Weighted Traveling Salesman Problem

李尚利

【摘　要】 本文从多种算法来分析和解决 WTSP(Weighted Traveling Salesman roblem)，其中包括基于深度优先搜索的朴素方法来解决 WTSP，以及动态规划的思想以及三种不同贪心策略来从多个角度解决 WTSP 问题，并从各种方法之中得到的结果比对和分析，来总结出相关的结论。

【关键词】 WTSP，**动态规划**，**贪心策略**

1　问题简介

1.1　Weighted Traveling Salesman Problem

Consider a weighted version of the traveling salesman problem. We are given a map with a set of locations v_i, $1 \leq i \leq n$, where v_1 is the starting location. Each location v_i has a reward r_i, and each edge e_{ij} between v_i and v_j has a cost c_{ij}. We wish to find a path starting from v_1, does not visit any vertex twice, and with total cost less than some given C, such that the total reward on the visited vertices are maximized.

考虑一个加权版本的旅行商问题。给定一个带有多个地点 v_i 的图，$1 \leq i \leq n$, 其中 v_1 为起点。对于每一个地点 v_i 而言都有一个回报 r_i, 图中的每条从 v_i 到达 v_j 的边 e_{ij} 都有一个 c_{ij} 的代价。我们希望发现一个从 v_1 出发的路径不会访问任意结点两次，且能在总代价小于所给的 C 的情况下使得访问总回报最大化。

2　问题分析

2.1　朴素算法: 基于深度优先搜索的求解方法

对于 WTSP 问题，一种比较容易得出的求解思路便是采用深度优先搜索来遍历所有的可能路径情况，一一比对这些得到路径代价和 C 的大小来判断这些可能路径是否为允许的答案。显然，对于这种方法的时间复杂度:$T(n) = (n-1)T(n-1) = ... = n!$.

Algorithm 1 基于深度优先搜索的方法 DFS

Require: $1 \leq location \leq N$
Ensure: $maxr$
  if $n == ct\ and\ reward + r[1] > maxrcost + graph[location][1] < C$ **then**
    $maxr = reward + r[1]$
    $maxc = cost + graph[location][1]$
    $return$
  end if
  for $i \leq totalNum$ **do**
    if $vis[i] == 0$ **then**
      $vis[i] = 1$
      $DFS(i)$
      $vis[i] = 0$
    end if
  end for

对于 Algorithm1 可以计算没有代价限制 C 下的路径最大 reward，但在存在约束限制 C 下，实际需要对访问的总结点进行取舍，在满足限制代价 C 下尽可能访问总回报最多的方案。因此需要对给定的限制 C 来寻求访问最多的结点数。一种简单的方法便是来直接按访问结点数从大到小遍历得到的最短路径和 C 的大小关系来确定结点数，即若该结点数 n 下的最短路径小于约束 C，那么说明在存在访问 n 个结点的最大回报能够满足题意。

实验时间: 2022-11-15
报告时间: 2022-11-17
† **指导教师**
***学号**：2022E8015082042
*E-mail：lishangli22@mails.ucas.ac.cn

Algorithm 2 求解最大结点数下的 WTSP

Require: $C; totalNum$
  for $i = totalNum; i \geq 1; i--$ **do**
    $maxc = DFS(i)$
    if $maxc < C$ **then**
      $print\ result$
      $break$
    end if
  end for

2.2 基于动态规划的求解方法

首先，对于采用动态规划解决 WTSP 问题需要对动态规划求解问题的可行性进行证明。这里不妨假设 s,s1,s2,...,sp,s 为一条满足小于路径长度 C 且 Reward 总和最大的简单路径，假设当前 s 到达下一个城市 s1 是已经求出的，那么该问题便是需要求一条 s1 到 s 的满足小于路径长度 $C - s_1 s$ 且 Reward+r[1] 最大，显然对于 $s_1, s_2, ..., s_p, s$ 一定是构成上述约束条件的解，因此对于 WTSP 问题是符合最优子结构的，所以采用动态规划思想处理该问题也是合理的。

对于 TSP 的动态规划处理方法，从数学方面进行处理可以简单表示为 $D[location, V]$。其含义为从地点 location 出发，经过点集 V 的最短路径，其中这里的 V 表示为未访问的地点集合。根据上述目标问题的表示，不难对于子问题也同样进行表示未 $D_{i,V}$。根据上述问题表示，可以得到递推关系为

$$\begin{aligned} D_{i,V} &= C_{ik} + D_{k,V-\{k\}} \\ D_{i,\emptyset} &= C_{i,location} \end{aligned} \tag{1}$$

而对于 WTSP 问题也就是在 TSP 问题的基础上增加了路径代价约束进行限制，并且 WTSP 追求的并非最短路径，而是满足路径代价 C 下的最大总回报。因此，递推关系除了需要满足上述的代价关系 $D_{i,V} < C$ 之外，需要额外满足回报递推关系式2

$$R_{i,V} = \max_k (R_{k,V-k} + r[k]) \tag{2}$$

其完整的算法流程见伪代码示意算法3

Algorithm 3 基于动态规划的解决方法

Require: 访问结点数 n; 起点 location; 代价限制 C
Ensure: 总回报 MaxReward; 路径代价和路径
  初始化集合 D、R
  for 对每一个起点 i 求解 **do**
    for 每一个未访问顶点集 V 求解最大回报 **do**
      if 顶点集不包含 i **then**
        for 遍历所有未访问顶点 V **do**
          if $R_{i,V} \leq r_k + R_{k,V-k}\ and\ c_{ik} + D_{k,V-k} \leq C$ **then**
            $R_{i,V} = r_k + R_{k,v-K}$
          end if
          if $D_{i,V} \leq c_{ik} + D_{k,V-k}\ and\ c_{ik} + D_{k,V-k} \leq C$ **then**
            $D_{i,V} = c_{ik} + D_{k,V-k}$
          end if
        end for
      end if
    end for
  end for
  $print\ result$

2.3 基于贪心思想的求解算法

贪心算法 (greedy algorithm) 一般指对问题求解时总是做出当前看来为最好的选择，很显然这种只寻求短期局部最大化收益的思路并非会在全局上得到最大化收益。而针对于不同的最大化收益，也可以对于 WTSP 有多种贪心算法进行求解。本文仅就对局部最近贪心、局部最大收益以及局部收益/距离比率等三种贪心思路来处理 WTSP。

对于算法4的局部最近优先贪心算法而言，只需要在更新条件将距当前地点 now 最近距离作为最优决策进行更新修改为未访问结点中加入最大回报便可以作为局部最大回报贪心算法，或者将此处修改为最大回报/距离作为最优决策进行更新可以作为局部回报/距离比率贪心算法。此处不再赘述，细节见代码main.cpp。

3 算法分析

3.1 时间复杂度

就针对上述的几种算法，现就针对它们的时间复杂度进行分析。首先对于暴力搜索遍历所有

Algorithm 4 局部最近贪心算法

Require: 路径限制 C; 起点 location;
Ensure: 最大回报；路径代价和路径

```
初始化参数 flag,mins,now,mini,reward,cost,v[N]
while flag!=0 do
    flag = 0;mins = N
    for 遍历未访问结点 do
        flag=1
        if c[i][now] ≤ mins then
            mins = c[i][now] mini = i
        end if
    end for
    if flag == 0 then
        break
    end if
    if cost+c[mini][now]+c[mini][1]≥C then
        v[mini] = 1; continue
    end if
    cost+ = c[mini][now]
    reward+ = r[mini]
    v[mini] = 1; now = mini
end while
打印结果
```

情况的朴素算法，不难分析得知:

$$\begin{aligned} T(n) &= (n-1)T(n-1) \\ T(1) &= O(1) \end{aligned} \tag{3}$$

上述递归式不难求出 $T(n) = O(n!)$.

而对于动态规划的算法而言，由于其不仅需要遍历所有状态 $(n * 2^n)$，且在每种状态更新时需要遍历所有未访问点集合 V 寻求最优决策，因此其时间复杂度 $T(n) = O(n^2 2^n)$.

而对于三种贪心策略而言，由于仅仅是在选择最优指标的区别，因此其时间消耗应该一致，不难分析伪代码有时间复杂度 $T(n) = \sum_{k=1}^{n} k = O(n^2)$。

针对实际 n=10 的情况实际运行上述算法代码得到的时间消耗有

3.2 结果分析

采用随机数随机生成得到 N=13 的地点图1。借助第二节的算法来处理该图的 WTSP 问题的解。从上文的算法分析可以得知，当容量限制最大时，也就是说没有限制时，得到的最大回报应该是所有的结点都能被访问得到最大回报。实际上在本例在取 C>342 以上便可以理解为 C 没有限制。而没有

Algorithm	RunTime(ms)		
	n=10	n=12	n=13
DFS	26.641701	2936.688477	37264.01
DP	6.996800	24.402399	52.368198
Nearest Greedy	0.002600	0.002800	0.002700
Reward Greedy	0.002200	0.002700	0.002700
Ratio Greedy	0.002500	0.002900	0.003200

表 1 各算法时间消耗

得到限制下的情况下所有算法的最大回报值应该都是 720。由于算法的差异，得到的最终路径解不一样，其中动态规划为 $s_1 \to s_7 \to s_8 \to s_9 \to s_4 \to s_3 \to s_{12} \to s_5 \to s_2 \to s_{10} \to s_6 \to s_{13} \to s_{11} \to s_1$. 朴素算法和其他的三种贪心算法这里不再赘述。其中动态规划得到的最小代价为 342，朴素算法为 681，而贪心算法依次为 362，589，405. 实际上，要访问到所有的 13 个结点，则需要 C>342, 因为求解该例的 TSP 问题得到的最小路径代价为 342，也就是说 C>342 一定能找到访问所有结点使得回报最大。

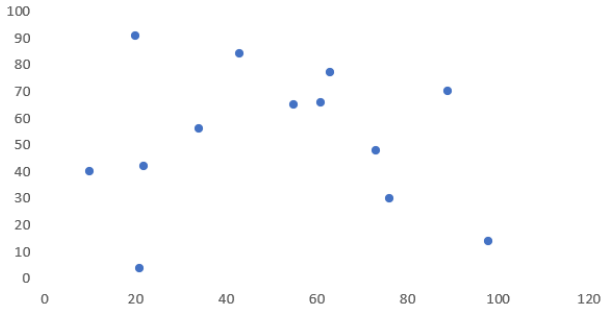

图 1 生成地点图

反过来说，当 C=341 时，则一定找不到访问到所有结点而路径代价还 <C 的路径解。这种情况下朴素算法和 DP 得到的能够访问最大结点数 N=12，此时得到的最大回报为 709，对应的代价为 339，而针对于 N=12 的 TSP 的最短路径代价为 298. 而此时对于贪心算法得到的解便不是最优解，而是符合条件的一个解。

实际上，对于 N=10 个结点的时候回报贪婪在 C=341 在符合存在最优解时并没有寻找到全局最优解，而另外两者贪婪算法是得到最优解. 而当 C=272 时，此时只存在访问到 9 个结点的最优解时，朴素算法和动态规划得到最优解，而其他三种算法无法得到最优解。这与 13 个结点的图结论一

致。

显然，朴素算法的时间消耗是最为巨大的，仅在 N=13 的时候便需要花费 37s 之久。不难得出，当 N 大于 13 后每增加一个，时间便会剧增。动态规划在 N=20 时，花费 14s, 当 N=25 时便进程被系统 kill. 而贪心算法在 N=100 时也只花费了 0.01ms 左右。而当 N=10000 时，实际不难通过算法复杂度得到时间将要到达 100ms 的量级，显然此时约束算法运行的限制并非时间，而是系统内存空间将会限制到贪心算法正常运行。

从这里不难看出，朴素算法的编程难度较低且能够得到最优解，但是消耗时间在 N=13 时便需要花费将近一 40s。而对于动态规划，编程难度较其他算法较大，但能够在一个较朴素算法而言的短时间得到 WTSP 的最优解，而当 N 逐渐增大也会逐渐产生指数爆炸。而贪心算法编程难度最低，且速度相比上述算法更快，所需内存空间也最小，但是它无法保证能够一定能够得到最优解。

4 结语与展望

本文通过对 WTSP 多种解决算法的实现来探讨 WTSP 问题中不同算法之间的优劣和算法特点。通过算法实现和代码运行实例进行比较得到相关数据，进而分析得到朴素算法和动态规划算法可以得到最优解，但实际情况下由于受时间限制无法处理 WTSP 这一类的 NP 问题，而贪心算法仅仅寻求短时间的一个可用解，无法保证得到的解为最优解。

WTSP 问题作为一个 NP 完全问题，像 DFS 和 DP 这种穷举算法的效率不高，可以通过一些多项式时间复杂度的随机启发式来逼近最优解。就目前而言，主流的解决 TSP 问题的方法主要有遗传算法、蚁群算法、模拟退火算法、粒子群算法等达到这一目标。未来也可以通过实现这些算法来解决 WTSP 问题来分析 WTSP 问题的一般高效的解决思路。