中国科学技术大学

博士学位论文

基于决策理论的多智能体系统规划问题研究

作者姓名： 吴锋

学科专业： 计算机应用技术

导师姓名： 陈小平 教授

完成时间： 二零一一年四月二十日

University of Science and Technology of China

A dissertation for doctor degree

Decision-Theoretic Planning for Multi-Agent Systems

Author :	Feng Wu
Speciality :	Computer Application and Technology
Supervisor :	Professor Xiao-Ping Chen
Finished Time :	April 20th, 2011

基于决策理论的多智能体系统规划问题研究

计算机科学与技术学院　吴锋　中国科学技术大学

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文, 是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____　　签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入《中国学位论文全文数据库》等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。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□ 公开　□ 保密 _____ 年

作者签名：_____　　导师签名：_____

签字日期：_____　　签字日期：_____

摘 要

不确定性环境下的决策和规划是人工智能的基本问题之一。决策论为这类问题的最优化求解提供了标准的理论框架。近年来，单智能体的决策理论取得了长足的发展，经典的 MDP 和 POMDP 算法已经能求解较大规模的问题。但多智能体的分布式决策却依然处在研究的初级阶段，通常只能求解极小规模的问题。作为马尔科夫决策理论在多智能体系统上的扩展，DEC-POMDP 模型涵盖了大多数的多智能体合作问题，但同时也具有极高的问题复杂度（NEXP 难）。因为在多智能体系统中，每个智能体不仅要考虑环境的变化还需要关注其他智能体的可能行为。DEC-POMDP 的复杂度具体表现在求解上就是问题具有极大的策略空间。如何对巨大的策略空间进行表示和推理并从中找出最优的策略是 DEC-POMDP 问题求解的关键。受限于问题复杂度，精确算法当前只能求解很小规模的问题。因此，本文研究的重点是为一般性的 DEC-POMDP 问题设计高效的近似算法。从求解方式上看，大体可分为在线和离线算法两类。本文在这两类算法上均有相应的工作，同时还求解了一类更具挑战的无模型规划问题。

在线规划算法在智能体与环境交互的过程中进行规划，因此只需要考虑智能体当前遇到的情况。由于每次执行过程中，智能体实际遇到的情况只是各种可能中很小的一部分，因此在大规模问题求解上，在线算法更具有优势。但在线算法本身也有需要解决的难题。因为智能体需要实时的对环境做出反应，因此每次可用于规划的时间非常的有限。在 DEC-POMDP 问题中，每个智能体获得的是各自不同的局部观察，所以需要一个分布式的计算框架来保证智能体行为之间的协调。为了与其他智能体进行合作，每个智能体必须考虑其他智能体所有可能拥有的信息，而这些信息会随时间的增加指数式的暴涨。同时由于带宽、环境和计算资源的限制，智能体之间的通讯往往是受限的。本文提出的通讯受限的多智能体在线规划算法 MAOP-COMM 较为系统的解决了这些问题。

离线规划算法在智能体与环境进行交互前，通过给定的模型计算出完整的策略。其主要优势在于有充足的时间来进行规划，而且不需要考虑在线的分布式协调，只要求计算出的策略能被每个智能体根据各自的观察分布式的执行。当前，最好的离线规划算法采用的是将动态规划和启发式搜索相结合的办法来构建一套完整的策略。对于大规模问题，其主要瓶颈在于每一步迭代都会产生极其多的子策略。这些子策略会快速的耗尽所有的存储空间和导致运算严重超时。为了解决这一问题，本文在前人工作的基础上提出了 PBPG 和 TBDP 这两个算法。PBPG 算法的主要创新点是彻底的改变了之前先枚举再选择的策略生成模式，通过构建最优化的模型为每个信念点直接生成所需的策略。TBDP 算

法主要针对的是大状态 DEC-POMDP 问题。其主要的创新点是使用基于测试的方法，只为可达的状态和需要使用到的策略计算值函数。无论是离线算法还是在线算法，在问题求解的时候都需要用到完整的 DEC-POMDP 模型。但在大规模的实际问题中，完整的 DEC-POMDP 模型并不容易获得。因此本文还提出了基于展开式采样的蒙特卡罗规划算法 DecRSPI。该算法仅利用能用于采样的环境或者仿真器就能直接计算策略，而无需事先建立完整的 DEC-POMDP 模型。

本文对多智能体规划问题研究的贡献主要有四点：(1) 较为系统的研究了多智能体在线规划问题，提出了能够保证智能体之间协调决策的 MAOP-COMM 在线规划算法。该算法使用了快速策略搜索用于满足在线规划的时间限制，同时对指数式增长的历史信息进行压缩，在使用有限内存的情况下尽可能的保留了最有价值的决策信息，最后算法还对压缩后信念与环境的一致性进行检测，并在此基础上提出了新的通讯策略，在通讯受限的情况下有效的使用了资源。实验结果验证了 MAOP-COMM 算法在多智能体在线规划上的诸多优势。(2) 较为系统的研究了多智能体离线规划的策略生成问题，提出了具有线性复杂度的信念点策略生成算法 PBPG。该算法彻底的改变了以往算法在策略生成上采用的先枚举再选择的模式，将策略生成问题建模为选择最优子映射的数值优化问题，并在此基础上提出了求解该优化问题的快速近似算法。在实验结果中，PBPG 算法在运行时间上比之前的算法提高了一个数量级，并能够保留更多的子策略，随着子策略数的增加能够对大部分的测试问题进行近似最优的求解。(3) 较为系统的研究了多智能体离线规划的策略评价问题，提出了基于测试反馈的 TBDP 算法。该算法能充分的利用问题本身具有的局部状态可达的特点，使用测试反馈的方法只针对可达状态进行策略评价，从而提高策略评价的效率。同时算法还引入了一种新的策略表示方法，在加速策略生产的同时，进一步界定需要评价的策略。从实现的角度，算法具有策略值缓存功能同时支持分布式并行策略评价，从而能够利用多处理系统的计算资源。从实验结果上，TBDP 算法可以有效的求解上万个状态的多智能体规划问题。(4) 引入并研究了多智能体系统的无模型规划问题，提出了基于蒙特卡罗方法的展开式采样策略迭代算法 DecRSPI。该算法能够只通过与环境的交换信息计算出分布式的策略，而无需事先建立问题的完整模型。同时该算法具有相对于智能体个数的线性时间和空间的算法复杂度，这使得算法能够求解智能体个数比以往算法所能求解的个数多得多的多智能体规划问题。在实验结果中，DecRSPI 算法有效的求解了超过二十个智能体的多智能体系统规划问题，比以往的算法提高了一个数量级。

关键词： 多智能体系统，马尔科夫决策模型，不确定环境下的规划，智能体间的合作与协调，分布式局部可观察马尔科夫决策过程。

ABSTRACT

Planning under uncertainty is one of the fundamental challenges in Artificial Intelligence. Decision theory offers a mathematical framework for optimizing decisions in these domains. In recent years, researchers have made rapid progresses for decision making in single-agent cases. This enables relatively large problems to be solved by state-of-the-art MDP or POMDP algorithms. However, the research of decentralized decision making is still in its infancy and existing solutions can only solve very small "toy" problems. As a nature extension of Markov decision theory to multi-agent systems, the DEC-POMDP model has very high computational complexity, namely NEXP-hard. This is not surprising given that agents in multi-agent settings not only have to keep tracking on the state transition of the environment but also the potential behaviors of the other agents. As a result, the joint policy space can be huge. Hence how to search over the large joint policy space and find the best one is a key challenge when solving a large DEC-POMDP. Due to the problem complexity, exact algorithms can solve merely tiny problems. Therefore, my thesis work focuses on developing effect algorithms for solving general DEC-POMDPs approximately. Generally, planning in multi-agent systems can work either in online or offline fashions. This thesis contributes to the literature by proposing both online and offline algorithms. Furthermore, a model-free algorithm is presented for planning when the exact model is not available.

Online algorithms do planning when interacting with the environment. During execution time, only very small regions of the joint policy space can be visited. Thus, online algorithm can scale better when solving large real-world applications. However, online algorithms have their own challenges. The time for planning is very limited because agents have to react to the environment immediately. In DEC-POMDP settings, each agent can only get its own local observations. Hence a distributed planning framework is required to guarantee the coordination among agents. In order to cooperate with others, agents must reason about all possible information held by others and this type of information grows exponentially over time. The communication resource is often limited by the band-width, environment or computation device. Therefore online algorithms must optimize the usage of communication during planning. In this thesis, a novel approach called MAOP-COMM was presented to handle these challenges properly.

Offline algorithms compute a complete plan prior to interacting with the environ-

ment. The key advantages are that there is no limit on planning time and planning can be done in a centralized manner, as long as the outcome policies can be executed by each agent according its local information. Currently, the leading offline approaches combine bottom-up dynamic programming with top-down heuristic search to construct policies. The bottleneck is that the policies trees built at each step grow exponentially and thereby may run out of time and memory very quickly. To address this challenge, this thesis improves the existing work and proposes two novel algorithms called PBPG and TBDP. The contribution of PBPG relies on constructing the best policy for each belief state directly instead of enumerating all candidates before selecting the best one. TBDP is developed to address the challenge of the large state space in DEC-POMDPs. The main contribution is to use trail-based policy evaluation for reachable states and only do the computation when necessary. The complete knowledge of a model is required by both offline and online algorithms. Unfortunately, the exact form of a DEC-POMDP may not be available. Therefore, it is significant to develop learning algorithms that can compute decentralized policies by only interacting with the environment. Motivated by the above reason, a Monte-Carlo algorithm called DecRSPI is proposed to learn policies using a set of rollout samples drew from the environment. DecRSPI is model-free and requires only a simulator or an environment that can be sampled.

The contributions of this thesis to the multi-agent planning community are mainly fourfold: (1) It systematically studied multi-agent online planning problems and proposed the MAOP-COMM algorithm, which guarantees coordination among agents. MAOP-COMM has three key components: a fast policy search method based on linear programming that meets online time-constraints, a policy-based merging solution for histories to identify the most useful information while bound the usage of memory and a new communication strategy by detecting the inconsistency of the beliefs to make a better use of the communication resource. In the experiments, MAOP-COMM performed very well in a variety of testing domains. (2) It systematically studied the policy generation problem in multi-agent offline planning and proposed the PBPG algorithm. PBPG completely replaces the backup operation and re-formulates the policy generation problem as an optimization for finding the best mapping. An approximate algorithm was also proposed to find the mapping efficiently. Consequentially, more policy trees can be kept as building blocks of the next iteration and the total solution quality is greatly improved. In the experiments, PBPG achieved an order of magnitude improvement in runtime and get near-optimal solutions when the number of sub-policies is sufficiently large. (3) It systematically studied the policy evaluation problem in multi-agent of-

fline planning and proposed the TBDP algorithm. TBDP uses trail-based evaluation for reachable states and only performs the computation when necessary. A new policy representation with layered structure and stochastic decision nodes was introduced. It formulates the policy construction as an optimization over the parameter space and further speeds up the policy generation process. Besides, TBDP can be implemented in a distributed manner and lead itself the computational power of multi-core computers. In the experiments, TBDP solved a problem with more than ten thousand states. (4) It introduced the model-free technique for multi-agent planning and proposed the DecRSPI algorithm. DecRSPI is a Monte-Carlo algorithm and requires only a simulated environment to learn a policy using rollout samples drew from the simulator. An important property of DecRSPI is its linear time and space complexity over the agent size. In the experiments, DecRSPI can solve problems up to 20 agents, an order of magnitude improvement over the agent size comparing with state-of-the-art algorithms.

Keywords: Multi-Agent Systems, Markov Decision Model, Planning Under Uncertainty, Coordination and Cooperation, Decentralized Partially-Observable Markov Decision Process (DEC-POMDP).

目 录

表格

插图

算法

主要符号对照表

MDP	马尔科夫决策过程
POMDP	局部可观察马尔科夫决策过程
MMDP	多智能体马尔科夫决策过程
DEC-MDP	分布式马尔科夫决策过程
DEC-POMDP	分布式局部可观察马尔科夫决策过程
I-POMDP	交互式局部可观察马尔科夫决策过程
POSG	局部可观察随机博弈问题
DP	多智能体的动态规划算法
JESP	基于联合均衡的策略搜索算法
PBDP	基于信念点的动态规划算法
MBDP	内存有限的动态规划算法
IMBDP	改进的内存有限的动态规划算法
MBDP-OC	基于观察压缩的内存有限的动态规划算法
PBIP	基于信念点的增量式策略删减算法
PBIP-IPG	基于信念点的增量式策略生成和删减算法
MAOP-COMM	通讯受限的多智能体在线规划算法
PBPG	基于信念点的近似策略生成算法
TBDP	基于测试反馈的动态规划算法
DGD	分布式梯度下降算法
RTDP	实时动态规划算法
DecRSPI	应用展开式采样的策略迭代算法
Dec-Comm	可能联合信念推理算法
BaGA	序列贝叶斯博弈近似算法
I	有限的智能体集合
S	有限的系统状态集合
s	系统的状态 $s \in S$
A_i	智能体 i 可采取的动作的集合
a_i	智能体 i 的动作 $a_i \in A_i$
$\vec{A}$	全体智能体的联合行动集 $\vec{A} = \times_{i \in I} A_i$
$\vec{a}$	全体智能体的联合行动 $\vec{a} \in \vec{A}$
Ω_i	智能体 i 可获得的观察的集合

o_i	智能体 i 的观察 $o_i \in \Omega_i$
$\vec{\Omega}$	全体智能体的联合观察集 $\vec{\Omega} = \times_{i \in I} \Omega_i$
$\vec{o}$	全体智能体的联合观察 $\vec{o} \in \vec{\Omega}$
$P(s'\|s, \vec{a})$	系统于状态 s 执行动作 $\vec{a}$ 后转移到状态 s' 的概率
$O(\vec{o}\|s, \vec{a})$	系统于状态 s 执行动作 $\vec{a}$ 后获得观察 $\vec{o}$ 的概率
$R(s, \vec{a})$	系统于状态 s 执行动作 $\vec{a}$ 后获得的收益
b^0	系统的初始状态分布 $b^0 \in \Delta(S)$
Q_i	智能体 i 的策略集
q_i	智能体 i 的策略 $q_i \in Q_i$
$\vec{Q}$	所有智能体的联合策略集 $\vec{Q} = \times_{i \in I} Q_i$
$\vec{q}$	所有智能体的联合策略 $\vec{q} \in \vec{Q}$
$V(s, \vec{q})$	联合策略 $\vec{q}$ 在状态 s 上的期望值
$V(b, \vec{q})$	联合策略 $\vec{q}$ 在信念点 b 上的期望值 $V(b, \vec{q}) = \sum_{s \in S} b(s) V(s, \vec{q})$
H_i^t	智能体 i 在 t 时刻的历史信息集
h_i^t	智能体 i 在 t 时刻的历史信息 $h_i^t \in H_i^t$
$\vec{H}^t$	所有智能体在 t 时刻的联合历史信息集 $\vec{H}^t = \times_{i \in I} H_i^t$
$\vec{h}^t$	所有智能体在 t 时刻的联合历史信息 $\vec{h}^t \in \vec{H}^t$

第 1 章　绪论

1.1　引言

近年来，随着计算机硬件设备和人工智能技术的快速发展，大型的智能系统逐渐渗入到人们生活的方方面面，并起着越来越重要的作用。典型的例子包括用于家庭和医院等日常环境下的服务机器人，用于火灾和地震等恶劣环境下的救援机器人，以及用于城市交通和生态环境的智能监控网络等等。在这些典型环境中，智能体（Agent）都必须能够自主的根据传感器提供的局部环境信息做出决策。通常来说，传感器收集到的信息是带有噪声的，同时执行部件的控制也往往存在一定的误差。这些不确定性给智能体的决策提供了很大的挑战，因为智能体需要在决策中分析和考虑多种可能的情况。当有多个智能体参与行动时，情况会变得更加复杂，因为每个智能体都必须要考虑到其他智能体可能采取的行动和策略。如何在不确定性的环境下设计出高效的多智能体决策算法是当前人工智能最为重要的研究课题之一。

在许多实际的应用问题中，智能体需要完成的是多步决策问题，也就是说智能体不仅仅要考虑行为的当前效果，还需要考虑该行为对未来决策的影响。多步决策问题同时也被称为序列化决策问题（Sequential Decision-Making Problem），而解决这一问题的过程被称作规划（Planning）。具体的说，规划的过程就是产生一个行动序列，使得智能体按照这个行动序列执行的时候能够达到某些目标，完成指定的任务。例如最短路径问题（Shortest Path Problem）就是经典的规划问题，规划的输入是一个连通的图以及一个起点和一个终点，图的边上带有权值表征两个节点之间的距离，规划的目标是找出这样的一个节点序列，使得机器人从起点移动到终点的路径和最小。

本文研究的重点是自动规划问题（Automatic Planning），目标是设计算法，输入问题的模型表示，通过计算自动的输出智能体所能执行的行动序列。经典的规划问题的模型由以下基本元素组成：一、问题的状态集，例如最短路径问题中的节点集；二、智能体的行动集，例如最短路径问题中从一个节点移动到另一个节点的动作；三、关于状态转移的描述，例如最短路径问题中，如果两个节点之间有边，则智能体可以从一个节点移动到另一个节点；四、问题的目标函数，例如最短路径问题中的目标点以及总路径最短的要求。许多人工智能的问题都可以转化为自动规划问题，例如国际象棋的求解就需要设计算法输入象棋的状态描述和规则，输出能战胜对手的下棋套路。

经典规划（Classical Planning）问题往往不考虑状态转移的不确定性，例如

象棋的下子是确定的。事实上，在许多现实问题中，动作的不确定性无处不在。例如机器人从一点移动到另一点可能因为轮子打滑等因素，不能准确地运行到目标点。在多步决策问题中，每一步行动的不确定性会被不断地积累放大，最终产生意想不到的结果，这个过程往往被通俗的称谓蝴蝶效应。解决的方案就是在规划的过程中完整的考虑每个动作各种可能的效果，同时要求从环境中获取动作的反馈，也就是说需要从无反馈的开环控制变为带反馈的闭环控制。在经典规划中每一步行动的效果是确定的，无须系统的反馈就可以准确的获知行动的效果。而在不确定性的环境下，动作可能有多种效果，就需要有反馈信息来预测动作实际产生的效果。例如在机器人移动问题中，机器人需要利用自身的传感器信息比如摄像头和轮子的反馈来预测自己的实际位置，并根据这个实际的位置作出下一步的决策。由此可见，不确定性环境下的规划问题比经典规划问题要难得多。解决这一问题更一般的方法就是马尔科夫决策理论，这便是本文关注的重点。

1.2　不确定性环境下的决策与规划

马尔科夫决策理论（Markov Decision Theory）为不确定性环境下的决策和规划提供了统一的模型表示和丰富的数学基础。它抽象出了智能体决策所需要的关键因素，并用数学模型进行严格的描述。这样就能最大限度的忽略智能体之间的个体差异，同时能体现出不同决策问题间的本质区别，方便对不同类别的决策问题进行理论上的分析。无论是智能软件（Software Agent）还是实体机器人（Physical Robot）的决策问题都可以用统一的模型来表示和求解。对于同一类的决策问题也可以从理论上分析出问题的复杂度，证明是否存在多项式时间复杂度的最优算法。在马氏决策理论中，最关键的决策因素就是系统的状态（State）。它抽象出了智能体当前决策所需要的所有信息，也就是说智能体根据当前的状态就可以做出最优的决策而无需考虑状态之外的其他信息，比如历史状态和动作等。这就是状态的马尔科夫性（Markovian Property），也是马氏决策理论和其他决策理论的主要区别之一。

对于单智能体（Single-Agent）的决策问题，可以用马尔科夫决策过程（MDP）来进行建模和求解。在 MDP 的每个决策周期间，智能体首先从环境中获得系统的状态，并根据状态选择一个行动（Action）作用到环境中。这个动作将导致系统从一个状态转移到另一个状态，同时智能体获得一定的收益（Reward）。MDP 的转移函数（Transition Function）描述了系统从一个状态转移到另一个状态的概率，能够方便的对环境和动作的不确定性进行建模。而 MDP 的收益函数（Reward Function）描述的是在特定状态下，智能体的某个行动所

获得的回报。与传统的基于目标的决策模型相比，MDP 的收益函数是对决策目标更加一般的表示。对于经典规划，目标状态可以用非常大的收益值来表示，而每一步行动的代价则可以用一些较小的负收益值来表示。求解这样的一个 MDP，所获得的决策就是用最少的步骤到达目标状态的一个规划。在 MDP 中，总的决策周期被称为 Horizon，可以是有限的也可以是无限的。MDP 的决策目标是求解出一个能够在总的决策周期内最大化系统收益和的策略（Policy)。正是因为状态的马尔科夫性方便了问题的求解，已经被证明 MDP 的问题复杂度是 P，也就是存在多项式时间的最优算法。尽管如此，在大规模问题下，MDP 的快速求解仍然是非常困难的，因为状态的个数可能非常的多。

MDP 的一个重要的扩展就是局部可观察的马尔科夫决策过程（POMDP)。在很多实际问题中，智能体并不能直接获得系统的状态信息，其对于状态的观察来源于传感器收集到的带噪声的局部信息。例如移动机器人获取环境信息的主要途径是通过其身上配备的摄像头，而摄像头传回的一帧帧的图像代表的只是机器人所处环境的一个局部。在 POMDP 中，这些局部的反馈信息称为观察（Observation)。POMDP 中的观察函数（Observation Function）建模了智能体信息获取的局部性和不确定性。与 MDP 中的状态不同，POMDP 中一步的观察信息不再具备马尔科夫性。在 POMDP 的决策过程中，智能体不仅需要考虑当前的观察信息，也需要考虑过去的历史信息，从而分析从自身所处状态的一个概率分布。这个关于状态的概率分布也被称为信念状态。由于概率分布是一个连续的空间，所以 POMDP 的求解要比 MDP 难得多。可以从理论上证明，POMDP 问题的复杂度是 PSPACE。所以对于一般问题要精确求解是十分困难的，因此研究的重点是找出高效的近似算法。

老虎问题（Tiger Problem）是最为经典的 POMDP 问题，同时也最能体现不确定性环境下决策与规划的特点。老虎问题描述的是这样一个场景：一个机器人被关在一个密室中，密室的左右两边各有一扇门；一扇门后是珍贵的珠宝，而另一扇门后面是凶恶的老虎；如果机器人打开珠宝的那扇门，则可以获得大量的财富；但如果机器人打开的是老虎的那扇门则会被撕成碎片；初始的条件下，机器人不知道哪边有老虎，只能隐约的听到门后老虎的咆哮声。这个问题用 POMDP 可以描述为：一、系统的状态有两个，分别是老虎在左边门后和老虎在右边门后，即 State={ Tiger_Left, Tiger_Right }；二、智能体（机器人）的动作有三个，分别是打开左边的门、打开右边的门和监听老虎的叫声，即 Action ={ Open_Left, Open_Right, Listen }；三、智能体的观察也有三个，分别是听到左边门后传来老虎的叫声、听到右边门后传来老虎的叫声和没有听到老虎的叫声，即 Observation = { Roar_Left, Roar_Right, None }。问题开始的时候，老虎和珠宝被随机的放置到两个门后。如果机器人知道老虎在哪扇门后，也就是说智能体

能够直接的获得系统的状态，老虎问题则退化成简单的 MDP 问题。根据其收益函数的定义，其最优策略是显而易见的：如果老虎在左边则打开右边的那扇门；如果老虎在右边则打开左边的那扇门。

在 POMDP 的问题设定中，智能体不是直接的获得系统的状态而是状态的一个观察，这个观察一定程度上反映了当前系统的状态，但观察本身是具有不确定性的。在老虎问题中，机器人听到的是老虎的叫声，这个叫声可能时有时无，也可能很轻微。因为环境的因素，机器人无法准确的判断老虎的叫声是从哪扇门后传来的。这些都可以通过 POMDP 的观察函数来建模。初始的条件下，因为机器人不知道具体老虎是在哪扇门后，所以对机器人来说老虎在左右两扇门后的可能性是等概率的。机器人要做的就是求解这个 POMDP，并得到一个策略：通过不断的监听老虎的叫声，直到以较大的概率确信老虎是在某扇门后，然后打来另一扇门，这样就能以较大的概率获得珠宝。

总的来说，在马氏决策理论中，转移函数建模的是在不确定性的环境下动作的效果，观察函数建模的是信息获取的不确定性和不完整性。而收益函数建模的是问题的求解目标，也就是智能体在这个问题中所要完成的任务。在老虎问题中，收益函数设计成获得珠宝并避免老虎的伤害。如果机器人的电池容量有限，而每一步的动作都要消耗一定的电能，则收益函数可以设计成每一步的动作都有一个较小的负收益。这样 POMDP 的求解就会自动权衡综合的收益情况，尽量在电池耗尽前打开最为确信有珠宝而没老虎的那扇门。从这个简单的分析中，我们可以体会到马氏决策理论在不确定性环境下问题求解的优势：通过统一的描述清楚的把握问题的特点，同时通过综合的收益值能够很容易的评价策略的期望效用。

经过几十年的发展，许多基于马氏决策理论的高效求解算法被提出，可求解的问题规模也在不断的变大。但大规模问题的求解，仍然极具挑战性。马氏决策模型求解的主要瓶颈就是著名的维度诅咒（The Curse of Dimensionality）。对于 MDP，维度诅咒表现为问题的状态数会随问题的规模呈指数式的增长。例如在平面移动的机器人控制这个问题中：如果只考虑机器人的空间位置、运动的速度以及加速度的话则有六个维度；假如每个维度离散为 N 个量，则总的状态数为 N^6。在实际问题中，上百万个状态的 MDP 模型是十分常见的。对于 POMDP，除了状态数外，维度诅咒还表现为可能的观察序列随决策的总步数呈指数式的增长。当有多个智能体参与决策的时候，维度诅咒对问题求解带来的挑战就更加的严峻。因为每个智能体的局部状态就有多个维度，所以总的问题得状态数随智能体的个数呈指数式的增长。此外，多智能体的规划问题还需要考虑其他智能体所可能获得的观察以及所可能采取的策略，因为只有这样多智能体之间才能互相协调，合作完成共同的任务。

1.3　多智能体系统的协调与合作

多智能体系统（Multi-Agent Systems）通常是指由超过一个智能体组成的联合决策系统。系统中的每一个智能体是一个独立的决策个体，它通过自己从环境中获得的信息自主的做出决策。每个智能体独立的对环境施加一个动作，而整个系统的行为则受所有智能体的联合行动（Joint Action）影响。很显然，每个智能体在做决策的时候都必须要考虑其他智能体所可能采取的动作，以及这个可能采取的动作对自己决策的影响。例如机器人足球（Robot Soccer）就是一个典型的多智能体系统。在机器人足球中，每个球员是一个自主决策的智能体，它在球场上运动，独立的收集信息，并根据自己的判断做出决策。每个球员做出决策的时候不仅要考虑自己队友的行为，同时也受到对手行为的影响。从理论上分析智能体彼此之间的影响是复杂的，而博弈论（Game Theory）则为这种分析提供了很好的数学框架。

范式博弈（Normal-Form Game）通常被用来分析智能体间的相互决策过程。在博弈问题中，每一个智能体被称为一个参与者（Player）。每个参与者都有自己的一个策略集，而对每一个联合策略参与者定义有一套自己的效用函数用来表征自己能得到的收益（负收益即为代价）。例如在著名的囚徒困境（Prisoner's Dilemma）中，每个囚徒是一个参与者，他可以选择认罪或者沉默，也就是说他的策略集是 {认罪，沉默}。对于甲囚犯来说：如果他和乙囚犯都认罪，则两人同时服刑两年；如果他和乙囚犯都沉默，则两人同时服刑半年；但如果他认罪而乙囚犯沉默，则他被无罪释放而乙服刑十年；反之如果他沉默而乙囚犯认罪，则他要服刑十年而乙被无罪释放。这就是甲囚犯对于联合策略的效用函数。同样的，可以定义乙囚犯的效用函数。由此可见，效用函数是定义在所有参与者的联合行动上的，每个参与者想最大化自己的收益都必须考虑到其他参与者所可能采取的策略。如果在某种情况下，没有一个参与者能独自改变自己的策略从而增加收益时，这个联合策略被称为纳什均衡（Nash Equilibrium）点。这是博弈问题中一个理性的稳定解，例如在囚徒困境中，两个囚徒同时选择认罪就是一个纳什均衡点。

在范式博弈中，形势化的描述了参与者的策略选择过程，但对这个策略怎么生成或者说博弈过程怎么进行没有进行描述。如果范式博弈中的每个策略对应的是智能体的一个具体化的行动，那么范式博弈分析的只是一步决策过程。对于多步博弈过程，例如国际象棋，通常用扩展式博弈（Extensive-Form Game）来进行分析。扩展式博弈把具体的博弈过程组织成树状结构。在树的节点，不同的参与者按照自己的策略选择一条边，而博弈的最终结果则反映在树的叶子节点上。在国际象棋中，参与比赛的甲乙两个选手轮流下棋，则博弈树中奇数

的节点代表甲方做决策，而偶数节点代表的是乙方做决策，而博弈树的每一条边则代表的是该选手在这一步棋所可能采用的棋招。叶子节点则代表棋局最终的输赢，通常用一个数字来表示，例如 1 表示甲赢乙输，而 -1 表示甲输乙赢，0 则表示该盘和棋。扩展式博弈的最大问题是所生成的博弈树可能极其的大。例如在国际象棋中，每一步可能有上千种不同的棋招，也就是说每一个节点可能延伸出上千条边，而博弈的步数也可能有上百步。博弈树的规模是随着博弈的总步数呈指数式的增长的。而且收益值只表示在最终的叶子节点，中间节点的收益值无法表示。在很多问题中，这是不恰当的，比如机器人足球中，机器人电池的带电量是有限的，每一步动作都要消耗不一样的电能。在扩展式的博弈中，各方是轮流出招的，而像机器人足球中参与者同时决策时，情形则有所不同。而动作带有不确定性时，扩展式的博弈同样很难表示。

马尔科夫博弈（Markov Game）则不存在扩展式博弈的这些问题。马氏博弈中引入了状态的概念，和 MDP 一样，这个状态具有马尔科夫性。也就是在博弈的过程中，参与者在做决策的时候无须考虑过往的博弈历史，而只要考虑博弈问题当前所处的状态。例如在国际象棋中，参赛的双方只需要考虑当前的棋局，而无须考虑之前双方的出招就可以确定下一步的策略。棋局中各个棋子的分布情况就是一个状态。马氏博弈中的每个参与者各自有一个动作集，里面包含了该参与者所能采取的全部动作。而状态的转移依赖的是所有参与者的联合行动以及转移函数的定义。和 MDP 类似，这个转移函数可以表示联合行动的不确定性。和范式博弈一样，马氏博弈的每个参与者有自己的效用函数，用来评价在特定状态下采用某个联合行动时自己所能得到的收益。

所有的博弈问题都可以分为非合作博弈（Non-Cooperative Game）和合作博弈（Cooperative Game）两类。这个分类特点都体现在效用函数的定义上。非合作博弈的极端情景就是所谓的零和博弈（Zero-Sum Game），在效用函数上表现为“我得即为彼失”，一般的棋类问题和球类竞技都是零和博弈的例子。而一般的现实问题都没有这么严格的冲突，而是既有冲突又有互惠，例如一般的商业行为和国家问题，此类更一般的问题称之为一般博弈（General-Sum Game）。合作博弈是一类特殊的博弈问题，所有的参与者拥有相同的效用函数，各人的利益即是群体的利益，也就是他们组成的是一个完全合作的团队。例如机器人足球问题中的一方，每个队员的行为关系到整个队伍的胜负，同时所有的队员为了队伍的胜利这个统一的目标而战。合作博弈是相对简单的博弈，因为最优解是一定存在的，而求解合作博弈的目标就是找到这样的一个最优解。

合作的马尔科夫博弈问题也被称为团队马尔科夫博弈（Team Markov Game），也就是所有的参与者拥有同一个效用函数，也就是他们具有相同的目标。在这个意义上，团队马尔科夫博弈和多智能体的马氏决策过程（MMDP）

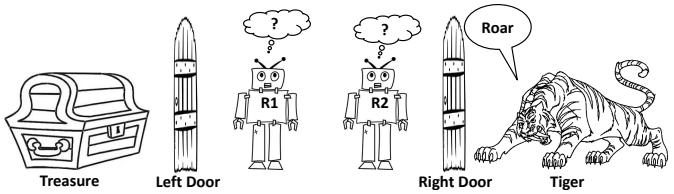

图 1.1 多智能体的老虎问题

是等价的。MMDP 和 MDP 在模型定义上的唯一差别就是用多智能体的联合行动代替 MDP 中的一个动作，而系统地状态是被所有的智能体完全获得的。在问题求解上 MMDP 和 MDP 也是类似的，只是 MMDP 的状态数和动作数都随着智能体的个数呈指数式的增长。MMDP 是决策论和博弈论的一个结合点，在多智能体协作问题的求解上，既可以应用决策论的成果，也可以参考博弈论的问题求解方法。通常问题求解的策略是：在整体问题上应用马氏决策理论的相关方法处理状态之间的转移，而在每一步的决策上采取的是和博弈论类似的方法进行动作选择。

多智能体的马氏决策过程（MMDP）[1] 要求在决策的每一个周期，所有的智能体都能完整而完美的获得系统的状态信息，这个要求在很多实际问题中都很难得到满足。而多智能体系统的一大优势恰恰是分布式的信息获取和决策，也就是说每个智能体只需要根据自己的局部信息就能进行决策。很多自然存在的多智能体系统都是分布式控制的。例如机器人足球中的每个机器人都必须根据自己传感器获得的球场上的局部信息独立的进行决策，完成跑位、传接球以及射门等行为。再比如在传感器网络中，要获得完整的信息就必须对每个传感器所获得的局部信息进行实时地同步，这将带来巨大的带宽消耗，而在传感器彼此距离很远的网络间，这样的实时同步也不现实。而大型的网络系统例如 Internet 更是无法提供完整的状态信息。在许多的多智能体系统中，虽然每个智能体只能获得局部的信息，但决策上却需要考虑整体团队的行为，特别是团队成员间有密切合作的情况下，例如足球机器人间的传接球。这为多智能体系统的分布式决策带来了极大的挑战。

本文所关注的分布式局部可观察的马尔科夫决策过程（DEC-POMDP）就是马氏决策理论中 POMDP 在多智能体协作问题上的自然扩展。例如上文提到的老虎问题可以很自然的拥有多智能体协作问题的版本，即所谓的多智能体老虎问题（Multi-Agent Tiger Problem）[2]。在这个问题中，密室里出现了两个机器人，各自能独立的用自己的机械手打开一扇门，也能独立的用自己的传感器监听老虎的叫声，但它们彼此之间不能通讯，即不能互相交换信息。它们的行为被设定成完全合作的：如果其中由一个机器人独自打开有老虎的那扇门则整个团队获得很大的惩罚；但如果两个机器人同时打开老虎的门，则获得的惩罚

较小。也就是在收益函数的设计上鼓励两个机器人彼此间展开合作。但由于机器人间获得的局部信息不一致，且彼此间不能交换信息，要完成这样的合作是困难的。事实上，从单智能体的 POMDP 扩展到多智能体的 DEC-POMDP，问题的难度发生了根本的变化，可以证明 DEC-POMDP 的问题复杂度是 NEXP[3] 比 POMDP 的 NSPACE 要难得多。此外，从博弈论的角度说，DEC-POMDP 也可以看作是局部可观察随机博弈（POSG）的拥有完全合作假设的一个特例。从博弈论的角度看，POSG 也是一个非常难解的问题。

1.4　论文的内容与组织结构

本文的余下部分，首先在接下来的第二章中介绍 DEC-POMDP 的背景知识，其中包括：一、DEC-POMDP 的形式化描述以及它跟其他类似模型之间的关系；二、基于 DEC-POMDP 的离线规划的相关背景知识以及与本文工作相关的一些经典的算法的分析讨论；三、基于 DEC-POMDP 的在线规划的相关背景和经典算法的分析讨论以及多智能在线通讯的一些基本策略。本章关于 DEC-POMDP 模型以及相关的规划算法的介绍为描述接下来三章中的主要工作做了重要的铺垫。第三章到第五章主要介绍作者的三个基于 DEC-POMDP 模型的主要工作。每个工作独立成章，首先分析该工作的主要动机，而后针对解决方案逐点进行详细的介绍，最后给出实验结果并对该工作进行一个的小结。本文的最后一章将对整个工作进行一个系统的总结，并在此基础上对未来的研究工作进行展望。

第三章介绍的是通讯受限的多智能体在线规划算法 MAOP-COMM。在这一章中首先介绍算法的基本流程以及一些重要的概念。与离线算法不同，在线算法需要在执行的过程中进行规划。由于每个智能体获取到的信息是局部的而且互不相同，因此就需要一个机制避免智能体行为之间的误协调。同时由于用于规划的时间十分有限，因此需要有一个快速的搜索算法来计算每一步的策略。一个智能体需要同其他智能体进行合作，因此就需要对其他智能体的信息进行推理，信念池就是一个用于存储和表示所有智能体信息的关键数据结构。但信念池的信息会随着时间的增加不断的暴涨，所以需要一个压缩算法来对信息进行压缩。信息压缩可能会带来误差，使得信念池中的信息与实际情况不一致，因此通讯被用来处理信念池中的不一致性。此外，这一章中还详细的描述了信念池在算法中的具体实现。在实验部分，MAOP-COMM 被用来求解通讯信道可靠和不可靠情况下一系列多智能体的合作问题。

第四章介绍的是两个离线规划算法，其中一个是基于信念点的策略生成算法 PBPG，另一个是基于测试反馈的策略评价算法 TBDP。在 PBPG 算法部分，

主要描述的是如何将策略生成问题转化为最优映射的求解问题。在理论上分析了该方法与 MBDP 的先枚举再选择的方式等价之后，提出了能够快速求解的近似算法。该近似算法彻底的解决了策略数指数式爆炸的问题，同时在实验部分发现对于大多数的问题，近似算法的效果都足够的好。最值得一提的是，随着子策略数的增加，PBPG 算法能够近似最优的求解一些标准的 DEC-POMDP 测试问题。在 TBDP 算法部分，主要关注的是策略的评价方面。由于要考虑状态，因此就出现了所有马尔科夫决策模型都共同存在的问题——“维度诅咒”。解决该问题的关键是进行状态的可达性分析，而进行可达性分析最好的方法就是利用测试反馈。这是基于这一思想，TBDP 算法被提出，并体现了其在大状态问题上的优势。同时，一个新的策略表示方法被提出，该方法所表示的策略可以被通过参数优化的方式快速的计算。同时，在计算参数是能够确定哪些值函数才是真正需要计算的。这种“惰性”评价方法进一步提高了策略评价的效率。而且基于测试反馈的方法可以方便的利用并行分布式计算资源，这位现实中大规模问题的快速求解提供了希望。

第五章介绍的是无需完整模型的应用展开式采样的策略迭代算法 DecRSPI。在经典的规划问题中，一般假设模型已经建立好，而规划算法只需要根据模型提供的信息计算出问题所需的策略。但很多情况下，要获得一个完整的模型是一个困难的过程。特别是对多智能体系统而言，由于智能体个数较多，它们之间的行为较为复杂，很难通过一定的概率转移来玩去的表示。即便能够通过测量的方法获得一定的概率，但由于这些概率函数都是状态、联合行动和联合观察的函数，而这些联合信息都是随着智能体的个数的增加指数式的增长，因此对于大规模问题，要完全表示和存储极其困难。因此，在 DecRSPI 算法中，通过对环境的采样来计算策略，从而避免了建立模型的过程中可能遇见的困难。DecRSPI 算法通过采样来获得每一步的状态分布，同样通过蒙特卡罗采样来估计计算策略时所需的相关参数。在实验中，基于仿真器的 DecRSPI 算法与基于完全模型的规划算法相比毫不逊色。同时在其他规划算法无法求解的智能体个数较多的问题，DecRSPI 算法题充分体现了它的优势。

这三章相互独立却又彼此联系，特别是第四章和第五章中提到的算法。也就是说，DecRSPI 算法是在 TBDP 算法的基础上进一步扩展而来。同时 TBDP 算法同 PBPG 算法同样有着紧密的联系，因为 TBDP 基于的就是 PBPG 算法中利用到的将策略参数化的思想。但这三个算法又有各自的特点，例如 TBDP 算法主要针对的是大状态问题，对于小状态问题完全可以通过普通的方法直接计算策略函数并利用 PBPG 算法得到最好的效果。而 DecRSPI 算法主要是针对模型无法严格表示但相关的仿真器却比较容易计算的问题，对于那些已经给出完整模型的问题 TBDP 或 PBPG 算法就能完全胜任。在实验中，DecRSPI 也用来

求解那些已经存在完整模型的 DEC-POMDP 标准测试问题，其目的在于表明在 DecRSPI 是一个通用的算法，可以用来求解各种类型的问题，同时对着这些问题通过采样计算的策略与通过完整模型计算的策略在实际效果上并不会有太大的差异。在线算法 MAOP-COMM 与离线算法之间就相对疏远，主要是因为二者在问题求解上所采样的方式上存在巨大的差异。但 MAOP-COMM 算法与另外三个算法也并不是毫无联系，因为 MAOP-COMM 算法的每一步决策相当于离线算法的一次迭代过程。所以在策略求解上存在一定的相似性。但从根本上说，二者关注的问题是不同的，例如在线算法关注的是分布式的协调、在线信息的压缩以及通信决策，而离线算法则关注策略的表示、策略的生产以及策略的评价等方面。

第 2 章　多智能体的马尔科夫决策模型

团队协作是多智能体研究的核心问题之一。早期对于多智能体团队协作的研究主要依赖的是合作协议的设计，这些协议或多或少的依赖于 BDI 逻辑，即用信念、愿望和意图来对智能体的行为进行表示和推理。对于多智能体的合作问题提出了联合意图的概念，即智能体之间为实现共同的目标而相互缔结的承诺。此外智能体之间还可以通过设定好的协商（Negotiation）模式来进行合作。这些基于逻辑协议的合作机制最大的劣势在于很难定量的对智能体的行为进行评估。特别是当环境存在不确定性时，对于合作协议效果的定量评估需要用到大量的场景和应用不同的规则。此外，合作协议往往是针对具体问题进行设计的。这种“自下而上”的研究方法最大的缺陷在于本质上相同的问题在不同的具体环境下被重复的解决了多次。也就是说合作协议通常不具备通用性，同时也很难在这个层面上对整个多智能体团队协作问题进行分门别类的研究。

DEC-POMDP 模型则还很大程度上弥补了这些缺陷。马尔科夫决策模型是一种通用的模型（Universal Model），遵循的是一种“自上而下”的研究思想。DEC-POMDP 的目标在于表示常见的多智能体团队协作问题，在这些问题中抽取出本质的决策因素，而忽略问题间具体实现细节的差别。而针对 DEC-POMDP 的理论研究和求解算法可以很容易的应用到这些问题中。对于不同的算法，可以设立一系列的标准测试问题（Benchmark Problem）来进行评估。从这个意义上说，DEC-POMDP 在一定程度上统一了多智能体团队协作的问题研究，不仅从理论上深层次的理解了问题的本质，同时也加大了各种算法的通用性，对多智能体问题乃至整个人工智能的研究都具有重大的意义。

2.1　分布式局部可观察马尔科夫决策过程

概括的说，DEC-POMDP 建模的是一组智能体在不确定性环境下的决策过程。下面给出了其严格的数学定义，并详细解释模型的各个组成部分：

定义 2.1.1 (DEC-POMDP). *分布式局部可观察马尔科夫决策过程 (DEC-POMDP) 可以形式化的定义为一个多元组* $\langle I, S, \{A_i\}, \{\Omega_i\}, P, O, R, b^0 \rangle$，*其中：*

- I *表示一个有限的智能体集合。为方便起见，我们将集合中的每个智能体编号为* $1, 2, \cdots, n$，*其中* $n = |I|$。*同时注意到当* $n = 1$ *时，DEC-POMDP 等价于单智能体的 POMDP 模型。*

- S 表示一个有限的系统状态集合。在马尔科夫决策理论中，一个状态 $s \in S$ 包含了系统决策所需要的所有信息，也就是说，系统的下一个状态由当前的状态和将采取的行动唯一确定，而与之前的状态和动作无关。这就是所谓的马尔科夫性，可形式化的表示为：

$$P(s^{t+1}|s^0, \vec{a}^0, \cdots, s^{t-1}, \vec{a}^{t-1}, s^t, \vec{a}^t) = P(s^{t+1}|s^t, \vec{a}^t)$$

- A_i 表示智能体 i 可采取的动作的集合。一般说来，一个智能体的动作集可以是连续的，也可以是离散的。在本文中，我们只考虑动作集是离散而且是有限的情形。不失一般性，$\vec{A} = \times_{i \in I} A_i$ 表示所有智能体的联合动作集，其中 $\vec{a} = \langle a_1, a_2, \cdots, a_n \rangle$ 表示一个联合行动。在模型中，我们假定每一步中智能体所采取的行动不能被其他智能体直接观察到。

- Ω_i 表示智能体 i 可获得的观察的集合。和动作集相似，一个观测集可以是离散的也可以是连续的，本文一律只考虑离散且有限集合的情况。同样的，我们也可以定义联合观察集 $\vec{\Omega} = \times_{i \in I} \Omega_i$，以及其中的一个联合观察 $\vec{o} = \langle o_1, o_2, \cdots, o_n \rangle$。在模型中，我们假定每个智能体智能获得各自的观察 o_i，而不能感知其他智能体的观察。

- $P : S \times \vec{A} \times S \to [0, 1]$ 表示系统的转移函数。$P(s'|s, \vec{a})$ 表示在状态 s 中采取联合行动 $\vec{a}$ 后转移到新的状态 s' 的概率。直观上说，转移函数建模了动作和环境的不确定性，在不确定性的环境中采取一个动作后所转移到的状态是不确定的，满足一定概率分布这个概率分布就是 $P(\cdot|s, \vec{a})$。在本文中，我们假定这个概率分布是不随时间的变化而发生变化，即是固定的。

- $O : S \times \vec{A} \times \vec{\Omega} \to [0, 1]$ 表示系统的观察函数。$O(\vec{o}|s', \vec{a})$ 表示采取联合行动 $\vec{a}$ 后转移到新状态 s' 时获得联合观察 $\vec{o}$ 的概率。直观上说，观测函数建模的是智能体感知的不确定性，即感知的噪声。同时智能体不能直接获取系统的状态信息，只能获得状态的一个表征也就是这里所说的观察。从而我们称智能体对系统的状态是局部可观察的。同时这个观察还带有概率，即不确定性。同样的，在本文中我们也假定观察函数不随时间的变化而变化，和转移函数一样，也是固定的。

- $R : S \times \vec{A} \to \Re$ 表示系统的收益函数。$R(s, \vec{a})$ 表示的是在状态 s 下采取联合行动 $\vec{a}$ 后整个队伍获得的收益，是一个实数值。在一些模型中采取的是代价函数，这与收益函数是等价的，因为代价即是负收益。直观上说，收益函数建模的是智能体所要完成的任务或者目标，收益函数表示了智能体

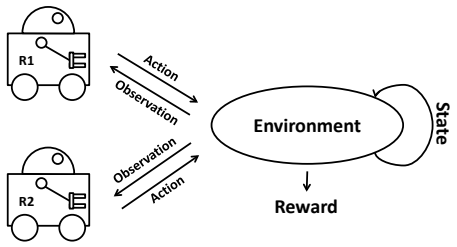

图 2.1 多智能体的决策模型

在每个状态下采取特定行动后所能获得的回报。在模型中，我们假定每一步种智能体不能感知到这样的一个收益反馈。

- $b^0 \in \Delta(S)$ 表示系统的初始状态分布。这个初始的状态分布是所有智能体所共知的，智能体根据这样一个出示的状态分布进行他们的动作规划。

在本文中，我们只考虑决策周期是离散的情形。总的决策周期称为步数(Horizon)，并用 T 来表示。在每个决策周期 $t = 0, 1, 2, \cdots, T-1$ 内，每个智能体做出决策并执行一个动作。所有智能体的联合行动导致系统从一个状态转移到另一个状态，系统进入下一个决策周期。在下一个决策周期中，每个智能体首先从系统获得各自的观察，然后做出决策并执行动作，整个过程周而复始。根据收益函数的定义，每个决策周期，智能体团队能从联合动作的执行过程中获得一个即时回报 $r(t) = R(s, \vec{a})$。所以整个决策过程的累积收益（Cumulative Reward）可以简单的定义为：$r(0) + r(1) + r(2) + \cdots + r(T-1)$。注意到在 DEC-POMDP 中，智能体的行动是带有不确定性的，所以最优化决策的依据是求解出这样一个联合策略 $\vec{q}$ 使得整个决策过程的期望累积收益值最大，即：

$$V(\vec{q}) \equiv \mathbb{E}\left[\sum_{t=0}^{T-1} R(s, \vec{a}) \middle| \vec{q}, b^0\right] \tag{2.1}$$

定义 2.1.2 (策略)**.** 智能体 i 的策略集 Q_i 定义为其观察的历史序列集 Ω_i^* 到动作集 A_i 的一个映射，即 $Q_i : \Omega_i^* \to A_i$。给定一个策略 $q_i \in Q_i$，智能体 i 就能够根据它的观察历史信息 $o_i^* = (o_i^1, o_i^2, \cdots, o_i^t)$，选择一个动作 $a_i \in A_i$。而一个联合策略 $\vec{q} = \langle q_1, q_2, \cdots, q_n \rangle$ 则定义为所有智能体策略的一个向量。

联合策略的期望累积收益值也被称为策略的值函数（Value Function)。在给定一个状态 s 的情况下，策略 $\vec{q}$ 的值函数可以利用贝尔曼等式递归的定义出来：

$$V(s, \vec{q}) = R(s, \vec{a}) + \sum_{s' \in S} \sum_{\vec{o} \in \vec{\Omega}} P(s'|s, \vec{a}) O(\vec{o}|s', \vec{a}) V(s', \vec{q_o}) \tag{2.2}$$

其中 $\vec{a}$ 是根据 $\vec{q}$ 给出的联合行动，而 $\vec{q_o}$ 是执行动作后根据 $\vec{q}$ 和所获得的观察信

息 $\vec{o}$ 给出的联合子策略。如果给定的是状态的概率分布 b，则策略的值函数可以简单的定义为：$V(b,\vec{q}) = \sum_{s\in S} b(s)V(s,\vec{q})$。所以求解一个 DEC-POMDP 的过程就是找出这样的一个联合策略 $\vec{q}$ 使得在给定初始状态分布 b^0 的情况下，其策略的值函数 $V(b^0,\vec{q})$ 最大。这就是 DEC-POMDP 问题求解的最优化标准。

注意到在 DEC-POMDP 中并没有显式的定义多智能体之间的通信，而事实上智能体间的通讯已经被包含在了原始模型的定义当中。在观察函数的定义中，一个智能体所能获得的观察依赖于系统的状态以及所有智能体的联合行动。所以其他智能体能够通过自己的行动改变其他智能体的观察，从而达到通讯的目的。例如甲智能体希望传递某种消息给乙智能体，甲只要执行特定的行动，该行动能使得乙获得特定的观察，而乙可以通过解析该观察获得这种消息。这种隐式的通讯方式与显式的预先约定通讯协议的方式在本质上是等价的。

在多智能体协作问题的研究上，DEC-POMDP 是一个非常通用的模型，是当前研究的主流。此外还有许多相关的模型被不同的学者研究过，在这里做一个简要的介绍。从表达能力和问题的复杂度上说，MTDP[4] 和 POIPSG[5] 是与 DEC-POMDP 等价的模型。也就是说，适用于 DEC-POMDP 的算法可以几乎不做修改的应用到这些模型中。此外，通过对 DEC-POMDP 模型增加不同的限制条件，可以获得问题复杂度从 NEXP 到 P 的各种子类[6]。在 DEC-POMDP 中，如果所有的智能体能直接获得系统状态的信息，则这个模型退化成 MMDP[1]；如果所有智能体不能直接获得系统的状态，但它们所得到的观察的并集与系统状态一一对应，则这个模型被称为 DEC-MDP 模型。DEC-MDP 与 DEC-POMDP 的差别在于 DEC-MDP 的观察代表的说智能体的一个局部的状态，或者说说系统状态的一个分割，当所有的状态放在一起的时候，这些局部状态的集合能够构成一个完整的系统状态。DEC-MDP 的问题复杂度和 DEC-POMDP 一样都是 NEXP，由此可见 DEC-POMDP 问题求解的最困难的地方在于分布式控制，而不仅仅是信息的不确定性。

从问题的复杂度上说，DEC-POMDP 是一个非常难解的问题。求解 DEC-POMDP 的策略之一是研究实际问题，分析问题所具有的特殊结构，并利用这些结构对其进行求解。例如状态独立转移模型（Transition-Independent DEC-MDP）[7] 利用的就是智能体之间的局部状态转移互不关联这样一个结构，从而提出高效的最优求解算法。该模型被应用到了多火星车的岩石采集问题上，岩石采集是一个整个团队的问题，但火星车在采集的时候互不干预对方的行为，也就是它们局部状态的转移是互相独立的。类似的模型还有基于目标的模型（Goal-Oriented DEC-POMDP）[6]、事件触发模型（Event-Driven DEC-MDP）[8] 以及可应用于大型传感器网络的分布式网络模型（Network Distributed POMDP）[9]。一方面这些模型系统的利用了问题的结构，从而加速了问题的求解；但另

一方面这些结构局限了算法可应用的问题范围。也就是说它们往往只能解决一小部分的多智能体协作问题。在本文中，作者关注的是一般的 DEC-POMDP 模型，也就是无特殊结构限制的模型。研究的目标是设计出能求解所有多智能体协作问题的通用算法。由于 DEC-POMDP 本身的问题复杂度，精确求解算法本身只能求解极小的问题，本文关注的重点是针对大规模问题的近似求解算法。

团队协作是 DEC-POMDP 问题求解的核心。在 DEC-POMDP 中，智能体的决策是分布式的，也就是说每个智能体根据自己所获得的局部观察信息独立的做出决策。模型中并没有显式的定义智能体间的合作协议，所以在决策的时候每个智能体所执行的策略都必须考虑到其他智能体的可能行为。解决这个问题一个直接的想法就是在每个智能体中为其他智能体的行为建立一个模型，这样智能体每次决策的时候就可以根据这个模型来推测其他智能体的行为，然后根据这个推测决定自己将采取的动作。与 DEC-POMDP 类似的交互式 POMDP 模型（I-POMDP)[10] 采取的就是这种方式。但这种方式有一个致命的缺陷，因为智能体在建立其他智能体模型的时候，这个模型需要再考虑到这个智能体本身，这样的相互推理会无穷的递归下去。通俗的说，就是我要想你怎么想的时候，会想到你想我现在在想什么，而我现在想的东西里面包含了你怎么想。针对这种嵌套信念（Nested Belief）的推理本身就是极其困难的，所以 I-POMDP 通常只具有理论上的意义。

与 I-POMDP 不同，DEC-POMDP 并不显式的对其他智能体的行为进行建模。在问题求解的过程中，不对直接对嵌套信念进行推理，而是把 DEC-POMDP 的问题求解看成是联合策略空间中的最优规划问题。根据这一思想，DEC-POMDP 的问题求解方式可以简单的分为离线规划和在线规划两种。

2.2　离线规划的基本原理及存在的问题

离线规划的基本原理是把 DEC-POMDP 的问题求解过程分为离线规划和在线执行两个阶段。在离线规划阶段需要设计一个求解器，输入一个 DEC-POMDP 模型，输出一个联合策略。联合策略的要求是能够方便的分配到每个智能体中进行执行，这就要求联合策略中每个智能体的策略不依赖于其他智能体的策略，而且每个智能体的策略仅仅需要其局部信息就能执行。有了这样一个联合策略，在线执行阶段就变得十分简单，只需要将每个智能体的策略分配到智能体的执行系统中，智能体就可以根据自己所获得的局部信息来进行动作选择。由此可见离线规划要解决的问题重点是：一、可分布式执行的联合策略表示；二、能计算出最优联合策略的问题求解器。

策略树（Decision Tree）是 DEC-POMDP 最常用的策略表示方式。每个智

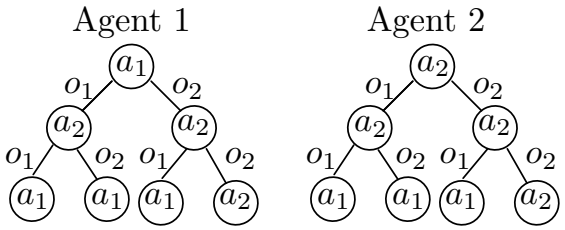

图 2.2 两智能体的策略树示例

能体的策略是一棵独立的策略树，而联合策略则是所有智能体的策略树的集合。在策略树中，树的节点代表的是一个将被执行的动作，而树的每一条边代表的是智能体执行动作后所可能获得的观察。树的根节点是智能体将要执行的第一个动作，而树的深度和总的决策步数 T 相等。但这个用策略树表示的联合策略被计算出来后，在策略的执行阶段，每个智能体获得自己的策略树，并一步一步的执行节点上的动作，并根据自己所获得的局部观察信息沿着相应的分支转移到下一个节点，知道完成所有的决策。由此可见，策略树的执行是十分简单的，可以说几乎没有计算量，而只是一些简单的分支转移。而且，由于执行的是同一个联合策略，整个团队行为的期望效果是有保证的。所以问题的焦点在于如何在离线规划阶段计算出这样的策略树集，使得其执行的希望效果足够的好。同时注意到，在 DEC-POMDP 中智能体的决策是分布的，但离线阶段的策略计算则可以采取集中的方式来进行，只要计算出来的策略树是可以分布式执行的就能够满足要求。

2.2.1　离线精确求解算法

计算策略树最直接的想法就是所谓的蛮力法（Brute-Force Method），即枚举生成所有可能的策略树，并对每个联合策略计算评价值，然后从中选择出最优的联合策略。通过简单的计算，可以知道，所有可能联合策略的个数大概在以下的量级：

$$\mathcal{O}\left(|A_i|^{\frac{(|A_i||\Omega_i|)^T-1}{|A_i||\Omega_i|-1}|I|}\right) \tag{2.3}$$

也就是说，可能的联合策略数是随着决策步数 T 双指数式的增长的。由此可见，DEC-POMDP 的联合策略空间是极其大的，这也从一个侧面反应了其 NEXP 的问题复杂度。用蛮力法来求解，哪怕是最小规模的 DEC-POMDP 问题都是不可能的，可以说完全不具有实用性。如此巨大的策略空间也成为了其他的 DEC-POMDP 求解算法的一个巨大的障碍。

多智能体的 A 星算法（MAA*）[11] 是比蛮力法更好的策略树枚举算法。从一个完整的联合策略出发，根据联合策略的一个启发值不断的展开这个联合策

算法 2.1　多智能体的动态规划

Initialize all depth-1 policy trees
for t=*1* **to** T **do** // backwards
    Perform full backup on $\vec{Q}^t$
    Evaluate policies in $\vec{Q}^{t+1}$
    Prune dominated policies in $\vec{Q}^{t+1}$
return the best joint policy in $\vec{Q}^T$ for b^0

略，直到找出所有最优的联合策略。由于完全展开一个策略树生成的候选策略树的个数也是十分惊人的，所以在 MAA* 中对策略树运用的是从根节点开始的逐步展开。每一步的展开生成的是一个不完全的策略树，也就是这棵策略树有从根节点到 $t-1$ 步的节点，而没有从 t 步到 $T-1$ 步的节点。对于这样一个不完全策略树的启发式评价值分为两个部分，从根节点到 $t-1$ 步的节点是根据贝尔曼等式进行迭代的计算，而从 t 步到 $T-1$ 步的节点由于在不完全策略树中实际没有相应的节点，所以采用的是一个启发值。这个值必须是实际值的一个上界，MAA* 才可以保证找到最优的解。通常采用的值是 DEC-POMDP 忽略局部观察后退化成 MMDP 后计算出的最优值函数，因为 MMDP 的值函数是实际值的一个上界，而且相对容易计算。MAA* 比蛮力法在实际效果上要好的多，而且依然是理论上能保证找到最优联合策略的算法。但是其时间和空间复杂度依然很大，特别是不完全策略树扩展到一定深度的时候，开表中的不完全策略树可能非常的多，每一步的扩展都需要耗费大量的时间和存储空间。所以 MAA* 依然只能解决很小的问题。

另一个遵循枚举策略 DEC-POMDP 最优化算法是近期提出的效果较好的混合整数线性规划法（MILP）[12]。在这个算法中，策略不是表示成策略树，而是采用序列化的表示形式（Sequence Form)。在序列化的策略表示当中，智能体的策略表示成许多动作观察序列，每个序列对应于智能体的一个决策轨迹。通过序列化的表示方式，最优化的联合策略生成问题可以等价的转化为一个大的组合优化问题。而这个组合优化的问题可以采用已有的混合整数线性规划求解器进行求解。

最值得一提的最优化算法是多智能体的动态规划算法（DP）[13]，该算法是许多其他近似求解算法的基础，而且其思想可以扩展到更加复杂的 POSG 问题的求解上。和一般的动态规划算法一样，DEC-POMDP 的动态规划算法也是从最后一步开始逐步向前，计算出一个完整的联合策略的。生成策略树的时候，从叶子节点开始，自底向上一步步的来构建。叶子节点也被称为 0 步策略树，是一棵只有根节点而无分支边的树。对于每个智能体来说，这样的 0 步策

算法 2.2　基于联合均衡的策略搜索

Generate a random joint policy
repeat
  foreach *agent* i **do**
    Fix policies of all agents except i
    for t=T **downto** *1* **do** // forwards
      Generate a set of all possible belief state:
      $B^t(\cdot|B^{t+1}, q^t_{-i}, a_i, o_i), \forall a_i \in A_i, \forall o_i \in \Omega_i$
    for t=*1* **to** T **do** // backwards
      foreach $b^t \in B^t$ **do**
        Compute the best value for $V^t(b^t, a_i)$
    forall the *possible observation sequences* **do**
      for t=T **downto** *1* **do** // forwards
        Update the belief state b^t given q^t_{-i}
        Select the best action by $V^t(b^t, a_i)$
until *no improvement in the policies of all agents*
return the current joint policy

略树是很容易枚举的，因为只需要对根节点赋予不同的动作就可以了。所以 0 步策略树的个数等同于智能体总的动作个数。给定 t 步策略树的情况下，生成 $t+1$ 步策略树的步骤称为备份（Backup），其实是一个 $t+1$ 步策略树的枚举操作：在根节点上赋予不同的动作，然后在每个观察分支赋予 t 步策略树。所以对 $t+1$ 步策略树来说 t 步策略树是它的一个子树。在生成所有的 $t+1$ 步策略树后，对 $t+1$ 步的所有联合策略进行值评价（Evaluation）。在评价完之后，将所有可以删减的 $t+1$ 步策略树剔除（Prune），利用剩下的 $t+1$ 步策略树继续生成 $t+2$ 步策略树，直到整棵策略树的高度为 T。虽然在动态规划中采用了一些策略树的剔除方法，但每一步生成的策略树还是非常的多：

$$|Q_i^{t+1}| = \mathcal{O}\left(|A_i||Q_i^t|^{|\Omega_i|}\right) \tag{2.4}$$

其中 Q_i^t 和 Q_i^{t+1} 分别是智能体 i 的 t 步和 $t+1$ 步的策略树集。在最坏情况下，动态规划算法考虑的策略树个数依然是随着决策步数的增多呈双指数式的增长。也就是说即使是对于很小规模的问题，动态规划生成的策略树都能很快的用完所有的存储空间。虽然 DEC-POMDP 的最优化算法都无法求解规模较大的问题，但这些方法为设计近似算法提供了参考思路，具有重大的理论意义。

算法 2.3　基于信念点的动态规划

Initialize all depth-1 policy trees
for t=*1* **to** T **do** // backwards
  Perform full backup on $\vec{Q}^t$
  Evaluate policies in $\vec{Q}^{t+1}$
  foreach *agent* i **do**
    foreach *possible history* h_i^{T-t-1} **do**
      Generate the set of all possible multi-agent belief states: $B(\cdot|h_i^{T-t-1}, \vec{Q}_{-i}^{t+1})$
      for *each* $b_i^{t+1} \in B(\cdot|h_i^{T-t-1}, \vec{Q}_{-i}^{t+1})$ **do**
        Select the best policy q_i^{t+1} for b_i^{t+1}
  Prune all policies except the selected ones
return the best joint policy in $\vec{Q}^T$ for b^0

2.2.2　离线近似求解算法

基于联合均衡的策略搜索算法（JESP）[2] 是第一个 DEC-POMDP 的近似求解算法。前面提到过，在多智能体的决策问题中，每个智能体不仅要考虑当前的系统状态，还要考虑其他智能体所可能采取的决策。在不知道其他智能体策略的情况下，对其进行嵌套推理是十分困难的。当如果其他智能体可能采取的策略集已经给定，那情况就变得十分的明了。给定其他智能体的策略集 Q_{-i}，智能体 i 的信念状态可以定义为系统状态和其他智能体策略集的联合概率分布 $b_i \in \Delta(S \times Q_{-i})$。根据这一思想，JESP 首先选定一个智能体 i，然后在固定其他智能体策略集的情况下，对智能体 i 的策略进行改进。改进的方法就是，在这个信念的状态空间给定的前提下，生出一些列的候选策略，然后在这些策略当中，选出智能体 i 的当前最好策略。这个过程一直持续下去，并且保证所有智能体都有等概率的机会被选择并进行策略改进，直到所有智能体的策略都无法改进为止。这意味着所有智能体的策略达到了一个纳什均衡。注意到纳什均衡点的联合策略不一定是最优策略，但均衡点的策略往往足够的好，所以该算法是近似求解算法。JESP 的主要问题在于很容易陷入局部最优策略而无法再进一步的改进，同时由于所定义的信念空间可能非常的大，所以其求解算法的时间复杂度依然是指数式的。

基于信念点的动态规划算法（PBDP）[14] 同样是利用生成的多智能体的信念空间来进行问题的求解，不同的是其只考虑巨大信念空间的可达信念状态点。而且该算法采用的是自底向上的一步步动态规划的方法来生成策略。多智能体的信念空间虽然是连续的概率分布，但在给定初始的状态分布 b^0 的前提下，很多部分都是不可达的，所谓不可达就是无论多智能体执行什么样的序列都无法

算法 2.4　内存有限的动态规划

Initialize all depth-1 policy trees
for t=*1* **to** T **do** // backwards
  Perform full backup on $\vec{Q}^t$
  Evaluate policies in $\vec{Q}^{t+1}$
  for k=*1* **to** $maxTrees$ **do**
    Select a heuristic h from the heuristic portfolio
    Generate a state distribution $b \in \Delta(S)$ using h
    Select the best joint policy $\vec{q}^{\,t+1}$ in $\vec{Q}^{t+1}$ for b
  Prune all policies except the selected ones
return the best joint policy in $\vec{Q}^T$ for b^0

出现的状态概率分布。所以在求解策略的时候，没有必要考虑整个的信念空间，而只需要考虑信念空间中那些可达的信念状态点。根据这个观察，在动态规划的第 t 步，PBDP 首先枚举生成所有的 t 步策略树，然后生成一系列的可达信念点，根据这些可达的信念点选出最优的 t 步策略树，然后进入 $t+1$ 步策略树的求解。但所有可达的信念点被完全枚举时，PBDP 能够求解出最优的联合策略。PBDP 采用了可达信念点的分析技巧来避免不必要的计算，但其主要问题还是可达的信念点可能非常的多，在最坏的情况下，需要考虑的策略树依然是双指数式的增长。

内存有限的动态规划算法（MBDP）[15] 的一大改进是使得策略树随求解的步数呈线性的增长，所以很多决策步数较多的问题可以被求解。在 MBDP 之前，能被求解的 DEC-POMDP 的总的决策步数 T 都不超过 10 步。而 MBDP 把同类问题的可求解决策步数扩大到了上百步甚至上千步，这也是 MBDP 求解算法的主要贡献。如图 2.3所示，MBDP 的主要思想与 PBDP 类似，将自顶向下的搜索和自底向上的动态规划相结合来生成策略树。与 PBDP 不同的是，MBDP 在自顶向下的搜索中只考虑状态的分布，而不考虑其他智能体的策略。状态的概率分布在单智能体的 POMDP 中也被称为信念状态，为了与多智能体的信念状态（考虑其他智能体的策略）相区别，仅考虑状态分布的信念状态也被称为联合信念状态（Joint Belief State)。联合信念比多智能体的信念更加的简单，从而也更容易生成。为了提高效果使得生成的信念状态更具有代表性，MBDP 实用组合启发函数（Heuristic Portfolio）来生成联合的信念。为了避免策略树随求解步数指数式的增长，MBDP 在动态规划的每一步迭代中，只生成固定个数的联合信念状态，用常数 $maxTree$ 来表示，然后每一步的生成的策略树中只保留那些在联合信念点上最好的策略树。这样每一步预留的策略树的个数都是固定的，在下一步的备份操作中，只有有限的策略子树可以选择，所以生成的新的

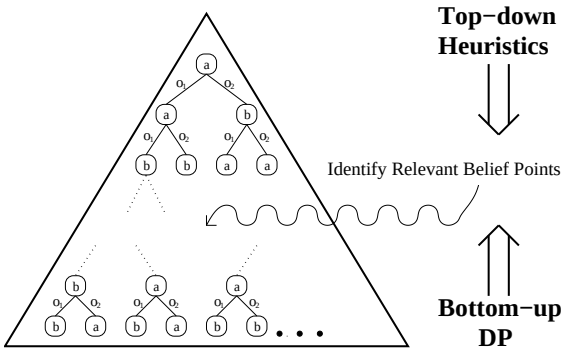

图 2.3 MBDP 算法的求解思想[15]

策略树也是固定的。这一做法背后的观察是：很多保留的策略子树在最后都被证明是没有用处的，所以在迭代早期被删除可以极大的提高算法的效率；此外，通过组合启发函数，可以选定那些具有代表性的策略子树，这些策略子树在下一步的策略生成中可以被重用。而在现实中确实存在这样的情况，比如积木，就是通过有限子模块的组合可以近似的完成各种大的模块。在这里，每个子策略树相当于完成一定的子任务的模块。

在实际问题中，MBDP 表现的非常的好，这也带动了实用性 DEC-POMDP 近似算法的研究。在此之后，有许多基于 MBDP 思想的算法被提出。在 MBDP 中，从 t 步策略树生成 $t+1$ 策略树的过程采用的依然是完全备份（Exhaustive Backup），也就是枚举所有可能生成的 $t+1$ 步策略树。根据式子 2.4，所有可能的 $t+1$ 策略树依然是观察数的指数，如此多的策略树要被生成出来并对每个联合策略进行评价计算量将会是非常大的。在改进算法 IMBDP[16] 中，部分备份（Partial Backup）取代了完全备份。IMBDP 首先对每个智能体在给定联合信念的情况下，计算每个观察可能出现的概率，然后只选择 $maxObs$ 个最可能出现的观察的分支进行备份。对于其他观察分支采取的是例如爬山法等局部搜索的方法赋予一个最好的子策略树。通过这个方法可以限制生成策略树的指数暴炸，因为 $maxObs$ 要比 $|\Omega_i|$ 要小的多，所以算法的速度提高很多。但由于选择可能出现观察的方法太过武断，所以很多好的策略树可能不会被生成。另一个改进算法 MBDP-OC[17] 采用了相对精细的方法来压缩备份中观察的数目。MBDP-OC 采用的是对相似观察分支进行合并的方法来减少所要考虑的观察个数，对于观察分支的相似性判断是基于分支合并前和合并后策略值函数的损失来判断的。只要这个损失足够的小，就可以在保证质量的前提下，尽可能少的考虑需要备份的观察。但这个方法需要计算策略的值函数，所以算法的时间消耗较大。而且这两个算法只能缓解策略树的指数爆炸，而不能从根本上解决问题。

通过对于 MBDP 策略删减方法的进一步分析可以发现，很多策略最终不

会被保留，所以也没有必要生成和进行评价。如果能在策略生成的过程中只构造那些可能会被选上的策略树，那就可以大大的改进算法的效率。正是基于这样的考虑提出了基于信念点的增量式策略删减（Point-Based Incremental Pruning）算法，简称 PBIP 算法[18]。它通过在联合策略空间中构建分支定界（Branch-and-Bound）搜索来代替 MBDP 的完全备份操作。在动态规划的每一步迭代中，PBIP 首先计算出新策略值的上界和下界。然后在策略生成过程中，通过策略值的上下界来提前删减不必要的策略。这个上下界可以通过启发式函数来进行计算，PBIP 算法能够提前去除策略值在上下界之外的策略。但是由于没有考虑到状态的可达性，因此在策略生产的时候依然要考虑各种可能子策略。增量式策略生成方法被提出并作为 PBIP 算法的重要补充，构成了 PBIP-IPG 算法[19]。该算法的重要发现是一个智能体的动作和获得的观察能够确定一个可能的状态分布，而无需考虑其他智能体的行为。因此在选择子策略的时候，可以通过这种状态可达性的分析来筛选子策略。PBIP-IPG 算法首先为每个智能体的动作观察队构建一个可达的子策略集，然后通过这个子策略集来生产新的策略。和 PBIP 一样，在生成新策略的时候，它也通过计算上下界来逐步删除冗余的策略，从而提高策略生成的效率。但即便是 PBIP-IPG 算法也没有彻底的改变生成的策略太多的问题，在最坏情况下其增长依然是指数式的。从本质上说，PBIP-IPG 沿袭的依然是 MBDP 算法先生成策略再进行策略选择的求解模式。

尽管 DEC-POMDP 的近似算法在可求解的问题规模上相比于最优化算法有了很大的提高。但对于较大规模的问题，以上提到的近似算法依然无能为力，主要的原因还是在每一步迭代的过程中生成的策略树个数太多。而事实上，这些算法生成的策略树与实际可能存在的策略树相比，已经是非常的少了。一方面组合启发函数并不能准确的定位那些有用的策略树，另一方面有用的策略树个数本身就可以有很多。所以要提高问题求解的质量，最直接的方法就是提高 $maxTree$ 的值。但提高 $maxTree$ 的值反过来又有可能导致生成的策略树的个数大规模的增长。本文提出的离线规划算法主要想解决的就是这些 DEC-POMDP 的焦点问题。从问题的本质上说，巨大的计算消耗可能来源于离线规划本身具有的局限性，因为在离线规划中问题求解器需要考虑所有的决策步骤，以及在这些决策步骤中所有可能出现的情况。而在实际的多智能体一次运行过程中，所遇到的情况肯定是有限的，而且也没有必要一次性的规划好所有的决策步骤。DEC-POMDP 在线规划的主要思想就是来源于这些基本的观察。

2.3　在线规划及受限通讯的一般解决方案

在线规划的基本原理是智能体在决策周期内一边进行规划一边执行动作。其主要优势在于，智能体无需考虑所有的可能，而仅仅需要对当前遇到的情况进行规划，这就大大的缩小了所需要规划的策略空间。例如在机器人足球中，如果当前队伍的比分占优，则只需要考虑巩固成绩的同时如何扩大战果，无须像比分落后的时候一样，孤注一掷进行破釜沉舟式的拼搏；而在离线规划中，需要同时考虑两种情况。在线规划的另一个好处在于它可以随时改变模型或者规划算法来处理一些突发事件，而离线规划中的联合策略是事先计算好的，在执行的时候无法改变。相比于离线规划，在线规划的主要劣势在于用于规划的时间往往非常的有限，因为许多多智能体系统每个决策的周期都非常的短，有些甚至是实时控制。比如在仿真机器人足球问题中，每个决策周期只有大约100 毫秒的时间。在这么短的时间进行大规模的精细规划显然是不显示的。而离线规划的时间则没有太多的限制，根据应用背景可以是一天、一个月、甚至一年。所以在线规划算法一般结合一些离线计算的启发式信息进行快速的规划。

2.3.1　在线协调机制

除了时间限制外，DEC-POMDP 的在线规划算法还需要考虑多智能体之间的协调与合作。因为在 DEC-POMDP 中，每个智能体获得的信息是各自不同的局部信息，如何根据这些不同的局部观察信息进行协作是多智能体在线规划算法需要解决的主要问题。例如在机器人足球中，因为视觉定位的误差，甲机器人根据它获得的信息觉得乙机器人离球近一些，应该由乙去抢球；而乙根据它的信息又觉得甲离球更近，应该由甲去抢。由于甲乙之间不能通讯和协商，所以造成的结果是甲乙两个机器人都不去抢球。而实际的情况是甲和乙离球都差不多的近，由甲或者乙去抢球都是可以接受的，但实际的结果却是甲和乙谁都不去抢球这个糟糕的结果。这就是多智能体系统在线规划中可能出现误协调（Mis-Coordination）的一个例子。根据 DEC-POMDP 的问题设定，每个智能体获得的都是不同的局部信息，根据这些不同的局部信息进行规划就有可能产出误协调。在实际问题中，误协调的后果往往是不可预测的，可能对系统造成非常严重的后果。在多智能体的在线规划中，由于时间的限制，要求最优化决策往往不现实。所以如何很好的协调智能体之间的行动，产生出好的团队决策就是多智能体在线规划的基本要求。

避免误协调最根本的解决办法就是在进行规划的时候考虑其他智能体可能采取的策略。由于在一般 DEC-POMDP 问题的设定中，智能体之间不能通讯和

协商，所以在执行的过程中无法直接获得其他智能体的实际策略。因此，在线规划的时候只能对其他智能体的行为进行大致的预测。进行预测的方式大体可以分为两种，一种是考虑其他智能体决策的最坏可能情况；另一种是考虑其他智能体决策的平均可能情况。最坏情况假设是一种非常强的假设，也就是无论其他智能体的决策有多么糟糕，它都能尽自己最大的可能来弥补；同时这也是一个非常悲观的假设，多数情况下其他智能体的行为都不遵循这个假设。例如在上面提到的两个机器人抢球的例子中，按照最坏情况假设，甲乙都会同时去抢球。随着甲乙两个机器人与球距离的减小，它们的信息也会越来越准确，最终会有一个放弃而另一个更适合抢球的机器人继续抢球。在这个例子中，甲乙中可能有一个机器人因为不必要的移动产生一些消耗，但毕竟有机器人去抢球，这个消耗和误协调比起来是非常小的。由此可见，最坏情况假设也有一定的合理性，它能以少量的代价最大限度的避免误协调带来的灾难。在平均可能假设中，通过一定的方式保证甲乙两个机器人有等同概率去抢球，所有不会出现同时抢球的情况。在该假设中，智能体会忽略一些局部的不一致信息，用等概率的方式取代该信息，从而达到信息的一致性。这么做可能出现的后果是，可能导致离球较远一些的机器人去抢球，但通常也不会远太多，所以带来的额外消耗比起误协调来都是可以接受的。

序列贝叶斯博弈近似（BaGA）算法[20] 是典型的最坏假设在线规划算法。BaGA 将 DEC-POMDP 问题分解成一系列小的贝叶斯博弈问题，通过求解这些小的贝叶斯博弈获得每一步的策略。而每一个小的贝叶斯博弈则通过轮换最大化（Alternating-Maximization）方法来进行求解。为了使得每个智能体能够有足够的信息来独立的构建相同的博弈问题，算法为每个智能体维护了一个类型（Type）空间，空间中的信息是关于智能体可能获得的历史观察序列。在进行动作选择上，BaGA 算法考虑的是最坏情况下其他智能体的可能行为。这个最坏情况包含了两层意思：首先它可能被执行的概率最大，再者它被执行后获得的收益最小。因此，BaGA 的策略是趋向于保守的策略，也即是考虑到其他智能体可能的最坏情况，然后选择对整个团队最安全的行动。

可能联合信念推理（Dec-Comm) 算法[21] 是典型的平均假设在线规划算法。在每一步的决策中，Dec-Comm 首先维护一个可能的联合信念集合，然后计算这个联合信念集合构成的平均状态分布。利用事先离线计算好的 Q_{POMDP} 启发式函数，每个智能体就可以通过这个平均状态分布来计算一个期望值最高的行动。很显然，在选择动作的时候 Dec-Comm 考虑的平均情况。也就是说在某些情况下，Dec-Comm 的动作选择可能会过于乐观，因为实际的情况可能要比平均情况要差。为了避免智能体之间的误协调，在 Dec-Comm 算法的信念更新中尽可能少的使用智能体当前获得的局部信息。由于在 DEC-POMDP 问题的设定

中，每个智能体获得观察是不一样的，而且智能体也不能得知其他智能体的观察。所以只有在尽量少使用局部信息的情况下才能保证每个智能体计算出来的平均状态分布能选择出相同的联合行动。在使用局部信息的方法上，Dec-Comm 算法利用到了智能体之间的通讯。

2.3.2　在线通讯策略

智能体间的通讯（Communication）是多智能体系统决策和规划的一个焦点问题。智能体可以通过彼此间的通讯进行信息共享和策略协商。虽然 DEC-POMDP 通过动作观察的方式隐式的表达了智能体间的通讯，但这种间接的通讯方式是模型建立的时候就已经确定好的，无法在问题求解的时候进行决策通讯。所谓的决策通讯就是对智能体信念和策略等相关决策信息进行通讯，智能体间的通讯往往与智能体自身所采用的决策算法有关。决策通讯对 DEC-POMDP 在线规划的作用是不言而喻的，因为显式的通讯提供了智能体之间交换决策信息的一个快速通道。在没有决策通讯的时候，智能体之间只能通过推理大概的估计其他智能体的决策行为。应用了决策通讯后，智能体可以直接并准确的从其他智能体获得它感兴趣决策信息，这就大大简化了多智能体的决策过程。事实上，在通讯没有限制的多智能体系统中，所有的智能体可以共享各自的观察信息，分布式控制的 DEC-POMDP 变为了集中式控制的 POMDP，问题复杂度也从 NEXP 变为 NSPACE。在实际问题中，智能体间的通讯往往受到物理条件的限制，所以通讯对决策而言是有代价的。例如在地下或者其他星球上工作的机器人，可能需要移动到特殊的部位才能够互相通讯。即使能够通讯，由于通讯媒介的限制，可能存在信息丢失需要不断的重传。即使能够进行可靠的通讯，由于带宽的限制也不可能一次性传递太多的信息，加上通讯的距离可能比较远，需要较多的时间。即使在物理媒介近乎完美的室内，通讯仍然要消耗大量的计算资源，导致机器人反应迟钝。

由于通讯的种种限制，在线规划算法在利用通讯时需要解决一下几个问题：一、如何在不能通讯的情况下也能进行决策；二、如何在能通讯时只在必要时进行通讯；三、在通讯时如何最小化通讯量。这也体现了在线规划和智能体的协商在利用通讯上的差别，前者旨在利用通讯提高决策的质量，而后者需要智能体间进行反复的通讯协商才能够获得策略。所以在线规划算法往往利用通讯来进行信息共享，而不是用来生成策略。共享的信息往往是各自动作和观察的历史信息，因为在 DEC-POMDP 模型中这些是每个智能体所拥有的最原始的私有信息。在通讯方式上，可以是广播式（Broadcast）的通讯，也可以是点对点（Peer-to-Peer）的通讯。其中广播式的通讯较为简单且满足信息的一致性，主要

是一个智能体发出一个信息，其他所有的智能体同时收到该信息。点对点的通讯比较复杂，需要考虑跟谁进行通讯，而且通讯以后只有部分信息发生变化，决策过程较为复杂。DEC-POMDP 是完全合作的问题，智能体之间没有隐私可言，所以通常只采用较为简单的广播式通讯。广播式的通讯中还分为主动通讯和被动通讯，主动通讯是广播者自动发起通讯，而被动通讯是其他智能体请求广播者进行通讯。

通讯策略中，最为简单的是在通讯可用时，智能体间进行定时的信息广播。定时广播的优点在于实现起来非常的简单，无需考虑智能体的决策过程，这同时也是它的主要缺陷。因为没有考虑到决策过程中对信息的需求，定时广播可能会浪费掉一些宝贵的通讯资源。如果决策过程严格依赖定时通讯，那么在通讯资源不可用的情况下就无法做出决策。考虑决策过程的通讯策略一般可以分为两种，一种是基于策略的结构，另一种是基于策略的值评价。这类通讯决策的基本原理就是预测通讯前和通讯后求解出的策略之间的差别。例如在 Dec-Comm 算法中，智能体先根据没有通讯的情况计算出一套策略，然后再假设有通讯的情况计算出另一套的策略。如果两个策略有差别就说明通讯能导致策略的改变，而通讯后求解的策略一般会更好，则说明这个通讯是有价值的。如果通讯前后的策略一致，则说明当前对这个智能体的决策来说没有通讯的必要，自然就无需浪费宝贵的通讯资源。但是，策略的不同并不代表策略的效果不同。存在两个结构不同的策略，其值评价是完全相同的，所以更好的办法不是比较策略的结构而是比较前后的策略值。使用策略的值评价的另一个特点就是可以量化通讯的代价，通常通讯后的策略值减去通讯前的策略值，再减去相应的通讯代价，得到的差值也被称为通讯的信息值（Value of Information）[22]。使用通讯的信息值可以更好的衡量一次通讯对于决策的价值，因此能够更好的利用有限的通讯资源。这种考虑通讯对于策略的差别的办法主要的问题在于无法准确的评估通讯后的策略，因为通讯决策是在实际通讯发生前做出的，无法确切的知道通讯时能获得什么样的信息，所以对于通讯后策略的评估只能是一个预测。而这种预测通常都无法很准确，从而无法准确的体现通讯的价值。

前面提到过在在线规划算法中，通讯的作用通常是共享信息而不是用于策略协商。所以考虑通讯对策略的影响来评价通讯的价值其实是一种间接的手段，更直接的手段是考虑通讯对于信息更新的价值。例如在 dec_POMDP_Valued_Com[23] 中，考虑的就是智能体信念的改变（Belief Divergence），而把通讯看成是对智能体信念进行修正的一个手段。具体做法是在某个决策周期设立一个信念状态的参考点，在随后的几个决策周期中，假定其他智能体不获得信息的前提下，根据智能体自己的信息更新信念状态。如果当前自己的信念状态与参考点的 KL 距离超过一定的阈值，则说明在这几个决策周

期中智能体的信息发生了较大的变化，需要通过通讯来同步变化后的信息。该方法与基于策略的通讯方法不同，直接的考虑了智能体信息对于通讯的需求以及通讯对于智能体信息的影响，因此在通讯的使用上更加有效。其重要缺陷在于，对于其他智能体不获得新信息的假设一般是不成立的，所以对实际通讯的需求会产生相应的误判断。

2.4　本章小结

本章给出了 DEC-POMDP 的数学定义，并系统的回顾了基于 DEC-POMDP 模型的一些经典的离线和在线算法。从决策论的角度，DEC-POMDP 是单智能体的 POMDP 模型在多智能体系统的上的扩展。而从博弈论的角度，DEC-POMDP 则是 POSG 模型在智能体之间完全合作情况下的特例。从问题复杂度上看，DEC-POMDP 是 NEXP 难问题，因此精确算法只能求解极其小的问题。从模型表现力上看，DEC-POMDP 中包含了多智能体合作问题所需的所有决策因素。其中还隐式的表达了智能体之间的通讯，因为一个智能体的动作会影响到另一个智能体的观察，所以可以通过相应的动作发送信息；而另一个智能体也可以通过解析特定的观察来进行信息接收。由于大规模的 DEC-POMDP 问题无法精确的求解，所以对于 DEC-POMDP 模型的研究主要分为两类：一、从现实问题中找到更容易求解的 DEC-POMDP 子问题；二、设计高效的近似算法来求解一般性的问题。由于本文主要关注的是设计近似算法求解一般问题，所以在背景介绍中着重介绍相关的近似算法。

从求解的方式上说，近似算法由可以分为离线算法和在线算法两类。离线算法是在离线状态下计算出一个完整的策略，也就是通常所说的规划阶段。对于 DEC-POMDP 模型，离线算法的主要要求是计算出的策略可以由各个智能体分布式的执行。在离线算法中主要用于表示策略的方法是策略树，树的节点是要执行的动作，而树的边表示的是获得的观察。在执行的时候，先执行节点中的动作，然后根据从环境获得的观察选择对于边，然后执行该边对应的下一个节点，这样不断的执行下去直到到达叶子节点。所以树的高度等于问题的决策总步数。离线算法的主要问题在于相关的联合策略空间非常的大，具体到策略树，就是可能的策略树的个数随着决策步数呈双指数式的增长。为了解决这个问题，MBDP 算法通过自底向上的动态规划一步步的构建策略，然后根据自顶向下的启发式函数选择最有价值的子策略树进行保留。通过每一步迭代保留固定数量的子策略，MBDP 算法具有相对于决策步数的线性空间和时间复杂度。但是算法的每一步迭代中产生的策略树个数依然非常的大，成为算法求解更大规模问题的主要瓶颈。这同时也是本文在离线算法部分尝试解决的主要问题。

与离线计算完整的策略不同，在线规划算法只要针对智能体当前遇到的情况进行求解。和完整的策略空间相比，智能体在执行阶段所能遇到的情况是其中非常小的一部分，这也是在线算法相对于离线算法的主要优势所在。但在线算法有自己需要解决的问题，例如每一步规划时间的限制以及分布式决策可能导致的误协调等。在线算法的另一个优势是可以方便的进行智能体之间的通讯，但通讯资源往往是受限的，所以需要进行通讯决策最大限度的发挥通讯资源的效用。这些也是本文在在线算法部分主要关心和解决的问题。

第 3 章　通讯受限的在线规划算法

在线规划可以只针对智能体在实际运行过程中遇到的情况进行求解，而无需像离线规划一样完整的考虑各种可能。但因为智能体需要实时的对环境做出反应，所以用于在线规划的时间往往非常的有限。例如在机器人足球问题中，每一步决策的时间只有 100 毫秒。而且每个智能体在获得不一样的局部信息的情况下要彼此间进行合作，因此多智能体的在线规划还需要考虑智能体间的分布式协调问题。例如足球场上的机器人，每个球员根据自己装备的摄像头观察到的是场上的一个局部，但合作对赢得比赛是必须的，因为足球是团队的游戏。通讯可以在决策过程中共享智能体各自的私有信息，从而提高决策的效率。但在实际问题中，由于带宽、环境等诸多因素的影响，通讯往往是受限的或者是有代价的。也即是说通常不能进行实时的通讯，只能是间歇性的或者存在一定的延迟。

针对这些问题，作者提出一种新型的通讯受限的多智能体在线规划算法(MAOP-COMM)，旨在解决在线规划中智能体间的策略协调和通讯资源的有效利用问题。策略协调的关键是智能体在分布式的环境中能够独立的计算出一套相同的且足够好的联合（团队）策略。例如在机器人足球中，甲球员想给乙球员传球，但由于由于信息不一致，所以乙丙两个球员都互相认为对方该接球。最终的结果可能是没有一个球员去接球，最终球出边线。由于信息不一致在模型中是客观存在的，而且通讯受限不能彼此进行策略协商，所以解决的方法是保证甲乙丙三个球员能够独立的计算出一套相同的接球方案。通常来说，即便计算的出的不是最佳的传接球方案，也比最终无人接球这样的失误要好。在通讯资源的利用上，本章更加关注的是通讯对于决策的影响，而不考虑具体的通讯过程。也就是说通讯决策的具体目标是尽量减少通讯的次数，而最大限度的提高决策的收益值。在本章中，并没有具体区分规划和决策的具体含义。通常来说，计算一个策略的过程称为规划，而策略的计算和执行这一整个过程都可以称为决策。在本文中，主要关注的是如何计算一个策略，具体策略执行的过程则相对简单。

3.1　保证多智能体策略协调性的规划框架

使用 MAOP-COMM 算法时，每个智能体并行的运行该算法，在每个决策周期内进行策略规划和动作执行。也就是说，在每一步的决策过程中，MAOP-COMM 可以具体的分为策略规划（Planning)、动作执行（Executing）和信息更

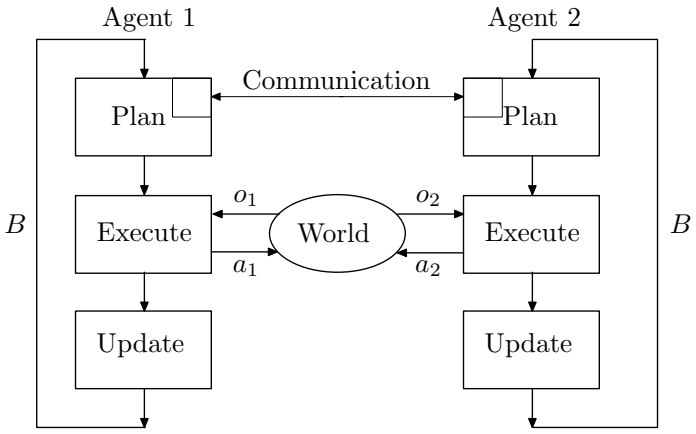

图 3.1 两智能体的在线规划流程

新（Updating）三个阶段。图 3.1给出了两个智能体在线规划的流程。

定义 3.1.1 (信念池)**.** 在决策周期 t 中，智能体的信念池可以定义为一个二元组 $\langle\{H_i^t|i\in I\}, B^t\rangle$，其中 H_i^t 是智能体 i 的可能动作观察历史序列集，而 B^t 是这些动作观察历史相对于的联合信念状态集，即 $B^t=\{b(h^t)|h^t\in H^t, H^t=\times_{i\in I}H_i^t\}$。

从本质上说，信念池是表示当前系统信息的一个数据结构。例如在机器人足球中，每个防守队员需要盯防对方的进攻队员。那防守队员的信念池中包含的信息就可能是离每个防守队员最近的一个进攻对手信息。只要每个智能体在线规划的过程中维护的相同的信念池，那么根据这个信念池求解出的联合策略也是相同的。也就是说只要每个防守队员关于最近进攻队员的信息一致，那么就能在线对防守人员进行统一的分配而不会出现误协调的情况。

定义 3.1.2 (局部和联合策略)**.** 智能体 i 的局部策略是从动作观察的历史信息到动作集的一个映射，$\delta_i: H_i\to A_i$，其中 $\delta_i(h_i)$ 表示在策略 δ_i 上历史信息 h_i 导出的动作。一个联合策略是所有智能体局部策略的一个多元组 $\delta=\langle\delta_1,\delta_2,\cdots,\delta_n\rangle$，其中 $\delta(h)$ 表示的是在联合策略 δ 上由联合历史 h 导出的一个联合动作。

在 MAOP-COMM 算法的规划阶段，每个智能体对信念池中的每个可能的历史信息计算一个联合策略 δ^t。在执行阶段，智能体 i 首先根据获得的观察更新自己的局部历史信息，$h_i^t\leftarrow h_i^{t-1}\circ o_i^t$，然后根据更新后的局部历史信息和前面计算出的联合策略执行动作 $a_i=\delta_i^t(h_i^t)$，最后再将执行过的动作加入到自己的局部历史信息中。这样以后，每个智能体根据联合策略更新自己信念池中的信息，具体的更新过程如算法 3.1所示，然后进入下一个决策周期进行相应的规划执行过程。

定义 3.1.3 (智能体间的协调)**.** 当所有的智能体分布式的计算出一个相同的联合策略，并一直遵循该联合策略执行相应动作时，我们称智能体之间的行为是协调的，反之称为误协调。

算法 3.1　历史展开和信念更新

Input: H^t, B^t, δ^t
$H^{t+1} \leftarrow \emptyset$; $B^{t+1} \leftarrow \emptyset$
for $\forall h^t \in H^t, \forall \vec{o} \in \vec{\Omega}$ **do**
  $\vec{a} \leftarrow \delta^t(h^t)$
  // append $\vec{a}, \vec{o}$ to the end of h^t.
  $h^{t+1} \leftarrow h^t \circ \vec{a} \circ \vec{o}$
  // calculate the distribution of h^{t+1}.
  $p(h^{t+1}) \leftarrow p(\vec{o}, \vec{a}|h^t)p(h^t)$
  // test if h^{t+1} is a reachable joint history.
  if $p(h^{t+1}) > 0$ **then**
    $H^{t+1} \leftarrow H^{t+1} \cup \{h^{t+1}\}$
    // compute the belief state of h^{t+1}.
    $b^{t+1}(\cdot|h^{t+1}) \leftarrow$ Update belief with $b^t(\cdot|h^t), \vec{a}, \vec{o}$
    // add b^{t+1} into a hash table indexed by h^{t+1}.
    $B^{t+1} \leftarrow B^{t+1} \cup \{b^{t+1}(h^{t+1})\}$
return H^{t+1}, B^{t+1}

前文提到，鉴于误协调可能产生的严重后果，多智能体系统规划算法的基本目标之一就是协调智能体之间的行为。在离线算法中，分布式执行离线计算好的统一联合策略，从而保证了智能体之间行为的协调。而在在线算法中，保证智能体行为的协调却要困难的多。因为在 DEC-POMDP 中，每个智能体获得的是不同的局部观察信息，如何在利用这种不一致信息的同时，保持智能体行为的协调是所有在线算法需要解决的首要问题。由于联合策略是实时在线计算的，需要一种机制保证计算出来的联合策略不会导致智能体之间的误协调。

在 MAOP-COMM 算法中，每个智能体维护的是一个相同的关于全体队员的信念池。这就保证了每个智能体能分布式的计算出一个相同的联合策略，从而保证了它们行为的协调性。虽然算法中采用了某些随机的方式进行策略生成，但算法在生成这些伪随机数的时候采用了完全相同的随机种子，这就保证了这些步骤拥有完全相同随机行为。在这里需要再次强调的是，每个智能体在进行规划的时候都是基于完全相同的前提信息。给定相同的信念池和相同的随机行为，每个智能体就能够计算出一个完全相同的联合策略。而每个智能体所获得的局部观察只用于联合策略的执行。

定义 3.1.4 (信念不一致性)**.** *当一个智能体的信念可以推导出 p 为真，而它所获得的观察信息却蕴含 $\neg p$ 时，我们称智能体的信念存在不一致性。*

从直观上理解，信念不一致性表示的是智能体的信念与智能体从环境中获得的观察之间存在矛盾的状态。例如在机器人足球中，防守队员甲发现实际离它最近的对手，也就是它需要盯防的对手与保存在信念中的信息不一致。该队

员就需要与其他的防守队员进行通讯，告知这一新的信息，并让其他的队员更新它们的信念。因为在算法中每个队员维护的是一个相同的信念信息，如果甲获得的新信息与甲的信念不一致，那同时说明了该信息也与其他队员的信念不一致，整个团队的信念信息存在滞后。因此，用通讯的方法解决信念信息的滞后可以让整个团队的规划基于更加准确和及时的信息，从而获得更好的联合策略。这同时也体现了通讯对于智能体决策的价值。

MAOP-COMM 中的通讯策略正是基于这样一个崭新的观察。每个智能体在决策周期开始时，利用自己新获得的观察检测信念池中信念的不一致性。一旦发现不一致且当前通讯资源可用，则由该智能体发起通讯。通讯的方式是该智能体将自己新获得的观察信息广播给全队。这样其他队员就可以根据广播后的信息更新自己的信息池，然后利用这个更新后的信息池进行规划，并计算出一个联合策略。在该决策周期末，每个智能体再利用自己获得的观察来执行该联合策略，并最后根据联合策略再次更新信息池中的信息。

注意到，通讯对于 MAOP-COMM 来说并不是保证团队协作的根本要素，而是一种用于提高团队协作效果的可选方式。这就使得通讯的使用可以更加的灵活，特别是对于一些领域中，通讯资源并不是一直可获得的情况。通讯的过程可以很方便的整合到给出的在线决策的框架中，只需要将其放置在规划阶段开始前作为更新信念池的一个额外组件。无论有无通讯，智能体之间的团队协作都是依然可以得到保证的，因为通讯给出的信息是公共的，每个智能体利用这一公共的信息对信息池进行相同的更新。换句话说，每个智能体所维护的信念池无论通讯有无都将是相同的。广播式的通讯保证了每个智能体在通讯过程中获得的是相同的通讯信息。

定理 3.1.1. *受限通讯的多智能体在线规划算法（MAOP-COMM）无论在有无通讯的情况下都可以保证智能体之间的协调。*

证明. MAOP-COMM 提供了多种机制保证了每个智能体能够维护一个相同的信念池，并能根据这个信念池在线计算出相同的联合策略，从而确保了智能体之间行为的协调一致性。首先用于更新信念池的信息都是整个团队所共知的信息。更新信息池的方式有两个：一、每个周期末用来自每个智能体计算出来的联合策略进行信念池的信息更新；二、在需要并可以通讯的情况下，周期开始时用通讯获得的信息对信念池进行更新。

在没有通讯的情况下，联合策略的一致性可以用递归的方式来证明。在决策开始时，信念池只包含模型给出的初始状态分布 b^0，所以这时候的信念池是相同的，计算出的联合策略也必然相同。根据相同的联合策略进行相同的更新后，所得到得信念池还是相同的。因此相同的信念池保证计算出相同的联合策

略，相同的联合策略保证更新出相同的信念池。

在存在通讯的情况下，通讯所获得的信息只在规划开始前对信念池进行更新。所以这个更新过程即不会影响到规划和执行过程，也不会影响到联合策略对于信念池的更新过程。也就是说，该过程只影响到信念池中的内容，只要对于所有智能体来说，更新以后还是相同的，则不会对联合策略的一致性产生影响。由于在 MAOP-COMM 算法中，每个智能体采用的是广播式的通讯，所以通讯以后，每个智能体获得的通讯信息是相同的。利用相同的通讯信息对信念池进行相同方式的更新，更新后的信念池对于所有的智能体来说还是相同的。因此计算出来的联合策略还都是相同的。

对于规划和更新过程中，MAOP-COMM 所采用的伪随机数。由于在每一步的决策开始前，每个智能体使用的是预先生成好的随机种子，所以即使是在分布式的情况下，每个智能体所生成的伪随机数还都是相同的，所以不影响规划和更新过程的一致性。综上所述，无论有无通讯，MAOP-COMM 算法都能保证每个智能体分布式的计算出相同的联合策略，从而保证智能体之间的协调。 □

在开始运行 MAOP-COMM 前，每个智能体将其信念池初始化为 b^0 和一个空历史信息，并根据这个信念池计算出相应的联合策略。在第一步获得相应的观察后，执行计算好的联合策略并进行信念池更新，而后按照 MAOP-COMM 设定的几个阶段周而复始的执行，算法 3.2给出了主要步骤的伪代码。在 MAOP-COMM 算法的设计当中，需要解决三个主要问题：一、如何根据已知的信念和历史信息计算出一个分布式的策略本质上是一个 NP 难的问题；二、信念池中的联合历史和信念状态信息可能随着时间指数式的增长，从而变得极其巨大；三、如何根据智能体获得的局部信息进行信念不一致性的检测一样是一个复杂的问题。在算法中，MAOP-COMM 检测信念不一致性的方法。在算法中，MAOP-COMM 利用一些列的线性规划来对策略进行近似求解，并提出了一种基于策略相似性的历史信息归并方法来压缩信念池，同时设计出了一套定量的方法来检测信念的不一致性。在对上面提到的每一点进行详细的论述之前，需要对将用到的一些概念进行形式化的定义：

定义 3.1.5 (历史信息). *在决策周期 t，智能体 i 的局部历史 h_i^t 定义为其执行过的动作和所获得的观察的一个序列，即：$h_i^t = (a_i^0, o_i^1, a_i^1, \cdots, o_i^{t-1}, a_i^{t-1}, o_i^t)$。而联合历史则为包含每个智能体的局部历史的一个多元组 $h^t = \langle h_1^t, h_2^t, \cdots, h_n^t \rangle$。*

定义 3.1.6 (联合信念). *联合信念 $b(\cdot|h) \in \Delta(S)$ 表示的是由联合历史 h 导出的关*

算法 3.2　通讯受限的在线规划

Input: b^0, $seed[1..T-1]$
foreach $i \in I$ (in parallel) **do**
  $\vec{a}^0 \leftarrow \arg\max_{\vec{a}} Q(\vec{a}, b^0)$
  Execute the action a_i^0 and initialize h_i^0
  $H^0 \leftarrow \{\vec{a}^0\};\ B^0 \leftarrow \{b^0\};\ \tau_{comm} \leftarrow false$
  for $t = 1$ ***to*** $T-1$ **do**
    Set the same random seed by $seed[t]$
    $H^t, B^t \leftarrow$ Expand histories and beliefs in H^{t-1}, B^{t-1}
    $o_i^t \leftarrow$ Get the observation from the environment
    $h_i^t \leftarrow$ Update agent i's own local history with o_i^t
    if H^t *is inconsistent with* o_i^t **then**
      $\tau_{comm} \leftarrow true$
    if $\tau_{comm} = true$ **and** *communication available* **then**
      Synch h_i^t with other agents
      $\tau_{comm} \leftarrow false$
    if *agents communicated* **then**
      $h^t \leftarrow$ Construct the communicated joint history
      $b^t(h^t) \leftarrow$ Calculate the joint belief state for h^t
      $\vec{a}^t \leftarrow \arg\max_{\vec{a}} Q(\vec{a}, b^t(h^t))$
      $H^t \leftarrow \{h^t\};\ B^t \leftarrow \{b^t(h^t)\}$
    else
      $\pi^t \leftarrow$ Search the stochastic policy for H^t, B^t
      $a_i^t \leftarrow$ Select an action according to $\pi^t(a_i|h_i^t)$
      $H^t, B^t \leftarrow$ Merge histories based on π^t
    $h_i^t \leftarrow$ Update agent i's own local history with a_i^t
    Execute the action a_i^t

于状态的概率分布。该状态分布可根据贝叶斯公式递归的计算出来：

$$\forall_{s' \in S}\ b^t(s'|h^t) = \alpha O(\vec{o}^t|s', \vec{a}^{t-1}) \sum_{s \in S} P(s'|s, \vec{a}^{t-1}) b^{t-1}(s|h^{t-1}) \tag{3.1}$$

其中，α 是归一化因子。在算法中，联合信念 $b(\cdot|h)$ 也常常简化的表示为 $b(h)$。

3.2　基于线性规划的快速在线策略求解

在 DEC-POMDP 中，每个智能体只能利用自己的传感器获得环境的局部信息。为了与其他智能体进行协作，每个智能体必须对其他智能体可以的信念状态进行预测，并考虑这些信念状态对自己的行为选择可能产生的影响。具体的说，智能体 i 在计算历史 h_i 对于的策略 q_i 时，需要考虑到其他智能体所可能持

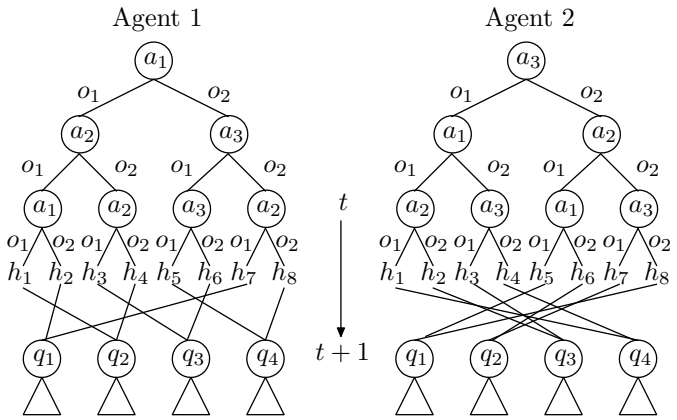

图 3.2 在线规与离线规划的相似性

有的历史 h_{-i} 以及这些历史所对应的策略。也即是说，MAOP-COMM 在规划的时候需要计算出一个联合策略 δ，使其能最大化下列的值函数：

$$V(\delta) = \sum_{h \in H} \sum_{s \in S} p(s|h) V(\delta(h), s) \tag{3.2}$$

其中 $p(s|h)$ 表示的是由联合历史 h 导出的状态分布。

值得注意的是，算法计算出的联合策略是一个完全的分布式策略，也就是说在这个策略中，每个智能体能够仅仅根据它所获得的局部观察进行动作的选择。在文中，计算出的联合策略表示为只依赖局部历史信息的局部策略的一个组合，即：$\delta(h) = \langle \delta_1(h_1), \delta_2(h_2), \cdots, \delta_n(h_n) \rangle$。而 MAOP-COMM 的目标是每个智能体独立的计算出这样一个相同的联合策略 $\delta(h)$，并在计算过程中充分的考虑其他智能体所可能采取的策略。有了这样一个联合策略，每个智能体就可以根据其实际获得的局部历史信息选择一个动作 $a_i = \delta_i(h_i)$，并在线的执行该动作。计算分布式策略的主要优势在于，在规划阶段可以在考虑其他智能体策略的前提下计算出统一的策略，从而保证智能体间的协调；而在执行阶段，每个智能体又可以根据自己所获得的局部观察信息进行相应的动作执行。也就是说，MAOP-COMM 的策略计算在利用了智能体所获得的局部信息的同时，很好的保证了智能体间行为的协调一致性。

事实上，MAOP-COMM 的分布式策略计算等同于 DEC-POMDP 中的一步离线规划，即求解 $T = 1$ 的 DEC-POMDP，而基于历史的策略表示也等价于离线规划中的一步策略树。在线规划的信念池的信念状态点可以当成这个一步 DEC-POMDP 问题的初始状态分布 b^0，所以一步的在线规划等价于求解多个不同初始状态分布的一步 DEC-POMDP 离线规划。在离线规划中，策略往往表示成策略树的形式，在动态规划每一步迭代的目标是为每个分支选择最优的子树，随着求解步数的增多，分支的个数也随指数式的增长。而在在线规划中，历史

信息表示成序列的形式，而不是离线规划中的树状，规划求解的目标是为每个历史信息选择一个最优的行动。显然，序列表示的历史信息比树形表示的分支要少的多，所以在线规划所考虑的情况其实要比离线规划少。

给定一个信念池，用蛮力法求解一个最优的联合策略的方法就是枚举和评估池中的历史信息与可能子策略的所有组合，然后从中选择最好的一个组合便是所求的联合策略。但是由于可能的组合数随着历史信息的增多呈指数式的增长，而池中的历史信息往往是非常多的，蛮力法需要耗费非常多的时间。对于时间有限的在线规划，该方法是行不通的。事实上，求解联合策略的问题等价于分布式决策问题，而该问题已被理论上证明是 NP 难问题[24]。在 MAOP-COMM 算法中，策略被表示成随机策略的方式，这样能够方便的使用线性规划对该问题进行快速的近似求解。

与确定性的策略不同，随机策略是历史到策略集的一个概率分布。$\pi_i(q_i|h_i)$ 表示给定 h_i 时，智能体 i 选择策略 q_i 的概率；与确定性策略类似，可以定义联合随机策略为 $\pi = \langle \pi_1, \pi_2, \cdots, \pi_n \rangle$，即各个智能体局部策略的一个多元组。联合随机策略 π 的值函数可以表示为：

$$V(\pi) = \sum_{h \in H} p(h) \sum_{\vec{q}} \prod_{i \in I} \pi_i(q_i|h_i) Q(\vec{q}, b(h)) \tag{3.3}$$

其中，$p(h)$ 表示联合历史 h 在信念池中出现的概率，$b(h)$ 表示 h 导出的联合信念状态，而 $Q(\vec{q}, b(h))$ 表示联合策略 $\vec{q}$ 在 $b(h)$ 下的值。注意到式子中的 $\vec{q}$ 表示的是从在线规划的当前决策周期开始到所有决策周期完成的一个完整的策略，所以 $Q(\vec{q}, b(h))$ 表示的是在给定状态分布 $b(h)$ 时，执行策略 $\vec{q}$ 将会获得的总期望收益值。问题是在 DEC-POMDP 的动态规划算法中不能够离线的获得最优的 Q 值，主要的原因是获得最优的 Q 值的复杂度等同于完全求解整个 DEC-POMDP 问题。虽然最优的 Q 值未知，但总能找到他的一个近似或者代替它的一个启发式函数。通常情况下，越接近最优值的近似值越难获得但实际效果也会越好，反之亦然。如何选择这样一个 Q 值往往由实际的问题来决定。

在 MAOP-COMM 中，采用的是一步前瞻的方法来近似预测子策略的值。也就是说我们考虑的子策略是一棵只有一个动作节点的策略树。在这个意义上，任何值函数 $V(s)$ 都可以被用来做该子策略值的启发式函数。在理想的情况下，启发式函数不仅要包含当前联合行动的即时回报，同时还要包含未来行为期望回报值。前文提到过，找到一个最优的评价函数是不现实的，因为它的计算量等同于完整求解 DEC-POMDP 问题。一种可能的解决方案是使用 MDP 的值函数，而该 MDP 是由原来的 DEC-POMDP 模型假定所有智能体能完全观察到系

表 3.1 线性规划的策略参数求解

Variables: $\varepsilon, \pi_i(q_i\|h_i)$
Objective: maximize ε
Improvement constraint:
$V(\pi) + \varepsilon \leq \sum_{h\in H} p(h) \sum_{\vec{q}} \pi_i(q_i\|h_i) \prod_{k\neq i} \pi_k(q_k\|h_k) Q(\vec{q}, b(h))$
Probability constraints:
$\forall h_i \in H_i, \sum_{q_i} \pi_i(q_i\|h_i) = 1$
$\forall h_i \in H_i, q_i, \pi_i(q_i\|h_i) \geq 0$

统状态的情况下诱导出来的。对于一步前瞻，策略 $q_i, \vec{q}$ 可以简化为动作 $a_i, \vec{a}$。而 Q_{MDP} 启发式函数可以表示为：

$$Q(b, \vec{a}) = \sum_{s\in S} b(s) \left[R(s, \vec{a}) + \sum_{s'\in S} P(s'|s, \vec{a}) V_{\text{MDP}}(s') \right] \tag{3.4}$$

其中 V_{MDP} 是上面提到的 MDP 问题的值函数。由于这个 MDP 假设所有的智能体能够完全获得系统的状态信息，所有由它计算出的 Q_{MDP} 的值是最优 Q 值的上界。Q_{POMDP} 启发函数是更加紧致的上界，该 POMDP 由原来的 DEC-POMDP 模型在假设所有智能体都能获得联合观察的假设下诱导得出的。当考虑到具体实际问题时，关于问题的领域知识也可以用来作为值评价饿启发式函数。例如在机器人足球问题中，可以假定越靠近对方球门的位置具有越高的期望回报。在 MAOP-COMM 的实验部分，采用的是较为简单的 Q_{MDP} 作为子策略的近似评价，因为这些测试问题的 MDP 的最优值函数能够被快速的求解。事实上，一步前瞻的启发式函数可以被扩展为多步前瞻，但对多智能体的规划问题进行这样的扩展并不简单。在为一个在线规划算法，MAOP-COMM 的研究重点在于如何快速的求解出策略，而对启发式函数的选择更加的依赖实际问题。

在给定了启发式函数之后，MAOP-COMM 算法就通过构造和求解一系列的线性规划来获得近似最优的联合策略。在算法进行初始化的时候，每个智能体的局部随机策略 π_i 设置为随机的确定性策略，也就是在动作选则上每次均匀随机的选取一个动作，并将该动作的概率值设为 1 而其他动作的概率值全部设置为 0。当然，也可以按照其他的方式来初始化随机策略，因为算法的收敛并不依赖初始策略的构造。然后，轮流的选择一个智能体，在假定其他智能体的策略不变的情况下，对该智能体的策略进行改进。假定这个被选定的智能体编号

为 i，改进其策略的方法就是求解满足以下不等式的最优参数 $\pi_i(q_i|h_i)$：

$$V(\pi) \leq \sum_{h\in H} p(h) \left[\sum_{\vec{q}} \pi_i(q_i|h_i)\pi_{-i}(q_{-i}|h_{-i})Q(\vec{q}, b(h)) \right] \tag{3.5}$$

其中 $\pi_{-i}(q_{-i}|h_{-i}) = \prod_{k\neq i} \pi_k(q_k|h_k)$。表 3.1中列出了用于改进策略 π_i 参数的线性规划。上述对于每个智能体的策略改进过程不断的持续下去，直到所有智能体的线性规划得到的 ε 值都足够小时才停止迭代过程，返回一个最终的联合策略 π。值得一提的是，虽然该算法可以在有限步骤后收敛，但并不能保证收敛到一个全局最优的策略。很有可能的是，该算法会返回一个局部最优的策略，该策略从博弈论的角度上说是一个纳什均衡点。也就是说在其他智能体的策略不变的情况下，每个智能体的策略都是最优的，因此无法通过这种方式改进的更好。事实上，算法返回的这个局部最优策略可能与想获得的全局最优策略相差甚远。因此，以任意初始策略开始的单方迭代改进不能保证求解的联合策略足够的好。

和所有的局部搜索（Local Search）算法一样，摆脱局部最优的一个最简单的方法就是所谓的多次随机重启（Random Restarts）。基本的原理是，不同的初始搜索点可能会收敛到不同的局部最优策略。通过多次的重启，以不同的搜索点进行搜索，以期某个收敛到的局部最优值与全局最优值能够足够接近，或者直接收敛到全局最优值。而重启的次数由在线规划所允许的决策时间来决定。如果智能体有较多的时间用于在线规划，则可以通过多次的重启，以不同的随机初始策略进行迭代，这样获得接近全局最优策略的几率就能够大大的提高。算法 3.3给出了主要步骤的伪代码。在实际问题中，这种随机重启的方法具有实现简单而且效果足够好等诸多优点，因此被 MAOP-COMM 算法所采用。

注意到，在策略求解的过程中，每个智能体采用的相同的随机种子。这样保证的智能体在分布式求解中，所使用的算法拥有相同的随机行为。在算法实现中，还常常会遇到两个策略 q_i 和 q_i' 具有相同期望值的情况。为了确保每个智能体能够计算出完全相同的联合策略，从而避免误协调，每个智能体在策略选择上对于具有相同值的策略按照一个事先约定好的次序进行选择。例如事先约定好的次序是 $q_i \prec q_i'$，在 q_i 和 q_i' 具有相同期望值的情况下优先选择 q_i'。

3.3　基于策略相似性的历史信息归并

在 DEC-POMDP 模型的在线规划过程中，对于某个决策智能体来说，系统的状态和其他智能体所获得的观察都是未知的。为了与其他智能体进行协作，该智能体必须考虑其他智能体所有可能的观察历史以及这些历史如何影

算法 3.3　带重启的随机策略搜索

Input: H, B
for *several restarts* **do**
  // select the start point randomly.
  $\pi \leftarrow$ Initialize the parameters to be deterministic with random actions
  repeat
    $\varepsilon \leftarrow 0$
    foreach $i \in I$ **do**
      // optimize the policy alternatively.
      $\pi_i, \varepsilon' \leftarrow$ Solve the linear program in Table 3.1 with H, B, π_{-i}
      $\varepsilon \leftarrow \varepsilon + \varepsilon'$
  until ε *is sufficiently small*
  if π *is the current best policy* **then**
    $\pi^* \leftarrow \pi$
return π^*

响它本身的动作选择。但是，关键的问题在于这个可能的联合历史信息随着决策步数的增加呈指数式的增长。具体的说，在假设每一步都执行相同联合策略的情况下，这个可能的联合历史信息量是 $\mathcal{O}(|\Omega_i|^{T|I|})$ 量级的，这其实是一个非常庞大的数目。举一个最简单的例子，在只有两个智能体并且每个智能体只有两个观察的问题中，在执行 100 步后，所有可能的联合历史信息数是 $2^{100\times 2} \approx 1.6 \times 10^{60}$。这么大的信息量，即便是最为先进的计算设备也无法在短时间内进行推理和求解。事实上，如何存储这些信息都将是一个非常难解决的问题。这为设计高效的 DEC-POMDP 在线求解算法提供了巨大的挑战。

通过细致的分析可以发现，多数的联合历史信息对于决策规划来说都是没有太多价值的。原因之一在于考虑其他智能体可能的历史的目的在于了解它们当前所具有的用于决策的信息。而在一次具体的执行中，每个智能体所获得的只是某一个特定的观察历史，而不是所有的可能历史。由于不能彼此通讯，每个智能体不能够直接获得其他智能体当前把握的历史信息，它要做的事情是根据自己所获得的信息维护一个关于它们可能历史信息的一个概率分布。从理论上说，只要这个分布足够精确，那并没有必要保留所有可能历史的详细信息。而且在这个概率分布中，有很多的历史可能是 0 概率的，也就是说完全不可能出现，提早的删除这些历史信息并不会对最终的决策造成影响，反而能大大提高问题求解的速度并节约存储空间。

另一个可能的原因是先前的历史信息可能对于当前的决策来说是完全没有价值的。例如在多智能体的老虎问题中，一个可能的 6 步决策过程是：每个智能体监听老虎的位置两次，如果两次同时在某个门后听见老虎的咆哮声则打开另一扇门；然后再监听两次，以同样的方式打开某扇门。在这个过程中，1 到

3 步所获得的历史信息对于 4 到 6 步的决策过程没有实质性的影响。因为一定门被打开，老虎的位置会被系统重置，也就是说老虎被重新随机的放置在某个门后。之前所获得的历史信息不仅不能作为这次老虎位置的参考，反而会影响决策过程。对于另外一个智能体来说，真正要考虑的是在第 3 步门有没有被另一个智能体打开过，也就是老虎的位置有没有因此而被重置。在这个例子中，3 步之前的历史信息可以按照有没有打开门来进行归并，从而缩减需要考虑的历史信息量。

定义 3.3.1 (概率等价)**.** 智能体 i 的两个历史信息 h_i, h_i' 是概率等价（*Probabilistically Equivalent*）的，当且仅当：$\forall h_{-i}, p(h) = p(h')$ 以及 $\forall h_{-i}, s, b(s|h) = b(s|h')$，其中 $h = \langle h_i, h_{-i}\rangle, h' = \langle h_i', h_{-i}\rangle$ 且 h_{-i} 是除智能体 i 之外的其他智能体的历史信息。

概率等价意味着智能体的两个历史信息虽然具有不同的观察动作序列，却具有相同的概率分布并且其诱导出的信念状态也是相同的。对于概率等价的历史信息具有以下性质:

引理 3.3.1 (Oliehoek et al.[25])**.** 当智能体的两个历史信息是概率等价时，其所对应的最优反应策略也是等价的，因此可以合并成一个历史信息而不改变对应的最优策略值。

虽然概率等价在理论上具有很好的性质，能够保证合并后策略的最优性。但该性质却无法有效的应用到实际的问题求解中，主要的原因就在于概率等价的判别需要知晓其他智能体所有可能的历史信息 h_{-i}。上文提到过所有可能的 h_{-i} 数量上会极其的大，因此不具有操作性。而对于概率等价的判定来说，检测每一个具体的 h_{-i} 是重要的，因为这个会影响到智能体 i 最终的策略选择。

为了提出更具实用性的历史合并方法，有必要对每个具体的 h_{-i} 如何影响智能体 i 的策略选择进行一个详细的分析。首先，在考虑策略的时候，智能体 i 会根据 h_{-i} 来推理出一个信念状态。和它当前从环境获得的历史信息 h_i 一起，智能体 i 计算出一个基于这个联合历史的状态分布 $b(\langle h_i, h_{-i}\rangle)$。然后，智能体 i 需要考虑在这个状态分布下，其他智能体所可能采取的策略，并根据其他智能体所可能采取的策略计算出自己的策略。从这个角度上看，考虑其他智能体的历史的根本目的是为了推导出其他智能体所可能采取的策略以及自己的策略。假设智能体 i 在历史 h_i 下的最优策略 q_i^* 已经给定，那智能体就可以根据自己的观察来执行这个策略而无需考虑自己的历史更不要关注其他智能体的历史信息。例如在多智能体的老虎问题中，如果一个智能体知道了其当前的最优策略就是只要连续听到两次老虎的叫声就打开另一扇门，那么它就没有必要记录自己过去听到的老虎叫声这些历史信息，同时更不需要考虑其他智能体的历史信息。

定义 3.3.2 (策略等价)**.** 智能体 i 的两个历史信息 h_i 和 h_i' 是策略等价的（*Policy Equivalent*），当且仅当：h_i 和 h_i' 对应的最优策略等价。

在这里，一个策略指对的是从当前步开始到执行结束的一个完整的条件规划（Conditional Plan）。通常这个策略可以表示成一个决策树的形式，其中节点代表的是要执行的动作，而分支代表的是可能获得的观察。在策略空间中，最优策略是一个不动点，在其他智能体也执行它们最优策略的情况下，整个多智能体系统具有最优的表现。因此，一个智能体的最优策略是整个团队最优联合策略的一部分。在 DEC-POMDP 模型中，必定存在至少一个这样的最优联合策略。当前，对于小问题的最优策略可以利用一些已知的离线方法来计算。

定理 3.3.1. 当智能体 i 的两个历史信息 h_i 和 h_i' 是策略等价时，它们可以被合并且选择保留任意一个历史信息，而不影响对应的最优策略值。

证明. 在第 0 步决策时，在给定从第 0 步到第 T 步最优策略的情况下，智能体能够利用模型给定的初始信念状态 b^0 选择出最优的行动。假设在第 t 步决策时，智能体 i 根据策略等价对历史信息 h_i^t 和 $h_i'^t$ 进行合并，也就是说 h_i^t 和 $h_i'^t$ 具有相同的最优策略 q_i^t。在未来的任意 $t+k$ 步中，对应于任意的 k 步后继历史 h_i^k，它和之前被合并的历史信息的组合 $h_i^t \circ h_i^k$ 和 $h_i'^t \circ h_i^k$ 依然对应这相同的最优策略 q_i^{t+k}。否则，h_i^t 和 $h_i'^t$ 将具有不同的最优策略，也就是说它们并不是策略等价。因此，根据最优策略的定义，q_i^{t+k} 是 q_i^t 的一个子策略。如果 $h_i^t \circ h_i^k$ 和 $h_i'^t \circ h_i^k$ 对应的最优策略不同，那么通过在策略 q_i^t 中替换掉不同的关于 q_i^{t+k} 部分的子策略，可以使得 h_i^t 和 $h_i'^t$ 独自拥有自己的最优策略。这与 h_i^t 和 $h_i'^t$ 具有相同的最优策略，即二者策略等价矛盾。因此合并 h_i^t 和 $h_i'^t$ 并不影响后继历史的决策，因为对任意 h_i^k，$h_i^t \circ h_i^k$ 和 $h_i'^t \circ h_i^k$ 对应的最优策略依然相同，从而结论得证。 □

定理 3.3.2. 当智能体的两个历史信息是概率等价时，它们也是策略等价的，意味着它们对于的最优策略至少一对是相同的。

证明. 假设第 t 步的历史信息 $h_i^t, h_i'^t$ 是概率等价的。则对于未来的任意 $t+k$ 步，它们具有相同 k 步后继的历史信息 $h_i^{t+k}, h_i'^{t+k}$ 也是概率等价的，因为 $h_i^{t+k}, h_i'^{t+k}$ 是通过在 $h_i^t, h_i'^t$ 后加上相同的 k 步的动作和观察序列形成的。由于 $h_i^{t+k}, h_i'^{t+k}$ 是概率等价，因此它们具有相同的最优 Q 函数[25]。在依据最优 Q 函数进行动作选择时，智能体 i 会为 $h_i^{t+k}, h_i'^{t+k}$ 选择相同的行动。因此，可通过考虑各种 $(k = 1, 2, 3, \cdots)$ 后继的情况递归的构建 $h_i^t, h_i'^t$ 的最优策略。因为在每一步决策中都会为对应的分支选择相同的最优行动，所以 $h_i^t, h_i'^t$ 对应的最有策略是相同的，即它们也是策略等价。综上所述，当智能体的两个历史信息是概率等价时，

可以通过一定的方法为二者构建一个策略，并保证该策略对二者都是最优的，即二者策略等价，从而结论得证。

□

但是，当智能体的两个历史信息是策略等价时，它们却不一定是概率等价，也就是说上述的定理反之不成立。例如在多智能体的老虎问题中，假设智能体 i 的策略是如果 h_i 开左边门（OL），如果 h_i' 则开右边的门（OR）。假设 h_i 和 h_i' 出现的概率不同，则开完门后的历史信息 $h_i \circ \{OL\}$ 和 $h_i' \circ \{OR\}$ 不是概率等价的。但由于开完门后，老虎的位置会被重置，智能体面临的是和刚开始相同的情况，因为对于的策略也是相同的。所以 $h_i \circ \{OL\}$ 和 $h_i' \circ \{OR\}$ 是策略等价的。由此可见，策略等价比概率等价更加一般化，或者说概率等价是策略等价的一个子类，即：概率等价 $\subset$ 策略等价。在单智能体的 POMDP 问题中，给定一个历史信息，策略树的值可以被计算出来。因为不同的策略树对应于一个历史信息可能具有相同的值，因为值等价可能是更加一般的合并条件，也就是：值等价 $\subset$ 概率等价。但在多智能体的决策问题中，只有在给定了一个联合历史信息的情况下，联合策略的值才能够被计算。当前，对于 DEC-POMDP 问题，给定单个智能体的历史信息还无法对智能体的局部策略进行评价。因此，无法在多智能体的问题上应用值等价这个条件。

在 DEC-POMDP 的离线规划中，策略等价相当于子策略的重用。从本质上说，对若干个历史信息重用一个策略同对一个策略应用若干个历史信息在效果上是相同的。当目标是策略时，称之为重用；而目标是历史信息时，称之为合并。在自底向上的动态规划中，从当前开始到末周期的最优策略是已知的。通过策略的重用，可以使用一小部分的子策略重用的构造出任意复杂的上层策略。在考虑近似规划也就是没有要求策略一定要最优而是接近最优时，条件可以进一步放松。例如某个机器人被派出清扫楼层中的 10 个房间。这些房间可以分为 3 类：5 个办公室、3 个会议室和 2 个洗手间。这样，该机器人只需要知道怎么清洗这 3 类房间，而没有必要为每个具体的房间单独计算清扫策略。也就是说在清扫这 10 个房间时，某类房间的一个清扫策略能够被重复利用多次。当然，由于同类房间之间也存在差异，所以该类房间的清扫策略未必对该类的某个房间都是最优的，但至少基本上能大概满足需求。事实上，离线规划的 MBDP 算法采取的就是这种近似方法：在每次迭代中只保持固定个数的子策略，然后通过重用这些子策略来近似构造下一步迭代的策略。类似的方法也可用于在线规划，来对庞大的历史信息进行近似归并。

主要的问题在于，每个智能体从当前步开始的最优策略在执行阶段是未知的。而且这个最优策略也无法离线获得，因为计算这样一个最优策略相当于离

线求解整个 DEC-POMDP 问题。所以在 MAOP-COMM 算法中，采用有线步骤的前瞻来近似这个最优策略。一个 k 步的前瞻联合策略是由每个智能体深度为 k 的策略树组成的多元组。该前瞻联合策略的值函数可以通过一下方式获得：对前 k 步的策略通过贝尔曼迭代来计算；而 k 步以后的策略通过启发式函数来近似评价。这样的 k 步前瞻策略可以通过离线计算的方式预先获得。通过比较两个 k 步前瞻策略树的结构来获得两个策略的相似性。而上文定义的策略等价条件就可以通过比较两个 k 步前瞻策略树的相似性来获得。更确切的说，智能体 i 的两个历史信息是近似策略等价的当且仅当它们对应的 k 步前瞻策略树具有类似的结构。在 MAOP-COMM 算法中，k 步前瞻策略树类似意味着它们具有完全相同的结构，但在其他场合这种类似性的定义条件可以进一步放宽。

在 MAOP-COMM 算法中，提出了一种新的用于历史信息的近似归并方法。算法 3.4给出了主要步骤的伪代码。该方法的优点在于可以让信念池中的历史信息满足有限内存的需求，同时保持智能体之间的策略协调性。使用有限内存的历史信息对于在线规划算法来说具有重要的意义，因为在线规划的时间和空间计算资源通常是十分有限的。聚类的方法往往无法保持所维护的信念池的空间是固定的。如果使用的空间过大，也就是历史信息的量过多，就有可能在策略求解的过程中超时，从而无法获得本周期所要执行的动作。更糟糕的是，没有计算出的联合策略，信念池也无法获得更新。在实际问题中，这些因素都可能导致系统运行混乱；如果没有相应的错误恢复机制，甚至会导致系统崩溃。在 MAOP-COMM 算法中，智能体的历史信息通过策略等价来进行归并，归并过程中随机保留同类历史中的任意一个。由于预先生成的 k 步前瞻策略树的个数是固定的，所以归并后的历史信息量具体的依赖策略树类似性的定义，但一定小于 k 步前瞻策略树的个数，因此也是固定的。在每一步的归并中，启发式函数被用于 k 步前瞻和策略生成。

3.4　基于信念一致性检测的通讯策略

在 MAOP-COMM 算法中，通讯可以在算法进入规划阶段前完成智能体之间的信息共享。如图 3.3所示，具体的流程为：首先，每个智能体检测通讯资源是否可用，然后进行通讯决策确定是否需要和其他的智能体进行通讯。如果不需要通讯，则智能体直接进入规划阶段。如果需要通讯，但当前通讯资源不可用，则智能体将通讯推迟到下一决策周期。如果需要通讯且资源可用，则该智能体发起通讯并根据通讯后的信息进行规划。在这里，通讯的信息是每个智能体从上一个通讯周期到当前从环境中获得的观察信息序列。而通讯资源指的是智能体所带的通讯设备以及通讯媒介，例如在无线网络中，通讯节点间无信号

算法 3.4　基于策略的历史信息归并

Input: H^t, Q^t, π^t
foreach $i \in I$ **do**
  $\tilde{H}_i^t \leftarrow \emptyset$
  // $\mathcal{H}_i$ is a hash table indexed by q_i.
  $\mathcal{H}_i(q_i) \leftarrow \emptyset, \forall q_i \in Q_i$
  // group histories based on the policy.
  foreach $h_i \in H_i^t$ **do**
    // get the policy of h_i according to π_i^t.
    $q_i \leftarrow$ Select a policy according to $\pi_i^t(q_i|h_i)$
    // add h_i to the hash table with key q_i.
    $\mathcal{H}_i(q_i) \leftarrow \mathcal{H}_i(q_i) \cup \{h_i\}$
  // generate a new set of histories.
  foreach $q_i \in Q_i^t$ **do**
    // keep one history per policy.
    if $\mathcal{H}_i(q_i)$ *is not empty* **then**
      $h_i \leftarrow$ Select a history from $\mathcal{H}_i(q_i)$ **randomly**
      $\tilde{H}_i^t \leftarrow \tilde{H}_i^t \cup \{h_i\}$
  // fill up the history set.
  while $|\tilde{H}_i^t| < |Q_i^t|$ **do**
    $q_i \leftarrow$ Select a policy from Q_i^t randomly
    if $\mathcal{H}_i(q_i)$ *is not empty* **then**
      $h_i \leftarrow$ Select a history from $\mathcal{H}_i(q_i)$ **randomly**
      $\tilde{H}_i^t \leftarrow \tilde{H}_i^t \cup \{h_i\}$
return $\tilde{H}^t$

连接则被认为通讯媒介不可用。所有的通讯失败都将在下一个周期进行重新的通讯，本文主要关注的是通讯决策而不是具体的通讯过程。在 MAOP-COMM 算法中，需要通讯的情况分为三种，即：一、有其他的智能体发起通讯；二、在上一步的决策中有推迟的通讯任务；三、智能体维护的信念池被检测出与实际从环境中获得的信息不一致。前面两种情况都较为简单，下面着重分析第三种情况。

如果知道当前系统的状态，信念池的不一致性检测将会变得非常简单。然而，在 DEC-POMDP 的在线执行阶段，每个智能体获得的不是系统的状态而是智能体的传感器获得的关于系统状态的一个局部观察信息。所幸的是，这个观察信息在一定程度上反映了当前系统的状态信息。所以通过检测智能体的局部观察与信念池的不一致性来近似判断信念池与系统状态的不一致性。具体的说，智能体 i 执行完行动后获得某个观察 o_i，如果根据信念池推断当前智能体获得该观察的可能性小于一个阈值 ϵ，则很有可能是因为信念池中的信息不准确，也就是说信念池信息与实际的环境信息不一致。

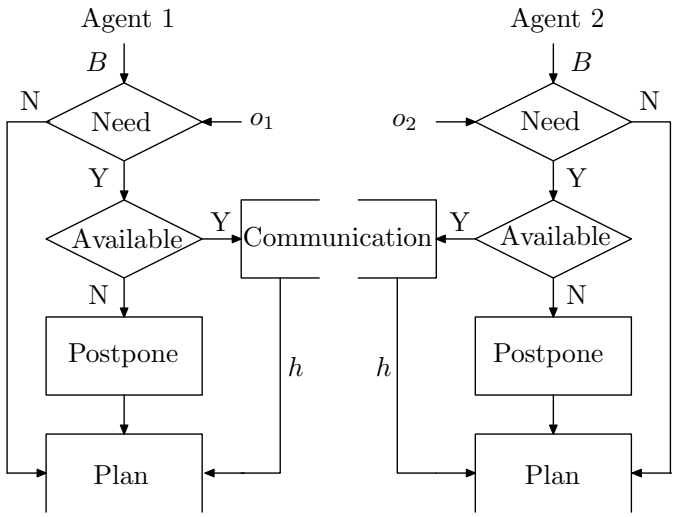

图 3.3 两智能体的通讯决策流程

在进行具体的形式化定义前，先引入一些重要的变量：h_i^t 表示的是智能体 i 在第 t 步时的从环境中获得的历史信息；o_i^t 表示智能体 i 在第 t 步时从环境中获得的新观察；而 $B(h_i^t)$ 表示的是信念池中与 h_i^t 相关的联合信念状态集。前面提到过，联合信念状态即为系统状态的一个概率分布。

定义 3.4.1 (观察的 ϵ 不一致性)**.** *在第 t 个决策周期中，智能体维护的信念池 B^t 被认为与智能体 i 获得的观察 o_i^t 具有 ϵ 不一致当且仅当下列不等式成立：*

$$\max_{\forall b, \forall o_{-i}^t} \left\{ \sum_{s' \in S} O(\vec{o}^t | s', \vec{a}) \sum_{s \in S} P(s' | s, \vec{a}) b(s) \right\} < \epsilon \tag{3.6}$$

其中 $o_{-i}^t \in \times_{k \neq i} \Omega_k$，$b \in B(h_i^t)$，而 $\vec{a}$ 是根据上一步计算出的联合策略以及联合观察 $\vec{o}$ 选择出的联合行动。

上述定义可以被用来检测智能体 i 的信念池与所获得的观察的不一致性，该不一致性为信念池与系统状态的不一致性提供了依据。MAOP-COMM 算法正是利用这一不一致性的定义来进行信念池的检测，从而决定是否需要通讯。其中阈值 ϵ 的大小由问题的观察函数的性质也就是智能体所能获得的观察的精度来决定。如果智能体所获得的精度高，说明其反映的系统状态信息十分的准确，则 ϵ 值应较小，反之亦然。需要注意的是，该定义并不能完全的检测信念池与系统状态的不一致性，因为只用到了当前从环境中获得的观察。但是，只要观察的不一致性被检测出来，信念池与系统状态之间就有很大的可能存在不一致性。

通常，通讯信息量的大小由观察的精度以及之前的决策二者共同决定。直观上说，只要智能体信念池中的历史信息与实际的联合历史足够接近，则智能体根据信念池能够计算出足够好的策略。但是在执行过程中，每个智能体不能

获知其他智能体的观察信息，所以也无法准确的得到实际的联合历史信息。在MAOP-COMM 算法中，智能体能够检测信念池的不一致性，不一致性从一个侧面反映了智能体维护的历史信息与实际的历史信息之间的差别。通过通讯，智能体能够共享它们之间的私有信息，从而纠正信念与实际状态之间的不一致性。在通讯过程中，每个智能体都获得实际的联合历史信息，通过这个历史信息能够刷新信念池中的信息，从而维护一个与实际状态相一致的信念。

和许多其他的通讯决策过程不同，MAOP-COMM 不要求通讯的即时性，也就是说通讯可以被推迟到下一个决策周期而不影响整个的决策进度。当然，通讯的推迟会影响到实际的决策效果，因为通讯能让智能体获得更准确的信息，从而做出更好的决策。但是在通讯资源不可用的情况下，推迟通讯也是必然的选择。因此，在 MAOP-COMM 算法中，通讯只是做为一种在可利用时提高决策质量的方法。而在很多其他类似算法中，通讯都是一个必要的决策因素。在决策周期中如果没有通讯，这些算法的决策要么退化成忽略局部信息的开环策略，或者是采用忽略其他智能体行为的贪心策略。显然，在通讯长时间不可用时，这种开环或者贪心策略的执行效果将可能任意的坏。而 MAOP-COMM 算法即便在没有通讯时也能很好的进行决策。它首先根据公共的信息计算出一个联合策略，然后再根据自己获得的局部观察信息从联合策略中选择智能体需要执行的动作。一方面，因为联合策略是根据公共信息计算的，能保证智能体之间的协调一致性；另一方面，由于利用到了智能体当前获得的局部观察信息，能够实时的对环境的变化作出反应。也就是说即便没有通讯，MAOP-COMM 也能在保证智能体协作的同时对当前的环境信息作出反应。

值得指出的是，MAOP-COMM 算法中利用到的信念不一致性的通讯策略与之前提到过的基于信念分歧[26] 的通讯策略具有本质的区别。基于信念分歧的方法首先设定一个信念状态的参考点，然后不断的比较当前信念状态与参考点之间的差别。信念状态之间的差别通过概率分布之间的 KL 距离来评价。如果这两个信念状态之间的差别也就是 KL 距离超过某个阈值，则智能体之间进行通讯并共享观察的历史信息。在这个方法中，某个智能体进行信念更新时假设其他智能体不单独的获取新的观察信息，也就是它们的信念状态在最后一次通讯后都一直保持不变。与此不同，MAOP-COMM 算法在更新联合信念状态时考虑到其他智能体所可能获得的各种观察信息。同时，信念的不一致性是根据智能体当前的观察与信念池之间的差别来进行评价的。

总的来说，MAOP-COMM 使用的通讯决策方法能够在资源不可用是推迟通讯，同时能够保证智能体间的协作和利用智能体当前的局部信息作出实时反映。虽然从理论上说，其他的方法也能够推迟某一步的通讯，但它们的决策过程将退化成开环或者贪心策略，从而严重的影响决策效果。比如在 Dec-Comm 算法

中，在没有通讯的情况下，智能体的决策将完全不考虑当前获得的观察信息。而在基于信念分歧的方法中，在没有通讯的情况下，智能体的决策将一直假设其他智能体没有获得新的观察信息。根据这个假设作出的决策将导致智能体之间的误协调。而 MAOP-COMM 即便在没有通讯的情况下，也不存在这些问题。

3.5　信念池结构的具体算法实现

从算法实现的角度，MAOP-COMM 最难实现的部分就是设计信念池的数据结构。上文提到过，MAOP-COMM 在具体实现的时候采用的是最简单的一步前瞻法。所以每个智能体的策略都是只有一个根节点的决策树，或者说是一个单独的动作节点。在存储历史信息的时候，并不需要在信念池中保存所有的动作和观察序列。而只需要为每个历史信息分配一个索引，并且用一个哈希表 $h^t = \langle \vec{\theta}, b^t \rangle$ 来表示联合历史信息，其中 $\vec{\theta} = \langle \theta_1, \cdots, \theta_n \rangle$，$\theta_i$ 是智能体 i 的历史信息索引，而 b 是由该联合历史信息诱导出的联合信念状态即状态的概率分布。在信念池的存储中，MAOP-COMM 只为每个策略保留一个历史信息，因此每个历史信息又可以简单的表示为 $\theta_i = \langle q_i^{t-1}, o_i^t \rangle$，其中 q_i^{t-1} 是上一步策略树所使用的索引，而 o_i^t 是当前从环境中获得的观察。对于一步前瞻来说，策略树可以进一步简化为动作，即 $\theta_i = \langle a_i^{t-1}, o_i^t \rangle$。因此，信念池中每个元素 $h^t \in H^t$ 的数据结构可以具体的表示为:

$$h^t = \langle \underbrace{\langle \underbrace{q_1^{t-1}, o_1^t}_{\theta_1} \rangle, \underbrace{q_2^{t-1}, o_2^t}_{\theta_2} \rangle, \cdots, \langle \underbrace{q_n^{t-1}, o_n^t}_{\theta_n} \rangle}_{\vec{\theta}}, b^t \rangle$$

在每一个决策周期末，算法首先对每一个历史信息索引进行更新，然后再对每一个新的联合历史计算对应的联合信念状态。注意到，新的历史信息是由每一个旧的历史信息末加上一个新的动作以及一个新的观察生成的，所以新历史的生成过程其实分为两个阶段。在算法中，不带新观察的中间态历史信息 $h_i \circ a_i$ 的索引表示为 $\langle q, * \rangle$。如果需要为每棵策略树保存多余一个历史信息，就需要重新设计一个新的索引表示形式。

图 3.4给出了一个历史信息展开和信念池更新的例子。在信念池中，联合历史 $\langle \vec{\theta}_0, b_0 \rangle$ 的两个主要部分部分 $\langle q_1, * \rangle$ 和 $\langle q_2, * \rangle$ 通过赋予不同的联合观察信息被展开成了 $\langle \vec{\theta}_1, b_1 \rangle, \langle \vec{\theta}_2, b_2 \rangle, \langle \vec{\theta}_3, b_3 \rangle, \langle \vec{\theta}_4, b_4 \rangle$，并为不同的展开量计算联合信念状态。然后，智能体 1 根据它从环境中获得的观察 o_1 将它的局部历史信息 $\langle q_1, * \rangle$ 更新为 $\langle q_1, o_1 \rangle$。同样的，智能体 2 也利用它的观察 o_2 将历史信息 $\langle q_2, * \rangle$ 更新为 $\langle q_2, o_2 \rangle$。

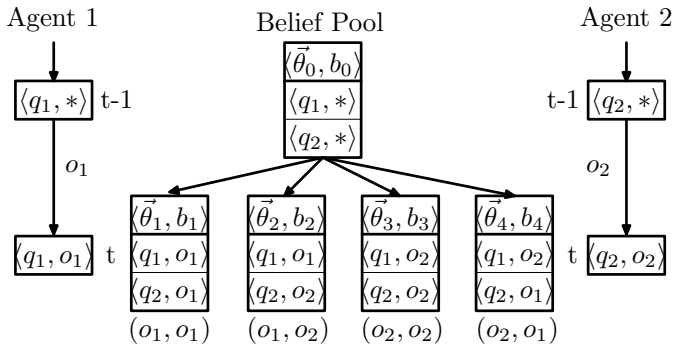

图 3.4 历史信息展开和信念更新示例

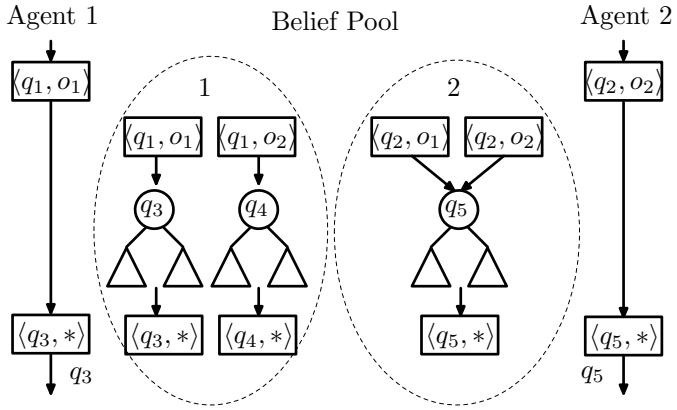

图 3.5 历史信息归并和策略执行示例

除了信念池，算法为每个智能体当前的历史信息也分配一个索引。需要注意的是，每个智能体当前的历史信息是一个智能体每步所实际执行的动作和实际从环境中获得的观察组成的序列。也是智能体在通讯时与其他智能体主要共享的信息源。为该历史信息分配一个索引的目的是与信念池中的历史信息表示相一致，便于策略的执行。与信念池中的历史索引相一致，智能体自己的历史信息索引也可以表示为 $\langle q_i^t, o_i \rangle$，其中 q_i^t 是智能体需要执行的策略索引，而 o_i 是当前从环境中获得的观察。当信息池中的一个历史信息 $\langle q_i^t, o_i \rangle$ 与另一个历史信息 $\langle q_i^t, o_i' \rangle$ 进行归并时，智能体自己的历史信息索引会自动更新为保留下来的那个历史信息的索引，比如说 $\langle q_i^t, o_i' \rangle$。因此，即使在对信息池中的历史信息进行归并后，智能体自己的历史信息也拥有指向信息池中对于历史的一个索引。这样就可以用来执行由信息池中的历史信息计算出来的联合策略。为了进行智能体间的通信，除了要维护一个指向信息池中的索引外，还需要保留具体的动作和观察序列 $\langle a_i^0, o_i^1, a_i^1, o_i^2, \cdots, a_i^{t-1}, o_i^t \rangle$。

图 3.5给出了历史信息的归并以及策略执行的具体例子。由于历史信息 $\langle q_2, o_1 \rangle$ 和 $\langle q_2, o_2 \rangle$ 对应相同的策略 q_5。因此它们被归并，且随机的选择一个（例如 $\langle q_2, o_1 \rangle$）保留到下一步。同时它们的索引也被更新为 $\langle q_5, * \rangle$。历史信息 $\langle q_1, o_1 \rangle$ 对应于 q_3，因此它的索引也被更新为 $\langle q_3, * \rangle$。同样的，对应于 q_4 的历史

信息 $\langle q_1, o_2\rangle$ 的索引更新为 $\langle q_4, *\rangle$。由于对应于智能体 1 的局部历史信息 $\langle q_1, o_2\rangle$ 的策略是 q_3，因此智能体 1 执行策略 q_3，同时更新其局部历史信息的索引为 $\langle q_3, *\rangle$。同理，智能体 2 执行策略 q_5 并更新局部历史的索引为 $\langle q_5, *\rangle$，因为同 $\langle q_2, o_2\rangle$ 相对应的策略是 q_5。

总的来说，算法实现的目的是设计一个高效的数据结构用于表示信念池以及智能体自己的历史信息。为了方便操作以及减少存储的数据量，对于信念池中的历史信息，算法采用的是索引的形式来表示而不是存储完整的动作和观察序列。每个历史信息的索引 $\langle q_i, o_i\rangle$ 具体由两部分组成：一个指向相应策略树的指针 q_i 以及当前获得的观察的索引 o_i。

3.6　实验结果

在实验部分，MAOP-COMM 算法被应用到了五个 DEC-POMDP 问题中：其中前四个是标准的测试问题，另一个问题是比前三个问题更加复杂的网格足球问题。在每次实验中，首先离线的求解问题相关的 MDP 问题，并将求的的值函数作为算法的启发式函数。这些问题的 MDP 都相对较小，因此可以采用简单的值迭代的方法求的最优值函数。然后，算法在每个问题上执行 20 次，并统计平均的效果。实验中记录了平均收益和（表格中的 Reward 项)、每一步的平均运行时间（表中的项为 Time(s)，单位为秒)、和通信总步数的百分比（表中的项为 Comm(%))。虽然在这些问题中通讯都是受限的且通讯决策的目的就是最小化通讯量，MAOP-COMM 算法并未直接量化每次通讯的代价，因为通讯与收益函数的关系在许多问题中都往往难以直接评估。设计这些实验的主要目的在于体现在利用了少量通讯的情况下，MAOP-COMM 能够在线快速的做出高质量的决策。具体的说，就是显示 MAOP-COMM 能够比其他经典算法使用明显更少的通讯的同时取得明显更好的收益。在试验中，MAOP-COMM 算法使用 Java 语言实现，并且运行在 2GB 内存的主频为 2.4GHz 的因特尔双核主机上。而线性规划使用的是开源的 lp_solve 5.5 的 Java 软件包求解。所有统计的时间都是精确到 0.01 秒的 CPU 时间。

算法 BaGA-Comm[27] 和 Dec-Comm[21] 是两个与 MAOP-COMM 类似的带通讯的在线规划算法。但是对于本实验涉及的测试问题，只有带粒子过滤（particle Filtering）的 Dec-Comm 版本（Dec-Comm-PF）[28] 能够顺利求解。前文提到过，出现这种局面的主要原因是 BaGA-Comm 和普通版本的 Dec-Comm 算法并没有限制历史信息（或者信念状态）的数量。在实验中，智能体可以超过 10 步没有进行通讯。在这种情况下，可能的联合历史信息的数量就会暴涨得非常大。例如在 2 智能体以及每个智能体 5 个观察的问题中，10 步以后这个数

量将是 $5^{2\times10}$。在这些问题中，甚至带聚类（Clustering）的 BaGA-Cluster 算法都无法将历史信息的数量控制在可操作的范围。这些指数式增长的历史信息会快速的塞满内存，导致一步的决策时间严重超时甚至无法求解出结果。因此，在实验中 MAOP-COMM 主要与 Dec-Comm-PF 算法进行比较，因为它是唯一已知的能够顺利运行于这些问题并获得结果的算法。

事实上，在大多数可被求解的问题中，BaGA-Comm 和 Dec-Comm 算法从平均收益值和通讯量的角度上看具有相似的表现。需要指出的另一个事实是 Dec-Comm-PF 和应用树结构的联合信念表示的 Dec-Comm 版本在实际表现上并无本质上的差别。因此，将 MAOP-COMM 与 Dec-Comm-PF ——这个实际存在的可求解这些测试问题的最好的算法进行比较，从实验的角度为体现 MAOP-COMM 算法的优势提供了最好的证明。为了更好体现算法的优势并更加深入的理解通讯在算法中所起的作用，实验还统计了两个特殊的情况：一、完全通讯的情况（FULL-COMM），也就是在每一步都进行通讯，相当于算法中的 ϵ 参数设置为正无穷大；二、完全没有通讯的情况（MAOP），相当于算法中的参数 ϵ 设置为 0。而一般的 MAOP-COMM 算法中采用的通讯决策阈值 $\epsilon = 0.01$。在比较实验中，Dec-Comm-PF 的粒子数取值 100。根据实验观察，Dec-Comm-PF 使用超过 100 的粒子数在策略收益值上没有明显的差别，但其每一步的在线运行时间却显著的增长。在实验中，DEC-POMDP 模型采用无折扣因子或者说折扣因子为 1 的方式来统计总的收益和，因为考虑的问题都是有限决策周期问题。

这里需要强调的是，完全通讯（FULL-COMM）的预期表现总要好于受限通讯的情形。在实验结果中提供 FULL-COMM 的结果，主要目的不是比较二者的优劣而是为 MAOP-COMM 算法的表现提供一个上界。虽然要求完全即时的通讯在许多实际问题中都不现实，但是从这个特殊的情况却可以看出其他算法在提供相同离线资源（MDP 启发式函数）时使用通讯的效率。在实现上，FULL-COMM 因为能在每周期通过通讯的方式获得所有智能体的局部观察，因此可以按 POMDP 的方式来运行：首先根据获得的联合观察维护一个联合信念状态，然后根据 MDP 值函数取得最好的联合行动，最后执行其相应的动作。FULL-COMM 每一步的运行时间很小，主要是因为其假设通讯是即时的，也就是不计通讯的时间开销。所以它的主要时间消耗就是联合信念状态的贝叶斯更新，而对于这些问题来说信念更新都可以快速的完成。上文提到过，完全通讯会让一个 DEC-POMDP 问题退化为 POMDP，变得相对容易求解的多（问题复杂度从 NEXP 降到 NSPACE）。在实验中提供 FULL-COMM 的主要目的是体现 MAOP-COMM 的表现与 FULL-COMM 这个理想的上界有多接近。

表 3.2 标准测试问题的实验结果

T	Algorithm	Reward	Time(s)	Comm(%)	Reward	Time(s)	Comm(%)
		Broadcast Channel			Meeting in a 3×3 Grid		
20	MAOP	18.25	< 0.01	0.0	3.10	0.15	0.0
	MAOP-COMM	18.35	< 0.01	0.0	3.35	0.26	11.0
	Dec-Comm-PF	18.45	< 0.01	0.0	2.90	0.22	70.50
	FULL-COMM	18.90	< 0.01	100.0	4.75	< 0.01	100.0
100	MAOP	89.95	< 0.01	0.0	15.30	0.19	0.0
	MAOP-COMM	90.35	< 0.01	0.0	17.10	0.30	12.10
	Dec-Comm-PF	90.0	< 0.01	0.0	14.90	0.24	78.20
	FULL-COMM	90.60	< 0.01	100.0	24.70	< 0.01	100.0
		Cooperative Box Pushing			Stochastic Mars Rover		
20	MAOP	7.50	0.14	0.0	18.19	0.97	0.0
	MAOP-COMM	99.30	0.16	11.50	46.19	0.05	17.0
	Dec-Comm-PF	136.75	0.35	83.50	45.02	2.46	22.0
	FULL-COMM	222.50	< 0.01	100.0	61.41	<0.01	100.0
100	MAOP	-16.0	0.13	0.0	51.15	2.20	0.0
	MAOP-COMM	441.95	0.13	12.26	222.86	0.09	18.00
	Dec-Comm-PF	296.50	0.36	59.87	133.04	2.39	35.00
	FULL-COMM	880.50	< 0.01	100.0	325.04	<0.01	100.0

3.6.1　完全可靠通讯信道问题

在这组实验中，通讯信道是完全可靠的，也就是说假定通讯资源一直可用，且没有通讯错误等因素使得通讯任务不能在一个决策周期内完成。

标准测试问题

首先介绍的是实验中使用的四个标准测试问题：信道广播问题（Broadcast Channel）[29]、格子相遇问题（Meeting in a Grid）[29]、合作推箱子问题（Cooperative Box Pushing）[16]、以及随机火星漫游者问题（Stochastic Mars Rover）[19]。这些问题被广泛的应用于评价使用 DEC-POMDP 模型的多智能体合作规划问题。由于在线规划的主要挑战之一是历史信息可能会变得很大，如何控制历史信息的大小是多智能体在线规划算法的关键所在。因此在选择测试问题时，有目的的选择观察数较多的问题。而其他常用的标准问题，例如多智能体老虎问题（Multi-Agent Tiger）[2]、废物回收机器人问题（Recycling Robots）[30]、以及救火机器人问题（Fire Fighting）[31] 则只有 2 个观察。

信道广播问题是一个简化的两智能体的网络通信问题。在一个决策周期中，每个智能体必须决定是否往公共的信道中发送一条消息。如果两个智能体同时发送消息，则公共的信道发生冲突，两条消息都不能成功发送。该问题总共有 4 个状态，每个智能体有 2 个动作和 5 个观察。表 3.2 中的结果显示所有的算

法都具有类似总收益且每一步的运行时间都小于 0.01 秒。允许通讯的两个算法 MAOP-COMM 以及 Dec-Comm-PF 在这个问题中都没有使用通讯资源。而且没有通讯和完全通讯的算法具有几乎相同的表现。这一结果可以有一个简单直观的解释：因为在这个问题中，每个智能体的缓冲区中消息到来的概率是不同的，一个是 0.9 而另一个是 0.1，因此该问题的智能体之间能够很好的相互协调，只要给消息到来概率高的智能体更多消息发送机会即可。

在格子相遇问题中，两个机器人在一个没有障碍物的格子世界中移动。它们的任务是用尽可能短的时间在同一个格子中相遇。为了使问题更具有挑战性，实验使用了 3×3 的格子，并且对每个机器人引入传感器噪声，也就是它们获得正确观察的几率降为 0.9。这个问题中每个机器人可以在 9 个格子中的任意一个，因此总共有 81 个状态。每个机器人有 5 个动作和 7 个合法的观察信息用来识别机器人身体前方的物体。表 3.2中的结果显示：即便是完全没有通讯，MAOP 算法也表现的非常的好。在这个问题中，智能体的一步前瞻的启发式函数起到了很好的作用。FULL-COMM 的结果表明，在这个问题中，通讯的作用并不十分的明显。MAOP-COMM 能用少的多的通讯量获得比 Dec-Comm-PF 更多的收益。而且 MAOP-COMM、MAOP 和 Dec-Comm 这三个算法每一步规划用的时间都非常短，而且不分伯仲。

在合作推箱子问题中，两个机器人在 3×4 的格子中移动，并要求把两个小箱子和一个大箱子推向制定的目标位置。智能体能够通过彼此合作一起推动大箱子，从而获得更多的收益。为了使问题更具有挑战性，实验随机的设定机器人的初始位置，并且为机器人的观察引入噪声，机器人获得正确观察的概率降低到 0.9。这个问题中有 4 个目标状态和 96 个一般状态，也即是总共 100 个状态。每个机器人有 4 个行动和 5 个观察。表 3.2中的结果表明：在这个问题中，通讯能够显著的提高决策的质量，而没有通讯的 MAOP 算法表现不佳。由于这个问题中，每个智能体有不同的选择（可以单独推各自的小箱子或者一起推一个大箱子），因此一步前瞻的启发式函数已经不能智能体的决策要求。给定较长的决策周期例如 $T = 100$ 时，MAOP-COMM 的表现要比 Dec-Comm-PF 要好，而且使用的通讯量也比 Dec-Com-PF 少得多。同时 MAOP 和 MAOP-COMM 的单步运行时间也比 Dec-Comm-PF 要少。

随机火星漫游者是有着两个智能体和总共 256 个状态的相对较大的问题。在这个问题中，每个智能体有 5 个动作和 8 个观察。在原始问题中，智能体获得的观察是确定性的。而在实验中，智能体的观察带有一定的误差，只有 0.9 的概率准确获得智能体前方的物体。表 3.2中的结果表明通讯对于这个问题来说也是十分的重要。没有通讯的 MAOP 算法获得的收益要比有通讯的 MAOP-COMM 算法要少。和前面类似，在多周期的情况下，MAOP-COMM 能使用少得多的通

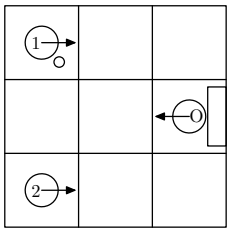

图 3.6 两个智能体的 3×3 格子足球问题

讯量获得比 Dec-Comm-PF 更多的收益值。

小节上面的实验结果，在所有的标准测试问题中，MAOP-COMM 都只要使用少的多的通讯量获得比 Dec-Comm-PF 更好的效果。在信道广播和格子相遇这些问题中，完全不带通讯的 MAOP 算法也有不俗的表现。虽然在上述实验中，总的决策周期最多只测试到 100，但 MAOP-COMM 完全可以应用在决策周期任意长的问题中。在 MAOP-COMM 算法的每步决策中只保留有限数量的历史信息，也就是说智能体需要维护的历史信息量不会随着决策周期的增多而指数式的增长。通常来说，由于误差的积累，长周期的问题在收益值上的差别也会比短周期问题略大。这些实验结果还说明了参数 ϵ 很好的权衡了通讯量和总收益值之间的关系。在某些问题中，例如合作推箱子问题，较大的 ϵ 值将会产生较多的通讯，从而提高总收益的值。

格子足球问题

为了检验算法在大规模问题中的求解效果，作者设计了一个较大规模的格子足球问题。如图 3.6所示，在这个足球问题，一支队伍有两个智能体而他们的对手的另一支队伍中的一个拥有固定策略的智能体。每个智能体有前后左右 4 个可能的身体朝向。对手能够完全观察整个场上的情形，并且它的动作是确定性的，也就是说不会有失误。但它的策略却是固定的：总是尽可能快的向球的方向移动。如果这个对手接触到带球的智能体，则球落入对手的脚下，整个比赛结束，失球的一方获得 -50 的收益。但如果带球的智能体进入球门，则比赛也结束，带球的一方获得 100 的收益。每个智能体具有 6 个可执行的动作，分别是：向东南西北移动一格、留在原位不动、以及传球给另一个智能体。每个动作有 0.9 的概率成功，而又 0.1 的概率依然保持在当前的状态。当一个智能体传球给另一个智能体时，当且仅当接球的智能体当前留在原地不动时，在下一个周期由接球的智能体控球。否则，比赛将以球出界告终，而控球的一方获得 -20 的收益。为了让智能体能尽快的把球带入球门区，对于每一个非终止的状态，智能体的每一步决策获得 -2 的收益。当比赛终止时，每个智能体的位置将随机重设。每个智能体的观察由 5 个其身前的信息（空位、墙、队友、对手和

表 3.3 格子足球问题的实验结果

T	Algorithm	Reward	Time(s)	Comm(%)	Reward	Time(s)	Comm(%)
		Grid Soccer 2×3			Grid Soccer 3×3		
20	MAOP	180.50	0.25	0.0	190.70	1.90	0.0
	MAOP-COMM	290.6	0.28	14.80	296.00	2.30	27.0
	Dec-Comm-PF	129.50	1.36	33.5	271.40	15.26	59.0
	FULL-COMM	373.90	< 0.01	100.0	356.0	< 0.01	100.0
100	MAOP	1157.80	0.14	0.0	803.60	1.96	0.0
	MAOP-COMM	1933.90	0.16	15.40	1679.50	2.50	26.90
	Dec-Comm-PF	1441.60	1.28	30.80	1044.40	9.48	70.70
	FULL-COMM	1933.60	< 0.01	100.0	1808.20	< 0.01	100.0

球门）以及两个关于球的信息（是否自己控球）组成，因此总共有 11 个可能的观察信息。而且这些观察信息带有不确定性，每次智能体有 0.9 的概率获得准确的信息，而又 0.01 的概率获得其他的任意信息。实验分别在 2×3 和 3×3 的格子中进行，前者总共有 3843 个状态，而后者具有 16131 个状态。由此可见，格子足球比当前所有已知的用于 DEC-POMDP 的标准测试问题都要大得多。

表 3.3中给出了两组实验的结果。从结果中可以看出，MAOP-COMM 算法的在决策质量以及通讯量的使用上都比 Dec-Comm-PF 要好得多。而没有通讯的 MAOP 算法的表示也十分的出色，这从一个侧面说明了当 MAOP-COMM 使用较小的 ϵ 值时，将会进一步的降低通讯量，同时还能保持较好的决策质量。而从算法运行时间的角度看，MAOP-COMM 和 MAOP 算法比 Dec-Comm-PF 算法快乐将近 10 倍。一个主要的原因是 MAOP-COMM 中的操作比 Dec-Comm-PF 中的粒子过滤方法要快得多。同时，在 MAOP-COMM 中需要保持的历史信息也比 Dec-Comm-PF 中要少。虽然 MAOP-COMM 保留的历史信息少了，但实际的决策质量却没有递减，足以看出 MAOP-COMM 在重要历史信息的选择上更有针对性。比较 MAOP-COMM 算法在 2×3 和 3×3 问题上的运行时间可以看出，虽然状态数在成倍的增长，运行时间却没有增加的太多。事实上，在 MAOP-COMM 中和状态数有关的主要部分是联合信念的贝叶斯更新。对于 16131 个状态，一次更新就需要耗费上百秒的时间。对于这些大状态问题，一个可用的技巧是利用问题状态转移的稀疏性，利用这个性质可以对极大的提高算法的效率。从这个分析也可以看出，历史信息的多少是在线算法执行时间快慢的关键，如果在保留较少历史信息的同时提高决策的质量是没有在线规划算法需要主要解决的问题。而这些实验结果说明，MAOP-COMM 在这一方面做的比已知存在的其他算法都要好。

3.6.2　非可靠通讯信道问题

在许多实际应用中，由于设备环境等因素，智能体之间的通讯往往是不可靠的。例如在无线网络中，由于具体天气等因素，传输信号可能会变得很不稳定，以至于一个消息需要多次重传才能完成。而 MAOP-COMM 算法的优势之一就是允许在通讯资源不可用的情况下延迟通讯。从一般意义上说，其他在线规划算法的通讯也可以被强制推迟，当对于这些算法来说强制推迟可能会严重的影响到决策质量。例如在 Dec-Comm 算法中，强制推迟通讯的结果是使得决策过程变成完全忽略局部观察信息的开环决策。再比如强制推迟基于信念分歧的决策中的通讯，将使得智能体对其他智能体做出武断的假设，从而使得计算出的贪心策略会导致智能体之间的误协调。和这些算法不同，在 MAOP-COMM 中，当通讯被推迟时，算法依然能够计算出一个联合策略，而这个联合策略的执行将能利用到当前的局部观察信息。而这个联合策略的执行保障了智能体之间行为的协调。

在这一节中，专门针对合作推箱子问题进行了一组在非可靠通讯环境下的实验。在这些环境中，通讯信道往往是间歇可利用的。实验通过仿真模拟这样的间歇通讯信道：首先，设定一个 0 到 1 之间的阈值；在每一个决策周期开始前，产生一个满足均匀分布的随机值，如果该值大于这个阈值，则说明通讯信道可用，反之则表示通讯信道不可用。在这组实验中，对该阈值在 0 到 1 之间进行变动，并记录 20 次执行获得的总收益和通讯量的均值。很显然，当阈值为 0 时，表明通讯资源一直可用；而当阈值为 1 时，表明通讯资源一直不可用。实验应用了 MAOP-COMM 算法在处理延迟通讯上的两个变种：一、MAOP-COMM-POSTPONE 算法会将通讯一直推迟到通讯资源可用；二、MAOP-COMM-DROP 算法在通讯资源不可用时直接放弃通讯请求。

图 3.7和图 3.8中分别给出了收益值和通讯量的实验结果。和预期相同，当通讯资源的可用性变得十分不稳定时，简单丢弃通讯请求的 MAOP-COMM-DROP 算法在收益值和通讯量上都有明显的下降。这意味着在合作推箱子问题中，智能体之间的通讯是十分重要的，简单的忽略这些通讯请求将大大的降低决策的质量。有趣的是，推迟通讯的两个算法在信道不确定性阈值处于某个范围的时候，收益值和通讯量都会有所上升。从主观上说，当通讯信道变得越来越不稳定时，智能体间的通讯量和收益值都应该逐渐下降，而实验的结果却和这个主观期望不相符。在信道不确定性的阈值在 0 到 0.4 之间时，MAOP-COMM-POSTPONE 算法的通讯量和收益值都有所上升。对于这一违反主观期望的现象，存在多种可能的解释。当通讯被推迟时，智能体依然需要在没有通讯的情况下根据它已经获得的信息进行决策。随着时间的推移，已经被

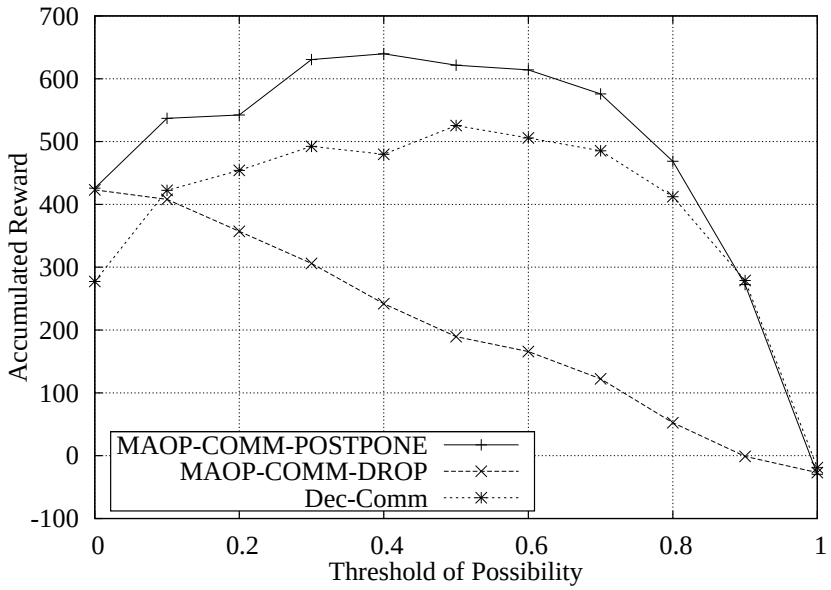

图 3.7 收益值随不确定阈值变化的实验结果

检查到不具有一致性的信念池内的信息会变得原来越不确定。同时，由于在算法中使用的启发式函数是关于未来期望收益的一个乐观估计。在不确定性增加的情况下，这种乐观的估计在一定程度上被削弱，从而产生一些正面的效益。

另一个可能的解释是针对像合作推箱子问题这样具有多个潜在目标的问题。在这类问题中，通讯的产生可能会让智能体改变它们原来坚持的目标。因为通讯的信息对信念池进行强制的刷新，因此旧的信息被抹去，注入的新的知识可能与信念池中的旧知识相差甚远。注意到，在评价未来收益的时候使用的仅仅是一个具有乐观估计的启发式函数，根据这个函数因为通讯多次的改变已有的目标在很多时候并不是一个明智的选择。例如，在合作推箱子问题中，假设智能体甲离小箱子很近，而且根据当前的信念池信息，它决定要独自推动小箱子。但同时，智能体乙离大箱子很近，它通过通讯告知智能体甲该信息。在进行通讯以后，两个智能体决定一起推动大箱子。但由于启发式函数只是一个乐观估计，推大箱子这个任务需要的步骤较多有可能实际上并不能够完成，而原来推小箱子的任务反而更有可能完成。从最终收益值的角度说，智能体在通讯以后可能因为对形势的过分乐观估计而错误放弃一些原来可以实现的任务，而是选择了一个不能实现的大任务，因此获得的收益值较小。这也说明了在智能体的通讯决策中，并不是越多越好或者越频繁越好，而是要在适当。

实验还将 MAOP-COMM 的两个版本与 Dec-Comm 强制推迟通讯的版本在非可靠通讯信道的环境中的表现进行了比较。从结果可以看出，MAOP-COMM-POSTPONE 能比 Dec-Comm 在使用更少通讯量的情况下获得更多的收益值。有趣的是，在不确定性的环境下，Dec-Comm 也呈现出与 MAOP-COMM-

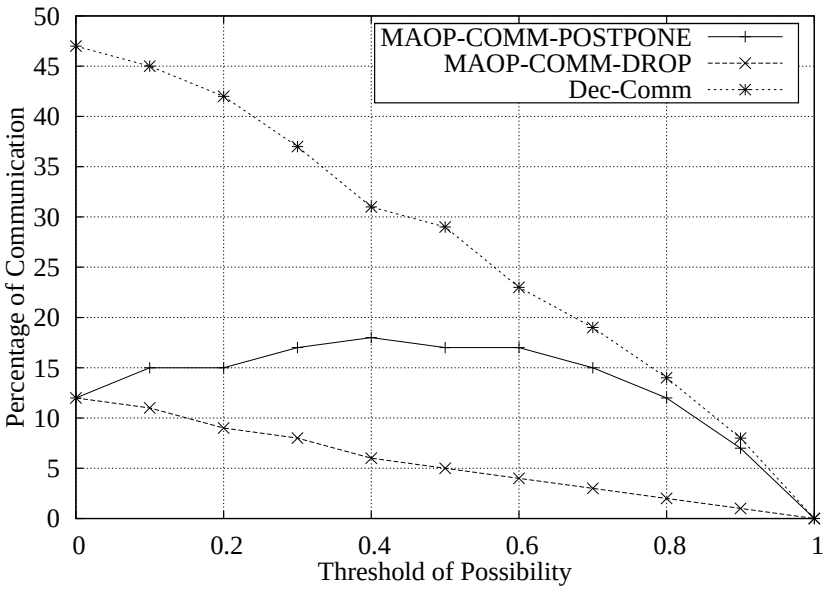

图 3.8 通讯量随不确定阈值变化的实验结果

POSTPONE 一样的规律——通讯量和收益值先升后降。总的来说，这种实验体现了 MAOP-COMM-POSTPONE 在信道不确定性的环境下的优势。因为在实际环境中，信道都是具有一定的不确定性的，所以 MAOP-COMM 和其他算法相比更能适应实际的通讯环境。

3.7　本章小结

这一章介绍了在通讯受限环境下多智能体的在线规划算法 MAOP-COMM。为了在保持智能体之间协作的基础上充分的利用各自的局部观察信息，算法维护了一个信念池的数据结构用于对大量的历史信息进行存储和推理。同时算法还发展出了一套检测信念不一致性的方法，当信念池存在不一致性时利用通讯来进行智能体间的信息交流。和以往的规划算法相比，MAOP-COMM 算法具有许多重要的优势。首先，它能够通过检测信念不一致性高效的利用通讯资源。而且当通讯资源不可用是，它还能延迟通讯并在完全没有通讯的情况下进行决策。算法的第二个优势是它在求解大规模问题时的可扩展性（Scalability）。它不能用比离线算法快的多的速度求解已知的标准测试集问题，还能求解 DEC-POMDP 的离线算法无法求解的大规模问题。最后，算法在许多的测试问题中具有很好的表现，和其他已知的在线算法相比，MAOP-COMM 算法能用少得多的通讯量获得更好的收益值。

通常来说，在线算法的表现很大程度上取决于用于指导智能体动作选择的启发式函数。对于在线算法来说，最优的值函数是不可知的，因为获得最优值

函数的过程相当于完全求解整个 DEC-POMDP 规划问题。但是它可以利用一些比较容易求解的值函数来近似，例如根据 MDP 计算得出的 Q 函数，这些值函数能为智能体的在线规划提供相应的启发式信息。但启发式函数的好坏主要还是与实际求解的问题相关。在多智能体系统中，要找到能完全考虑智能体之间复杂依赖关系的启发式函数是困难的。特别是在通用的 DEC-POMDP 模型中，一个智能体的行为可以改变其他智能体的观察。这种隐含的通讯行为对于智能体之间的协作是至关重要的。对于这用关系的进一步分析和利用将是智能体在线规划研究的一个重点。

之前分析过，在多智能体的在线规划算法中不可能对所有可能的历史信息进行存储和推理。因此，MAOP-COMM 算法中引入了基于策略等价的历史归并方法来保证信念池中的历史信息不会发生指数爆炸。很显然，任意使用有限内存的信息压缩技术都不可避免的要引入误差，从而使得信念池中的信息与实际情况产生不一致。而 MAOP-COMM 算法是第一个关注到历史信念信息的不一致并提出用通讯来解决这种不一致性的在线规划算法。算法在每一步的决策中，不断的对信念池进行监测，一旦有智能体检测到信念池具有不一致性，则通过通讯来分析彼此的信息从而消除信念池与实际情况的不一致性。值得一提的是，MAOP-COMM 算法的检测效果还取决于 DEC-POMDP 模型观察的精度，因为在每一步的决策中信念池的不一致性是通过将智能体的局部观察与信念池中的信念相比较来进行的。如果智能体的观察具有较大的噪声，则很难判断检测到的不一致到底是信念池与实际环境不一致还是智能体获得观察的不确定性造成的。好在许多实际问题中，智能体的传感器都有足够的精度来检测信念池的不一致性。即便是没有检测到不一致性，智能体也能在不浪费通讯资源的情况下继续进行规划和决策。对于这个问题一个可能进一步研究的方向是考虑使用智能体之间的通讯来进行策略协商。

从更大意义上来说，MAOP-COMM 所关注的多智能体在线规划算法作为离线算法的一个重要补充，在基于决策理论的多智能体规划问题上具有重大的意义。作为问题求解的一类重要方式，MAOP-COMM 算法展示了多智能体在线规划算法的主要优势，使用较少的时间和有限的通讯求解更大规模的规划问题。该算法的另一个主要贡献在于为今后的研究提供了一个完整的在线规划框架，同时提出了在线规划算法需要研究的重点，即利用局部信息的同时保证智能体间的协作、利用有限的时间和空间处理大量的历史信息以及如何更有效的使用通讯资源。从实验结果上看，新提出的 MAOP-COMM 算法在这三个方面都有较好的表现。

第 4 章　有限资源离线规划算法

离线规划通过搜索智能体的联合策略空间，从而获完整的可分布式执行的联合策略。由于求解过程是在离线状态下完成，所以无需考虑智能体之间的分布式协作，同时也没有决策周期在规划时间上的严格限制。但由于联合策略空间往往极其庞大，如何利用有限的时间和空间资源在联合策略空间中获得较好的联合策略成为离线规划算法需要解决的主要问题。

在 DEC-POMDP 的离线求解中，MBDP 算法[15] 很好的结合了自顶向下和自底向上的方法来构建和优化策略。算法使用自顶向下的启发式函数来生成一系列的可达信念状态。然后采用自底向上的动态规划来一步步的迭代，根据上一步的策略来构建当前的策略。在动态规划的每一步迭代中，仅仅保留了在生成的信念点上取得最优值的策略，然后将这些策略作为下一步迭代的子策略。每一步迭代保留的策略个数在算法中由参数 maxTrees 来决定，它同时也是自上而下生成的策略点的总个数。通过在每一步选择和保留固定个数的子策略，MBDP 的主要优势是具有相对于总的决策周期 T 的线性时间和空间复杂度[15]，这在 DEC-POMDP 问题的近似求解中是一个较大的突破。但在大规模问题求解中，MBDP 算法本身也存在一些需要改进的因素。

首先，MBDP 算法在从上一步迭代构建当前的策略集时，使用的是完全备份（Exhaustive Backup）的方法。完全备份的方法完全的枚举了所有可能的当前策略，因此效率较低。在大规模问题的求解中，一步迭代生成的策略数就有可能成千上万。具体的说，当智能体 i 的上一步迭代的策略数为 $|Q_i|$，完全枚举时，当前所有可能的策略数就为 $|A_i||Q_i|^{|\Omega_i|}$，相对于上一步的策略数是一个指数式的增长。当前，许多旨在改进 MBDP 完全备份的算法[16–19] 被提出。这些算法虽然在一定程度上提高了完全备份的效率，但在每一步的策略选择上只能保留较小个数的子策略，一般来说 maxTrees≤10。小的子策略数意味着在下一步的策略生成中具有较少的选择，这就有可能严重的影响到决策的质量。

本文提出的基于信念点的策略生成算法（PBPG）是 MBDP 算法的一个改进。PBPG 的主要目标是引入一种全新的策略生成方法，使得在保留较多子策略的情况下更有效的生成策略，从而更快更好的对大规模的 DEC-POMDP 问题进行求解。PBPG 算法的主要观察是：可以根据每个信念点直接生成最优的策略，而无需像完全备份一样首先枚举出所有可能的策略，然后再进行选择。也就是说 PBPG 合并了 MBDP 中策略枚举和策略选择这两个步骤，改成了直接构建基于信念点的联合策略。这就从本质上避免了对所有策略的枚举，因此大大提高了 MBDP 算法的效率。而且当 PBPG 生成的策略是最优时，效果与 MBDP

的完全备份等价。这从理论上证明了 PBPG 算法策略生成的质量。

策略评价是 MBDP 算法的另一个瓶颈。在动态规划的每一步迭代中，需要对生成的联合策略进行评价，具体的做法是对每一个联合策略计算一个值函数。但由于每个智能体一步生成的策略已经非常的多，而联合策略是每个智能体生成策略的一个组合，其个数随着智能体数目的增加呈指数式的增长。同时，每个联合策略的值函数是关于每个状态的映射值，因此对于状态数很多的问题来说，对值函数的保存和处理可能需要耗费大量的空间和时间。更重要的是，多智能体的问题由于系统的状态是每个智能体自身状态的一个组合，所以常常具有非常大的状态数。这一切问题的根源在马氏决策论中被称为的“维度诅咒”。

克服维度诅咒就根本的方法就是分析值函数的结构，对值函数进行近似的表示和计算。其中基于测试（Trial）的方法最为常用，广泛的应用于单智能体的 MDP 和 POMDP 问题求解。本文针对多智能体的离线规划问题，提出基于测试反馈的近似动态规划算法（TBDP）。TBDP 通过对概率上最可能的状态的分析，来避免不必要的计算并对策略进行近似的评价。同单智能体的实时动态规划算法（RTDP）[32] 一样，TBDP 在大规模问题的计算中具有很强的可扩展性（Scalability），同时能够很好的利用当前流行的多核多处理器的计算资源来加速问题的求解。值得一提的是，在多智能体的离线规划算法中，TBDP 是首个能利用到多处理器资源的近似算法。

本文提出的两个算法 PBPG 和 TBDP 都是针对 MBDP 算法的关键问题进行的改进。同时，TBDP 算法在 PBPG 快速策略生成算法的基础上进行了改进，引入了不同于策略树的新的策略表示方法。这种策略表示能够通过数值的方法直接计算出每个信念点上的最优联合策略。同时，也可以在计算的过程中得知哪些子策略的值函数才需要计算。这在很大程度上减少了需要计算的值函数的个数，从而加速问题的求解。

4.1　基于信念状态的快速策略生成

在这一节中先形式化的描述动态规划迭代过程中的策略生成问题，然后给出高效的近似算法 PBPG，并简要分析了该算法的复杂度。

4.1.1　生成策略的问题描述

设计 PBPG 算法的主要目标是在 MBDP 迭代的每一步为给定的信念点直接生成最优的策略树。在给定一个信念点的情况下，基本的流程如下：首先为每个联合行动的每个观察分支选择最好的子策略树；然后，为该信念点选择一个

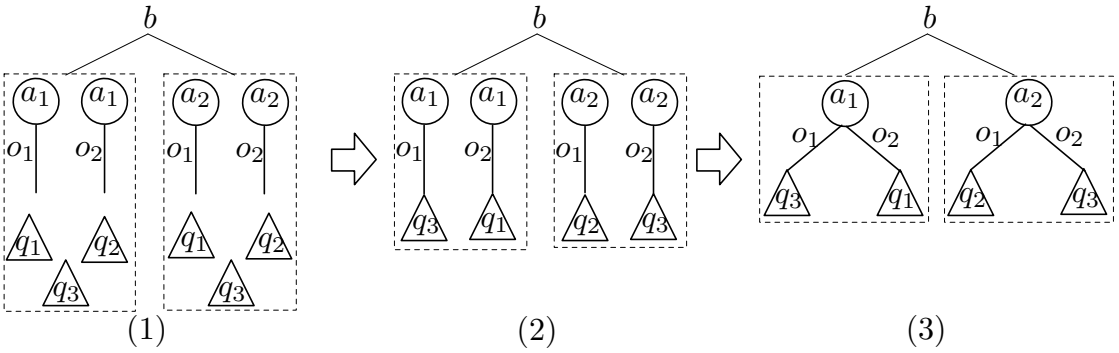

图 4.1 基于信念点的策略树生成示例

最好的最好的联合行动，并根据上面计算提供的子策略信息构建出在这个信念点上的最优策略树。子策略选择问题可以形式化的描述如下:

定义 4.1.1 (子策略选择问题)**.** 求解子策略选择问题就是在给定信念点 b，联合行动 $\vec{a}$，t 步迭代的策略树集 $\vec{Q}^t = \langle Q_1^t, Q_2^t, \cdots, Q_n^t \rangle$ 和值函数 $V^t : \vec{Q}^t \times S \to \Re$ 的情况下，为每个智能体 i 寻找一个观察到策略的映射 $\delta_i : \Omega_i \to Q_i^t, \forall i \in I$，最大化 $t+1$ 步迭代的值函数：

$$V^{t+1}(\vec{a}, b) = R(\vec{a}, b) + \sum_{s', \vec{o}} Pr(\vec{o}, s' | \vec{a}, b) V^t(\vec{\delta}(\vec{o}), s') \tag{4.1}$$

其中联合映射 $\vec{\delta}(\vec{o}) = \langle \delta_1(o_1), \delta_2(o_2), \cdots, \delta_n(o_n) \rangle = \langle q_1^t, q_2^t, \cdots, q_n^t \rangle$，收益函数 $R(\vec{a}, b) = \sum_s b(s) R(s, \vec{a})$，以及条件概率 $Pr(\vec{o}, s' | \vec{a}, b) = O(\vec{o} | s', \vec{a}) \sum_s P(s' | s, \vec{a}) b(s)$。

上述问题的主要目标是为每个观察选择一个第 t 步的子树使得 $t+1$ 步的值函数最大。但是为了方便后面的策略树构造，这个子策略的选择必须能够满足一些分布式运行的条件。也即是说对于智能体 i 来说，它的每个局部观察 o_i 对应一个子策略树 q_i^t。整个过程的主要目标是计算出满足要求的一个局部观察到子策略的映射函数 $\delta_i : \Omega_i \to Q_i^t, \forall i \in I$。只要这个映射关系被计算出来，就可以方便的构建第 $t+1$ 步迭代的策略树。

计算出这个映射以后，可以根据下列过程构建策略树：对于任意智能体 i，选择动作 a_i 作为新策略树的根节点，然后对每一个观察分支，根据映射 δ_i 赋予相应的子策略树。图 4.1给出了两智能体问题策略树构建的具体过程，其中使用了下列子策略映射——智能体 1 为 $\delta_1 : o_1 \to q_3, o_2 \to q_1$，智能体 2 为 $\delta_2 : o_1 \to q_2, o_2 \to q_3$。在和每个联合行动相关的策略树被构建出来后，就可以通过计算这些策略树的值来为信念点 b 选择一组最好的策略树。也就是说选择一个最好联合行动 $\vec{a}* = \langle a_1^*, \cdots, a_n^* \rangle$ 以及它所对于的策略树，使得:

$$\vec{a}* = \arg\max_{\vec{a} \in \vec{A}} V^{t+1}(\vec{a}, b) \tag{4.2}$$

定理 4.1.1. *对于一个给定的信念点 b，PBPG 算法构建出的最优联合策略的值与 MBDP 算法在每一步的完全备份中计算出的最优策略值相同。*

证明. 注意到对 PBPG 算法和 MBDP 算法来说，最底层也就是深度为 1 的策略树都是相同的，即只有一个根节点且节点上是所有可能行动的策略树。假设对于两个算法来说，深度为 t 的策略树也是相同的。对于每个信念点 b，MBDP 算法首先将深度为 t 的策略树作为子树，枚举出所有可能的深度为 $t+1$ 的策略树；然后在从这些深度为 $t+1$ 的策略树中选出最大化下列值函数的联合策略 $\vec{q}^{*t+1}$：

$$V^{t+1}(\vec{q}^{*t+1}, b) = \sum\nolimits_{s\in S} b(s)V^{t+1}(\vec{q}^{*t+1}, s) \tag{4.3}$$

其中 $V^{t+1}(\vec{q}^{*t+1}, s)$ 是策略 $\vec{q}^{*t+1}$ 在状态 s 下的值函数。而 PBPG 算法首先为每个联合行动构造出使等式 4.1 的值最大化的策略，然后根据等式 4.2计算出每个联合行动相关的策略的值，并选出值最大的策略树。不失一般性的，这些等式之间有如下关系：

$$\begin{aligned}
V^{t+1}(\vec{q}^{*t+1}, b) =& \sum_{s} b(s)[R(s,\vec{a}) + \sum_{s',\vec{o}} P(s'|s,\vec{a}) \quad O(\vec{o}|s',\vec{a})V^{t}(\vec{q}^{t}_{\vec{o}}, s')] && Eq.4.3\\
=& \sum_{s} b(s)R(s,\vec{a}) + \sum_{s',\vec{o}} [O(\vec{o}|s',\vec{a}) \quad \sum_{s} P(s'|s,\vec{a})b(s)]V^{t}(\vec{q}^{t}_{\vec{o}}, s')\\
=& R(\vec{a}, b) + \sum_{s',\vec{o}} Pr(\vec{o}, s'|\vec{a}, b)V^{t}(\vec{q}^{t}_{\vec{o}}, s') && Eq.4.1\\
=& V^{t+1}(\vec{a}, b)
\end{aligned}$$

其中 $\vec{a}$ 是策略 $\vec{q}^{*t+1}$ 根节点的行动，而 $\vec{q}^{t}_{\vec{o}}$ 是策略树 $\vec{q}^{*t+1}$ 给定观察 $\vec{o}$ 后对应的子树。因此，两种方法生成的 $t+1$ 联合策略对于信念点 b 具有相同的值。综上所述，根据归纳法则上述定理得证。 □

图 4.1给出了 PBPG 算法在两智能体的问题中的一步策略生成过程：(1) 给出的是算法的主要输入，即信念点 b、一个联合行动 $\langle a_1, a_2\rangle$、和用三角形表示的深度为 t 的子策略树；(2) 显示的是在给定信念点和联合行动后，通过计算得出的从每个观察到深度为 t 的子策略树的映射；(3) 给出的是利用这个映射关系构造出来的一个联合策略树。

4.1.2　子策略映射的近似求解

该定理从理论上说明了通过 PBPG 构建的联合策略与 MBDP 算法的计算出的最优策略一样的好。所以问题的关键是如何求解出这样的一个观察到子策略的映射 $\delta_i, \forall i \in I$。注意到在给定一个联合行动时，从观察到子策略的所有可能

的映射个数为 $(|Q_i^t|^{|\Omega_i|})^{|I|}$ 。所以通过蛮力的方法枚举所有可能的映射是非常低效的。事实上，该问题等价于控制理论中的分布式决策问题，而该问题在理论上被证明即便只有两个智能体问题复杂度也是 NP 难的[24]。所以在 PBPG 算法中，采用的是构建和求解一系列线性规划的方法来近似构建这样一个映射。

在近似求解中，PBPG 算法使用了随机映射来方便构建线性规划。随机映射具体定义如下:

$$\pi_i : \Omega_i \times Q_i^t \to \Re, \forall i \in I.$$

也就是说 $\pi_i(q_i^t|o_i)$ 定义了智能体 i 在给定观察 o_i 时对应于子策略 q_i^t 的一个概率分布。同 $\vec{\delta}$ 类似，可以定义联合随机映射为 $\vec{\pi} = \langle \pi_1, \pi_2, \cdots, \pi_n \rangle$。注意到，在给定信念点 b 和联合行动 $\vec{a}$ 时，收益函数 $R(b, \vec{a})$ 是一个常量，所以在求解映射的过程中可以不参加运算。因此，对于最大化策略值函数的目标函数可以改写为:

$$V^{t+1}(\vec{\delta}, b) = \sum_{s', \vec{o}} Pr(\vec{o}, s'|\vec{a}, b) V^t(\vec{\delta}(\vec{o}), s') \tag{4.4}$$

给定随机映射时，该目标函数还可以再次改写为:

$$V^{t+1}(\vec{\pi}, b) = \sum_{s', \vec{o}} Pr(\vec{o}, s'|\vec{a}, b) \sum_{\vec{q}^t} \prod_i \pi_i(q_i^t|o_i) V^t(\vec{q}^t, s') \tag{4.5}$$

给定这样一个目标函数之后，可以用任意最优化的数值计算方法求解随机映射 $\pi_i, \forall i \in I$。在 PBPG 算法的近似求解中，具体使用的方法是先初始化一组随机映射，然后轮流的选择一个智能体，在保持其他智能体策略参数不变的情况下，优化该智能体的映射参数，直到所有智能体的映射参数都无法通过这种方式进一步优化。当对总的参数优化量，也即是优化前和优化后参数在策略值上的差别设定一个最小阈值，则该迭代优化过程能够在有限步数内停止。事实上，当把该问题看出是多智能体的分布式决策时，通过上述方法求解出的解等价于一个 Nash 均衡点。在这个点上，所有的智能体都无法在不改变其他智能体策略的前提下，单独的改进自己的策略。

在迭代优化过程开始前，首先要对每个智能体的随机映射 $\pi_i, i \in I$ 进行初始化。具体做法是，对每一个观察，以均匀的概率随机选择一个子策略 $q_i^t \in Q_i^t$，并在随机映射 π_i 中把该子策略的概率值设为 1，而其他子策略的概率值设为 0。在所有的随机策略都被初始化之后，按任意的顺序轮流的选取一个智能体，在保持其他智能体策略不变的情况下，改进这个被选取到的智能体的策略。保持其他智能体策略不变的意思是对其他智能体实用本次迭代前已经计算好的参数来构建策略。对任意被选定的智能体 i，改进其策略的过程可具体化的描述为寻

表 4.1 线性规划的随机映射求解

Variables: $\varepsilon, \pi_i'(q_i^t|o_i)$
Objective: maximize ε
Subject to:
  Improvement constraint:
    $V^{t+1}(\vec{\pi}, b) + \varepsilon \leq \sum_{s',\vec{o}} Pr(\vec{o}, s'|\vec{a}, b) \sum_{\vec{q}} \pi_i'(q_i^t|o_i) \cdot \pi_{-i}(q_{-i}^t|o_{-i}) V^t(\vec{q}^t, s')$
  Probability constraints:
    $\forall o_i \in \Omega_i, \sum_{q_i^t \in Q_i^t} \pi_i'(q_i^t|o_i) = 1;$
    $\forall o_i \in \Omega_i, q_i^t \in Q_i^t, \pi_i'(q_i^t|o_i) \geq 0.$

算法 4.1　基于信念点的近似策略生成

```
T ← horizon of the DEC-POMDP model
maxTrees ← max number of trees at each step
Q⃗^1 ← initialize and evaluate all 1-step policy trees
for t = 1 to T − 1 do
    Q⃗^{t+1} ← {}
    for k = 1 to maxTrees do
        b ← generate a belief using a heuristic portfolio
        ν* ← −∞
        for a⃗ ∈ A⃗ do
            π⃗* ← compute the best mappings with b, a⃗
            q⃗ ← build a joint policy tree based on a⃗, π⃗*
            ν ← evaluate q⃗ by given the belief state b
            if ν > ν* then q⃗* ← q⃗, ν* ← ν
        Q⃗^{t+1} ← Q⃗^{t+1} ∪ {q⃗*}
    evaluate every joint policy in Q⃗^{t+1}
q⃗*^T ← select the best joint policy from Q⃗^T for b^0
return q⃗*^T
```

找一组参数 $\pi_i'(q_i^t|o_i)$ 满足以下的不等式：

$$V^{t+1}(\vec{\pi}, b) \leq \sum_{s',\vec{o},\vec{q}^t} Pr(\vec{o}, s'|\vec{a}, b) \pi_i'(q_i^t|o_i) \pi_{-i}(q_{-i}^t|o_{-i}) V^t(\vec{q}^t, s')$$

其中 $\pi_{-i}(q_{-i}^t|o_{-i}) = \prod_{k \neq i} \pi_k(q_k^t|o_k)$。

表 4.1中给出了求解新参数的线性规划描述。当对所有的智能体线性规划获得的 ε 都足够小时，轮流的迭代优化过程停止并返回计算得到的联合随机映射 $\vec{\pi}$。在迭代求解中，不同的初始随机映射可能会收敛到不同的随机映射值。为了避免陷入局部最优，算法通过多次以不同的初始值进行求解，并从得到的解中选择一个最好的随机映射来作为最终的结果。

4.1.3　算法的描述和复杂度分析

一旦每个智能体 i 的随机映射 π_i 被计算出来，算法就可以根据 $\pi_i(q_i^t|o_i)$ 给出的概率为每个观察分支 o_i 选择一个子策略 q_i^t。算法 4.1给出了 PBPG 算法主要步骤的伪代码。在 PBPG 算法的每一步迭代中，产生的联合策略树的个数不超过 $(|\vec{A}|maxTrees)$，比 MBDP 算法中完全备份产生的联合策略数 $(|A_i|maxTrees^{|\Omega_i|})^{|I|}$ 要少的多。前面提到过，策略的评价也是一个非常耗时的操作，同时有可能耗尽内存，特别是大规模的规划问题。因为策略评价需要为每一个联合策略计算一个在状态空间上的值函数。在 PBPG 的一步迭代中，实际需要被评价的联合策略数是 $(|A_i|^{|I|}maxTrees + maxTrees^{|I|})$，也比原来的 MBDP 算法要少。因为很多的联合策略在计算子策略映射时已经被提早的排除在外而无需评价，所有 PBPG 算法比 MBDP 算法有更高的效率。这个效率还可以通过利用问题转移和观察矩阵的稀疏性进一步提高。所谓的稀疏性就是指在实际问题中，模型的概率函数具有很多 0 值元素。这个性质可以利用来快速的计算 PBPG 算法中的一些中间等式。

和 MBDP 算法一样，PBPG 算法也是通过一个启发函数组合来生成一系列的信念状态点。从策略构造的角度上看，每一个启发式函数能够被用来选择一个策略树的子集。在 PBPG 算法的实现中，主要使用了两类的启发式信息，即 MDP 启发式函数和随机启发式函数。MDP 启发式函数在计算信念点时，在假设 DEC-POMDP 中的智能体都可以完全观察到系统的状态的前提下，计算 MDP 策略所能达到的状态分布。而随机启发式函数的计算就更为简单，在假设所有智能体随机选择动作的前提下，计算当前可达的状态分布。在有些情况下，几个不同的信念点可能选择了一个相同的策略。主要原因是它们描述的状态分布十分的接近以致无法区分不同的策略。对于这些信念点，PBPG 会重新对可达的信念状态进行采样，计算出一些新的不同的信念点。和 MBDP 算法不同，PBPG 算法无需采用递归策略来计算信念点。这里所谓的递归策略就是执行 MBDP 先前计算好的策略来计算可达状态分布。在对于标准测试集的实验中，PBPG 在只利用这两个简单的启发函数的情况下，就极大的改进了计算得到的策略的质量。

在最初的 MBDP 算法中，计算每一步策略的方法是先通过完全备份生成所有可能的策略，然后在通过信念点在这一系列的策略中选择最好的一些作为构建下一步策略的子策略。算法 IMBDP 通过忽略一些概率小的观察分支的方法来改进备份的效率，而算法 MBDP-OC 则利用基于值的观察归并来减轻策略生成的负担。算法 PBIP 通过策略值的上下界来提早的删减不必要的策略。它的 IPG 版本则通过可达状态分析的方法进一步的缩减子策略的数量。在这些改进

算法中，一个共同的思想是通过限制观察或者策略来改进备份操作，从而提高MBDP 算法的效率。而 PBPG 算法则是采用全新的更加有效的策略生成方法来完全代替策略备份操作。为了避免产生大量的策略树，PBPG 算法通过线性规划为每一个信念点单独生成了一个很小的备选策略集。而这个备选策略集中策略的个数不超过所有的联合行动的个数。这就改变了原来 MBDP 算法一步迭代生成的策略数呈指数式增长的局面，从根本上改进了 MBDP 算法的效率。事实上，之前在 MBDP 的备份操作上的一些改进方法，例如观察归并和增量式策略生成，也可以被利用来进一步的改进 PBPG 的效率。对观察进行归并以及缩减可能的子策略数可以大量的减少线性规划模型中参数的个数，从而提高线性规划的速度和减少不必要的空间利用。

4.2　基于测试反馈的近似策略评估

在单智能体的 (PO)MDP 模型中，有一类被称为实时动态规划的算法（RTDP）[32]。这类算法采取的是基于测试的方法来对可达的（信念）状态进行值迭代。在基于测试的算法中，智能体能够直接的与环境进行交互，并且只需要对它们所遇到的情况进行决策。因此，只需要对局部的（信念）状态进行策略或值的改进，而不需要对整个（信念）状态空间进行规划。在单智能体的规划算法中，基于测试的方法被认为是缓解维度诅咒的一个非常有效的途径。TBDP 算法将基于测试的求解思想引入到多智能体的离线规划中。在离线状态下，智能体不能够与环境发生交互，但是给给定 DEC-POMDP 模型时，很容易设计一个与实际环境相同的仿真器。TBDP 算法可以在离线的情况下，通过与仿真器的交互进行规划。

需要说明的是 TBDP 依然是 MBDP 的一个改进，因此在构建策略树时采用的还是自底向上的动态规划。在每一步的迭代过程中，TBDP 首先通过测试的方法自顶向下采样一组状态，然后根据这组状态组成的状态分布来对联合策略进行参数改进。其中，策略评价也是采用足够多的测试集来完成的。算法 4.2中给出了 TBDP 算法的主要步骤。下面针对这些步骤进行详细的说明。

4.2.1　信念状态生成

在单智能体的 POMDP 模型中，信念状态是基于系统状态的一个概率分布，即 $b \in \Delta(S)$。在 DEC-POMDP 中，仅仅考虑系统的状态不足以完全决策，因为每个智能体在选择自己的策略时还需要兼顾到其他智能体的行为。为此有些学者提出了基于其他智能体信念的信念状态。但是这会导致无止境的信念嵌套，

算法 4.2　基于测试反馈的动态规划

```
Generate a random joint policy
for t=1 to T do // bottom-up iteration
    foreach unimproved joint policy q⃗^t do
        Sample a state distribution b^t by trials
        repeat
            foreach agent i do
                Fix the policies of all the agents except i
                begin formulate a linear program
                    if V(s', q⃗^{t-1}) is required then
                        Evaluate s', q⃗^{t-1} by trials
                Improve the policy q_i^t by solving the LP
        until no improvement in the policies of all agents
return the current joint policy
```

因为其他智能体的信念里也包含了该智能体的信念。另一种方法是基于系统的状态和其他智能体策略的概率分布，这种分布也被称为多智能体的信念状态。考虑其他智能体的策略而不是信念，可以避免信念的嵌套给推理带来的困难，但前提是必须首先知道其他智能体的备选策略集。在自底向上的动态规划算法中，其他智能体的策略的备选集是已知的。像 PBDP 和 MBDP 算法都可以很方便的利用上一步迭代保留的策略构建当前的备选策略集。所以只需要考虑状态的概率分布，这个分布在多智能体系统中也被称为联合信念状态。为了方便起见，如不加说明下文中的信念状态均指联合信念状态。

通常信念状态 b' 可以通过贝叶斯（Bayes）公式递归的计算出来：

$$b'(s') = \alpha O(\vec{o}|s', \vec{a}) \sum_{s \in S} P(s'|s, \vec{a}) b(s) \tag{4.6}$$

其中 α 是归一因子而 b 是上一步决策的信念状态。但是这么计算的效率不高，特别是在环境的状态很多时，其主要原因即是在每次计算时都需要考虑所有的状态。在 TBDP 算法中，信念的计算是采用如算法 4.3描述的测试过程来完成。这个测试过程可以是同实际环境的交互或者是通过运行仿真器获得。注意到在动态规划的计算中，采样所需的策略还没有被计算出来。为了使采样的过程更具有针对性，也即是更能反应实际的可达状态，TBDP 的信念采样同时使用了多种启发式策略。例如最常用的随机策略和 MDP 策略。需要特别指出的是，通过这种方法计算出来的信念状态是非常稀疏的。也就是在状态的分布中，只有少数的状态具有非零的概率值。所以在存储和表示信念状态时可以采用稀疏向量来节约空间和时间。

算法 4.3　基于启发策略是信念采样

Input: t: the sampled step, δ: the heuristic policy
$b(s) \leftarrow 0$ for every state $s \in S$
for *several number of trials* **do**
  $s \leftarrow$ draw a state from the start distribution b^T
  for k=T **downto** t **do** // top-down trial
    $\vec{a} \leftarrow$ execute a joint action according to δ
    $s', \vec{o} \leftarrow$ get the system responses
    $s \leftarrow s'$
  $b(s) \leftarrow b(s) + 1$
return the normalized b

4.2.2　策略的参数改进

在离线规划的策略表示中，策略树可以看成是一个带条件的执行方案。在这个方案中，节点是要执行的动作，而边代表的是根据观察进行选择的分支。在策略树的执行阶段，每个智能体根据获得的观察，逐步执行从根节点开始到也节点的一条路径上的所有动作。在传统的策略树表示方法中，每个动作和分支的选择都是确定性的，也就是说在节点的时候确定性的选择一个动作，而获得一个观察时，可以确定的选择一个分支并执行该分支所对应的子树。在 TBDP 中，引入了随机策略树的表示方法。在随机策略树中，节点存储的不再是一个动作，而是一个关于动作的概率分布；每个观察分支存储的也不是一个子树的指针，而是一个关于所有子树的分布。使用随机策略树的优势是能方便 TBDP 算法中策略的改进。事实上，随机策略树与在无限阶段决策问题中采用随机有限控制器（Finite State Controller）十分相似。在博弈论中，这种带随机概率的策略也被称为混合策略（Mixed Strategy）。

定义 4.2.1 (随机策略树). 智能体 i 的深度为 t 的随机策略树 $q_i \in Q_i^t$ 可以形式化的定义为一个二元组 $\langle \psi_i, \eta_i \rangle$，其中：

- ψ_i 是一个动作选择函数，定义了在智能体动作集 A_i 上的一个概率分布 $p(a_i|q_i)$。
- η_i 是一个分支转移函数，定义了给定观察 o_i 时子随机策略树 $q_i' \in Q_i^{t-1}$ 上的一个概率分布 $p(q_i'|q_i, o_i)$。

函数 ψ_i 和 η_i 一起定义了一个联合分布 $p(q_i', a_i|q_i, o_i) = p(a_i|q_i)p(q_i'|q_i, o_i)$。在 TBDP 算法开始运行前，所有策略树都被预先生成，并对每个策略树 $q_i \in Q_i^t$ 的参数进行随机赋值。注意到，每个智能体 i 的任一决策周期 t 上的策略的节点个数都是相同的，而且是预先固定的。

$$
\begin{aligned}
\text{Maximize} \quad & \sum_{\vec{a}} x(a_i|q_i) \prod_{k\neq i} p(a_k|q_k) R(b,\vec{a}) + \sum_{\vec{a},\vec{o},s'} P(s',\vec{o}|b,\vec{a}) \sum_{\vec{q}'} x(q_i',a_i|q_i,o_i) \\
& \qquad \cdot \prod_{k\neq i} p(q_k',a_k|q_k,o_k) V(s',\vec{q}') \\
\text{subject to} \quad & \forall_{a_i} x(a_i|q_i) \geq 0, \forall_{a_i,o_i} \textstyle\sum_{q_i'} x(q_i',a_i|q_i,o_i) = x(a_i|q_i), \\
& \textstyle\sum_{a_i} x(a_i|q_i) = 1, \forall_{a_i,o_i,q_i'} x(q_i',a_i|q_i,o_i) \geq 0.
\end{aligned}
$$

表 4.2 线性规划的随机策略改进

一个联合随机策略树 $\vec{q}$ 在状态 s 下的值可以根据下列等式进行递归的计算：

$$
V(s,\vec{q}) = \sum_{\vec{a}} \prod_{i} p(a_i|q_i) R(s,\vec{a}) + \sum_{\vec{a},\vec{o},s'} P(s',\vec{o}|s,\vec{a}) \sum_{\vec{q}'} \prod_{i} p(q_i',a_i|q_i,o_i) V(s',\vec{q}') \tag{4.7}
$$

其中 $P(s',\vec{o}|s,\vec{a}) = \sum_s P(s'|s,\vec{a}) O(\vec{o}|s',\vec{a})$。如果给定的是信念状态 b，则联合随机策略 $\vec{q}$ 的值可以计算为 $V(b,\vec{q}) = \sum_s b(s) V(s,\vec{q})$。因此，策略的改进过程就可以归结为找到一组新的参数使得在给定信念状态 b 的情况下策略的值最大，这个策略可以由 $\vec{q}^* = \arg\max_{\vec{q}} V(b,\vec{q})$ 计算得到。

TBDP 所采用的改进策略参数的方法是轮流选择智能体，在保证其他智能体策略的参数不变的情况下，计算该智能体参数的一个改进。该方法与前面提到的 PBPG 算法中采用的方法在基本思想上类似，不同的是 TBDP 的参数优化过程是在新的随机策略的结构下进行的。给定任意智能体 i 的策略 q_i，改进参数的过程可由求解表 4.2中的线性规划来完成。线性规划的目标是最大化智能体 i 的策略对整个团队联合策略值的贡献。而约束条件保证了所有策略参数的分布满足概率的基本性质，即加和为 1。从本质上说，TBDP 中采用的预先初始化得随机策略与 MBDP 中子策略树重用的思想是一致的。因为在随机策略中，每一层（即每一个决策周期）中的总节点数都是固定的，这些节点通过分支选择函数与下层的节点之间相连接，这样就可以通过调整分支选择函数的参数达到组合任意子策略的目的，从而对子策略进行重用。和 PBPG 算法一样，策略改进过程通过不同初始参数的重启计算来摆脱局部最优的限制。

4.2.3　策略期望值评价

通过等式 5.2来计算一个策略的评价值是一个非常耗时的过程，因为在等式右边的计算需要考虑每个状态每个观察和每个子策略。同时值矩阵的表示形式也是非常耗费空间的，特别是对状态非常多的问题，因为需要计算一个保护

算法 4.4　基于测试的策略值评价

Input: $s, \vec{q}^{\,t-1}, V$: the value table, c: the count table
for *several number of trials* **do**
  $s' \leftarrow s, v \leftarrow 0, w \leftarrow \emptyset$
  for k=t **downto** *1* **do** // forward trial
    if $c(s', \vec{q}^{\,k}) \geq numTrials$ **then**
      $w^k \leftarrow \langle s', \vec{q}^{\,k}, V(s', \vec{q}^{\,k})\rangle$ and **break**
    $\vec{a} \leftarrow$ execute a joint action according to $\vec{q}^{\,k}$
    $s'', \vec{o} \leftarrow$ get the system responses
    $r \leftarrow$ get the current reward
    $w^k \leftarrow \langle s', \vec{q}^{\,k}, r\rangle$
    $\vec{q}^{\,k-1} \leftarrow \vec{q}^{\,k}(\vec{o})$
    $s' \leftarrow s''$
  for k=*1* **to** *length(w)* **do** // backward update
    $s', \vec{q}, r \leftarrow w^k$
    $n \leftarrow c(s', \vec{q}), v \leftarrow v + r$
    $V(s', \vec{q}) \leftarrow [nV(s', \vec{q}) + v]\,/(n+1)$
    $c(s', \vec{q}) \leftarrow n + 1$
return $V(s, \vec{q}^{\,t-1})$

每个状态和每个联合策略的表格。前面提到过，在许多实际问题中，实际可达的状态都是一个非常小的集合，因此没有必要对所有的状态计算一个值函数。特别注意到联合策略的值函数在策略计算过程中起到得唯一作用是用来构建表 4.2中的线性规划过程。因此在 TBDP 采用惰性评价（Lazy Evaluation）的方式来计算策略值，也就是说 TBDP 算法只计算需要在线性规划构建过程中需要的策略值函数。这种有针对性的计算方法有效的提高了时间和空间的利用率。

更确切的说，根据表 4.2，智能体 i 的子策略 q'_i 的值函数 $V(s', \vec{q}')$ 需要被计算的必要条件是：

$$P(s', \vec{o}|b, \vec{a}) \prod\nolimits_{k \neq i} p(q'_k, a_k|q_k, o_k) > 0 \tag{4.8}$$

很显然，当该等式的左边等于 0 时，表中 $x(q'_i, a_i|q_i, o_i)$ 的系数永远为 0，因此无需计算和该系数相关的值函数。因此，和预先计算好所有策略在各个状态下的值函数的方法不同，TBDP 算法将子策略的评价推迟到了计算新策略参数的过程中。这样就无需计算那些永远不会被线性规划利用到的策略和状态的值函数，因此有效的节约了时间和空间。更重要的是，基于测试的方法比直接计算等式 5.2的效率要高，因此该方法为近似评价策略值提供了一个有效的新途径。

当需要一个策略的值函数，可以通过算法 4.4中描述的基于测试的方法来计算。从直觉上说，基于测试的评价方法最大的缺陷在于需要多次将策略从当前步遍历到最后一步。为了评价联合策略 $\vec{q}^{\,t}$，算法需要多次执行节点序列

算法 4.5　多线程的异步策略评价

```
repeat in parallel // run on other processors
    t ← the current step of the main process
    foreach joint policy q⃗ of step t-1 do
        S ← sort states in descending order by c(s, q⃗)
        foreach s ∈ S do
            while c(s, q⃗) < numTrials do
                V(s, q⃗) ← evaluate s, q⃗ by trial
until the main process is terminated
```

$\vec{q}^{t-1}\cdots\vec{q}^{1}$。对于长周期的规划问题，每一次的遍历都可能要耗费较多的时间，特别是对最后一步迭代的值函数计算，需要遍历所有的决策过程。为了提高测试计算的效率，TBDP 算法为测试中经历过的状态和策略节点 $\langle s,\vec{q}\rangle$ 维护一个哈希计数器 $c(s,\vec{q})$。如果节点 $\langle s,\vec{q}\rangle$ 已经被遍历过足够多次，则表明该节点的值函数已经足够准确，可以直接返回其值而无需再对其后继节点进行测试。每一次测试到节点 $\langle s,\vec{q}\rangle$ 时，将其计数器 $c(s,\vec{q})$ 加 1，同时根据获得的测试值更新值函数 $V(s,\vec{q})$。通过这个值函数缓存（Value Cache）的方法，可以有效的减少每一条测试所需要的时间。特别是对一些经常要访问的的节点，效果尤其明显。类似于单智能体的 RTDP 算法，TBDP 算法具有如下收敛性：

性质 4.2.1. 如果在测试集中，每个策略和状态节点都被访问过无限多次，则通过测试计算出的策略值函数收敛于策略的实际值函数。

4.2.4　算法的多线程实现

基于测试的方法最主要的特点之一就是不需要对状态进行系统的计算，也不要求测试的过程按一定方式组织。测试集的计算可以是通过异步的增量式的方法完成。这个特点恰恰满足多处理器并行计算的需求，因为在并行计算中，不同处理器之间的通信和同步往往开销最大。在之前的动态规划算法，每一步的迭代都是有次序的，需要先生出所有的备选策略，然后对所有的策略进行评价，最后在从其中选择最好的作为下一步迭代的子策略。如何在这些算法中利用高效的多线程资源不是一个简单的问题。而基于测试方法异步增量式的特点恰恰方便了多线程算法的实现。很显然多线程多处理器算法能够同时利用更多的计算资源，因此在大规模问题求解中具有很大的优势。特别是一些需要大型的高性能计算机群来求解的问题，类似 TBDP 的基于测试的规划能够被容易的实现。算法 4.5 中给出了基于测试的策略评价的多线程实现的主要步骤。在算法中，每一个处理器可以选择一个策略节点来计算测试集。TBDP 还根据节点

计数器 $c(s,\vec{q})$ 进行从大到小的排列，使得经常被访问到的状态能够被优先计算。显然，测试集越大获得的策略值估计就会更加的准确。通过多线程算法的实现，可以更好的利用当前多核多处理器的计算资源，从而提高 DEC-POMDP 规划算法的问题求解规模。

4.3 实验结果

在这一节中，分别对 PBPG 算法和 TBDP 算法设计了不同的实验。在 PBPG 算法中，主要是针对不同的 maxTrees 参数进行实验。参数 maxTrees 决定的是每一次迭代中所能保留的子策略数，而 PBPG 的主要优势就能够大幅度的提高 maxTrees 的值从而提高求解质量。在 TBDP 算法中，主要通过对大状态问题求解来体现 TBDP 算法的优势。

4.3.1 基于信念点的策略生成算法实验

实验挑选了 DEC-POMDP 的标准测试集中最具挑战性的问题。从算法的角度，PBPG 和 MBDP 一样都有相对于决策步数的线性的时间和空间的复杂度。因此，这组实验的主要目的是比较不同算法之于 maxTrees 的运行效率。从直观上说，增加 maxTrees 能够提高策略的质量，同时也会大幅度的增加时间和空间的使用量。实验的结果主要同当前最好的 DEC-POMDP 离线算法 PBIP-IPG[19] 进行比较，因为该算法从运行效率上说大大超越了之前的 MBDP、IMBDP、MBDP-OC 以及 PBIP 算法。在这些算法中，参数 maxTrees 决定了每步保留的子树的数量。通常来说，构造下一步的子树越多，产生的新的策略树的质量也就越高。这组实验通过不断的增加 maxTrees 的值来展示 PBPG 算法在策略生成上的优势。同时，不同的 maxTrees 参数的使用也为设计 Anytime 算法提供了一种可能的途径。

在实验中，使用了两种启发式策略，即随机策略和 MDP 策略。为了实验的公平性，算法在使用这两种策略时的比例与原先的 PBIP-IPG 算法相同，即 45% 的信念点使用 MDP 策略，而 55% 的信念点使用随机策略。事实上，PBIP-IPG 还使用了迭代启发策略，也就是把原先算法计算得出的策略再用了生成信念点，然后通过这些信念点计算新的策略。显然，这种新的启发式策略的使用对于求解的质量是有很大的帮助的。但实验中 PBPG 算法仅仅使用的是随机策略和 MDP 策略，这也从一个方面说明了 PBPG 算法的优势——使用简单的启发式策略就能获得比复杂启发式函数更好的效果。同时，在实验中还显示了不同的启发式组合能够对最终策略的质量产生不一样的影响。

表 4.3 PBPG 算法的实验结果

$maxTrees$	PBIP-IPG		PBPG	
	Average Time	Average Value	Average Time	Average Value
Meeting in a 3×3 Grid, $\|S\| = 81,\ \|O\| = 9,\ T = 100$				
3	3084s	92.12	27.21s	87.01
10	x	x	201.50s	93.46
20	x	x	799.90s	93.90
50	x	x	8345.13s	94.79
100	x	x	52231.85s	95.42
$V_{MDP} = 96.08$				
Cooperative Box Pushing, $\|S\| = 100,\ \|O\| = 5,\ T = 100$				
3	181s	598.40	11.34s	552.79
10	x	x	69.12s	715.95
20	x	x	287.42s	815.72
50	x	x	2935.24s	931.43
100	x	x	19945.56s	995.50
$V_{MDP} = 1658.25$				
Stochastic Mars Rover, $\|S\| = 256,\ \|O\| = 8,\ T = 20$				
3	14947s	37.81	12.47s	41.28
10	x	x	59.97s	44.30
20	x	x	199.45s	45.48
50	x	x	987.13s	47.15
100	x	x	5830.07s	48.41
$V_{MDP} = 65.11$				
Grid Soccer 2×3, $\|S\| = 3843,\ \|O\| = 11,\ T = 20$				
3	x	x	10986.79s	386.53
$V_{MDP} = 388.65$				

由于算法本身具有的随机性，所以实验结果中显示的是算法 10 次运行得到的平均时间和收益。所有的运行时间都是精度为 0.01 的 CUP 时间。由于当 maxTrees 较大时，PBIP-IPG 算法无法顺利求解。在表 4.3中，符号“x”表示的是该算法无法在 12 小时内获得解。实验结果还给出了每个问题的 MDP 上界。需要注意的是 MDP 上界并不是十分紧致的上界，因此如果求解的结果与该上界十分接近则可说明实验结果是近似最优的。因为问题本身的最优解无法获得，所以需要通过这个方法来近似判断解的质量。实验中，PBPG 算法是用 Java 语言实现的并且运行在内存为 2GB 主频为 2.8GHz 的四核机器上。线性规划通过使用软件包 lp_solve 5.5 来进行求解。

3×3 格子的相遇（Meeting in a 3×3 Grid）问题[29] 是典型的 DEC-POMDP 问题。本实验使用的版本[19] 总共具有 81 个状态且每个机器人有 5 个动作和 9 个

观察。对于这个问题，表 4.3中的数据表明在参数 maxTrees 相同时，PBPG 算法可以用少得多的时间获得与 PBIP-IPG 算法几乎一样的结果。即使是 maxTrees 为 20 的情况下，PBPG 算法使用的时间依然比 maxTrees 是 3 时的 PBIP-IPG 算法要少。当然和预期的一样，maxTrees 为 20 时能获得比 maxTree 为 3 时更好的策略。最值得一提的是，PBPG 算法能够求解 maxTrees 为 100 的问题，而 PBIP-IPG 在 maxTrees 大于等于 10 的时候就因为运行超时（12 小时）无法获得结果。注意到该问题具有 9 个观察，这使得生产的策略树具有更多需要考虑的分支，因此对于离线算法来说是一个相当难解的问题。比较该问题的 MDP 上界值可以发现，maxTrees 为 100 时，PBPG 求解的值已经十分的接近最优值。

合作推箱子（Cooperative Box Pushing）问题[16] 是另外一个典型的 DEC-POMDP 问题。该问题总共有 100 个状态，每个机器人有 4 个动作和 5 个观察。从表 4.3中可以看出，PBPG 算法在 maxTrees 为 3 的情况下能够获得与 PBIP-IPG 几乎相同的结果，但使用的时间却比后者少了将近 10 倍。随着 maxTrees 的增加，和期望一致，PBPG 算法计算出的策略值也越来越好。但 maxTrees 为 100 时，实验观察到组合启发函数确定的不同信念点生成了大量相同的策略树，这意味着对于这个问题每一步迭代中真正有意义的策略数可能少于 100。处理这些冗余的策略树给 PBPG 算法带来了一些额外的时间开销。事实上，在 maxTrees 为 100 时，PBPG 算法计算出的测量值 999.53 可能已经很接近该问题的最优策略的值。也就是说，通过增加 maxTrees 的值，PBPG 算法能对该问题进行近似最优的求解。

随机火星漫游者（Stochastic Mars Rover）[19] 是标准策略集中一个较大的 DEC-POMDP 问题。这个问题具有 256 个状态，每个机器人具有 6 个动作和 8 个观察。由于状态数较多，该问题在时间消耗上明显的大于前面两个问题。因为 PBIP-IPG 在求解决策周期为 100 时需要时间开销过大，因此在该实验中使用的决策周期数为 20。注意到，PBPG 算法和 PBIP-IPG 算法在决策周期数上都是线性变化的，因此适当的减少决策周期并不影响实验结果的比较。值得注意的是，即便状态数较多，PBPG 算法也能快速的求解该问题。和状态数是 100 的推箱子问题比较，火星漫游者问题的状态数是其 2 倍，但 PBPG 算法使用的时间却没有增加的太多。主要原因是该问题的转移函数和观察函数具有较大的稀疏性。PBPG 算法在 maxTrees 为 100 时的运行速度比 PBIP-IPG 在 maxTrees 为 3 时还要快。同样的，更大的 maxTrees 带来更好的策略值。

为了进一步测试算法在问题状态方面的可扩展性，实验还尝试求解了 2×3 的格子足球问题。这个问题总共有 3843 个状态，每个机器人有 6 个动作和 11 个观察。该问题之前只能通过在线算法近似的求解，而 PBPG 是第一个能够离线完整求解该问题的算法。相比于在线算法，离线算法的优势在于更能完整的

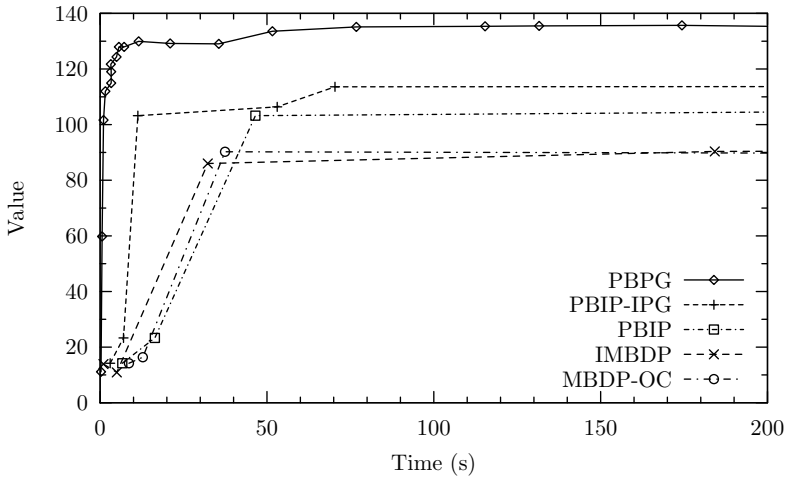

图 4.2 不同算法的运行时间和收益值

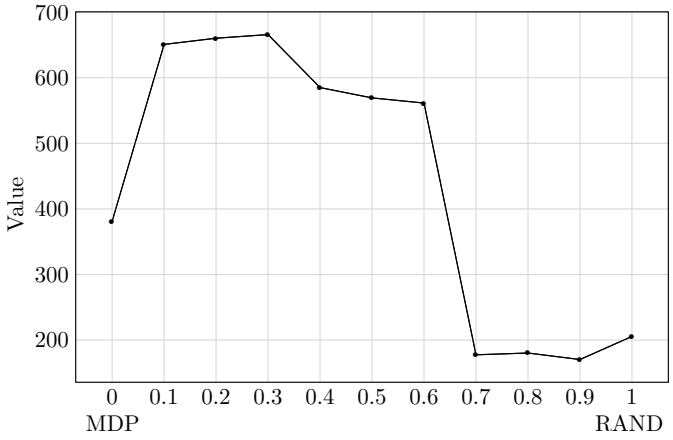

图 4.3 不同启发式组合的策略收益值

考虑整个问题而且可以应用于完全没有通讯的场合，而在线算法往往需要通讯的支持。具体的说，PBPG 算法能够求解出的策略可以在完全没有通讯的情况下获得 386.53 的总收益。对该问题来说，这个策略是近似最优的。

从上面的分析可以看出，参数 maxTrees 不仅决定了每一步迭代的子策略数，还提供了一种权衡算法时间和决策质量的方法。也就是说，当 maxTrees 增加时能获得更好的策略，但相应的离线规划的时间也会增加。对于 10 周期的推箱子问题，实验记录了不同算法在不同的 maxTrees 下所需要的时间和获得的收益值。并将不同算法的时间和收益值的关系通过图 4.2展示出来。从图中可以看出，在给定相同时间时，PBPG 算法获得的收益值要比其他算法要多得多，而且只要很少的时间就能获得其他算法无法获得的收益值。注意到该问题的 MDP 上界值是 175.13。因此，PBPG 算法在收益值上已经十分接近最优。而且需要强调的是，图中比较的各种算法在决策周期上都具有线性的时间和空间复杂度。在这里使用较短的决策周期是因为当决策周期较长且 maxTrees 值较大时，其他算法将无法运行或者运行超时。

在图 4.3中，使用不同的启发式函数比例时，推箱子问题解的收益值。其中 x 坐标表示的是随机策略使用的频率。图中，最左端的 0 点代表的是完全不使用随机策略，也就是全部使用 MDP 策略的情况。而最右端的 1 点代表的是完全使用随机策略而不使用 MDP 策略的情况。中间代表的是两种启发式策略的不同组合。从图中可以看出，对于这个问题最好的值在 $x = 3$ 点取得。也就是说，当 30% 的情况使用随机策略而剩下的 70% 使用 MDP 策略的话，能求解出收益值最高的策略。事实上，实验中默认使用的比例（MDP：45%，随机：55%）并不是一个最优的比例。因此，从实验中也可以看出通过调整组合启发函数中不同策略的比例或者使用新的启发式函数都有助于进一步提高 PBPG 算法的决策质量。

4.3.2　基于测试的策略评价算法实验

这组实验主要求解的问题可以分为 DEC-POMDP 的标准测试集和大规模拓展性问题这两类。在标准测试集实验中，TBDP 算法主要同当前最好的 DEC-POMDP 算法 PBIP-IPG 进行比较，主要目的是为了通过实验证明 TBDP 在标准测试问题中也能取得很好的效果。而拓展性问题主要体现的是 TBDP 算法在求解大规模问题上的优势。本组实验引入的新问题被称为合作废物回收问题。由于当前还没有其他已知的算法能够求解该问题，因此在对该问题的求解上主要从几个不同的角度反映 TBDP 的特点。这些角度的选择都是通过设定 TBDP 算法的不同参数来完成的。和之前的算法一样，TBDP 也的问题求解也有一定的随机性，因此在记录实验结果时给出的是算法 20 次运行的均值。实验结果中的时间是以秒记的 CPU 时间。TBDP 算法是用 Java 语言实现，所有的实验都是运行在一台 2GB 内存主频为 2.8GHz 的四核主机上。在多线程算法的实现上，TBDP 采用的是 Java 的并发程序包来实现。算法中的线性规划求解用的是开源的 lp_solve 5.5 软件包来实现。

标准测试集问题

首先是 DEC-POMDP 的标准测试集问题。实验中选择了三个标准问题，分别是 3×3 的格子相遇问题、合作推箱子问题以及随机火星漫游者问题。这三个问题都是当前 DEC-POMDP 的标准测试集中最难求解的问题。TBDP 算法的实验结果主要是与 PBIP-IPG 算法进行比较。从实验效果上说，PBIP-IPG[19] 算法明显超越之前的 MBDP、IMBDP、MBDP-OC 和 PBIP 这些经典算法，成为当前求解 DEC-POMDP 问题最好的离线规划算法。在这些实验中，TBDP 算法使用

表 4.4 TBDP 算法的实验结果

Horizon	Algorithm	Average Value	Average Time
Meeting in a 3×3 Grid $\|S\|$=81, $\|A_i\|$=5, $\|\Omega_i\|$=9			
100	PBIP-IPG	92.12	3084.0s
	TBDP	92.8	427.1s
200	PBIP-IPG	193.39	13875.0s
	TBDP	192.10	1371.9s
Cooperative Box Pushing $\|S\|$=100, $\|A_i\|$=4, $\|\Omega_i\|$=5			
100	PBIP-IPG	598.40	181.0s
	TBDP	611.0	76.3s
1000	PBIP-IPG	5707.59	2147.0s
	TBDP	5857.40	1327.9s
Stochastic Mars Rover $\|S\|$=256, $\|A_i\|$=6, $\|\Omega_i\|$=8			
10	PBIP-IPG	21.18	976.0s
	TBDP	20.1	40.5s
20	PBIP-IPG	37.81	14947.0s
	TBDP	38.30	119.7s

的测试采样的大小是 20。需要说明的是对于基于测试的算法来说，20 是一个非常小的数。而实验想说明的是即便使用这么小的采样数，TBDP 算法已经能大幅度的超越之前的 PBIP-IPG 算法。表 4.4中给出了标准测试集的结果。从实验结果中可以清楚的看出，TBDP 算法只用了很少的时间就能够达到和 PBIP-IPG 算法几乎一样的值。而且值得一提的是，虽然在实验中使用的 TBDP 多线程实现还十分的初步，在长周期问题的求解中已经很好的体现了它的优势。比如实验观察到，在没有多线程实现的情况下，TBDP 算法对 1000 周期的合作推箱子问题会出现运行超时的状况，而在有多线程实现的情况下，即使是多余 1000 周期的问题都可以顺利的求解。虽然在实验中，TBDP 算法并没有采用并行机群，但将这个多线程实现应用到大型的并行机中是很容易的。而要将 PBIP-IPG 高效和充分的利用多处理器计算资源则是一个非常困难的问题。

合作废物回收问题

实验中引入的合作废物回收问题是一个非常具有挑战性的多智能体规划问题。这个问题的主要特点就是状态数非常的多，而且它代表了在实际应用中一类多机器人的合作问题。该问题主要是从单智能体的机器人导航问题改进而来，环境使用的是代表建筑物内布局的一个地图。如图 4.4所示，在这个问题中两个机器人在一栋建筑物中移动，并适时的对放在特定位置的垃圾箱内的废物进行

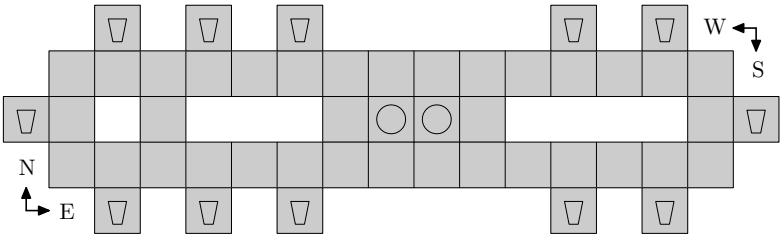

图 4.4 两机器人的合作废物回收问题

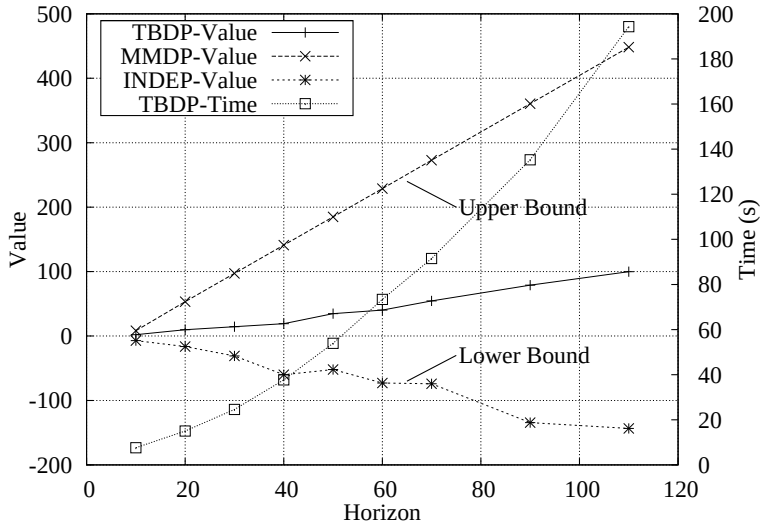

图 4.5 不同决策周期下 TBDP 算法的实验结果

处理。每个机器人有向左转、向右转、向后转、向前移动和留在原地这 5 个动作。它还有 4 个观察用来感知机器人身前的情况，这 4 个观察分别是：墙、其他机器人、垃圾箱和空间空旷。每个动作带有一定的不确定性，有 0.8 的概率能够成功被执行。系统的状态由两个智能体的位置和身体的东南西北四个朝向来决定。每个没有垃圾箱的格子中只能允许一个机器人占有，因此该问题总共有 37824 个状态。收益函数的设计是让两个机器人能够通过合作获得更多的收益。总的来说，该问题中的机器人有两个情况需要合作：一、如果两个机器人同时向一个没有垃圾箱的格子中移动，则有一个机器人会不成功，整个团队会获得值为 5 的惩罚，所以这需要两个机器人通过合作来避免的；二、如果两个机器人同时对同一个垃圾箱进行废物回收，则它们会获得最高的收益 100，但如果它们只是单独的回收一个垃圾箱中的废物，则只能获得 10 的收益。每次当一个垃圾箱被处理结束时，机器人的位置会被随机的重置到地图中的一个位置。为了让机器人能尽快的检查垃圾箱，机器人的每一步执行都有 0.1 的消耗。

图 4.5给出了 TBDP 算法在不同的决策周期下的求解值和运行时间的关系。在这组实验中，TBDP 的测试采样的大小固定为 20。和前面提到的一样，TBDP 算法的值和运行时间随着决策周期数呈近似线性的增长。在图中还给出了两

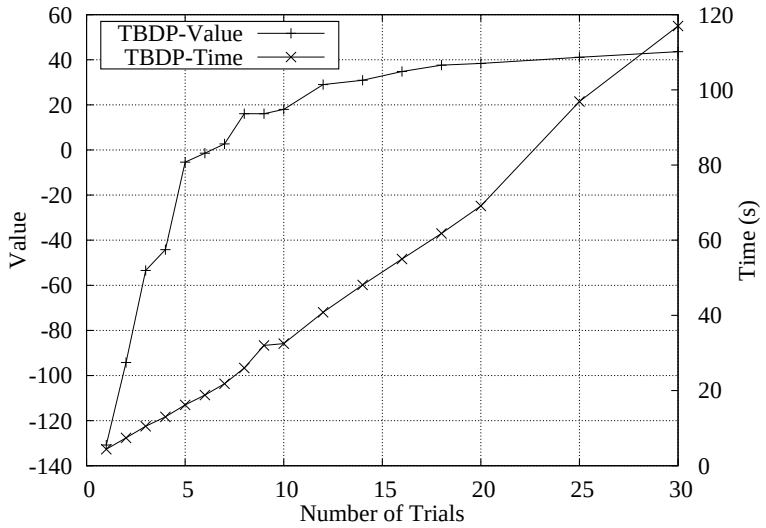

图 4.6 不同测试采样下 TBDP 算法的实验结果

类特殊情形的结果，其中 MMDP 表示对这个问题中的智能体进行集中式的控制且假设它们都能直接获得系统的状态；而 INDEP 则表示的是每个智能体根据它自己的局部观察进行决策而不考虑合作。一般说来，这两种情况提供了 DEC-POMDP 问题的一个可能上下界值。在图中，TBDP 算法的值和上界有着相同的增长趋势，而比下界的结果要好很多。这说明了 TBDP 算法却实让两个智能体在一定程度上进行了合作。需要指出的是，MMDP 代表的上界值是非常乐观的估计，实际 DEC-POMDP 的最优值可能比这个要小的多。

实验还对不同测试采样的 TBDP 算法进行了比较。图 4.6给出了决策周期是 60 的 20 次运行的平均结果。其中可以发现，TBDP 算法的值随着采样的增多而快速增长，在测试采样为 10 时就已经开始趋向于稳定。当然算法的运行时间也随着采样的增多而增长，但是这种增长是线性的。主要原因在于采样的增多会增加 TBDP 算法实际考虑的状态的个数。总的说来，测试采样的大小提供了权衡算法值和运行时间的一个很好的方法。在问题中，如果每一步决策的实际可达状态如果很小的话，TBDP 算法获得足够好值时需要的测试采样集相对来说也会比较小。当然这还要同时考虑启发式策略的效果。不过这体现了 TBDP 算法在考虑状态时具有一定的针对性，能够通过测试反馈的信息快速计算出较好的协作策略。

4.4　本章小结

在这一章中首先介绍的是基于信念点的策略生成算法 PBPG。和其他基于 MBDP 框架的算法一样，PBPG 结合了自顶向下的启发式信息和自底向上的动

态规划来构建完整的联合策略。算法中使用 maxTrees 参数限制了每一步迭代中可以保留的策略树的数目，因此对于决策步数具有线性的时间和空间算法复杂度。PBPG 算法的主要贡献在于提出了一种直接从信念点生成策略的方法，避免了复杂的备份操作带来的指数式的时间和空间消耗。通过使用这种策略生成方法，相比于之前的动态规划算法，PBPG 可以在每一步的迭代中保留更多的策略树作为一下一步迭代的子策略。由于可选子策略树的增加，问题求解的质量有了巨大的提高。而在使用相同的 maxTrees 情况下，PBPG 算法要比其他的算法少的多的时间消耗获得几乎一样的策略值。PBPG 算法另一个重要的特点是在状态空间方面具有较好的扩展性，能够求解之前的离线算法都无法求解的大状态问题。而实验结果也证实了在许多的标准测试问题中，PBPG 算法都比已知的其他算法有更好的表现。

在有了 PBPG 提出的快速策略生成算法后，在 MBDP 求解框架下，策略评价成为了制约算法效率的主要瓶颈。解决这个问题的主要手段是充分的利用状态的可达性，因为在许多的实际问题中虽然整个的状态空间非常的大，但每一步决策中可达的状态却只是其中的很小一部分。MBDP 算法评价策略的方法是为策略计算在所有状态上的一个值函数，而事实上很多状态上的计算是不必要的因为在当前的决策中该状态描述的情况根本不可能发生，也就是通常所说的状态不可达。因此，作为 PBPG 算法的重要补充本章还提出了基于测试的动态规划算法 TBDP。总的来说，TBDP 提供了一种机制用于分析问题的可达性结构并在实际的求解中加以利用，因此对每一步的可达状态数很小的问题是非常有效的。在 TBDP 算法中新引入的策略表示方法同 MBDP 一样使用有限的储存空间，但更加容易被高效的求解。在策略评价上，TBDP 还采用了惰性评价的方法，也就是只有在策略值需呀被应用的情况下才对策略进行评价。和单智能体的 RTDP 算法一样，TBDP 算法能够有效的利用多处理器和多内核的计算资源，更加方便的利用并行和分布式计算的方法来改进算法对于大规模问题的求解效率。实验结果显示，TBDP 算法在标准测试问题上的运行时间相比于同类算法有了数量级上的提高。更重要的是，TBDP 算法能求解状态空间十分巨大的问题。而这些问题对于其他算法是不可能求解的，因此 TBDP 算法大大的扩展了 DEC-POMDP 离线规划算法可求解的问题规模。而且，TBDP 算法的提出也为局部可观察的多智能体学习算法提供一个可能的途径。

第 5 章　无模型的蒙特卡罗规划算法

求解多智能体规划问题的一般过程是先对问题建立 DEC-POMDP 模型，然后利用 DEC-POMDP 的各种离线和在线规划算法来求得智能体可执行的策略。从 DEC-POMDP 问题求解的角度说，虽然当前的相关算法已经有了很大的发展，但这些算法能求解的问题规模依然非常的有限。从 DEC-POMDP 的问题复杂度上看这样的结果并不意外，因为它是 NEXP 难[3] 的问题，也就是最优算法求解的时间和空间是随问题的规模双指数（Double Exponential）式的增长的。从直观上说，求解 DEC-POMDP 问题主要的瓶颈在于信念状态的表示过于复杂且不易计算出一个策略的值函数。而在单智能体的 (PO)MDP 算法中，这些是相对容易做到的。通常，问题的状态和联合行动的个数随着智能体个数的增加呈指数式的增长。所以，要在巨大的联合策略空间中寻找到在各种场合都足够好的策略是十分困难的。

在大规模问题求解中，另一个制约因素就是如何为问题建立 DEC-POMDP 模型。在许多问题中，系统状态的变化都牵扯到复杂的物理过程，这些过程用 DEC-POMDP 的状态转移函数和观察函数是不易描述的。而且 DEC-POMDP 的求解算法对模型完全已知的假设对于很多实际问题来说都是不现实的。例如在机器人足球中，就很难考虑一些随机的因素比如场地上的风。求解在时间域上的智能体控制问题通常使用的是强化学习（Reinforcement Learning）[33] 算法。但是当前大多数的完全合作多智能体的强化学习算法通常假设系统的状态对于所有的智能体是完全可观察的[34]。而在局部可观察的多智能体系统要通过普通强化学习求解策略是极其困难的，主要在于联合策略空间的规模太大，而每个智能体只能获得关于环境的局部信息。

在强化学习中，蒙特卡罗（Monte Carlo）采样是一类可以让智能体仅从同环境的交互中学习策略的方法。该方法无需完整的知道系统模型的信息，而只要求系统能够提供一样用来采样的环境或者仿真器。虽然搭建这样的环境或者仿真器也需要一定的模型信息，但相比于完整的 DEC-POMDP 模型来说，这又是相对容易的。在建立 DEC-POMDP 模型时需要知道完整的关于状态转移和观察的概率分布，而仿真环境只需要为算法提供采样信息。在实际问题中，近似的仿真一个模型往往比计算所有的状态转移和观察函数要容易的多。比如在机器人足球问题中，已经存在各种精度的仿真器。对于实体机器人来说，也可以在场地上多次摆放和运行机器人来测量相应的采样信息。

在这一节中针对离线模式提出了 DecRSPI 规划算法。该方法主要是基于蒙特卡罗采样的思想来进行问题求解，因此无需事先建立完整的 DEC-POMDP

算法 5.1　应用展开式采样的策略迭代

```
generate a random joint policy Q⃗ given T, N
sample a set of beliefs B for t ∈ 1..T, n ∈ 1..N
for t=T to 1 do
    for n=1 to N do
        b ← B_n^t, q⃗ ← Q⃗_n^t
        repeat
            foreach agent i ∈ I do
                keep the other agents' policies q_{-i} fixed
                foreach action a_i ∈ A_i do
                    Φ_i ← estimate the parameter matrix
                    build a linear program with Φ_i
                    π_i ← solve the linear program
                    Δ_i ← Δ_i ∪ {⟨a_i, π_i⟩}
                ⟨a_i, π_i⟩* ← arg max_{Δ_i} Rollout(b, ⟨a_i, π_i⟩)
                update agent i's policy q_i by ⟨a_i, π_i⟩*
        until no improvement in all agents' policies
return the joint policy Q⃗
```

模型。DecRSPI 算法主要是从 TBDP 算法的基础上改进而来的，使用的是同 TBDP 一样的策略表示方法。而且 DecRSPI 算法同样具有 TBDP 算法的诸多优点，例如对于决策周期数的线性时间和空间复杂度。但不同的 TBDP 算法的是，DecRSPI 算法不需要知道完整的 DEC-POMDP 模型，而只需要问题的一个仿真环境。因此在实际应用上，RSPI 算法可以应用在更大规模的问题求解上。

5.1　蒙特卡罗离线规划算法

和 TBDP 算法一样，DecRSPI 算法的求解过程也可以分为信念生成、策略评估和改进这三个过程。不同的是在 DecRSPI 算法中这三个过程都是利用蒙特卡罗采样在不需要完整模型的情况下进行的。算法 5.1中给出了 DecRSPI 主要过程的伪代码。注意到虽然 DecRSPI 算法的策略求解是在离线的情况下通过集中式的方法得出的，但其计算出的策略确实可以完全分布式执行的。在使用仿真器的时候，算法假设仿真器中系统的状态可以根据需要进行设置，而且系统在执行完联合行动后的信息（状态、观察等）可以被获取。事实上，在规划阶段这些信息通常是已知的。注意到这些信息并不等同于 DEC-POMDP 模型的状态转移和观察函数的信息，它是仿真器实时维护的一组量用于表示系统的状态。而状态转移和观察函数则需要考虑所有的状态、动作和观察，然后维护一个完整的表格。这在大规模问题中是不现实的，因为完整的存储这些信息就有可能

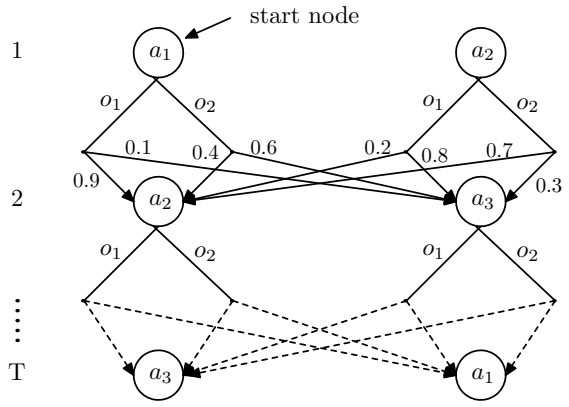

图 5.1 分层随机策略表示

用光系统的所有内存。例如在机器人足球问题中，当前机器人的位置和球的位置是已知的，可以通过坐标、速度等信息来表征。而状态转移函数的闭形式却是不可知的，一是因为状态数太过庞大，二是因为物体间变化的实际物理规律太过复杂，无法推导出相应的数学形式也无法获得完整的变化数据。而当前对于这些问题却有许多工业级的仿真引擎来完成相应的计算，而且逼真程度已经和真实情况十分的接近。能够使用这些仿真器也是 DecRSPI 算法的一大优势。

在算法开始时，DecRSPI 首先通过为第一层到最后一层的节点给定随机参数来进行策略初始化。策略表示成分层的结构，一层策略代表一个决策周期。每一层上有固定个数的决策节点，而每个节点上包含动作选择函数和节点转移函数。节点转移函数定义的是在给定观察时，从上节点转移到下层某节点的概率。图 5.1中给出了这种策略的一个结构示意图。在规划阶段，算法为每个智能体离线的计算出一个这样的策略。在执行阶段，每个智能体根据决策节点的动作分布选择一个动作，然后根据从环境获得的观察转移到下一层的某个决策节点。对于离线算法来说，策略的执行都是简单明了的，所以重点是介绍在规划阶段算法如何生成这样的策略。在每一步迭代中，DecRSPI 首先为每个智能体选择一个决策节点，然后计算出一个到这些节点可能的状态分布，最后根据这个分布对这些决策节点进行优化改进。整个过程不断的持续下去，直到没有智能体的策略能够再进一步改进为止。

5.1.1　基于启发式策略的状态采样

通常要计算状态分布，可以通过递归的贝叶斯更新 $b^{t+1} = Pr(S|b^t, \vec{a}^t, \vec{o}^{\,t+1})$ 从上一步的分布 b^t 来计算当前分布。在 DecRSPI 中，由于状态转移和观察函授都不可知，所以不能通过这种方法来进行计算。在算法中，状态的分布是通过蒙特卡罗的方法来完成的。对于每一个决策周期 t，首先采样 K 个样本 θ^t。每

个样本集 $\theta^t, t \in 1 \cdots T$ 代表的是下列状态分布：

$$b^t(s) = \frac{\sum_{k=1}^{K} \{1 : b_k^t(s) \in \theta^t\}}{K}, \forall s \in S \tag{5.1}$$

其中 $b_k^t(s)$ 代表的是 θ^t 中的第 k 个样本。通过这个方法就可以在没有完整模型的情况下，只利用仿真器来计算状态分布。

另一个重要的问题是如何选择启发式策略。事实上，启发式策略的使用很大程度上依赖与所有求解的具体问题。经常性的，可以通过组合不同的启发式策略来获得更好的计算效果。在没有模型的情况下，首先最简单的启发策略就是随机策略，也就是每个智能体随机的选择一个动作。另一种方式是通过 MDP 的学习方法来学习一个 MDP 策略，然后通过这个策略来进行近似的动作选择。在像机器人足球这样的大规模问题中，要学习这样的 MDP 策略也是极其困难的，因此可以人为的根据常识来手工编码一些策略。或者先利用 DecRSPI 和随机策略来大概的计算出一个策略，然后再根据这个策略来采集样本，计算出新的更好的策略。

5.1.2　基于参数估计的策略改进

在策略改进阶段，DecRSPI 算法着重寻找一个联合策略 $\vec{q}$，从而最大化下列定义的策略值函数：

$$V(b, \vec{q}) = R(b, \vec{a}) + \sum_{s', \vec{o}, \vec{q}'} Pr(s', \vec{o}|b, \vec{a}) \prod_i \pi(q_i'|o_i) V(s', \vec{q}') \tag{5.2}$$

其中 $Pr(s', \vec{o}|b, \vec{a}) = \sum_{s \in S} b(s) P(s'|s, \vec{a}) O(\vec{o}|s', \vec{a})$ 和 $R(b, \vec{a}) = \sum_{s \in S} b(s) R(s, \vec{a})$。注意到 DecRSPI 算法是在没有完整问题模型的情况下进行策略计算的，因此不能通过状态转移和观察函数来计算策略的值。为了在不直接计算策略值的情况下生成和选择策略，DecRSPI 使用了蒙特卡罗的方法来对参数进行近似估计。算法 5.2中给出了主要过程的伪代码，总的来说分为两步：首先为每一个动作 $a_i \in A_i$ 找到一个最好的节点转移函数，生成一个和该动作相关的策略节点；然后通过评估该策略在给定状态分布上的值来选择最好的节点，并改进相应的参数。给定动作和状态分布的情况下，可以通过下面描述的线性规划来计算最大化策略值的的节点转移函数：

$$\begin{aligned} &\text{Maximize}_x \quad \sum_{o_i \in \Omega_i} \sum_{q_i' \in Q_i^{t+1}} \Phi_i(o_i, q_i') x(o_i, q_i') \\ &\text{subject to} \quad \forall_{o_i, q_i'} x(o_i, q_i') \geq 0, \forall_{o_i} \sum_{q_i'} x(o_i, q_i') = 1 \end{aligned}$$

算法 5.2　蒙特卡罗参数估计

Input: b, a_i, q_{-i}
$a_{-i} \leftarrow$ get actions from q_{-i}
for k=*1* **to** K **do**
  $s \leftarrow$ draw a state from b
  $s', \vec{o} \leftarrow$ simulate the model with $s, \vec{a}$
  $\omega_{o_i}(s', o_{-i}) \leftarrow \omega_{o_i}(s', o_{-i}) + 1$
normalize ω_{o_i} for $\forall o_i \in \Omega_i$
foreach $o_i \in \Omega_i$, $q_i' \in Q_i^{t+1}$ **do**
  for k=*1* **to** K **do**
    $s', o_{-i} \leftarrow$ draw a sample from ω_{o_i}
    $q'_{-i} \leftarrow$ get other agents' policy $\pi(\cdot|q_{-i}, o_{-i})$
    $\Phi_i(o_i, q_i')_k \leftarrow$ **Rollout**$(s', \vec{q}')$
  $\Phi_i(o_i, q_i') \leftarrow \frac{1}{K}\sum_{k=1}^{K} \Phi_i(o_i, q_i')_k$
return the parameter matrix Φ_i

在线性规划中，符号 Φ_i 代表的是一个具有如下值的矩阵：

$$\Phi_i(o_i, q_i') = \sum_{s', o_{-i}, q'_{-i}} Pr(s', \vec{o}|b, \vec{a})\pi(q'_{-i}|o_{-i})V(s', \vec{q}')$$

其中 $\pi(q'_{-i}|o_{-i}) = \prod_{k \neq i} \pi(q'_k|o_k)$。注意到由于模型位置，因此不能够通过数学解析的方法来计算这个矩阵，因此算法 5.2的主要任务就是通过蒙特卡罗的方法来近似的计算这个矩阵的值。它首先通过一步仿真的 K 个采样开估计 $Pr(s', \vec{o}|b, \vec{a})$ 的值，然后通过对策略函数 $\pi(q'_{-i}|o_{-i})$ 的另外 K 个采样来计算 Φ_i 中每一项的值。

在 Φ_i 的计算中还需要用到下一层节点的值函数。对于该值的蒙特卡罗估计可以通过从 $\langle s, \vec{q} \rangle$ 开始的展开式采样来完成。具体过程是：首先将仿真器的状态设定为 s，然后执行 $\vec{q}$ 定义的动作，并根据仿真器返回的观察来选择下一层的节点进行执行，直到完成所有剩下的决策周期。通过这个方法获得的 K 个采样可以用来估计 $V(s, \vec{q})$ 的值。从理论上说随着样本数的增加，对 $V(s, \vec{q})$ 的估计只也会更加的精确。事实上，从 $\langle s, \vec{q} \rangle$ 开始的仿真过程可以看成是期望值为 $V(s, \vec{q})$ 的一组随机变量。而 v_k 是该随机变量的一个样本，因此所有样本的均值 $\tilde{V}$ 可以看成是 $V(s, \vec{q})$ 的一个无偏估计。因此可以通过 Hoeffding 定理来分析这个估计的精确程度。

性质 5.1.1 (Hoeffding 不等式)**.** *设 V 是取值范围为 $[V_{\min}, V_{\max}]$ 的随机变量，且 $\bar{V} = E[V]$ 是它的一个期望，$v_1, v_2, \cdots, v_K$ 是随机变量 V 的一组均值为 $\tilde{V} =$*

$\frac{1}{K}\sum_{k=1}^{K} v_k$ 的采样样本；则有下列不等式：

$$\begin{aligned} Pr(\tilde{V} \leq \bar{V} + \varepsilon) &\geq 1 - \exp\left(-2K\varepsilon^2 / V_\Delta^2\right) \\ Pr(\tilde{V} \geq \bar{V} - \varepsilon) &\geq 1 - \exp\left(-2K\varepsilon^2 / V_\Delta^2\right) \end{aligned}$$

其中 $V_\Delta = V_{\max} - V_{\min}$ 是该随机变量所在值域的宽度.

给定样本数 K 和信心阈值 δ 时，估计值 $\tilde{V}$ 的 PAC 容许误差 ε 为:

$$\varepsilon = \sqrt{\frac{V_\Delta^2 \ln\left(\frac{1}{\delta}\right)}{2K}} \tag{5.3}$$

反过来，给定 ε 和 δ 时，满足条件的 $\tilde{V}$ 所需的最少采样数为:

$$K(\varepsilon, \delta) = \frac{V_\Delta^2 \ln\left(\frac{1}{\delta}\right)}{2\varepsilon^2} \tag{5.4}$$

性质 5.1.2. *在给定的样本数 K 趋向无限的情况下，DecRSPI 算法返回的样本均值 $\tilde{V}$ 将收敛于该联合策略的实际收益值 $\bar{V}$。*

5.1.3　算法复杂度分析

注意到在 DecRSPI 算法中，每个智能体策略的层数 T 以及每层中节点的个数 N 都是预先确定的，特别的策略的层数等于问题需要求解的总的决策步数。在每一次的迭代中，算法为每个智能体在预先生成的策略中选择一个需要改进的节点。因此，对于 n 个智能体的问题来说用于存储策略的总的空间需求是 $\mathcal{O}(nTN)$。也就是说是 DecRSPI 算法的空间使用量是随着求解步数和智能体的个数线性的增长。在迭代中，还需要多次进行展开式采样，和决策周期相关的总的时间需求是 $1+2+\cdots+T = (T^2+T)/2$，即决策步数 T 的二次函数 $\mathcal{O}(T^2)$。

定理 5.1.1. *DecRSPI 算法具有相对总步数 T 的线性空间和二次时间复杂度。*

从上面的分析可以看出，算法的空间复杂度相对于智能体的个数 n 来说也是线性的。在迭代过程中，算法每次需要选出 N 个联合策略的节点，然后对每个智能体的节点进行轮流的优化直到所有智能体的策略都无法进一步改进为止。在实际使用中，算法设定了最小的改进阈值（例如 10^{-4}）和最大的允许改进步数（例如 100)。这样策略改进的步骤就能够在有限的时间内停止。从理论上说，策略改进步骤中使用的展开式采样，其时间复杂度并不直接依赖于智能体的个数。不过采样需要使用到仿真器，而对于更多的智能体仿真器需要更多

的时间来进行仿真，因此采样的时间也会受到影响。但是这个是由于仿真器的时间消耗增加造成的，而不是 DecRSPI 算法本身的原因。

定理 5.1.2. *在假定系统仿真时间为 1 的情况下，DecRSPI 算法具有相对于智能体个数 $|I|$ 的线性时间和空间的复杂度。*

5.2　实验结果

5.2.1　标准测试集问题

在实验中，DecRSPI 算法首先求解的是一组 DEC-POMDP 的标准测试问题。虽然在这些问题中完整的 DEC-POMDP 模型是已知的，没有必要通过学习的方法获得策略。但是给出 DecRSPI 算法在这些问题中所学到的策略的质量，并同利用完整模型进行规划的算法做比较，能够更好的评估 DecRSPI 算法在策略学习中的实际效果。为了让 DecRSPI 算法能够运行，实验特地根据这些问题的 DEC-POMDP 模型建立仿真器。在给定状态转移和观察函数的情况下，这样做是非常容易的。有了这些问题的仿真器之后，DecRSPI 算法就可以从仿真器中直接学习到问题的策略。考虑到蒙特卡罗方法的不确定性，实验记录的是算法 20 次运行的平均时间和策略值。在默认的情况下，一层策略的节点数 N 为 3 而蒙特卡罗样本集的大小 K 为 20。

在已知的算法中，能用于学习 DEC-POMDP 策略的并不多，分布式梯度下降（DGD）[5] 的方法是这其中仅有的一个。大多数的学习算法都假设智能体是能够完全观察系统状态的。在 DGD 算法中，每个智能体单独的利用梯度下降的方法来改进自己的策略，因此没有考虑到智能体彼此之间的合作。此外，实验还给出了标准测试问题的离线规划解 PBIP-IPG[19]。注意到这些规划算法利用到了完整的 DEC-POMDP 模型进行规划，因此从效果上来说应该是最好的。实验提供这些方法的结果主要目的是为这些问题给出一个值的上界，看 DecRSPI 算法所计算出来的值是否接近这个上界。如果学习算法的效果接近规划算法，那就说明学习算法计算的策略是可以被接受的。

图 5.2(a)、5.2(b)和5.2(c)中分别给出了 DecRSPI 算法在 3×3 格子相遇问题[29]、合作推箱子问题[16] 以及随机火星漫游者[19] 的实验结果。从这三个实验结果中可以看出，DecRSPI 算法的实验效果都大大的超越了 DGD 算法。特别是在格子相遇问题中，DecRSPI 算法的实验结果与规划算法 PBIP-IPG 十分的接近。这说明了，在这个问题域中 DecRSPI 的学习方法起到了很好的效果，能够接近直接使用模型的规划算法。而在合作推箱子问题中，DecRSPI 和 PBIP-IPG 算法的实验结果差距较大。这是因为在这个问题中，智能体之间的协作关系比

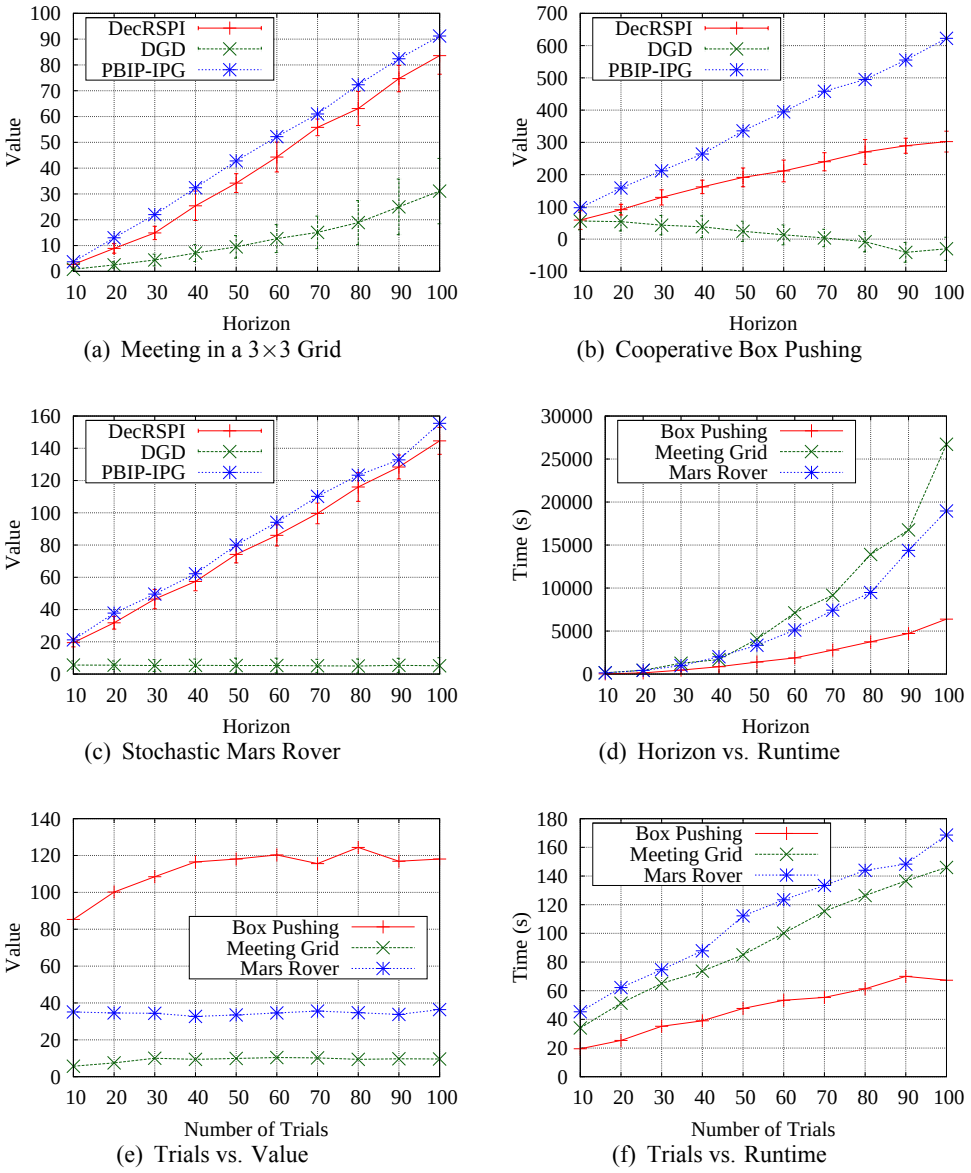

图 5.2 标准测试问题的实验结果

另外两个都要来的复杂，而 DecRSPI 算法并没有显式的处理这些关系。有意思的是，在这个问题中 DGD 算法的实验结果随着决策的周期数下降，这也从一个侧面说明了在这个问题中协作的重要性——决策周期数越长可能发生冲突的机会就越多。

为了进一步分析 DecRSPI 算法中每个参数对实验结果的影响，图中还给出了一组使用不同参数的实验结果。图 5.2(d)中给出的是每个测试问题中 DecRSPI 算法的运行时间和决策周期数 T 之间的关系。从中可以看出，时间随着决策周期的增多大概呈二次函数的增长，这与理论分析的结果是相一致的。图 5.2(e)中给出的是在决策周期 T 为 20 的情况下不同问题中 DecRSPI 算法使用不同的采样数 K 的实验结果。在三个策略问题中，策略的值都随采样数的增加而快速的

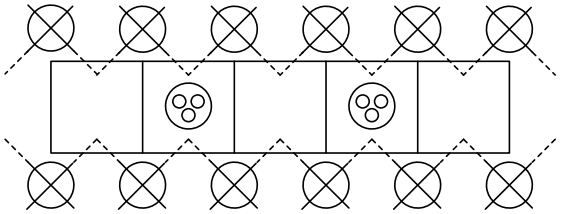

图 5.3 分布式传感器网络问题

增长。在格子相遇和火星漫游问题中，采样数达到 10 时实验结果就已经开始稳定。而在推箱子问题中，采样数要在 40 以上才能得到比较稳定的结果，而且这个结果与 PBIP-IPG 规划算法的结果非常的接近。图 5.2(f) 中的结果显示了 DecRSPI 算法的运行时间随着采样数的增加呈线性的增长。

5.2.2　分布式传感器网络问题示例

分布式传感器网络（Distributed Sensor Network）[35] 问题是标准测试集问题都要大的多的问题。如图 5.3所示，主要两条有着相同个数传感器的链组成。两个链之间被分为若干个格子，每个格子包围着四个传感器。在格子中，有两个需要检测的目标在以相同的概率向左或右的格子移动。每个目标物体的初始能力为 2 个单位。当一个目标物体的能力降为 0 时，表示被捕获并从格子中移走。每个传感器有 3 个动作，分别是检测左边的格子、检测右边的格子和什么都不检测。同时每个传感器有 4 个观察分别代表所监测的左右是否有目标物体。这样，所有的传感器就有 $3^{|I|}$ 个联合行动和 $4^{|I|}$ 个联合观察。这说明该问题的联合动作和观察的空间是十分巨大的，例如在 20 个传感器的问题中，需要考虑的联合行动就有 $3^{20} \approx 3.5 \times 10^9$，而联合观察就有 $4^{20} \approx 1.1 \times 10^{12}$ 个。传感器的每个检测行动会有 1 个单位的消耗。同时当目标物体被格子周围 3 个以上的传感器同时检测到时，其能力减少 1 个单位。当一个目标物体被捕获也就是能力降为 0 时，整个团队获得收益 10。当两个目标物体都被捕获时，问题重启并随机初始化两个目标物体的位置。

引入该问题的主要目的是用以展示 DecRSPI 算法在智能体个数较多时的求解效率。和之前的 DEC-POMDP 标准测试集问题只有两个智能体的情况不同，分布式传感器网络问题可以具有任意多个的智能体（传感器）。对于已知的 DEC-POMDP 算法来说，要求解智能体个数较多的问题是非常困难的，主要就是该问题的联合行动和观察的个数极其的多。一方面，该问题的状态转移和观察函数无法完整的表示；另一方面，联合策略空间更是大的惊人。图 5.4 中给出了 DecRSPI 算法求解决策周期为 10 的分布式传感器网络的结果。和之前的结果一样，DecRSPI 算法能获得比 DGD 算法要好的多的策略值。算法计算的策

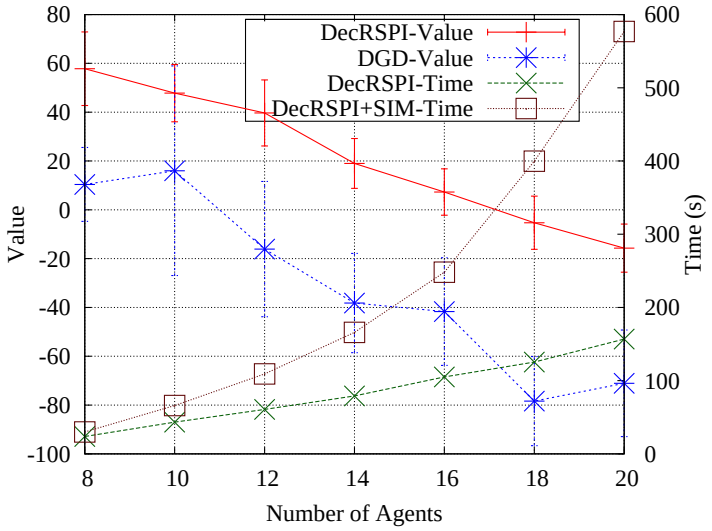

图 5.4 分布式传感器网络的实验结果

略值随着智能体个数的增多略有下降。主要原因是智能体个数增加后，允许目标物体移动的格子数目也随之增加，同时智能体之间的检测行为也增加了系统的消耗（每个传感器的每个检测行动具有 1 个单位的代价）。注意到，DecRSPI 算法在求解这个问题是并没有用到问题的任何特殊结构，同时计算出来的策略也是完全可以分布式运行的，不需要额外的使用通信资源或者对智能体的观察做进一步的假设。图中还给出了算法的实际运行时间：其中 DecRSPI-Time 表示的是单纯 DecRSPI 算法的运行时间，而 DecRSPI+SIM-Time 则同时考虑到了仿真器的时间消耗。和预期的一样，两个时间都随着智能体个数的增多而增加。和理论分析的结果一致，在不考虑仿真器的运行时间时，算法的运行时间 DecRSPI-Time 随着智能体的增加呈线性增长。这说明了 DecRSPI 在求解智能体个数较多的 DEC-POMDP 问题上具有较好的时间复杂度。

5.3　本章小结

在这一节中着重介绍了应用展开式采样的策略迭代算法 DecRSPI。该算法主要应用于完全合作的且具有局部可观察特性的无模型多智能体规划问题，更具体的说就是能在没有完整的 DEC-POMDP 的模型的问题中计算出可运行的分布式策略。DecRSPI 算法的主要特点或者说优势是它的计算过程不要求事先为问题建立良好的 DEC-POMDP 模型，而只需要可以用于采样的问题环境或者说仿真器。在很多实际问题中，问题的 DEC-POMDP 模型通常很难被精确的建立或者说因为模型太大而不能很好的被表示。DecRSPI 可以通过与环境的交换获得知识，并根据这些知识学习出相应的策略。学习出来的策略是完全分布式的，

运行这些策略既不需要进行通讯也不需要智能体能获得环境的全局观察。算法的另一个优势是其计算过程能够只关注那些可达的状态。实验结果表明当智能体之间的交换结构相当稀疏时，算法只需要很少的采样信息就可以计算出很好的策略，这些策略的值甚至能够与当前最好的有模型的规划算法相媲美。更重要的是，DecRSPI 算法有相对于智能体个数的线性时间和空间的算法复杂度。在实验中，DecRSPI 算法能够求解超过 20 个智能体的一般 DEC-POMDP 规划问题。同时，和其他 MBDP 算法一样，DecRSPI 具有相对于决策步数的线性空间复杂度。DecRSPI 的一个可能后继工作是更进一步的利用智能体之间的交互结构来更好的利用采样信息。

第 6 章　总结与展望

6.1　工作总结

在多智能体系统（Multi-Agent System）中，有多个独立行动的智能体参与同环境的交互。环境中的每个智能体是一个独立的决策者，根据它们从环境获得的观察和已有的领域信息来自动的做出决策，并执行相应的行动来达成某些目标。多智能体系统对于人工智能研究的多个方面都是颇有裨益的。特别是当研究的多个主体在功能上和地理位置上具有明显的分布式特性时，多智能体系统的相关工作能够很自然的发挥其效用。典型的领域包括多机器人系统（例如空间探索机器人）和多传感器网络（例如天气监测雷达阵列）等。通常情况下，合作能够使得不同智能体之间的工作更加有效率，而且能够完成它们单独无法完成的任务。甚至在智能体能够集中控制的领域，多智能体系统也能通过并行的动作执行来提高系统的效率、鲁棒性以及可扩展性。一般情况下，多智能体系统中的智能体可以有各自不同甚至相互冲突的目标。本文主要研究的是完全合作的多智能体系统，也就是说系统中所有的智能体只拥有一个相同的目标。

在彼此相互合作的情况下，虽然每个智能体进行独立的决策和动作选择，但作用于整个系统的却是所有智能体的联合行动。因此智能体之间如何彼此协调便是一个需要解决的主要方面。协作的主要目的是将每个智能体的独立行为作为一个整体时能形成对于整个团队的最优或者近似最优的策略。这是一个极其复杂的问题，特别是当智能体活动于具有高程度不确定性的环境中时。例如在机器人足球问题中，每个机器人是一个完全自主的决策个体，但同时它们也是整个足球团队的一份子。因此，足球机器人人之间需要彼此进行合作来赢得比赛。由于客观环境的限制，机器人的传感器和执行机构（例如轮子）将引入相当大的不确定性。使得该问题特别具有挑战性的关键因素还在于：在执行过程中，每个机器人从环境中获得不同的信息，这些信息代表的是环境的一个局部（例如摄像头所对的一个区域)。足球机器人之间能够进行受限的交流，但无节制的完全共享彼此的信息是不可能的。而且在这些环境中，机器人还需要通过长周期的决策来完成射门得分这一目标，这其中涉及到传球、带球和射门等一些列行动。

针对序列性决策（Sequential Decision-Making）问题的研究，存在多种不同的数学模型。其中，从五十年代起就人工智能和运筹学的研究者就对不确定性环境下规划的决策论模型进行了广泛的研究。基于决策论的单智能体规划问题通常可以形式化的表示为一个马尔科夫决策过程（MDP)。在 MDP 中，一个智

能体不断的与随机变化着得环境进行交互，并根据收益或者代价函数确定的评价机制来最优化自己的行为。MDP 模型假设智能体能够完全的获得系统状态的信息，因此问题的主要不确定性主要来源于环境和智能体的行动，也就是动作的具体执行机构。而局部可观察的马尔科夫决策过程（POMDP）将 MDP 模型扩展至智能体信息获取的不确定性。也就是说，智能体通常传感器从环境中获得信息是有噪声的而且只是系统状态的一个局部表现。在多智能体系统中，每个独立的智能体获取到的是关于环境状态和其他智能体的不同局部信息。在过去的十多年中，不同的模型被不同的学者提出并加以研究过。本文采用的分布式局部可观察的马尔科夫决策过程（DEC-POMDP) 是当前最为流行的多智能体决策模型。它建模的是一组彼此需要进行合作的智能体在随机和局部可观察的环境中进行交互的过程。

已经从理论上证明多智能体的分布式决策问题比当智能体的决策问题要困难的多，而且很可能是无法被最优求解的。特别的，DEC-POMDP 模型的问题复杂度是 NEXP 难[3] 的，该结论甚至当只有两个智能体时也一样成立。在过去的几年中，许多基于 DEC-POMDP 模型的近似求解算法被提出。总的来说，问题的求解模式可以分为离线规划和在线规划两类。所谓的离线规划就是在给定模型的基础上，离线的计算出可以被执行的完整策略。离线规划的优势在于没有时间的限制，而且在计算策略时可以是集中式的，只要求被计算出来的策略能够被每个智能体分布式的执行。已有的 DEC-POMDP 上的工作大多数属于这一类。虽然离线规划算法往往能够获得较好的收益值，而且一旦策略被计算出来，在执行阶段便可以完全不需要通讯；但由于需要完整的考虑各种可能的情况，因为需要考虑的策略空间极其庞大，通常只能求解很小的问题。而在线规划算法只需要为当时当地所遇见的情况作出决策，因此不需要考虑完整的情况，能够求解更大规模的问题。但是要分布式的保证智能体之间行为的协调性是十分复杂的，而且每一步可以用于规划的时间非常的有限。所以在进行在线规划时通常使用的是和有选择的通讯策略相结合的做法来改进在线规划的效果。但是通讯资源往往是受限的，或者说对其频繁使用是有代价的，因此在规划的过程中还需要考虑如何更好的使用通讯资源即通讯决策。

对于多智能体规划问题的研究，本文的主要贡献和创新点在于：

一、提出了通讯受限的多智能体在线规划算法 MAOP-COMM。该算法中至少包括以下四个创新点：(1) 一种在执行过程中快速搜索策略的方法；(2) 一种通过独立维护共同的信念池并保证智能体之间行为协调性的方法；(3) 一种有效的基于策略等价的历史信息归并方法；(4) 一种新的基于信念不一致性的通讯决策方法。在多智能体环境中，每个智能体只能获得关于环境和其他智能体的有限局部信息，因此需要对其他智能体所有可能的信念和策略进行推

理，并分析它们对自己决策可能产生的影响。所以即便在只需要为当前情况做出决策的在线规划算法中，智能体仍然需要考虑很多可能的情况才能做出决策。在 MAOP-COMM 算法中使用了一种基于线性规划的策略搜索方法来快速计算当前情况下的一个可行策略。在线规划算法的另一个巨大挑战是可能的历史信息随着决策步数的增加指数式的增长，最终用尽所有的内存。因此，在 MAOP-COMM 算法中提出了一种新的历史信息归并方法，使用该方法可以让在使用有限内存的情况下尽可能的保留有用的历史信息。对于多智能体系统来说，智能体之间的通讯往往能够有效的提高智能体的决策质量。特别是对在线规划算法来说，智能体之间必须独立的在分布式的情况下为全队的行为做出规划，因此智能体之间的通讯就显得尤为重要。但对于多智能体系统来说，通讯资源往往是有限的。所以如何使用有限的通讯资源最大限度的提高智能体之间的决策质量成为在线规划算法所要解决的主要问题。在 MAOP-COMM 算法中，智能体只在信念的不一致性被检测到时才进行通讯，并利用通讯来更新信念池中的历史信息，这种方法能够有效的保证信息共享式的通讯能够被合理的利用从而改善决策质量。

二、提出了针对离线规划的基于信念点的策略生成算法 PBPG。PBPG 算法的主要创新点在于为每一个信念点直接生成所需的策略从而避免了其他算法中复杂的备份过程。同其它基于 MBDP 框架的近似算法一样，PBPG 结合了自顶向下的启发式信念点生成方法和自底向上的动态规划策略构建方法。在每一步迭代中，以往的算法采用的都是先枚举各种可能的策略，然后再根据信念点进行策略选择。这么做主要的问题在于，每一步所需要枚举的策略的个数会出现指数爆炸。所以对于大规模的问题，生成的备选策略会快速的塞满所有的内存空间。PBPG 则避免了策略的枚举直接为每一个策略点构建策略。它将为每个信念点构建策略的问题转化为为每个动作观察对选择最优子策略的映射问题。并证明如果这样的最优映射能够被找到，PBPG 算法生成的策略与 MBDP 算法先枚举再选择出的策略具有相同的收益值，也就是在所有可能的策略中都是最优的。同时 PBPG 算法还分析了选择最优映射的问题复杂度（NP 难问题），继而提出了一种基于线性规划的近似最优映射的计算方法。通过实验可以证明，PBPG 算法的策略生成比最好的 MBDP 算法要快上一个数量级，并且每一步也能够保持更多的子策略。从直观上说，更多的子策略树能够极大的提高算法计算出的策略的质量，而实验结果也进一步验证了这个说法。

三、提出了针对离线规划的基于测试的策略评价算法 TBDP。PBPG 算法在策略生成方面的出色表现使得策略评价成为制约离线算法效率的主要瓶颈。TBDP 算法的主要创新点在于使用基于测试的方法充分的利用到了问题的状态可达性，从而大大的提高了策略评价的效率。在策略生成方法，TBDP 算法还发

展了 PBPG 的直接策略，并提出了一种新的基于层次结构的策略表示方法。使用这种策略，可以使得原先的策略树构建问题转化为决策节点的参数优化问题，从而进一步的提高了求解的效率。在马氏决策理论中定量评价一个策略的方法是为该策略计算一个关于状态的值函数。巨大的状态空间是马氏决策理论中最为棘手的难题，被称为“维度诅咒”。在马氏决策框架内要彻底决策这个问题是困难的，但好在实际的问题往往有许多可以应用的结构，其中状态的可达性便是一个重要的因素。所谓的状态可达性就是在每一步的决策中实际可能遇见的状态通常只是整个状态空间中一个很小的部分，因此在计算策略值函数时无需对所有的状态进行计算只需要对于那些可达的状态进行评价即可。在 TBDP 算法中，基于测试的方法应用到了自顶向下的信念采样和自底向上的策略评价两个方面。而且新的策略结构使得策略计算的过程无需进行策略树的表示和存储，而只需要通过线性规划来优化相应决策节点的参数。这进一步提高了策略计算的效率，同时在使用问题结构方面，TBDP 算法提出了一种惰性的策略评价机制。主要做法就是将策略的评价过程延迟到线性规划的构建阶段，并且只对那些线性规划需要用到的策略值函数进行计算。这一做法进一步减少了策略评价的计算量。同时 TBDP 算法还发挥了基于测试的算法可异步和可并行的特点，提出了策略评价的值缓存和并行评价的方法。这些方法都能够被用来对大规模问题进行求解。在实验部分，TBDP 顺利的求解了上万个状态的 DEC-POMDP 问题，而之前的离线算法只能求解大于上百个状态。

四、提出了基于蒙特卡罗方法的展开式采样策略迭代算法 DecRSPI。该算法主要面向的是智能体个数较多且模型状态转移、观察和收益函数都不可知的 DEC-POMDP 问题。DecRSPI 算法的主要创新点在于利用蒙特卡罗方法在只通过与环境或仿真的交互信息进行策略计算，而且该算法具有关于智能体个数的线性空间和时间复杂度。在许多现实问题中，要完整的建立 DEC-POMDP 模型是困难的。主要的原因有两个：首先，实际问题复杂的物理特性使得状态转移和观察的概率很难被直接计算；再者对于大规模的问题状态空间、动作空间以及观察空间都十分的巨大，即便转移函数的值能够被计算，存储和表示这么大的一个数据结构也是困难的。但现实问题往往可以通过简单的物理规则建立相应的测试环境或者仿真器，机器人足球问题便是最好的例子。而 DecRSPI 的主要特点就是只需要与这些测试环境进行交互便可计算出和有模型的规划算法一样的策略。在具体做法上，DecRSPI 在 TBDP 算法的基础上，通过蒙特卡罗采样对构建线性规划所需的参数进行估计，然后在通过求解该线性规划获得策略。而且还证明只要蒙特卡罗采样的样本足够多，通过 DecRSPI 算法计算出的近似策略值能够收敛到最优最优策略值。而且算法充分的发挥了蒙特卡罗算法的优势，使得算法的计算量不直接依赖智能体的个数。在一般情况下，状态空间、

联合行动、联合观察以及联合策略空间都是随着智能体的个数的增加而指数式的增长。而 DecRSPI 通过蒙特卡罗的方法获得了关于智能体个数的线性复杂度。在实验中，DecRSPI 很好的计算了智能体个数超过 20 个的一般 DEC-POMDP 问题，而之前的算法[15–19] 能顺利计算的问题智能体个数一般都只有 2 个。

综上所述，本文研究的重点是大规模的多智能体规划问题，从决策论的角度这些问题可以用 DEC-POMDP 模型来进行形式化的描述。主要工作包括一系列的 DEC-POMDP 的在线和离线规划算法。从求解方式上说，在线和离线算法各有利弊：离线算法没有时间的限制且可集中式进行，但需要考虑所有可能的情况并计算出完整的策略；在线算法只需要为当前的决策进行规划而且可以灵活的使用通讯，但在规划的时间上有严格的限制且所有的计算必须在分布式的条件下完成。本文提出的 MAOP-COMM 算法着重解决了在线规划需要关注的问题，其中包括：快速的策略搜索、紧致的信息表示和有效的通讯决策。主要特点是在充分利用局部信息的同时保证了在分布式规划的情况下智能体之间行为的协调。本文提出的三个离线规划算法 PBPG、TBDP 和 DecRSPI 主要关注的是当前离线规划算法主要存在的三个缺陷。这三个缺陷分别是：策略生成、策略评价以及对模型的依赖。从算法的效果上说，PBPG 增加了每一步迭代中备选的子策略的个数，TBDP 算法极大的扩张了可求解问题的状态数，而 DecRSPI 算法则增加了可求解问题的智能体个数。这三个算法彼此之间具有紧密的联系，后者都是在前者的基础上进一步发展而来。在模型可知的情况，TBDP 算法能够快速的计算出策略；而当模型不可知的情况下，DecRSPI 算法也能够通过与环境的交互进行问题求解。总的来说，这三个算法从不同的层面拓展了 DEC-POMDP 问题可求解的规模，也为后继的研究提供了相应的基础。

6.2　前景展望

总的来说，多智能体的规划问题都是非常难解的问题。DEC-POMDP 模型的问题复杂度是 NEXP 难[3]，所以说对于一般的问题都无法精确求解。而实际环境中的问题一般都是规模巨大的问题，要求解这些问题就需要设计相应的近似求解算法。而设计近似算法的关键就是如何充分的利用已有问题的特殊结构。对于有些问题，特有的结构是明确的，而且可以通过 DEC-POMDP 模型中的概念来清楚的界定。比如独立状态转移[7] 问题，就代表了一类智能体的局部状态转移不依赖其他智能体行为的问题。但在很多情况下，问题的结构是不明确的，需要算法通过相应的方法来发现和利用这些结构，比如 TBDP 算法利用的就是可达状态较小这个问题结构。DEC-POMDP 问题求解的目标之一就是能够求解智能体个数较多的问题。目前多数基于 DEC-POMDP 模型的算法只能求解很少

数量智能体的问题，一般少于 10 个。DecRSPI 算法在这方面具有很强的可扩展性，在实验中对一般的 DEC-POMDP 问题求解的智能体个数超过了 20，而且证明了该算法具有与智能体个数相关的线性时间和空间复杂度。DEC-POMDP 算法发展的倾向就是要能求解成百上千个智能体的问题。这是和实际应用相关的，例如普通的 Internet 用户就有上亿个，而每个主机都可以看成是一个智能体。当需要求解如此大规模的问题时，当前的 DEC-POMDP 近似算法就不能够满足要求。但主导的思想依然不变，就是要充分的利用实际问题的结构。在类似 Internet 这样的问题中，主要的结构就是网状的交互式结构，也就是说每个智能体只跟网络上有限个数的邻居互相交互。如何有效的利用这个局部式交互的特点将是今后 DEC-POMDP 模型研究的一个重点。

随着 DEC-POMDP 规划技术的不断发展，其在现实社会中的应用也将会越来越多。其中一个重要的变化就是在多智能体的协作当中考虑人的因素。虽然在过去的几十年，多智能体合作的技术有了长足的发展，但这些技术或多或少的依赖一定的预协调（Pre-Coordination）。一些算法[36] 假设策略的规划过程能够完全的在离线的环境下完成，然后每个智能体严格的遵循规划好的策略进行执行。这对有人类参与的问题来说是基本行不通的，因为人的决策行为及其复杂，不可能完全在离线的环境中完成；另一方面即便能够计算出完整的策略，在执行的阶段，人也不可能像一般智能体一样严重的遵照这些策略进行执行。另一些算法[37] 允许智能体在执行阶段进行规划，但是假定了每个智能体执行的是相同的规划算法，而且按照事先规定好的协议进行通讯，而人不可能做到这一点。最近一个被称为“乌合”团队（Ad Hoc Team）[38] 的挑战问题被提出并作为多智能体研究一个长期的挑战。其主要关注的问题就是智能体需要再未预协调的情况下，参与到团队合作之中。而智能体设计的目的就是要能与这些“乌合之众”进行协作，并最终完成预订的任务。这个问题很适合有人参与到智能体团队中的情况，因为人很少能与智能体或机器人在合作之前进行预协调。而智能体需要实时的观察人类的行为，在配合人类行为的同时进行相应的引导。当然，“乌合”团队问题的意义还不仅于此。其最终目标是能够即时的同某些未知的团队成员进行合作，这些未知的团队成员可能是人也可能是某个位置的机器人或者是网络上某台刚刚上线的主机。因此，本文所介绍的在线规划算法便很适合处理该问题，但原有的预协调机制都将改变，例如独立维护一个共同的信念信息和使用相同的启发式函数等。取而代之的是通过观察队友的行为进行学习的能力，以及利用学习到的信息进行推理，然后计算出相对于未知队友更加合适的策略。

参考文献

[1] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 195–210, 1996.

[2] Ranjit Nair, Milind Tambe, Makoto Yokoo, David V. Pynadath, and Stacy Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 705–711, 2003.

[3] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of markov decision processes. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 32–37, 2000.

[4] David V. Pynadath and Milind Tambe. The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of Artificial Intelligence Research*, 16:389–423, 2002.

[5] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie Pack Kaelbling. Learning to cooperate via policy search. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 489–496, 2000.

[6] Claudia V. Goldman and Shlomo Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, 2004.

[7] Raphen Becker, Shlomo Zilberstein, Victor R. Lesser, and Claudia V. Goldman. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004.

[8] Raphen Becker, Shlomo Zilberstein, and Victor R. Lesser. Decentralized markov decision processes with event-driven interactions. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 302–309, 2004.

[9] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked distributed POMDPs: A synthesis of distributed constraint optimization and POMDPs. In *Proceedings of the 20st National Conference on Artificial Intelligence*, pages 133–139, 2005.

[10] P. J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multiagent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

[11] Daniel Szer, Francois Charpillet, and Shlomo Zilberstein. MAA*: A heuristic search algorithm for solving decentralized POMDPs. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, pages 576–590, 2005.

[12] Raghav Aras, Alain Dutech, and Francois Charpillet. Mixed integer linear programming for exact finite-horizon planning in decentralized POMDPs. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, pages 18–25, 2007.

[13] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th National Conference on Artificial Intelligence*, pages 709–715, 2004.

[14] Daniel Szer and Francois Charpillet. Point-based dynamic programming for DEC-POMDPs. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 1233–1238, 2006.

[15] Sven Seuken and Shlomo Zilberstein. Memory-bounded dynamic programming for DEC-POMDPs. In *Proceedings of the 20th Internationall Joint Conference on Artificial Intelligence*, pages 2009–2015, 2007.

[16] Sven Seuken and Shlomo Zilberstein. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the 23rd Conference in Uncertainty in Artificial Intelligence*, pages 344–351, 2007.

[17] Alan Carlin and Shlomo Zilberstein. Value-based observation compression for DEC-POMDPs. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 501–508, 2008.

[18] Jilles Steeve Dibangoye, Abdel-Illah Mouaddib, and Brahim Chaib-draa. Point-based incremental pruning heuristic for solving finite-horizon DEC-POMDPs. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 569–576, 2009.

[19] Chistopher Amato, Jilles Steeve Dibangoye, and Shlomo Zilberstein. Incremental policy generation for finite-horizon DEC-POMDPs. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*, pages 2–9, 2009.

[20] Rosemary Emery-Montemerlo, Geoffrey J. Gordon, Jeff G. Schneider, and Sebastian Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 136–143, 2004.

[21] Maayan Roth, Reid G. Simmons, and Manuela M. Veloso. Reasoning about joint beliefs for execution-time communication decisions. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 786–793, 2005.

[22] Raphen Becker, Victor R. Lesser, and Shlomo Zilberstein. Analyzing myopic approaches for multi-agent communication. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 550–557, 2005.

[23] Simon A. Williamson, Enrico H. Gerding, and Nicholas R. Jennings. Reward shaping for valuing communications during multi-agent coordination. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 641–648, 2009.

[24] John Tsitsiklis and Michael Athans. On the complexity of decentralized decision making and detection problems. *IEEE Transaction on Automatic Control*, 30:440–446, 1985.

[25] Frans A. Oliehoek, Shimon Whiteson, and Matthijs T. J. Spaan. Lossless clustering of histories in decentralized POMDPs. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 577–584, 2009.

[26] Simon A. Williamson, Enrico H. Gerding, and Nicholas R. Jennings. A principled information valuation for communication during multi-agent coordination. In *The 3rd Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains*, 2008.

[27] Rosemary Emery-Montemerlo. *Game-Theoretic Control for Robot Teams*. Doctoral Dissertation, Robotics Institute, Carnegie Mellon University, August 2005.

[28] Maayan Roth. *Execution-Time Communication Decisions for Coordination of Multi-Agent Teams*. PhD thesis, The Robotics Institute, Carnegie Mellon University, 2007.

[29] Daniel S. Bernstein, Eric A. Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1287–1292, 2005.

[30] Christopher Amato, Alan Carlin, and Shlomo Zilberstein. Bounded dynamic programming for decetralized POMDPs. In *AAMAS 2007 Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, 2007.

[31] Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[32] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995.

[33] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[34] Lucian Busoniu, Robert Babuska, and Bart D. Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2): 156–172, 2008.

[35] Xinhua Zhang, Douglas Aberdeen, and S. V. N. Vishwanathan. Conditional random fields for multi-agent reinforcement learning. In *Proceedings of the 24th International Conference on Machine Learning*, volume 227, pages 1143–1150, 2007.

[36] Sven Seuken and Shlomo Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[37] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence*, 175:2:487–511, 2011.

[38] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proc. of the 24th AAAI Conf. on Artificial Intelligence*, pages 1504–1509, 2010.

[39] IF Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[40] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Autonomous Agents and Multi-Agent Systems*, 2009.

[41] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Optimizing memory-bounded controllers for decentralized POMDPs. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 1–8, 2007.

[42] Christopher Amato and Shlomo Zilberstein. Achieving goals in decentralized POMDPs. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 593–600, 2009.

[43] Raphen Becker, Alan Carlin, Victor Lesser, and Shlomo Zilberstein. Analyzing myopic approaches for multi-agent communication. *Computational Intelligence*, 25(1):31–50, 2009.

[44] Raphen Becker, Shlomo Zilberstein, Victor R. Lesser, and Claudia V. Goldman. Transition-independent decentralized markov decision processes. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 41–48, 2003.

[45] R. Bellman. *Dynamic programming*. Princeton University Press, 1957.

[46] Daniel S. Bernstein, Christopher Amato, Eric A. Hansen, and Shlomo Zilberstein. Policy iteration for decentralized control of markov decision processes. *Journal of Artificial Intelligence Research*, 34:89–132, 2009.

[47] Aurelie Beynier and Abdel-Illah Mouaddib. An iterative algorithm for solving constrained decentralized markov decision processes. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 1089–1094, 2006.

[48] Aurelie Beynier and Abdel-Illah Mouaddib. A polynomial algorithm for decentralized markov decision processes with temporal constraints. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 963–969, 2005.

[49] Blai Bonet and Hector Geffner. Solving POMDPs: Rtdp-bel vs. point-based algorithms. In *Proceedings of the 21st International Joint Conference on Artificial*, pages 1641–1646, 2009.

[50] Blai Bonet and Hector Geffner. Labeled rtdp: Improving the convergence of real-time dynamic programming. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling*, pages 12–31, 2003.

[51] Abdeslam Boularias and Brahim Chaib-draa. Exact dynamic programming for decentralized POMDPs with lossless policy compression. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, 2008.

[52] Alan Carlin and Shlomo Zilberstein. Myopic and non-myopic communication under partial observability. In *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 331–338, 2009.

[53] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *National Conference on Artificial Intelligence*, 1994.

[54] Rosemary Emery-Montemerlo, Geoffrey J. Gordon, Jeff G. Schneider, and Sebastian Thrun. Game theoretic control for robot teams. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1163–1169, 2005.

[55] Hector Geffner and Blai Bonet. Solving large POMDPs using real time dynamic programming. In *AAAI Fall Symposium on POMPDs*, 1998.

[56] Mohammad Ghavamzadeh and Sridhar Mahadevan. Learning to communicate and act using hierarchical reinforcement learning. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 1114–1121, 2004.

[57] Claudia V. Goldman, Martin Allen, and Shlomo Zilberstein. Learning to communicate in a decentralized environment. *Autonomous Agents and Multi-Agent Systems*, 15(1):47–90, 2007.

[58] Claudia V. Goldman and Shlomo Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 137–144, 2003.

[59] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the 12th International Conference on Machine Learning*, pages 362–370, 1995.

[60] Janusz Marecki and Milind Tambe. On opportunistic techniques for solving decentralized markov decision processes with temporal constraints. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 825–832, 2007.

[61] H. Brendan McMahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *Proceedings of the 23rd International Conference on Machine Learning*, volume 119, pages 569–576, 2005.

[62] Francisco S. Melo. Exploiting locality of interactions using a policy-gradient approach in multiagent learning. In *Proceedings of the 18th European Conference on Artificial Intelligence*, volume 178, pages 157–161, 2008.

[63] Aaron Christopher Morris, David Ferguson, Zachary Omohundro, David Bradley, David Silver, Christopher Baker, Scott Thayer, Warren Whittaker, and William (Red) L. Whittaker. Recent developments in subterranean robotics. *Journal of Field Robotics*, 23(1):35–57, 2006.

[64] Ranjit Nair, Milind Tambe, Maayan Roth, and Makoto Yokoo. Communication for improving policy computation in distributed POMDPs. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1098–1105, 2004.

[65] Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Dec-POMDPs with delayed communication. In *The 2nd Workshop on Multi-agent Sequential Decision-Making in Uncertain Domains*, 2007.

[66] Frans A. Oliehoek, Matthijs T. J. Spaan, Shimon Whiteson, and Nikos A. Vlassis. Exploiting locality of interaction in factored Dec-POMDPs. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 517–524, 2008.

[67] Frans A. Oliehoek and Nikos Vlassis. Q-value functions for decentralized POMDPs. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 833–840, 2007.

[68] C. H. Papadimitriou and J. N. Tsitsiklis. On the complexity of designing distributed protocols. *Information and Control*, 53(3):211–218, 1982.

[69] Sebastien Paquet, Ludovic Tobin, and Brahim Chaib-draa. An online POMDP algorithm for complex multiagent environments. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 970–977, 2005.

[70] Maayan Roth, Reid Simmons, and Manuela Veloso. What to communicate? execution-time decision in multi-agent POMDPs. In *Proceedings of the 8th International Symposium on Distributed Autonomous Robotic Systems*, 2006.

[71] Maayan Roth, Reid G. Simmons, and Manuela M. Veloso. Exploiting factored representations for decentralized execution in multiagent teams. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 457–463, 2007.

[72] Scott Sanner, Robby Goetschalckx, Kurt Driessens, and Guy Shani. Bayesian real-time dynamic programming. In *Proceedings of the 21st International Joint Conference on Artificial*, pages 1784–1789, 2009.

[73] Jiaying Shen, Victor R. Lesser, and Norman Carver. Minimizing communication cost in a distributed bayesian network using a decentralized mdp. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 678–685, 2003.

[74] Trey Smith and Reid G. Simmons. Focused real-time dynamic programming for mdps: Squeezing more out of a heuristic. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006.

[75] Matthijs T. J. Spaan, Geoffrey J. Gordon, and Nikos Vlassis. Decentralized planning under uncertainty for teams of communicating agents. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 249–256, 2006.

[76] Matthijs T. J. Spaan and Francisco S. Melo. Interaction-driven markov games for decentralized multiagent planning under uncertainty. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 525–532, 2008.

[77] Matthijs T. J. Spaan, Frans A. Oliehoek, and Nikos Vlassis. Multiagent planning under uncertainty with stochastic communication delays. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, pages 338–345, 2008.

[78] Peter Stone and Manuela M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.

[79] Daniel Szer and Franois Charpillet. Improving coordination with communication in multi-agent reinforcement learning. In *Proceedings of the 6th IEEE International Conference on Tools with Artificial Intelligence*, pages 436–440, 2004.

[80] Milind Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[81] Feng Wu, Shlomo Zilberstein, and Xiaoping Chen. Multi-agent online planning with communication. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*, pages 321–328, 2009.

[82] Ping Xuan, Victor Lesser, and Shlomo Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 616–623, 2001.

[83] Shlomo Zilberstein. Optimizing decision quality with contract algorithms. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1576–1582, 1995.

第 A 章　机器人足球问题简介

机器人足球是本文在举例过程中反复提及的问题，现做一个简要的介绍。机器人足球顾名思义就是用机器人来踢足球。如图 A.1所示，与人类的足球相一致，每场比赛也是由两个队伍参加，每个队伍有 11 个球员，包括 1 个守门员和 10 个普通球员。比赛的每一方通过队员们的通力合作，尽可能多的把球射入对方的球门；同时守住己方的球门不让对方攻破。所以说机器人足球是典型的多智能体问题，其中包含了多智能体间的（同队）合作和（异队）对抗。为了配合机器人的决策，比赛过程被离散为了 6000 个周期，上下半场各 3000 个周期。上半场结束后，双方互换场地并重新开球继续进行下半场的比赛。每个决策周期有 100 毫秒的时间供每个机器人做出决策。每个周期开始前，机器人通过自身的传感器从场地上收集信息，包括给自己定位、寻找球的位置以及查看周围的队友和对手等。每个机器人只能获得与自己相关的场上局部位置的信息，如图 A.1中球员前方的漏斗状区域。因此每个球员获得的信息是互不相同的，而且这也是与人类足球的真实情况相符合的。根据从场上收集到的的信息，每个球员独立的进行决策，并在本周期结束前执行相应的行动。可能的行动包括：向前或向后移动、转变身体朝向、踢球拿球等。通过这些原始行动的组合，机器人还可以具有一些高级的行为，例如传球、带球、射门等。每个机器人主要通过视觉获得场地的信息。机器人之间彼此可以进行通讯，但通讯的带宽是严格受到限制的。因为考虑到实际人类足球的比赛中，也不可能进行太多的语言交流，只能是普通的手势或者特定的喊话等。每个队伍的目标很明确，那就是尽可能的多进球少失球并最终赢得比赛。

从决策论的角度上说，机器人足球有合作又有对抗，所以是一个标准的 POSG 问题。每一个足球机器人就是一个单独的智能体，它们有各自的动作集

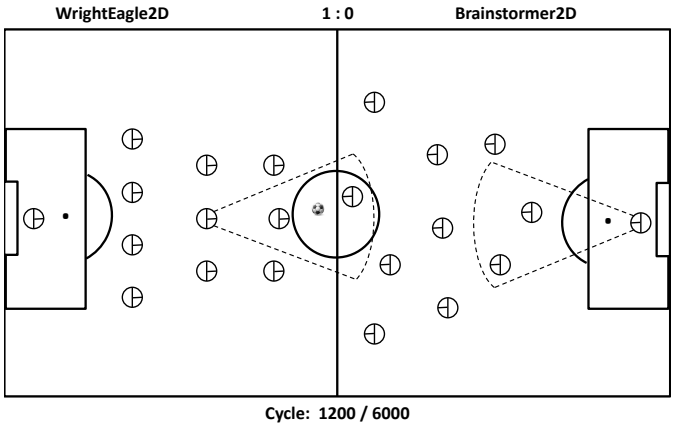

图 A.1 机器人足球仿真 2D 界面

合和观察集合。问题的状态就是场上所有球员和球的位置、速度、加速度等信息的组合。所以从模型上说，机器人足球是一个规模极其大的多智能体决策问题。问题的转移函数，就是每个机器人和球在运动过程中需要满足的力学物理规律。而观察函数与每个传感器的物理特性直接相关。收益函数也很明确，进球获得一个较大的正收益，而失球获得一个较大的负收益，同时每一步的动作执行还需要消耗一定的电能，因此也有一个较小的负收益。赢得比赛这个最终目标就可以通过模型描述为最大化所有收益的总和。在每场比赛中，对手的模型是不可知的，而且在实际的竞赛中还需要通过比赛战胜各式各样的未知对手。所以足球是一个不确定性很大的问题。在实际的处理中，通常给对手假定一个模型，然后根据这个模型来推导出自己这一方球员的决策。为了简化对手建模的过程，通常假设对手遵循的是一个执行固定策略的反应式模型。所以在队员的决策设计上，对手通常可以看成是不确定性环境的一部分。这样，就可以专注队员之间的合作，提高决策的效果。当把对手当成环境一部分时，POSG 问题就退化成完全合作的 DEC-POMDP 问题。

需要特别说明的是本文描述的算法并不能直接的应用到足球机器人的控制上，需要对问题进行进一步的抽象。因为底层行动的控制都是连续的物理量，因此可以将原子动作封装成固定的模块，例如向固定点跑位、向特定队友传球等。同时状态也不能是纯粹的几何位置信息，需要对信息进行进一步的提取，例如计算谁在控球、自己是否处于禁区内等。通过这一些列的抽象过程，最终在高层决策部分可以应用文中叙述的算法。在实际的编码过程中，为了提高算法的效果可能需要对具体问题进行一些特别的优化。但从算法思想的角度上来说具体的步骤是基本不变的。例如 MAOP-COMM 算法的思想主要用于在防守阶段为四个后卫分配统一的防守策略。足球的防守是一个需要队友特别是后卫充分进行合作的问题。在防守过程中，四个后卫各司其职严密的盯好对方的各个前锋，谨防对方的突破。因此 MAOP-COMM 算法就用来在在线的情况下，为每个后卫分配特定的对手并执行相应的防守行为，例如甲球员盯防对方的 A 前锋，而乙球员对持球的 B 前锋进行逼抢等。这需要各个队友在具有不同信息的情况下获得统一的分配策略，漏防、错防的后果是致命的，可能直接导致失球。在比如本文的离线算法主要用来生成各个球员的策略，从而避免复杂的手工编码过程。特别是本文描述的基于测试和采样的方法特别适合机器人足球问题，因为要建立机器人足球的马尔科夫模型是困难的，但可以用于测试和采样的仿真器确实现成的。因此直接通过采样来计算策略对于求解机器人足球问题意义更加的重大。作者从大学二年级起进入实验室，并参加中科大蓝鹰机器人足球队的相关工作，一直努力从理论和实践两个方面对机器人足球等典型大规模多智能体系统的规划问题进行求解。

致　谢

时光飞逝，转眼间在中国科学技术大学九年的求学生涯即将结束，留给我的是一段段美好的回忆。在这美丽安静的校园中度过的是我人生中最重要的一个九年，一路走来有艰辛也有欢笑。在博士论文的撰写即将完成之际，我特别需要感谢我的导师、同学、亲人和朋友们，是你们让我的大学和研究生活变得多彩而充实。正是因为有了你们无私的帮助和支持，我才能在这个季节收获成功并以更加饱满的姿态重新扬帆起航。

首先，我要感谢我的导师陈小平教授。从 2002 年 8 月 1 日正式进入实验室以来，我就一直在您的关心和帮助下成长。加入您领导的多智能体实验室和蓝鹰机器人足球队成为我人生中一个重要的机遇。正是从这时候开始，在您的指导下，开始接触人工智能的前沿研究并逐渐深入，为我今后的学习和工作打下坚实的基础。在研究工作上，您的治学态度、处事风范、还有对研究的敏锐触觉和准确的把握都不断的启发我、引导我定位难题并帮助我寻求突破。在日常生活中，您给予的许多帮助和建议让我受益良多。特别需要感谢的是您给我创造的许多接触前沿研究的经历，包括多次参加的机器人大赛、学术会议和国外交流机会。这些都将成为我人生中最为宝贵的财富，在此对您表示深深的感谢！

其次，我要感谢多智能体实验室和蓝鹰机器人足球队的成员们。感谢杨斌老师，您开设的本科研讨班让我有机会接触到机器人足球并加入蓝鹰机器人足球队。感谢范长杰博士对仿真机器人足球工作上的耐心指导。您一丝不苟的工作态度和认真刻苦的拼搏精神深深的感染了我，同时也让我学习到了很多关于机器人足球的相关知识。和您工作的两年，我们一起收获了一个世界亚军和一个世界冠军，这对我一直是一个很好的激励。感谢实验室的徐凯博士、刘飞博士、刘津甦博士、吉建民博士、薛峰博士、靳国强博士、章宗长博士、王锋博士，你们都在我的工作和生活中给予许多帮助。

再次，我要感谢在美国马赛诸塞大学阿莫斯特分校求学期间，对我进行细心指导的 Shlomo Zilberstein 教授。您对我的帮助和每周一次的讨论，让我在多智能体规划研究领域极大的开阔了视野，并积累了许多新的研究经验。还要感谢实验室的其他成员，同你们的组会和讨论，让我在研究工作上颇多受益。

最后，我要感谢我的父亲、母亲、我的妹妹吴茜茜以及我的爱人周虹薇。没有你们在背后默默的支持和帮助，我就无法顺利的完成学业。你们给予我的亲情是我成长和奋斗最主要的动力。在这里，我向你们表达我最诚挚的谢意。

谨以此文献给我的父亲和母亲！

吴锋

在读期间发表的学术论文与取得的研究成果

已发表论文：

[1] **Feng Wu**, Shlomo Zilberstein, and Xiaoping Chen, Online Planning for Ad Hoc Autonomous Agent Team, In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (***IJCAI***)*, Barcelona, Spain, July 2011.

- Conference Rank: 1

[2] **Feng Wu**, Shlomo Zilberstein, and Xiaoping Chen, Online Planning for Multi-Agent Systems with Bounded Communication. In *Artificial Intelligence(***AIJ***)*, Volume 175, Issue 2, Page 487-511, February 2011.

- EI Indexed, Accession Number: 20110213567282
- SCI Indexed, IDS Number: 720JJ
- Impact Factor: 3.036
- SCI 期刊 1 区

[3] **Feng Wu**, Shlomo Zilberstein, and Xiaoping Chen, Rollout Sampling Policy Iteration for Decentralized POMDPs, In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (***UAI***)*, Page 666-673, Catalina Island, USA, July 2010.

- Conference Rank: 1

[4] **Feng Wu**, Shlomo Zilberstein, and Xiaoping Chen, Trial-Based Dynamic Programming for Multi-Agent Planning, In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (***AAAI***)*, Page 908-914, Atlanta, USA, July 2010.

- EI Indexed, Accession Number: 20104413339564
- Conference Rank: 1

[5] **Feng Wu**, Shlomo Zilberstein, and Xiaoping Chen, Point-Based Policy Generation for Decentralized POMDPs, In *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent Systems (***AAMAS***)*, Page 1307-1314, Toronto, Canada, May 2010.

- Conference Rank: 1

[6] **Feng Wu**, Shlomo Zilberstein, and Xiaoping Chen, Multi-Agent Online Planning with Communication, In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (***ICAPS***)*, Page 321-329, Thessaloniki, Greece, September 2009.

- EI Indexed, Accession Number: 20105213533404
- Best Paper Nominated

[7] **Feng Wu** and Xiaoping Chen, Solving Large-Scale and Sparse-Reward DEC-POMDPs with Correlation-MDPs, In *Proceedings of the Robot Soccer World Cup XI Symposium (***RoboCup***)*, Page 208-219, Atlanta, USA, July 2007.

- EI Indexed, Accession Number: 20083611512089
- CPCI-S (ISTP) Indexed, IDS Number: BIB09
- Acceptance Rate: 13.5%

注： 以上所列会议论文均为全文录取并做口头报告。

其他学术研究成果：

[1] **Feng Wu**, Limin Zhao, Xufeng Han, and Xiaoping Chen, WE2007: WrightEagle Microsoft Robotic Studios Team, *The* **World Champion** *of RoboCup MSRS Challenge in RoboCup 2007*, Atlanta, GA, USA, July 2007. (Team Leader)

[2] **Feng Wu**, Changjie Fan, and Xiaoping Chen, WE2006: WrightEagle Simulation 2D Team, *The* **World Champion** *of RoboCup Simulation 2D Competition*, Breman, German, June 2006. (Team Leader)

[3] Changjie Fan, **Feng Wu**, and Xiaoping Chen, WE2005: WrightEagle Simulation 2D Team, *The* **Second Place** *of RoboCup Simulation 2D Competition*, Osaka, Japan, July 2005.

[4] **吴锋**, 范长杰, 陈小平, WE2006: 蓝鹰仿真 2D 机器人足球队, 中国机器人大赛暨首届 *RoboCup* 中国公开赛仿真 *2D* 组冠军, 苏州, 江苏, 2006.

[5] 范长杰, **吴锋**, 陈小平, WE2005: 蓝鹰仿真 2D 机器人足球队, 中国机器人大赛仿真 *2D* 组冠军, 常州, 江苏, 2005.

注： 作者自 2004 年起成为正式成员，以上所列项目期间均为核心主力。

参与完成的科研项目：

[1] 国家自然科学基金项目（60745002）“慎思式适应的基本机制、框架和实验研究”，承担其中多智能体规划理论部分。

[2] 国家 863 计划项目（2008AA01Z150）“闭环任务的慎思式适应的核心技术与原型系统”，承担其中多智能体规划算法部分。

其他奖励：

[1] 首届全国“博士研究生学术新人奖”(教育部)。

[2] 安徽省普通高等学校品学兼优毕业生。