# 基于任务层次分解的多智能体决策规划

孙　清,杨马英,贾玉博

(浙江工业大学 信息工程学院,浙江 杭州　310023)

**摘要**:对于较大规模的多智能体决策规划问题,用传统的基于 Markov 决策过程的方法一般很难解决。本文研究完全分布式控制方式、存在部分感知的条件下智能体的决策规划方法,以达到多智能体协同完成总体任务的目标。基于 MAXQ 值函数分解方法对问题进行层次分解,设计利用有限感知的多智能体决策框架结构,提出了采用与或图表示可行策略路径并结合 $Q$-learning 学习算法在线策略求解的算法。在 RoboCup2D 平台上对算法的性能进行试验,结果表明该算法在保证实时性的同时,能得到较好的策略。

**关键词**:Markov 决策过程;MAXQ;任务层次分解;与或图;$Q$-learning 算法

## 0　引言

Markov 决策过程(MDP)是处理智能体决策规划问题的一个通用模型[1]。一般 MDP 问题的求解方法可以分为离线算法和在线算法。基本的离线算法有后向迭代算法和前向搜索算法[2]。这些算法的基本原理是通过遍历问题的状态空间,搜索出累积回报最高的策略来执行。随着问题的不确定性以及状态空间的增加,策略求解将变得非常困难。在线算法在交互过程中交替地进行决策计算和动作执行,且通常随着时间的推移,解的质量不断提升。实时动态规划(RTDP)算法是最早的基于动态规划的实时算法[3],该算法不处理停止问题。

基于 MAXQ 值函数分解[4]的 MDP 任务分解方法是 Thomas G. Dietterich 在 1998 年提出来的,在总结之前分层方法的同时,提出了一种有效表示分层结构值函数的方法,将 MDP 先进行简化。

目前 MDP 任务求解的主流方法之一是强化学习的方法。它的优势在于无需对所处的动态环境建模,耗时少,并且能在 Agent 与动态环境交互时在线使用;可以调用函数逼近算法(如神经网络)来表示其知识。但是基于学习的方法本身有一些缺点:第一,一般很难对所有状态给出足够的学习例子[5],基于学习的方法一般需要状态值对或状态策略对的表示,这在大规模连续状态问题中很难实现;第二,针对大规模的问题,算法的复杂度呈指数增长,一般不能被直接使用求解问题。

本文针对大规模、动态实时的分布式多智能体协同任务,结合 MAXQ 任务层次分解方法,提出采用与或图表示策略路径的 $Q$-learning 算法。利用该算法设计了一种有效部分感知的多智能体决策框架,并引入启发式搜索方法[6]的优势,只考虑当前状态下的可达状态,从而有效地降低策略求解的复杂度,得到最优分层策略。将该方法用于 RoboCup2D 球员智能体决策问题中,验证算法的效果。

## 1　多智能体协作框架设计

在多智能体应用问题中,由于传感器的限制,Agent 只有部分的感知能力,即符合部分可观察 MDP(POMDP)模型特性。因此,笔者设计了用以解决感知受限问题的多智能体决策框架。通过贝叶斯估计的方法,得到信念状态的后验概率分布,然后直接使用最大似然法,取当前概率最大的一个信念状态为 MDP 的当前状态[7]。这样,将 POMDP 转化为一个 MDP 进行求解。

假设观测向量是 $y=[y_1,y_2,\cdots,y_n]^{\mathrm{T}}$,概率密度函数为 $p(y/x)$。其中,$x=[x_1,x_2,\cdots,x_n]^{\mathrm{T}}$ 是未知参数向量。假设在得到观察数据 $y$ 之前,对状态 $x$ 有一定认识,即

$$p(y/x)p(x)=p(y,x)=p(y)p(x/y)$$

当得到观测向量后,得到的条件密度为

$$p(x/y)=\frac{p(y/x)p(x)}{p(y)}=\frac{p(y/x)p(x)}{\sum_x p(y/x)p(x)}$$

**收稿日期**:2014-10-10

**作者简介**:孙　清(1990-),男,浙江衢州人,硕士研究生,研究方向为机器人控制。



?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

注意到分母是与 $x$ 无关的，也就是说后验概率分布与先验概率和观察向量的概率密度函数成比例。$p(x)$是先验概率，$p(x/y)$是后验概率，$p(y/x)$也称为似然函数。该公式表明在得到观察样本 $y$ 时，未知参数 $x$ 的后验分布与先验分布和似然函数的乘积成比例。得到未知参数的后验概率，即信念状态的概率分布，可以直接使用最大似然法确定一个最可能的状态，作为当前环境的状态进行决策规划。该方法可以有效地消除部分可观特性，将 POMDP 转化为标准 MDP 模型进行策略求解。单个智能体决策模型如图 1 所示，表示状态，多智能体间的交互则体现在全局状态和策略的通信两方面。

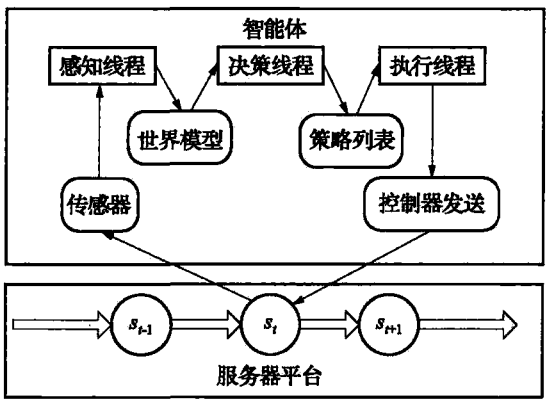


**图 1　单个智能体决策结构**

感知器模块从每个周期开始时，从服务器接收场地信息，包括视觉信息、听觉信息和身体感知信息，这些信息本身由于自身传感器的限制，存在部分可观的特性。

决策模块将 MDP 任务分解成子 MDPs 任务。在每个决策周期，通过策略生成器模块产生在该状态下可行的策略集合，存放在与或图表示的结构中。结合分层 $Q$-learning 学习算法得到最大 $Q$ 值进行动作选取。

执行模块根据已选策略第 1 步要采取的行动，规划要实现这个行动所需要采取的原子动作及其参数，如移动的加速度值、动作的力量值等。执行模块将最后的原子动作及参数发送给服务器 server，然后继续等待下一步决策的结果。

## 2　MAXQ 任务层次分解

大规模 MDP 分解成子任务有很多优势：1）子任务中好的策略可以供很多更高层的任务分享；2）子任务的值函数可以分享，有任务需要重用这个子任务时，就可以很快计算出值函数；3）使用状态抽象，总的值函数可以表示成相互独立的状态变量子集的和。

任务层次分解技术将给定的 MDP 分解成一系列按层结构组织的子 MDPs，表示为 $\{M_0, M_1, M_2, \cdots, M_n\}$，如图 2 所示。

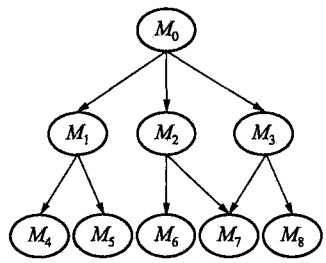


**图 2　MAXQ 任务分解**

图 2 中每个相同层次中的任务属于相同的任务层级。其中，$M_0$是根任务，解决了 $M_0$就相当于解决了原 MDP $M$ 本身。子任务 $M_i$的定义如下。

定义 1：一个无参的子任务 $M_i$定义为一个三元组$\langle T_i, A_i, \tilde{R}_i \rangle$。

$T_i$是终止判定条件，将子任务 $M_i$的状态划分成一系列活动状态 $S_i$和终止态 $T_i$。子任务只有在当前状态是 $S_i$时才能执行；当子任务进入 $T_i$，则子任务立即终止。

$A_i$可以是原 MDP 的原始动作集 $A$ 中的动作，也可以是子任务 $M_i$中用来完成该子任务的一系列动作，动作集 $A_i$表示为任务 $M_0, M_1, M_2, \cdots, M_n$的有向图。

$\tilde{R}_i$是伪报酬函数（Pseudo-reward Function），它确定从每个活动状态转移到终止状态的伪报酬，定义任务中所有到达终止态的伪报酬为 0，其他非终止态的伪报酬为负值。

MAXQ 算法本身规定子任务带有参数，不同参数都确定了一个独立的子任务。分层 MDP 的解是一个分层策略的集合，定义如下。

定义 2：一个分层策略 $\pi$ 是一个包含问题中每个子任务对应的一个相应策略的集合，即

$$\pi = \{\pi_0, \pi_1, \cdots, \pi_n\}$$

$\pi_i (i = 1, 2, \cdots, n)$是子任务从活动状态 $s_i$到动作 $A_i$的映射，即

$$\pi_i : s_i \rightarrow A_i$$

定义子任务 $M_i$ 的投影值函数为智能体在状态 $s_i$ 下执行策略，直到某个终止状态的函数，其形式等同于半马尔科夫模型下的贝尔曼方程。



?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

$$V^{\pi}(i,s) = V^{\pi}(\pi_i(s),s) + \sum_{s',N} P_i^{\pi}(s',N \mid s,\pi_i(s))\gamma^N V^{\pi}(i,s') \tag{1}$$

为得到投影值函数的层次分解，定义子任务的投影行动值函数为

$$Q^{\pi}(i,s,a) = V^{\pi}(a,s) + \sum_{s',N} P_i^{\pi}(s',N \mid s,a)\gamma^N Q^{\pi}(i,s',\pi(s')) \tag{2}$$

记上面等式右边第 2 项为完成函数，即

$$C^{\pi}(i,s,a) = \sum_{s',N} P_i^{\pi}(s',N \mid s,a)\gamma^N Q^{\pi}(i,s',\pi(s')) \tag{3}$$

可以递归地表示为

$$V^{\pi}(i,s) = \begin{cases} Q^{\pi}(i,s,\pi_i(s)) & i \text{ 是组合任务} \\ \sum_{s'} P(s' \mid s,i)R(s' \mid s,i) & i \text{ 是原始任务} \end{cases} \tag{4}$$

称式(2)，(3)，(4)为 MAXQ 分层在一个给定策略下的分解公式。一个 MDP 的分层最优策略就是在所有符合给定层次结构的策略中，使累积报酬达到最大的策略。

## 3 MAXQ 任务分层求解

首先利用 MAXQ 任务分层技术将 MDP 问题简化分解，针对其中的每个子任务，利用启发式信息的与或图进行状态动作的删减。结合 $Q$-learning 算法，通过递归计算子任务中的 $Q$ 值函数，得到总的最优值函数，算法以概率 1 收敛于递归最优策略。

### 3.1 $Q$-learning 算法

$Q$-learning 算法是一种无教师学习，获取并处理动态环境中的不完全、不确定信息，产生最佳策略，选择最优行动，从而最大限度地影响其所处的动态环境。

算法观测世界状态 $s$，选择动作 $a$，接收一个立即报酬 $r$，观测下一时刻状态 $s'$。其公式如下：

$$Q_t(s,a) = (1-\alpha_t)Q_{t-1}(s,a) + \alpha_t[r + \gamma \max_{b \in A} Q_{t-1}(s',b)] \tag{5}$$

式中，$\alpha_t$为学习率参数。

经证明，对于任意状态下的任何动作，若满足下式：

$$\lim_{T\to\infty} \sum_{t=1}^{T} \alpha_t = \infty, \lim_{T\to\infty} \sum_{t=1}^{T} \alpha_t^2 < \infty$$

则 $Q_t$ 以概率 1 收敛于最优动作值函数 $Q^*$ (Bertsekas & Tsitsiklis, 1996, Jaakkola et al. 1994)。

### 3.2 与或图策略表示

与或图是实时动态规划算法常用的策略表示方法，如图 3 所示。与或图中圆圈表示状态(或节点)，方框表示这个状态下的可选行动(与节点)。在与或图中，每条从根节点到子节点的路径即代表从当前状态到一个目标状态的策略路径，策略规划时通过评估各条路径的分层 $Q$ 函数来选择最优分层策略。

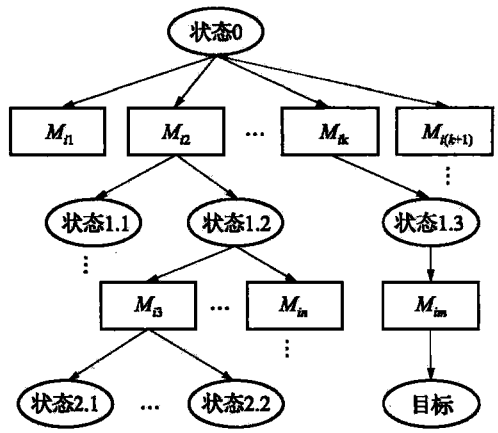


图 3 与或图示例

图 3 中每一条从根节点到叶节点的路径都是一个可行的策略执行过程。图中根节点对应的是当前的实际世界状态，其他或节点对应的状态均为预测状态，与节点代表相应状态下的可选子任务。

### 3.3 结合 $Q$-learning 的分层策略求解

对于连续状态空间问题，基于与或图表示策略路径，设计基于任务分层的 $Q$-learning 算法，主要包括两个核心模块，即策略生成器和策略评估器。

每个决策周期，策略生成器生成所有从当前状态到某个目标状态的与或图，也就表示了当前状态下所有可行的策略集合。由于生成的与或图节点可能会很多，所以首先对与或图进行裁剪。子节点对应的是不同的终止状态。对于明显不理想的终止状态对应的路径，直接删除；对于动作安全性较低的中间动作的节点及其后续节点也全部删除；对于剩余的与或图，使用策略评估模块根据每条策略路径的分层 $Q$ 值函数进行评价，最终选



?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

择最优的分层策略。在评估每个可行策略值函数时,也需要递归地评估其子任务的值,根据分层 $Q$ 值函数的大小选择最优的分层策略。显而易见,该规划算法的关键步骤是策略生成器和策略评估器。策略生成器一般根据问题域的领域知识使用一定的启发式方法设计。

在一次决策完成后,执行选定的策略。由于一个执行步骤只是执行很小的一个原子动作,在很多情况下,相邻几个决策周期中得到的最优策略是相近的。在下一次决策开始,首先判断是否可以继续执行上一步得到的策略,决定是否利用计算好的决策信息。如果可以继续执行上一步的策略,则继续执行;否则,再根据这个周期得到的状态信息,重新规划。

策略评估器即利用 $Q$-learning 学习算法进行 $Q$ 函数的求解,在每一个子任务中 $Q$-learning 算法如下进行。

Repeat:

1)初始化 $Q_0(s,a)$,$\alpha_t$;

2)观测当前时刻世界状态 $s$;

3)根据最大 $Q$ 函数选择相应行动 $a$,将动作传给 server 执行;

4)得到立即报酬 $r$,并观测下一时刻状态 $s'$;

5)修正学习率 $\alpha_t$,根据式(6)更新 $Q(s,a)$ 的值为

$$Q_t(s,a)=(1-\alpha_t)Q_{t-1}(s,a)+\alpha_t[r+\gamma\max_{b\in A}Q_{t-1}(s',b)] \quad (6)$$

6)更新世界状态;

7)返回 2),进入下一个周期。

这里的后继状态 $Q_{t-1}(s',b)$ 可以通过 server 所提供的运动模型得出下一时刻的位置,然后利用人工神经网络(HNN)算法来求取位置估值(HNN 在此不做介绍)。

对于整个 MDP 问题求解过程,需将每个子任务的 $Q$ 函数相加,求其递归和。这里需要注意一点,每一层的 $Q$ 值函数不能通过贪婪策略进行选取,因为这容易陷入局部最优或者根本找不到解的情况,本文所采取的算法最终会收敛到递归最优解。

### 3.4 MAXQ 任务分层求解

上述引出了两个问题:(1)如何判断上一个周期的策略是否可以继续执行;(2)如果继续执行上周期的策略,那么本周期的决策时间该如何安排。

对于第 1 个问题,通过分析与或图的结构发现,一个动作节点的动作完成是要通过多步的原子动作的执行实现的。也就是说,在当前的状态下,执行上一步的策略,应该以近似的概率到达这个节点动作的后继状态。基于这一点,我们通过计算在当前步执行上一周期策略到达第 1 个后继状态的概率,与上周期计算得到的状态转移概率是否相近,来决定是否可以继续执行上一步得到的策略。

决策模型对原始问题做了较大的近似,所以有必要利用尽可能多的资源来弥补这种近似带来的策略优异度损失。可以设计一个待执行的动作列表,在决策时,将决策得到的较好策略放入列表中,每次入列表采用插入排序算法,决策结束后选取最优策略并执行。基于以上分析,设计算法流程如图 4 所示。

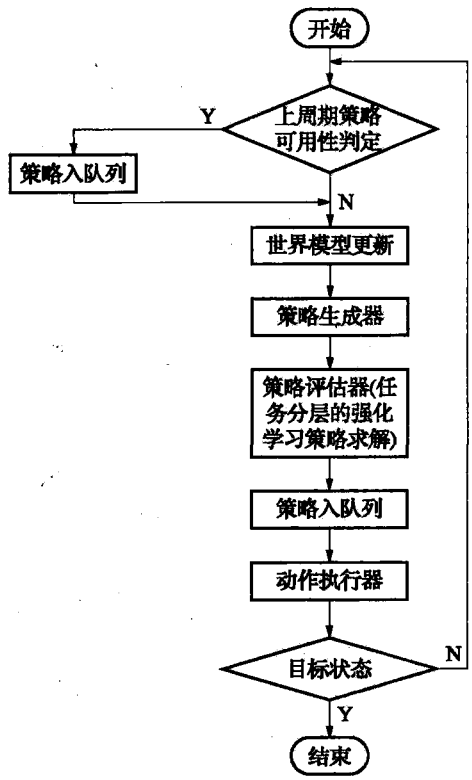


图 4 大规模 MDP 问题求解决策流程

## 4 算法在 RoboCup2D 上的实验

RoboCup2D 是 RoboCup 比赛中的一个重要项目。该平台采用 11VS11 比赛,有连续的状态空间和动作空间,平台每个周期独立地给每个球员提供部分的视觉和身体感知信息,允许球员在一场比赛中进行有限次数的通信,但是不保证通信的



?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

成功率。每个决策周期只有 100 ms,因此对决策系统的实时性有较高的要求[7]。

### 4.1 RoboCup2D 的层次分解模型

首先,对球员智能体的决策问题进行 MDP 建模。将对手和队友均看成环境的一部分,采用因子化的状态表示法表示某时刻场上 23 个对象的位置、速度和角度等信息。动作空间包含平台定义的原子动作,如 dash,kick,turn 等。因为每个动作的参数都是连续变量,动作空间是连续的。在报酬函数方面,定义每个原子动作执行的立即收益为 -0.1,以此激励智能体尽可能快地完成任务。根据 MAXQ 分层技术将 MDP 决策问题分解,如图 5 所示。

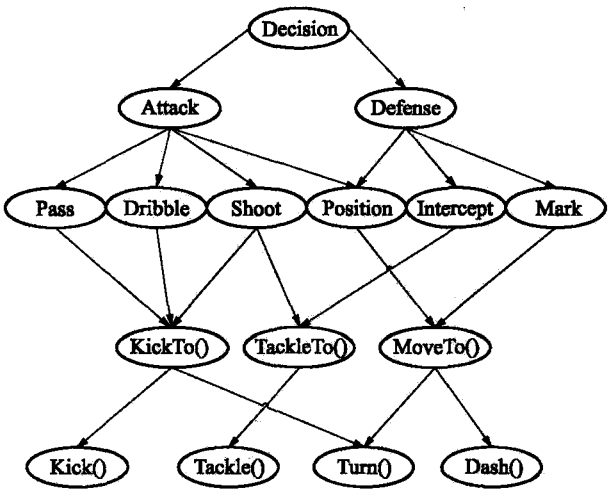


图 5 任务分解

除根任务(Decision)外,从上到下依次为团队策略层、战术行为层、技术动作层和原子动作层。

分层结构图中技术动作层和原子动作层子任务带参数,每个不同的参数对应不同的任务。另外,分层结构也隐式地引进了状态抽象。例如,MoveTo()子任务只需要考虑该智能体本身的状态信息,而无需考虑其他球员的信息;KickTo()子任务也无需考虑其他球员的状态信息。

### 4.2 实验及结果

本文的程序是基于 WrightEagleBase4.0 版本的基本框架,将 SuperMagic(本文自行设计代码,代表球队)与中国科学技术大学的原始球队 WrightEagle 进行比较,进行 10 次比赛共 60 min 后,运行所得结果如表 1 和表 2 所示。

表 1 WrightEagle VS SuperMagic 的统计结果

| 策略 | 总控球时间比 | 传球成功数/传球次数 | 得分/射门次数 |
|---|---|---|---|
| WrightEagle | 29.6% | 85/344 | 5/29 |
| SuperMagic | 70.4% | 567/679 | 61/83 |

从比赛结果统计可以看出,SuperMagic 从控球传球上均以较大优势胜出于 WrightEagle 队。其得益于任务层次分解方法,SuperMagic 对状态空间进行抽象,结合任务分层的 $Q$-learning 算法,可以快速得到较好的分层策略。

表 2 WrightEagle VS SuperMagic 性能统计

| 球队 | 最大时间/ms | 最小时间/ms | 平均时间/ms |
|---|---|---|---|
| WrightEagle | 60 | 11 | 42.5 |
| SuperMagic | 49 | 7 | 38.3 |

从单步决策时间统计表可以看出,Agent 每步决策的时间不等,并且最大单步决策时间与最小时间相差较大。这主要是由于智能体在遇到不同比赛情形时的可选动作空间相差较大,但平均起来,单步决策时间一般在 40 ms 左右,相对于 RoboCup2D 比赛 100 ms 的决策周期限制来说,决策规划的实时性完全能够满足。

### 参考文献:

[1]Stuart J. Russell ,Peter Norvig,. Artificial Intelligence: A Modem Approach [M]. [S.l.]: Pearson Education Inc. 2003.

[2]Hansen E A, S Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops[J]. Artificial Intelligence,2001(1):35-62.

[3]Barto A G., S J Bradtke,S P Singh. Learning to act using real-time dynamic programming [J]. Artificial Intelligence, 1995,(1-2): 81-138.

[4]Dietterich, T G. Hierarchical reinforcement learning with the MAXQ value function decomposition[J]. J. Artificial Intelligence Res., 2000, 13(1): 227-303.

[5]范长杰, 陈小平. 实时动态规划的最优行动判据及算法改进[J]. 软件学报, 2008. 19(11): 2 869-2 878.

[6]Akiyama H, Noda I. Multi-agent positioning mechanism in the dynamic environment[M]. [S.l.]:Springer Berlin Heidelberg, 2008:377-384.

[7]胡 凡, 基于 RoboCup 仿真平台的机器人足球协作策略的研究[D],武汉:武汉科技大学,2009.

[编辑:初秀兰]



?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net