西安交通大學

货郎担问题实验报告

何宜晖
计算机46
2140504137

2016年11月

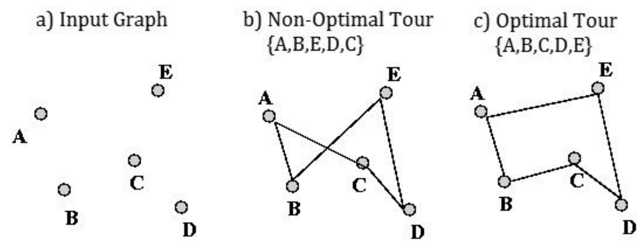

图 1:

1 实验内容

探究货郎担问题，从某个城市出发，每个城市只允许访问一次，最后又回到原来的城市，寻找一条最短距离的路径。[4]

2 实验目的

货郎担问题TSP (Traveling Salesman Problem)，又称旅行推销员或旅行商问题。由于问题本身的组合特性，其求解计算量随着城市的个数n增加而呈指数关系增长，穷举搜索法虽然能保证得到全局最优解，但面临着计算量组合爆炸的困难。

本实验探究了对于货郎担问题的不同解决方案。

3 实验原理

正确得说，货郎担问题的距离定义是广义的，可以是图表示，连接表示。这样比较复杂。在本实验中仅仅考虑几何货郎担问题，根据三角形定义如下：

$$dist(A,C) < dist(A,B) + dist(B,C) \tag{1}$$

这样的好处是，方便直观地显示出来，如图2 是美国城市的一个 货郎担问题最优解。

本实验采用学术界标准的.tsp 格式[3]来表示城市，文件第一行为城市的个数，剩下的行利用（坐标x，坐标y）的二元组来表示城市。下面给出一个例子：

```
15
-0.0000000400893815      0.0000000358808126
-28.8732862244731230     -0.0000008724121069
-79.2915791686897506     21.4033307581457670
-14.6577381710829471     43.3895496964974043
-64.7472605264735108    -21.8981713360336698
-29.0584693142401171     43.2167287683090606
-72.0785319657452987     -0.1815834632498404
-36.0366489745023770     21.6135482886620949
```

```
-50.4808382862985496     -7.3744722432402208
-50.5859026832315024     21.5881966132975371
-0.1358203773809326     28.7292896751977480
-65.0865638413727368     36.0624693073746769
-21.4983260706612533     -7.3194159498090388
-57.5687244704708050     43.2505562436354225
-43.0700258454450875    -14.5548396888330487
```

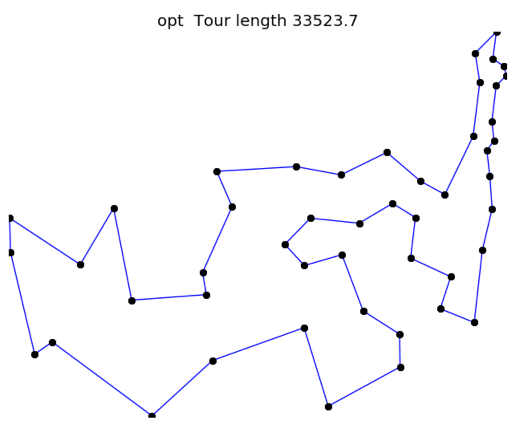

图 2: 最优路径

4 程序代码

在这里概述了主要的算法实现，具体代码请参见对应的程序文件。

4.1 随机路径

随机路径比较简单，仅仅是将顺序序列打乱，随机访问。

```
np.random.choice(n,n,replace=False)
```

4.2 贪心算法

类似于弗洛伊德算法，每加入一个城市后，考虑是否需要改变与新城市相连的节点。[2]

```
def greedy_algorithm(nodes):
    free_nodes = nodes[:]
    solution = []
```

```
    n = free_nodes[0]
    free_nodes.remove(n)
    solution.append( n )
    while len(free_nodes) > 0:
        min_l = None
        min_n = None
        for c in free_nodes:
            l = length( c, n )
            if min_l is None:
                min_l = l
                min_n = c
            elif l < min_l:
                min_l = l
                min_n = c
        solution.append(min_n)
        free_nodes.remove(min_n)
        n = min_n
    return solution
```

4.3 二优化算法

随机选取路径出现重叠的两条路径，通过消解重叠的路径，降低总路程，克服了贪心算法的缺陷。[?]

```
def optimize2opt(nodes, solution, number_of_nodes):
    best = 0
    best_move = None
    for ci in range(0, number_of_nodes):
        for xi in range(0, number_of_nodes):
            yi = (ci + 1) % number_of_nodes
            zi = (xi + 1) % number_of_nodes
            c = solution[ ci ]
            y = solution[ yi ]
            x = solution[ xi ]
            z = solution[ zi ]
            cy = length( c, y )
            xz = length( x, z )
            cx = length( c, x )
            yz = length( y, z )
            if xi != ci and xi != yi:
                gain = (cy + xz) - (cx + yz)
                if gain > best:
                    best_move = (ci,yi,xi,zi)
                    best = gain
    if best_move is not None:
        (ci,yi,xi,zi) = best_move
```

4.4 退火算法

模拟退火算法来源于固体退火原理，将固体加温至充分高，再让其徐徐冷却，加温时，固体内部粒子随温升变为无序状，内能增大，而徐徐冷却时粒子渐趋有序，在每个温度都达到平衡态，最后在常温时达到基态，内能减为最小。

有点类似二优化算法，不同之处在于，退火算法有一定随机性，有时候甚至会增加总路径（变异，取决与温度），容易逃离局部极小值。初始的温度应当足够高，才能保证系统找到足够好的解。[1]

```
def step(nodes, solution, number_of_nodes, t):
    global nn
    ci = random.randint(0, number_of_nodes-1)
    yi = (ci + 1) % number_of_nodes
    xi = random.randint(0, number_of_nodes-1)
    zi = (xi + 1) % number_of_nodes
    if xi != ci and xi != yi:
        c = solution[ci]
        y = solution[yi]
        x = solution[xi]
        z = solution[zi]
        cy = length(c, y)
        xz = length(x, z)
        cx = length(c, x)
        yz = length(y, z)
        gain = (cy + xz) - (cx + yz)
        if gain < 0:
            u = math.exp( gain / t )
        elif gain > 0.05:
            u = 1
        else:
            u = 0
        if (random.random() < u):
            nn = nn + 1
            new_solution = range(0,number_of_nodes)
            new_solution[0] = solution[ci]
            n = 1
            while xi != yi:
                new_solution[n] = solution[xi]
                n = n + 1
                xi = (xi-1)%number_of_nodes
            new_solution[n] = solution[yi]
            n = n + 1
            while zi != ci:
                new_solution[n] = solution[zi]
                n = n + 1
                zi = (zi+1)%number_of_nodes
            if anim:
                frame(nodes, new_solution, number_of_nodes, t, c, y, x,
```

```
                  z, gain)

def sa_algorithm(nodes):
    number_of_nodes = len(nodes)
    solution = [n for n in nodes]
    t = 100
    l_min = total_length( nodes, solution )
    best_solution = []
    i = 0
    while t > 0.1:
        i = i + 1
        solution = step(nodes, solution, number_of_nodes, t)
        if i >= 200:
            i = 0
            l = total_length( nodes, solution )
            t = t*0.9995
            if l_min is None:
                l_min = l
            elif l < l_min:
                l_min = l
                best_solution = solution[:]
            else:
                pass
    return best_solution
```

5 实验结果分析

在美国地图货郎担问题中，图2给出了最优解，图3给出了随机产生的路径结果，图4给出了贪婪算法的结果5，图6给出了模拟退火算法的结果。

如7所示，在小型数据集上，所有算法都差不多。然而在较大的数据集上，退火算法展示了较好的效果。

如8所示，二优化算法的时间消耗随着数据集增大，呈指数爆炸形式，而退火算法和贪心算法表现出了较高的效率。

6 实验总结

通过上面分析可以看出，货郎担问题无法被完美解决，但是在数据集较大的情况下，还是能给出不错的亚最优的结果。通过本次实验，我巩固了人工智能方面的知识，增强了对人工智能的好奇心。

最终还要感谢相明老师的耐心指导。

参考文献

[1] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference*

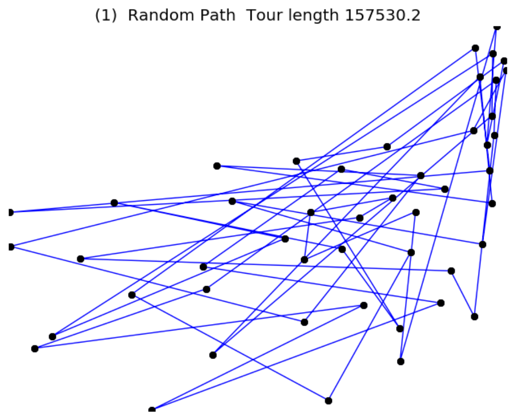

图 3: 随机路径

on Genetic Algorithms and their Applications, pages 160–168. Lawrence Erlbaum, New Jersey (160-168), 1985.

[2] G. Gutin and A. Yeo. The greedy algorithm for the symmetric tsp. *Algorithmic Operations Research*, 2(1):33–36, 2007.

[3] G. Reinelt. Tsplib, 1995. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

[4] 鲍军鹏，张选平. 人工智能导论, 2010.

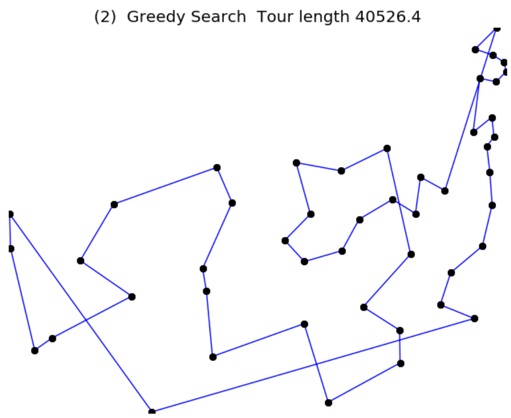

图 4: 贪心算法

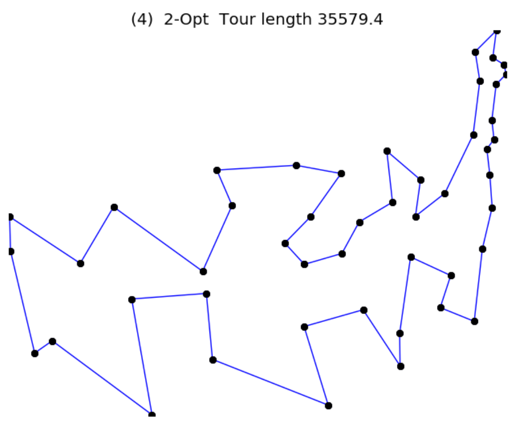

图 5: 二优化算法

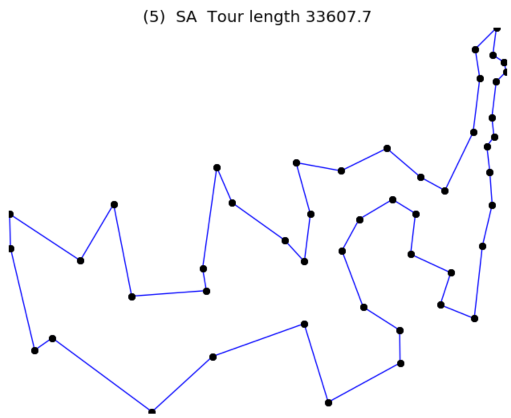

图 6: 退火算法

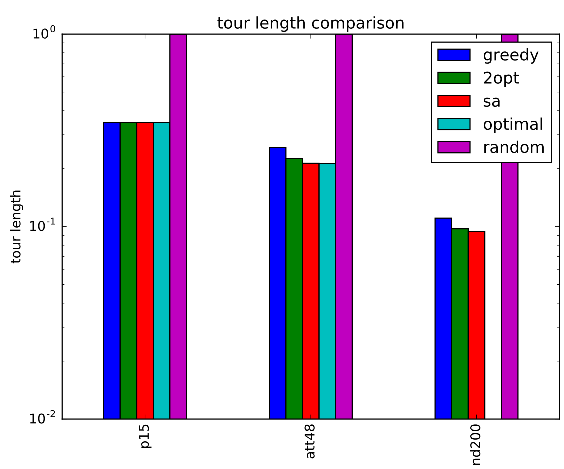

图 7: 路径长度分析

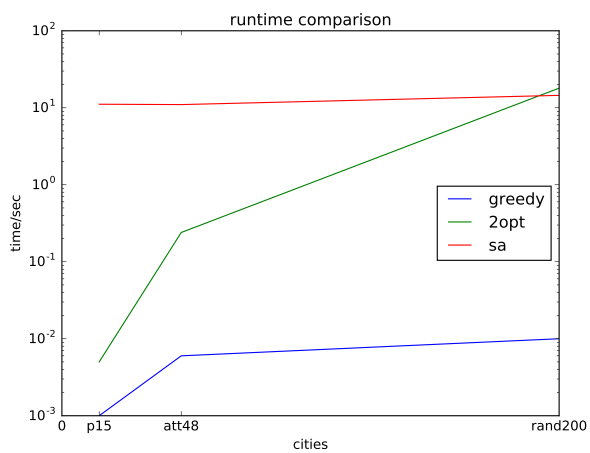
runtime comparison
time/sec
cities
greedy
2opt
sa
0
p15
att48
rand200
10^2
10^1
10^0
10^-1
10^-2
10^-3

图 8: 时间消耗分析