proceedings of Soft Computing, 2015.

[2] Kramer O. K-Nearest Neighbors[C]. . 2013.

[3] Song Y Y, Lu Y. Decision tree methods: applications for classification and prediction.[J]. Shanghai archives of psychiatry, 2015, 27:130–135.

[4] Zheng Z. Naive Bayesian Classifier Committees[C]. Proceedings of European conference on Machine Learning, 1998.

[5] Maalouf M. Logistic regression in data analysis: an overview[J]. International Journal of Data Analysis Techniques and Strategies, 2011, 3:281–299.

[6] Gentleman R, Carey V J. Unsupervised Machine Learning[C]. . 2008.

[7] Wong J A H A. Algorithm AS 136: A K-Means Clustering Algorithm[J]. Journal of the Royal Statistical Society, 1979, 28(1):100–108.

[8] Hardoon D R, Szedmak S, Shawe-Taylor J. Canonical Correlation Analysis: An Overview with Application to Learning Methods[J]. Neural Computation, 2004, 16(12):2639–2664.

[9] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.

[10] Watkins C J, Dayan P. Q-learning[J]. Machine learning, 1992, 8(3-4):279–292.

[11] LeCun Y, Bengio Y, Hinton G E. Deep learning[J]. Nature, 2015, 521:436–444.

[12] Kingma D P, Welling M. An Introduction to Variational Autoencoders.[J]. arXiv: Learning, 2019..

[13] Goodfellow I, Pouget-Abadie J, Mirza M, et al. Generative adversarial networks[J]. Communications of The ACM, 2020, 63:139–144.

[14] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587):484–489.

[15] Xin L, Song W, Cao Z, et al. Step-wise Deep Learning Models for Solving Routing Problems[J]. IEEE Transactions on Industrial Informatics, 2020. 1–1.

[16] Liang H, Zhang X, Hong X, et al. Reinforcement Learning Enabled Dynamic Resource Allocation in Internet of Vehicles[J]. IEEE Transactions on Industrial Informatics, 2020. 1–1.

[17] Wang S, Bi S, Zhang Y A. Reinforcement Learning for Real-Time Pricing and Scheduling Control in EV Charging Stations[J]. IEEE Transactions on Industrial Informatics, 2021, 17(2):849–859.

[18] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013..

[19] Rowland M, Bellemare M G, Dabney W, et al. An Analysis of Categorical Distributional Reinforcement Learning[J]. arXiv: Machine Learning, 2018..

[20] Hessel M, Modayil J, Van Hasselt H, et al. Rainbow: Combining improvements in deep reinforcement learning[C]. Proceedings of Thirty-second AAAI conference on artificial intelligence, 2018.

[21] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017..

[22] Schulman J, Levine S, Abbeel P, et al. Trust Region Policy Optimization[C]. Proceedings of International Conference on Machine Learning, 2015.

[23] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning[C]. Proceedings of International Conference on Machine Learning, 2016.

[24] Browne C, Powley E J, Whitehouse D, et al. A Survey of Monte Carlo Tree Search Methods[J]. IEEE Transactions on Computational Intelligence and AI in Games, 2012, 4:1–43.

[25] Silver D, Schrittwieser J, Simonyan K, et al. Mastering the game of Go without human knowledge[J]. Nature, 2017, 550:354–359.

[26] Sutton R. Integrated architecture for learning, planning, and reacting based on approximating dynamic programming[C]. Proceedings of International Conference on Machine Learning, 1990.

[27] Kaiser L, Babaeizadeh M, Milos P, et al. Model-based reinforcement learning for atari[J]. arXiv preprint arXiv:1903.00374, 2019..

[28] Racanière S, Weber T, Reichert D P, et al. Imagination-Augmented Agents for Deep Reinforcement Learning[C]. Proceedings of Neural Information Processing Systems, 2017.

[29] Ha D, Schmidhuber J. World Models[J]. arXiv: Learning, 2018..

[30] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015..

[31] Haarnoja T, Zhou A, Abbeel P, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor[C]. Proceedings of International conference on machine learning. PMLR, 2018. 1861–1870.

[32] Fujimoto S, Hoof H, Meger D. Addressing Function Approximation Error in Actor-Critic Methods[J]. arXiv: Artificial Intelligence, 2018..

[33] Bellemare M G, Naddaf Y, Veness J, et al. The arcade learning environment: An evaluation platform for general agents[J]. Journal of Artificial Intelligence Research, 2013, 47:253–279.

[34] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540):529–533.

[35] Thrun S, Schwartz A. Issues in using function approximation for reinforcement learning[C]. Proceedings of Proceedings of the Fourth Connectionist Models Summer School. Hillsdale, NJ, 1993. 255–263.

[36] Hasselt H. Double Q-learning[J]. Advances in neural information processing systems, 2010, 23:2613–2621.

[37] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning[C]. Proceedings of Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.

[38] Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning[C]. Proceedings of International conference on machine learning. PMLR, 2016. 1995–2003.

[39] Fortunato M, Azar M G, Piot B, et al. Noisy networks for exploration[J]. arXiv preprint arXiv:1706.10295, 2017..

[40] Schaul T, Quan J, Antonoglou I, et al. Prioritized Experience Replay[J]. arXiv: Learning, 2015..

[41] Horgan D, Quan J, Budden D, et al. Distributed prioritized experience replay[J]. arXiv preprint arXiv:1803.00933, 2018..

[42] Anschel O, Baram N, Shimkin N. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning[C]. Proceedings of International conference on machine learning. PMLR,

2017. 176–185.

[43] Lan Q, Pan Y, Fyshe A, et al. Maxmin q-learning: Controlling the estimation bias of q-learning[J]. arXiv preprint arXiv:2002.06487, 2020..

[44] Peer O, Tessler C, Merlis N, et al. Ensemble Bootstrapping for Q-Learning[J]. arXiv preprint arXiv:2103.00445, 2021..

[45] Osband I, Blundell C, Pritzel A, et al. Deep exploration via bootstrapped DQN[J]. Advances in neural information processing systems, 2016, 29:4026–4034.

[46] Osband I, Aslanides J, Cassirer A. Randomized prior functions for deep reinforcement learning[J]. arXiv preprint arXiv:1806.03335, 2018..

[47] Chen R Y, Sidor S, Abbeel P, et al. UCB exploration via Q-ensembles[J]. arXiv preprint arXiv:1706.01502, 2017..

[48] Lee K, Laskin M, Srinivas A, et al. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning[C]. Proceedings of International Conference on Machine Learning. PMLR, 2021. 6131–6141.

[49] Wren A, Rousseau J M. Bus Driver Scheduling - An Overview[J]. 1994..

[50] Loth M, Sebag M, Hamadi Y, et al. Hybridizing constraint programming and monte-carlo tree search: Application to the job shop problem[C]. Proceedings of International Conference on Learning and Intelligent Optimization. Springer, 2013. 315–320.

[51] Martinez-Alfaro H, Minero J, Alanis G E, et al. Using simulated annealing to solve the classroom assignment problem[C]. Proceedings of Isai/ifis Mexico-usa Collaboration in Intelligent Systems Technologies, 1996.

[52] Fischetti M, Martello S, Toth P. The Fixed Job Schedule Problem with Spread-Time Constraints[J]. Operations Research, 1989, 37(3):395–403.

[53] Smith B M, Wren A. A bus crew scheduling system using a set covering formulation[J]. Transportation Research Part A: General, 1988, 22(2):97–108.

[54] Yaghini M, Karimi M, Rahbar M. A set covering approach for multi-depot train driver scheduling[J]. Journal of Combinatorial Optimization, 2015, 29(3):636–654.

[55] Beasley J E, Chu P C. A genetic algorithm for the set covering problem[J]. European Journal of Operational Research, 1996, 94(2):392–404.

[56] Caprara A, Fischetti M. A Heuristic Method for the Set Covering Problem[J]. Operations Research, 1999, 47(5).

[57] Shijun, Chen, Yindong, et al. An Improved Column Generation Algorithm for Crew Scheduling Problems[J]. Journal of Information and Computational Science, 2013..

[58] Paias A, Paix O J. State space relaxation for set covering problems related to bus driver scheduling[J]. European Journal of Operational Research, 1993, 71(2):303–316.

[59] Ma J, Tao L, Song C. A Lagrangian relaxation-based heuristic for the bus driver scheduling problem: a case study of Beijing[C]. Proceedings of Iet International Conference on Information Science and Control Engineering, 2014.

[60] Carraresi P, Gallo G, Rousseau J M. Relaxation approaches to large scale bus driver scheduling problems[J]. Transportation Research Part B, 1982, 16(5):383–397.

[61] Zhao L. A heuristic method for analyzing driver scheduling problem[J]. IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans, 2006, 36(3):521–531.

[62] Constantino A A, Calvi R, Mendona N C F X, et al. Solving a Large Real-world Bus Driver Scheduling Problem with a Multi-assignment based Heuristic[J]. Journal of Universal Computer Science, 2017, 23(5):479–504.

[63] Leone R D, Festa P, Marchitto E. A Bus Driver Scheduling Problem: a new mathematical model and a GRASP approximate solution[J]. Journal of Heurs, 2011, 17(4):441–466.

[64] Kuhn H W. The Hungarian method for the assignment problem[J]. Naval Research Logs, 2010, 52(1-2):7–21.

[65] Baier H, Winands M H M. Monte-Carlo Tree Search and Minimax Hybrids with Heuristic Evaluation Functions[C]. Proceedings of Workshop on Computer Games, 2014.

[66] Lu L, Zhang W, Gu X, et al. HMCTS-OP: Hierarchical MCTS Based Online Planning in the Asymmetric Adversarial Environment[J]. Symmetry, 2020, 12(5):719.

[67] Parascandolo G, Buesing L, Merel J, et al. Divide-and-Conquer Monte Carlo Tree Search For Goal-Directed Planning[J]. 2020..

[68] Kurzer K, Hrtnagl C, Zllner J M. Parallelization of Monte Carlo Tree Search in Continuous Domains[J]. 2020..

[69] Bai A, Wu F, Chen X. Online Planning for Large Markov Decision Processes with Hierarchical Decomposition[J]. ACM Trans. Intell. Syst. Technol., 2015, 6(4).

[70] Ottens B, Dimitrakakis C, Faltings B. DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems[J]. ACM Trans. Intell. Syst. Technol., 2017, 8(5).

[71] Chaslot G, Jong S D. Monte-Carlo Tree Search in Production Management Problems[C]. . 2008.

[72] Runarsson T P, Schoenauer M, Sebag M. Pilot, rollout and monte carlo tree search methods for job shop scheduling[C]. Proceedings of International Conference on Learning and Intelligent Optimization. Springer, 2012. 160–174.

[73] Furuoka R, Matsumoto S. Worker' s knowledge evaluation with single-player Monte Carlo tree search for a practical reentrant scheduling problem[J]. Artificial Life and Robotics, 2017, 22(1):130–138.

[74] Lubosch M, Kunath M, Winkler H. Industrial scheduling with Monte Carlo tree search and machine learning[J]. Procedia CIRP, 2018, 72:1283–1287.

[75] Huang H, Weng J, Liu L, et al. A Method for Bus Operation Cost Calculation Based on Multi-source Data[J]. Journal of Transport Information and Safety, 2013..

[76] Auer P. Finite-time analysis of the multiarmed bandit problem[J]. Machine Learning, 2002, 47.

[77] Miller F P, Vandome A F, Mcbrewster J. Game complexity[M]. Alphascript Publishing, 2010.

[78] Fei Z, Wen W, Quan L, et al. A Deep Q-Network Method Based on Upper Confidence Bound Experience Sampling[J]. Journal of Computer Research and Development, 2018, 55:1694.

[79] Brockman G, Cheung V, Pettersson L, et al. OpenAI Gym[J]. arXiv: Learning, 2016..

[80] Young K, Tian T. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments[J]. arXiv preprint arXiv:1903.03176, 2019..

[81] O' Donoghue B, Osband I, Munos R, et al. The uncertainty bellman equation and exploration[C]. Proceedings of International Conference on Machine Learning, 2018. 3836–3845.

[82] Janz D, Hron J, Mazur P, et al. Successor uncertainties: exploration and uncertainty in temporal difference learning[J]. Advances in Neural Information Processing Systems, 2019, 32:4507–4516.

[83] Dabney W, Ostrovski G, Barreto A. Temporally-Extended epsilon-greedy exploration[J]. arXiv

preprint arXiv:2006.01782, 2020..

[84] Kumar A, Gupta A, Levine S. Discor: Corrective feedback in reinforcement learning via distribution correction[J]. arXiv preprint arXiv:2003.07305, 2020..

[85] Chen X, Wang C, Zhou Z, et al. Randomized ensembled double q-learning: Learning fast without a model[J]. arXiv preprint arXiv:2101.05982, 2021..

[86] Pathak D, Gandhi D, Gupta A. Self-supervised exploration via disagreement[C]. Proceedings of International conference on machine learning. PMLR, 2019. 5062–5071.

[87] Espeholt L, Soyer H, Munos R, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures[C]. Proceedings of International Conference on Machine Learning. PMLR, 2018. 1407–1416.

[88] Osband I, Van Roy B, Wen Z. Generalization and exploration via randomized value functions[C]. Proceedings of International Conference on Machine Learning. PMLR, 2016. 2377–2386.

[89] Dabney W, Ostrovski G, Silver D, et al. Implicit quantile networks for distributional reinforcement learning[C]. Proceedings of International conference on machine learning. PMLR, 2018. 1096–1105.

致　　谢

转眼间，自从 2015 年进入南航理学院学习信息与计算科学，到进入计算机科学与技术学院学习机器学习，我已经不知不觉地在南航度过了快七年的时间。我也从当初什么也不懂的高中生变为现在的硕士研究生。在起初进入南航时，我不适应数学系的课程安排，甚至想要申请转专业。但逐渐地，我慢慢领悟到了数学所带给我的魅力。感谢理学院陈晓红老师送我的《数学之美》以及在完成本科毕业设计时对我的悉心指导，让我进入了机器学习的领域。并感谢陈松灿老师在与我邮件交流时对我的鼓励，以及为我写推荐信。更要感谢的是我导师谭晓阳老师，在我研究生阶段他严谨的科研态度和高深的学术水平都深深地影响了我。感谢实验室师兄王宇辉，甘耀中，李尧和张哲对我的指导，与你们进行学术讨论和合作让我收益良多。感谢我的家人对我的支持，在外求学的这些年里只有过年才会回家团圆。每次回家看到父母头发变白我都感到非常自责，没有尽到做儿子的责任。感谢蔡梓霖同学对我的照顾和耐心陪伴，希望你能够健康快乐地完成未来的学业工作。

在研究生的学习生涯中，我收获的不只是强化学习相关的技术知识，更有其对我生活的指导意义。人的一生就如同马尔科夫决策过程一样，我们对一件事情的决策会对我们未来产生影响，我们的目标就是为了最大化我们自身的价值。但在现实中，我们没法预测遥远的未来，也没有办法通过大量的试错再进行抉择。我们只能通过前人的经验教训与自己对这个世界的观测总结来做出当下自己认为对最好的决定。人的每一次决策都要落子无悔，希望自己未来能够一切顺利。

在学期间的研究成果及学术论文情况

攻读硕士学位期间发表（录用）论文情况

1. 吴卿源, 谭晓阳. 基于 UCB 算法的交替深度 Q 网络. 中国机器学习会议 (CCML), 2021(已录用).
2. Qingyuan Wu*, Xiaoyang Tan. Graph-Constrained Monte Carlo Tree Search for Low Variance Bus Driver Scheduling. (ACM Transaction on Intelligent Systems and Technology, 期刊在审).

攻读硕士学位期间专利情况

1. 谭晓阳，吴卿源. 一种基于最大置信度上界的交替深度 Q 网络算法，发明专利，2021(已公开).

研究生期间参与的科研项目

1. 国家自然科学基金 (No.61976115, 61672280, 61732006).
2. 装备预研项目 (No. 6140312020413).
3. 南京航空航天大学 AI+ 项目 (No.56XZA18009).
4. 南京航空航天大学研究生创新基金项目 (No.Kfjj20191608).

附录 A 定理证明

A.1 第三章定理证明

以下定理表明使用 LVS 更新得到的新计划表比原先的计划表有更低的方差。

定理 A.1：LVS 算法能够将一个可行计划表 S 优化成一个新的可行计划表 S^*，其有着更低的方差。即 $Var(S^*) \leq Var(S)$, 其中 $S^* \leftarrow LVS(S)$。

证明: $Var(S)$ 表示一个可行计划表 S 的工作时间方差:

$$Var(S) = \frac{1}{n}\sum_{k=1}^{n}(g_k - g_{aver})^2 \tag{A.1}$$

其中 g_k 为第 k 个轮班的工作时间，$g_{aver} = \frac{1}{n}\sum_{k=1}^{n} g_k$ 为计划表 S 中 n 个司机的平均工作时间。

根据 LVS 算法，我们先从计划表 S 中选择了两个轮班 s_i 和 s_j，其满足条件3.1 方差 $Var(S)$ 可表示为:

$$\begin{aligned} Var(S) \quad = \quad & \tfrac{1}{n}\textstyle\sum_{k=1,k\neq i,j}^{n}(g_k - g_{aver})^2 \\ & +\tfrac{1}{n}[(g_i - g_{aver})^2 + (g_j - g_{aver})^2] \end{aligned} \tag{A.2}$$

其中 $(s_i, s_j) \leftarrow Select(S)$。

接着 $(s_i^1, s_i^2) \leftarrow Decompose(s_i)$ 将轮班 s_i 分别为两个子轮班 s_i^1 和 s_i^2，其满足条件3.2。将公式A.2视作常数 $Const$，如下公式A.3) 所示:

$$Const = \frac{1}{n}\sum_{k=1,k\neq i,j}^{n}(g_k - g_{aver})^2 \tag{A.3}$$

两个分解的子轮班 s_i^1, s_i^2 之间的休息时间为 $w_1 = g_i - g_i^1 - g_i^2$。通常其相对于 g_i 来说很小。为了方便计算，我们将其忽略，直接 $g_i \approx g_i^1 + g_i^2$。因此得到如下等式:

$$\begin{aligned} Var(S) \quad & = Const + \tfrac{1}{n}[(g_i - g_{aver})^2 + (g_j - g_{aver})^2] \\ & = Const + \tfrac{1}{n}[(g_i^1 + g_i^2 + w_1 - g_{aver})^2 + (g_j - g_{aver})^2] \\ & \approx Const + \tfrac{1}{n}[(g_i^1 + g_i^2 - g_{aver})^2 + (g_j - g_{aver})^2] \end{aligned} \tag{A.4}$$

为了不失一般性，假设我们采取了第一种 $Swap\&Merge$ 方式，交换 s_j 和 s_i^1 并将其与 s_i^2 结合得到新的轮班 s_i^*。并且 $Swap\&Merge$ 是成功的，所生成的新轮班 s_i^* 是可行的。否则我们直

接将方差下降量设置为 0。这是新计划表 S^* 的方差 $Var(S^*)$ 可以写作:

$$\begin{aligned} Var(S^*) &= Const + \tfrac{1}{n}[(g_i^1 - g_{aver})^2 + (g_j + g_i^2 + w_2 - g_{aver})^2] \\ &\approx Const + \tfrac{1}{n}[(g_i^1 - g_{aver})^2 + (g_j + g_i^2 - g_{aver})^2] \end{aligned} \tag{A.5}$$

其中 w_2 为班次 s_j 和子班次 s_i^2 之间的休息时间，可以和 w_1 一样进行忽略。

因此，LVS 所优化计划表 S 的方差下降量 Δ_1 可以得到:

$$\begin{aligned} \Delta_1 &= Var(S) - Var(S^*) \\ &= \tfrac{1}{n}[(g_i - g_{aver})^2 + (g_j - g_{aver})^2] \\ &\quad -\tfrac{1}{n}[(g_i^1 - g_{aver})^2 + (g_i^2 + g_j - g_{aver})^2] \\ &= \tfrac{1}{n}[(g_i^1 + g_i^2 - g_{aver})^2 + (g_j - g_{aver})^2] \\ &\quad -\tfrac{1}{n}[(g_i^1 - g_{aver})^2 + (g_i^2 + g_j - g_{aver})^2] \\ &= \tfrac{1}{n}[(g_i^1)^2 + 2g_i^1 g_i^2 + (g_i^2)^2 - 2g_i^1 g_{aver} - 2g_i^2 g_{aver} + g_{aver}^2 \\ &\quad +(g_j)^2 - 2g_j g_{aver} + g_{aver}^2 \\ &\quad -(g_i^1)^2 + 2g_i^1 g_{aver} - g_{aver}^2 \\ &\quad -(g_i^2)^2 - 2g_i^2 g_j - g_j^2 + 2g_i^2 g_{aver} + 2g_j g_{aver} - g_{aver}^2] \\ &= \tfrac{2}{n} g_i^2 (g_i^1 - g_j) \end{aligned} \tag{A.6}$$

同样的，选择第二种 $Swap\&Merge$ 方式，轮班 s_j 与子轮班 s_i^2 进行交换并和子轮班 s_i^1 结合，其方差下降量可以写作:

$$\Delta_2 = \frac{2}{n} g_i^1 (g_i^2 - g_j) \tag{A.7}$$

基于条件3.1和条件3.2，可以得到 $\Delta_1 \geq 0$ 且 $\Delta_2 \geq 0$，这意味着新的计划表 S^* 比计划表 S 有更小的方差。 □

A.2 第五章定理证明

定理 A.2：(多样的状态访问量) 为方便分析，假设马尔科夫决策过程和策略函数是确定型的。则有:

$$|\mathbb{S}^{\text{vary}}| \leq K^T, |\mathbb{S}^{\text{consitent}}| \leq KT \tag{A.8}$$

特别地，马尔科夫决策过程满足假设5.1，并且状态转移方程是单射，如 $\mathcal{T}(s,a) \neq \mathcal{T}(s',a')$ 对任意 $s \neq s'$ 或 $a \neq a'$。并且所有策略都各不相同 $\pi_k(s) \neq \pi_{k'}(s)$ 对任 $k \neq k'$, 则有:

$$|\mathbb{S}^{\text{vary}}| = K^T, |\mathbb{S}^{\text{consitent}}| = KT \tag{A.9}$$

证明: 假设马尔科夫决策过程和所有的策略函数都是确定性的。并且所有的策略都从一个确定的初始状态开始。对于时序变化性集成探索策略，在每一个时刻其有 K 个可能的到达状态。因此在第 T 个时刻时，可得

$$|\mathbb{S}^{\text{vary}}| \leq K^T \tag{A.10}$$

特别地，如果马尔科夫决策过程和所有策略函数在任何的状态 $s \neq s'$ 或动作 $a \neq a'$ 满足 $\mathcal{T}(s,a) \neq \mathcal{T}(s',a')$，并且对所有的 $k \neq k'$ 满足 $\pi_k(s) \neq \pi_{k'}(s)$。这时智能体能够在每一个时刻访问个 K 不同的状态。并且，若马尔科夫决策过程满足假设5.1，即以为这智能体不能在一幕中重复访问同一个状态，因此有:

$$|\mathbb{S}^{\text{vary}}| = K^T \tag{A.11}$$

对于时序持续性集成探索策略，K 个策略中的每一个策略在 T 个时刻中最多访问 T 个状态。因此有

$$|\mathbb{S}^{\text{consistent}}| \leq KT \tag{A.12}$$

同样特别地，如果马尔科夫决策过程和所有策略函数在任何的状态 $s \neq s'$ 或动作 $a \neq a'$ 满足 $\mathcal{T}(s,a) \neq \mathcal{T}(s',a')$，并且对所有的 $k \neq k'$ 满足 $\pi_k(s) \neq \pi_{k'}(s)$。这时智能体能够在每一个时刻访问个 K 不同的状态。并且，若马尔科夫决策过程满足假设5.1，即以为这智能体不能在一幕中重复访问同一个状态，因此有:

$$|\mathbb{S}^{\text{consistent}}| = KT \tag{A.13}$$

□

定理 A.3：(收缩性) 对于任意 $\mathbf{q}, \mathbf{q}' \in \mathbb{Q}^K$，则有

$$||\mathbf{Tq} - \mathbf{Tq}'|| \leq \gamma ||\mathbf{q} - \mathbf{q}'|| \tag{A.14}$$

证明: 首先，通过使用算子 $\mathcal{T}_1$，可得:

$$
\begin{aligned}
&\|\mathcal{T}_1\mathbf{q} - \mathcal{T}_1\mathbf{q}'\|_1 \\
&= \left\| \mathbb{E}\left[r + \gamma \sum_{k'=1}^{K} w_{k'} \max_{a'} q_{k'}(s', a') \right] - \mathbb{E}\left[r + \gamma \sum_{k'=1}^{K} w_{k'} \max_{a'} q'_{k'}(s', a') \right] \right\| \\
&\leq \gamma \max_{s'} \left| \sum_{k'=1}^{K} w_{k'} \max_{a'} q_{k'}(s', a') - \sum_{k'=1}^{K} w_{k'} \max_{a'} q'_{k'}(s', a') \right| \\
&\leq \gamma \sum_{k'=1}^{K} w_{k'} \max_{s'} \left| \max_{a'} q_{k'}(s', a') - \max_{a'} q'_{k'}(s', a') \right| \\
&\leq \gamma \sum_{k'=1}^{K} w_{k'} \max_{s', a'} \left| q_{k'}(s', a') - q'_{k'}(s', a') \right| \\
&= \gamma \sum_{k'=1}^{K} w_{k'} \|\mathbf{q} - \mathbf{q}'\|_{k'} \\
&\leq \gamma \max_{k} \|\mathbf{q} - \mathbf{q}'\|_{k} \\
&= \gamma_1 \|\mathbf{q} - \mathbf{q}'\|
\end{aligned}
\tag{A.15}
$$

紧接着，通过使用算子 $\mathcal{T}_{2:1}$，可得:

$$
\begin{aligned}
&\|\mathcal{T}_{2:1}\mathbf{q} - \mathcal{T}_{2:1}\mathbf{q}'\|_2 \\
&= \left\| \mathbb{E}\left[r + \gamma \sum_{k'=1}^{K} w_{k'} \max_{a'} (\mathcal{T}_1\mathbf{q})_{k'}(s', a') \right] - \mathbb{E}\left[r + \gamma \sum_{k'=1}^{K} w_{k'} \max_{a'} (\mathcal{T}_1\mathbf{q}')_{k'}(s', a') \right] \right\| \\
&\leq \gamma \max_{s'} \left| w_1 \max_{a'} (\mathcal{T}_1\mathbf{q})_1(s', a') - w_1 \max_{a'} (\mathcal{T}_1\mathbf{q}')_1(s', a') \right| \\
&+ \gamma \max_{s'} \left| \sum_{k'=2}^{K} w_{k'} \max_{a'} (\mathcal{T}_1\mathbf{q})_{k'}(s', a') - \sum_{k'=2}^{K} w_{k'} \max_{a'} (\mathcal{T}_1\mathbf{q}')_{k'}(s', a') \right| \\
&\leq \gamma w_1 \max_{s', a'} |(\mathcal{T}_1\mathbf{q})_1(s', a') - (\mathcal{T}_1\mathbf{q}')_1(s', a')| \\
&+ \gamma \sum_{k'=2}^{K} w_{k'} \max_{s', a'} |(\mathcal{T}_1\mathbf{q})_{k'}(s', a') - (\mathcal{T}_1\mathbf{q}')_{k'}(s', a')| \\
&= \gamma w_1 \|\mathcal{T}_1\mathbf{q} - \mathcal{T}_1\mathbf{q}'\|_1 + \gamma \sum_{k'=2}^{K} w_{k'} \|\mathbf{q} - \mathbf{q}'\|_{k'} \\
&\leq \gamma w_1 \gamma_1 \|\mathbf{q} - \mathbf{q}'\| + \gamma \sum_{k'=2}^{K} w_{k'} \|\mathbf{q} - \mathbf{q}'\|_{k'} \\
&\leq \gamma(1 + w_1\gamma_1 - w_1) \|\mathbf{q} - \mathbf{q}'\| \\
&= \gamma_2 \|\mathbf{q} - \mathbf{q}'\| \\
&\leq \gamma \|\mathbf{q} - \mathbf{q}'\|
\end{aligned}
\tag{A.16}
$$

紧接着，通过使用算子 $\mathcal{T}_{k:1}$，可得:

$$\begin{aligned}
&\left\|\mathcal{T}_{k:1}\mathbf{q}-\mathcal{T}_{k:1}\mathbf{q}'\right\|_k \\
&\leq \gamma\sum_{k'=1}^{k-1} w_{k'}\left\|\mathcal{T}_{k':1}\mathbf{q}-\mathcal{T}_{k':1}\mathbf{q}'\right\|_{k'}+\gamma\sum_{k'=k}^{K} w_{k'}\left\|\mathbf{q}-\mathbf{q}'\right\|_{k'} \\
&\leq \gamma\sum_{k'=1}^{k-1} w_{k'}\gamma_{k'}\left\|\mathbf{q}-\mathbf{q}'\right\|+\gamma\sum_{k'=k}^{K} w_{k'}\left\|\mathbf{q}-\mathbf{q}'\right\|_{k'} \\
&\leq \gamma(1+\sum_{k'=1}^{k-1} w_{k'}\gamma_{k'}-\sum_{k'=1}^{k-1} w_{k'})\left\|\mathbf{q}-\mathbf{q}'\right\| \\
&= \gamma_k\left\|\mathbf{q}-\mathbf{q}'\right\| \\
&\leq \gamma\left\|\mathbf{q}-\mathbf{q}'\right\|
\end{aligned} \tag{A.17}$$

因此，基于以上可得:

$$\left\|\mathcal{T}_{K:1}\mathbf{q}-\mathcal{T}_{K:1}\mathbf{q}'\right\|_k \leq \gamma_k\left\|\mathbf{q}-\mathbf{q}'\right\| \leq \gamma\left\|\mathbf{q}-\mathbf{q}'\right\| \tag{A.18}$$

其中 $\gamma_1=\gamma, \gamma_k=\gamma(1+\sum_{k'=1}^{k-1} w_{k'}\gamma_{k'}-\sum_{k'=1}^{k-1} w_{k'})$ for $k=2,\cdots,K$.

综上可得:

$$\left\|\mathbf{T}\mathbf{q}-\mathbf{T}\mathbf{q}'\right\|=\max_k\left\|\mathcal{T}_{K:1}\mathbf{q}-\mathcal{T}_{K:1}\mathbf{q}'\right\|_k \leq \gamma\left\|\mathbf{q}-\mathbf{q}'\right\| \tag{A.19}$$

□

定理 A.4：(唯一不动点) 任意 $\mathbf{q}\in\mathbb{Q}^K$ 通过迭代地计算 $\mathbf{q}^{i+1}=\mathbf{T}\mathbf{q}^i(i=,0,1,\cdots)$ 收敛到一个唯一的不动点 $\mathbf{T}\mathbf{Q}^*=\mathbf{Q}^*$ 即有:

$$\lim_{i\to\infty}\mathbf{q}^i\to\mathbf{Q}^* \tag{A.20}$$

证明: 首先，对于 $\mathcal{T}_1\mathbf{Q}^*$ 可得:

$$\left(\mathcal{T}_1\mathbf{Q}^*\right)_1(s,a)=E\left[r+\gamma\sum_{k'=1}^{K} w_{k'}\max_a Q^*_{k'}(s,a)\right]=Q^*(s,a),$$

这则意味:

$$\mathcal{T}_1\mathbf{Q}^*=\mathbf{Q}^*$$

因此，假设 $\mathcal{T}_{k:1}\mathbf{Q}^* = \mathbf{Q}^*$，可得:

$$\begin{aligned}
&(\mathcal{T}_{k+1:1}\mathbf{Q}^*)_{k+1}(s,a) \\
&= (\mathcal{T}_{k+1}(\mathcal{T}_{k:1}\mathbf{Q}^*))_{k+1}(s,a) \\
&= (\mathcal{T}_{k+1}\mathbf{Q}^*)_{k+1}(s,a) \\
&= \mathbb{E}\left[r + \gamma \sum_{k'=1}^{K} w_{k'} \max_a Q^*_{k'}(s,a)\right] \\
&= Q^*(s,a)
\end{aligned} \tag{A.21}$$

这意味着:

$$\mathcal{T}_{k+1:1}\mathbf{Q}^* = \mathbf{Q}^*$$

因此，可得:

$$\mathcal{T}_{2:1}\mathbf{Q}^* = \mathbf{Q}^*, \mathcal{T}_{3:1}\mathbf{Q}^* = \mathbf{Q}^*, \cdots, \mathcal{T}_{K:1}\mathbf{Q}^* = \mathbf{Q}^*$$

(由于 $\mathbf{T} \triangleq \mathcal{T}_{K:1}$)

基于上述可得，并且定理5.2表明 $\mathbf{Q}^*$ 是收缩算子 $\mathbf{T}$ 的唯一不同点。因此可得定理5.3。 □

定理 A.5：(收敛速度) 对 $\mathbf{q} \in \mathbb{Q}^K$，若 $\mathbf{q}$ 满足条件5.3，则有:

$$||\mathbf{Tq} - \mathbf{Q}^*|| \leq ||\mathbf{T}^{\mathrm{ME}}\mathbf{q} - \mathbf{Q}^*|| \tag{A.22}$$

证明: 首先，对任何 (s,a)，可得:

$$\begin{aligned}
(\mathcal{T}_1\mathbf{q})_1(s,a) &= \mathbb{E}\left[r + \gamma \sum_{k'=1}^{K} w_{k'} \max_{a'} q_{k'}(s',a')\right] \\
&\geq \mathbb{E}\left[r + \gamma \max_{a'} \sum_{k'=1}^{K} w_{k'} q_{k'}(s',a')\right] \\
&= (\mathcal{T}_1^{\mathrm{ME}}\mathbf{q})_1(s,a)
\end{aligned} \tag{A.23}$$

接着，可得:

$$\begin{aligned}
(\mathcal{T}_{2:1}\mathbf{q})_2 &= \mathbb{E}\left[r+\gamma\sum_{k'=1}^{K} w_{k'}\max_{a'}(\mathcal{T}_1\mathbf{q})_{k'}(s',a')\right] \\
&= \mathbb{E}\left[r+\gamma w_1\max_{a'}(\mathcal{T}_1\mathbf{q})_1(s',a')+\gamma\sum_{k'=2}^{K} w_{k'}\max_{a'} q_{k'}(s',a')\right] \\
&\overset{\text{Eq.}A.23}{\geq} \mathbb{E}\left[r+\gamma w_1\max_{a'}(\mathcal{T}_1^{\mathrm{ME}}\mathbf{q})_1(s',a')+\gamma\sum_{k'=2}^{K} w_{k'}\max_{a'} q_{k'}(s',a')\right] \\
&= \mathbb{E}\left[r+\gamma\sum_{k'=1}^{K} w_{k'}\max_{a'}(\mathcal{T}_1^{\mathrm{ME}}\mathbf{q})_{k'}(s',a')\right] \\
&= (\mathcal{T}_{2:1}^{\mathrm{ME}}\mathbf{q})_2
\end{aligned} \tag{A.24}$$

紧接着，可得:

$$(\mathcal{T}_{3:1}\mathbf{q})_3 \geq (\mathcal{T}_{3:1}^{\mathrm{ME}}\mathbf{q})_3, \cdots, (\mathcal{T}_{K:1}\mathbf{q})_K \geq (\mathcal{T}_{K:1}^{\mathrm{ME}}\mathbf{q})_K$$

因此，可得:

$$\mathbf{T}\mathbf{q} \geq \mathbf{T}^{\mathrm{ME}}\mathbf{q}$$

这是，若 $\mathbf{q} \leq \mathbf{Q}^*$，这时有:

$$\mathbf{T}\mathbf{q} \leq \mathbf{Q}^*, \mathbf{T}^{\mathrm{ME}}\mathbf{q} \leq \mathbf{Q}^* \tag{A.25}$$

若

$$k', s', a' = \arg\max_{k,s,a} |(\mathbf{T}\mathbf{q})_k(s,a) - Q^*(s,a)|$$

则有:

$$\begin{aligned}
\|\mathbf{T}\mathbf{q}-\mathbf{Q}^*\| &= \max_k \max_{s,a} |(\mathbf{T}\mathbf{q})_k(s,a) - Q^*(s,a)| \\
&= |(\mathbf{T}\mathbf{q})_{k'}(s',a') - Q^*(s',a')| \\
&\overset{\text{Eq. }A.25}{=} Q^*(s',a') - (\mathbf{T}\mathbf{q})_{k'}(s',a') \\
&\overset{\text{Eq. }A.23}{\leq} Q^*(s',a') - (\mathbf{T}^{\mathrm{ME}}\mathbf{q})_{k'}(s',a') \\
&\overset{\text{Eq. }A.25}{=} \left|(\mathbf{T}^{\mathrm{ME}}\mathbf{q})_{k'}(s',a') - Q^*(s',a')\right| \\
&\leq \max_k \max_{s,a} \left|(\mathbf{T}^{\mathrm{ME}}\mathbf{q})_k(s,a) - Q^*(s,a)\right| \\
&= \left\|\mathbf{T}^{\mathrm{ME}}\mathbf{q}-\mathbf{Q}^*\right\|
\end{aligned} \tag{A.26}$$

□