维普资讯 http://www.cqvip.com

逻辑马尔可夫决策编程和关系马尔可夫决策编程若干最新进展*)

王蓁蓁 邢汉承 张志政 倪庆剑
(东南大学计算机科学与工程学院 南京 210096)

摘 要 逻辑马尔可夫决策过程和关系马尔可夫决策过程的引入,使得人们可能简洁地、陈述地表达复杂的马尔可夫决策过程。本文首先介绍有关逻辑马尔可夫决策过程和关系马尔可夫决策过程的概念,然后重点介绍它们与普通的马尔可夫决策过程根本不同的一些算法:①依赖于基本状态空间 RL 的转换法;②把 Bellman 方程推广到抽象状态空间的方法;③利用策略偏置空间寻求近似最优策略方法。最后对它们的研究现状进行总结及其对它们发展的一些展望。

关键词 逻辑马尔可夫决策过程,关系马尔可夫决策过程

Several New Advances of Logical Markov Decision Processes and Relational Markov Decision Processes

WANG Zhen-Zhen XING Han-Cheng ZHANG Zhi-Zheng NI Qing-Jian
(School of Computer Science & Engineering, Southeast University, Nanjing 210096)

Abstract Using logical Markov decision processes (LOMDPs) and relational Markov decision processes (RMDPs) one can compactly and declaratively represent complex Markov decision processes. This paper firstly introduces central concepts of LOMDPs and RMDPs. Then several algorithms that are different from regular Markov decision processes are reviewed: 1. The transition method relying on ground state space. 2. A relational upgrade of the Bellman update operation. 3. Approximate policy iteration using policy bias space. Finally, the paper gives conclusions of recent work and suggests future work. In this way, people are able to get an intensive, comprehensive and in-depth understanding of LOMDPs and RMDPs.

Keywords Logical Markov decision processes, Relational Markov decision processes

1 逻辑马尔可夫决策过程和关系马尔可夫决策过程基本概念

2003 年,Kristian Kersting 和 Luc De Raedt 首先提出逻辑马尔可夫决策过程,简写成 LOMDPs,它把马尔可夫决策过程(MDPs)与逻辑编程结合起来。2004 年,Martijn Van Otterlo 提出关系因子化马尔可夫决策过程,简写为 RMDPs,在抽象状态-行动空间上建立抽象关系模式 CARCASS,它作为一个表达工具,构成了 RMDPs 的骨架。

在精神实质上,LOMDPs 和 RMDPs 中的 CARCASS 是相同的,下面我们介绍的概念均取自文[1~3]。

1.1 逻辑马尔可夫决策过程 LOMDPs

一个逻辑马尔可夫决策过程(LOMDP)是一个元组$\mathbb{M}=(\Sigma,\mathbb{A},\mathbb{T}\lambda)$,其中$\Sigma$是一个逻辑字母表,$\mathbb{A}$是一个抽象行动集合,$\mathbb{T}$是一个有限的基于$\mathbb{A}$中行动上的抽象转移的集合,以及$0\leqslant\lambda<1$是一个折扣因子,并且下面等式成立:

$$\forall\mathbb{B}\in\mathbb{B},\forall a\in\mathbb{A}(\mathbb{B})\quad\sum_{\substack{T\in\mathbb{T}\\ hdy(T)=\mathbb{B}\\ act(T)=a}}P(T)=1.0\qquad(1)$$

下面给出上述定义中出现的有关术语的详细说明。Σ是一个一阶字母表,是由变量、若干关系符号 r(具有目 $n\geqslant0$)和若干函数符号 f(具有目 $n\geqslant0$)组成的集合。当 $n=0$ 时,函数符号 f 被称为常数。如果 $m=0$,则关系符号 r 被称为命题。形式 $r(t_1,t_2,\cdots,t_m)$称之为原子,其中 r 为关系符号 t_i,$i=1,2,\cdots,m$ 为项。所谓项,它或者是一个变量,或者形如 $f(t_1,t_2,\cdots,t_n)$,其中 f 是函数符号,诸 t_i,$i=1,2,\cdots,n$ 是项。原子的集合称之为合取,合取被假定是存在量化的。一个置换 $\theta=\{X_1/t_1,\cdots,X_n/t_n\}$是一个指派,即把项 t_i 指派给变量 X_i,$i=1,2,\cdots,n$。两个合取 A,B 之间,如果存在一个置换 θ,使得 $B\theta\subset A$ 成立,则称 A 是关于 θ 被包含于 B,记之为 $A\leqslant_\theta B$。一个项、一个原子或一个语句 E 称之为基本的,如果它不包含变量。我们用符号 $mgu(a,b)$表示两原子 a,b 之间最一般的合一,用符号 hb_Σ(或写为 hb^Σ)表示由Σ的 Herbrand 基,它是由Σ中的谓词和函数符号构造的所有基本原子的集合。

一个抽象状态是合取$\mathbb{Z}$,它由若干逻辑原子所组成。实质上,抽象状态的语义是它为基本状态的一个集合。我们用积木世界作例,一个抽象状态$\mathbb{Z}$,比如 $on(X,Y),bl(Y),bl(X),cl(X)$,它表达所有如此状态(在给定的字母表$\Sigma$上)的集合,该状态有两个积木块,一块在另一块之上。例如,基本状态 $Z,on(a,b),on(b,fl),bl(a),bl(b),cl(a)$,它表明积木 a 在积木 b 之上,b 在地板(fl 指称地板)上,a 上没有别的积木块(cl 表明 a 上没有其它东西),等等,该基本状态 Z 就包含在抽象状态$\mathbb{Z}$所代表的基本状态集合中。更精确地说,一个抽象状态$\mathbb{Z}$表达所有状态 Z,对于 Z,存在一个置换 θ,使得$\mathbb{Z}\theta\subseteq Z$ 成立,在上面例子中,所需的置换 $\theta=\{X/a,Y/b\}$。今后我们用 $S(\mathbb{Z})$表明抽象状态所表达的基本状态所组成的集合。

一个抽象转移 T 是一个形如下式的表达式:$\mathbb{H}\xleftarrow{p:r:a}$

*)国家自然科学基金(No. 90412014)。**王蓁蓁** 博士研究生,主要研究领域为马尔可夫决策过程、增强学习、群智能;**邢汉承** 教授、博士生导师,主要研究领域为人工智能、机器学习、模式识别;**张志政** 博士研究生,主要研究领域为知识表示和推理;**倪庆剑** 博士研究生,主要研究领域为群智能。

维普资讯 http://www.cqvip.com

$\mathbb{B}$，其中 $P(T):=p\in[0,1]$ 表示转移概率，$R(T):=r\in\mathcal{R}$ 表达该转移所获得的回报，a 是一个抽象行动。并且定义 $body(T):=\mathbb{B}$ 和 $head(T):=\mathbb{H}$，它们分别称为转移的体和头，均为抽象状态。此外，还假定 $vars(\mathbb{H})\subseteq vars(\mathbb{B})$，$vars(a)\subseteq vars(\mathbb{B})$，其中例如 $vars(\mathbb{B})$ 表示 $\mathbb{B}$ 中所有变量的集合。现在我们说明上面抽象转移的语义：如果 agent 处于状态 Z，使得 $B\theta\subseteq Z$，当 agent 执行行动 $a\theta$ 时，则 agent 将以概率 p 走到状态 $Z':=[Z\backslash B\theta]\cup H\theta$，并且获得数量为 r 的回报。仍以积木世界为例，考虑下面抽象转移 T：

$$on(X,fl),cl(X),cl(Y)\xleftarrow{0.9:-1:move(X,fl)}on(X,Y),\ cl(X)$$

把它应用到状态 Z：$on(a,b),on(b,fl),on(c,fl),bl(a),bl(c),cl(a),cl(c)$，该抽象转移告诉我们，当我们执行行动 $move(a,fl)$ 后（意指把 a 移到地板上），则后继状态变为：$on(a,fl),on(b,fl),on(c,fl),cl(a),cl(b),cl(c),bl(a),bl(b),bl(c)$，该具体转移的概率为 0.9，所收到的回报为 -1。

$\mathbb{T}$ 为所有上述形式的抽象转移的集合。令 $\mathbb{B}$ 为所有抽象转移的体所组成的集合，令 $\mathbb{A}$ 为所有抽象行动的集合，对任意的 $\forall\mathbb{B}\in\mathbb{B}$，令 $\mathbb{A}(\mathbb{B})$ 表示所有如此抽象行动 a 的集合，该行动 a 使得某个转移 $\mathbb{H}\xleftarrow{p:r:a}\mathbb{B}$ 存在。很自然地，在 LOMDP 中，我们要求定义中的等式(1)成立。

最后，为了解决抽象转移的矛盾冲突，文[1]采用类似 Prolog 简单易行方法，首先在 $\mathbb{T}$ 中按行动-体组，对所有转移建立一个全序关系 $\prec$，并按前向搜索在行动-体组中找到第一个匹配的体。

1.2 关系因子化马尔可夫决策过程 RMDPs

一个关系因子化马尔可夫决策过程（Relationally factored MDP，RMDP）M 是一个元组 $\langle P,A,D,T,R\rangle$，其中 D 是一个（有限）对象领域，P 是一个（有限）谓词模板的集合，A 是一个（有限）行动模板的集合。状态空间 $S=S(M)$，是所有在 P 和 D 上的一阶解释组成的集合的一个单一（unique）子集。行动集合 $A'=hb^{(AUD)}$。给定 S 和 A'，转移函数 T 和回报函数 R 按通常马尔可夫决策过程 MDPs 定义，即 $\langle S,A',T,R\rangle$ 构成一个基本 MDP。

在 LOMDPs 中，限制函数符号的目 $n=0$，则 LOMDPs 的定义和 RMDPs 的定义在实质上是一致的。直观上，一个解释由若干个基本关系（即 $hb^{(PUD)}$ 的元素）所组成，于是所有解释组成的集合即为 $hb^{(PUD)}$ 上的幂集，而 $S=S(M)$ 是该幂集的子集。因此，一阶解释就是相应于 MDPs 中的状态，而下面将要提到的抽象状态（即合取）它是 $S=S(M)$ 中若干状态（即解释）的集合，这与 LOMDPs 中的抽象状态的定义完全一致。

值得注意的是，一个 RMDP 的状态空间 S 的尺寸可能非常大，仅仅对于并不太多的关系谓词。以积木世界为例，如果有 10 个积木块，则 S 中的状态数目便达到可观的量级，约为 5900 万。反过来看，这就是引入 RMDP 的真正动机：由简洁的 RMDP 的抽象类型来聚类物理状态和物理行动，即把抽象类型看作“类似性”分类工具。上述论断对于 LOMDP 也是成立的。

与 LOMDP 不同，Van Otterlo 还引入背景知识 BK 概念，它为若干 Horn 子句 $H\leftarrow B$ 的集合，其中 B 是一个合取或者是 $\emptyset$，H 是原子，它称之为头（Head），并要求 $vars(H)\subseteq vars(B)$。于是抽象状态 $\tilde{S}$ 的定义是，它为 P,D 和 BK 中的 *head* 上的合取，而抽象行动 $\tilde{a}$ 是 A 和 D 上的原子。并用符号 $S(\tilde{s})$ 表示由抽象状态 $\tilde{s}$ 形成的所有基本状态的一个集合（聚类），即：$S(\tilde{s})=\{s\mid s\in S(M),s\vdash_\theta\tilde{s}\}$，其中 $\vdash_\theta$ 表示蕴含，例如两个合取 $A,B,A\vdash_\theta B$ 表示合取 A 蕴含 B，如果存在一个置换 θ，并且运用 BK 作为公理从 A 能够证明（即推出）B 成立的话。$\vdash_\theta$ 相当于 LOMDPs 中的 $\leqslant_\theta$。

如果抽象状态集合 $\{\tilde{s}_1,\tilde{s}_2,\cdots,\tilde{s}_n\}$ 完全包含了状态空间 $S(M)$，文[3]也采取在诸 $\tilde{s}_i$ 中排序的方法（不妨设就是上面的序），引入 $S(M)$ 上的一个分割。即定义 $[\![\tilde{s}_1]\!]=S(\tilde{s})$，对于 $i=2,\cdots,n$，定义 $[\![\tilde{s}_i]\!]=S(S_i)\backslash(\bigcup_{j=1}^{i-1}[\![\tilde{s}_j]\!])$。

接着，抽象行动规则 $\tilde{t}$ 定义为形如 $\tilde{s}\to\tilde{a}$（或者写成 $(\tilde{s},\tilde{a})$）的规则，其中 $\tilde{s}$ 是抽象状态，$\tilde{a}$ 是抽象行动，并且 $vars(\tilde{a})\subseteq vars(\tilde{s})$。规则 $\tilde{t}$ 的语义是，如果 agent 处于某个状态 $s\in[\![\tilde{s}]\!]$，则由所选择的抽象行动 $\tilde{a}$，它将非确定性选择某个行动 $a=\tilde{a}\theta$，其中 $s\vdash_\theta\tilde{s}$，为了简单起见，该文假定选择某个 a 的概率为 $\frac{1}{n}$，n 指不同置换 θ（满足 $S\vdash_\theta\tilde{s}$ 关系）的数目。于是由某个抽象行动规则（也称为抽象转移规则）可聚集一些基本状态-行动对以组成集合：$[\![\tilde{s},\tilde{a}]\!]=\{(s,a)\mid s\in[\![\tilde{s}]\!]\wedge s\vdash_\theta\tilde{s}\wedge a=\tilde{a}\theta\}$。由此，$\forall s\in[\![\tilde{s}]\!]$，$[\![\tilde{s},\tilde{a}]\!]^s=\{a\mid(s,a)\in[\![\tilde{s},\tilde{a}]\!]\}$ 便表示相对于某个抽象转移规则在某个基本状态 s 上可以运用的所有基本行动的集合，这些行动相对于抽象层次上的应用而言，都是“类似”的，我们在抽象意义上不能区分它们。从而，由所有行动规则组成的集合称之为一个策略 $\tilde{\pi}$，自然在规则之间如同在上述抽象状态之间一样，规定一个全序。

最后，设 $M=\langle P,A,D,T,R\rangle$ 是一个 RMDP，用记号 $\tilde{C}(M)$ 表示关于 M 的一个 CARCASS（Compact Abstraction using Relational Conjunctions for Aggregation of State-action Spaces），它定义为如此一个结构 $(\widetilde{sa},\leqslant,BK)$，其中 $\widetilde{sa}$ 是一个集合 $\{(\tilde{s},\tilde{A})\}$，该集合每一个元素 $(\tilde{s},\tilde{A})$ 是由一个抽象状态 $\tilde{s}$ 连同所有在 $\tilde{s}$ 可应用的抽象行动组成的集合 $\tilde{A}$ 构成的序对，$\leqslant$ 是一个在抽象状态上的全序，BK 是 Horn 子句组成的集合。并且对所有 $(\tilde{s},\tilde{A})\in\widetilde{sa}$ 和 $\tilde{a}\in\tilde{A}$，$var(\tilde{a})\subseteq var(\tilde{s})$。今后用记号 $\tilde{S}(\tilde{C})$ 表示 $\tilde{C}(M)$ 中的抽象状态，$\forall\tilde{s}\in\tilde{S}(\tilde{C})$，它为 P 和 BK 中的 *heads* 上的一个合取。显然，CARCASS $\tilde{C}(M)$ 是 RMDP M 的骨架，它分割状态空间 $S(M)$，并形成状态－行动空间上不同的聚类，从而把“相似”的状态以及“相似”的行动分别聚集在一起，以简约的方式思考问题，特别是背景知识 BK 的应用，更能提高抽象状态的表现能力。

通过上面的叙述，不难看出 LOMDPs 和 RMDPs 在本质上是一样的，尽管它们的出发点有所不同。

2 最优抽象策略的求法——转换法

2.1 LOMDPs 中最优抽象策略的求法

文[1]中规定在 Σ 上的一个抽象策略 $\tilde{\pi}$ 为一些形如 $a\leftarrow L$ 决策规则的集合，其中 a 是一个抽象行动，L 是一个抽象状态。

抽象决策规则 $a\leftarrow\mathbb{L}$ 的语义是：如果 agent 处于状态 Z，使得 $L\theta\subseteq Z$，则 agent 将要执行行动 $a\theta$，它用 $\tilde{\pi}(Z)$ 表示。

如果抽象策略 $\tilde{\pi}$ 由多个决策规则组成，则在规则之间建立一个全序 $\prec$。令 $\mathbb{L}=\{\mathbb{L}_1,\cdots,\mathbb{L}_m\}$ 是按 $\prec$ 序排列的策略 $\tilde{\pi}$ 的体所组成的集合，称它为策略 $\tilde{\pi}$ 的抽象层次。如果它能覆盖 LOMDP 所有可能实际状态，则 $\mathbb{L}$ 能形成状态空间的一个

维普资讯 http://www.cqvip.com

分割。并用$[\mathbb{L}_1]$,…,$[\mathbb{L}_m]$表示相应的等价类集合,即$[\mathbb{L}_1]=S(\tilde{\mathbb{L}}_1)$,对于$i=2,\cdots,m$,有$[\mathbb{L}_i]=S(\mathbb{L}_i)\backslash\bigcup_{j=1}^{i-1}[\mathbb{L}_j]$。

文[1]证明了一系列定理和命题。

定理1 每一个LOMDP $\mathbb{M}=(\Sigma,\mathbb{A},\mathbb{T},\lambda)$确定一个实际的MDP $M(\mathbb{M})=(S,A,\mathbf{T},\lambda)$。

以定理1为基础,可得到如下论断:任意抽象策略$\tilde{\pi}$确定一个在基本状态空间上的非确定策略π。因此,当$\mathbb{M}$是LOMDP,而$M(\mathbb{M})$是由它产生的MDP。我们就可以定义$\forall\mathbb{L}\in\mathbb{L}$的期望回报为所有在$[\mathbb{L}]$中状态的期望回报,即

$$V_{\tilde{\pi}}(\mathbb{L})=E_{[\mathbb{L}]}[E_{\pi}\{\sum_{k=1}^{\infty}\lambda^k r_{t+k}|Z_t=Z\}] \tag{2}$$

其中r_t表示在$M(\mathbb{M})$中,遵循基本层次策略π(它由$\tilde{\pi}$产生)在时间t所获得的回报,中括号里的期望E_{π}是基本空间上系统从时间t处于状态$Z\in S$(记之为$Z_t=Z$)开始的长期累积折扣期望,中括号外的期望$E_{[\mathbb{L}]}$是关于所有$[\mathbb{L}]$中的元素而求的。

定理2 设$\mathbb{M}=(\Sigma,\mathbb{A},\mathbb{T},\lambda)$是一个LOMDP,$\tilde{\pi}$是一个抽象策略,其抽象层次$\mathbb{L}=\{\mathbb{L}_1,\cdots,\mathbb{L}_m\}$,则存在一个有限的MDP $L=(\{l_1,\cdots,l_m\},A_L,\mathbf{T}_L,\lambda)$和一个关于$L$的策略$\pi$,使得$V_{\tilde{\pi}}(\mathbb{L}_i)=V_{\pi}(l_i)$,$i=1,2,\cdots,m$。

在定理2的证明过程中,说明了马尔可夫决策过程L的具体构造。于是在原则上,对于在LOMDP中抽象策略$\tilde{\pi}$的评估可以转化到在MDP中L的策略π的评估,也就是说我们得到了抽象值函数$V_{\tilde{\pi}}$的计算方法,从而在L中最优策略的计算便可以运用到在LOMDP中关于抽象层次$\mathbb{L}$上最优策略的计算。因为它把抽象层次的最优策略问题归约到实际状态的最优策略问题,所以我们称这种方法为"转换法"。

然而在实际马尔可夫决策过程L中,其转移概率T_L的计算要牵涉到$M(\mathbb{M})$中的状态空间的分割和概率分布问题,即使计算是可能的,也是很复杂的。因此文[1]给出一个随机迭代动态编程,并结合了传统的Q学习,引入一个(模型自由的)逻辑Q学习,简称LQ学习,此方法解决上述寻求抽象最优策略的问题(见文[1]P.5)。

2.2 RMDP中抽象$\tilde{Q}$函数的求法

作为抽象状态-行动对的抽象$\tilde{Q}$函数,主要是应用CARCASS工具来学习的。其基本精神也是在抽象状态和实际状态之间通过CARCASS进行转换,并运用实际状态上Q学习方法来学习$\tilde{Q}$函数(见文[3]Algorithm 1)。

此算法及上面的LQ学习都是模型自由方法,文[3]还给出了以模型为基础的增强学习技术,该技术主要包括两个方面,首先利用$\tilde{C}(M)$分割RMDP M的实际状态空间,即它本质上定义了一个新的状态空间,即$\{[\![\tilde{s}_1]\!],\cdots,[\![\tilde{s}_n]\!]\}$空间。利用该空间,算法设置了计算器,它们在每次行动后都相应加以更新:$C_{\tilde{s}i}^{\tilde{a}}$是从状态$\tilde{s}$执行行动$\tilde{a}$后转移到状态$\tilde{t}$的次数;$C_{\tilde{s}}^{\tilde{a}}$是agent在状态$\tilde{s}$处执行行动$\tilde{a}$的次数;$R_{\tilde{s}i}^{\tilde{a}}$是agent在状态$\tilde{s}$处执行行动$\tilde{a}$后转移到状态$\tilde{t}$所收到的回报总和。于是算法学习的转移和回报模型如下:

$$\tilde{P}_{\tilde{s}i}(\tilde{a})\leftarrow\frac{C_{\tilde{s}i}^{\tilde{a}}}{C_{\tilde{s}}^{\tilde{a}}},\tilde{R}_{\tilde{s}i}(\tilde{a})\leftarrow\frac{R_{\tilde{s}i}^{\tilde{a}}}{C_{\tilde{s}i}^{\tilde{a}}} \tag{3}$$

其中$\tilde{P}_{\tilde{s}i}(\tilde{a})$是近似转移概率,而$\tilde{R}_{\tilde{s}i}(\tilde{a})$是近似(抽象)回报函数。其次,在学到的现时模型基础上,运用PS(Prioritized Sweeping)优先扫描技术,使得传统的Q学习(它只仅仅更新沿着实验路径的Q值)方法得到改进,即PS更新更多的在状态-行动序对空间中的Q值。算法设置优先队列PQ,它是以状态的V值函数的改变量$\delta(\cdot)$的大小来动态排序的,它决定该状态是否要更新,并且更新也由该次序来进行。每当一状态比如$\tilde{s}$从PQ的顶部移出,令$\delta(\tilde{s})=0$,然后对$\tilde{s}$的所有先辈(即该状态执行某行动后能导致到$\tilde{s}$),及其它们与所有可执行的行动队的Q值进行更新,相应地修改它们的V值,并计算它们V值改变的累积量$\delta(\cdot)$,如果(比如)$\delta(\tilde{t})>\varepsilon$(预先设定的正数),则根据$|\delta(\tilde{t})|$的大小排列到PQ队列中去(参考文[3]Algorithm 2)。值得注意的是,文[3]还用实例指出在抽象层次上,决策问题可能不再具有马尔可夫性。为此,作者还给出聚类RMDP $\tilde{C}(M)$仍然保留马尔可夫性的先决条件。

3 最优抽象策略的求法——推广Bellman方程到关系领域

上一节描述的有关最优策略的求法问题,实质上仍然以实际MDP中传统算法为基础,并不是"纯粹"地在抽象状态空间上寻找(抽象)值函数和(抽象)行动值函数,即V值和Q值函数。有趣的是,文[1]、[3]的作者共同在文[4]里开发了一个算法,即关系Bellman改良操作(relational Bellman update operator)REBEL,它运用约束的逻辑编程语言简洁地表达了关系领域上的马尔可夫决策过程,并在抽象层次上直接进行值迭代,根本不涉及到基础状态空间,换言之,把Bellman方程推广到关系领域,作为逻辑编程的一部分,这个框架为关系增强学习领域带来了一个新的方向。众所周知,基于Bellman最优方程引入的传统的值迭代VI(value iteration)技术要求所有状态及其值都必须清晰地表达在一个表中,但这在关系领域,由于状态数目可能非常大,甚至无限,这种技术是无法实现的。因此,推广Bellman方程到逻辑编程抽象层次上,由此得到的(抽象)值迭代VI技术,就更具有重大实践意义。

3.1 马尔可夫决策编程(Markov Decision Programs)

这里所说的马尔可夫决策编程,大部分和前面介绍的逻辑(或关系)马尔可夫决策编程一致,只有部分作了改动。我们重点介绍改动的地方,相同之处只简略地说一下,甚至不提。注意这里考虑的实例,也是积木世界,并且用小写字母表示具体的积木或实际状态,大写字母表示抽象对象或变量。

(1)一个合取A被称为θ-包含于合取B,记为$A\leqslant_{\theta}B$,是指存在一个置换θ,使得$B\theta\subseteq A$成立。两个合取项A,B的最大下界(greatest lower bound)glb是一个被A,B所包含的最一般的合取项。即如果$C\leqslant_{\theta}A$,$C\leqslant_{\theta}B$,并且对于任意一个满足下式的D而言:$D\leqslant_{\theta'}A$,$D\leqslant_{\theta'}B$,皆存在一个置换θ'使得:$\theta\theta''=\theta'$,且$D\leqslant_{\theta''}C$,则称C为A,B的最大下界glb,写之为glb$(A,B)=C$。注意包含概念和glb概念也可以在子句上定义。

(2)一个抽象行动是一个由有限个行动规则组成的集合:$H_i\overset{p_i:A}{\leftarrow}B$,其中$A$是一个原子,它既是行动的名字,也是该行动的一个组成部分。B是一个抽象状态,它指示执行A的先决条件。H_i是A的第i个可能结果,其概率为p_i,并要求$\Sigma_i p_i=1$。抽象行动相当于LOMDPs中抽象行动和转移规则的合并概念,相当于RMDPs中的抽象行动规则。

(3)抽象回报模型R指定agent进入抽象状态时所获得的回报。在文[4]里,作者们把它作为初始抽象状态值函数V_0。

(4)而一个抽象状态值函数V是有限个形如$c\leftarrow B$值规

维普资讯 http://www.cqvip.com

则的列表，其中 B 是抽象状态，而 $c\in\mathcal{R}$ 为实数。

对于任意抽象状态 Z，值函数 $V(Z)$ 定义为所有匹配 Z 的值规则 c←B 中最大的值 c，而值规则匹配的意思是指 $Z\leqslant_\theta B$ 成立。考虑 $V_0(=\mathcal{R})$，例如它为

$10.0\leftarrow on(a,b), 0.0\leftarrow true$

注意这里是把 $on(a,b)$ 作为目标状态，并在最后值规则中运用 *true*，它保证了所有状态都能指定值。

(5)一个抽象状态-行动值函数 Q 是有限个形如 $c:A\leftarrow B$ 所谓 Q-规则的集合，其中 B 是一个抽象状态，A 是一个抽象行动，而 $c\in\mathcal{R}$。

对于任意的抽象状态-行动"队"B 和 A，Q 指定包含 $A\leftarrow B$ 的所有抽象状态行动规则的最大值 c 为其值。

在 RL，阶段(episodic) 任务将要用一个特殊的吸收状态编码。文[4]用一个人为确定的行动，例如 $on(a,b)\xleftarrow{1:0:\text{absorbing}}on(a,b)$ 来做到这一点，它表明所有状态如果它被 $on(a,b)$ 所包含，则它仅仅转移到自身，并收取零回报。

(6)一个完全约束(integrity constraints)及其解决方法。一个完全约束是由领域知识所强加的。我们用集合 C 表示完全约束的集合，而每一个完全约束是一个 Horn 子句。例如在积木世界里，下面两个 Horn 子句 $false\leftarrow on(X,Y)$，$cl(Y)$ 以及 $X\neq Y\leftarrow on(X,Y)$ 分别表明两个完全约束，即如果一个积木的顶部有另一个积木，则它就不是"清空的"和一个积木不可能在自己的上面。

于是一个抽象状态 Z 的完备(completion)描述是指关于 $C\cup\{Z\}$ 的最小不动点(least fix-point)，即由从 $C\cup\{Z\}$ 推断出的所有事实。例如 $on(a,b)$ 并没有指出 a 不同于 b，运用上述规则，这个状态可以完备为 $on(a,b), a\neq b$。特别地，如果"完备"中包含 *false*，它表明状态不满足约束，因此它是一个不合法的状态。为了处理完全约束问题，我们还必须修改有关行动定义和一般化的概念。例如行动定义现在应该约束，使得它们不能导致不合法状态。文[4]把完全约束作为背景理论，并且同样可以证明，这里介绍的马尔可夫决策编程可以产生一个(可能无限)MDP。

3.2 关系值迭代：REBEL

对于任意一个马尔可夫决策编程，给定一个抽象回报模型 R，则把它作为初始抽象状态值函数 V_0，然后相继计算下一个抽象状态值函数 $V_t, t=1,2,\cdots$。

其中主要思想是把传统的 Bellman 方程回溯运算提升到抽象层次，即引入关系 Bellman 回溯运算 REBEL。每次迭代包括以下 3 个步骤：

1^0 从 V_t 回归所有"前象"(即前提)状态。

2^0 在回归的状态上计算 Q_{t+1} 值。

3^0 用最大化方法计算 V_{t+1}。

下面依次讨论每一步。

(1)回归算法

首先给出一个重要定义：如果 S 是一个抽象状态，当遵循某个行动规则 $H_i\xleftarrow{p_i:A}B$ 后能导致到抽象状态 S'，则称它为 S' 关于行动 A 的第 i 个结果的一个最弱前置条件状态，用记号 $wp_i(A,S')$ 表示所有 S' 关于 A 的第 i 个结果的最弱前置条件状态集合。文[4]给出 WEAKESTPRE 程序，它返回关于行动规则 $H_i\xleftarrow{p_i:A}B$ 和抽象状态 S'(给定了一个完全约束集合 C)的最弱前置条件状态集合 wp_i。下面我们举例说明该算法。

例如，考虑行动 *move*：

$$on(X,Y),cl(X),cl(Z),\ X\neq Y,Y\neq Z,X\neq Z \xleftarrow{0.9:move(X,Y,Z)} cl(X),cl(Y),on(X,Z),\ X\neq Y,Y\neq Z,X\neq Z$$

$$on(X,Z),cl(X),cl(Y),\ X\neq Y,Y\neq Z,X\neq Z \xleftarrow{0.1:move(X,Y,Z)} cl(X),cl(Y),on(X,Z),\ X\neq Y,Y\neq Z,X\neq Z$$

它有两个结果：第一个结果是该行动转移 X 到 Y 上，其概率为 0.9，而行动失效为第二个结果概率为 0.1，即它并没有改变状态的概率为 0.1。令 $S\equiv(cl(a),cl(b),on(a,c),on(b,d))$(不等式关系约束省略了)，$S'\equiv on(a,b)$。显然 $S\in wp_1(move(X,Y,Z),S')$，但是 $S\notin wp_2(move(X,Y,Z),S')$。

为了计算(例如)$wp_1(move(X,Y,Z),S')$，我们能够假定从 S 可以移到 S'。因此，1^0. 行动规则的条件必须在 S 状态下能够满足；2^0. S' 是部分由该行动的第一个结果导致的。第 1^0 点是明显的，为了阐明第 2^0 点，现在考虑 $S'=on(a,b)$，可以有两种情况：其一，移动导致 $on(a,b)$。例如当我们处于状态 $S_1\equiv(cl(a),cl(b),on(a,Z),a\neq b,a\neq Z,b\neq Z)$ 时，*move* $X=a$ 到 $Y=b$ 上。其二，移动并未导致 $on(a,b)$，我们移动 X 到 Y 上但不是 a 到 b 上。例如当我们处于状态 $T\equiv(cl(X),cl(Y),on(X,Z),on(a,b),X\neq Y,X\neq Z,Y\neq Z)$，它满足我们并未移动 a 到 b 上的条件，换言之，我们行动并未涉及 $on(a,b)$，即 $on(X,Y)\neq on(a,b)$ 以及 $on(X,Z)\neq on(a,b)$。只是因为 T 中原来就包含 $on(a,b)$，而约束条件又保证了当我们在 T 中应用 $move(X,Y,Z)$ 时又保留了 $on(a,b)$。于是在这种情况下，前置条件状态的定义可以简化为：$S_2\equiv(T\wedge X\neq a)$，$S_3\equiv(T\wedge X\neq a\wedge Z\neq b)$，$S_4\equiv(T\wedge Y\neq b\wedge X\neq a)$ 以及 $S_5\equiv(T\wedge Y\neq b\wedge Z\neq b)$，而所有 $S_i(i=2,3,4,5)$ 都可以完备成相同的状态，即 $S_6\equiv cl(A),cl(B),on(a,b),on(A,C)$，其中所有变量和常量都不相同。从而抽象状态 S_1,S_6 在一起逻辑地定义了 $wp_1(move(X,Y,Z),S')=(S_1\vee S_6)$。

直到现在，我们考虑的只是仅有一个"效果"即 $S'=(on(a,b))$ 的情况，对于一般情况，可能多种组合"效果"，也可以类似处理。

有了最弱前置条件的概念，我们可以大体描述关系 Bellman 后向传播运算，假设 V_t 是目前得到的抽象状态值函数，比如说 V_0，并且考虑抽象行动 *move*。那么作为一次 Bellman 反向传播，所有抽象状态 S 当执行行动 *move*，它能导致 V_0 的一个条件，都应该被计算。但在计算 V_{t+1} 之前，我们要计算 Q_{t+1}，即进入第二个步骤。

(2)计算抽象状态行动值函数

给定上述回归的抽象状态和目前抽象状态值函数 V_t，文[4]给出程序 2 QRULES，它是一个计算抽象状态-行动值函数 Q_{t+1} 的算法。

当给定了回报模型 R，目前值函数 V_t 和折扣因子 γ，该算法返回一个行动 A 的 Q-规则。为了计算一个行动的 Q-规则，必须：1^0. 把一个行动 A 的每一个结果都看成是单一行动，并计算相应的抽象状态-行动值函数。2^0. 对于行动 A，我们结合所有结果的值为一个抽象状态-行动值函数。下面举例说明这两步，在说明时，直到下一段之前，我们都省略掉约束条件的陈述。

关于第 1^0 步，考虑前面关于 *move* 第一个结果的例子，这时 $wp_1(move(X,Y,Z),S')\equiv S_1\vee S_6$。回忆回报函数模型(即 V_0)，S_6 是吸收的，所以执行行动 *move*，我们指定该抽象状态-行动值为 10，即 $10:move(X,Y,Z)\leftarrow S_6$。至于 S_1，有关

维普资讯 http://www.cqvip.com

计算必须依赖于 $V_t(S')$，即我们例子中的 $V_0(S')$。现假设折扣因子是 0.9，这导致 $R(S)+p_1 \cdot 0.9 \cdot V_0(S')=0+0.9\times 0.9\times 10=8.1$，即 8.1：$move(a,b,Z)\leftarrow S_1$。对于 V_0 中所有其他规则都做了相同事情以后，我们得到：

〈a〉10：$move(X,Y,Z)\leftarrow cl(X),cl(Y),on(a,b),on(X,Z)$

〈b〉8.1：$move(a,b,Z)\leftarrow cl(a),cl(b),on(a,Z)$

〈c〉0.0：$move(X,Y,Z)\leftarrow cl(X),cl(Y),on(X,Z)$

因为首先考虑的是关于行动 *move* 第一个结果，那么由 *move* 第一个结果所产生的〈a〉，〈b〉，〈c〉就作为一个“暂时结果”储存，对于第一个结果无须执行 2^0 步。考虑 *move* 行动的第二结果，同样由 1^0 步得到：

〈d〉1.0：$move(a,X,b)\leftarrow cl(a),cl(X),on(a,b)$

〈e〉1.0：$move(X,Y,Z)\leftarrow cl(X),cl(Y),on(a,b),on(X,Z)$

〈f〉0.0：$move(X,Y,Z)\leftarrow cl(X),cl(Y),on(X,Z)$

现在我们执行第 2^0 步。关于第 2^0 步，我们注意到这些规则的每一个都描述如此情景，即我们如果处于该状态，那么我们可能得到某些值，它是关于行动 A 的第 i 结果方面的。因此这些信息就必须结合起来，成为关于 A 的抽象状态-行动的值。为了做到这一点，我们从〈a〉－〈c〉里选择一个规则，比如说〈b〉，并且从〈d〉－〈f〉里选择一个规则，比如说〈f〉，然后检查我们是否能同时处于那两个状态，以及是否能够运用相同的行动。换言之，我们计算这些逻辑子句（在两个值规则里）的最大下界（glb）。如果 glb（其中行动必须合一）存在，并且它是合法状态，则它就作为新规则储存，但该新规则的值是结合规则的值的和。回到〈b〉和〈f〉，这导致 8.1：$move(a,b,Z)\leftarrow cl(a),cl(b),on(a,Z)$。作为对比，〈b〉和〈d〉就不能结合给出新规则。

在上述积木世界例子里，当 QRULES 运用到 V_0 和 *move*，*absorbing* 两行动，则产生下述抽象状态-行动值函数：

〈1〉10：$absorbing\leftarrow on(a,b)$

〈2〉10：$move(X,Y,Z)\leftarrow cl(X),cl(Y),on(a,b),on(X,Z)$

〈3〉8.1：$move(a,b,X)\leftarrow cl(a),cl(b),on(a,X)$

〈4〉0.0：$move(X,Y,Z)\leftarrow cl(X),cl(Y),on(X,Z)$

（3）计算抽象值函数

由 QRULES 程序得到的 Q-规则（Q-rules）集合使得我们能够去计算下一步抽象状态值函数 V_{t+1}。和传统情况相比，表示抽象状态-行动值的 Q-rules 可能重叠，例如上面 Q-rules 里的〈1〉和〈2〉。为了计算抽象状态值函数，我们根据传统做法，利用下面事实 $V_{t+1}(S)=\max_A Q_{t+1}(S,A)$。鉴于此，文[4]运用一个简单的分而治之（Separate-and-Conquer）规则学习方法（见文[4]程序 3VRULES）。如果对于所有行动都从 V_t 计算了 Q-rules，则通过程序 3VRULES 返回值函数 V_{t+1}。

程序首先在 Q-rules 中搜索到具有最大 Q-值的 Q-rule m，然后分离出那些覆盖的 Q-rules，再递归处理剩下来的 Q-rules，靠选择更多的规则直到没有 Q-rules 保留。注意最主要的地方是，仅仅当在 Qrules 里没有其他如此 Q-rule 留下，该 Q-rule 于 Q-rule m 有相同的值，且它的体（body）包含了 m 的体，这时才选择 m 并把它加到 V_{t+1} 中去。

在上面运行的例子，我们从规则〈1〉开始，因为它没有被任意其他具有相同值的规则所包含，所以我们增加 $10\leftarrow on(a,b)$ 到 V_1 中去。而且由于它包含了规则〈2〉，我们从 Q-rules 中移去〈2〉，于是剩下来有最高值的规则是〈3〉，再迭代下去。最后，通过完备化，这样产生了新的值函数（再一次列出约束）：

$10\leftarrow on(a,b),a\neq b$

$8.1\leftarrow cl(a),cl(b),on(a,X),a\neq b,a\neq X,b\neq X$

$0\leftarrow cl(X),cl(Y),on(X,Z),X\neq Y,X\neq Z,Y\neq Z$

总结：上面 3 个步骤组成的算法称之为 REBEL 算法（Relational Bellman Backup Operator），它把 Bellman 方程推广到抽象关系领域，从而可以“纯粹”地脱离实际状态空间独立寻找抽象状态空间的最优策略，这是该算法的可贵之处。

4 最优策略的求法——运用策略语言偏置的近似抽象策略迭代

精确的策略迭代算法对于大型状态空间（关系规划领域往往会出现这种情况）是难以驾驭的，因此 AI 大都采用近似策略迭代 API（Approximately policy iteration）算法。既然设置 LOMDPs 和 RMDPs 的初衷是为了解决由于实际状态空间的庞大而造成的寻找最优策略的困难，把 Bellman 方程推广到关系领域就是这种努力的一种表现，所以 LOMDPs 和 RMDPs 也遵循近似策略算法，尽可能寻找最优抽象策略。然而由于以前的 API 方法是基于近似值函数的学习，而值函数的表达和学习在关系领域是十分困难的，文[5]可贵之处是它提出的算法，避开了这个困难，直接在偏置（用一种表达策略的语言而得到）的策略空间里搜寻最好策略，从而为抽象 API 提供一个范例。

文[5]认为，简单地说一个策略就是一个（可能是随机的）分类器，它把状态映射到行动。于是策略选择就可以当作标准的分类器学习问题。1999 年，Khardon 研究这类学习设置问题并证明了类似 PAC（Probably Approximately Correct）可学习性结果，他证明在一定假设下，一个小数目的轨迹对于学习一个接近目标的策略是充分的。另外，在关系规划领域（1999 年 Khardon，2000 年 Martin&Geffner；2002 年 Yoon 等）证明了运用表达语言确定状态-行动映射，好的策略能够从好的策略的抽样轨迹中学习到。这些结果激励文[5]产生他们算法的基本思想：给定一个策略 π，如果人们能够产生改进策略的轨迹，那么基于这些轨迹，人们就能学习到近似改进的策略。用他们的算法，主程序 API 是从一个初始策略 π_0 出发，产生一系列的近似改进策略。每次迭代涉及两个主要步骤（子程序）：第一步（子程序 Improved-Trajectories）：产生改良策略的轨迹。给定一个基础策略 π，假设 $\pi'=\mathcal{PI}^{\pi}$ 是对 n 运用一步策略迭代所得的改进结果，其中 $\mathcal{PI}^{\pi}(s)=\arg\max_{a\in A}Q^{\pi}(s,a)$，而 $Q^{\pi}(s,a)$ 是熟知的 Q-值函数。程序运用称为策略“转出”（Policy rollout）的模拟技术计算 π' 的近似 $\hat{\pi}$。从而，对于给定的状态 s，策略转出程序并不需要 π' 在所有其他状态上的值，而计算出 $\hat{\pi}(S)$，因此它提供了在一个大型状态空间 MDPs 中近似模拟改进策略 π' 的一个可以驾驭的方式。

Policy rollout 根据等式 $\mathcal{PI}^{\pi}(s)=\arg\max_{a\in A}Q^{\pi}(s,a)$ 靠估计 $Q^{\pi}(s,a)$ 在每个行动 a 的值然后取其中具有最大值的行为作为 $\hat{\pi}(S)$ 的“值”，它是 $\pi'(S)$ 的估计。而每一个 $Q^{\pi}(s,a)$ 是这样估计的：引出 w 个长度 h 的轨迹，每个轨迹都是从 s 处采取行动 a 开始，然后遵循策略 π 选择行动进行 h-1 步。沿着上述每一个轨迹计算累积的折扣回报，并求 w 个值的平均值，该平均值就是 $Q^{\pi}(s,a)$ 的估计，记为 $\hat{Q}(s,a)$。

程序 Improved-Trajectories 就是运用 Policy rollout 产生

维普资讯 http://www.cqvip.com

n 个长度为 h 的关于 $\hat{\pi}$ 的轨迹，每一个轨迹开始的状态皆是随机抽取的。程序并不仅仅记录沿着每一轨迹的状态和行动，还为策略学习算法储存一些附加信息。特别地，一个轨迹的第 i 个元素具有形式 $\langle s_i, \pi(s_i), Q(S_i, a_1), \cdots, \hat{Q}(s_i, a_m)\rangle$，它表示沿着轨迹的第 i 个元素是 s_i，运用当前(即未改进)的策略在 s_i 处选择的行动为 $\pi(S_i)$ 以及对于每个行动的 Q 值估计 $\hat{Q}(s_i, a)$。注意给定了在 s_i 处 Q-值信息，学习算法就能决定近似改进的行动 $\hat{\pi}(S)$，自然是取有最大 Q-值的行动。

第二步(子程序 Learn-Policy)：学习策略 Learn Policy。直觉上，我们想 Learn-Policy 选择一个新的策略，它能与训练轨迹紧密匹配。文[5]用策略表达语言指定一个策略偏置空间，在该空间里，基于贪婪搜索(greedy search)开发了一个相对简单的学习算法。下面结合关系规划问题详细介绍策略语言学习偏置以及与其相联的学习算法。

4.1 规划领域看作为关系因子化 MDPs

一个经典的规划领域描述一组具有相关结构的问题实例，其中一个问题实例给定了一个初始世界事实和目标条件。例如积木世界是一个经典的规划领域，其中每个问题实例都指定一个初始积木构形和一组目标适定条件。经典的规划者们都试图去寻找该领域里指定的问题实例的解，而文[5]的目标是解决全部规划领域，即寻找一个策略，使得它能运用到所有问题事例。为此，文[5]看待一个经典规划领域为关系 MDP，其中每个 MDP 状态相应于一个问题实例。下面给出较详细说明。

(1)状态和行动空间

每一个经典规划领域都指定一个行动类型集合 Y，世界谓词 W，以及可能的实际对象 O。Y 和 O 在一起定义了 MDP 的行动空间。其中一个行动是指一个行动类型运用到适当数目的对象，例如 $putdown(a)$ 是积木世界的一个行动，于是所有如此行动构造的集合称为行动空间 A。MDP 的每一个状态相应于该规划领域里的一个问题实例(即一个世界事实和目标)，它是由指定的当前世界和目标组成的。实际上，令 $P = W \cup G$ 为关系 MDP 的谓词集合，其中 G 是目标谓词的集合，对每一个在 W 的世界谓词(例如世界谓词 $clear$)，前面加上一个“g”($gclear$)便成为目标谓词，它包含于 G 中。有了这个 P 的定义，一个事实是指 P 中谓词运用到适当数目的对象，例如 $on(a,b)$ 是积木世界的一个事实。于是 MDP 的状态是目标和世界事实的集合，它指定一个问题实例的实际世界事实和目标的适定条件。而所有的状态组成状态空间 S。特别要注意的是，MDP 的行动仅仅改变世界事实而不改变目标事实，因此，这个大型关系泛化 MDP 就能看作为互不连接的子-MDPs(sub-MDPs)的集合，每一个 sub-MDP 相应于一个区别的目标适定条件。

(2)回报函数 R

给定一个 MDP 状态，我们的目标是达到 MDP 的另一个状态，在那里目标事实是相应的世界事实的子集，即达到一个世界状态，它满足目标。今后称如此状态为 MDP 的目标状态。为了描述(迅速地)到达目标状态这个目的，我们定义 R 如下：如果行动采取在目标状态，则回报为零；否则(即行动采取在所有其他状态)，回报为负值，它们分别表示采取各自行动的代价。在规划领域，一般的行动代价均设为 -1，但文[5]允许在行动中有各种代价。

(3)转移函数

每个经典规划领域都提供一个行动模拟器(action simulator)，即给定了世界状态和行动，它返回一个新世界状态。现在定义 MDP 的转移函数 T：T 为一个模拟器，它终止于目标状态并且在一个 MDP 状态里，它保留所有目标谓词而不改变它们。为了简单起见，文[5]讨论的关系 MDP 的定义并没有清晰表达行动的先决条件(而这是经典规划领域为定义合法行动而必须指定的)，但是假定算法必定进入先决条件，因此仅仅考虑合法行动，对于不合法行动，T 处理为 no-ops。

(4)初始状态分布

最后，初始状态分布 I 可能是任意程序 Program，它能产生规划领域的合法问题实例(MDP 状态)。这样我们便得到了一个关系 MDP $M = \langle S, A, T, R, I\rangle$，它相应于一个规划领域。显然，从一个初始状态(它由分布 I 引出)开始相继执行一系列具有低代价的行动而达到目标状态的策略就是一个好的策略。注意这里的策略是一个从问题实例到行动的映射，并且该映射敏感于目标适定条件。以这种方式，文[5]所学习到的策略就具有在各种不同的目标下泛化的能力。下面是文[5]为如此一般化策略而选定的语言。

4.2 分类决策列表策略(Taxonomic Decision List Policies)

(1)分类句法(Taxonomic Syntax)

分类句法提供了一个语言，该语言能描述类表达式。类表达式是具有某些感兴趣的属性的对象的集合，并用来作为构造策略的基础片段。类表达式是由 MDP 的谓词和变量构成的，其中，谓词可能包含比较谓词，比较谓词是从世界和目标谓词推演出来的(例如对每一个世界谓词 p 及其相应的目标谓词 gp，于是定义新的比较谓词为 p 和 gp 的合取，记之为 cp，这就是说，一个比较谓词指定的事实为真，当且仅当相应的世界和目标谓词指定的事实为真。例如 $con(a,b)$ 为真，当且仅当 $on(a,b)$ 和 $gon(a,b)$ 皆为真)。在文[5]的策略表达里，变量是被用作表示行动的项的，并在运行时用具体对象实例化。为了简化，文[5]仅考虑目为 1 和 2 的谓词，并分别称它们为原始类(Primitive classes)和原始关系。若当领域包含目为 3 或 3 个以上的谓词时，文[5]自动转换它们为多层辅助的二目谓词(multiple auxiliary binary predicates)。给定变量列表 $X = (x_1, \cdots, x_k)$，类表达式给定如下：

$$C[X] ::= C_0 \mid x_i \mid \text{a-thing} \mid \neg C[X] \mid (RC[X]) \mid (\min R)$$

$$R ::= R_0 \mid R^{-1} \mid R^*$$

其中 $C[X]$ 是类表达式，R 是一个关系表达式，C_0 是一个原始类，R_0 是一个原始关系，x_i 是 X 中的一个变量。现在用 $d(C[X])$ 表示类表达式 $C[X]$ 的深度，它的定义如下：如果 $C[X]$ 是一个原始类，或是 a－thing，或是一个变量，或者是 $(\min R)$，则 $d(C[X]) = 1$，其他情况，我们定义 $d(\neg C[X])$ 和 $d(R, C[X])$ 为 $d(C[X]) + 1$，其中 R 是关系表达式而 $C[X]$ 是类表达式。对于给定的关系 MDP，我们用记号 $C_d[X]$ 表示所有深度不超过 d 的类表达式 $C[X]$。

直觉上，类表达式 $(R, C[X])$ 表示这些对象的集合，该对象通过关系 R 相关于某个在集合 $C[X]$ 里的对象。类表达式 $(R^* C[X])$ 表示这些对象的集合，该对象通过某个“R chain”(R 链)相关于某个在集合 $C[X]$ 里的对象，这个概念对于表达递归概念(recursive concepts)很重要(例如，所有在 a 上面的积木块)。类表达式 $(\min R)$ 表示这些对象的集合，该对象在关系 R 下是最小的(假设对象已经排序)。

更形式地，令 s 是 MDP 的一个状态，令 $O = (o_1, o_2, \cdots, o_k)$ 是变量的一个指派，它指派对象 o_i 到变量 x_i。于是 $C[X]$

维普资讯 http://www.cqvip.com

相关于 s 和 O 的解释是一个对象集合，记之为 $C[X]^{s,O}$。因此，一个原始类 C_0 的解释为如此对象的集合，该对象使得谓词符号 C_0 在 s 状态下为真。类似地，原始关系 R_0 的解释是所有对象元组的集合，该对象元组使得关系 R_0 在 s 状态下为真。类表达式 a－thing 就表示为在状态 s 下所有对象的集合。类表达式 x_i（x_i 是一个变量）的解释就是一个单元素的集合$\{O_i\}$。至于混合(compound)表达式，它们的解释如下：

$(\neg C[X])^{s,O}=\{o\mid o\notin C[X]^{s,O}\}$

$(RC[X])^{s,O}=\{o\mid \exists o'\in C[X]^{s,O}, \text{s. t. } (o',o)\in R^{s,O}\}$

$(\min R)^{s,O}=\{o\mid \exists o', \text{s. t. } (o,o')\in R^{s,O}, \neg\exists o', \text{s. t. } (o', o)\in R^{s,O}\}$

$(R^*)^{s,O}=\mathrm{ID}\cup\{(o_1,o_v)\mid \exists o_2,\cdots,o_{v-1}\ s.\ t.\ (o_i,o_{i+1})\in R^{s,O} \text{ for } 1\leqslant i<v\}$

$(R^{-1})^{s,O}=\{(o,o')\mid (o',o)\in R^{s,O}\}$

其中 $C[X]$是类表达式，R 是关系表达式，ID 是恒等关系(identity relation)。例如在积木世界里，若给定原始类 *clear*，*gclear*，*holding*，*con-table*，沿着原始关系 *on*，*gon*，*con*，可以给出一个积木世界概念的例子：

• $(on^*(on\ gclear))$ 它的深度为 3。它表示这样的木块，当前位于我们想要清空(clear)的木块之上。

(2)决策列表策略 (Decision List Policies)

文[5]用行动-选择规则组成的决策列表来表示策略。每个规则具有形式，$a(x_1,\cdots,x_k):L_1,L_2,\cdots,L_m$，其中 a 是一个具有 k-项的行动类型，而每一个 L_i 是一个文字，每一个 x_i 是行动-项的变量。用 $X=(x_1,\cdots,x_k)$表示行动项变量的列表。每一个文字具有形式 $x\in C[X]$，其中 $C[X]$是一个(分类句法)类表达式，而 x 是一个行动-项变量。

给定一个 MDP 状态 s 和一个行动-项对象 $O=(o_1,\cdots,o_k)$的列表，我们称文字 $x_i\in C[X]$在给定了 s 和 O 下为真当且仅当 $o_i\in C[X]^{s,O}$。我们称一个规则 $R=a(x_1,\cdots,x_k):L_1,L_2,\cdots,L_m$ 允许行动 $a(o_1,\cdots,o_k)$在 s 状态下执行，当且仅当在规则 R 里的每个文字对于给定的 s 和 O 为真。注意，如果在一个规则里对于行动类型 a 没有文字，则该规则允许类型 a 的所有可能行动。一个规则可以看作在一个行动类型能够运用到一组对象里设置了相互约束关系。注意一个单一规则可能不允许有行动或者有许多同一类型的行动。如果给了如此规则的一组决策列表，我们称一个行动是这个列表所允许的，是指如果它被列表里某个规则所允许而在该规则的前面没有规则允许任意行动。同样，一个决策列表可能不允许有行动或者一个类型的多种行动。在一个 MDP 里，一个决策列表 L 定义该 MDP 的一个确定性策略 $\pi[L]$。在一个关系 MDP 里，可以在行动类型和对象上排序，并按字典序对行动排序。如果在状态 s，L 不允许有行动，则 $\pi[L](s)$执行在状态 s 下最小的行动；否则，$\pi[L](s)$执行被 L 允许的最小行动。注意 $\pi[L]$仅考虑合法行动，虽然规则并不必要编码行动的先决条件，但我们认为每个规则蕴含该规则行动类型的先决条件。

例如考虑一个简单的积木世界领域，其中目标适定条件总是把所有红色积木块上的积木移走。在这个领域里，原始类是 *red*，*clear* 和 *holding*，并且一个原始关系是 *on*，下面是一个分类决策列表，它就是能在该领域里解决任何问题的策略：

$putdown(x_1):x_1\in holding$

$pickup(x_1):x_1\in clear, x_1\in(on^*(on\ red))$

第一个规则将导致 agent 放下他手中任意一个木块，如果他手中没有木块，则他寻找一个木块 x_1，该木块上面没有其它木块并且它在红色木块之上(由$(on^*(on\ red))$所表达)，然后拿起它。

4.3 学习分类决策列表(Learning Taxonomic Decision Lists)

给定一个关系泛化 MDP，令 $\mathcal{R}_{d,l}$是所有满足下述约束的行动-选择规则的集合，其中每个规则最多包含 l 个文字而且它的类表达式的深度最多为 d。令 $H_{d,l}$表示如此策略组成的空间，每个策略是一个决策列表，其中规则取自于 $\mathcal{R}_{d,l}$。显然 $H_{d,l}$是有限空间，运用上面提到的子程序 Learn-Policy，就可以学习 $H_{d,l}$里的策略，其中 d、l 由用户指定。

回忆程序 API 的说明，提供给 Learn-Policy 学习的训练集合，它包含 rollout policy 的轨迹。因为有 n 个长度为 h 的轨迹，所以作为 Learn-Policy 输入的训练集合 D，就包含 $n\cdot h$ 训练样例。其中每个训练样例具有形式$\langle s,\pi(S),\hat{Q}(S,a,),\cdots,\hat{Q}(s,a_m)\rangle$，它们的意思前面已表明，不再赘述。

学习分类决策列表，实质上是类似于机器学习中的学习规则列表，它的精神实质是：如果在状态 s 决策列表 L 能建议一个行动，则称 L 覆盖训练样例$\langle s,\pi(s),\hat{Q}(s,a_1)\cdots,\hat{Q}(s,a_m)\rangle$。初始，令决策列表为空列表，它不覆盖任何训练样例。在每次迭代中，我们寻找高质量的规则 R，它的性能是由当前尚未覆盖的训练样例而测量的。对于选到的规则，把它加到当前的规则列表，并把规则所覆盖的样例从训练集合移去。这个过程重复直到列表覆盖所有的训练样例。

这个分类决策列表(即规则列表)算法的成功与否完全依赖下面两个方面：

1^0 怎样学习单个规则?

2^0 怎样评价一个规则的质量?

关于后者，算法给出一个新的启发函数。关于前者，它主要利用启发函数通过柱状搜索，直到搜索终止时，返回具有最大的启发函数的值。因篇幅所限，故不赘述。详见文[5]。

历史总结及其展望 最近，AI 研究者对在 MDP 中运用更丰富、更有力的表达语言的兴趣越来越高涨。Boutilier 等于 2001 年(文[6])基于结构动态编程在关系 MDPs 中提出了第一个精确解技术。在文[6]中，他们运用(允许随机行动的)情景演算动态表达一个 MDP，并且通过一个称为决策理论回归(decision-theoretic regression)运算，精确产生最优值函数和最优策略的逻辑描述，它是 Reiter's 情景演算(situation calculus)的概率延伸，即是一阶决策理论回归。虽然情景演算语言富有表达力，然而它比较复杂。2003 年，Kristian Kersting 和 Luc De Raedt 提出逻辑马尔可夫决策过程，2004 年，Martijn Van Otterlo 提出关系马尔可夫决策过程，这些都是基于逻辑编程的系统，在某种意义上说，LOMDPs 和 RMDPs 和文[6]中的思想是一致的，但比较简单些，也是基于文[6]中的思想，Kersting，Van Otterlo & De Raedt，于 2004 年，成功地把 Bellman 方程推广到关系领域，从而在逻辑层次上精确求解抽象值函数和抽象最优策略。在近似求解和模型自由(model free)方面，Kerstian Kersting 和 Luc De Raedt 在 LOMDPs 中，Martijn Van Otterlo 在 RMDPs 中都提出了算法，但是这些算法都是基于传统的近似值函数算法，2006 年，Alan Fern，Sungwook Yoon & Robert Givan 在文[5]中运用类似于“可能学习近似正确假设”的机器学习方法，直接在偏

(下转第 23 页)

维普资讯 http://www.cqvip.com

略所用的时间在策略数目不同时进行了实验，结果如图 8 所示。

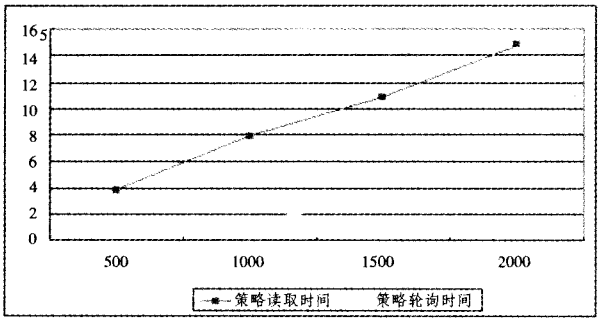

图 8　系统读取策略构造队列及轮询策略队列所用时间(单位:s)

我们的原型系统采用 Java 开发，策略存储库和系统数据库分别采用局域网内的 OpenLDAP[11] 和 Access 承当，系统运行的 PC 机配置为 PIII 800MHZ，128M SDRAM。图 8 中的结果显示系统的响应时间比较长，考虑到我们采用的策略存储库是一种开放源代码的软件，如果采用商业的目录服务器软件，读取策略的时间应该会少一些。图 8 中的轮询策略队列的时间也比较长，我们在实验原型系统中轮询策略队列所用时间时，考虑的是复杂情况下的时间，即策略队列中的每一条策略条件都会满足，每一策略都会被执行并读写系统数据库。在实际系统中，轮询策略队列时一般不会每条策略都被执行，因此轮询策略队列所用时间应该会少一些并且也可以选择其他的系统数据库来提高系统的响应速度。

结论与待研究问题　各研究组织提出的策略信息模型中，策略之间因为缺少传递信息的机制所以难以实现协作。本文分析了基于策略的安全管理系统中策略协作的需求，提出了一种采用标签的方式来传递策略之间的信息从而实现策略协作的方法。本文提出的方法经实验验证能够实现策略之间的协作，但是由于在本文提出的方法中安全管理系统采用轮询的机制来查询策略及其标签的值，所以在大规模安全管理系统中或是安全管理系统需要建立大量标签的情况下，安全管理系统有可能存在实时性的问题。在实际应用中可以采用一些优化算法来提高安全管理系统的响应速度，如可以将策略按照一定的原则保存到多个策略队列中，然后采用多线程的方式来轮询这些策略队列。如何提高安全管理系统的响应速度避免采用本文提出的方法可能带来的实时性问题是我们以后研究的一个方向。

参 考 文 献

1　Verma D C. Simplifying network administration using policy-based management. IEEE Network, 2002, 16(2): 20～26
2　Flegkas P, Trimintzios P, Pavlou G. A policy-based quality of service management system for IP DiffServ networks. IEEE Network, 2002, 16(2): 50～56
3　DMTF. Common Information Model v2. 7, www. dmtf. org/standards/cim
4　IETF RFC3060. Policy Core Information Model -- Version 1 Specification, 2001
5　IETF RFC3460. Policy Core Information Model (PCIM) Extensions, 2003
6　IETF RFC3644. Policy Quality of Service (QoS) Information Model, 2003
7　TMF. Shared Information/Data Model Addendum 1-POL Common Business Entity Definitions - Policy, August 2003
8　Kanada Y, O'Keefe B J. Rule-based Building-Block Architecture for Policy-Based Networking. Journal of Network and Systems Management, 2003, 11(3): 253～275
9　Kanada Y. Policy Division and Fusion: Examples and A Method - or, Multiple Classifiers Considered Harmful. In: 7th IFIP/IEEE International Symposium on Integrated Network Management (IM2001), 2001. 545～560
10　IETF RFC2475. An Architecture for Differentiated Services, 1998
11　IETF RFC3198. Terminology for Policy-Based Management, 2001
12　OpenLDAP homepage. http://www. OpenLDAP. org
13　IETF RFC2252. Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions, 1997
14　LDAP Browser/Editor version 2. 8. 1 http://www. iit. edu/～gawojar/ldap/index. html

(上接第 7 页)

置的策略空间中学习近似最优策略。当然，关于 LOMDPs 和 RMDPs 的文献及其最优策略的算法是较多的，但为了使我们的介绍既简单而又不流于空泛，在我们阅读许多文献后，根据自己的研究，把自己认为最主要的概念和代表不同风格和趋向的算法精选出来，加以较详细的介绍。至于其他作者的工作大都在我们所引的文献中有所涉及，故略而不述，有兴趣的读者可在我们所列的文献中找到有关他们的信息。

最后，对于 LOMDPs 和 RMDPs 今后的发展，我们简单提出几点看法。

(1)所有文献都指出抽象层次上的最优策略是有价值的，但这个论点至今尚未看到清晰的、完整的证明。我们认为，既然 LOMDPs 和 RMDPs 都基于这样一个事实，即逻辑层次上的表述能把基础层次上的状态和行动按某种类似性分类，那么我们就可以运用现代概率论中的条件数学期望概念来论证在抽象层次上的最优策略和实际层次上的具体最优策略之间的关系。我们认为，可以证明抽象层次上最优策略在平均意义上是基础层次上的最优策略。

(2)所有文献都把抽象层次和基础层次交织在一起，这为寻求抽象最优策略带来了很大麻烦。我们认为，也可以把这两个层次"完全"分离，转变为两个层次上的马尔可夫决策过程，这样在抽象层次上，利用它比实际状态数目较少的优点，较简单地运用传统方法求抽象最优策略，再按某种演算具体在实际状态空间实施，可能起到事半功倍之效。这个思路更符合人类思维特点，因为人们总是首先在大体上思考问题解决方案，然后再付诸实践，具体实现之。

参 考 文 献

1　Kersting K, De Raedt L. Logical Markov Decision Programs. In: Working Notes of the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data (SRL-03), Acapulco, Mexico, 2003. 63～70
2　Kersting K, De Raedt L. Logical Markov Decision Programs and the Convergence of Logical TD(λ). In: Proceeding of The 14th International Conference of Inductive Logic Programming, Porto, Portugal, 2004. 180～197
3　van Otterlo M. Reinforcement Learning for Relational MDPs. In: Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands. Brussels, Belgium, 2004. 138～145
4　Kersting K, van Otterlo M, De Raedt L. Bellman goes to Relational. In: Proceedings of the 21st International Conference on Machine Learning, Banff, Canada, 2004
5　Fern A, Yoon S, Givan R. Approxiamte Policy Iteration With a Policy Language Bias: Solving Relational Markov Decision Processes. Journal of Artificial Intelligence Research, 2006, 25: 75～118
6　Boutilier C, Reiter R, Price B. Symbolic Dynamic Programming for First-Order MDPs. In: Seventeenth International Joint Conference on Artificial Intelligence(IJCAI-01). Seattle, USA, 2001. 690～700