第3章 旅行商问题优化
启发式算法及Matlab编程实现

TSP问题及其优化方法

1. • 最近邻算法及Matlab编程实现
2. • 两点互换算法及Matlab编程实现
3. • 两边三边互换算法及Matlab编程实现

1 最近邻法Nearest Neighbor，NN

NN：顾名思义，最近邻法是从城市1开始，查找同其距离最近的城市作为旅行商的第二个到达城市，再查找同第二个到达城市最近的且不在已经过路径中的城市作为第三个到达城市，依次类推，直至经过全部城市后返回城市1。

id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	47	67	48	25	74	28	91	52	71	24	78	23	29	13
2	47	0	86	83	42	45	31	76	44	71	71	78	67	75	59
3	67	86	0	33	48	72	57	57	46	30	73	32	80	64	60
4	48	83	33	0	42	87	53	84	58	56	44	61	53	34	36
5	25	42	48	42	0	53	11	65	28	47	44	54	46	42	27
6	74	45	72	87	53	0	46	35	29	45	96	49	97	96	80
7	28	31	57	53	11	46	0	64	26	49	51	57	51	51	35
8	91	76	57	84	66	35	64	0	39	28	109	26	112	104	92
9	52	44	46	58	28	29	26	39	0	27	72	34	74	69	55
10	71	71	30	56	47	45	49	28	27	0	85	8	90	79	69
11	24	71	73	44	44	96	51	109	72	85	0	92	9	11	17
12	78	78	32	61	54	49	37	26	34	8	92	0	97	86	76
13	23	67	80	53	46	97	51	112	74	90	9	97	0	20	21
14	29	75	64	34	42	96	51	104	69	79	11	86	20	0	16
15	13	59	60	36	27	80	35	92	55	69	17	76	21	16	0

路径：1-[13]-15-[16]-14-[11]-11-[9]-13-[46]-5-[11]-7-[26]-9-[27]-10-[8]-12-[26]-8-[35]-6-[45]-2-[83]-4-[33]-3-[67]-1

距离：13+16+11+9+46+11+26+27+ 8+ 26+ 35+ 45+83+33+67 =456

1 最近邻法Nearest Neighbor，NN

NN：顾名思义，最近邻法是从城市1开始，查找同其距离最近的城市作为旅行商的第二个到达城市，再查找同第二个到达城市最近的且不在已经过路径中的城市作为第三个到达城市，依次类推，直至经过全部城市后返回城市1。

算法流程如下：

Step1：路径集合 P={1}，邻域 N={2,3,...,n}，i=1；

Step2：从邻域 N 中查找具有最小 d_{ij} 的城市为 i_0，并令 $P = P \cup \{i_0\}$，$N = N - \{i_0\}$；

Step3：判断 $N = \varnothing$ 是否成立，若成立则停止算法并输出结果 P，否则令 $i = i_0$，重复 Step2。

思考题

- 从1开始使用NN
- 从6开始使用NN
- 结果有何异同？为什么？

	1	2	3	4	5	6	7	8	9
1		18	48	40	46	31	30	16	41
2	18		36	35	45	9	18	24	44
3	48	36		30	49	36	49	42	27
4	40	35	30		42	21	31	22	53
5	46	45	49	42		31	29	54	30
6	31	9	36	21	31		22	48	36
7	30	18	49	31	29	22		9	16
8	16	24	42	22	54	48	9		30
9	41	44	27	53	30	36	16	30	

	1	2	3	4	5	6	7	8	9
1		18	48	40	46	31	30	16	41
2	18		36	35	45	9	18	24	44
3	48	36		30	49	36	49	42	27
4	40	35	30		42	21	31	22	53
5	46	45	49	42		31	29	54	30
6	31	9	36	21	31		22	48	36
7	30	18	49	31	29	22		9	16
8	16	24	42	22	54	48	9		30
9	41	44	27	53	30	36	16	30	

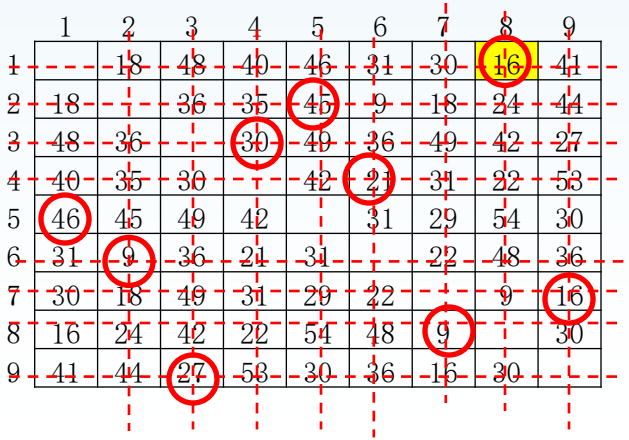
思考题

- 从1开始使用NN
- 从4开始使用NN
- 结果有何异同？为什么？

	1	2	3	4	5	6	7	8	9
1		18	48	40	46	31	30	16	41
2	18		36	35	45	9	18	24	44
3	48	36		30	49	36	49	42	27
4	40	35	30		42	21	31	22	53
5	46	45	49	42		31	29	54	30
6	31	9	36	21	31		22	48	36
7	30	18	49	31	29	22		9	16
8	16	24	42	22	54	48	9		30
9	41	44	27	53	30	36	16	30	

路径： 4 -> 6 -> 2 -> 1 -> 8 -> 7 -> 9 -> 3 -> 5 -> 4

距离： 21 + 9 + 18 + 16 + 9 + 16 + 27 + 49 + 42 = 207



路径：1 -> 8 -> 7 -> 9 -> 3 -> 4 -> 6 -> 2 -> 5 -> 1

距离：16 + 9 + 16 + 27 + 30 + 21 + 9 + 45 + 46 = 219

1 最近邻法NN Matlab编程实现

```
%根据传入的节点距离矩阵，使用最近邻法获得路径及其总里程
function [routes,sumDist]=myNearNeighborTSP(dists)
  nodeQty = size(dists,1);
  %算法实现
  routes=1;%使用节点1作为路径的起点
  neighbors=2:nodeQty;
  i=1;%当前所处节点编号
  sumDist=0;%用于累计路径里程长度
  while size(neighbors,2)>0
    %找出第i个节点与全部剩余邻居的距离
    neighborQty = size(neighbors,2);
    midDists =zeros(1,neighborQty);%定义当前节点i同邻居距离的单行数组
    for j=1:neighborQty%挨个将i同邻居的距离写入数组midDists
      myId =i;
      neighborId = neighbors(j);
      midDists(j)=dists(myId,neighborId);
    end
    %找出最近邻居的id
    [~,idx]=sort(midDists);%进行数值升序排序
    nearNeighborId =neighbors( idx(1));
    %将最近邻居加入路径，同时从剩余邻居中删除
    routes=[routes nearNeighborId];
    neighbors(idx(1))=[];
    sumDist = sumDist + dists(i,nearNeighborId);
    %将找到的最近邻居作为新的起点
    i = nearNeighborId;
  end
  sumDist =sumDist + dists(routes(nodeQty),1);
  %结果输出
end
```

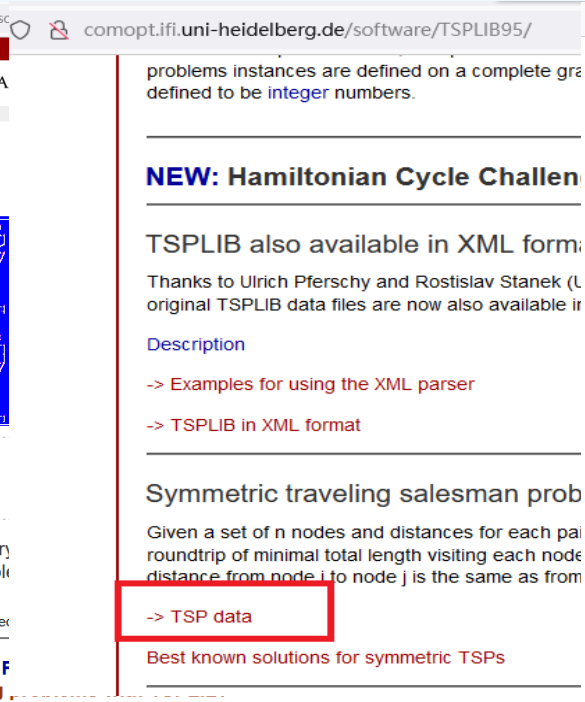
optTSP2024.m

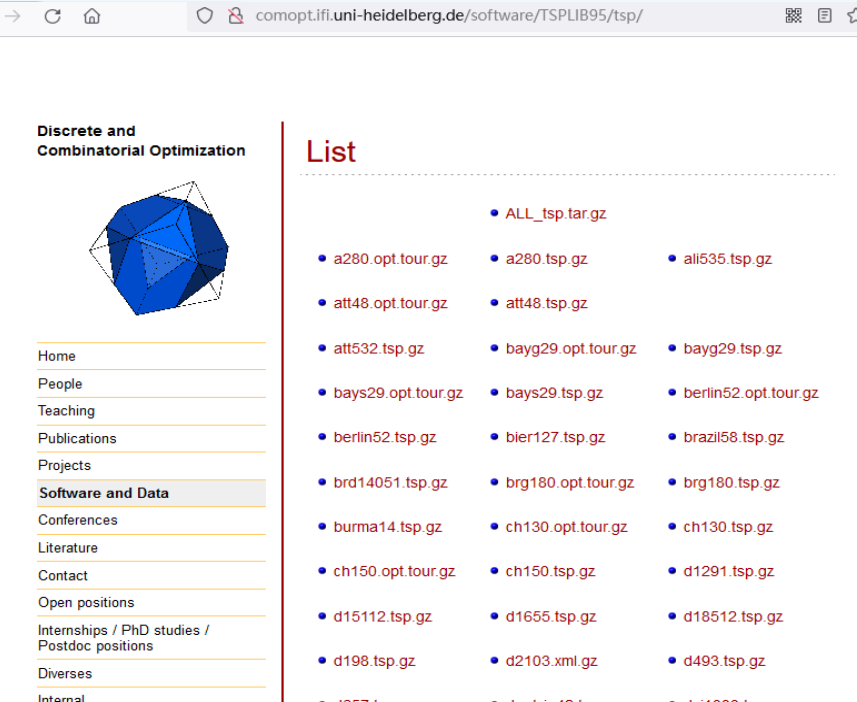
```
function optTSP2024
    [routes,sumDist]=myNearNeighborTSP(initialDist_att48);
    disp(routes);
    disp(sumDist);
end
%使用att48算例数据作为实验数据
function dist_att48 = initialDist_att48
    att48=[1 6734 1453;2 2233 10;3 5530 1424;4 401 841;5 3082 1644;
6 7608 4458;7 7573 3716;8 7265 1268;9 6898 1885;10 1112 2049;
11 5468 2606;12 5989 2873;13 4706 2674;14 4612 2035;15 6347 2683;
16 6107 669;17 7611 5184;18 7462 3590;19 7732 4723;20 5900 3561;
21 4483 3369;22 6101 1110;23 5199 2182;24 1633 2809;25 4307 2322;
26 675 1006;27 7555 4819;28 7541 3981;29 3177 756;30 7352 4506;
31 7545 2801;32 3245 3305;33 6426 3173;34 4608 1198;35 23 2216;
36 7248 3779;37 7762 4595;38 7392 2244;39 3484 2829;40 6271 2135;
41 4985 140;42 1916 1569;43 7280 4899;44 7509 3239;45 10 2676;
46 6807 2993;47 5185 3258;48 3023 1942];
    dist_att48=dist(att48(:,2:3)');%dist根据两行多列数组计算距离矩阵
end

%根据传入的节点距离矩阵，使用最近邻法获得路径及其总里程
function [routes,sumDist]=myNearNeighborTSP(dists)...
```

练习题

1. 完成TSP的NN算法的Matlab程序；
2. 实现从不同起点出发的TSP的NN算法Matlab编程；
3. 使用大规模TSP算法测试Cplex和最近邻法的求解效率





TSP问题及其优化方法

1. 最近邻算法及Matlab编程实现
2. 两点互换算法及Matlab编程实现
3. 两边三边互换算法及Matlab编程实现

2 两点互换算法2opt

- **2opt:** 两点互换算法属于一种改进式邻域搜索算法。邻域搜索算法的基本思想是采用某种方法先产生一个可行解，甚至随机产生一个可行解S，然后通过对当前可行解的邻居N中搜索目标函数值优于S的解S'，并令S'替换S，继续对其邻居进行搜索以改善解的目标函数值，直至某些条件成立停止算法搜索，输出最优的结果。

2 两点互换算法2opt

- **如何生成两点互换的邻域【找邻居】？**
  - 进行全部点对的互换，例如仅有5个节点的TSP问题，去除首位和末尾固定节点1，其他四个位置可以形成两两点对{1-2、1-3、1-4、2-3、2-4、3-4}，
  - 例如，若初始解为1-4-3-2-5-1，去除首位和末尾的节点1，则其他节点的顺序为：4-3-2-5，基于位置点对进行第一次互换，可以生成邻域解：
    - 1-3-4-2-5-1, 1-2-3-4-5-1, 1-5-3-2-4-1, 1-4-2-3-5-1, 1-4-5-2-3-1, 1-4-3-5-2-1
  - 若获得最好的解为1-5-3-2-4-1,则进行第二次互换，可以生成邻域解：
    - 1-3-5-2-4-1, 1-2-3-5-4-1, 1-4-3-2-5-1, 1-5-2-3-4-1, 1-5-4-2-3-1,1-5-3-4-2-1
  - 可以看出虽然点对组合相同，但是不同的解使用相同的2opt方法获得邻域解基本不相同，因而可以进行较大范围的搜索，以改善解的质量。
- **对于有n个节点的tsp问题，如果起始点固定为节点1，则点对有多少个？**
  - **pairQty = (n-1)*(n-2)/2** **<<< (n-1)! 遍历算法**

2 两点互换算法2opt

- 如何节约计算工作量？
- 只比较邻居间变化部分的目标函数值

id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	47	67	48	25	74	28	91	52	71	24	78	23	29	13
2	47	0	86	83	42	45	31	76	44	71	71	78	67	75	59
3	67	86	0	33	48	72	57	57	46	30	73	32	80	64	60
4	48	83	33	0	42	87	53	84	58	56	44	61	53	34	36
5	25	42	48	42	0	53	11	66	28	47	44	54	46	42	27
6	74	45	72	87	53	0	46	35	29	45	96	49	97	96	80
7	28	31	57	53	11	46	0	64	26	49	51	57	51	51	35
8	91	76	57	84	66	35	64	0	39	28	109	26	112	104	92
9	52	44	46	58	28	29	26	39	0	27	72	34	74	69	55
10	71	71	30	56	47	45	49	28	27	0	85	8	90	79	69
11	24	71	73	44	44	96	51	109	72	85	0	92	9	11	17
12	78	78	32	61	54	49	57	26	34	8	92	0	97	86	76
13	23	67	80	53	46	97	51	112	74	90	9	97	0	20	21
14	29	75	64	34	42	96	51	104	69	79	11	86	20	0	16
15	13	59	60	36	27	80	35	92	55	69	17	76	21	16	0

路径S：1-[13]-15-[16]-14-[11]-11-[9]-13-[46]-5-[11]-7-[26]-9-[27]-10-[8]-12-[26]-8-[35]-6-[45]-2-[83]-4-[33]-3-[67]-1

路径S1：1-[29]-14-[16]-15-[17]-11-[9]-13-[46]-5-[11]-7-[26]-9-[27]-10-[8]-12-[26]-8-[35]-6-[45]-2-[83]-4-[33]-3-[67]-1，总里程变化量：29+16+17-[13+16+11]=22×

路径S：1-[13]-15-[16]-14-[11]-11-[9]-13-[46]-5-[11]-7-[26]-9-[27]-10-[8]-12-[26]-8-[35]-6-[45]-2-[86]-3-[33]-4-[48]-1，总里程变化量：86+33+48-[83+33+67]=-16√

2 两点互换算法2opt

- 算法流程【迭代贪婪算法的通用流程】

Step1:首先随机生成一个可行解xnew，并令f(xbest)=f(xnew),记录xbest=xnew

Step2：根据一定规则【两点互换】产生xnew的全部邻域解集N（xnew）

Step3：计算N（xnew）中每个解的值，将这些解中的最优解设定为f(xnow)

Step4：如果f(xnow)与f(xbest)相同，则停止搜索，执行Step5；否则，若f(xnow)优于f(xbest)，则令f(xbest)=f(xnow)，令xbest=xnow，xnew=xnow，并将xnew的解向前随机移动一些位数，增加搜索能力，重复step2

Step5：输出最优解xbest

2 两点互换算法2opt Matlab编程实现

myIterativeTSP.m

```
%迭代贪婪算法求解TSP问题
function optRoute=myIterativeTSP(dists)
  %生成初始解-随机生成
  cityQty = size(dists,1);
  disp('随机初始的路径');  route = randperm(cityQty);
  nowFit = calDist(route,dists);
  optFit =nowFit;  optRoute=route;%记录最优目标函数值和路径
  improved =1;
  while(improved==1)
     disp('while loop route...');      improved=0;
     pairs = combnk(1:cityQty,2);%按照n中取2生成点对
     neighborQty = size(pairs,1);
     neighborRoutes=zeros(neighborQty,cityQty);
     for i=1:neighborQty...
    % neighborRoutes
     %计算每个邻居的目标函数值
     fitnesses = zeros(1,neighborQty);
     for i=1:neighborQty...
     %从邻居中找到最短路的解
     [~,idx]=sort(fitnesses);
     goodNBId=idx(1);%最好邻居的索引
     goodRoute = neighborRoutes(goodNBId,:);
     goodFit = fitnesses(goodNBId);
     if goodFit<=optFit...
  end
  optFit;
end
%根据节点距离矩阵和路径排序获得该路径的总里程
%route是全部节点的乱序
function sumDist=calDist(route,dists)...
```

```
    for i=1:neighborQty
      %依次互换当前解两个位置获得新解
      p1 = pairs(i,1);p2=pairs(i,2);
      p1Val = route(p1);p2Val = route(p2);
      neighborRoutes(i,:)=route;
      neighborRoutes(i,p1)=p2Val;
      neighborRoutes(i,p2)=p1Val;
    end
   % neighborRoutes
   %计算每个邻居的目标函数值
   fitnesses = zeros(1,neighborQty);
   for i=1:neighborQty
      fitnesses(i)=calDist(neighborRoutes(i,:),dists);
   end
```

```
%根据节点距离矩阵和路径排序获得该路径的总里程
%route是全部节点的乱序
function sumDist=calDist(route,dists)
  sumDist =0;
  cityQty = size(dists,1);
  for i=1:cityQty-1
     preId = route(i);
     rearId = route(i+1);
     sumDist = sumDist + dists(preId,rearId);
  end
  sumDist = sumDist+dists(route(cityQty),route(1));
end
```

2 两点互换算法2opt Matlab编程实现

- 调用主程序

myIterativeTSP.m | tspTest2023.m

```
%用于测试tsp算法的主程序
function tspTest2023()
 %问题数据设定
    %初始数据
    nodeQty =20;
    %rng(5);
    dists =ceil(10+190*rand(nodeQty,nodeQty)); %节点间距离均匀分布在【10,200】
    for i=1:nodeQty
        dists(i,i)=0;
    end
 %调用不同算法
 route1 = myIterativeTSP(dists)
 route2=myNearNeighborTSP(dists)
 %比较算法结果
 disp('迭代贪婪算法结果：');
 calDist(route1,dists)
 disp('最近邻算法结果：');
 calDist(route2,dists)
end
%扩大问题规模，增加城市节点数量，比较两种算法结果

%根据节点距离矩阵和路径排序获得该路径的总里程
%route是全部节点的乱序
function sumDist=calDist(route,dists)
  sumDist =0;
  cityQty = size(dists,1);
  for i=1:cityQty-1
      preId = route(i);
      rearId = route(i+1);
      sumDist = sumDist + dists(preId,rearId);
  end
  sumDist = sumDist+dists(route(cityQty),route(1));
end
```

思考题

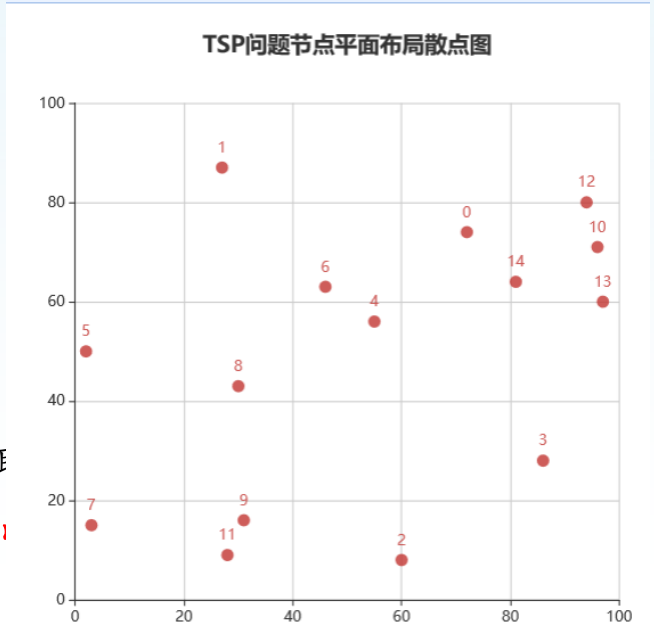
1. 完成TSP的2opt算法Matlab程序；
2. 使用大规模TSP算法测试Cplex、NN和2opt算法的求解效率

TSP问题及其优化方法

1. • 最近邻算法及Matlab编程实现
2. • 两点互换算法及Matlab编程实现
3. • 两边三边互换算法及Matlab编程实现

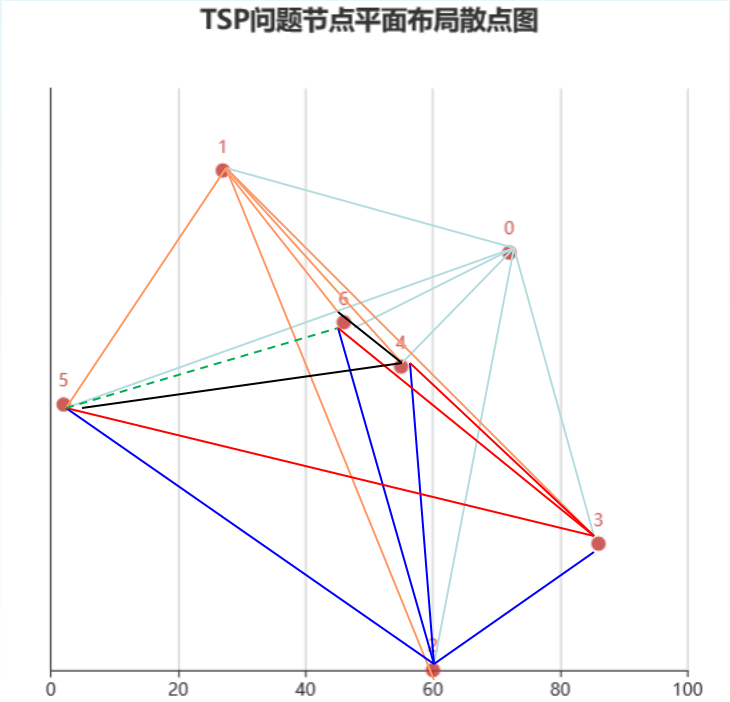
两边互换算法Lin2opt – 边集基础

- Lin2opt：既然TSP问题中点互换能够改善解的质量，那么边的互换应该也能够改善解的质量，Lin在1965年给出了边互换算法的雏形，并于1973年同Kernighan对其算法进行详细设计和算例实验，发现按照这种思路进行TSP问题优化求解的质量要好于之前的一些启发式算法，后续将该类算法使用LK缩写代替。
- 思考题：
  - 对于有n个节点的对称TSP问题，
    - 总共有多少条边？
      - n*(n-1)/2
    - 它的一个可行解有多少条边？
      - n
    - 获得可行解的过程就是从其全部n*(n-1)/2条边中获取
    - 获取最优解的过程就是从其全部可行解中找出目标



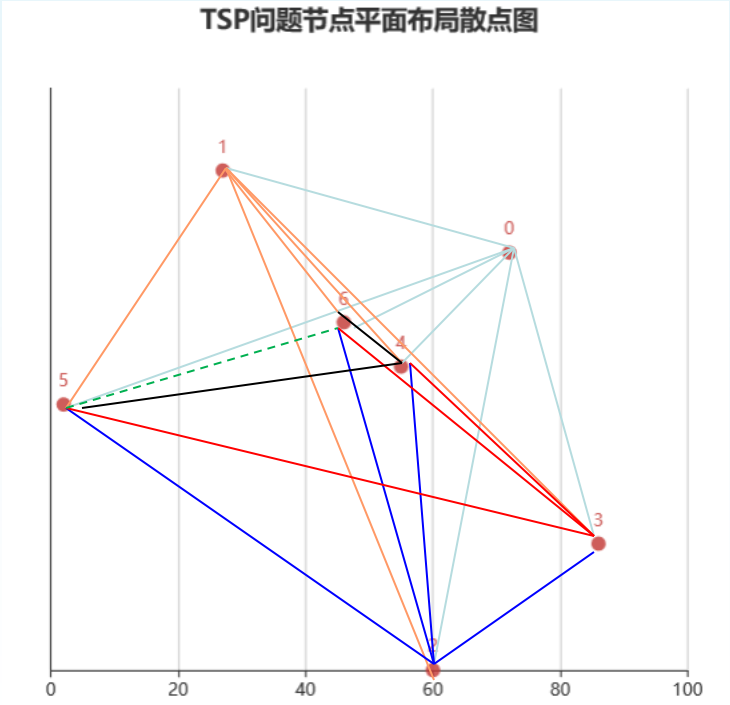
两边互换算法Lin2opt – 边集基础

- 思考题：
  - 以前述示例中前7个点为例来列举其边和可行解
    - 全部边：数量=7*6/2=21
      - 0-1,0-2,0-3,0-4,0-5,0-6
      - 1-2,1-3,1-4,1-5,1-6
      - 2-3,2-4,2-5,2-6
      - 3-4,3-5,3-6
      - 4-5,4-6
      - 5-6
    - 可行解：数量为多少？



两边互换算法Lin2opt – 边集基础

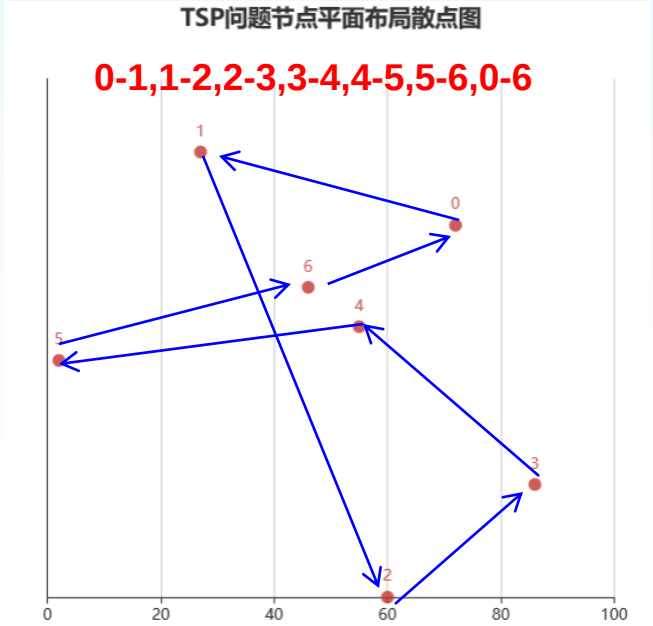
- 思考题：
  - 以前述示例中前7个点为例来列举其边和可行解
    - 全部边：数量=7*6/2=21
      - 0-1,0-2,0-3,0-4,0-5,0-6 【*n*-1中取1】
      - 1-2,1-3,1-4,1-5,1-6 【*n*-2中取1】
      - 2-3,2-4,2-5,2-6 【*n*-3中取1】
      - 3-4,3-5,3-6 【*n*-4中取1】
      - 4-5,4-6 【*n*-5中取1】
      - 5-6 【*n*-6中取1】
    - 可行解：数量为多少？
      - =(n-1)*(n-2)*(n-3)*…*2*1 =(n-1)!
      - =6！=720

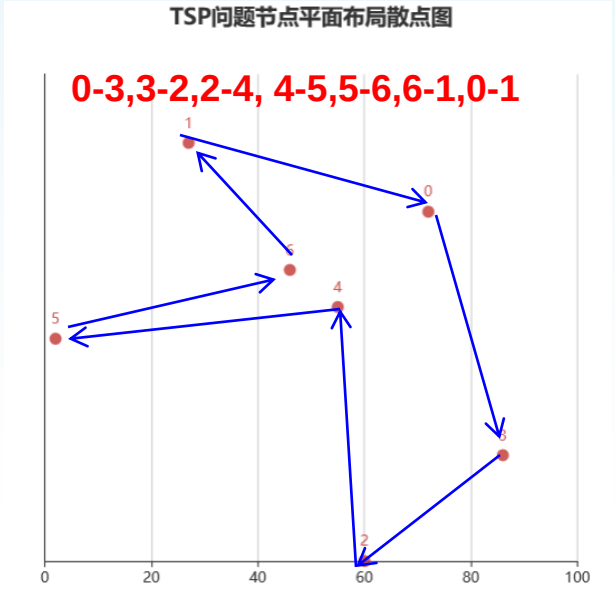


两边互换算法Lin2opt – 边集基础

可行解示例-以前述示例中前7个点为例来列举可行解

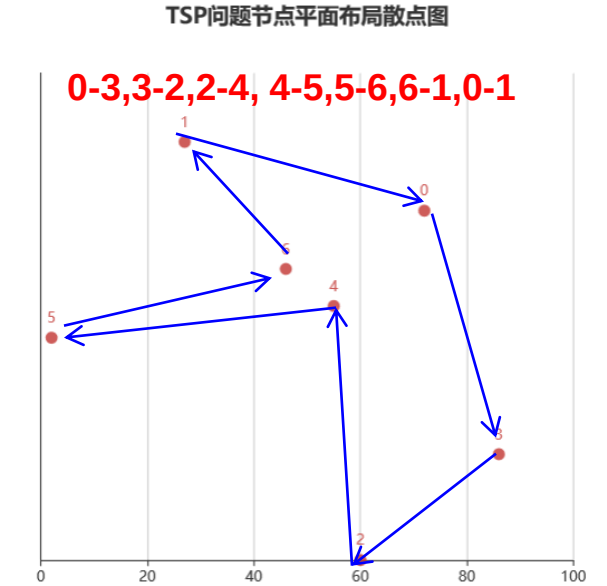
- 全部边集合
  - 0-1,0-2,0-3,0-4,0-5,0-6,1-2,1-3,1-4,1-5,1-6,2-3,2-4,2-5,2-6,3-4,3-5,3-6,4-5,4-6,5-6
- 可行解边集合：



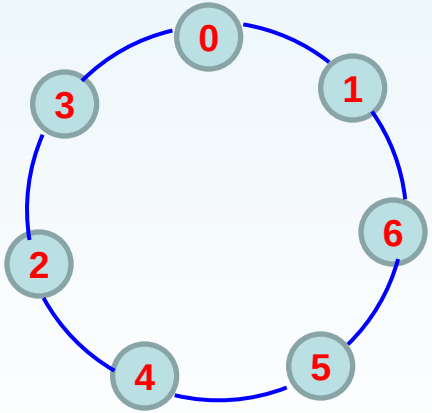


两边互换算法Lin2opt – 算法步骤

Lin2opt算法：边互换的注意事项



转换



两边互换算法Lin2opt – 算法步骤

Lin2opt算法基本步骤：

① 先随机生成或按照其他简单启发式算法生成一条可行解，

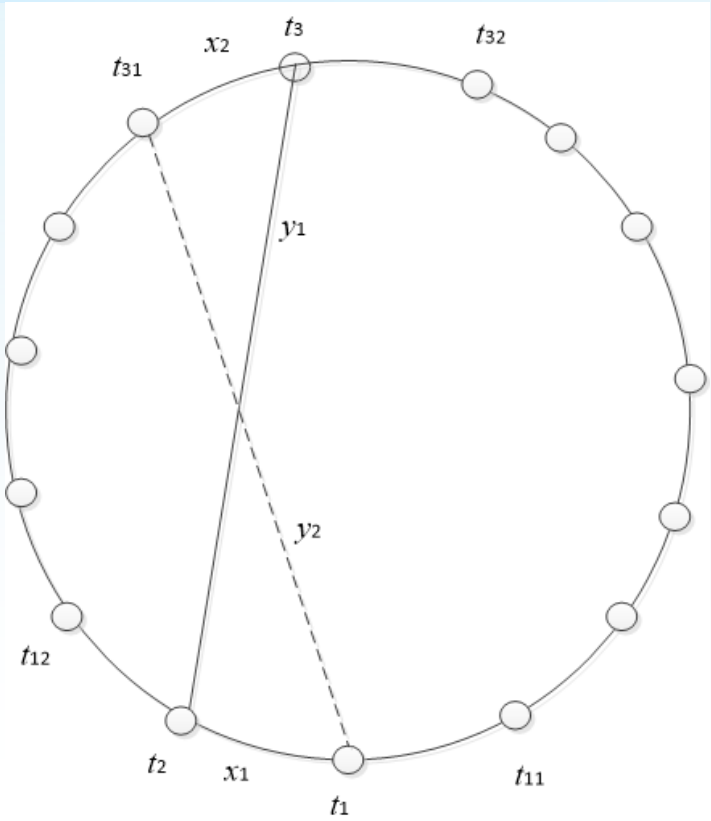
② 然后依次以路径各个位置的点作为t_1，据此确定换出边x_1，

③ 然后寻找以t_2一个端点的不在当前解中的边y_1，也就确定了点t_3，

④ 再以t_3为一个端点找满足图中情况的t_{31}及其换出边x_2，则确定了t_{31}和t_1为两个端点边y_2。

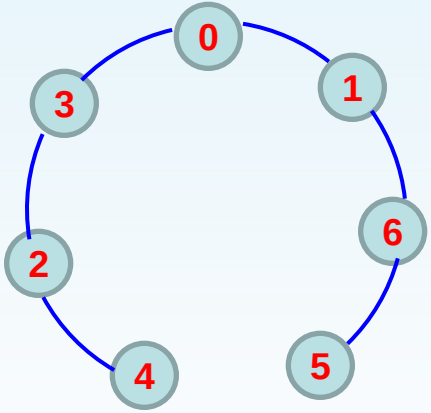
⑤ 由于确定了x_1之后，t_3可以在除了t_1、t_2和t_{12}这三点之外的任何一点中选择，即t_3可选对象数量有n-3个；当t_1、t_2和t_3确定之后，t_{31}也随之确定，没有第二种选择，因此确定x_1之后，可以生成n-3个可行解，

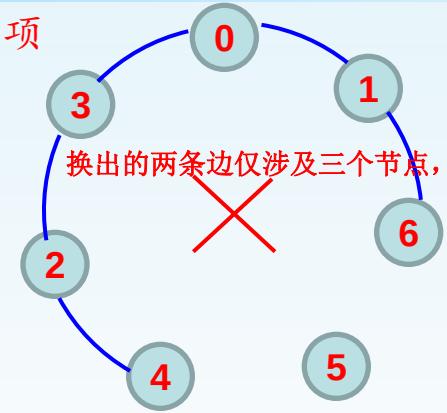
⑥ 然后从这n-3个可行解中找出最好的解同原有可行解比较，如果有改善则使用新解替换原有可行解，重复次循环过程，直至t_1对整个路径循环了一个周期。



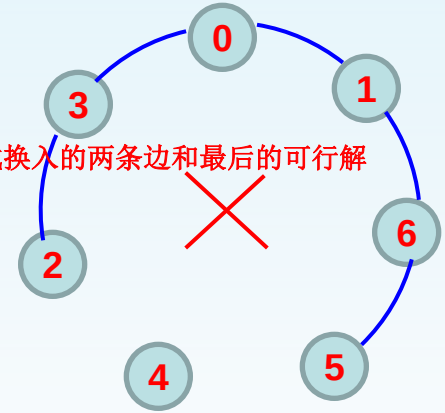
两边互换算法Lin2opt －注意事项

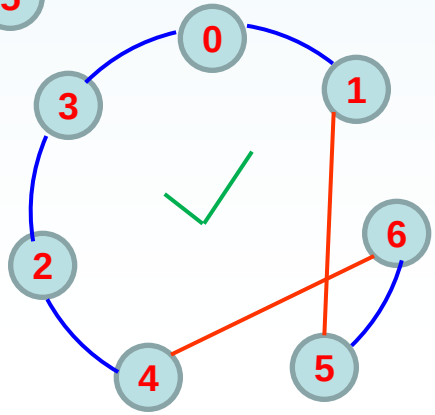
Lin2opt算法：边互换的注意事项

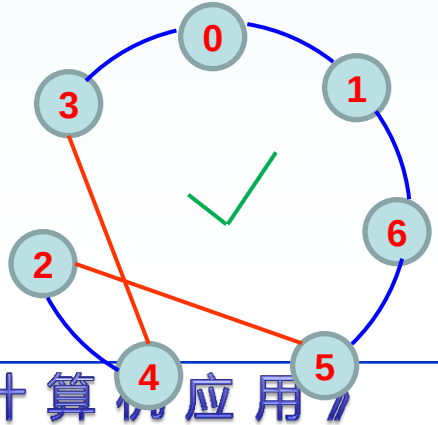




换出的两条边仅涉及三个节点，所以无法形成换入的两条边和最后的可行解







TSP算法实例结果对比

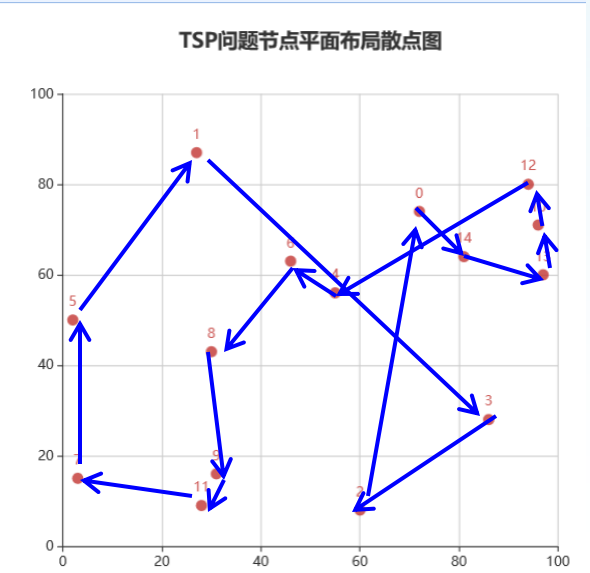
NN：[0, 14, 13, 10, 12, 4, 6, 8, 9, 11, 7, 5, 1, 3, 2], 456

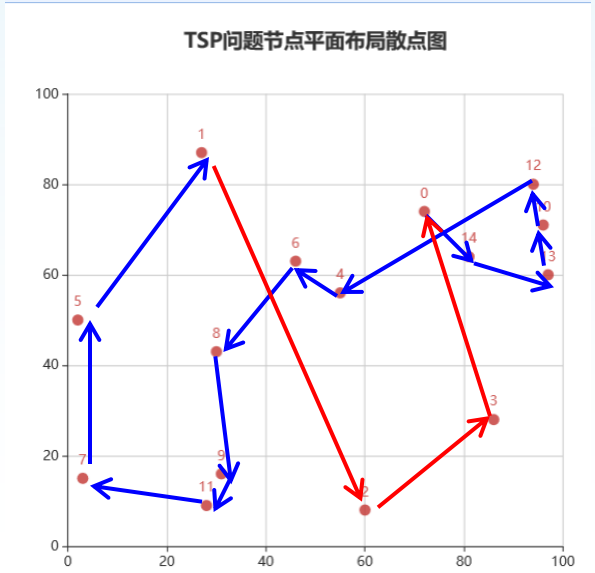
2点互换：[0, 14, 13, 10, 12, 4, 6, 8, 9, 11, 7, 5, 1, 2, 3], 440

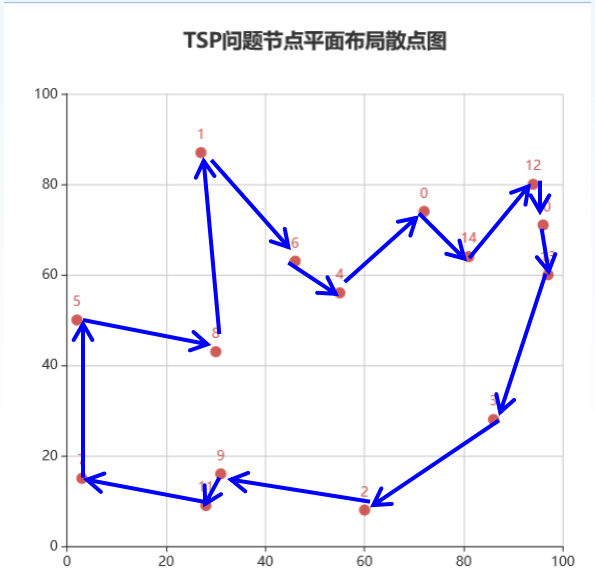
2边互换：[0, 14, 10, 13, 12, 3, 2, 9, 11, 7, 5, 8, 1, 6, 4],360

思考题【列出运算过程，并进行结果对比】：

1. 将NN的结果进行一个轮次的Lin2opt运算







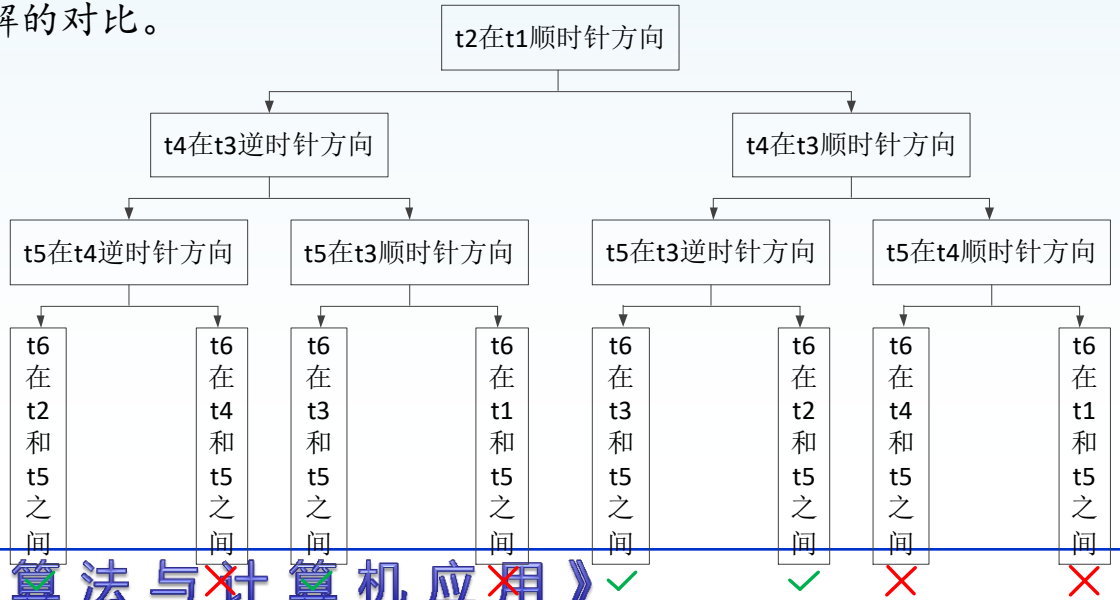
三边互换算法Lin3opt - 算法步骤

Lin3opt算法：边互换的注意事项

两边进行互换产生新可行解的过程相对不是太复杂，而进行三边互换的过程就比较复杂。

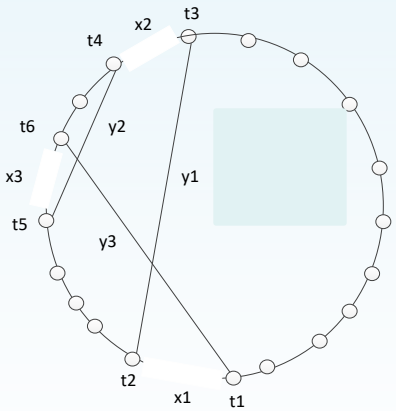
由于三对边的互换涉及到6个点，需要根据点的相对位置进行各种情况下边的选择可行性分析。

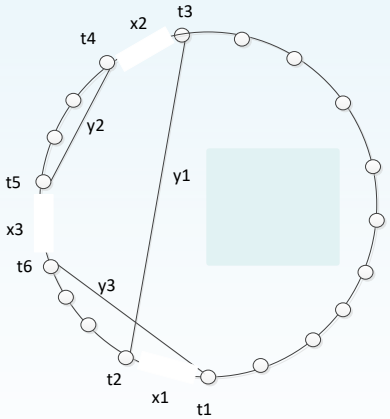
下面以边x1两个端点中t2在t1顺时针或逆时针方向进行分别分析，两种情形下后续点的选择均有八种可能性，而八种可能性中只有四种可以产生可行解，因此对某一可行解进行LK3-opt算法进行优化时需要对八种产生可行解的情况进行分析和解的对比。

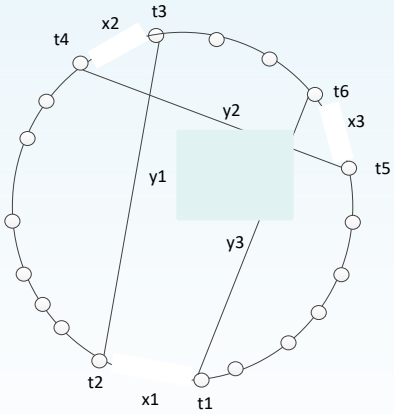


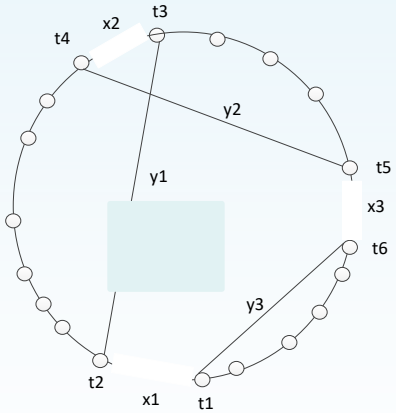
三边互换算法Lin3opt – 算法步骤

Lin3opt算法：顺时针可行性分析



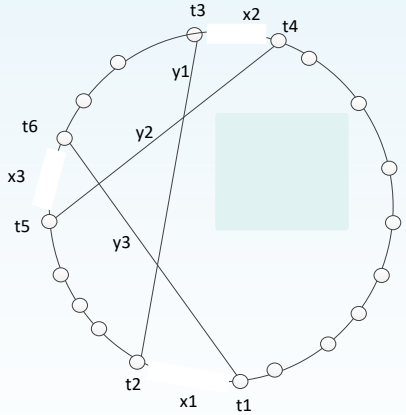


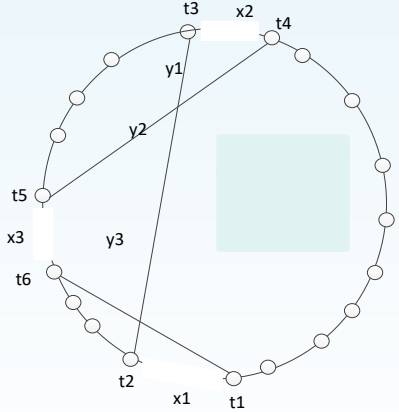


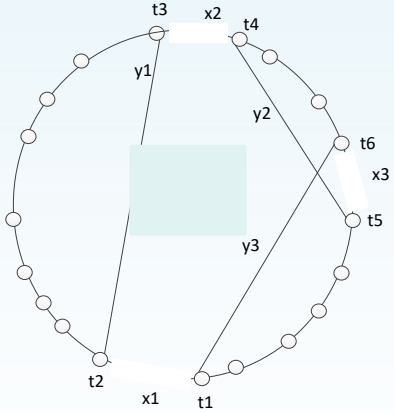


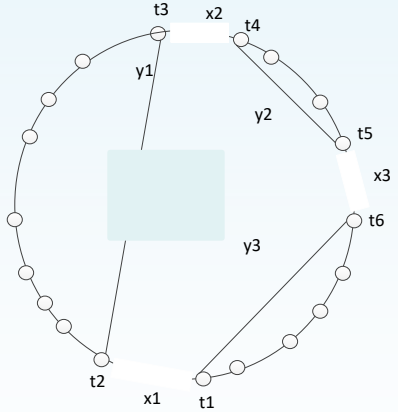
三边互换算法Lin3opt – 算法步骤

Lin3opt算法：顺时针可行性分析









TSP算法运算性能对比

算例	NN	2opt	Lin2opt	Lin3opt
bayg29	10209	9812	9205	9224
att48	40583	40000	34880	34270
eil51	511	502	446	433
pr76	153462	149309	115920	111443
lin105	20356	20226	16928	14759
bier127	135737	128305	123659	120462
kroA150	33633	33212	29463	27625
a280	3157	3151	2952	2708
pr299	59890	59471	53355	49894
lin318	54019	53800	50021	43857

结果总里程

运算时间ms

算例	NN	2opt	Lin2opt	Lin3opt
bayg29	7	17	90	74
att48	2	31	226	61
eil51	1	35	262	42
pr76	2	70	620	107
lin105	1	61	1097	181
bier127	2	126	1818	339
kroA150	4	141	2888	400
a280	9	907	15075	1600
pr299	13	1148	25724	1775
lin318	10	1095	25767	2238

练习题