文章编号: 1002-0446(2003)04-0344-06

一种新的多智能体强化学习算法及其在多机器人协作任务中的应用*

顾国昌[1], 仲 宇[1], 张汝波[1,2]

(1. 哈尔滨工程大学计算机科学与技术学院, 哈尔滨 150001; 2. 中国科学院沈阳自动化研究所机器人学重点实验室, 沈阳 110016)

摘 要: 在多机器人系统中, 评价一个机器人行为的好坏常常依赖于其它机器人的行为, 此时必须采用组合动作以实现多机器人的协作, 但采用组合动作的强化学习算法由于学习空间异常庞大而收敛得极慢. 本文提出的新方法通过预测各机器人执行动作的概率来降低学习空间的维数, 并应用于多机器人协作任务之中. 实验结果表明, 基于预测的加速强化学习算法可以比原始算法更快地获得多机器人的协作策略.

关键词: 分布式强化学习; 加速算法; 多智能体系统

中图分类号: TP24 文献标识码: B

A NEW MULTI-AGENT REINFORCEMENT LEARNING ALGORITHM AND ITS APPLICATION TO MULTI-ROBOT COOPERATION TASKS

GU Guo-chang[1], ZHONG Yu[1], ZHANG Ru-bo[1,2]

(1. *School of Computer Science and Technology, Harbin Engineering University, Harbin* 150001;
2. *Robotics Laboratory, Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang* 110016, *China*)

Abstract: In multi-robot systems, joint-action must be employed to achieve cooperation because the evaluation to the behavior of a robot often depends on the other robots' behaviors. However, joint-action reinforcement learning algorithms suffer the slow convergence rate because of the enormous learning space produced by joint-action. In this paper, a prediction-based reinforcement learning algorithm is presented for multi-robot cooperation tasks, which demands all robots to learn paper predict the probabilities of actions that other robots may execute. A multi-robot cooperation experiment is made to test the efficacy of the new algorithm, and the experiment results show that the new algorithm can achieve the cooperation strategy much faster than the primitive reinforcement learning algorithm.

Keywords: distributed reinforcement learning, accelerating algorithm, multi-agent system

1 引言(Introduction)

强化学习(Reinforcement Learning, RL)是一种重要的机器学习方法, 其本质是一种基于 Markov 决策过程的异步动态规划方法, 可用来获得动态环境下的自适应性反应式行为[1]. 瞬时差分算法(Temporal Difference, TD)[2]、Q 学习算法[3]和自适应启发评价算法(Adaptive Heuristic Critic, AHC)[4]是最重要的三种强化学习算法.

标准的强化学习过程是以一个单独的学习单元为载体进行的, 这个学习单元可以看作一个智能体(Agent). 这个 Agent 在感知到当前环境状态的条件下通过对环境采取试探行为, 根据环境对此"状态—动作对"(State-Action Pair, SAP)的评价信号来不断修改从状态到动作的映射策略, 最终学会对所有环境状态的反应策略.

强化学习是一个人工智能范畴中的概念, 如果引申到分布式人工智能范畴, 相应地就得到分布式强化学习的概念. 分布式强化学习并没有一个公认

* 基金项目: 本文得到中国科学院机器人学开放研究实验室基金资助(RL200106); 武器装备预研基金项目及国防基础研究基金的资助.

收稿日期: 2003-02-11

的定义, 笔者认为, 如果强化学习系统由多个学习单元组成, 每个单元独立地执行部分的强化学习任务, 最后达到整个系统意义上的学习目标, 这个学习系统就称作分布式强化学习系统, 这个学习过程就可以称作分布式强化学习过程. 现有的分布式强化学习算法大多都是基于 Q 学习的, 这可能是 Q 学习最符合强化学习流程从而得到最广泛应用的结果.

分布式强化学习系统中的每个 Agent 可以直接应用标准的强化学习算法, 这时系统中的 Agent 只考虑自己的状态和动作而不关心其它 Agent 的状态和动作, 各个 Agent 获得的强化信号也只与自己的状态和动作相联系, 这种分布式强化学习算法可以称为独立强化学习(Reinforcement Learning Individually, RLI). 梯度法[5]、嵌入先验信息[6]、模糊学习[7]、状态空间划分[8]、分层学习[9]等现有的强化学习加速算法可以直接应用于 RLI 系统.

由于 RLI 系统中的 Agent 都以自我为中心, 所以不易达成协作. 为了克服 RLI 系统的这种局限性, 产生了群体强化学习方法(Reinforcement Learning in Groups, RLG), 它采用组合动作(所有 Agent 的动作组成的动作向量)来保证整个团队协调地完成预定任务. 为了保证必需的环境变迁信息, RLG 算法通常也采用组合状态, 这样每个 Agent 的 Q 函数表都是组合状态和组合动作到 Q 值的映射, 进行决策时必须考虑其它 Agent 的状态和动作, 所以其学习空间的规模是 Agent 个数的指数函数, Agent 个数略多就会使得学习速度慢得不能忍受. 基于存储的协同学习(Memory Based CoLearning, MBCL)和基于树的协同学习(Tree Based CoLearning, TBCL)两种算法采用了结构化的存储方法, 其学习速度比原始 RLG 方法有所改善, 但未解决 RLG 学习空间规模是 Agent 个数的指数函数这个根本问题[10]. 针对这个问题, 本文将探讨一种降低 RLG 学习空间维数、加快 RLG 算法收敛速度的新方法.

2　RLG 学习算法(RLG learning algorithms)

Littman 首先提出第一个 RLG 算法[11], 并于 1996 年证明了此算法的收敛性[12]. Littman 的算法限定一个 Agent 的收益必须等于其它 Agent 的损失, 为了摆脱这个限制, Hu Jinling 同样以 Markov 竞赛为框架提出了基于 Nash 均衡思想的另一种 RLG 算法, 并证明了此算法的收敛性[13]. RLI 算法与 RLG 算法的学习步骤相同, 它们的区别在于 Q 函数的更新规则.

RLI 系统中 Agent k 的 Q 函数更新规则为

$$Q_t^k(s_t^k, a_t^k) = (1 - \alpha_t) Q_{t-1}^k(s_t^k, a_t^k) + \alpha_t[r_t^k + \beta \max_{a^k \in A^k} Q_{t-1}^k(s_{t+1}^k, a^k)] \tag{1}$$

其中:

s_t^k—— Agent k 在 t 时刻的状态;

a_t^k—— Agent k 在 t 时刻选择的动作;

r_t^k—— Agent k 在 t 时刻收到的强化信号;

α_t—— t 时刻的学习率;

$Q_t^k(s_t^k, a_t^k)$—— Agent k 在 t 时刻的 Q 值.

RLG 系统中 Agent k 的 Q 函数更新规则为

$$Q_t^k(\vec{s_t}, \vec{a_t}) = (1 - \alpha_t) Q_{t-1}^k(\vec{s_t}, \vec{a_t}) + \alpha_t[r_t^k + \beta \pi^1(\vec{s_{t+1}}) \cdots \pi^n(\vec{s_{t+1}}) Q_{t-1}^k(\vec{s_{t+1}})] \tag{2}$$

其中:

$\vec{s_t}$—— t 时刻所有 Agent 的组合状态, $\vec{s_t} \equiv (s_t^1, s_t^2, \cdots, s_t^n)$;

$\vec{a_t}$—— t 时刻所有 Agent 的组合动作, $\vec{a_t} \equiv (a_t^1, a_t^2, \cdots, a_t^n)$;

$Q_t^k(\vec{s_t}, \vec{a_t})$—— Agent k 在 t 时刻的 Q 值;

$Q_{t-1}^k(\vec{s_{t+1}})$—— Agent k 在 t 时刻的 n 维 Q 矩阵, 其元素为 $Q_{t-1}^k(\vec{s_{t+1}}, a^1, a^2, \cdots, a^n)$;

$\pi_t^k(\vec{s_{t+1}})$—— Agent k 在 t 时刻的策略.

3　通过预测加快分布式强化学习算法的收敛速度(Accelerating the convergence of distributed reinforcement learning algorithms by prediction)

3.1　基本思想

人类个体通常只考虑自己周围的环境状态, 同时根据对其他人可能采取什么行为的猜测来决定自己的行为, 别人所处状态仅用作猜测的根据. RLG 系统中的 Agent 可以参照人类的决策过程来减少学习过程中需要考虑的因素, 从而缩短学习过程.

另外, 由于所有 Agent 同时选择动作, 这样在 t 时刻任一 Agent 都无法得知其它 Agent 将执行什么动作, 所以 Agent 仍然无法根据来选择自己的动作. 下面讨论的预测法不仅能降低 RLG 算法的组合强度, 而且可以用来实现 RLG 算法的动作选择策略.

3.2　预测动作以加快学习和实现动作选择机制

对 Agent i 可能执行动作的概率预测函数 I 的更新规则如下

$I_{t+1}^i(\vec{s_t}, a_k^i) =$

$$\begin{cases} I_t^i(\vec{s_t}, a_k^i) + \beta \sum_{a_t^i \in A - \{a_k^i\}} I_t^i(\vec{s_t}, a_l^i) & \text{if } a_k^i = a_t^i \\ (1 - \beta) I_t^i(\vec{s_t}, a_k^i) & \text{otherwise} \end{cases} \tag{3}$$

可以看出, (3) 实际上是一个随机学习自动机 (Stochastic Learning Automaton, SLA), 其中 a_k^i 表示 Agent i 动作集 A^i 中的第 k 个动作(a_t^i 的下标若为 t 或 $t+1$ 则表示时间, 否则表示在动作集中的序号), β 是预测学习模型的学习率.

根据(3)式更新的概率预测函数 I 总能保证 $t+1$ 时刻对所有 a_k^i 的预测概率之和与 t 时刻对所有 a_k^i 的预测概率之和相等

$$\begin{aligned} \sum_{a_k^i \in A} I_{t+1}^i(\vec{s_t}, a_k^i) &= I_{t+1}^i(\vec{s_t}, a_t^i) + \sum_{a_k^i \in A - \{a_t^i\}} I_{t+1}^i(\vec{s_t}, a_k^i) \\ &= (I_t^i(\vec{s_t}, a_t^i) + \beta \sum_{a_k^i \in A - \{a_t^i\}} I_t^i(\vec{s_t}, a_k^i)) \\ &\quad + \sum_{a_k^i \in A - \{a_t^i\}} (1 - \beta) I_t^i(\vec{s_t}, a_k^i) \\ &= I_t^i(\vec{s_t}, a_t^i) + \beta \sum_{a_k^i \in A - \{a_t^i\}} I_t^i(\vec{s_t}, a_k^i) \\ &\quad + (1 - \beta) \sum_{a_k^i \in A - \{a_t^i\}} I_t^i(\vec{s_t}, a_k^i) \\ &= I_t^i(\vec{s_t}, a_k^i) + \sum_{a_k^i \in A - \{a_t^i\}} I_t^i(\vec{s_t}, a_k^i) \\ &= \sum_{a_k^i \in A} I_t^i(\vec{s_t}, a_k^i) \end{aligned} \tag{4}$$

因此, 只要 0 时刻对所有 α_k^i 的预测概率之和设定为 1, 就可以保证在任何时刻对所有 a_k^i 的预测概率之和为 1, 即对任意时刻 t 有

$$\sum_{a_k^i \in A} I_t^i(\vec{s_t}, a_k^i) = 1 \tag{5}$$

由于预测函数 I 中已经隐含了 $\vec{s}$, 所以 Agent 的 Q 表中只需保留对自己的状态和别人的动作的评价, 即用 $Q(s^1, \vec{a})$ 来代替 $Q(\vec{s}, \vec{a})$.

这时 Agent k 的 Q 函数更新规则为

$$\begin{aligned} Q_t^k(s_t^k, \vec{a_t}) &= (1 - \alpha_t) Q_{t-1}^k(s_t^k, \vec{a_t}) \\ &\quad + \beta \pi^1(\vec{s_{t+1}}) \cdots \pi^n(\vec{s_{t+1}}) Q_{t-1}^k(s_{t+1}^k)\ ] \end{aligned} \tag{6}$$

其中

$$\pi^1(\vec{s_{t+1}}) \cdots \pi^n(\vec{s_{t+1}}) Q_{t-1}^k(s_{t+1}^k) = \sum_{a^1 \in A} \sum_{a^2 \in A} \cdots \sum_{a^n \in A} I_t^1(\vec{s_{t+1}}, a^1) \cdots I_t^n(\vec{s_{t+1}}, a^n) Q_{t-1}^k(s_{t+1}^k, a^1, a^2, \cdots, a^n) \tag{7}$$

对具有相同状态空间 S 和动作空间 A 的 n 个 Agent 来说, 采用形如 $Q(\vec{s}, \vec{a})$ 的 Q 表需要大小为 $n \times |S|^n \times |A|^n$ 的存储空间, 而采用形如 $Q(s^1, \vec{a})$ 的 Q 表需要大小为 $n \times |S| \times |A|^n$ 的存储空间, 预测函数 I 需要大小为 $n \times |S|^n \times |A|$ 的存储空间, 总共需要大小为 $n \times |S| \times |A|^n + n \times |S|^n \times |A|$ 的存储空间, 远远小于 $n \times |S|^n \times |A|^n$, 所以通过预测动作的确可以减小学习空间, 加快学习速度.

本文提出的预测方法还可以很容易地实现 RLG 算法的动作选择机制. 最简单的方法可以将 $a^2, \cdots, a^n$ 取为最大的预测概率值对应的动作, 然后使用 Boltzmann 机选择 a^1; 或者对 $a^2, \cdots, a^n$ 按照预测概率加权, 首先获得

$$Q(s^1, a^1, a^2, \cdots, s^{n-1}) = \sum_{a^n \in A} I^n(\vec{s}, a^n) Q(s^1, a^1, a^2, \cdots, a^n) \tag{8}$$

依次下去, 最后得到

$$Q(s^1, a^1) = \sum_{a^2 \in A} \sum_{a^3 \in A} \cdots \sum_{a^n \in A} I^2(\vec{s}, a^2) I^3(\vec{s}, a^3) \cdots I^n(\vec{s}, a^n) Q(s^1, a^1, a^2, \cdots, a^n) \tag{9}$$

然后用 Boltzmann 机根据 $Q(s^1, a^1)$ 选择 a^1, 如下式

$$\text{prob}(a_k^1) = \frac{e^{\gamma Q(s^1, a_k^1)/T}}{\sum_{a_t^1 \in A^1} e^{\gamma Q(s^1, a_t^1)/T}} \tag{10}$$

其它 Agent 的动作选择过程与此类似, Hu 已经证明 RLG 算法的收敛性与其动作选择机制无关, 所以用以上方法进行动作选择不会影响 RLG 算法的收敛性[13].

3.3 预测状态以加快学习和实现动作选择机制

某些任务背景下, Agent 对于其它 Agent 下一个时刻将达到什么状态比对它们下一个时刻将执行什么动作更感兴趣. 例如在追捕任务中, 猎手 Agent 关心的不是猎物 Agent 向哪个方向移动这种相对动作, 而是猎物 Agent 将移动到什么位置这种后续状态, 此时对猎物 Agent 下一个时刻的状态进行预测将更有价值.

对 Agent i 在 $t+1$ 时刻将到达状态的概率预测函数 I 的更新规则如下

$$I_{t+1}^i(\vec{s_t}, s_k^i) = \begin{cases} I_t^i(\vec{s_t}, s_k^i) + \beta \sum_{s_l^i \in S - \{s_k^i\}} I_t^i(\vec{s_t}, s_l^i) & \text{if } s_k^i = s_{t+1}^i \\ (1 - \beta) I_t^i(\vec{s_t}, s_k^i) & \text{otherwise} \end{cases} \tag{11}$$

s_k^i 表示 Agent i 状态集中的第 k 个状态. 与(5)类似, 对任意时刻 t 有

$$\sum_{s_k^i \in S} I_t^i(\vec{s_t}, s_k^i) = 1 \tag{12}$$

同样有

$$Q(s^1,a^1)=\sum_{s^2\in S}\sum_{s^3\in S}\cdots\sum_{s^n\in S}I^2(\vec{s},s^2)I^3(\vec{s},s^3)\cdots I^n(\vec{s},s^n)Q(s^1,s^2,s^3,\cdots,s^n,a^1) \tag{13}$$

然后用如(10)所示 Boltzmann 机根据 $Q(s^1,a^1)$ 选择 a^1 即可.

4　仿真实验(Simulation)

4.1　实验场景

为了验证提出的加速算法的加速效果, 本文设计了一个新的实验背景对原始 RLG 算法和加速 RLG 算法的收敛速度进行了比较. 此实验中有 4 个机器人叠在一起, 它们必须协调地左右移动才能阻止球落下, 如图 1 所示.

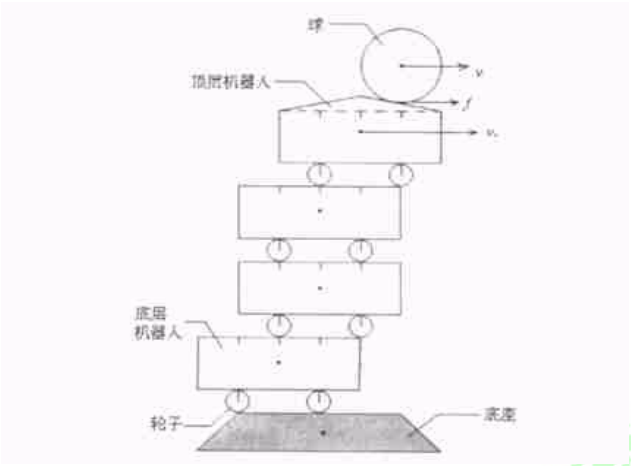

图 1　仿真实验场景图

Fig. 1　The playground

可以假设球与顶层机器人之间无相对滑动, 则下一时刻球的绝对速度与当前时刻球的水平速度、球的角速度、顶层机器人的水平速度、以及下一时刻顶层机器人的水平速度等物理量相关. 球的运动模型可描述为

$$\begin{cases}\frac{1}{2}mv^2+\frac{1}{2}I\tilde{\omega}^2+fs=\frac{1}{2}mv'^2+\frac{1}{2}\tilde{\omega}'^2\\ a=(v'-v)/t=v'-v\\ I=\frac{2}{5}mr^2\\ f=ma\\ s=(v'_c-\frac{v'+v}{2})t=v'_c-\frac{v'+v}{2}\\ r\tilde{\omega}'=v'_c-v'\\ r\tilde{\omega}=v_c-v\end{cases} \tag{14}$$

其中 v_c、v_c' 是当前时刻和下一时刻顶层机器人的水平速度, 也就是所有机器人水平速度之和; v、$\tilde{\omega}$ 是当前时刻球的水平速度和角速度, v'、$\tilde{\omega}'$ 是下一时刻球的水平速度和角速度; I、m、r 分别是球的转动惯量、质量和半径; s、a 分别是当前时刻到下一时刻球的位移和平均水平加速度; 球所受水平合力 f 中已经包含了重力的水平分量, 所以 a 中也包含了重力加速度 g 的水平分量; 由于本模型模拟的是相邻两个时刻之间球的运动状况, 所以设定 $t=1$. 求解(14), 最后得到 $v'=\frac{2}{5}v_c-\frac{4}{5}v'_c-\frac{1}{5}v$. 为了保证状态集为有限集, 需要将 v' 圆整为整数, 以避免球的位置落到状态集之外.

4.2　学习系统的状态、动作与强化机制

在初始时刻, 底座、所有机器人以及球的中心点都在同一条垂线上. 机器人每个时间步执行的动作可以是相对于下面的机器人(对最底层机器人来说是相对于底座)左移 1 cm、不动、右移 1 cm 三者之一, 所以机器人的动作集为{-1, 0, 1}; 设定每个机器人长 4cm, 轮距 2cm, 机器人(或球)的状态定义为自身与下面的机器人(对最底层机器人来说是相对于底座)重心垂线的水平距离, 由于机器人只能处于与下面的机器人(对最底层机器人来说是相对于底座)重心垂线重合或相差 1 cm 的位置上, 否则就会落下, 所以机器人的状态集为{-1, 0, 1}, 与此类似球的状态集为{-2, -1, 0, 1, 2}.

强化信号的定义为: 当机器人的车轮超出下面的机器人(对最底层机器人来说是相对于底座)的边缘而落下时, 对此机器人给予-1 的惩罚强化信号, 当球超出顶层机器人的边缘而落下时, 对所有机器人给予-1 的惩罚强化信号; 当机器人与其下面的机器人的重心垂线重合时, 对此机器人给予 0.3 的奖励强化信号, 当机器人与其下面的机器人的重心垂线水平距离为 1cm 时, 对此机器人给予 0.1 的奖励强化信号, 当球与其下面的机器人的重心垂线水平距离为 0、1、2cm 时, 对所有机器人给予 0.7、0.5、0.2 的奖励强化信号.

根据以上的状态和动作定义, 对每一个机器人来说, 原始 RLG 算法和加速 RLG 算法的 Q 表都有 81 列, 对应于 4 个机器人所有的组合动作; 原始 RLG 算法的 Q 表有 405 行, 对应于 4 个机器人和球所有的组合状态, 而加速 RLG 算法的 Q 表只有 15 行, 对应于本机器人和球的组合状态, 显然加速 RLG 算法的 Q 表远远小于原始 RLG 算法的 Q 表.

4.3　实验结果

同时运行原始 RLG 算法和加速 RLG 算法以比较两种算法的收敛速度. 两种算法都是每学习 10 个

时间步就进行一次试验, 并记录此次试验中球保持不落的步数, 如果球落下或者 20 个时间步后球仍然不落则结束本次试验. 球保持不落步数的 10 次试验(100 个时间步)平均值变化情况如图 2 所示, 可见加速 RLG 算法在 10000 步之内即可收敛, 而原始 RLG 算法在学习 50 万步后仍未完全收敛, 显然本文提出的加速 RLG 算法收敛速度大大超过原始 RLG 算法.

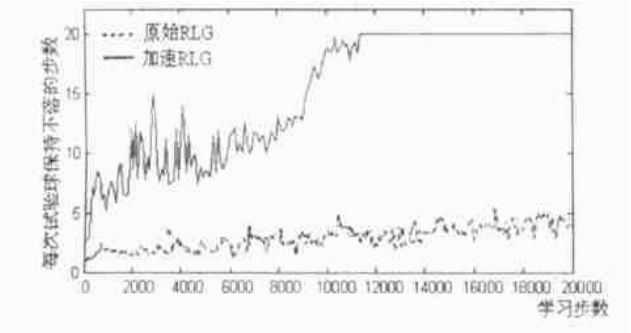

图 2 原始 RLG 算法和加速 RLG 算法的收敛速度比较

Fig. 2 The convergence comparison between the primitive RLG and the accelerated RLG

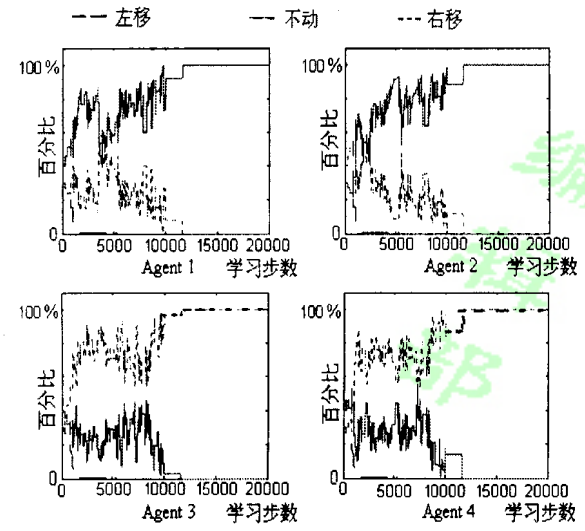

图 3 预测函数 I 的收敛情况

Fig. 3 The convergence of predictive function I

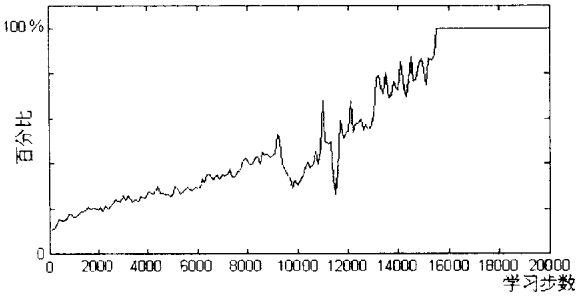

图 4 预测正确率的变化曲线

Fig. 4 Graph of correct predictions

图 3 显示以组合状态(- 1, 0, - 1, 0, 1)为例时预测函数 I 的收敛情况, 可以看出预测函数 I 与在学习 10000 步左右同步收敛, 这说明 I 与通过相互影响而达到共同收敛. 图 4 为每 100 步平均预测正确率的变化曲线图, 可见预测函数 I 对被执行动作的预测越来越准确.

5 结论(Conclusion)

本文提出一种基于预测的 RLG 加速算法, 预测函数将组合状态变换为对各 Agent 执行所有可能动作的概率, 然后按照标准的强化学习算法流程进行学习, 从而摆脱组合状态对 Q 函数的影响.

仿真实验证明, 本方法的确可以大大减小 RLG 算法的学习空间从而加快其学习速度, 同时还可获得 RLG 算法的动作选择机制. 由于以往对 RLG 算法最多用 2 个 Agent 进行讨论, 本文通过学习能够使得 4 个 Agent 达成协作的事实本身就证明了预测算法的加速效果.

根据 Watkins 的 Q 学习收敛性论断, 形如 $Q(s, \vec{a})$ 或 $I(\vec{s}, a)$ 的 Q 学习方法都不能保证收敛, 但是在本文中将这两种形式结合起来代替形如 $Q(\vec{s}, \vec{a})$ 的 RLG 算法却获得了成功. 这是由于 $I(\vec{s}, a)$ 提供了 $Q(s, \vec{a})$ 中缺少的状态信息, 而 $Q(s, \vec{a})$ 的收敛使得预测函数 $I(\vec{s}, a)$ 逐渐稳定, 二者相互促进, 最后达到共同收敛.

参考文献 (References)

[1] 张汝波, 顾国昌, 刘照德, 王醒策. 强化学习理论、算法及应用[J]. 控制理论与应用. 2000, **17**(5): 637- 642.

[2] Sutton R S. Learning to predict by the methods of temporal difference[J]. Machine Learning. 1988, (3): 9- 44.

[3] Watkins J C H, Dayan Peter. Q-learning[J]. Machine Learning. 1992, (8): 279- 292.

[4] Sutton R S. Temporal credit assignment in reinforcement learning[D]. University of Massachusetts, Amherst, MA, 1984.

[5] Masayuki Yamamura, Takashi Onozuka. Reinforcement learning with knowledge by using a stochastic gradient method on a bayesian network[A]. Proceedings of the 1998 IEEE International Conference on Neural Networks[C]. May 4-9 1998. Anchorage, Alaska, USA: 2045- 2050.

[6] Carlos H C Ribeiro. Embedding a priori knowledge in reinforcement learning[J]. Journal of Intelligent and Robotic Systems. 1998, 21: 51- 71.

[7] ChiHyon Oh, Tomoharu Nakashima, Hisao Ishibuchi. Initialization of Q-values by fuzzy rules for accelerating Q-learning[A]. Proceedings of the 1998 IEEE International Conference on Neural Networks[C]. May 4-9 1998. Anchorage, Alaska, USA: 2051- 2056

(下转第 362 页)