DOI: 10.7641/CTA.2016.60512

概率近似正确的强化学习算法解决连续状态空间控制问题

朱圆恒, 赵冬斌†

(中国科学院自动化研究所 复杂系统管理与控制国家重点实验室, 北京 100190)

摘要: 在线学习时长是强化学习算法的一个重要指标. 传统在线强化学习算法如Q学习、状态–动作–奖励–状态–动作(state-action-reward-state-action, SARSA)等算法不能从理论分析角度给出定量的在线学习时长上界. 本文引入概率近似正确(probably approximately correct, PAC)原理, 为连续时间确定性系统设计基于数据的在线强化学习算法. 这类算法有效记录在线数据, 同时考虑强化学习算法对状态空间探索的需求, 能够在有限在线学习时间内输出近似最优的控制. 我们提出算法的两种实现方式, 分别使用状态离散化和kd树(k-dimensional树)技术, 存储数据和计算在线策略. 最后我们将提出的两个算法应用在双连杆机械臂运动控制上, 观察算法的效果并进行比较.

关键词: 强化学习; 概率近似正确; kd树; 双连杆机械臂

中图分类号: TP273　　**文献标识码**: A

Probably approximately correct reinforcement learning solving continuous-state control problem

ZHU Yuan-heng, ZHAO Dong-bin†

(State Key Laboratory of Management and Control for Complex Systems, Institution of Automation, Chinese Academy of Sciences, Beijing 100190, China)

Abstract: One important factor of reinforcement learning (RL) algorithms is the online learning time. Conventional algorithms such Q-learning and state-action-reward-state-action (SARSA) can not give the quantitative analysis on the upper bound of the online learning time. In this paper, we employ the idea of probably approximately correct (PAC) and design the data-driven online RL algorithm for continuous-time deterministic systems. This class of algorithms efficiently record online observations and keep in mind the exploration required by online RL. They are capable to learn the near-optimal policy within a finite time length. Two algorithms are developed, separately based on state discretization and kd-tree technique, which are used to store data and compute online policies. Both algorithms are applied to the two-link manipulator to observe the performance.

Key words: reinforcement learning; probably approximately correct; kd-tree; two-link manipulator

1 引言(Introduction)

强化学习(reinforcement learning, RL)[1–4]通过在线学习的方式, 与被控系统或环境进行交互, 调整策略使系统获得尽可能高的累加奖励. 这类方法在解决模型未知系统的控制问题时有着显著的意义. 但是传统RL[5–6]具有数据利用率低、探索效率差的缺点, 算法没有明确的结束运行的标准, 在任意一个时刻都无法保证学到的结果已经是最优或近似最优. 此外算法的学习是随机的, 每次实验结果都不一致. 近年来研究者提出了深度强化学习[7–9], 目的是直接学习从像素到动作的策略, 学习难度被大大增加, 因此更需要考虑在线学习时长的问题. 从本质上讲, 传统RL没有从理论角度分析算法运行时间上限, 以及最终输出策略的近似最优性.

近年来提出的近似最优在线强化学习能够克服传统RL存在的问题. 这一类算法的特点是将在线观测量有效地存储起来作为模型信息, 然后利用这些信息求解性能函数和控制策略. 同时兼顾对模型未知状态区域的探索. 对未知区域增加一定的奖励或设计转移动作, 使求得的策略能够将系统转移到这些状态区域, 获取新的模型信息(如图1). 经过完整的理论分析, 证明这类算法从开始到无穷时刻, 系统在线执行非近似

收稿日期: 2016−07−14; 录用日期: 2016−10−17.

†通信作者. E-mail: dongbin.zhao@ia.ac.cn.

本文责任编委: 苏剑波.

国家自然科学基金项目(61273136, 61573353, 61533017, 61603382), 复杂系统管理与控制国家重点实验室优秀人才基金项目资助.

Supported by National Natural Science Foundation of China (61273136, 61573353, 61533017, 61603382) and Early Career Development Award of SKLMCCS.

最优动作或策略的时刻是有限的, 即满足概率近似正确(probably approximately correct, PAC)原理. 如果系统初始状态可重复, 那么这些算法只需要有限的在线运行时间即可学到近似最优控制策略.

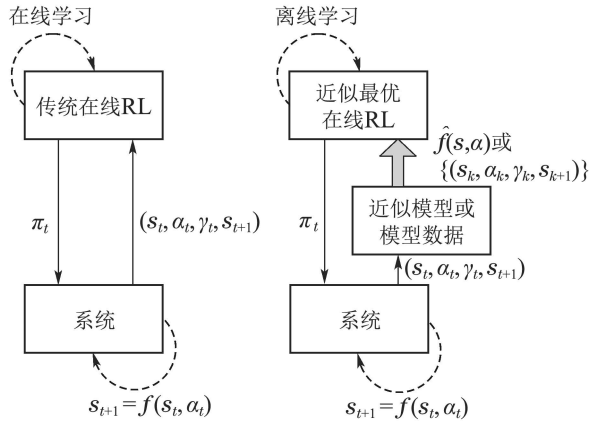


图 1 传统在线强化学习和近似最优强化学习在算法结构上的比较

Fig. 1 The differences between conventional RL and near-optimal RL in structures

针对有限状态的马尔可夫决策问题(Markov decision problems, MDPs)目前已经提出很多近似最优在线RL算法, 如文献[10–11]. 但是这些算法只能适用于有限状态–动作的系统. 为了解决连续状态系统问题, Bernstein和Shimkin[12]提出了自适应分辨率强化学习(adaptive-resolution reinforcement learning, ARL) 算法, 将在线观测量存储起来, 同时添加置信度表示邻域区域的模型信息, 由此计算得到的性能函数是最优性能函数的上界. 他们证明了ARL算法满足PAC原理, 并且执行非近似最优控制策略的时刻是有限的. 但是算法在计算置信度时需要系统模型的相关参数. 当模型完全未知时, 该算法将无法正常运行. 除了ARL算法外, 连续概率近似正确最优探索 (continuous PAC optimal exploration, C–PACE) 算法也面临同样的问题[13].

在本文中笔者考虑连续状态离散动作系统的控制问题, 提出近似最优在线强化学习算法, 给出两种算法实现的方式. 分别使用状态离散化和kd树技术, 将在线观测量有选择地存储在数据集中, 然后定义迭代算子利用存储的数据计算性能函数和在线执行的控制策略. 经过分析, 证明两个算法都满足PAC原理. 读者可以在文献[14]和文献[15]中找到引用的引理和定理的完整表述和证明. 与其他近似最优在线RL算法相比, 两个新算法完全不依赖系统模型信息, 有着更广泛的应用范围. 最后, 笔者在双连杆机械臂上比较两个算法的学习效果.

2 问题描述(Problem description)

本文要研究的对象是连续状态离散动作系统, 用(S,A,r,f)表示. 其中: S是连续状态空间, A是离散动作集, $r(s,a)$是在状态$s\in S$和动作$a\in A$下的奖励函数, $f(s,a)$是状态转移函数, 代表(s,a)的下一时刻状态. 假设S不是无限延伸的, 而是有界的. 奖励函数同样有界, 满足$r_{\min}\leqslant r(s,a)\leqslant r_{\max}$. 需要强调的是在算法运行过程中$f$是完全未知的.

假定当前时刻是t, 将系统过去时刻的状态和动作表示成

$$h_t=\{s_0,a_0,s_1,a_1,\cdots,s_{t-1},a_{t-1},s_t\}.$$

对于在线算法, 策略在运行过程中会被实时调整. 因而系统执行的策略是非静态的, 控制动作是由一系列不同时刻的策略所决定, $\pi=\{\pi_t\}_{t=0}^{\infty}$, 即$a_t=\pi_t(s_t)$.

选择衰减收益作为评判一个策略π好坏的准则

$$J^{\pi}(s)=\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t)|s_0=s,\ a_t=\pi_t(s_t),$$

其中γ是衰减因子, 满足$0<\gamma<1$. 对于有些系统, 系统运行过程中会遇到终止状态. 在这些终止状态系统将不再进行状态转移, 意味着当前运行的结束. 这种情况的衰减收益定义为

$$J^{\pi}(s)=\sum_{t=0}^{T-1}\gamma^t r_t+V(s_{\rm T})|s_0=s,\ a_t=\pi_t(s_t),$$

其中: $s_{\rm T}$代表终止状态, $V(s_{\rm T})$是一个预定义的值, 代表在状态$s_{\rm T}$下的奖惩. 带终止状态的性能函数定义可以看作是无终止状态的一种特例, 因而在下文中只考虑无终止状态的情况.

最优控制的目标是使收益最大, 也就是找到最优性能函数:

$$V^*(s)=\max_{\pi}J^{\pi}(s).$$

对应的策略称为最优控制策略:

$$\pi^*=\arg\max_{\pi}J^{\pi}.$$

但是最优的标准有时太过苛刻, 反而求解近似最优也能得到相近的控制效果. 对于策略π, 如果在$\forall s\in S$下都有$J^{\pi}(s)\geqslant V^*(s)-\varepsilon$, 那么本文称之为$\varepsilon$近似最优.

为了方便算法的运行选择Q函数作为性能函数. 那么最优控制便是求解下列等式关于最优Q函数的解:

$$Q^*(s,a)=r(s,a)+\gamma\max_{a'}Q^*(s',a'),$$

其中$s'=f(s,a)$. 而最优控制策略则由下式计算得到:

$$\pi^*(s)=\arg\max_{a}Q^*(s,a).$$

由于假设奖励函数限定在$[r_{\min},r_{\max}]$范围内, 所以任意策略的衰减收益都有相同的下界和上界, 分别是$V_{\min}=\dfrac{r_{\min}}{1-\gamma}$和$V_{\max}=\dfrac{r_{\max}}{1-\gamma}$. 此外任意两个策略的收益差的绝对值不会超过$V_{\rm b}=\dfrac{1}{1-\gamma}(r_{\max}-r_{\min})$.

如果一个算法在线学习过程中, 最多只需要有限的运行时间即可学到近似最优的策略, 那么这个算法就称为近似最优在线RL算法. 一个算法要称为近似

最优在线RL算法需要满足PAC原理. 目前已有多种PAC标准, 这里选择较为严格的非最优控制策略时刻总和(policy-mistake count, PMC)准则.

定义 1(PMC)　在一个算法在线学习过程中, s_t代表系统t时刻的状态, $A_t=\{\pi_k\}_{k=t}^{\infty}$代表算法在$t$时刻执行的策略, A_t在t时刻的衰减收益等于$J^{A_t}(s_t)=\sum\limits_{k=t}^{\infty}\gamma^{(k-t)}r(s_k,a_k)|a_k=\pi_k(s_k)$. 那么算法非最优控制策略时刻总和定义为

$$\mathrm{PMC}(\varepsilon)=\sum_{t=0}^{\infty}\mathbb{I}\{J^{A_t}(s_t)<V(s_t)-\varepsilon\},$$

其中$\mathbb{I}\{\cdot\}$是一个激活函数. 当括号内的事件发生时输出1, 否则输出0.

3　均匀子集多样本算法(Multisample in each cell (MEC) alglorithm)

3.1　状态离散化(State discretization)

由于状态变量是连续的, 需要有效的手段近似连续性能函数和控制策略. 为此首先引入状态离散化方法将连续状态空间划分成相邻的子集[16]. 不同子集相互之间是非重叠的. 假设一共划分成N_{grid}个子集, 每个子集用$C_i(1\leqslant i\leqslant N_{\mathrm{grid}})$表示. $\Omega(C_i)$代表C_i包含的状态区域.

在给出本文的算法之前, 首先定义关于系统连续性的假设. 用$d:S\times S\to\mathbb{R}$表示任意两个状态之间的距离.

假设 1(连续性[12])　对$\forall s_1,s_2\in S$和$\forall a\in A$, 存在两个常数α和β满足

$$|r(s_1,a)-r(s_2,a)|\leqslant\alpha d(s_1,s_2),$$
$$d(f(s_1,a),f(s_2,a))\leqslant\beta d(s_1,s_2),$$

α和β分别称为奖励函数和状态转移函数的连续常数.

与文献[12]类似, 在假设1的基础上可以推导出关于最优Q函数连续性的引理.

引理 1[14]　对$\forall s_1,s_2\in S$和$\forall a\in A$都有

$$|Q^*(s_1,a)-Q^*(s_2,a)|\leqslant\bar{\omega}(d(s_1,s_2)).$$

$\bar{\omega}$定义为

1) 如果$\gamma\beta<1$, $\bar{\omega}(z)=\dfrac{\alpha}{1-\gamma\beta}z$.

2) 如果$\gamma\beta>1$, $\bar{\omega}(z)=cz^{\log_\beta(1/\gamma)}$, 其中

$$c=2\beta\left(\frac{\alpha}{\gamma\beta-1}\right)^{\log_\beta(1/\gamma)}V_b^{\log_\beta(\gamma\beta)}.$$

与文献[12]不同的是上面的引理是关于$Q^*(s,a)$的连续性, 而文献给出的是关于$V^*(s)$的连续性引理.

根据假设1和引理1, 在同一个动作下相邻的两个状态会有相似的奖励、状态转移、和最优性能函数. 为了定量描述同一个子集中状态的相近性, 本文给出下面关于分辨率的定义.

定义 2(分辨率)　分辨率δ定义为在整个状态空间, 同一个子集中任意两个状态之间的最大距离, 表示为

$$\delta=\max_{i=1,\cdots,N_{\mathrm{grid}}}\ \max_{s_1,s_2\in\Omega(C_i)}d(s_1,s_2).$$

根据δ的定义和假设 1, 对$\forall s_1,s_2\in\Omega(C_i)$, $1\leqslant i\leqslant N_{\mathrm{grid}}$, 都有

$$|r(s_1,a)-r(s_2,a)|\leqslant\alpha\delta,\tag{1}$$
$$d(f(s_1,a),f(s_2,a))\leqslant\beta\delta,\tag{2}$$

α, β和$\bar{\omega}$是关于系统模型的参数和函数, 通常情况下都是未知的. 下面会详细介绍提出的均匀子集多样本算法 (multisample in each cell, MEC) 算法. 需要注意的是在算法运行过程中, 这些参数和函数是不需要的.

3.2　数据集(Data set)

在当前时刻t下, 假设已有数据集$D_t=\{(\hat{s}_k,a_k,\hat{r}_k,\hat{s}'_k)\}_{0\leqslant k\leqslant t-1}$, 包含过去时刻的部分观测量. 其中: $\hat{r}_k=r(\hat{s}_k,a_k)$, $\hat{s}'_k=f(\hat{s}_k,a_k)$. 对任意$(\hat{s}_k,a_k,\hat{r}_k,\hat{s}'_k)\in D_t$ [1], $\hat{s}_k$一定是属于状态离散化中的一个子集. 假设C_i是包含$\hat{s}_k$的子集, 并且C_i有可能会存储多个数据(multisamples). 为了表示方便, 使用符号$D_t(C_i,a)$代表D_t中在动作a下属于C_i的数据集. 如果C_i没有存储a对应的数据, 那么称$D_t(C_i,a)$为空集, 表示成$D_t(C_i,a)=\varnothing$.

给定一个子集C_i, 如果$D_t(C_i,a)\neq\varnothing$, 意味着至少存在一个数据$(\hat{s},a)\in D_t(C_i,a)$, 满足$\hat{s}\in\Omega(C_i)$. 在$C_i$范围内的任意$s\in\Omega(C_i)$, 根据分辨率定义都有$d(s,\hat{s})\leqslant\delta$. 但是, $D_t(C_i,a)$中的数据即使相互近邻只有δ的最大距离, 它们在同一动作的下一时刻状态很有可能会互相远离, 转移到不同子集中. 图2给出了一个示例. 3个数据$\hat{s}_1,\hat{s}_2,\hat{s}_3$是在一个子集, 但是它们在相同动作下的转移状态被分布到不同子集中. 然而根据式(2), 这些转移状态之间的最大距离不会超过$\beta\delta$. 本文定义直径为$\beta\delta$的区域最多可以覆盖$N_{\beta\delta}$个子集. 因而对同一子集的所有数据, 在相同动作下它们的转移状态最多分散到$N_{\beta\delta}$个子集中.

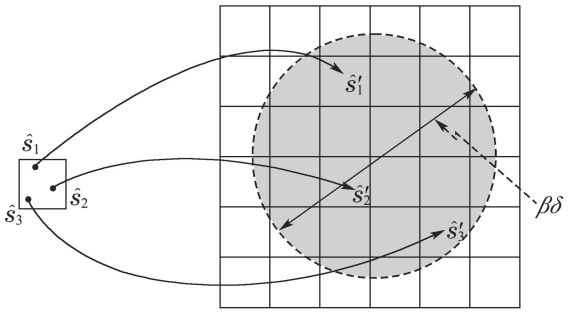


图 2　示例: 同一子集中不同数据的转移状态分布在不同的子集中

Fig. 2　Example of samples in a same cell falling into different cells for their next states

[1] $(\hat{s}_k,a_k,\hat{r}_k,\hat{s}'_k)$也可简写成$(\hat{s}_k,a_k)$, 在下文笔者会混合使用这两种表述方式.

3.3 近似上界Q迭代(Near-upper Q iteration)

现在, 利用存储的数据集定义一个从函数到函数的映射关系, 称为近似上界Q迭代(near-upper Q iteration, NUQI)算子.

定义 3(NUQI) 给定一个函数$g\colon S\times A\to\mathbb{R}$, 对任意$s$假定$s\in\Omega(C_i)$. NUQI算子$\bar{\mathcal{T}}$定义为

$$\bar{\mathcal{T}}(g)(s,a)=\begin{cases}\min\limits_{(\hat{s},a,\hat{r},\hat{s}')\in D_t(C_i,a)}[\hat{r}+\gamma\max\limits_{a'}g(\hat{s}',a')],\\ \qquad\text{若}D_t(C_i,a)\neq\varnothing,\\ V_{\max},\ \text{否则}.\end{cases}\tag{3}$$

NUQI算子含义是对一组(s,a), 如果它所在的C_i中没有存储任何数据, 就将收益的上界$V_{\max}$作为这组状态动作的性能函数值. 那么最大收益会鼓励系统向未遍历过的状态区域探索. 反之如果C_i是非空的, 使用$D_t(C_i,a)$中的数据计算性能函数. 更具体地等于公式(3)等号右边的最小结果. 选取最小值的目的是为了得到更为紧致的上界.

可以证明$\bar{\mathcal{T}}$是一个收缩算子, 所以存在唯一的固定解, 称为近似上界Q函数(near-upper Q function, NUQF). 由于$\bar{\mathcal{T}}$是基于当前t时刻的D_t定义而来的, 用$\bar{Q}_t$表示$\bar{\mathcal{T}}$在D_t下的固定解, 即$\bar{Q}_t=\bar{\mathcal{T}}(\bar{Q}_t)$.

引理 2[14] 算子$\bar{\mathcal{T}}$在无穷范数上是关于γ的收缩算子.

引理 3[14] 在任意时刻t, 对$\forall s\in S$和$\forall a\in A$, $\bar{Q}_t$与最优性能函数Q^*都满足

$$\bar{Q}_t(s,a)\geqslant Q^*(s,a)-\frac{\alpha\delta+\gamma\bar{\omega}(\beta\delta)}{1-\gamma}.$$

由于$\bar{\mathcal{T}}$是收缩的, 所以值迭代(value iteration, VI)和策略迭代(policy iteration, PI)都可以用来求解$\bar{Q}_t$. 此外同一个子集C_i内的所有状态都具有相同的$\bar{Q}_t$值. 这是因为它们在根据式(3)计算时都共用相同的数据子集$D_t(C_i,a)$. 为了简化, 令$\bar{Q}_t(C_i,a)$表示所有$s\in\Omega(C_i)$在a下的结果. 因而在迭代计算过程中无需对所有状态求解$\bar{Q}_t$值, 而是以子集作为单位进行计算. 由于VI方法具有实现简便的特点, 在此以VI为例, 假定已经得到第j次迭代的结果$\bar{Q}_t^j(C_i,a)$. 那么下一次迭代则根据

$$\bar{Q}_t^{j+1}(C_i,a)=\begin{cases}\min\limits_{(\hat{s},a,\hat{r},\hat{s}')\in D_t(C_i,a)}[\hat{r}+\gamma\max\limits_{a'}\bar{Q}_t^j(\hat{s}',a')],\\ \qquad\text{如果}D_t(C_i,a)\neq\varnothing,\\ V_{\max},\ \text{否则}\end{cases}$$

进行计算. 当相邻两次迭代结果之间的误差足够小时, 笔者认为已经收敛并输出$\bar{Q}_t$.

紧接着从$\bar{Q}_t$提取出贪心策略

$$\pi_t(s)=\arg\max_a\bar{Q}_t(s,a),$$

并施加到系统上继续在线运行, 获得新的观测量.

3.4 逃脱事件(Escape event)

接下来给出一个强化学习中常见的时域时间定义. 它的含义是$T_{\varepsilon/3}$时长之后的奖励对当前收益的影响最多只有$\varepsilon/3$.

定义 4 $\varepsilon/3$–时域时间$T_{\varepsilon/3}$定义为

$$T_{\varepsilon/3}=\log_{1/\gamma}\frac{3V_b}{\varepsilon}.$$

在t时刻将贪心策略π_t施加到系统上得到新的观测量(s_t,a_t,r_t,s_t'). 根据下面这条关于已知的定义, 判断是否将新观测量加入到D_t中.

定义 5(已知) 给定一个观测量(s,a,r,s'), 假定$s\in\Omega(C_i)$. 如果$D_t(C_i,a)\neq\varnothing$并且存在某个数据$(\hat{s},a,\hat{r},\hat{s}')\in D_t(C_i,a)$, 满足$s'$和$\hat{s}'$是在同一个子集中, 就称$(s,a)$为已知. 否则$(s,a)$称为未知.

如果一个观测量是已知的, 不仅它的状态和数据集中某些数据的状态是近邻的, 而且它的转移状态也和某一个数据的转移状态相近邻. 图3出了关于已知和未知的示例. 继续以图2为例并假设已画出子集中全部数据, 即$\hat{s}_1,\hat{s}_2,\hat{s}_3$. 根据定义, 新观测量$(s_2,a)$是已知的. 这是由于$s_2'$和$\hat{s}_2'$位于同一个子集中. 另一个观测量$(s_1,a)$是未知的. 因为不存在这样的数据, 它的转移状态位于s_1'所在的子集中.

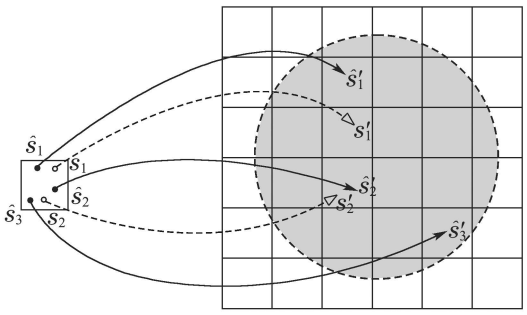


图 3 已知、未知观测量示例

Fig. 3 Examples of known and unknown observations

如果(s_t,a_t,r_t,s_t')在D_t中是已知的, 笔者就断定新的观测量并没有包含关于系统任何新的模型信息. 因此算法会忽略该观测量然后继续执行之前的策略. 否则的话(s_t,a_t,r_t,s_t')被加入到D_t中, D_t变成D_{t+1}. 并且根据D_{t+1}重新计算$\bar{Q}_{t+1}$, 然后得到新的π_{t+1}. 为了更清晰的表述上述过程, 全文用逃脱事件进行定义.

定义 6(逃脱事件) 在时刻t从状态s开始, 如果出现下面括号内的情况, 本文称在t时刻出现逃脱事件, 用$E_t(s)$表示:

$$E_t(s)=\{\text{系统在}t\text{时刻从}s\text{出发, 执行策略}A_t.\ \text{在}T_{\varepsilon/3}\ \text{时间内会遇到一个观测量}(s_\tau,a_\tau),\ \text{它对}D_t\ \text{是未知的, 其中}t\leqslant\tau\leqslant t+T_{\varepsilon/3}-1\}.$$

在t时刻如果出现逃脱事件, 就意味着在接下来的$T_{\varepsilon/3}$时间里会观测到一个未知的观测量, 算法会更新数据集, 学习新的模型信息并调整控制策略. 下面这条引理阐述了执行策略A_t的收益和近似上界Q函数$\bar{Q}_t$的差值, 与逃脱事件的关系.

引理 4[14]　给定一个误差ε并假设如下不等式成立:

$$\frac{\alpha\delta}{1-\gamma} \leqslant \frac{\varepsilon}{3},$$

那么在任意时刻都有

$$\max_a \bar{Q}_t(s_t,a) - J^{A_t}(s_t) \leqslant \frac{2}{3}\varepsilon + \mathbb{I}\{E_t(s_t)\}V_{\mathrm{b}}.$$

3.5　主要定理(Main theorem)

最后给出主要定理:

定理 1[14]　定义一个误差ε并假设如下条件成立:

$$\frac{\alpha\delta + \gamma\varpi(\beta\delta)}{1-\gamma} \leqslant \frac{\varepsilon}{3},$$

那么MEC算法在线学习过程中的非近似最优控制策略时刻是有限的, 满足

$$\mathrm{PMC}(\varepsilon) \leqslant N_{\mathrm{grid}} N_{\beta\delta}\,|A| \log_{1/\gamma} \frac{3V_{\mathrm{b}}}{\varepsilon}.$$

根据上述定理可知MEC算法是满足PAC原理的. 针对常用的在线学习方式, 近一步推出下面这条更为直观的新定理.

定理 2　对于一个被控系统, 期望找到从s_0出发的近似最优控制策略. 在线学习过程中每次都以s_0作为初始状态, 运行一段固定的时长T_{episode}后, 将状态重置为s_0然后继续下一个时段的学习. 如果使用MEC算法最多只需要$N_{\mathrm{grid}}N_{\beta\delta}|A|$个时段的在线学习就可以找到近似最优的策略. 也就是说系统最多运行$N_{\mathrm{grid}}N_{\beta\delta}|A|T_{\mathrm{episode}}$步后算法会自动停止, 然后输出一个近似最优控制策略.

从定理1可以看出, 算法最终输出策略的最优误差与子集分辨率有关. δ值越小, 状态离散化越精细, 那么输出策略越接近最优. 但是δ的减小会使子集数量N_{grid}增加, 增加数据集存储量, 延长算法运行时间. 因此分辨率的选择需要根据系统实际控制需求, 综合考虑运行效率和控制效果两方面因素.

另一个需要注意的是最终策略的近似最优性. 从定理2的结果可以看出策略只是对从初始点s_0出发运行T_{episode}步的轨迹来讲是近似最优的, 并没有考虑其他状态区域的近似最优性. 不过对在线问题而言, 主要的学习目标就是从初始点对系统进行最优控制. 因此这里简化称最终策略是近似最优的.

算法1给出了整个MEC算法的流程. MEC的含义是多个不同的数据被存储在同一个子集中. 在这些数据基础上计算近似上界Q函数并提取控制策略. 需要注意的是算法在运行过程中并不需要任何模型参数.

算法 1　MEC算法.

Input: 收益上界$V_{\max}$; 状态离散化子集$\{C_i\}$;

Output: 近似最优控制策略π_t;

初始化 $t=0$, $D_0 \leftarrow \varnothing$, $\bar{Q}_0 \leftarrow V_{\max}$ 和 $\pi_0(s) = \arg\max_a \bar{Q}_0(s,a)$;

repeat

　在系统上执行 π_t 得到 t 时刻观测量 (s_t, a_t, r_t, s_{t+1});

　if (s_t,a_t) 在D_t中是未知的 then

　　(s_t,a_t,r_t,s_{t+1}) 被加入到D_t中变成D_{t+1};

　　根据式(3)计算$\bar{Q}_{t+1}$;

　　根据式(4)计算π_{t+1};

　else

　　$\pi_{t+1} = \pi_t$;

　end if

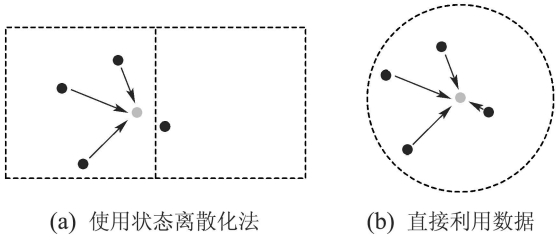
　　$t = t+1$;

until 在一个时段内D_t不再变化.

4　Kd–CPAC算法(Kd–CPAC algorithm)

在上一节提出的MEC算法使用简单直观的状态离散化方法, 存在数据利用率低的问题. 具体来讲, 算法只对同一子集中的状态认为具有相似的模型信息, 用子集中的数据近似该区域. 但是对相邻子集中的一些数据, 即使它们很靠近某些状态, 但是由于不在同一子集内, 该子集便不会利用这些数据的信息(如图4(a)所示). 这就造成了数据利用率低, 模型信息重复存储, 增加算法运行时间.

回归树方法将数据存储在树型结构, 然后用叶子结点中的数据近似目标函数[17]. 在RL中直接利用数据的好处是可以提高数据的利用率. 如图4(b)所示, 一个状态的所有邻域数据都可以用来近似该状态的模型信息. 其中: 黑色代表存储的数据, 灰色代表近似的状态. 因此对上一节提出的MEC算法进行改进. 直接使用数据构建近似最优在线强化学习算法, 解决连续状态离散动作系统的控制问题. 对任意状态使用邻域中的数据近似模型信息, 定义新的迭代算子计算性能函数和控制策略. 经过完整的理论分析证明新算法依然满足PAC原理, 因而是一个近似最优在线RL算法. 同时, 为了提高数据的存储和查找速率, 引入kd树技术存储在线观测量. 由于使用kd树并且考虑连续状态系统的PAC(continuous PAC)算法, 新算法称为kd–CPAC.



(a) 使用状态离散化法　　(b) 直接利用数据

图 4　数据利用示例

Fig. 4　Examples of data utilization

4.1 Kd树(Kd–tree)

kd树作为有效划分状态空间和存储数据的方法, 已经广泛应用在RL领域. 如Ernst等[17]研究基于kd树的回归算法, 与值迭代相结合. Munos和Moore[18]使用kd-tree(kd树的一种变形), 实现可变分辨率状态空间的离散化, 成功应用在动态规划上. 更多关于kd树的介绍可以参见文献[19].

假定将要存储的数据是$(\hat{s},\hat{a},\hat{r},\hat{s}')$, 使用$\hat{s}$作为该数据的关键字. 为了方便存储, 对每个动作都设计一个kd树, 因此共有$|A|$个kd树($|A|$代表动作个数). 在算法初始阶段, 每个树都只有一个空的根结点, 包含整个状态空间. 当有数据到达时, 根据数据的动作存储在不同的根结点中. 随着算法的运行, 结点中的数据会逐渐增加. 当一个结点中的数据到达一定上限时, 使用一个超平面在某一维度上将结点空间分割成两个子空间, 形成两个子结点. 用N_{split}代表结点分裂时数据的上限. 经过分裂后, 原结点中的N_{split}个数据被分配到两个子结点中. 当有更多数据到达时这个过程会不断持续, 树的深度也不断增加.

结点分裂时选择的维度和超平面根据如下方式决定: 计算该结点所有数据在每个维度上的方差. 为了获得均匀的存储效果, 在计算方差之前可以用状态变量的取值范围归一化每个数据. 沿方差最大的维度分裂结点. 然后选择分裂维度上数据的中值作为分割结点的超平面.

当有新的数据$(\hat{s},\hat{a},\hat{r},\hat{s}')$需要加入到kd树时, 查找动作$\hat{a}$对应树中$\hat{s}$所在的叶子结点, 然后将数据加入到该叶子结点中.

使用kd树还可以方便查找任意状态的邻域数据. 给定状态s, 它的邻域数据指的是一组满足$d(s,\hat{s})\leqslant\delta$的数据集$\{(\hat{s},\hat{a},\hat{r},\hat{s}')\}$. d代表状态的距离函数, 定义为$d: S\times S\to\mathbb{R}$. δ称为邻域距离. 从根结点开始判断是否当前结点包含的区域与s距离小于δ. 如果不是, 那么它的子结点以及包含的数据和s的距离也会超过δ, 因此不需要进一步的判断. 如果结点和s之间的距离小于δ, 并且结点有子结点, 就继续用同样的方式判断子结点, 直到到达叶子结点. 从叶子结点中选择和s距离小于δ的数据作为邻域数据.

在kd–CPAC算法中, 同样使用上一节关于连续性的假设1和引理1.

4.2 数据集(Data set)

假设当前时刻是t, 将过去时刻的观测量有选择地存储在kd树中, 构成数据集$D_t=\{(\hat{s}_k,a_k,\hat{r}_k,\hat{s}'_k\}_{0\leqslant k\leqslant t-1}$. 使用$D_t(a)$代表动作$a$对应的数据子集. 对任意状态$s$构建动作$a$下的邻域集$\mathcal{N}_t(s,a)$, 包含$D_t(a)$中的邻域数据$(\hat{s},a,\hat{r},\hat{s}')$, 满足$d(s,\hat{s})\leqslant\delta$. 因此$\mathcal{N}_t(s,a)$包含的是和$s$距离不超过$\delta$的数据, 可以用于近似$(s,a)$的模型信息. 如果邻域集中不包括任何数据, 那么就称$\mathcal{N}_t(s,a)=\varnothing$. 同时对任意$(s,a)$和它的邻域数据$(\hat{s},a,\hat{r},\hat{s}')$, 根据假设1有下列不等式关系:

$$|r(s_1,a)-r(s_2,a)|\leqslant\alpha\delta, \tag{4}$$

$$d(f(s_1,a),f(s_2,a))\leqslant\beta\delta. \tag{5}$$

4.3 基于数据的Q迭代(Data-based Q iteration)

利用存储的数据集定义基于数据的Q迭代(data-based Q iteration, DBQI)算子.

定义 7(DBQI) 给定一个函数$g\colon S\times A\to\mathbb{R}$和任意$(s,a)$, DBQI算子$\tilde{\mathcal{T}}$定义为

$$\tilde{\mathcal{T}}(g)(s,a)=\begin{cases}\min\limits_{(\hat{s},a,\hat{r},\hat{s}')\in\mathcal{N}_t(s,a)}[\hat{r}+\gamma\max\limits_{a'}g(\hat{s}',a')],\\ \qquad\text{如果}\mathcal{N}_t(s,a)\neq\varnothing,\\ V_{\max},\ \text{否则}.\end{cases} \tag{6}$$

DBQI算子的含意是对一组(s,a), 如果它的$\mathcal{N}_t$是空的, 就用性能函数的上界$V_{\max}$对(s,a)赋值. 否则使用$\mathcal{N}_t$中的邻域数据计算它的性能函数值. 更具体地是取式(6)等式右边中括号内的最小结果. 整个DBQI算子的计算过程完全只依赖数据集存储的数据.

可以证明$\tilde{\mathcal{T}}$是一个收缩算子, 因而存在一个固定解$\tilde{Q}_t$, 满足$\tilde{Q}_t=\tilde{\mathcal{T}}(\tilde{Q}_t)$. 本文称$\tilde{Q}_t$为基于数据的$Q$函数(data-based Q function, DBQF).

引理 5[15] 算子$\tilde{\mathcal{T}}$在无穷范数上是关于γ的收缩算子.

引理 6[15] 在任意时刻, 对任意$s\in S$和$a\in A$, $\tilde{Q}_t$与最优性能函数Q^*都满足

$$\tilde{Q}_t(s,a)\geqslant Q^*(s,a)-\frac{\alpha\delta+\gamma\bar{\omega}(\beta\delta)}{1-\gamma}.$$

为了计算得到$\tilde{Q}_t$, 可以使用值迭代和策略迭代方法解上面的等式. 如果使用VI, 只需要迭代计算数据集中数据的Q值, 即可得到关于整个状态空间的精确$\tilde{Q}_t$. 具体来讲, 首先初始化函数$\tilde{Q}_t^{(0)}$. 可以设定成任意一个常数, 通常选择0或$V_{\max}$. 然后对数据集每个数据$(\hat{s},a,\hat{r},\hat{s}')$计算$Q$值, 根据

$$\tilde{q}_t^{(0)}(\hat{s},a)=\hat{r}+\gamma\max_{a'}\tilde{Q}_t^{(0)}(\hat{s}',a').$$

接着, 利用如下公式计算新的$\tilde{Q}_t^{(1)}$:

$$\tilde{Q}_t^{(1)}(s,a)=\begin{cases}\min\limits_{(\hat{s},a,\hat{r},\hat{s}')\in\mathcal{N}_t(s,a)}\tilde{q}_t^{(0)}(\hat{s},a),\\ \qquad\text{如果}\mathcal{N}_t(s,a)\neq\varnothing,\\ V_{\max},\ \text{否则}.\end{cases}$$

上述过程的计算结果与根据式(6)由$\tilde{Q}_t^{(0)}$计算得到的$\tilde{Q}_t^{(1)}$是完全一致的.

重复上面的计算过程. 假设已经得到第j次迭代结果$\tilde{Q}_t^{(j)}$, 计算数据集中数据的Q值

$$\tilde{q}_t^{(j)}(\hat{s},a)=\hat{r}+\gamma\max_{a'}\tilde{Q}_t^{(j)}(\hat{s}',a'),$$

那么第$(j+1)$次迭代计算的结果$\tilde{Q}_t^{(j+1)}$就可以根据下面公式得到:

$$\tilde{Q}_t^{(j+1)}(s,a)=\begin{cases}\min\limits_{(\hat{s},a,\hat{r},\hat{s}')\in\mathcal{N}_t(s,a)}\tilde{q}_t^{(j)}(\hat{s},a),\\ \qquad\text{如果}\mathcal{N}_t(s,a)\neq\varnothing,\\ V_{\max},\ \text{否则}.\end{cases}$$

由于上述过程是使用VI求解式(6)的一种变型, 因而是收敛的并且结果是精确的. 更重要的是, 计算过程中只需要保存数据的Q值. 之后整个状态空间的$\tilde{Q}_t$函数可根据数据的Q值计算得到.

接下来由$\tilde{Q}_t$提取出贪心策略

$$\pi_t(s)=\arg\max_a\tilde{Q}_t(s,a).\tag{7}$$

将这个策略施加到系统上继续在线运行, 得到下一时刻新的观测量. 而每一时刻的π_t构成了算法实时的执行策略$A_t=\{\pi_k\}_{k=t}^{\infty}$.

4.4　逃脱事件(Escape event)

下一个需要考虑的问题: 是否将新的观测量加入到数据集D_t中. 判断的准则是数据集只存储包含新鲜模型信息的数据. 下面给出相关的定义.

定义 8(已知)　给定一个观测量(s,a,r,s'), 如果$\mathcal{N}_t(s,a)\neq\varnothing$并且存在某个数据$(\hat{s},a,\hat{r},\hat{s}')\in\mathcal{N}_t(s,a)$, 满足

$$|\max_{a_1'}\tilde{Q}_t(s',a_1')-\max_{a_2'}\tilde{Q}_t(\hat{s}',a_2')|\leqslant\varepsilon_K,$$

就称这个观测量是已知的. 否则称之为未知. 参数ε_K代表已知误差.

在定义8的基础上推出下面这条引理.

引理 7[15]　对任意s和a, $\mathcal{N}_t(s,a)$最多可以包含$\lfloor V_b/\varepsilon_K\rfloor$个数据, 用符号$N_c$表示.

笔者认定未知的观测量包含新的模型信息, 并将它们加入到数据集中. 下面给出kd-CPAC算法关于逃脱事件的定义.

定义 9($\varepsilon_{\rm H}$-时域时间)　$\varepsilon_{\rm H}$-时域时间$T_{\varepsilon_{\rm H}}$定义为

$$T_{\varepsilon_{\rm H}}=\log_{1/\gamma}\frac{V_{\rm b}}{\varepsilon_{\rm H}}.$$

定义 10(逃脱事件)　在t时刻从状态s开始, 如果出现下面括号内的情况, 本文称在t时刻出现逃脱事件, 用$E_t(s)$表示:

$$E_t(s)=\{\text{系统在}t\text{时刻从}s\text{出发, 执行策略}A_t.\ \text{在}T_{\varepsilon_{\rm H}}\text{时间内会遇到一个观测量}(s_\tau,a_\tau),\ \text{它对}D_t\text{是未知的, 其中}t\leqslant\tau\leqslant t+T_{\varepsilon_{\rm H}}-1\}.$$

每当出现逃脱事件, 数据集就会增长, 基于数据的Q函数也会被更新. 下面这条引理阐明了执行策略A_t和$\tilde{Q}_t$之间的关系.

引理 8[15]　在每一个时刻t都有

$$\max_a\tilde{Q}_t(s_t,a)-J^{A_t}(s_t)\leqslant\mathbb{I}\{E_t(s_t)\}V_{\rm b}+\frac{\alpha\delta}{1-\gamma}+\frac{\gamma}{1-\gamma}\varepsilon_K+\varepsilon_{\rm H}.$$

4.5　主要定理(Main theorem)

在给出主要定理之前, 先引入另一个对定理有用的定义.

定义 11(最大最小覆盖[20])　对一个完整的状态空间, δ覆盖指的是一组数据集, 使得对任意状态s都存在一个数据点$\hat{s}$满足$d(s,\hat{s})\leqslant\delta$. 如果一个$\delta$覆盖具有这样的属性: 去掉集合中任意一个数据点都会导致该集合不再是δ覆盖. 那么具有这样属性的最大δ覆盖就称为最大最小δ覆盖. 它的数据点个数用N_δ表示.

定理 3[15]　在kd-CPAC算法运行过程中, 它的非近似最优控制策略时刻总和满足

$$\mathrm{PMC}(\varepsilon)\leqslant N_\delta N_{\rm C}|A|\log_{1/\gamma}\frac{V_{\rm b}}{\varepsilon_{\rm H}},$$

其中

$$\varepsilon=\frac{2\alpha\delta+\gamma\bar{\omega}(\beta\delta)}{1-\gamma}+\frac{\gamma}{1-\gamma}\varepsilon_K+\varepsilon_{\rm H}.$$

根据上面这条定理可知, 算法在线学习过程中策略是非近似最优的时刻是有限的. 因此kd-CPAC算法满足PAC原理. 进一步可以推出下面这条新的结论, 约束算法学到近似最优控制策略需要的运行时间上界.

定理 4　对于一个被控系统, 要求找到从s_0出发的近似最优控制策略. 在线学习过程中每次都以s_0作为初始状态, 运行固定的时长$T_{\rm episode}$之后, 将状态重置为s_0然后继续下一个时段的学习. 使用kd-CPAC算法最多运行$N_\delta N_{\rm C}|A|$个时段即可学到近似最优的策略, 即最多需要$N_\delta N_{\rm C}|A|T_{\rm episode}$步的在线学习.

因此算法学到近似最优控制策略的运行时间上界与系统模型和kd树的参数有关. 同时, 最终策略的最优误差ε受邻域距离δ, 已知误差ε_K, 以及时域时间误差$\varepsilon_{\rm H}$的影响. 如果这些参数设定的值越小, 那么最终策略越接近最优. 但是这样会导致N_δ和$N_{\rm C}$的增大, 增加算法的运行时间. 不过在上面的定理中$N_\delta N_{\rm C}$是在最坏情况下数据集能够存储的数据个数. 在应用中, 当算法结束时实际存储的数据量要远小于$N_\delta N_{\rm C}$.

整个kd-CPAC算法的步骤显示在算法2中. 由于kd-CPAC算法能够充分利用邻域数据近似模型信息, 避免使用逼近器导致的相似数据的重复存储, 因而kd-CPAC算法要比MEC算法具有更高的数据利用率,

以及更快的学习速率.

算法 2 Kd–CPAC算法

Input: 收益上界 $V_{\max}$; $|A|$ 个kd树; 邻域距离 δ; 已知误差ε_K;

Output: 近似最优控制策略π_t;

初始化 $t=0$, $D_0 \leftarrow \varnothing$, $\tilde{Q}_0 \leftarrow V_{\max}$ 和 $\pi_0(s)=\arg\max_a \tilde{Q}_0(s,a)$;

repeat

在系统上执行 π_t 得到 t 时刻观测量 (s_t, a_t, r_t, s_{t+1});

if (s_t, a_t) 在D_t中是未知的 then

(s_t, a_t, r_t, s_{t+1}) 被加入到D_t中变成D_{t+1};

根据式(6)计算$\tilde{Q}_{t+1}$;

根据式(7)计算π_{t+1};

else

$\pi_{t+1}=\pi_t$;

end if

$t=t+1$;

until 在一个时段内D_t不再变化.

5 双连杆机械臂实验(Two-link manipulator experiments)

为了验证提出的两个算法, 本文选择双连杆机械臂系统[21]作为应用对象, 实现运动控制. 双连杆机械臂是一个在水平面上运动的机械结构, 包括两个关节和两个连杆. 其中一根连杆将两个关节连接在一起. 通过关节的转动末端连杆可以指向水平面上任意方向.

图5显示的是双连杆机械臂的示意图. 它有4个状态变量$s=[\vartheta_1, \dot{\vartheta}_1, \vartheta_2, \dot{\vartheta}_2]^{\mathrm{T}}$, 分别对应两个杆的角度和角速度. 2个动作变量是$a=[\tau_1, \tau_2]^{\mathrm{T}}$, 对应两个关节的转矩. 系统动力学方程为

$$M(\vartheta)\ddot{\vartheta}+C(\vartheta,\dot{\vartheta})\dot{\vartheta}=\tau,$$

其中: $\vartheta=[\vartheta_1,\vartheta_2]^{\mathrm{T}}$, $\tau=[\tau_1,\tau_2]^{\mathrm{T}}$. 2个角度的取值范围在$[-\pi, \pi)$ rad之间, 角速度限制在$[-2\pi, 2\pi]$ rad/s范围内. 转矩的取值范围满足$\tau_1 \in [-1.5, 1.5]$ Nm 和$\tau_2 \in [-1, 1]$ Nm. 系统采样时间选择$T_{\mathrm{s}}=0.05$ s. 此外, 动力学方程中的$M(\vartheta)$和$C(\vartheta,\dot{\vartheta})$具体形式是

$$M(\vartheta)=\begin{bmatrix} P_1+P_2+2P_3\cos\vartheta_2 & P_2+P_3\cos\vartheta_2 \\ P_2+P_3\cos\vartheta_2 & P_2 \end{bmatrix},$$

$$C(\vartheta,\dot{\vartheta})=\begin{bmatrix} b_1-P_3\dot{\vartheta}_2\sin\vartheta_2 & -P_3(\dot{\vartheta}_1+\dot{\vartheta}_2)\sin\vartheta_2 \\ P_3\dot{\vartheta}_1\sin\vartheta_2 & b_2 \end{bmatrix}.$$

相关物理量的含义和取值显示在表1中. 根据这些物理量可以计算出上式中的相关变量:

$$P_1=m_1{c_1}^2+m_2{l_1}^2+I_1,$$

$$P_2=m_2{c_2}^2+I_2,$$

$$P_3=m_2 l_1 c_2.$$

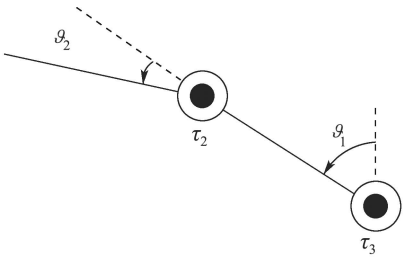


图 5 双连杆机械臂示意图

Fig. 5 The schematic of two-link manipulator system

表 1 双连杆机械臂动力学模型参数

Table 1 Parameters of the two-link manipulator

符号	值	单位	含义
$l_1; l_2$	0.4; 0.4	m	连杆长度
$m_1; m_2$	1.25; 0.8	kg	连杆重量
$I_1; I_2$	0.066; 0.043	kg·m^2	连杆惯性
$c_1; c_2$	0.2; 0.2	m	连杆质心位置
$b_1; b_2$	0.08; 0.02	kg/s	关节阻尼

控制目标是希望将机械臂从初始状态$[-\pi, 0, -\pi, 0]^{\mathrm{T}}$移动到$[0,0,0,0]^{\mathrm{T}}$. 奖励函数选择

$$r(s,a)=-s^{\mathrm{T}}Qs,$$

$$Q=\mathrm{diag}\{1, 0.05, 1, 0.05\}.$$

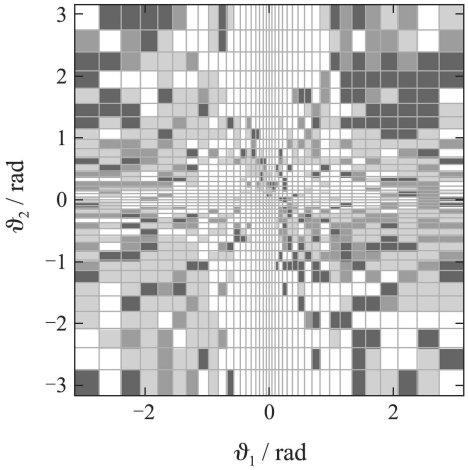
设定衰减因子γ等于0.98, 每个时段长度是6 s, 重新开始一个时段时状态都初始化为$s_0=[-\pi, 0, -\pi, 0]^{\mathrm{T}}$. 控制动作选择$a=[\tau_1,\tau_2]^{\mathrm{T}}$, $\tau_1 \in \{-1.5, 0, 1.5\}$, $\tau_2 \in \{-1, 0, 1\}$, 共9个动作.

5.1 MEC算法(MEC algorithm)

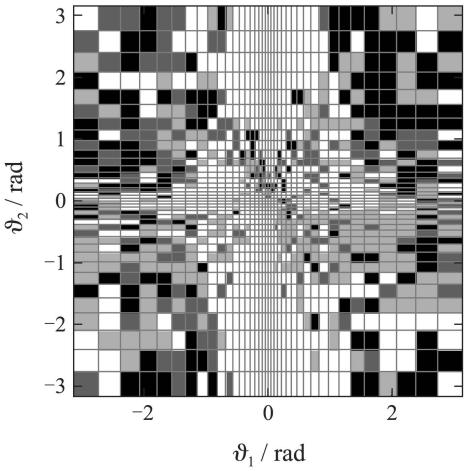
通过分析系统特性不难发现在远离原点的状态区域, 控制动作的选择要相对单一, 相应的控制目标是将机械臂向原点附近移动. 而离原点越近的状态区域则要求更高精度的控制策略. 因此为了减小计算量, 降低算法运行的难度, 在使用MEC算法时对状态空间进行对数离散化. 在离原点较远的区域设计跨度较大的子集, 而离原点越近的区域则选择越精细的子集. 并且对双连杆系统, 角度在控制精度的影响方面要比角速度大一些. 因此对两个角度变量, 在原点两侧分别设计19个对数空间的子集. 对两个角速度变量, 则在原点两侧分别设计4个对数空间的子集.

将设计好的MEC算法应用在双连杆机械臂上, 成功实现在线学习的控制目标. 图6画出了最终策略在固定$\dot{\vartheta}_1=0.001$ rad/s, $\dot{\vartheta}_2=0.001$ rad/s时, 控制动作τ_1, τ_2与角度变量ϑ_1, ϑ_2之间的关系. 其中: 不同灰度代表不同转矩的动作, 白色代表未完全探索区域. 同时也在图中画出了对数空间的状态离散化子集. 靠近

原点位置的子集具有较小的尺寸, 而离原点越远的子集尺寸越大. 同样也发现图中存在空白的区域, 在这些区域没有观测到任何的数据. 这是由于系统动力学特性, 在线运行过程中无法或很难达到这些区域, 或者这些区域的数据对策略的提升没有任何影响, 因此算法没有向它们进行探索. 因而也有效地减少了数据的存储.



(a) $\tau_1(\vartheta_1, 0.001, \vartheta_2, 0.001)$ Nm

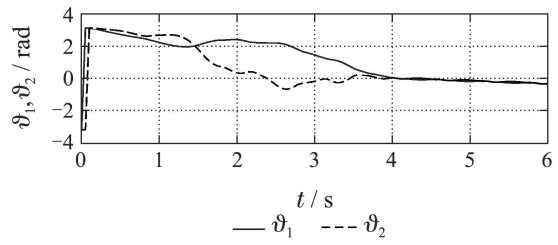


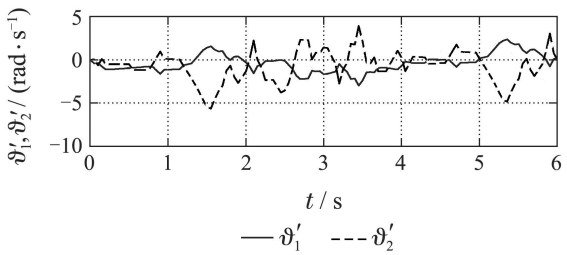
(b) $\tau_2(\vartheta_1, 0.001, \vartheta_2, 0.001)$ Nm

图 6　双连杆机械臂实验中MEC算法最终策略示意图

Fig. 6　The policy outputted by MEC algorithm

将最终策略应用到双连杆机械臂系统上, 得到图7的控制轨迹. 初始状态等于$[-\pi, 0, -\pi, 0]^{\mathrm{T}}$. 通过两个转矩动作的控制, 两个连杆在一定时间内(4 s左右)运动到了零点位置.





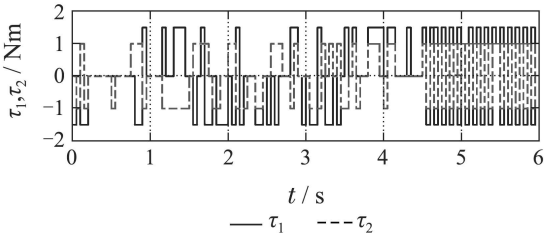


图 7　双连杆机械臂实验中MEC算法结果的控制轨迹

Fig. 7　Trajectories under the policy by MEC algorithm

5.2　Kd–CPAC算法(Kd–CPAC algorithm)

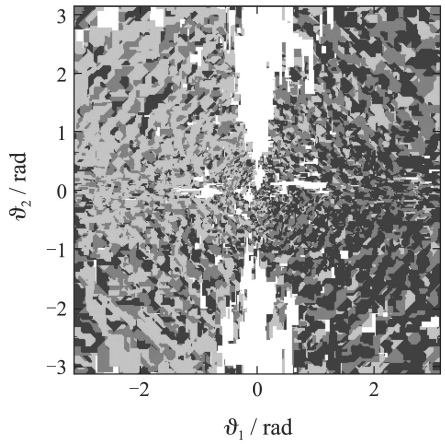
接着将kd–CPAC算法应用在双连杆机械臂上. 同样为了减小计算量, 分别对角度和角速度变量的邻域距离设计如下线性形式:

$$\delta_1(\vartheta) = \delta_3(\vartheta) = (1 - 10^{-\frac{1}{19}})(\frac{|\vartheta|}{\vartheta_{\sup}} + \frac{1}{9}),$$

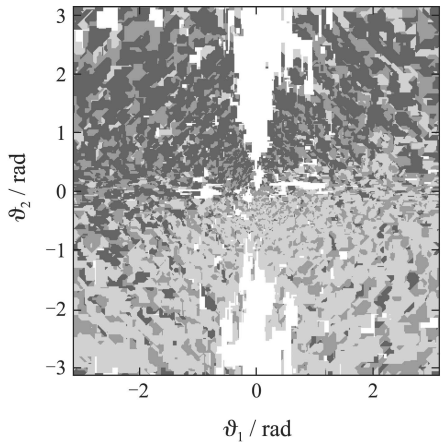
$$\delta_2(\dot{\vartheta}) = \delta_4(\dot{\vartheta}) = (1 - 10^{-\frac{1}{4}})(\frac{|\dot{\vartheta}|}{\dot{\vartheta}_{\sup}} + \frac{1}{9}).$$

离原点越近的点, 它的邻域距离越小, 并且角度的邻域距离函数比角速度有更小的取值. 这样在存储数据和计算策略时会更多地考虑角度的因素.

将kd–CPAC算法应用在双连杆机械臂系统上, 经过在线学习得到最终策略. 为了在二维空间上可视化最终策略, 固定$\dot{\vartheta}_1 = 0.001$ rad/s, $\dot{\vartheta}_2 = 0.001$ rad/s 画出控制动作τ_1, τ_2同ϑ_1, ϑ_2之间的关系图, 如图8所示. 与上面使用MEC算法得到的策略图6相比, 新算法的策略更加完整, 覆盖更加广泛的状态空间. 同时不同的动作在图8上有更好的区分. 图8中: 不同灰度代表不同转矩的动作, 白色代表未完全探索区域. 将 kd–CPAC得到的策略施加到系统上得到图9的控制轨迹. 两个连杆在不到2.5 s时就被控制到达原点位置. 而使用 MEC 算法得到的轨迹图 7 则显示两个连杆大约在4 s时移动到原点. 为了更直观的比较kd–CPAC与MEC两个算法的控制效果, 将两个控制轨迹中的奖励变化曲线共同画在图10中. 经过比较可以看出kd–CPAC算法对应的奖励曲线有更快的上升速度, 从而证明kd–CPAC算法学到的策略要比MEC算法更优. 但是由于kd树的引入, kd–CPAC算法的设计难度和计算复杂度要比MEC算法大很多. 在实际应用中应当根据问题本身的要求, 选择合适的算法进行应用.



(a) $\tau_1(\vartheta_1, 0.001,\ \vartheta_2, 0.001)$ Nm



(b) $\tau_2(\vartheta_1, 0.001,\ \vartheta_2, 0.001)$ Nm

图 8 双连杆机械臂实验中kd–CPAC算法最终策略示意图

Fig. 8 The policy outputted by kd–CPAC algorithm

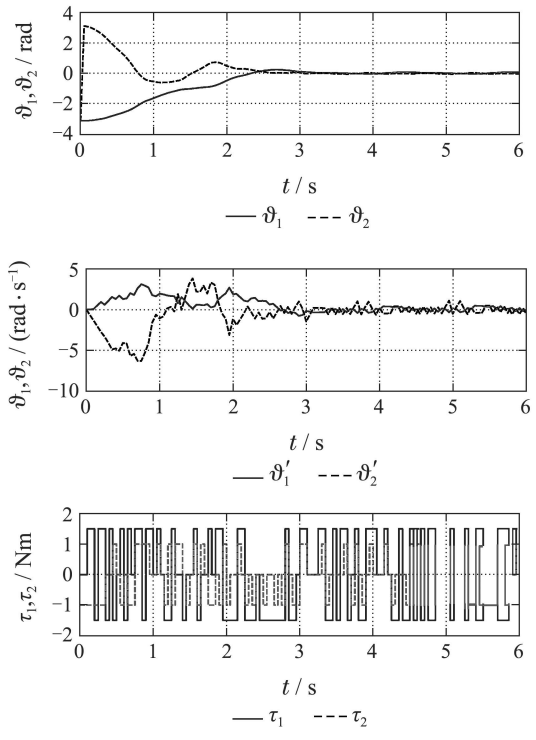


图 9 双连杆机械臂kd–CPAC算法实验结果的控制轨迹

Fig. 9 Trajectories under the policy by kd–CPAC algorithm

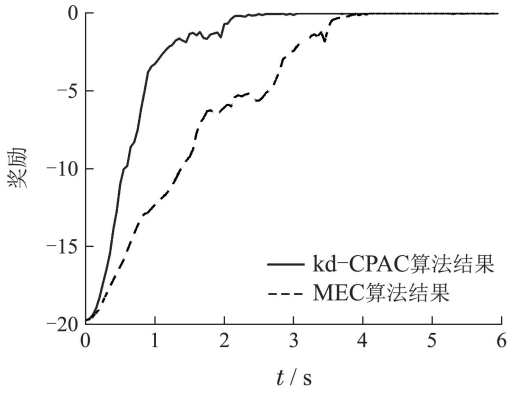


图 10 双连杆机械臂kd–CPAC和MEC算法实验结果在奖励变化曲线上的比较

Fig. 10 Comparison in rewards between the policies learned by MEC and kd–CPAC algorithms

6 结论(Conclusions)

针对连续时间系统, 本文基于概率近似正确原理提出了两种RL算法, 能够在有限时长内在线学到近似最优的策略. 基于状态离散化设计的MEC算法具有易于实现的特点, 但是数据利用率相对较低. 而使用kd树技术设计的kd–CPAC算法能够有效利用在线数据, 找到更好的策略, 同时也增加了一定的算法复杂度. 近年来随着基于图像控制问题的深入研究, 有效利用在线数据, 缩短在线学习时间成为了扩大算法应用范围的关键因素. 而笔者提出的近似最优在线RL算法, 能够为这一类问题提供重要的解决思路.

与现有的基于神经网络的强化学习算法相比, 本文提出的方法能够准确分析计算结果和理论值的误差上界, 保证了结果的近似最优性. 而神经网络虽然具有强大的通用逼近能力, 但是在理论层面无法保证结果的准确性. 但是本文方法的计算复杂度却比神经网络大很多. 神经网络在少量权重参数下即可表示复杂的函数关系, 而本文方法需要存储整个数据集. 因此如何减少数据集的存储, 提高模型动力学的表达形式成为了方法进一步提升的方向.

参考文献(References):

[1] ZHANG Rubo, GU Guochang, LIU Zhaode, et al. Reinforcement learning theory, algorithms and its application [J]. *Control Theory & Applications*, 2000, 17(5): 637 – 642.
(张汝波, 顾国昌, 刘照德, 等. 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, BARTO A. *Reinforcement Learning: An Introduction* [M]. Cambridge MA: MIT Press, 1998.

[3] GAO Yang, CHEN Shifu, LU Xin. Research on reinforcement learning technology: a review [J]. *Acta Automatica Sinica*, 2004, 30(1): 86 – 100.
(高阳, 陈世福, 陆鑫. 强化学习研究综述 [J]. 自动化学报, 2004, 30(1): 86 – 100.)

[4] ZHANG H, QIN C, LUO Y. Neural-network-based constrained optimal control scheme for discrete-time switched nonlinear system using

dual heuristic programming [J]. *IEEE Transactions on Automation Science and Engineering*, 2014, 11(3): 839 – 849.

[5] RUMMERY G A, NIRANJAN M. *On-Line Q-Learning Using Connectionist Systems* [M]. Cambridge, UK: University of Cambridge Press, 1994.

[6] WATKINS C J C H. Learning from delayed rewards [J]. *Robotics & Autonomous Systems*, 1995, 15(4): 233 – 235.

[7] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Human-level control through deep reinforcement learning [J]. *Nature*, 2015, 518(7540): 529 – 33.

[8] 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.

[9] ZHAO Dongbin, SHAO Kun, ZHU Yuanheng, et al. Review of deep reinforcement learning and discussions on the development of computer Go. [J] *Control Theory & Applications*, 2016, 33(6): 701 – 717. (赵冬斌, 邵坤, 朱圆恒, 等. 深度强化学习综述: 兼论计算机围棋的发展 [J]. 控制理论与应用, 2016, 33(6): 701 – 717.)

[10] KEARNS M, SINGH S. Near-optimal reinforcement learning in polynomial time [J]. *Machine Learning*, 2002, 49(2/3): 209 – 232.

[11] BRAFMAN R I, TENNENHOLTZ M. R-max – a general polynomial time algorithm for near-optimal reinforcement learning [J]. *Journal of Machine Learning Research*, 2003, 3(2): 213 – 231.

[12] BERNSTEIN A, SHIMKIN N. Adaptive-resolution reinforcement learning with polynomial exploration in deterministic domains [J]. *Machine Learning*, 2010, 81(3): 359 – 397.

[13] PAZIS J, PARR R. PAC optimal exploration in continuous space Markov decision processes [C] *//The 27th AAAI Conference on Artificial Intelligence*. Menlo Park, California: AAAI, 2013: 774 – 781.

[14] ZHAO D, ZHU Y. MEC–a near-optimal online reinforcement learning algorithm for continuous deterministic systems [J]. *IEEE Transactions on Neural Networks and Learning Systems*, 2014, 26(2): 346 – 356.

[15] ZHU Y, ZHAO D. A data-based online reinforcement learning algorithm satisfying probably approximately correct principle [J]. *Neural Computing and Applications*, 2015, 26(4): 775 – 787.

[16] JIANG Guofei, GAO Huiqi, WU Cangpu. Convergence of discretization procedure in Q-learning [J]. *Control Theory & Applications*, 1999, 16(2): 194 – 198.
(蒋国飞, 高慧琪, 吴沧浦. Q学习算法中网格离散化方法的收敛性分析 [J]. 控制理论与应用, 1999, 16(2): 194 – 198.)

[17] ERNST D, GEURTS P, WEHENKEL L. Tree-based batch mode reinforcement learning. [J]. *Journal of Machine Learning Research*, 2005, 6(2): 503 – 556.

[18] MUNOS R, MOORE A. Variable resolution discretization in optimal control [J]. *Machine Learning*, 1999, 49(2/3): 291 – 323.

[19] PREPARATA F P, SHAMOS M. *Computational Geometry: An Introduction* [M]. New York, USA: Springer-Verlag, 2012.

[20] KAKADE S, KEARNS M J, LANGFORD J. Exploration in metric state spaces [C] *//Proceedings of the 20th International Conference on Machine Learning (ICML–03)*. Menlo Park, California: AAAI, 2003: 306 – 312.

[21] BUSONIU L, BABUSKA R, SCHUTTER B D, et al. *Reinforcement Learning and Dynamic Programming Using Function Approximators* [M]. Boca Raton, USA: CRC Press, 2010.

作者简介:

朱圆恒　(1989–), 男, 博士, 助理研究员, 研究方向为强化学习、自适应动态规划, E-mail: yuanheng.zhu@ia.ac.cn;

赵冬斌　(1972–), 男, 博士,研究员, 研究方向为强化学习、自适应动态规划、智能交通、机器人、过程控制等, E-mail: dongbin.zhao@ia.ac.cn.