教务处排课问题

排课问题中没门课程都有一定的排课教师或时间要求，在一定的轮数内希望找到能使更多数量的课程达到要求的排课方案。

形式化定义：

1. 状态空间 (State Space):所有可能的课程安排组合，包括每门课程被安排在特定的教室和时间段。
2. 初始状态 (Initial State):一个初始的课程安排状态，可以是一个随机的课程安排方案。
3. 后继函数 (Actions):涉及课程安排的操作，例如将某门课程从一个教室或时间调整到另一个教室或时间。
4. 目标状态 (Goal State):一个课程安排状态，使得尽可能多的课程满足其先决条件。
5. 成本 (Cost):每次评估所有课程的先决条件是否被满足的代价为1。

搜索算法及时空代价分析：

爬山法搜索算法：

```
Function CourseScheduling_HillClimbing(MaxIterations):
    currentSchedule = RandomSchedule()  // 产生随机课程安排
    currentScore = Evaluate(currentSchedule)  // 评估当前课程安排得分
    iterations = 0

    while iterations < MaxIterations:
        neighborSchedules = GenerateNeighborSchedules(currentSchedule)  // 生成当
前课程安排的邻居安排，例如新安排和旧安排只有一节课的安排不同
        bestNeighbor = null
        bestNeighborScore = currentScore

        for each neighbor in neighborSchedules:
            neighborScore = Evaluate(neighbor)  // 评估邻居课程安排得分
            if neighborScore > bestNeighborScore:
                bestNeighbor = neighbor
                bestNeighborScore = neighborScore

        if bestNeighborScore > currentScore:
            currentSchedule = bestNeighbor
            currentScore = bestNeighborScore

        iterations += 1

    return currentSchedule  // 返回局部最优课程安排
```

- 时间代价：每次评估所有课程的先决条件是否被满足的代价为1。在每次迭代中，需要评估当前课程安排及其所有邻居安排的得分，时间代价为O(MaxIterations*d)。
- 空间代价：存储当前课程安排及其可能的邻居安排。通常只需要存储当前状态和相关信息，因此空间代价相对较低，为O(n)。

改进的算法及时空代价分析：

模拟退火算法搜索算法：

```
Function CourseScheduling_SimulatedAnnealing(MaxIterations, InitialTemperature,
CoolingRate):
    currentSchedule = RandomSchedule()  // 产生随机课程安排
    currentScore = Evaluate(currentSchedule)  // 评估当前课程安排得分
    iterations = 0
    temperature = InitialTemperature

    while iterations < MaxIterations and temperature > 0:
        neighborSchedule = RandomNeighbor(currentSchedule)  // 选择一个随机的邻居课
程安排
        neighborScore = Evaluate(neighborSchedule)  // 评估邻居课程安排得分
        deltaScore = neighborScore - currentScore

        if deltaScore > 0 or RandomUniform(0, 1) < e^(deltaScore / temperature):
            currentSchedule = neighborSchedule
            currentScore = neighborScore

        temperature = temperature * CoolingRate
        iterations += 1

    return currentSchedule  // 返回当前温度下的课程安排
```

- 时间代价：接受不良移动的概率策略有助于避免局部最优解。由于接受概率策略，相对于爬山法具有更好的鲁棒性，每次迭代需要评估当前课程安排及其某个邻居安排的得分，时间代价比爬山法小，为O(MaxIterations*d)。
- 空间代价：与爬山法类似，模拟退火算法也只需要存储当前状态和相关信息，空间代价为O(n)。