湖南大學

HUNAN UNIVERSITY

人工智能实验二报告
约束满足问题

目录

一、实验目的

1. 求解约束满足问题；
2. 使用回溯搜索算法求解八皇后问题。

二、实验描述（算法原理）

回溯搜索算法是一种通过试错来解决问题的算法，它尝试分步解决一个问题，如果在某一步发现之前的选择不能得到有效的解决方案，就回退一步，撤销之前的选择，再尝试其他的选择。这种算法常用于解决组合问题、优化问题和约束满足问题（CSP）。

基本思想：

1.选择（Choice）：从当前状态出发，选择一个可能的候选解，并进入下一个状态。

2.约束（Constrain）：在每一步选择后，检查是否满足问题的约束条件。如果满足，继续进行；如果不满足，进行回溯。

3.目标（Goal）：检查当前状态是否达到了目标状态。如果是，记录这个解；如果不是，继续搜索。

4.回溯（Backtrack）：如果当前选择不能导致有效解，撤销这一选择，回退到上一个状态，尝试其他可能的选择。

数据集处理：

使用八皇后问题的书中描述：

书中第三章：

八皇后问题的目标是在国际象棋棋盘中放置 8 个皇后，使得任何一个皇后都不会攻击到其他任一皇后。（皇后可以攻击和它在同一行、同一列或者同一对角线的任何棋子。）图 3.5 给出了失败的尝试：最右下角的皇后与最左上角的皇后可能互相攻击。

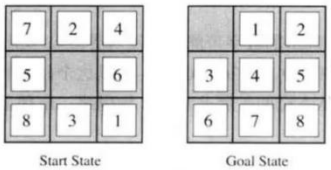


图 3.4 八数码问题

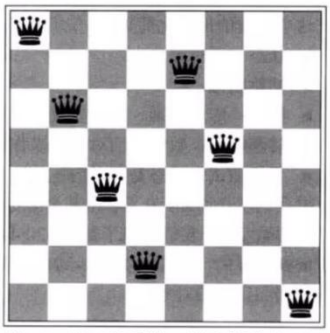


图 3.5 八皇后问题的一种近乎是解的局面（真正的解留作练习）

尽管求解 n 皇后问题存在一些有效的专用算法，但对于搜索算法而言此类问题仍然是

有用的测试用例。这类问题的形式化主要分为两类。**增量形式化**（incremental formulation）包括了算符来增加状态描述，从空状态开始；对于八皇后问题，即每次行动添加一个皇后到状态中去。另一类是**完整状态形式化**（complete-state formulation），8 个皇后都在棋盘上并且不断移动。无论哪种情况，都无需考虑路径消耗，只需考虑最终状态。增量形式化可以如下考虑：

- **状态**：棋盘上 0 到 8 个皇后的任一摆放都是一个状态。
- **初始状态**：棋盘上没有皇后。
- **行动**：在任一空格增加摆放 1 个皇后。
- **转移模型**：将增加了皇后的棋盘返回。
- **目标测试**：8 个皇后都在棋盘上，并且无法互相攻击。

这种形式化我们需要考查 $64 \times 63 \times \cdots \times 57 \approx 1.8 \times 10^{14}$ 个可能序列。如果禁止把一个皇后放到可能被攻击的格子里，这样的形式化可能更好：

- **状态**：n 皇后在棋盘上（$0 \leqslant n \leqslant 8$）的任意摆放，满足从最左边 n 列里每列一个皇后，保证没有皇后能攻击另一个。
- **行动**：在最左侧的空列中选择一格摆放 1 皇后，要求该格子未受到其他皇后攻击。

这样的形式化把八皇后问题的状态空间从 1.8×10^{14} 降到了 2057，解就容易找到了。另一方面，对于 100 个皇后，状态空间从约 10^{400} 个状态减少到约 10^{52} 个状态（习题 3.5），这是很大的改进，但还不足以使得问题容易求解。4.1 节给出了完整状态的形式化，第 6 章给出了一个简单的算法，可以轻易地解决甚至百万个皇后问题。

书中第六章：

令人惊讶的是最小冲突对许多 CSP 都有效。神奇的是在 n 后问题中，如果不依赖于皇后的初始放置情况，最少冲突算法的运行时间大体上独立于问题的规模。它甚至能在平均（初始赋值之后）50 步之内求解百万皇后问题。这个不同寻常的现象导致 20 世纪 90 年代大量研究关注局部搜索和难易问题之间的区别，第 7 章中会进一步讨论。大致来说，对局部搜索求解 n 后问题十分容易，因为解密集地分布于整个状态空间。最少冲突算法也适用于难题求解。例如，它用于安排哈勃太空望远镜的观察日程时间表，安排一周的观察日程所花费的时间从三周（!）减少到了大概 10 分钟。

以八皇后问题为例，回溯搜索算法的处理流程如下：

初始化：创建一个8x8的棋盘，用于放置皇后。

递归函数：定义一个递归函数，用于在棋盘上放置皇后，并检查放置是否合法。

约束检查：在放置每个皇后时，检查是否与已放置的皇后在同一行、列或对角线上。

回溯：如果放置皇后后发现不合法，回退一步，尝试在上一行放置皇后的其他位置。

目标检查：如果所有皇后都成功放置，记录这一解决方案。

三、实验及结果分析

（1）开发语言及运行环境

与实验一相同，不再赘述。

（2）实验的具体步骤

1. 初始化 __init__

设置棋盘大小 board_size（默认8×8），初始化解决方案列表 solutions，设置最大解决方案数量 max_solutions（默认5），初始化棋盘 board：如果没有初始棋盘，创建全-1的空棋盘；如果有初始棋盘，验证长度并复制。

安全性检查 is_safe：遍历当前行之前的所有行，检查同列冲突和对角线冲突。如果没有冲突，返回True。

解决方案入口 solve_n_queens：找到第一个未放置皇后的行（start_row），用回溯函数backtrack开始求解，并返回所有解决方案。

2. 回溯的核心实现 backtrack

终止条件检查：如果解的数量达到上限，返回；如果处理完所有行，将当前解加入solutions。

回溯过程：

在当前行的每一列尝试放置皇后。

对每个安全的位置：放置皇后（board[row] = col）、递归处理下一行、撤销当前选择（board[row] = -1）。

print_solution 打印单个解决方案，print_all_solutions 打印所有解决方案。

3. 主函数 main

示例1：使用空棋盘求解

示例2：使用带初始皇后位置的棋盘求解

执行流程：

1. 创建求解器实例
2. 调用solve_n_queens开始求解
3. 通过回溯算法寻找所有解
4. 输出结果和可视化展示

（3）根据实验数据集，按实验要求给出相应的结果（截图）并对实验结果进行简要分析。

① main.py:

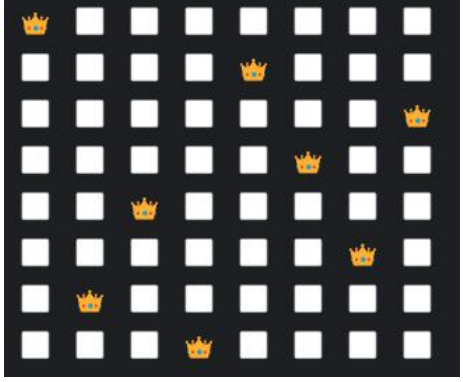
代码的输出有两部分，第一部分是对空棋盘的求解，算法将给出 5 种解决方案，第二部分是自定义棋盘的求解，如果有解会输出解，没有则输出未找到解决方案。

对于空棋盘：仅展示第一个解决方案。

```
示例1：默认空棋盘
找到的解决方案数量： 5

找到 5 个解决方案：

解决方案 1：
```



对于自定义棋盘：

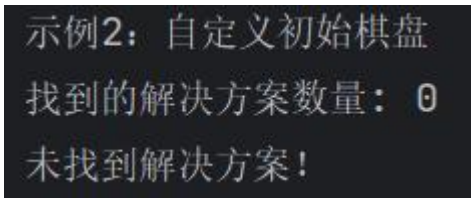
重现教材图 3.5 的情况，它是一种失败的尝试，因为左上角和右下角在同一对角线上，在我的代码中会显示错误：

```
92      initial_board = [-1] * 8
93      initial_board[0] = 0  # 第一行第一列
94      initial_board[1] = 4
95      initial_board[2] = 1
96      initial_board[3] = 5
97      initial_board[4] = 2
98      initial_board[6] = 3
99      initial_board[7] = 7
```

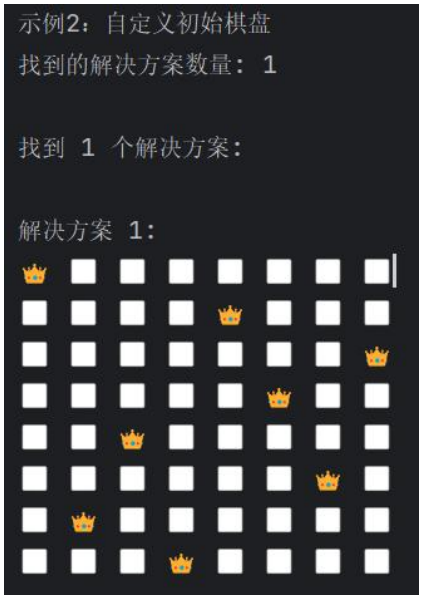
```
示例2：自定义初始棋盘
Traceback (most recent call last):
  File "/home/ma-user/work/project2/main.py", line 107, in <module>
    main()
  File "/home/ma-user/work/project2/main.py", line 100, in main
    solver2 = EightQueensSolver(initial_board=initial_board)
  File "/home/ma-user/work/project2/main.py", line 22, in __init__
    raise ValueError(f"在第 {row} 行，第 {col} 列的皇后放置无效")
ValueError: 在第 7 行，第 7 列的皇后放置无效
```

因为此时会发生冲突，这种初始化方式是不正确的，此时去掉(7,7)的皇后，

再次运行代码。是没有解的情况：



保留前两行，有解的情况：



② **main2.py:**

修改了一点输出方式，可以得到回溯的过程。

四、心得

没有遇到特别的困难，但对八皇后问题（主要是棋盘有关的问题）有了更深

的理解，比如如何计算冲突，横着和对角线。

五、程序文件名清单

main.py：源代码

main.py.pdf：源代码 pdf 版

main2.py：源代码 2

main2.py.pdf：源代码 2 pdf 版

六、附录

代码如下。

main.py

```python
import mindspore
import mindspore.nn as nn
import numpy as np


class EightQueensSolver:
    def __init__(self, board_size=8, initial_board=None, max_solutions=5):
        self.board_size = board_size
        self.solutions = []
        self.max_solutions = max_solutions

        if initial_board is None:
            self.board = [-1] * board_size
        else:
            if len(initial_board) != board_size:
                raise ValueError(f"初始棋盘长度必须为 {board_size}")
            self.board = initial_board.copy()

    def is_safe(self, board, row, col):
        # 检查当前行之前的所有行
        for i in range(row):
            # 只检查已经放置了皇后的位置
            if board[i] != -1:
                # 检查是否在同一列
                if board[i] == col:
                    return False

                # 检查对角线
                # 两点在对角线上的条件：行之差的绝对值 = 列之差的绝对值
                row_diff = row - i
                col_diff = abs(col - board[i])
                if row_diff == col_diff:
                    return False
        return True

    def solve_n_queens(self):
        """解决八皇后问题"""
        # 找到第一个未放置皇后的行
        start_row = 0
        while start_row < self.board_size and self.board[start_row] != -1:
            start_row += 1

        self.backtrack(self.board, start_row)
        return self.solutions

    def backtrack(self, board, row):
        """使用回溯法求解"""
        # 如果已经超过最大解数量，停止搜索
        if len(self.solutions) >= self.max_solutions:
            return

        # 如果已经处理完所有行，说明找到一个解
        if row == self.board_size:
            self.solutions.append(board.copy())
            return

        # 在当前行尝试每一列
        for col in range(self.board_size):
            if self.is_safe(board, row, col):
                board[row] = col  # 放置皇后
                self.backtrack(board, row + 1)  # 递归处理下一行
                board[row] = -1  # 回溯，撤销当前选择

    def print_solution(self, solution, solution_num=None):
        if solution_num is not None:
            print(f"\n解决方案 {solution_num}:")
```

```
67
68          for row in range(self.board_size):
69              line = []
70              for col in range(self.board_size):
71                  line.append('👑' if solution[row] == col else '⬜')
72              print(' '.join(line))
73
74      def print_all_solutions(self):
75          """打印所有找到的解决方案（最多5个）"""
76          if not self.solutions:
77              print("未找到解决方案！")
78              return
79
80          print(f"\n找到 {len(self.solutions)} 个解决方案:")
81          for idx, solution in enumerate(self.solutions, 1):
82              self.print_solution(solution, idx)
83              print()
84
85
86  def main():
87      # 示例1：默认空棋盘
88      print("示例1：默认空棋盘")
89      solver1 = EightQueensSolver()
90      solver1.solve_n_queens()
91      print(f"找到的解决方案数量: {len(solver1.solutions)}")
92      solver1.print_all_solutions()
93
94      # 示例2：自定义初始棋盘
95      print("\n示例2：自定义初始棋盘")
96      initial_board = [-1] * 8
97      initial_board[0] = 0  # 第一行第一列
98      initial_board[1] = 4
99      #initial_board[2] = 1
100     #initial_board[3] = 5
101     #initial_board[4] = 2
102     #initial_board[6] = 3
103     #initial_board[7] = 7
104     solver2 = EightQueensSolver(initial_board=initial_board)
105     solver2.solve_n_queens()
106     print(f"找到的解决方案数量: {len(solver2.solutions)}")
107     solver2.print_all_solutions()
108
109
110 if __name__ == "__main__":
111     main()
```

main2.py

```python
import mindspore
import mindspore.nn as nn
import numpy as np


class EightQueensSolver:
    def __init__(self, board_size=8, initial_board=None, max_solutions=1):
        self.board_size = board_size
        self.solutions = []
        self.max_solutions = max_solutions
        self.step_count = 0

        if initial_board is None:
            self.board = [-1] * board_size
        else:
            if len(initial_board) != board_size:
                raise ValueError(f"初始棋盘长度必须为 {board_size}")
            self.board = initial_board.copy()

    def is_safe(self, board, row, col):
        for i in range(row):
            if board[i] != -1:
                if board[i] == col:
                    return False
                row_diff = row - i
                col_diff = abs(col - board[i])
                if row_diff == col_diff:
                    return False
        return True

    def solve_n_queens(self):
        self.step_count = 0
        start_row = 0
        while start_row < self.board_size and self.board[start_row] != -1:
            start_row += 1

        print("\n开始求解过程：")
        print("初始状态：")
        self.print_solution(self.board)
        print("\n")

        self.backtrack(self.board, start_row)
        return self.solutions

    def backtrack(self, board, row):
        if len(self.solutions) >= self.max_solutions:
            return

        if row == self.board_size:
            self.solutions.append(board.copy())
            return

        for col in range(self.board_size):
            if self.is_safe(board, row, col):
                board[row] = col
                self.step_count += 1

                print(f"步骤 {self.step_count}:")
                print(f"在第 {row} 行，第 {col} 列放置皇后")
                self.print_solution(board)
                print("\n")

                self.backtrack(board, row + 1)

                if self.solutions:
                    return
```

```
67
68                  board[row] = -1
69                  print(f"回溯：移除第 {row} 行，第 {col} 列的皇后")
70                  self.print_solution(board)
71                  print("\n")
72
73      def print_solution(self, solution, solution_num=None):
74          if solution_num is not None:
75              print(f"\n解决方案 {solution_num}:")
76
77          print("   " + " ".join(str(i) for i in range(self.board_size)))
78
79          for row in range(self.board_size):
80              print(f"{row}: ", end="")
81              line = []
82              for col in range(self.board_size):
83                  line.append('👑' if solution[row] == col else '⬜')
84              print(' '.join(line))
85
86
87  def main():
88      '''# 示例1：默认空棋盘
89      print("示例1：默认空棋盘")
90      solver1 = EightQueensSolver()
91      solver1.solve_n_queens()
92      print(f"找到的解决方案数量：{len(solver1.solutions)}")'''
93
94      # 示例2：自定义初始棋盘
95      print("\n示例2：自定义初始棋盘")
96      initial_board = [-1] * 8
97      initial_board[0] = 0  # 第一行第一列
98      initial_board[1] = 4
99      #initial_board[2] = 1
100     #initial_board[3] = 5
101     # initial_board[4] = 2
102     # initial_board[6] = 3
103     # initial_board[7] = 7
104
105     solver2 = EightQueensSolver(initial_board=initial_board)
106     solver2.solve_n_queens()
107     print(f"找到的解决方案数量：{len(solver2.solutions)}")
108
109
110 if __name__ == "__main__":
111     main()
```