读书报告——各种各样的专家系统及其应用

什么是专家系统

专家系统 （Expert System） 是应用人工智能 （AI） 的一个分支，由 AI 社区在 1960 年代中期开发。ES 背后的基本思想很简单，专业知识，即大量特定于任务的知识，从人类转移到计算机。然后，这些知识被存储在计算机中，用户根据需要调用计算机寻求具体建议。计算机可以进行推理并得出特定的结论。然后，像人类顾问一样，它提供建议，并在必要时解释建议背后的逻辑（Turban&Aronson，2001）。ES 为各种问题提供了强大而灵活的方法，这些问题通常无法通过其他更传统和正统的方法处理。因此，它们的使用正在扩散到我们社会和技术生活的许多领域，在这些领域，它们的应用被证明在决策支持和解决问题的过程中至关重要。

借助知识工程和推理机制，旨在模拟和模仿人类领域专家在特定领域内的问题解决和决策能力。该系统通过收集、组织、表示和应用领域专家的经验性知识，以帮助用户在该领域内做出智能化的决策和解决问题。专家系统通常由以下组成部分构成：

1. **知识库：** 包含专业领域的知识，以规则、事实、案例等形式进行表示。这些知识是从领域专家那里获取的，用于解决特定类型的问题。
2. **推理引擎：** 负责对知识库中的信息进行推理和推断，以产生新的结论或解决问题。推理引擎使用事实和规则之间的逻辑关系来推导结论。
3. **用户界面：** 提供与用户进行交互的界面，通常包括问题提问、解释推理过程、展示结果等功能。用户通过界面向专家系统提供输入，并查看系统的输出结果。
4. **解释器：** 负责解释和解释专家系统的决策过程，向用户解释系统为何得出特定的结论或建议。

专家系统的目标是在特定领域中达到或超越人类专家水平的问题解决和决策能力。这种系统在医学诊断、工程设计、金融分析等多个领域得到应用，为用户提供有针对性的建议和解决方案。

一个专家系统必须具备三要素：

1. 领域专家级知识
2. 模拟专家思维
3. 达到专家级的水准

专家系统分类

根据知识表示方式（According to the working principle）：

1. 基于规则的专家系统（The Rule-Based Expert System）：

   基于规则的专家系统是最早出现的专家系统类型，也是目前研究人员构建专家系统最常用的方式。由生产规则完成的专家系统已经被整合到各个研究领域，帮助研究人员用预输入的知识解决各种问题。

基于规则的专家系统有很多优点。使用“条件-结果”语句来表达知识规则对人类来说是很自然的。同时，在基于规则的专家系统中，知识和推理是分开存储和处理的，符合人们的日常习惯。但也存在一些不足，阻碍了该专家系统的进一步发展。例如，当规则匹配条件时，必须严格按照数据库中的语句编写语句的表达式，即使由于匹配精度要求而拒绝了相对细微的差异。

此外，与其他类型的专家系统相比，基于规则的专家系统的速度并不占主导地位，因为在使用规则时需要扫描数据库中的整个规则集。

2. 基于框架的专家系统（ The Framework-Based Expert System ）：

基于框架的专家系统利用数据库中的框架来处理输入的具体问题，并通过推理引擎输出新的信息。一个框架代表一个“类”的概念，一个框架可能是另一个框架的“子类”。

子类框架继承其父类框架的所有属性，从而消除了重新输入进程属性的需要。但是，我们要注意一些特殊情况:一些子类和它们的父类可能在一个共同的属性上有所不同。当一个帧同时属于多个帧时，它也继承了所有这些帧的属性。

这种知识表示方法使得专家系统能够更自然地表达领域知识，并且可以更容易地进行推理和问题解决。基于框架的专家系统在某些应用中能够提供更灵活、可扩展的知识表示方式。

3. 基于模糊逻辑的专家系统（The Fuzzy Logic-Based Expert System）：

在基于模糊逻辑的专家系统中，模糊逻辑是专家系统推理的基础。这种推理方法以模糊规则为前提，利用模糊语言规则推导出近似的模糊判断结论。

在基于模糊逻辑的专家系统中，主要的组成部分包括：

模糊规则库+模糊推理引擎+模糊集+隶属函数

基于模糊逻辑的专家系统适用于处理那些由于不确定性或模糊性而难以精确建模的问题。例如，在温度控制、速度调节、语音识别等领域，模糊专家系统可以更好地处理模糊的输入和模糊的条件，提供更灵活的决策和控制。这种方法使得专家系统能够更好地模拟人类专家在处理模糊信息时的思维方式。

基于模糊逻辑的专家系统的优点是能够表达专家技能的高技能性，具有足够的鲁棒性，还可以进行启发式和试探性推理。然而，这种专家系统获取知识困难，其推理依赖于模糊知识库，学习能力不强，容易出错。

4. 基于神经网络的专家系统（The Expert System Based on Neural Network）：

神经网络模型与上面描述的逻辑系统有本质上的不同。在神经网络中，知识不再通过人工处理转化为显式规则，而是由学习算法自动获取并产生自己的隐式规则。与传统的专家系统相比，神经网络具有更强大的功能:比传统的串行运算效率更高;具有一定的容错能力;神经网络连接的权值可以改变等。

神经网络通过学习实例自动获取知识。专家提供示例和期望的解决方案，神经网络学习算法不断修改网络的权值分布，在训练后实现稳定的输出。

• 基于规则、框架的专家系统，虽然受到各自技术的限制，但由于其编码相对容易，这两类专家系统在处理相对简单的问题时仍然有用。模糊基于逻辑的专家系统能够反映现实中的模糊现象，模仿人类不精确推理的过程来完成其工作，在许多领域发挥着非常重要的作用。专家系统基于神经网络利用神经网络来进一步提高自己的能力，扩大获取知识的范围

根据推理机制：

- 正向推理系统： 从已知事实推导出结论。
- 逆向推理系统： 从目标目标出发，寻找符合条件的事实。
- 混合型推理系统： 结合多种推理机制，如正向、逆向、模糊推理等。

根据问题解决方式：

- 决策支持系统： 提供决策建议，用户仍负责最终决策。
- 问题诊断系统： 用于诊断问题和提供解决方案。
- 规划系统： 用于制定行动计划或设计方案。
- 监控和控制系统： 用于监测和控制系统的运行。

根据学习能力：

- 静态专家系统： 知识是固定的，无法根据经验进行更新。
- 动态专家系统：具有学习能力，可以根据新的信息进行知识更新。

根据应用领域：

- 医学专家系统： 用于医学诊断和治疗建议。
- 工程专家系统： 用于工程设计、控制和优化。
- 金融专家系统： 用于风险评估、投资建议等。
- 军事专家系统： 用于战术决策、情报分析等。
- 教育专家系统： 用于个性化学习建议和教学。

根据结构和复杂度：

- 单一专家系统： 由单个专家知识构建而成。
- 分布式专家系统： 多个专家系统相互协作，分担任务。

专家系统的应用

专家系统在各个领域都有广泛的应用，它们通过模拟人类专家的知识和推理能力，为解决复杂问题和提供决策支持提供了有效的工具。以下是一些专家系统的常见应用领域：

1. **医学诊断：** 在医学领域，专家系统用于辅助医生进行疾病诊断、制定治疗方案和提供药物推荐。这些系统可以利用大量的医学知识和病例数据，辅助医生做出准确的诊断。
2. **金融和投资：** 专家系统在金融领域被用于风险评估、投资建议、财务规划等方面。它们可以分析市场数据、制定交易策略，并提供个性化的财务建议。
3. **工业控制和生产优化：** 专家系统被应用于工业控制系统，用于监测和优化生产过程。它们可以提高生产效率、减少成本，并帮助解决生产中的技术问题。
4. **客户服务和支持：** 在客户服务领域，专家系统可以用于自动回答常见问题、提供产品支持、进行故障排除等。这有助于提高服务效率和用户满意度。
5. **教育和培训：** 专家系统被用于教育领域，支持在线学习和培训。它们可以提供个性化的学习路径、答疑解惑，帮助学生和职业人士获取专业知识。
6. **环境监测和管理：** 在环境科学中，专家系统可用于分析大气、水质、土壤等环境数据，预测自然灾害，制定环保政策。
7. **法律咨询：** 专家系统可以用于提供法律咨询和支持，帮助律师和法律专业人士分析案件、查找相关法规和判例，并提供法律建议。
8. **交通和物流：** 在交通管理和物流领域，专家系统可以用于交通流优化、路径规划、货物配送等，以提高交通效率和减少运输成本。

具体的诊断专家系统示例

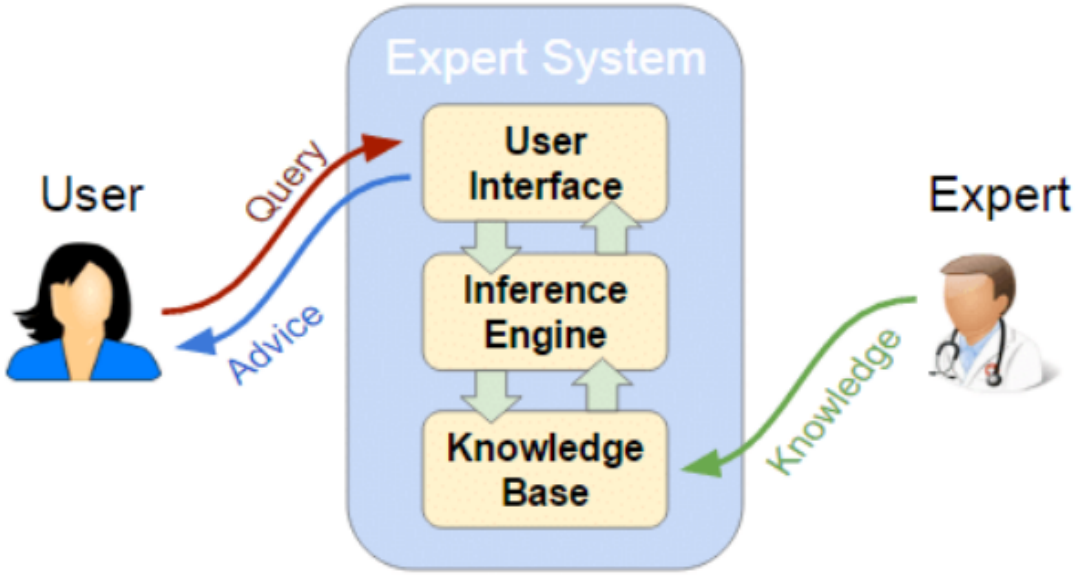
诊断抑郁症的专家系统（An Expert System for Depression Diagnosis）

目标:该专家系统的主要目标是对疾病进行正确的诊断和正确的治疗，并通过与疾病有关的几个提示给出适当的治疗方法，我们将通过专家系统的应用来看到这一点。

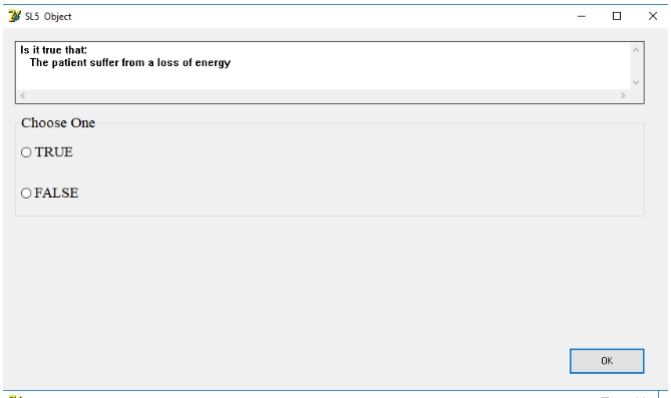
方法:在本文中提出的专家系统的设计是为了帮助心理学家诊断抑郁症的症状，如:精力不足，食欲改变，睡眠多或少，焦虑，注意力不集中，优柔寡断，不安，无价值感，内疚或绝望，自残或自杀的想法。提出的专家系统介绍了抑郁症的概况，概述了疾病的原因，并尽可能给出了疾病的治疗方法。

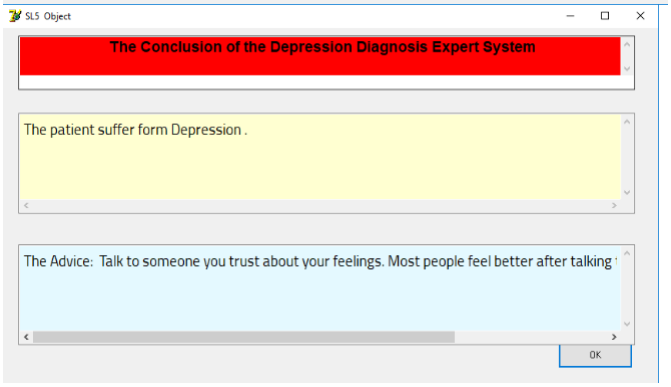
结果:心理学专业学生对所构建的抑郁症诊断专家系统进行了评价，并对系统的性能表示满意。结论:所建立的专家系统对心理学家、抑郁症患者和刚毕业的心理学家具有一定的实用价值。

抑郁症的诊断非常复杂，因为它有许多症状，并可能影响人体健康。所以，他们需要对抑郁症有丰富经验的心理学家。基于上述所有原因，我们开发了这个专家系统来帮助心理学家诊断抑郁症，以便开出适当的治疗处方。专家系统是人工智能(AI)的计算机应用，它包含一个知识库和一个推理引擎，其主要组成部分和细节如图。



提出的专家系统对人类生命各个阶段的抑郁症进行诊断，从简单的症状开始，通过问“是”或“否”的问题。建议的专家系统将要求用户在每个屏幕中选择正确的答案。在对话结束时，建议的专家系统向用户提供疾病的诊断和建议。如图:





该专家系统的主要知识来源是心理学家和抑郁症专业网站。捕获的知识被转换成SL5对象知识库语法(事实、规则和对象)。目前专家系统有9条规则涵盖抑郁症。

目前提出的专家系统专门用于诊断具有以下症状的抑郁症:精力不足，食欲改变，睡眠或多或少，焦虑，注意力不集中，优柔寡断，不安，无价值感，内疚或绝望以及自残或自杀的想法。

这个专家系统不需要密集的训练就可以使用;它易于使用，具有用户友好的界面。

参考文献

《Expert system methodologies and applications—a decade review from 1995 to 2004》Shu-Hsien Liao

Wikipedia——Expert System

《A brief history and technical review of the expert system research》Hao-Cheng Tan

《An Expert System for Depression Diagnosis》Izzeddin A. Alshawwa, Mohammed Elkahlout, Hosni Qasim El-Mashharawi, Samy S. Abu-Naser

实验1——重排九宫

问题描述

给定一个 3x3 的方格，其中包含数字 1 到 8 和一个空格（用 0 表示），目标是通过交换相邻的数字，将方格中的数字重新排列成一个特定的目标状态。每一步可以将空格与相邻的数字交换，这样就能够逐步达到目标状态。

算法描述

总体概述

以下是给定程序的简要算法描述：

1. **数据结构定义：**
   - 定义了一个结构体 `nineSquare` 用于表示九宫格状态。
   - 使用两个数组 `open` 和 `close` 分别表示开放列表和关闭列表。
   - 使用变量 `otop`、`oinr`、`obot` 表示开放列表的顶部、当前内部指针和底部。
   - 使用变量 `ctop` 表示关闭列表的顶部。
2. **函数定义：**
   - `over()`：检查是否发生溢出。
   - `disp(a[3][3])`：打印九宫格状态。
   - `check(a[3][3])`：检查是否达到目标状态。
   - `find_zero(a[3][3])`：找到空格的位置。
   - `exchange(i, j, a[3][3])`：交换两个位置的值。
   - `bfs(a)`、`dfs(a)`、`astar(a)`：分别为广度优先搜索、深度优先搜索和A*搜索算法。
   - `cmp(a[3][3], b[3][3])`：比较两个九宫格状态是否相同。
   - `cost(a)`：计算九宫格状态的代价函数。
   - `path()`：打印搜索路径。
   - `index(a[], at, b[], bt)`：在A*算法中找到代价最小的节点的索引。
3. **主函数：**
   - 初始化九宫格的初始状态 `a`。
   - 检查是否已经达到目标状态，若是，则输出不需要操作。
   - 选择算法类型（BFS、DFS、A*）。
   - 使用相应的算法进行搜索，并记录搜索路径。
   - 打印搜索路径。
4. **搜索算法：**
   - **BFS（广度优先搜索）：** 使用循环不断扩展状态，直到达到目标状态。
   - **DFS（深度优先搜索）：** 使用递归深度优先搜索，遇到回溯时回退。
   - **A*搜索：** 使用代价函数计算节点代价，选择代价最小的节点进行扩展。
5. **搜索路径显示：**
   - 使用 `path()` 函数回溯并显示搜索路径。

这个程序主要是针对八数码问题进行搜索，其中实现了广度优先搜索、深度优先搜索和A*搜索三种不同的搜索算法。算法的选择由用户在程序运行时输入。

BFS

- `bfs` 函数用于进行广度优先搜索。
- 通过找到空格的位置，对其相邻的四个方向进行扩展，即左、上、下、右。
- 对每一次扩展，将新的状态存入 `open` 列表尾部，并检查是否溢出。如果溢出，说明队列已满，退出搜索。

DFS

- `dfs` 函数用于进行深度优先搜索。
- 通过找到空格的位置，对其相邻的四个方向进行扩展，即左、上、下、右。
- 对每一次扩展，将新的状态存入 `open` 列表，并检查是否溢出。如果溢出，说明栈已满，退出搜索。
- 使用 `a.dp` 记录深度，每次扩展后递增或递减，用于控制深度。

A*

- `astar` 函数用于进行A*搜索。
- 通过找到空格的位置，对其相邻的四个方向进行扩展，即左、上、下、右。
- 对每一次扩展，将新的状态存入 `open` 列表，并检查是否溢出。如果溢出，说明队列已满，退出搜索。
- 使用 `a.dp` 记录深度，每次扩展后递增或递减，用于控制深度。
- 在 A* 中，对每个节点计算代价函数 `cost(a)`，选择代价最小的节点进行扩展。其中 `cost(a)` 包括节点到目标状态的代价和节点深度。通过 `index` 函数找到代价最小的节点。

实验结果

BFS

```
select a algorithm:
bfs
26 extended
2 8 3
1   4
7 6 5
  ↓
2   3
1 8 4
7 6 5
  ↓
  2 3
1 8 4
7 6 5
  ↓
1 2 3
  8 4
7 6 5
  ↓
1 2 3
8   4
7 6 5
```

DFS

```
select a algorithm:
dfs
46 extended
2 8 3
1   4
7 6 5
  ↓
2   3
1 8 4
7 6 5
  ↓
  2 3
1 8 4
7 6 5
  ↓
1 2 3
  8 4
7 6 5
  ↓
1 2 3
8   4
7 6 5
```

A*

```
select a algorithm:
astar
5 extended
2 8 3
1   4
7 6 5
  ↓
2   3
1 8 4
7 6 5
  ↓
  2 3
1 8 4
7 6 5
  ↓
1 2 3
  8 4
7 6 5
  ↓
1 2 3
8   4
7 6 5
```

实验结果分析

从以上的结果截图来看，三种算法得到的路径都相同。

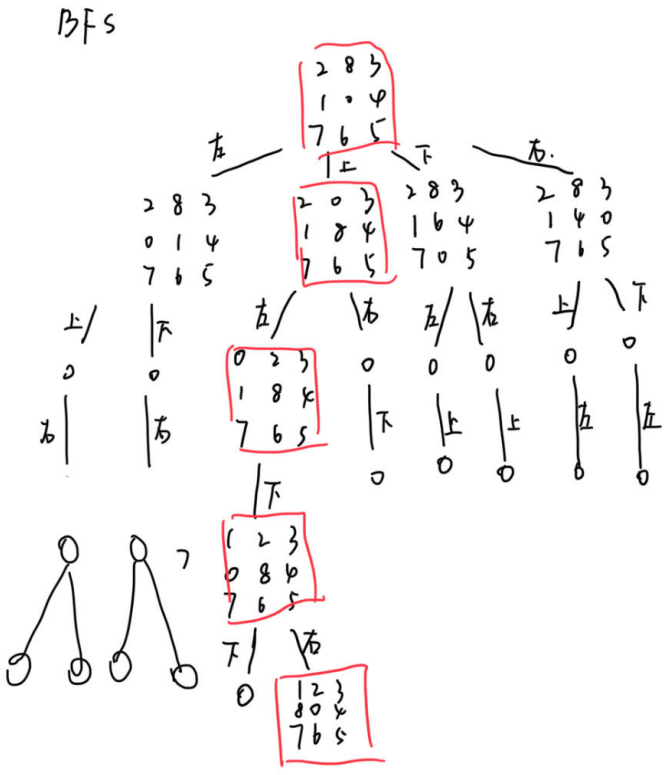
尽管路径相同，但是我们可以知道他们检索的路径并不相同。

BFS算法总共扩展的节点数为26个

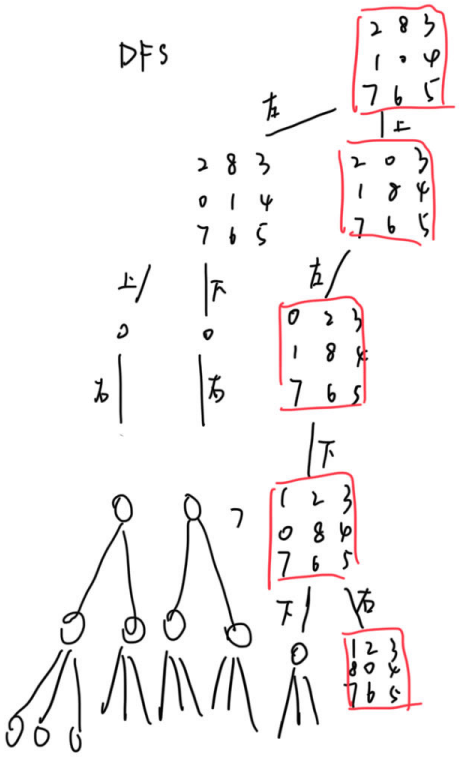
DFS算法总共扩展的节点数为46个 *有界DFS深度为5*

A*算法总共扩展的节点数为5个

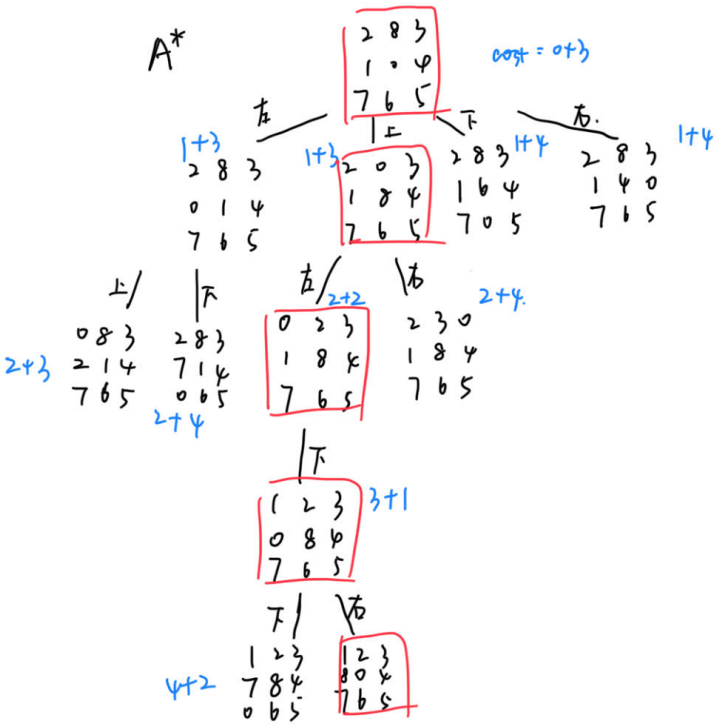
BFS



DFS



A*



不同算法的性能对比

1.**差异**:

广度优先搜索是将结点 n 的子结点放入到 OPEN 表的尾部 （类似队列）；

深度优先搜索是把结点 n 的子结点放入到 OPEN 表的首部（类似栈）；

A* 算法则是将结点 n 的子结点放入到 OPEN 表后，再将 OPEN 表中的结点按照估价函数进行排序（类似优先队列）。

2.**性能**:

BFS算法总共扩展的节点数为26个

DFS算法总共扩展的节点数为46个 *有界DFS深度为5*

A*算法总共扩展的节点数为5个

耗时在如此短的算法中都非常小。

可以看到A*算法的拓展节点是最少的，但是需要花费额外的时间找出代价最小的节点进行拓展；

BFS算法扩展的节点数适中，每次拓展open表头就可，不需要花费额外的时间寻找节点；

而DFS有界的搜索过程和限制的界深度有直接关系，深度为5时拓展的节点是46个，深度为4时拓展就为26个；每次拓展open表头，拓展节点放入表头，也不需要额外时间寻找合适的节点。当然没有界深度限制就有极大可能找不到解。

总而言之：A*算法的各方面都比较优秀，在许多情况下都是一个优秀的选择，它结合了广度优先搜索和启发式函数，具有较好的搜索速度和空间效率。

源代码

```
#include <iostream>
#include <cstring>
#include <chrono>

using namespace std;

typedef struct nineSquare{
    int nine[3][3];
    int dp;
    struct nineSquare *n;
}nS;

nS open[1000000];
nS close[1000000];
int otop=1;
int oinr=1;
int obot=0;
int ctop=0;
//check overflow
int over(){
    if(otop==999999) {
        cout<<"overflow"<<endl;
        return 1;
    }
    return 0;
}
```

```
//print the result
void disp(int a[3][3]){
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            if(a[i][j]!=0) cout<<a[i][j]<<" ";
            else cout<<"  ";
        }
        cout<<endl;
    }
}
//check the array
int check(int a[3][3]){
    if(a[0][0]!=1||a[0][1]!=2||a[0][2]!=3) return 1;
    if(a[1][0]!=8||a[1][1]!=0||a[1][2]!=4) return 1;
    if(a[2][0]!=7||a[2][1]!=6||a[2][2]!=5) return 1;
    return 0;
}
//find the location of 0
int find_zero(int a[3][3]){
    int lo=0;
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            if(a[i][j]==0) return lo;
            lo++;
        }
    }
    return lo;
}
//swap
void exchange(int i,int j,int a[3][3]){
    int t=a[i/3][i%3];
    a[i/3][i%3]=a[j/3][j%3];
    a[j/3][j%3]=t;
}
//BFS
void bfs(nS a){
    int zero=find_zero(a.nine);
    if((zero%3)!=0){//left ext
        exchange(zero-1,zero,a.nine);
        open[otop++]=a;
        if(over()) return ;
        exchange(zero-1,zero,a.nine);
    }
    if(zero>=3){//up ext
        exchange(zero-3,zero,a.nine);
        open[otop++]=a;
        if(over()) return ;
        exchange(zero-3,zero,a.nine);
    }
    if(zero<=5){//down ext
        exchange(zero+3,zero,a.nine);
        open[otop++]=a;
        if(over()) return ;
        exchange(zero+3,zero,a.nine);
    }
```

```
    if((zero+1)%3 != 0){//right ext
        exchange(zero+1,zero,a.nine);
        open[otop++]=a;
        if(over()) return ;
        exchange(zero+1,zero,a.nine);
    }
}
//DFS
void dfs(nS a){
    int zero=find_zero(a.nine);
    if((zero%3)!=0){//left ext
        exchange(zero-1,zero,a.nine);
        a.dp++;
        open[otop++]=a;
        a.dp--;
        if(over()) return ;
        exchange(zero-1,zero,a.nine);
    }
    if(zero>=3){//up ext
        exchange(zero-3,zero,a.nine);
        a.dp++;
        open[otop++]=a;
        a.dp--;
        if(over()) return ;
        exchange(zero-3,zero,a.nine);
    }
    if(zero<=5){//down ext
        exchange(zero+3,zero,a.nine);
        a.dp++;
        open[otop++]=a;
        a.dp--;
        if(over()) return ;
        exchange(zero+3,zero,a.nine);
    }
    if((zero+1)%3 != 0){//right ext
        exchange(zero+1,zero,a.nine);
        a.dp++;
        open[otop++]=a;
        a.dp--;
        if(over()) return ;
        exchange(zero+1,zero,a.nine);
    }
}
//Astar
void astar(nS a){
    int zero=find_zero(a.nine);
    if((zero%3)!=0){//left ext
        exchange(zero-1,zero,a.nine);
        a.dp++;
        open[otop++]=a;
        a.dp--;
        if(over()) return ;
        exchange(zero-1,zero,a.nine);
    }
    if(zero>=3){//up ext
```

```
        exchange(zero-3,zero,a.nine);
        a.dp++;
        open[otop++]=a;
        a.dp--;
        if(over()) return ;
        exchange(zero-3,zero,a.nine);
    }
    if(zero<=5){//down ext
        exchange(zero+3,zero,a.nine);
        a.dp++;
        open[otop++]=a;
        a.dp--;
        if(over()) return ;
        exchange(zero+3,zero,a.nine);
    }
    if((zero+1)%3 != 0){//right ext
        exchange(zero+1,zero,a.nine);
        a.dp++;
        open[otop++]=a;
        a.dp--;
        if(over()) return ;
        exchange(zero+1,zero,a.nine);
    }
}
//same
int cmp(int a[3][3],int b[3][3]){
    for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
            if(a[i][j]!=b[i][j]) return 1;
        }
    }
    return 0;
}
//cost
int cost(nS a){
    int num=0;
    if(a.nine[0][0]!=1&&a.nine[0][0]!=0) num++;
    if(a.nine[0][1]!=2&&a.nine[0][1]!=0) num++;
    if(a.nine[0][2]!=3&&a.nine[0][2]!=0) num++;
    if(a.nine[1][0]!=8&&a.nine[1][0]!=0) num++;
    if(a.nine[1][1]!=0) num++;
    if(a.nine[1][2]!=4&&a.nine[1][2]!=0) num++;
    if(a.nine[2][0]!=7&&a.nine[2][0]!=0) num++;
    if(a.nine[2][1]!=6&&a.nine[2][1]!=0) num++;
    if(a.nine[2][2]!=5&&a.nine[2][2]!=0) num++;

    return num+a.dp;
}
//show path
void path(){
    nS *b=&open[obot];
    nS *c[1000];
    int i=0;
    while(1){
        c[i++]=b;
```

```
        if(b->n==NULL) break;
        b=b->n;
    }
    for(int j=i-1;j>0;j--){
        disp(c[j]->nine);
        cout<<"  ↓   "<<endl;
    }
    disp(c[0]->nine);
}
//astar find the minist
int index(nS a[],int at,nS b[],int bt){
    int flag[10000]={0};
    for(int i=0;i<at;i++){
        for(int j=0;j<bt;j++){
            if(cmp(a[i].nine,b[j].nine)==0){
                flag[i]=1;
                break;
            }
        }
    }
    int min=100;
    int index=0;
    for(int i=0;i<at;i++){
        if(flag[i]==0){
            if(cost(a[i])<min){
                min=cost(a[i]);
                index=i;
            }
        }
    }
    return index;
}

int main(){

    nS a;
    a.n=NULL;
    a.dp=0;

    a.nine[0][0]=2;
    a.nine[0][1]=8;
    a.nine[0][2]=3;
    a.nine[1][0]=1;
    a.nine[1][1]=0;
    a.nine[1][2]=4;
    a.nine[2][0]=7;
    a.nine[2][1]=6;
    a.nine[2][2]=5;

    if(check(a.nine)==0){
        cout<<"your input need no operation"<<endl;
        return 0;
    }

    //cout<<cost(a)<<endl;;
```

```
cout<<"select a algorithm: "<<endl;
char al[3][10]={"bfs","dfs","astar"};
char p[10];
cin>>p;
int i=0;
for(i=0;i<3;i++){
    if(strcmp(p,al[i])==0) break;
}


open[0]=a;
open[0].dp=0;
open[0].n=NULL;

auto start_time = chrono::high_resolution_clock::now();

if(i==0){//bfs
    while(check(open[obot].nine)==1){
        int flag=0;
        for(int i=0;i<ctop;i++){
            if(cmp(open[obot].nine,close[i].nine)==0){
                obot++;
                flag=1;
                break;
            }
        }

        if(flag==1) continue;

        close[ctop]=open[obot];
        obot++;
        bfs(close[ctop]);
        if(over()) return 1;
        for(int i=oinr;i<otop;i++){
            open[i].n=&close[ctop];
        }
        ctop++;
        oinr=otop;
    }
}
else if(i==1){//dfs
    while(check(open[obot].nine)==1&&otop!=ctop){
        int flag=0;
        for(int i=0;i<ctop;i++){
            if(cmp(open[obot].nine,close[i].nine)==0){
                obot--;
                flag=1;
                break;
            }
        }
        if(flag==1) continue;

        if(open[obot].dp==5){
            close[ctop++]=open[obot--];
```

```
                flag=1;
            }
            if(flag==1) continue;

            close[ctop]=open[obot];
            obot--;
            dfs(close[ctop]);
            if(over()) return 1;
            for(int i=oinr;i<otop;i++){
                open[i].n=&close[ctop];
            }
            ctop++;
            oinr=otop;
            obot=otop-1;
        }
    }
    else{//astar
        while(check(open[obot].nine)==1&&otop!=ctop){

            close[ctop]=open[obot];

            astar(close[ctop]);

            if(over()) return 1;
            for(int i=oinr;i<otop;i++){
                open[i].n=&close[ctop];
            }

            ctop++;
            oinr=otop;

            obot=index(open,otop,close,ctop);

        }
    }
    cout<<ctop<<" extended"<<endl;
    auto end_time = chrono::high_resolution_clock::now();
    auto duration = chrono::duration_cast<chrono::milliseconds>(end_time -
start_time);
    cout << "Algorithm executed in " << duration.count() << " milliseconds." <<
endl;
    path();

    return 0;
}
```

实验2——主观Bayes

问题描述

主观Bayes方法推理的任务就是根据证据E的概率P(E)及LS、LN的值，把H的先验概率P(H)更新为后验概率P(H|E)或P(H|¬E)。

1. `pH`： 先验概率，即在考虑任何新证据之前，我们对事件 H 的初始信仰。在你的代码中，它表示在没有考虑条件概率的情况下，事件 H 发生的先验概率。
2. `pE`： 类似于 `pH`，它是事件 E 的先验概率，即在考虑任何新证据之前，我们对事件 E 的初始信仰。
3. `LS`： 条件概率中的似然性，表示在给定事件 S 的情况下，事件 E 和事件 H 之间的关联程度。它影响了在有新证据的情况下如何更新我们对事件 H 的信仰。
4. `LN`： 类似于 `LS`，它表示在给定事件非S的情况下，事件 E 和事件 H 之间的关联程度。它也是用于更新信仰的参数。

算法描述

1. **先验概率：**
   - `pH` 和 `pE` 分别表示事件 H 和事件 E 的先验概率。它们代表在考虑任何新证据之前，我们对事件 H 和事件 E 的初始信仰水平。
2. **条件概率似然性：**
   - `LS` 表示在给定事件 S 的情况下，事件 E 和事件 H 之间的关联程度（似然性）。
   - `LN` 表示在给定事件非S的情况下，事件 E 和事件 H 之间的关联程度（似然性）。
3. **贝叶斯公式的实现：**

$$P(H \mid E) = \frac{LS \times P(H)}{(LS-1)P(H)+1}$$

$$P(H \mid \neg E) = \frac{LN \times P(H)}{(LN-1) \times P(H)+1}$$

$$P(H \mid S) = \begin{cases} P(H \mid \neg E) + \dfrac{P(H)-P(H \mid \neg E)}{P(E)} \times P(E \mid S), & 0 \le P(E \mid S) < P(E) \\ P(H) + \dfrac{P(H \mid E)-P(H)}{1-P(E)} \times [P(E \mid S)-P(E)], & P(E) \le P(E \mid S) \le 1 \end{cases}$$

   - `EH` 函数实现了贝叶斯公式的一部分。根据贝叶斯公式，后验概率 P(H|E) 可以通过先验概率 P(H)、先验概率 P(E) 以及条件概率似然性 P(E|H) 和 P(E|~H) 的组合计算得到。
   - 具体来说，根据输入的 P(E|S)（在代码中为 `pE_S`），函数计算并返回 P(H|S)。
4. **绘制概率图：**
   - 使用 Matplotlib 绘制概率图，横轴表示条件概率 P(E|S)，纵轴表示相应的条件概率 P(H|S)。

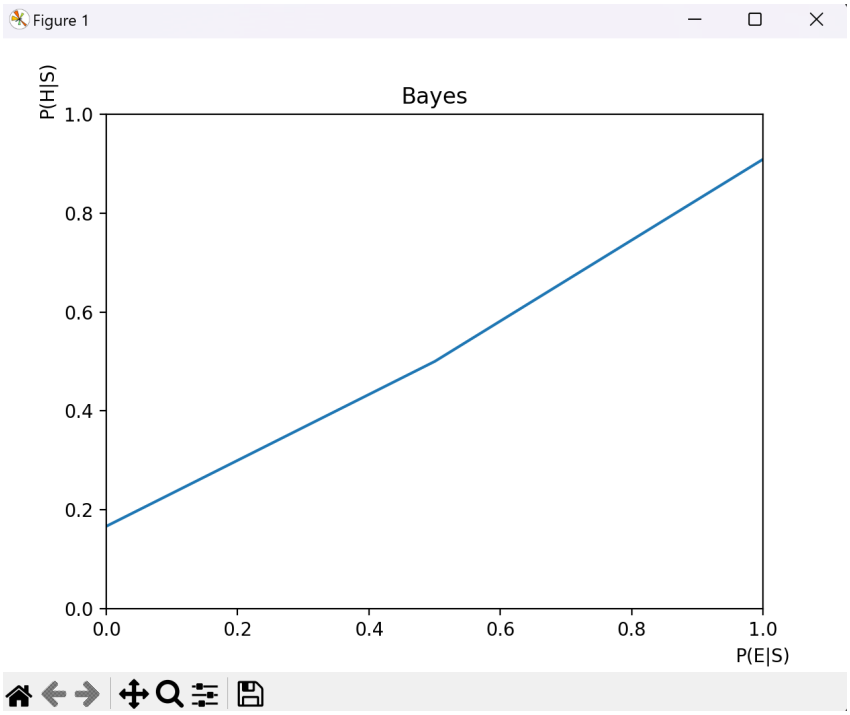
- `func = np.vectorize(EH)` 向量化了 `EH` 函数，使其能够处理整个数组而不仅仅是单个值。
- 通过生成一系列横轴值 `x`，计算相应的纵轴值 `y`，并绘制曲线。

5. **参数**

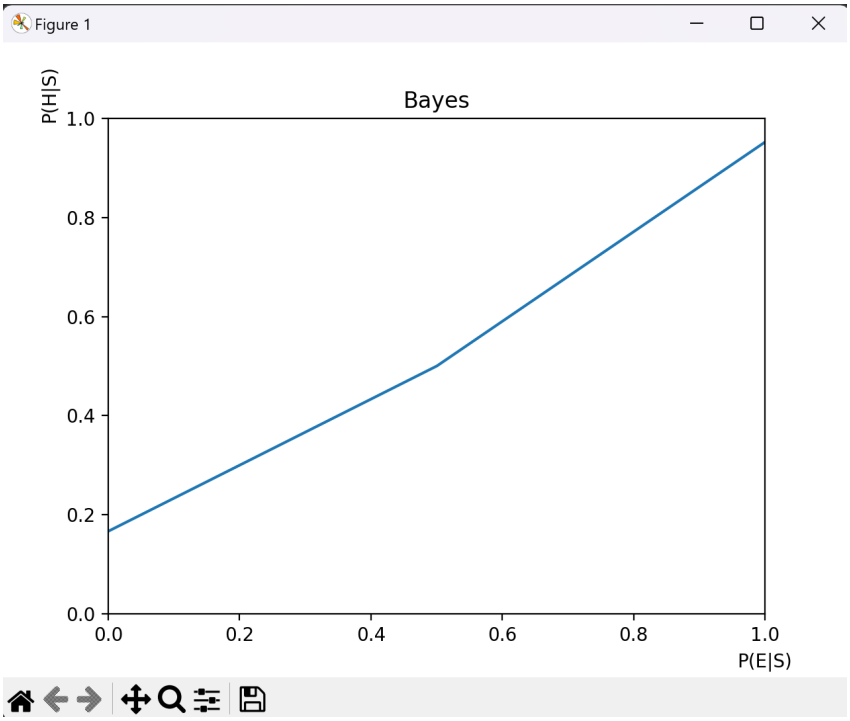
- `pH` 和 `pE`：先验概率，表示在考虑任何新证据之前，我们对事件 H 和事件 E 的初始信仰水平。
- `LS` 和 `LN`：条件概率似然性，表示在给定事件 S 或非S 的情况下，事件 E 和事件 H 之间的关联程度。

实验结果

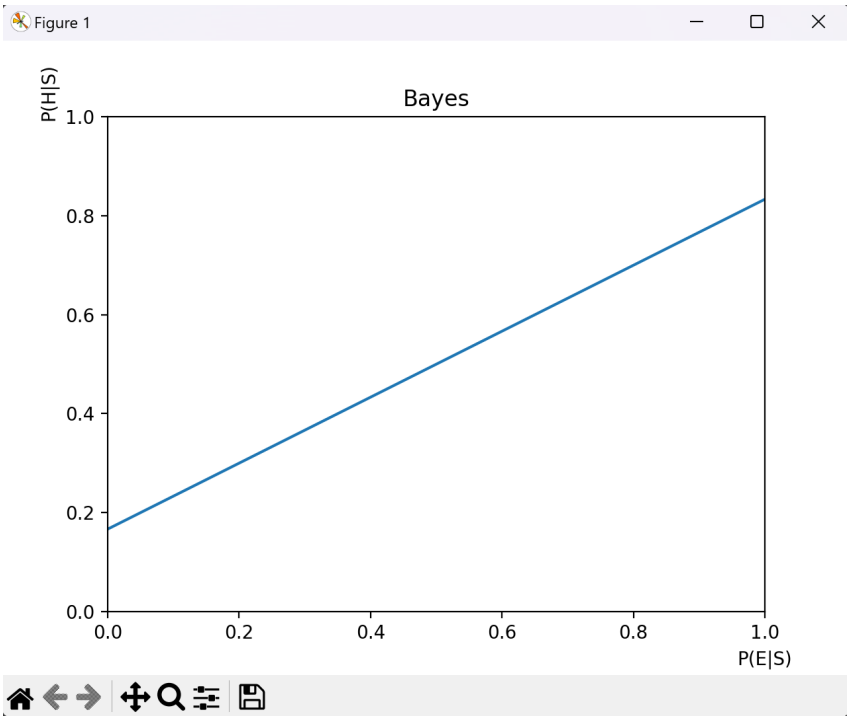
1. **默认参数组合：**
   - `pH = 0.5`（先验概率）
   - `pE = 0.5`（先验概率）
   - `LS = 10.0`（条件概率似然性，S的情况）
   - `LN = 0.2`（条件概率似然性，非S的情况）



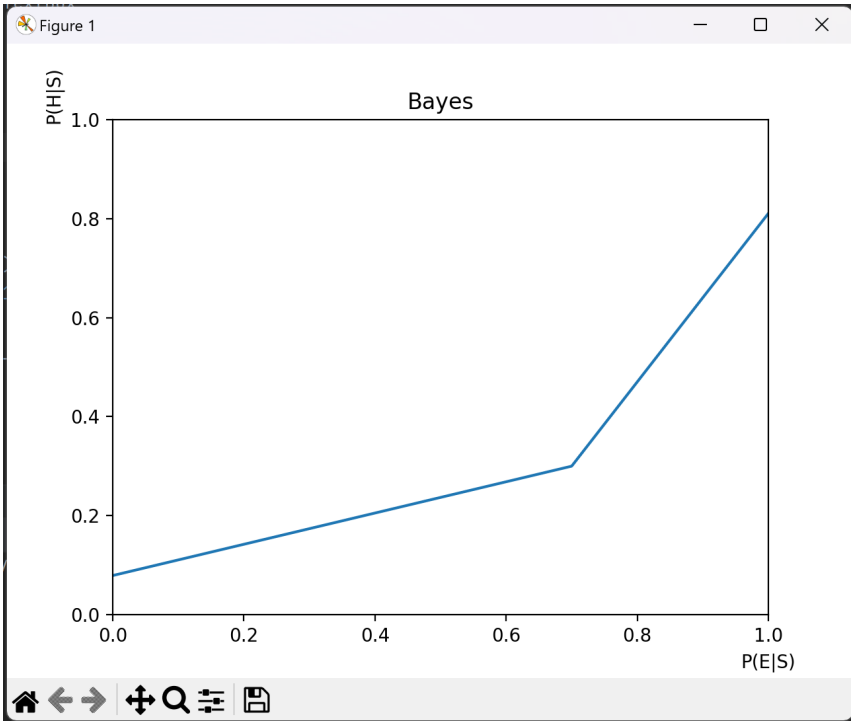
2. **更强烈的似然性：**
   - 增大 `LS`，观察曲线变化。例如，将 `LS` 设为 20.0 。



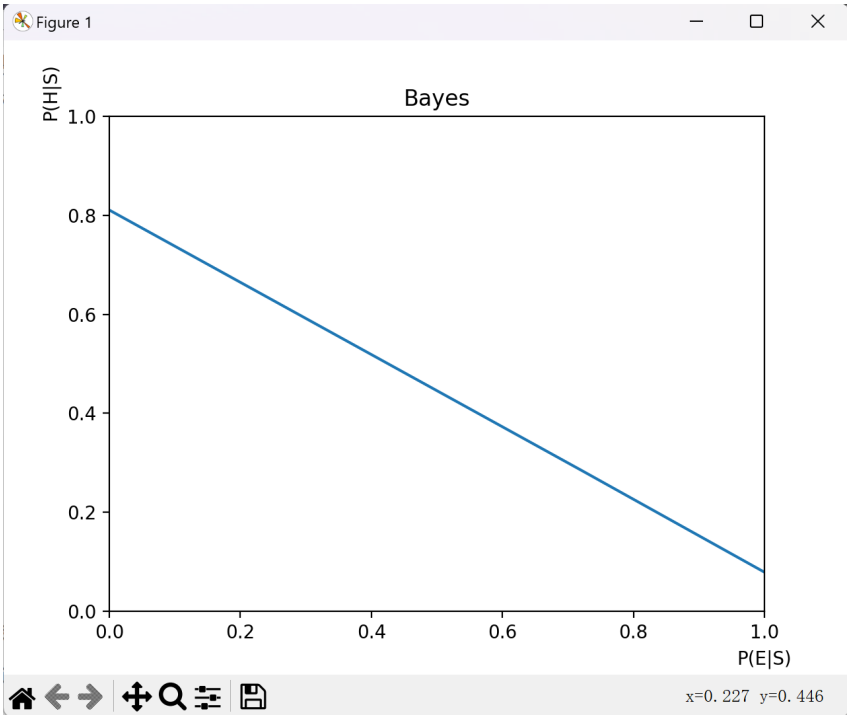
3. **更弱的似然性：**
   - 尝试减小 `LS`，观察曲线变化。例如，将 `LS` 设为 5.0 。



4. **不同的先验概率：**
   - 尝试修改 `pH` 和 `pE`，观察曲线如何受到先验概率的影响。例如，将 `pH` 设为 0.3 或 `pE` 设为 0.7。



5. **不同的条件概率似然性（LN）：**
   - 尝试修改 `LN`。例如，将 `LN` 设为 大于1 或更大。



实验结果分析

1. `pH` **和** `pE` **的影响：**
   - 增大 `pH` 或 `pE` 会增加相应的先验概率，可能使 P(E|S) 和 P(H|S) 的曲线整体上升。
   - 减小 `pH` 或 `pE` 会减少相应的先验概率，可能使 P(E|S) 和 P(H|S) 的曲线整体下降。
2. `LS` **和** `LN` **的影响：**
   - 增大 `LS` 会增强在事件 S 的情况下，P(E|S) 和 P(H|S) 之间的关联程度。这可能导致曲线在 S 处更陡峭。
   - 减小 `LS` 会减弱这种关联性，使曲线在 S 处变得更平缓。
   - 增大 `LN` 会增强在事件非S的情况下，P(E|S) 和 P(H|S) 之间的关联程度。这可能导致曲线在非S处更陡峭。

- 减小 `LN` 会减弱这种关联性，使曲线在非S处变得更平缓。

总体而言，这些调整可能导致概率曲线在条件概率的图上出现以下变化：

- **上移或下移：** 全局的上移或下移表示整体的先验概率发生变化。
- **曲线的陡峭程度：** 曲线在 S 或非S 处的陡峭程度可能会随着 `LS` 和 `LN` 的变化而改变。

源代码

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import TextBox

pH = 0.5
pE = 0.5
LN = 0.2
LS = 10.0


def EH(pE_S):
    pH_E = (LS * pH) / ((LS - 1) * pH + 1)
    pH_nE = (LN * pH) / ((LN - 1) * pH + 1)
    if 0 <= pE_S < pE:
        pH_S = pH_nE + (pH - pH_nE) / pE * pE_S
    elif pE <= pE_S <= 1:
        pH_S = pH + (pH_E - pH) / (1 - pE) * (pE_S - pE)
    return pH_S


func = np.vectorize(EH)
x = np.linspace(0, 1, 1000)
y = func(x)
fig, ax = plt.subplots()
ax.set_title('Bayes')
l, = plt.plot(x, y)
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('P(E|S)', x=1)
plt.ylabel('P(H|S)', y=1.05, rotation=90)
plt.subplots_adjust(bottom=0.1)
plt.show()
```