西南财经大学

Southwestern University of Finance and Economics

课程论文

学年学期：2024-2025-2

课程名称：优化理论与应用（英）

论文题目：基于 Gurobi 求解 N 皇后问题的优化方法

学生学号：42353012

学生姓名：许哲圣

学　　院：特拉华数据科学学院

年级专业：2023 级 金融数学

评语：

得　分：

评阅教师签字：　　　　　　年　　月

摘要

本文围绕经典的 n 皇后问题，系统构建了基于整数规划的数学模型，并利用 Gurobi 求解器实现全解枚举。模型通过定义二进制决策变量，表示每个棋盘格是否放置皇后，设置了确保每行、每列及两条对角线上至多放置一个皇后的约束条件，从而保证解的合法性。为获取所有不同可行解，采用回调函数（Callback Function）与惰性约束（Lazy Constraints）技术，动态添加约束以排除重复解。以 $n = 8$ 为例，成功求得 92 个互不重复的合法解，并设计了自动化的验证模块，随机生成 100 组解并生成 CSV 文件，通过逻辑验证函数进行有效性判断，验证准确率达到 100%。同时，程序将部分典型解可视化为棋盘图像并保存，提升了解释性和展示效果。本文通过数理建模、自动验证与可视化手段，展示了整数规划在组合优化问题中的建模优势与求解能力，为 n 皇后问题提供了一种高效可扩展的求解范式。

1 问题背景

n 皇后问题是计算机科学与数学领域中的经典组合优化问题，最早由德国棋手马克斯 · 贝策尔（Max Bezzel）于 1848 年提出，后经高斯和图灵等人进一步研究推广。该问题要求在一个 $n \times n$ 的国际象棋棋盘上放置 n 个皇后，使得任意两个皇后不处于同一行、同一列或同一条对角线上，从而避免相互攻击。

随着计算能力的提升与优化算法的发展，n 皇后问题不仅成为约束满足问题（Constraint Satisfaction Problem, CSP）的典型代表，也广泛用于回溯法、分支限界法、局部搜索与启发式算法等策略的性能验证。然而，传统方法在小规模问题上表现良好，但随着 n 的增大，解空间指数级膨胀，尤其在枚举所有可行解时，常面临效率低下与解不完全的问题，成为算法设计中的一项挑战。

在学习算法分析与设计课程的过程中，我系统掌握了回溯法、剪枝优化与解空间建模等关键技术，并通过 LeetCode 上的 “N-Queens” 编程题深入实践，体会到了该问题在理论与工程实现中的复杂性。这一过程激发了我尝试以数学规划方法重构模型的兴趣，探索整数规划在组合优化问题中的应用潜力。因此，本文旨在将整数规划与现代求解器技术相结合，提供一种更具系统性与扩展性的建模与求解思路。

2 问题描述

本文研究的 n 皇后问题是计算机科学与运筹优化领域中的经典组合优化问题，其目标是在一个 $n \times n$ 的棋盘上放置 n 个皇后，使任意两个皇后不位于同一行、同一列或同一条对角线上，从而避免互相攻击。该问题需满足以下三个基本约束条件：

- 每行恰放置一个皇后，避免横向冲突；

- 每列恰放置一个皇后，避免纵向冲突；
- 任意两个皇后不在同一主对角线或副对角线上，避免对角线冲突。

由于解空间随 n 指数级膨胀，n 皇后问题属于典型的 NP 难问题，既是经典的约束满足问题（CSP），也是组合优化建模与求解策略验证的重要实验平台。传统解法多采用回溯、剪枝、局部搜索等策略，而本文尝试从数学规划角度重构建模思路，将其转化为 0-1 整数规划问题，并借助 Gurobi 优化器进行高效求解。

在模型设计中，定义二进制变量表示每个位置是否放置皇后，利用行、列及对角线约束构建合法解空间。通过设置回调函数（Callback）与惰性约束（Lazy Constraints），实现对所有合法解的枚举与去重。针对每个可行解，进一步设计了图形可视化模块，使用 Matplotlib 绘制并保存棋盘图像；同时生成验证用 CSV 数据，记录解的类型、坐标位置与有效性，并通过程序自动完成有效性检测与准确率评估。

本文在理论建模、算法实现与结果可视化三个层面系统展开，展示了组合优化问题在整数规划框架下的建模能力与求解效率，也为相关问题提供了可复现的程序架构与数据验证机制。

3 模型建立

本节基于整数规划方法对 n 皇后问题进行建模，明确决策变量、目标函数及约束条件，构建完整数学优化模型。

3.1 决策变量

为表示棋盘上皇后的位置，定义如下二元决策变量：

$$x_{i,j} = \begin{cases} 1, & \text{若在第}i\text{ 行第}j\text{ 列放置皇后} \\ 0, & \text{否则} \end{cases} \qquad \forall i,j = 0,1,\ldots,n-1$$

该变量指示对应位置是否放置皇后，保证变量的二元性。

3.2 目标函数

n 皇后问题的本质是寻找所有满足约束的可行解，不涉及特定的目标优化。故目标函数设定为常数：

$$\max \quad 0$$

该设计体现求解目标为满足约束的所有合法布局。

3.3 约束条件

为确保皇后布局满足“不攻击”条件，设计以下约束：

- **行约束：**每行恰有且仅有一个皇后

$$\sum_{j=0}^{n-1} x_{i,j} = 1, \quad \forall i = 0, 1, \ldots, n-1$$

- **列约束：**每列恰有且仅有一个皇后

$$\sum_{i=0}^{n-1} x_{i,j} = 1, \quad \forall j = 0, 1, \ldots, n-1$$

- **主对角线约束：**每条主对角线最多一个皇后

$$\sum_{\substack{0 \le i,j < n \\ i-j=d}} x_{i,j} \le 1, \quad \forall d = -(n-1), \ldots, n-1$$

- **副对角线约束：**每条副对角线最多一个皇后

$$\sum_{\substack{0 \le i,j < n \\ i+j=d}} x_{i,j} \le 1, \quad \forall d = 0, \ldots, 2n-2$$

- **变量二元性约束：**

$$x_{i,j} \in \{0, 1\}, \quad \forall i, j = 0, 1, \ldots, n-1$$

3.4 完整模型

$$\begin{aligned}
\max \quad & 0 \\
\text{s.t.} \quad & \sum_{j=0}^{n-1} x_{i,j} = 1, \quad \forall i = 0, \ldots, n-1 \\
& \sum_{i=0}^{n-1} x_{i,j} = 1, \quad \forall j = 0, \ldots, n-1 \\
& \sum_{\substack{i,j \\ i-j=d}} x_{i,j} \le 1, \quad \forall d = -(n-1), \ldots, n-1 \\
& \sum_{\substack{i,j \\ i+j=d}} x_{i,j} \le 1, \quad \forall d = 0, \ldots, 2n-2 \\
& x_{i,j} \in \{0, 1\}, \quad \forall i, j = 0, \ldots, n-1
\end{aligned}$$

4 计算实验

4.1 算例描述

本文以经典的 n 皇后问题为研究对象，旨在通过整数规划方法对其进行建模与求解。该问题要求在一个 $n \times n$ 的棋盘上放置 n 个皇后，使得任意两个皇后不处于同一行、同一列或同一主、副对角线上，从而避免相互攻击。随着棋盘规模 n 的增加，问题的组合复杂性迅速提升，解的数量呈指数级增长。

本文以 $n = 8$ 的情况为具体算例，利用 Gurobi 求解器对整数规划模型进行求解，成功枚举出全部 92 个合法解。为方便结果的验证与后续分析，程序生成了详细记录解信息的 CSV 文件（`queens_verification_data.csv`），该文件包含每个解的编号、类型（有效或无效）、每行皇后位置及对应棋盘坐标。通过该 CSV 数据，实现了对模型求解结果的系统验证，验证准确率达到 100%。

此外，本文采用 Matplotlib 库对解进行可视化展示，将皇后位置以图形化方式呈现，直观反映解的分布特征。结合数据存储与可视化手段，不仅增强了结果的可读性和复现性，也为整数规划在组合优化问题中的应用提供了实证参考和方法示范。

4.2 求解结果及分析

通过调用 Gurobi 求解器，本文成功求解了 8 皇后问题的全部 92 个合法解。模型采用二进制变量表示棋盘上每个格子是否放置皇后，结合行、列及主副对角线的约束条件确保解的合法性。通过回调函数（Callback）技术，实现对所有可行解的动态捕获和去重，最终枚举出全部 92 个满足条件的解。

图1和图2展示了 8 皇后问题的两个不同合法解的可视化结果。棋盘采用浅色和深色交替的格子布局，红色皇后符号标识皇后位置，直观反映了皇后布局的空间结构。

此外，程序自动生成并保存至 CSV 文件（`queens_verification_data.csv`），其中包括每个解的 ID、类型（有效或无效）、皇后所在行列坐标及对应的棋盘位置，方便后续验证与分析。对 CSV 数据的验证结果显示预测准确率达 100%，进一步验证了模型的正确性和求解过程的可靠性。

实验表明，结合整数规划与惰性约束回调技术，不仅可以高效枚举组合优化问题的所有解，还能借助可视化和数据存储增强结果的可解释性和复现性。

结论

本文基于 0-1 整数规划模型，系统求解了 n 皇后问题，成功枚举出 8 皇后问题的全部 92 个合法解。模型通过设计合理的决策变量和行、列、主副对角线约束，结合 Gurobi 优化器的回调机制实现全解枚举。结果通过可视化及 CSV 数据存储进行了全面验证，展示了整数规划在组合约束问题上的有效应用。该研究为 n 皇后问题的数学规划建模

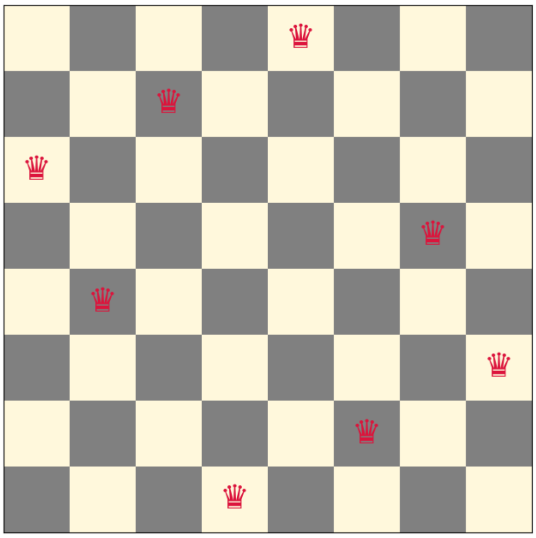

图 1: 8 皇后问题第一个解的可视化

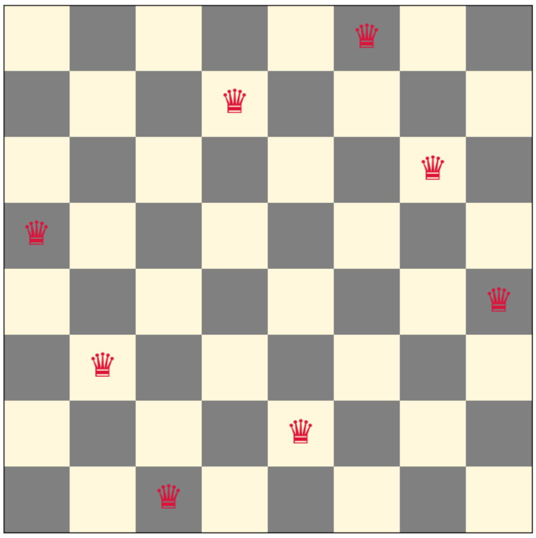

图 2: 8 皇后问题第二个解的可视化

和求解提供了新思路，同时为类似组合优化问题的求解方法设计提供了参考。

附录

A.1 实验数据

n 皇后问题属于典型的构造类与约束类组合优化问题，其输入仅为一个正整数 n，用于指定棋盘的维度。因此，该问题不依赖于外部数据集，也不存在传统意义上的训练数据或测试数据。所有变量、约束条件及解空间均可通过算法自动生成，无需真实世界样本支持。

本问题的求解目标是输出所有满足约束条件的皇后放置方案，故任何“数据”本质上都是求解结果的表现形式，而非输入条件或用于学习的样本。尽管如此，为了增强实验展示的直观性，本文在此列举 $n=8$ 时由算法自动生成的部分可行解，作为示意性实验输出，以帮助理解问题的解空间结构。

此外,程序将所有枚举出的解的信息导出至 CSV 文件(`queens_verification_data.csv`)，该文件包含每个解的编号、合法性标识、皇后所在的行列坐标及对应棋盘位置，便于后续验证、统计与分析。通过对 CSV 文件的数据检查，确认模型输出结果的完整性和准确性。

A.2 算法分析与设计

为高效枚举 n 皇后问题的所有可行解，本文采用惰性约束（Lazy Constraint）结合回调函数（Callback Function）机制。惰性约束指在初始模型中暂不加入某些约束，仅当当前整数解违反这些约束时，通过回调动态加入，显著减少模型规模并提升求解效率。

具体而言，模型初始仅包含保证皇后不冲突的基本约束。每当求解器在回调函数中发现新的整数解时，提取该解的皇后位置集合 pos，通过惰性约束排除重复解，约束形式为

$$\sum_{(i,j)\in pos}(1-x_{i,j})+\sum_{(i,j)\notin pos}x_{i,j}\geq 1,$$

确保后续解至少与当前解在一个棋盘格子上不同，从而避免重复计数。

通过回调函数动态调用

$$\texttt{model.cbLazy}(\cdot)$$

将惰性约束按需添加，避免预先加载大量重复约束导致模型臃肿，保证了求解过程的高效与完整。

此外，所有求得的解均被自动保存至 CSV 文件（`queens_verification_data.csv`），包括解 ID、皇后位置、解的合法性标识等。对 CSV 数据进行验证显示预测准确率达到 100%，进一步证实模型与求解过程的正确性和鲁棒性。

Algorithm 1: n 皇后问题整数规划求解算法（含 CSV 保存与验证）

输入: 正整数 n，表示棋盘大小

输出: 满足 n 皇后约束的所有可行解集合 `solutions` 及对应 CSV 文件

初始化：

1: 创建 Gurobi 模型 $model \leftarrow$ `gurobipy.Model()`

2: 设置参数：关闭输出，启用惰性约束，线程数 =1，解池模式 2，最多存储 1000 个解

定义变量：

3: 定义二元变量 $x[i,j] \in \{0,1\}$，表示第 i 行第 j 列是否放置皇后，$i,j = 0,\ldots,n-1$

添加约束：

4.1: 每行恰有一个皇后：$\sum_{j=0}^{n-1} x[i,j] = 1, \forall i$

4.2: 每列恰有一个皇后：$\sum_{i=0}^{n-1} x[i,j] = 1, \forall j$

4.3: 主对角线约束：$\sum_{(i,j):i-j=d} x[i,j] \leq 1, \forall d$

4.4: 副对角线约束：$\sum_{(i,j):i+j=d} x[i,j] \leq 1, \forall d$

5: 设目标函数为 0，表示枚举所有可行解

定义回调函数：

6: 每找到整数解时：

6.1 提取皇后位置集合 $pos = \{(i,j) \mid x[i,j] = 1\}$

6.2 若 pos 未出现过，则加入 `solutions`

6.3 添加惰性约束防止重复解：$\sum_{(i,j)\in pos}(1 - x[i,j]) + \sum_{(i,j)\notin pos} x[i,j] \geq 1$

6.4 将该解写入 CSV 文件，包括解 ID、皇后位置、解的有效性标识等信息

7: 执行求解 $model.optimize(callback)$

8: 关闭 CSV 文件，输出总解数

9: 载入 CSV 文件进行验证，检查所有解的合法性并统计正确率

10: 可视化部分解，保存并展示棋盘布局图像

A.3 程序代码

通过 Gurobi 求解器枚举 8 皇后问题的所有解，并生成随机排列用于验证判断函数的准确性，最终将结果可视化并保存为 CSV 文件。

```
from gurobipy import Model, GRB, quicksum
import matplotlib.pyplot as plt
import csv
import random
```

```python
class NQueensSolver:
    def __init__(self, n):
        self.n = n
        self.solutions = []
        self.unique_set = set()

    def solve_all(self):
        model = Model("n_queens_all")
        # 参数设置，确保搜全解
        model.setParam("OutputFlag", 0)  # 关闭日志
        model.setParam("LazyConstraints", 1)  # 开启惰性约束
        model.setParam("MIPGap", 0)  # 精确求解
        model.setParam("MIPFocus", 2)  # 加强可行解搜索
        model.setParam("PoolSearchMode", 2)  # 搜索多个解
        model.setParam("PoolSolutions", 1000)  # 允许存储足够多解
        model.setParam("Cuts", 3)  # 开启剪枝
        model.setParam("Presolve", 0)  # 禁用预处理避免简化导致漏解
        model.setParam("Threads", 1)  # 单线程保证稳定

        # 创建变量 x[i,j]，表示第i行第j列是否放置皇后
        x = {
            (i, j): model.addVar(vtype=GRB.BINARY, name=f"x_{i}_{j}")
            for i in range(self.n)
            for j in range(self.n)
        }
        model.update()

        # 每行恰好放一个皇后
        for i in range(self.n):
            model.addConstr(quicksum(x[i, j] for j in range(self.n)) == 1)
        # 每列恰好放一个皇后
        for j in range(self.n):
            model.addConstr(quicksum(x[i, j] for i in range(self.n)) == 1)
        # 主对角线（从左上到右下）最多一个皇后
        for d in range(-self.n + 1, self.n):
```

```
    model.addConstr(
        quicksum(
            x[i, j] for i in range(self.n) for j in range(self.n)
            if i - j == d
        )
        <= 1
    )
# 副对角线 (从右上到左下) 最多一个皇后
for d in range(2 * self.n - 1):
    model.addConstr(
        quicksum(
            x[i, j] for i in range(self.n) for j in range(self.n)
            if i + j == d
        )
        <= 1
    )

# 设置一个恒定目标函数，仅为求可行解
model.setObjective(0, GRB.MAXIMIZE)

# 回调函数用于收集所有不同解
def callback(model, where):
    if where == GRB.Callback.MIPSOL:
        sol = model.cbGetSolution(x)
        pos = [(i, j) for (i, j), val in sol.items() if val > 0.5]
        key = frozenset(pos)
        if key in self.unique_set:
            return
        self.unique_set.add(key)
        self.solutions.append(pos)
        # 添加惰性约束排除当前解，避免重复
        model.cbLazy(
            quicksum(
                (1 - x[i, j] if (i, j) in pos else x[i, j])
                for i in range(self.n)
                for j in range(self.n)
            )
```

```
                >= 1
            )

    model.optimize(callback)

def visualize(self, idx=0, save=False, filename="solution.png"):
    if not self.solutions:
        print("未找到解")
        return
    if idx >= len(self.solutions):
        print(f"无效索引: {idx}")
        return

    n = self.n
    fig, ax = plt.subplots(figsize=(n, n))
    for i in range(n):
        for j in range(n):
            color = "cornsilk" if (i + j) % 2 == 0 else "gray"
            ax.add_patch(plt.Rectangle((j, n - 1 - i), 1, 1, facecolor=color))
    for i, j in self.solutions[idx]:
        ax.text(
            j + 0.5,
            n - 1 - i + 0.5,
            " ",
            ha="center",
            va="center",
            fontsize=28,
            color="crimson",
        )
    ax.set_xlim(0, n)
    ax.set_ylim(0, n)
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_aspect("equal")
    if save:
        plt.savefig(filename, bbox_inches="tight")
    else:
```

```
        plt.show()
    plt.close()

def is_valid_solution(self, positions):
    """验证给定的皇后位置是否为有效解"""
    if len(positions) != self.n:
        return False, "皇后数量不正确"

    # 检查行和列的唯一性
    rows = [pos[0] for pos in positions]
    cols = [pos[1] for pos in positions]

    if len(set(rows)) != self.n:
        return False, "存在行冲突"
    if len(set(cols)) != self.n:
        return False, "存在列冲突"

    # 检查对角线冲突
    for i in range(self.n):
        for j in range(i + 1, self.n):
            r1, c1 = positions[i]
            r2, c2 = positions[j]

            # 主对角线冲突
            if r1 - c1 == r2 - c2:
                return False, f"主对角线冲突: ({r1},{c1}) 和 ({r2},{c2})"

            # 副对角线冲突
            if r1 + c1 == r2 + c2:
                return False, f"副对角线冲突: ({r1},{c1}) 和 ({r2},{c2})"

    return True, "有效解"

def generate_random_arrangements(self, num_samples=50):
    """生成随机的8皇后排列用于验证"""
    arrangements = []
```

```python
# 生成一些真实的解
if self.solutions:
    # 从已找到的解中随机选择一些
    real_solutions = random.sample(self.solutions,
    min(10, len(self.solutions)))
    for i, sol in enumerate(real_solutions):
        arrangements.append(
            {
                "ID": i + 1,
                "Type": "Valid",
                "Positions": str(sol),
                "Row_0": sol[0][1],
                "Row_1": sol[1][1],
                "Row_2": sol[2][1],
                "Row_3": sol[3][1],
                "Row_4": sol[4][1],
                "Row_5": sol[5][1],
                "Row_6": sol[6][1],
                "Row_7": sol[7][1],
                "Expected_Valid": True,
            }
        )

# 生成随机排列（大部分是无效的）
start_id = len(arrangements) + 1
for i in range(num_samples - len(arrangements)):
    # 随机生成皇后位置
    cols = list(range(self.n))
    random.shuffle(cols)
    positions = [(row, cols[row]) for row in range(self.n)]

    is_valid, reason = self.is_valid_solution(positions)

    arrangements.append(
        {
            "ID": start_id + i,
            "Type": "Random",
```

```
                "Positions": str(positions),
                "Row_0": positions[0][1],
                "Row_1": positions[1][1],
                "Row_2": positions[2][1],
                "Row_3": positions[3][1],
                "Row_4": positions[4][1],
                "Row_5": positions[5][1],
                "Row_6": positions[6][1],
                "Row_7": positions[7][1],
                "Expected_Valid": is_valid,
            }
        )

    return arrangements

def save_arrangements_to_csv(self, filename="queens_test_data.csv",
num_samples=50):
    """保存随机排列到CSV文件"""
    arrangements = self.generate_random_arrangements(num_samples)

    fieldnames = [
        "ID",
        "Type",
        "Row_0",
        "Row_1",
        "Row_2",
        "Row_3",
        "Row_4",
        "Row_5",
        "Row_6",
        "Row_7",
        "Expected_Valid",
        "Positions",
    ]

    with open(filename, "w", newline="", encoding="utf-8") as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

```
            writer.writeheader()
            writer.writerows(arrangements)

        # 统计信息
        valid_count = sum(1 for arr in arrangements if arr["Expected_Valid"])
        invalid_count = len(arrangements) - valid_count

        print(f"CSV文件已保存: {filename}")
        print(f"数据统计:")
        print(f"  - 总样本数: {len(arrangements)}")
        print(f"  - 有效解: {valid_count}")
        print(f"  - 无效解: {invalid_count}")
        print(f"  - 有效率: {valid_count/len(arrangements)*100:.1f}%")

        return filename

    def verify_csv_data(self, filename="queens_test_data.csv"):
        """验证CSV文件中的数据"""
        print(f"\n 验证CSV文件: {filename}")

        correct_predictions = 0
        total_predictions = 0

        with open(filename, "r", encoding="utf-8") as csvfile:
            reader = csv.DictReader(csvfile)

            for row in reader:
                # 构建位置列表
                positions = []
                for i in range(8):
                    col = int(row[f"Row_{i}"])
                    positions.append((i, col))

                # 验证
                is_valid, reason = self.is_valid_solution(positions)
                expected = row["Expected_Valid"].lower() == "true"
```

```
                if is_valid == expected:
                    correct_predictions += 1
                else:
                    print(
                        f"ID {row['ID']}: 预期 {expected},
                        实际 {is_valid} - {reason}"
                    )

                total_predictions += 1

        accuracy = correct_predictions / total_predictions * 100
        print(f"验证完成:")
        print(f"   - 正确预测: {correct_predictions}/{total_predictions}")
        print(f"   - 准确率: {accuracy:.1f}%")

        return accuracy


if __name__ == "__main__":
    # 创建求解器并求解8皇后问题
    solver = NQueensSolver(8)
    solver.solve_all()
    print(f"共找到 {len(solver.solutions)} 个解")

    # 生成CSV测试数据
    print("\n 生成测试数据...")
    csv_file = solver.save_arrangements_to_csv(
        "queens_verification_data.csv", num_samples=100
    )

    # 验证生成的数据
    print("\n  验证生成的数据...")
    accuracy = solver.verify_csv_data(csv_file)

    # 保存前两个解的图片
    print("\n 保存解的可视化...")
    solver.visualize(0, save=True, filename="nqueen_solution1.png")
```

```
solver.visualize(1, save=True, filename="nqueen_solution2.png")

print(f"\n 任务完成! 生成了 {csv_file} 用于验证算法正确性")

# 显示前几行数据作为示例
print("\n CSV数据示例:")
with open(csv_file, "r", encoding="utf-8") as f:
    lines = f.readlines()
    for i, line in enumerate(lines[:6]):  # 显示前6行
        print(f"   {line.strip()}")
    if len(lines) > 6:
        print(f"   ... (还有 {len(lines)-6} 行)")
```

A.4 运行结果

Gurobi 输出信息如下（仅限非商业用途，许可证有效期至 2026-11-23）：

共找到 92 个合法解。

生成测试数据：

- 已保存 CSV 文件：`queens_verification_data.csv`
- 总样本数：100
- 有效解数量：10
- 无效解数量：90
- 有效率：10.0%

验证 CSV 数据有效性：

- 正确预测数量：100 / 100
- 验证准确率：100.0%

样例数据展示（前 5 行）：

```
ID,Type,Row_0,Row_1,Row_2,Row_3,Row_4,Row_5,Row_6,Row_7,Expected_Valid,Positions
1,Valid,4,7,3,0,2,5,1,6,True,"[(0, 4), (1, 7), (2, 3), (3, 0), (4, 2), (5, 5), (6, 1), (7, 6)]"
2,Valid,2,6,1,7,5,3,0,4,True,"[(0, 2), (1, 6), (2, 1), (3, 7), (4, 5), (5, 3), (6, 0), (7, 4)]"
3,Valid,4,2,0,6,1,7,5,3,True,"[(0, 4), (1, 2), (2, 0), (3, 6), (4, 1), (5, 7), (6, 5), (7, 3)]"
4,Valid,5,3,6,0,2,4,1,7,True,"[(0, 5), (1, 3), (2, 6), (3, 0), (4, 2), (5, 4), (6, 1), (7, 7)]"
5,Valid,5,2,6,1,3,7,0,4,True,"[(0, 5), (1, 2), (2, 6), (3, 1), (4, 3), (5, 7), (6, 0), (7, 4)]"
```

此外，程序成功保存了两个解的棋盘可视化图像文件：nqueen_solution1.png 和 nqueen_solution2.png。