实验三 遗传算法VS回溯法

姓名： 张椿旭 **学号：**2017303024 **班级：**14011702

一．问题描述

回溯法可以处理货郎担问题，遗传算法也可以处理货郎担问题，回溯法和遗传算法哪个算法处理货郎担问题效率更高呢？在相同计算时间内，哪个算法得到的解更好呢？

实现遗传算法，通过随机产生10个不同规模的算例（城市数量分别为10，20，40，80，100，120，160，180，200，500，或者其它规模），比较上次实验实现的回溯法和遗传算法。

二．实验目的及要求

1.理解遗传算法思想，并实现遗传算法；

2.掌握不同的参数和遗传操作（选择、交叉、变异）的优缺点；

3.比较遗传算法和回溯法的执行效率，分析遗传算法和回溯法的优缺点。

三．实验分析

Travelling salesman problem(TSP) 是这样一个问题：给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路。它是组合优化中的一个NP困难问题，在运筹学和理论计算机科学中非常重要。该问题被划分为NP完全问题。已知TSP算法最坏情况下的时间复杂度随着城市数量的增多而成超多项式（可能是指数）级别增长。

1. 遗传算法实现

   遗传算法（genetic algorithm (GA) ）是计算数学中用于解决最佳化的搜索算法，是进化算法的一种。进化算法最初是借鉴了进化生物学中的一些现象而发展起来的，这些现象包括遗传、突变、自然选择以及杂交等。 遗传算法通常实现方式为一种计算机模拟。对于一个最优化问题，一定数量的候选解（称为个体）可抽象表示为染色體，使种群向更好的解进化。传统上，解用二进制表示（即0和1的串），但也可以用其他表示方法。进化从完全随机个体的种群开始，之后一代一代发生。在每一代中评价整个种群的适应度，从当前种群中随机地选择多个个体（基于它们的适应度），通过自然选择和突变产生新的生命种群，该种群在算法的下一次迭代中成为当前种群。
   具体来讲, 将每个个体当作一个问题的解, 每个个体的基因就是通过的节点号码, 如:(1, 3, 2, 4). 同时, 适应度为`\frac{1}{\sum distance(i, i+1)}`. 采用轮盘赌随机选去两个个体, 用段交叉的方法作为交叉过程. 在每次迭代后, 安一定概率进行全体的基因扰动.

2. 回溯法实现 回溯法是暴力搜尋法中的一种。对于某些计算问题而言，回溯法是一种可以找出所有（或一部分）解的一般性算法，尤其适用于约束满足问题（在解决约束满足问题时，我们逐步构造更多的候选解，并且在确定某一部分候选解不可能补全成正确解之后放弃继续搜索这个部分候选解本身及其可以拓展出的子候选解，转而测试其他的部分候选解）。 具体而言, 在本题中我们可以定义在一次回溯迭代中状态为走过的且仅一次城市数组. 那么在走过的城市数量为总数量时, 这就是一个解. 最后对所有解进行一次统计最小花费, 即可得到最优解.

四．算法伪代码或流程图

1. 遗传算法

```
iter = 0
while iter++ < iterNum:
    for i = 0->length:
        fitness[i] = computeFitness(props[i],matrix)    // 计算适应度

    p = randomDouble()
    if (p < pc): for i = 0->popNum/2:                // 随机抽取两个个体
        selected[0] = roundBet(props, fitness)
        selected[1] = roundBet(props, fitness)
        cross(selected[0], selected[1])              // 交叉

    p = randomDouble()
    if (p < pm)
        disturbance(props)                           // 全体扰动
```

2. 回溯法

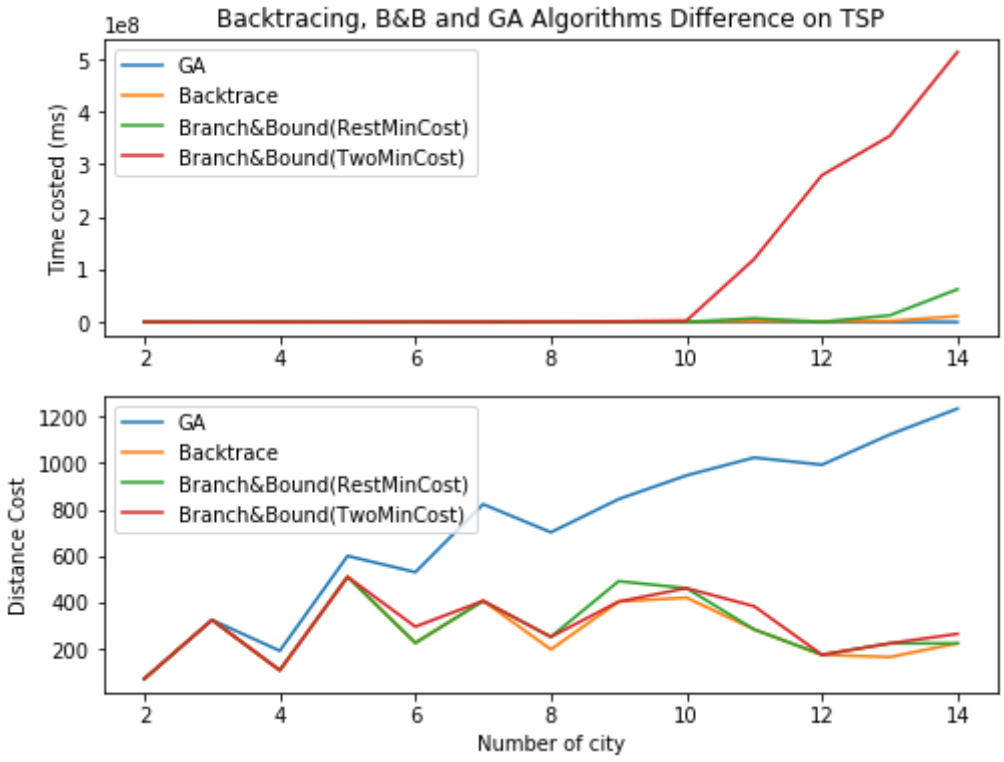
```
function backtrack(int depth, int cost, int path):
    if depth > n then
        cost += graph[path.last][1]
        if cost < minCost then
            minCost = cost
            answerPath = path + [1]
        end
    else:
        for j = i -> n:
            if check(i): continue
            if cutting(i): continue

            backtrack(depth+1, cost + graph[path.last][j])
    end
end
```

五．算法时间复杂性分析

1. 理论分析

    1. 遗传算法 一种最优化方法, 采用迭代来选择当代较优的个体, 来进行下一代的生成. 理论上的复杂度很好计算, 但是牵扯到概率问题, 所以我这里用渐进复杂度来描述, 即`O(n^2m)`
    2. 回溯法 暴力求出所有可行解, 最后统计最优解. 由于给定的图不一定是完全图, 所以有些边不存在, 那么分析时间复杂度比较困难. 但是我们假设图为完全图, 那么依据解空间树可知最坏情况下复杂度为`O(n!)`. 若带剪枝, 那么复杂度会更优秀, 但难以定量计算复杂度.
    3. 分支界限法 若不带分析上下解的界限函数和剪枝的话, 显然复杂度为`O((n-1)!log(n-1)!)`. 因为存在每次迭代都做一次优先队列的维护, 而找到一个解或最优解, 对于解空间树只需要计算n-1层即可. 若带分析上下解的界限函数和剪枝函数, 那么时间花费将大大减少, 但同样难以定量计算复杂度.

2. 实验分析 将遗传算法, 回溯法和分支界限法用不同的界限函数和问题规模, 对计算所用时间花费做度量. 问题求解所需的临界矩阵是随机生成的, 其中边不存在是有1/3的概率. 为了防止特殊的实验对结果的影响, 我对每个规模生成了25次随机数据, 对每种算法进行测试, 得到如下实验结果:



六. 问题思考与总结

1. 对于`n = 18`的问题规模, 基于分支界限的最快解法已经需要耗费分钟级别的时间. 用更大的规模进行计算是不明智的. 难以想象题目要求中对于`n = 500`规模需要几年时间来计算, 使用这种基于搜索的算法处理TSP问题是显然低效的, 同时求得的较优解是否称得上`优`. 事实上我认为对于大于20的规模, 不应该用基于搜索的算法, 而是尝试最优化算法(常见的遗传算法, 退火, 粒子群, nn等).
2. 对于基于搜索的算法而言, 最优化方法如遗传算法之类, 他们的优点在于快速. 而缺点也是十分明显, 那就是解的质量低(在`n=14`的情况下, 最差的分支界限法的结果比遗传算法好4倍). 对于特定问题, 我们应选择对应的算法解决. 这里也体现了一切从实际出发、主观与客观相符合、按照客观规律办事的哲学道理。

七. 实验中出现的问题及总结

总结:

1. 对于`n = 18`的问题规模, 基于分支界限的最快解法已经需要耗费分钟级别的时间. 用更大的规模进行计算是不明智的. 难以想象题目要求中对于`n = 500`规模需要几年时间来计算, 使用这种基于搜索的算法处理TSP问题是显然低效的, 同时求得的较优解是否称得上`优`. 事实上我认为对于大于20的规模, 不应该用基于搜索的算法, 而是尝试最优化算法(常见的遗传算法, 退火, 粒子群, nn等).
2. 对于基于搜索的算法而言, 最优化方法如遗传算法之类, 他们的优点在于快速. 而缺点也是十分明显, 那就是解的质量低(在`n=14`的情况下, 最差的分支界限法的结果比遗传算法好4倍). 对于特定问题, 我们应选择对应的算法解决. 这里也体现了一切从实际出发、主观与客观相符合、按照客观规律办事的哲学道理。