# 课程作业3：捉鬼敢死队实验报告

**摘　要**：本报告深入研究了贝叶斯网络和概率推理在“捉鬼敢死队”项目中的应用。我们详细介绍了贝叶斯网络结构的构建，包括变量之间的依赖关系及其值域定义，以准确模拟吃豆人游戏环境。实验涵盖了变量消元、精确与近似推理、置信度更新等关键任务。通过自动评分系统的测试，我们证明了所实现方法在模拟游戏环境中的准确性和稳定性。这一研究不仅加深了对概率推理和贝叶斯网络的理解，而且展示了这些概念在AI领域中处理不确定性和动态环境的实用性和效果。

**关键词**：贝叶斯网络、概率推理、吃豆人游戏、精确推理、近似推理、AI应用。

## 1　引言

在人工智能领域，概率推理和贝叶斯网络是两个核心概念，它们在处理不确定性和作出基于不完全信息的决策时发挥着至关重要的作用。本次课程作业的目标是通过“捉鬼敢死队”项目，将这些理论概念与实际应用相结合。在这个项目中，我们被赋予任务，设计一个智能体 —— 吃豆人，来追踪和捕捉隐形的幽灵。

本作业不仅要求我们实现贝叶斯网络的构建和精确推理，还需要运用近似推理技术来处理更复杂的动态场景。通过这一过程，我们可以更深入地理解如何在实时、不确定的环境中应用概率推理。这不仅是对理论知识的实践应用，也是对我们编程技巧和解决问题能力的考验。

吃豆人游戏提供了一个理想的框架，用于探索和实验这些概念。在游戏中，传感器数据的不确定性和幽灵的随机移动模式为我们提供了一个复杂且动态的测试场景。通过在此环境中的编程实践，我们不仅能够巩固我们的理论知识，还能学习如何在实际应用中有效地利用这些概念。

在本报告中，我们将详细介绍我们在完成这项作业过程中的思路、所面临的挑战、解决问题的策略以及所得到的结论。通过这个项目，我们不仅能够更好地理解概率推理和贝叶斯网络，还能够体会到将这些理论知识应用于解决现实问题的重要性和价值。

## 2　理解贝叶斯网络

贝叶斯网络，也称为信念网络或决策网络，是一种用于表示和推理具有不确定性的知识的概率图模型。它基于概率理论和图论，由节点和有向边组成，其中节点表示随机变量，边表示变量之间的概率依赖关系。每个节点都有一个概率表，用于量化其与父节点的关系。这种图形化表示使得贝叶斯网络在处理复杂的概率模型时变得直观且高效。

在贝叶斯网络中进行推理，意味着根据给定的证据更新对某些变量的信念。这通常涉及计算条件概率，即在已知一部分信息的情况下，推断其他未知信息的概率。例如，在吃豆人项目中，我们使用贝叶斯推理来根据传感器读数估计幽灵的位置。

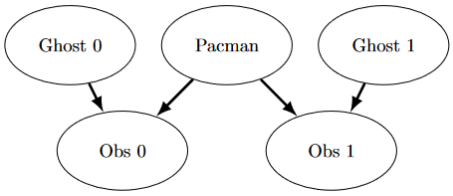
在本项目中，贝叶斯网络的结构被设计为反映吃豆人世界中的不同元素及其相互作用。例如，节点可以代表幽灵的可能位置，而边可以代表位置之间的概率关系。通过这种结构，我们能够计算各种场景下幽灵出现在特定位置的概率。这种应用展示了贝叶斯网络在人工智能领域，尤其是在处理高不确定性情况下的广泛应用价值。

# 3 实验内容

## 3.1 任务一：贝叶斯网络结构（Bayes Net Structure）

### 3.1.1 代码实现

在本部分，我们实现了`constructBayesNet`函数，构建一个贝叶斯网络，其结构根据项目描述中的图1（如下图所示）。




首先，我们定义了吃豆人（`PAC`）、两个幽灵（`GHOST0`和`GHOST1`）以及它们的观测值（`OBS0`和`OBS1`）作为网络节点。为了表达这些变量之间的依赖关系，我构建了如下边：`(GHOST0, OBS0)`，`(PAC, OBS0)`，`(PAC, OBS1)`和`(GHOST1, OBS1)`。

接着，我们定义了每个变量的值域，以确保网络可以准确反映游戏环境。吃豆人和幽灵的位置由地图的宽度和高度决定的所有可能坐标组成，而观测值则基于最大噪声值（`MAX_NOISE`）和地图尺寸来确定所有可能的观测距离。这样的设置允许网络处理声纳传感器可能接收到的所有噪声数据，并据此推断幽灵的位置。

### 3.1.2 实验结果

通过运行`python autograder.py -q q1`，我们获得了如下结果：

```
Provisional grades
==================
Question q1: 2/2
------------------
Total: 2/2
```

这证明我们成功构建了一个能够模拟吃豆人游戏中吃豆人和幽灵交互的贝叶斯网络。这个网络不仅能反映吃豆人和幽灵之间的关系，还能处理由声纳传感器接收的噪声数据。

## 3.2 任务二：连接因子 Join Factors

### 3.2.1 代码实现

在`joinFactors`函数的实现中，我们目标是合并一组因子，并生成包含所有输入因子的非条件变量和条件变量的新因子。具体实现步骤如下：

1. **变量集合的确定与检查**：

   对每个输入因子的非条件变量集合进行了收集。通过`functools.reduce`方法，我们计算了这些集合的交集，确保每个非条件变量只在一个因子中出现。如果发现任何非条件变量在多个因子中出现，则抛出一个`ValueError`异常。

2. **整合变量集合**：

   整合所有输入因子的非条件变量和条件变量。通过转换为集合并执行去重操作，我们创建了新因子的非条件变量和条件变量列表。同时，为了保证一致性，我们从第一个因子中获取了变量域字典（`variableDomainsDict`）。

3. **创建新因子**：

   基于上述确定的变量集合和变量域，我们构建了一个新的因子（`newFactor`）。

4. **概率计算**：

   对于新因子的每个可能赋值，我们通过遍历每个输入因子并乘以它们的概率值来计算总概率。这里我们使用了`factor.getProbability(assignment)`来获取每个因子在给定赋值下的概率，并将这些概率相乘得到最终的概率值，然后使用`newFactor.setProbability(assignment, probability)`来设置新因子在该赋值下的概率。

通过这个过程，我们成功地实现了因子的联合操作，生成了一个全新的因子，该因子在结构上和概率上都是输入因子的正确代表。

#### 3.2.2 实验结果

实验结果如下所述：

- 在命令行运行`python autograder.py -q q2`对函数进行了测试。
- 测试结果显示所有测试用例均通过，包括：`1-product-rule.test`、`2-product-rule-extended.test`、`3-disjoint-right.test`、`4-common-right.test`、`5-grade-join.test`和`6-product-rule-nonsingleton-var.test`
- 每个测试的通过证明了`joinFactors`函数在处理不同类型的因子联合时的正确性和稳定性，能够有效地处理包含不同变量集合的因子，生成正确的联合因子。

### 3.3 任务三：消元（Eliminate）

#### 3.3.1 代码实现

在`eliminate`函数的实现中，我们的目标是从给定的因子中消除一个指定的非条件变量，并生成一个新的因子。以下是具体的实现步骤：

1. **变量检查与处理**：

   我们首先确认了要消除的变量（`eliminationVariable`）确实是因子中的一个非条件变量。如果不是，我们抛出了`ValueError`异常。如果因子中只有一个非条件变量，即消除变量是唯一的非条件变量，我们同样抛出了`ValueError`异常。

2. **变量集合更新**：

   我们遍历了因子中的非条件变量和条件变量集合，除去了要消除的变量，从而形成了新因子的非条件变量和条件变量集合。

3. **新因子创建**：

   我们使用更新后的非条件变量和条件变量集合，以及原因子的变量域字典，创建了一个新的因子`newFactor`。

4. **概率计算**：

   对于新因子的每个可能赋值，我们计算了所有包含消除变量的赋值情况下的概率总和。通过遍历消除变量的所有可能值，并对每个值计算原因子在该赋值下的概率，然后将这些概率相加，得到了在不考虑消除变量的情况下的概率总和。最后，我们将这个概率总和设置为新因子在对应赋值下的概率。

这个过程确保了从原因子中正确地消除了指定的非条件变量，并生成了一个在结构和概率上都与原因子一致的新因子。

#### 3.3.2 实验结果

实验结果如下所述：

- 在命令行运行 `python autograder.py -q q3` 对函数进行了测试。
- 所有测试用例均通过，包括：`1-simple-eliminate.test`、`2-simple-eliminate-extended.test`、`3-eliminate-conditioned.test`、`4-grade-eliminate.test`、`5-simple-eliminate-nonsingleton-var.test`和`6-simple-eliminate-int.test`
- 每个测试的通过表明了 `eliminate` 函数在处理不同类型的消元操作时的正确性和稳定性，能够有效地处理因子中的非条件变量的消除，并生成正确的新因子。

### 3.4 任务四：变量消元（Variable Elimination）

#### 3.4.1 代码实现

在 `inferenceByVariableElimination` 函数中，我们的目标是通过变量消元法执行概率推理查询，返回因子 $P(\text{queryVariables}|\text{evidenceDict})$。以下是具体的实现步骤：

1. **因子获取与消元准备**：

   首先，我们获取了所有与证据变量相关的条件概率表，即从贝叶斯网络中使用 `bayesNet.getAllCPTsWithEvidence(evidenceDict)` 方法提取因子。我们定义了一个消元顺序，如果没有给定顺序，则基于贝叶斯网络的变量集合和查询变量创建了一个。

2. **因子联合与消元**：

   对于消元顺序中的每个变量，我们首先调用 `joinFactorsByVariable` 方法，联合所有包含该变量的因子。然后，我们检查联合后的新因子中的非条件变量数量。如果大于1，则通过调用 `eliminate` 方法消除该变量，并将结果因子添加到因子列表中。如果新因子仅含一个非条件变量，则不执行消元，直接舍弃该因子。

3. **生成最终答案因子**：

   完成所有变量的消元后，我们通过调用 `joinFactors` 方法联合剩余的所有因子，生成一个包含所有查询变量和证据变量的单一因子。

4. **因子归一化**：

   最后，我们对最终生成的因子进行归一化处理，以确保概率总和为1，这通过调用 `normalize` 方法实现。

这个过程确保了按照给定的消元顺序逐步联合和消除因子，最终生成一个代表查询变量在给定证据下的条件概率分布的因子。

#### 3.4.2 实验结果

实验结果如下所述：

- 在命令行运行 `python autograder.py -q q4` 对函数进行了测试。
- 所有测试用例均通过，包括：`1-disconnected-eliminate.test`、`2-independent-eliminate.test`、`3-independent-eliminate-extended.test`、`4-common-effect-eliminate.test`、`5-grade-var-elim.test`和`6-large-bayesNet-elim.test`

- 每个测试的通过表明了 `inferenceByVariableElimination` 函数在处理不同类型的贝叶斯网络和变量消元操作时的正确性和稳定性。

## 3.5 任务五

### 3.5.1 DiscreteDistribution 类的实现

#### 3.5.1.1 代码实现

在实现 `DiscreteDistribution` 类时，我们的目标是构建一个模型，用于表示和处理离散键集上的信念分布和权重分布。

1. **归一化（Normalize）方法**：

   这个方法的目的是将分布归一化，使所有键的值总和为1，同时保持各键值之间的比例不变。实现中，我们首先检查了分布的总和是否为0。如果不为0，我们遍历了分布中的每个键，并将其值除以总和来进行归一化。这样确保了分布的总和为1，且各键值比例保持不变。

2. **采样（Sample）方法**：

   这个方法的目的是根据分布的值为每个键抽取一个随机样本。实现中，我们首先计算了分布的总和。如果总和为0，返回 `None`。如果总和不为0，我们使用了 `random.uniform` 方法生成一个0到总和之间的随机数。然后，我们遍历了分布中的每个键值对，累加值直到达到或超过这个随机数，返回对应的键。这种方法根据分布中的值为每个键生成加权随机样本。

### 3.5.2 观测概率（Observation Probability）

#### 3.5.2.1 代码实现

在 `getObservationProb` 函数中，我的目标是计算给定吃豆人位置和幽灵位置时，嘈杂距离观测值的概率 $P(\text{noisyDistance}|\text{pacmanPosition}, \text{ghostPosition})$。

1. **处理幽灵在监狱的情况**：

   当幽灵位置等于监狱位置时，如果观测到的嘈杂距离为 `None`（表示没有幽灵），概率返回 1，否则返回 0。这是因为幽灵在监狱时不应该有任何距离观测。

2. **处理幽灵不在监狱的情况**：

   当幽灵不在监狱时，如果观测到的嘈杂距离为 `None`，概率返回 0，因为幽灵不在监狱应该有距离观测。如果有嘈杂距离观测，我们使用 `busters.getObservationProbability` 方法来计算概率。这个方法根据嘈杂距离和吃豆人与幽灵之间的曼哈顿距离来确定观测概率。

这个过程确保了在不同情境下，能够正确地计算出嘈杂距离观测的概率，无论是幽灵在监狱还是在游戏场地内。

### 3.5.3 实验结果

实验结果如下所述：

- 在命令行运行 `python autograder.py -q q5` 对类的功能进行了测试。
- 所有测试用例均通过，包括：`1-DiscreteDist.test`、`1-DiscreteDist-a1.test` 和 `1-ObsProb.test`
- 每个测试的通过表明
  - `DiscreteDistribution` 类和在处理分布的归一化和采样操作时的正确性和效能。

– `getObservationProb` 函数在不同的幽灵位置和嘈杂距离观测情境下的正确性和效能。

## 3.6 任务6：精确推理：观测（Exact Inference Observation）

### 3.6.1 代码实现

在本部分，我们着重分析了`observeUpdate`函数，该函数的目的是根据观测值和Pacman的位置更新对鬼魂位置的置信度分布。

1. **获取当前状态信息**：

   通过`gameState.getPacmanPosition()`获取Pacman当前的位置，并通过`self.getJailPosition()`获取监狱位置（鬼魂被抓后的位置）。

2. **更新置信度分布**：

   遍历所有可能的鬼魂位置（包括监狱位置）。对于每一个位置，使用`self.getObservationProb`方法计算给定观测值、Pacman位置、鬼魂的假设位置和监狱位置下的观测概率。这个概率乘以之前的置信度，从而更新置信度分布。

3. **归一化处理**：

   由于更新后的置信度可能不再形成一个有效的概率分布，因此需要调用`self.beliefs.normalize()`对其进行归一化处理，确保所有位置的置信度总和为1。

### 3.6.2 实验结果

实验结果如下所述：

- 在命令行运行`python autograder.py -q q6`对类的功能进行了测试。
- 所有测试用例均通过，包括：`1-ExactUpdate.test`、`2-ExactUpdate.test`、`3-ExactUpdate.test`和`4-ExactUpdate.test`
- 每个测试的通过表明该函数在不同的测试用例中均未出现推理错误，该算法能够准确地根据Pacman的观测数据更新对鬼魂位置的置信度。

## 3.7 任务7：精确推理：时间推移 Exact Inference with Time Elapse）

### 3.7.1 代码实现

在任务7中，我们关注的是时间推移对于鬼魂位置置信度的影响。

1. **预测新的置信度分布**：

   首先，我们创建一个新的`DiscreteDistribution`对象`newBeliefs`，用于存储时间推移后的置信度分布。

2. **更新置信度分布**：

   对于`self.allPositions`中的每一个可能的鬼魂旧位置`oldPos`，我们首先使用`self.getPositionDistribution`方法获取在给定Pacman当前状态和鬼魂的旧位置下，鬼魂可能转移到的新位置分布`newPosDist`。然后，我们遍历这个分布中的每一个新位置和对应的转移概率`prob`，并将旧位置的信念值乘以转移概率累加到新位置的信念值上。

3. **替换旧的置信度分布**：

   在完成上述遍历和累加后，我们用`newBeliefs`替换原有的置信度分布`self.beliefs`，完成时间推移后置信度的更新。

#### 3.7.2 实验结果

实验结果如下所述：

- 在命令行运行`python autograder.py -q q7`对类的功能进行了测试。
- 所有测试用例均通过，包括：`1-ExactUpdate.test`、`2-ExactUpdate.test`、`3-ExactUpdate.test`和`4-ExactUpdate.test`
- 每个测试的通过表明该函数在不同的测试用例中均未出现推理错误，该算法能够准确地根据Pacman的观测数据更新对鬼魂位置的置信度。

### 3.8 任务8：精确推理：完整测试（Exact Inference Full Test）

#### 3.8.1 代码实现

在任务8中，我们关注的是使用精确推理来指导Pacman的行动。

1. **初始化变量**：获取Pacman当前位置、合法动作列表、存活的鬼魂及其位置分布。
2. **找到最近的鬼魂**：通过遍历存活鬼魂的位置分布，找到每个鬼魂最可能所在的位置。计算每个鬼魂与Pacman的距离，保留最近的鬼魂位置`closestGhostPosition`和对应距离`closestGhostDistance`。
3. **选择最佳行动**：对每个合法动作，计算执行该动作后Pacman的位置与最近鬼魂的距离。选择能使这个距离最小化的动作`closestAction`。

#### 3.8.2 实验结果

实验结果如下所述：

- 在命令行运行`python autograder.py -q q8`对类的功能进行了测试。
- 所有测试用例均通过，包括：`1-ExactFull.test`和`2-ExactFull.test`
- 在10次游戏中均获得胜利，平均得分763.3，实验结果表明，这种基于精确推理的行动选择策略在不同的测试用例中均没有出现推理错误，展示了策略的有效性和稳定性。

### 3.9 任务9：近似推理：初始化与置信度（Approximate Inference Initialization and Beliefs）

#### 3.9.1 代码实现

在任务9中，我们关注的是使用近似推理来初始化粒子，并将粒子集转换为置信度分布。具体的实现步骤如下：

1. **初始化粒子**：

   在`initializeUniformly`方法中，我们首先计算每个合法位置应有的粒子数（均匀分布）。然后，遍历每个合法位置，按照计算出的数量将粒子添加到`self.particles`列表中。如果粒子总数不能被合法位置数整除，剩余的粒子将均匀分布在部分位置上。

2. **构建置信度分布**：

   在`getBeliefDistribution`方法中，我们创建一个`DiscreteDistribution`对象`beliefs`。遍历所有粒子，对于每个粒子位置，在`beliefs`中相应位置的计数加一。最后，我们对`beliefs`进行归一化处理，确保其成为一个有效的概率分布。

#### 3.9.2 实验结果

实验结果如下所述：

- 在命令行运行`python autograder.py -q q9`对类的功能进行了测试。
- 所有测试用例均通过，包括：`1-ParticleInit.test`
- 实验结果显示，在粒子滤波器初始化测试中未出现推理错误，说明该方法能有效地初始化粒子并准确地将粒子集转换为置信度分布。

### 3.10 任务10：近似推理：观测（Approximate Inference Observation）

#### 3.10.1 代码实现

任务10聚焦于如何通过近似推理来更新基于观测的置信度。

1. **获取Pacman位置和监狱位置**：

   首先，我们需要知道Pacman当前的位置（`pacmanPosition`）和监狱位置（`jailPosition`），这是更新置信度的基础。

2. **构建更新后的置信度分布**：

   创建一个空的离散分布（`DiscreteDistribution`）对象`beliefs`。接着，遍历当前所有粒子（代表鬼魂可能的位置），根据观测数据、Pacman位置、粒子位置和监狱位置，使用`self.getObservationProb`方法计算每个粒子的权重，并累加到`beliefs`中。

3. **处理所有粒子权重为零的情况**：

   如果`beliefs`的总权重为零，意味着所有粒子均不可能，此时需要重新初始化粒子群，调用`self.initializeUniformly(gameState)`方法。

4. **归一化并重新抽样粒子**：

   如果不是所有粒子权重均为零，对`beliefs`进行归一化处理，确保其成为有效的概率分布。根据更新后的`beliefs`分布，对粒子群进行重新抽样，以匹配更新后的置信度分布。

#### 3.10.2 实验结果

实验结果如下所述：

- 在命令行运行`python autograder.py -q q10`对类的功能进行了测试。
- 所有测试用例均通过，包括：`1-ParticleUpdate.test`、`2-ParticleUpdate.test`、`3-ParticleUpdate.test`、`4-ParticleUpdate.test`和`5-ParticleUpdate.test`
- 根据实验结果，近似推理在观测更新测试中未出现推理错误。此外，该方法在10场游戏中均获胜，平均得分为180.2，显示了其在实际应用中的有效性和稳定性。

### 3.11 任务11：近似推理：时间推移（ Approximate Inference with Time Elapse）

#### 3.11.1 代码实现

任务11专注于如何通过近似推理处理时间的推移。具体实现步骤如下：

1. **粒子状态的采样**：

   创建一个新的粒子列表`new_particles`。遍历当前粒子列表`self.particles`。对于每个粒子（代表鬼魂的当前假设位置），使用`self.getPositionDistribution`方法获取鬼魂从当前位置到下一个可能位置的概率分布。对于每个粒子，根据这个概率分布采样出一个新位置，添加到`new_particles`中。

2. **更新粒子列表**：

   将`self.particles`更新为新采样的粒子列表`new_particles`。

#### 3.11.2 实验结果

实验结果如下所述：

- 在命令行运行`python autograder.py -q q11`对类的功能进行了测试。
- 所有测试用例均通过，包括：`1-ParticlePredict.test`、`2-ParticlePredict.test`、`3-ParticlePredict.test`、`4-ParticlePredict.test`和`5-ParticlePredict.test`
- 实验结果表明，近似推理在处理时间推移的测试中未出现推理错误。此外，该方法在5场游戏中均获胜，平均得分为382.8，显示了其在实际应用中的有效性和稳定性。

## 4 结论

本实验报告详细探讨了贝叶斯网络和概率推理在“捉鬼敢死队”项目中的应用，展示了理论和实践的紧密结合。报告从贝叶斯网络结构的构建开始，详细描述了网络中各变量的依赖关系和值域定义，确保网络能准确反映吃豆人游戏的动态环境。在实验内容方面，报告涵盖了多个关键任务，包括变量消元、精确和近似推理、置信度更新等。

每个任务的实现都体现了对概率推理细节的深刻理解。例如，变量消元过程中，报告展示了如何通过合并和消除因子来简化网络，以及如何最终生成一个包含所有查询变量和证据变量的单一因子。近似推理部分突出了在处理大规模或复杂网络时的实用性，通过粒子滤波器有效处理不确定性和动态信息。

实验结果部分展示了所实现方法的有效性和稳定性。通过自动评分系统的测试，报告证明了各种推理技术在模拟的吃豆人游戏环境中的准确性和可靠性。这些结果不仅验证了理论的实际应用价值，而且展示了概率推理在AI领域处理不确定性和动态环境时的重要性和效果。

总的来说，本报告通过“捉鬼敢死队”项目，成功地将概率推理和贝叶斯网络的理论知识应用于解决具体问题，不仅增强了对相关概念的理解，还提升了在动态和不确定环境下的问题解决能力。这些成果不仅对于学术研究有重要意义，也为将来在AI领域的实际应用打下了坚实的基础。