囚犯抽签问题

- IO部分（包含输入输出提示）：通过两个独立函数实现用户输入和输出的交互，同时将输出与绘图代码分离，另外使用默认值，交互上更友好
- 主体算法逻辑（大模拟，包含拆分单轮模拟和多轮模拟）
- 理论值根据wiki结果计算
- 启发式搜索策略模块：将搜索策略抽象为类，作为模拟时的方案，实现搜索策略与模拟逻辑的解耦（这里实现了随机策略与循环策略）
- 多命令设计（模拟与绘图）：使用fire包来实现命令行接口，便于不同功能的调用

额外优化

- 使用uv依赖头，便于直接使用uv来执行脚本
- 使用ruff对代码进行格式化，规范性更好

实验结果

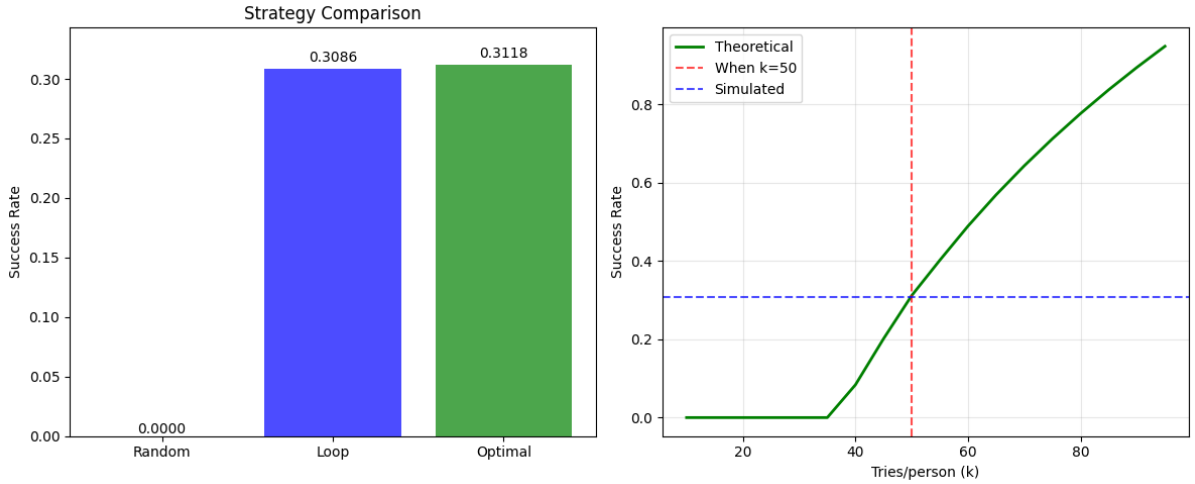
运行结果

使用默认配置运行 循环策略的结果与理论值相当

```
uv run -p 3.11 prison.py solve
```

```
% uv run -p 3.11 prisondraw/prison.py solve
请输入囚犯数量 N（默认100）:
请输入每人尝试次数 K（默认50）:
请输入模拟轮次 T（默认10000）:

开始模拟 (n=100, k=50, trials=10000)...

==================================================
囚犯抽签问题模拟结果
==================================================
参数设置:
  囚犯数量: 100
  每人尝试次数: 50
  模拟轮次: 10000

结果:
  策略1（随机搜索）成功率: 0.0000 (0.00%)
  策略2（循环策略）成功率: 0.3086 (30.86%)

  循环策略相对优势: 308600.00倍
  理论计算的循环策略成功率: 0.3118
==================================================

是否显示结果图表？(y/n, 默认y):
```

下面左图为随机策略，循环策略与理论值的对比。右图为该配置下，理论值随人均尝试数的变化。



敏感性分析

```
uv run -p 3.11 prison.py analyze
```

下面分别展示对于囚犯数n=50和n=100的敏感性分析结果。

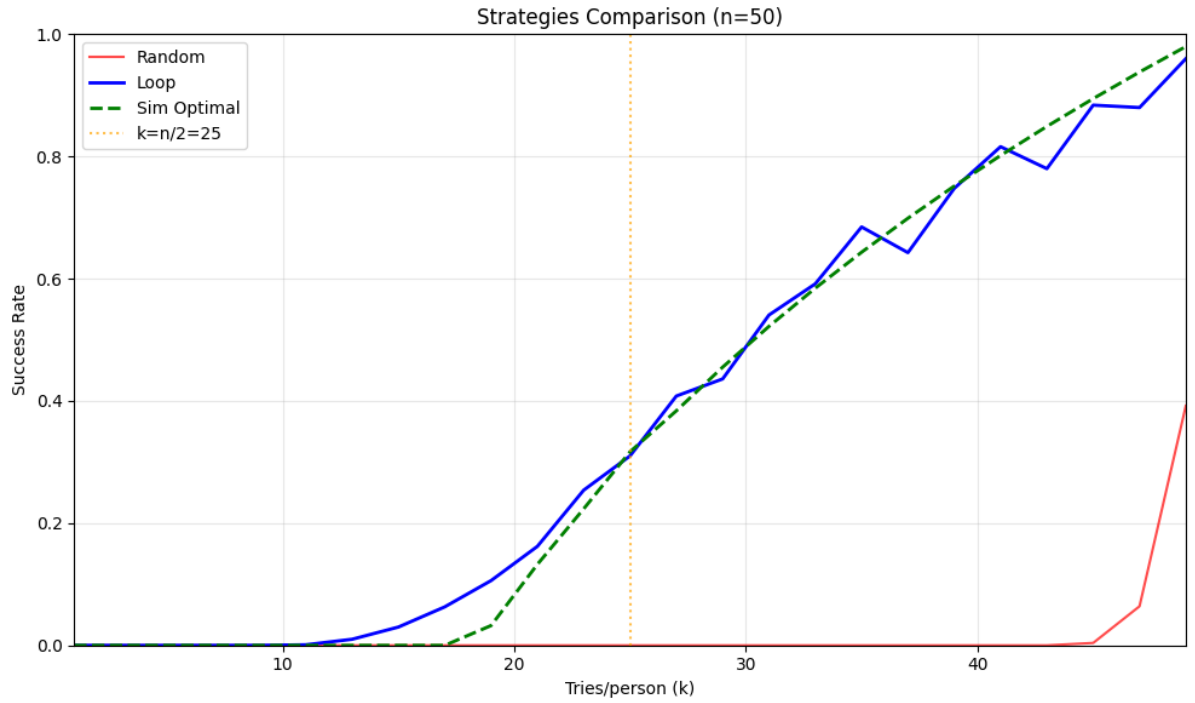


Figure 1: n=50

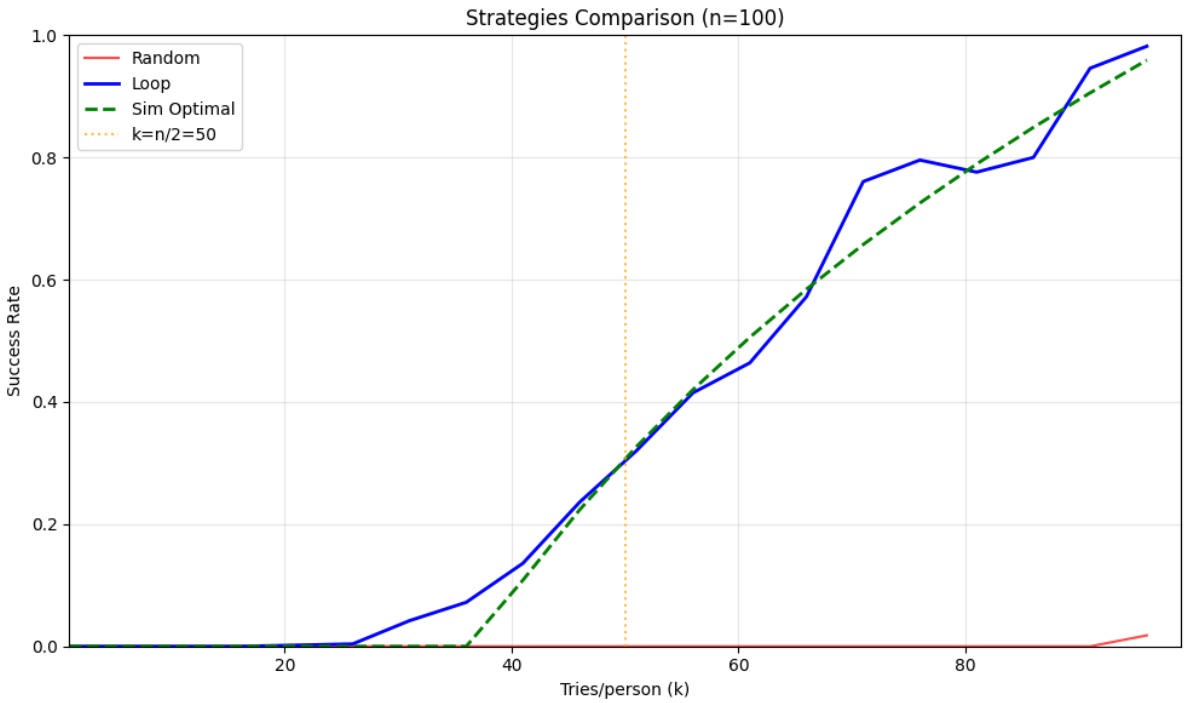


Figure 2: n=100