CVRP启发式优化算法

CVRP启发式优化算法

1. • CVRP定义
2. • CW节约里程法+改进节约里程法
3. • sweep扫描算法
4. • λ互换下降法

一、 CVRP定义

车辆路径问题定义

- 容量约束车辆路径问题（*Capacitated VRP*）是最基本的*VRP*问题，即车辆装载的货物体积或重量不能超过其最大承载量。*CVRP*中所有客户所需的物品量已知，且同一个顾客的需求只能由一辆车进行送达，不能拆分多次送达；所以车辆的容量或载重量等特征都是一样的，而且都停放在一个中心车场。*CVRP*问题优化目标是获得能够为全部顾客运送货物的一种车辆路径方法，且该方案具有最小的运作总成本。
- *CVRP*数学表述：一个车场配备了m辆容量为C的运输车辆，用于给在地理位置不同的n个客户送货，第i个客户需要的货物量为q_i，节点间距离或成本为d_{ij}（c_{ij}）试安排车辆的服务客户及其先后顺序，实现总运输里程最短。

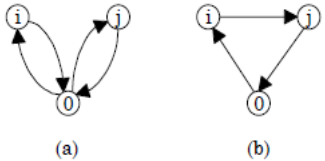
CVRP启发式优化算法

1. • CVRP定义
2. • CW节约里程法+改进节约里程法
3. • sweep扫描算法
4. • λ互换下降法

二、CW节约里程法+改进节约里程法

CW节约里程法

- CW节约算法是有Clarke和Wright在1964年提出的一种求解经典CVRP的启发式算法，作为相对较早提出的一种启发式算法，虽然CW节约算法一般情况下不能获取CVRP的最优解，但是却可以很容易的获取相对比较好的可行解，而且求解思路简单明了。
- 基本思想：
  - 两个客户分别由两辆车服务的行驶总里程必然大于由一辆车服务的总里程【三角不等式】；
    - 节约里程 = 两辆车分别服务两个客户总里程 - 一辆车集中服务两个客户的总里程
    - Saving=$d_{0i}+d_{i0}+d_{0j}+d_{j0}-[d_{0i}+d_{ij}+d_{j0}]=d_{i0}+d_{j0}-d_{ij}$
  - 如何选择客户合并： 优先选择两个客户节约总里程最大的、可以装入一辆车的节点放入一条路径可以节约更多的行驶里程【总成本】，则形成的解目标函数将比较好。



二、CW节约里程法+改进节约里程法

CW节约里程法-示例

- 现有一个车场和6个客户的CVRP问题，客户需求量分别为28、35、30、40、45、25，车辆容量为100，车场和客户之间的距离矩阵如下表，试使用节约里程法求解。

	0	1	2	3	4	5	6
0		30	65	67	53	54	28
1			43	72	50	74	53
2				52	27	89	65
3					20	49	40
4						60	43
5							15
6							

- 要不先用六辆车，然后看看哪些车可以合并？
- 手工做一下！

二、CW节约里程法+改进节约里程法

CW节约里程法-示例

- 客户需求量28、35、30、40、45、25，车辆容量为100
- 一客户一车辆一路经

	0	1	2	3	4	5	6
0		30	65	67	53	54	28
1			43	72	50	74	53
2				52	27	89	65
3					20	49	40
4						60	43
5							15
6							

车辆路径	行驶里程	总里程
0-1-0	30+30=60	
0-2-0	65+65=130	
0-3-0	67+67=134	594
0-4-0	53+53=106	
0-5-0	54+54=108	
0-6-0	28+28=56	

怎么合并路径，改善目标

优先选择两个节约总里程最大客户合并

全部点对节约总里程算出来

行号	节点对	节约量
1	3-4：	100
2	2-4：	91
3	2-3：	80
4	3-5：	63
5	5-6：	67
6	3-6：	55
7	1-2：	52
8	4-5：	47

行号	节点对	节约量
9	4-6：	38
10	1-4：	33
11	2-5：	30
12	2-6：	28
13	1-3：	25
14	1-5：	10
15	1-6：	5

$$Saving = d_{i0} + d_{j0} - d_{ij}$$

二、CW节约里程法+改进节约里程法

CW节约里程法-示例

- 客户需求量28、35、30、40、45、25，车辆容量为100
- 一客户一车辆一路经

车辆路径	行驶里程	总里程
0-1-0	30+30=60	
0-2-0	65+65=130	
0-3-0	67+67=134	594
0-4-0	53+53=106	
0-5-0	54+54=108	
0-6-0	28+28=56	

怎么合并路径，改善目标

行号	节点对	节约量
1	3-4:	100
2	2-4:	91
3	2-3:	80
4	3-5:	63
5	5-6:	67
6	3-6:	55
7	1-2:	52
8	4-5:	47

行号	节点对	节约量
9	4-6:	38
10	1-4:	33
11	2-5:	30
12	2-6:	28
13	1-3:	25
14	1-5:	10
15	1-6:	5

- 行1：0-3-4-0【30+40】√；
- →行2：【70+35>100】×；
- →行3：【70+35>100】×；
- →行4：【70+45>100】×；
- →行5：两点都不包含在合并路径中；
- 行6：0-6-3-4-0【70+25】√；
- 后续均无法加入，路径形成：0-6-3-4-0
- 从未删除数组逐次将有一个点在已有路径中的行删除，其他保留

- 行7：0-1-2-0【28+35=63】√；
- →行11：【63+45>100】×；
- →行14：【63+45>100】×；
- 节约里程表为空，路径形成：0-1-2-0
- 从未删除数组逐次将有一个点在已有路径中的行删除，其他保留

- 节约里程表为空；
- 查找未合并的节点作为单节点路径
- 0-5-0

- 计算总里程385

二、CW节约里程法+改进节约里程法

CW节约里程法-示例

- 客户需求量28、35、30、40、45、25，车辆容量为100
- 一客户一车辆一路经

车辆路径	行驶里程	总里程
0-1-0	30+30=60	
0-2-0	65+65=130	
0-3-0	67+67=134	594
0-4-0	53+53=106	
0-5-0	54+54=108	
0-6-0	28+28=56	

怎么合并路径，改善目标

行号	节点对	节约量
1	3-4:	100
2	2-4:	91
3	2-3:	80
4	3-5:	63
5	5-6:	67
6	3-6:	55
7	1-2:	52
8	4-5:	47
9	4-6:	38
10	1-4:	33
11	2-5:	30
12	2-6:	28
13	1-3:	25
14	1-5:	10
15	1-6:	5

- 行1：0-3-4-0【30+40】√；
- →行2：【70+35>100】×；
- →行3：【70+35>100】×；
- →行4：【70+45>100】×；
- →行5：两点都不包含在合并路径中；可以直接生成0-5-6-0的路径√
- 行6：3、6分别属于两条路径，不合并×；
- 行7：两点都不包含在合并路径中；可以直接生成0-1-2-0的路径√
- 后续行中两点都处于不同路径，不合并，结束路径合并，形成三条路径0-3-4-0, 0-5-6-0, 0-1-2-0.

- 节约里程表为空；
- 查找是否有未合并的节点，有则将其作为单节点路径

- 计算总里程375

顺序节约里程法
Sequential Saving

并行节约里程法
Parallel Saving

路径构造法
Construct method

二、CW节约里程法+改进节约里程法

改进节约里程法-示例

- Gaskell和Yellow分别在1967和1970年提出的CW节约里程法改进算法
- Saving = $d_{i0} + d_{j0} - \lambda d_{ij}$
- λ：路径形状系数[0,1]，该参数越大，准备合并的两个顶点之间的路径长度在表达式中占的权重越大

	0	1	2	3	4	5	6
0		30	65	67	53	54	28
1			43	72	50	74	53
2				52	27	89	65
3					20	49	40
4						60	43
5							15
6							

车辆路径	行驶里程	总里程
0-1-0	30+30=60	
0-2-0	65+65=130	
0-3-0	67+67=134	594
0-4-0	53+53=106	
0-5-0	54+54=108	
0-6-0	28+28=56	

怎么合并路径，改善目标

优先选择两个节约总里程最大客户合并

全部点对节约总里程算出来

Saving = $d_{i0} + d_{j0} - \lambda d_{ij}$

二、CW节约里程法+改进节约里程法

- 算法编码流程
  - 问题参数输入：车辆容量C、节点距离矩阵dist[n+1,n+1]，不考虑车辆数量约束；
  - 生成节约里程表savingTable
    - 生成n+1个节点编号的点对pairArray，该数据使用C[n+1,2]行两列数组表示
    - 使用公式算出点对间节约的里程数，列于pairArray的第三列
    - 将3列pairArray根据第三列降序排列
    - 按照并行或顺序方式生成路径

	0	1	2	3	4	5	6
0		30	65	67	53	54	28
1			43	72	50	74	53
2				52	27	89	65
3					20	49	40
4						60	43
5							15
6							

pareArray

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

作业题

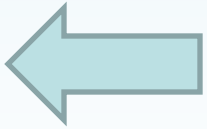
- 问题参数：
  - 一个车场和9个客户的地理坐标如下，距离矩阵如下表：
    - {{375 12 142 245 412 632 452 758 278 398 };
    - {460 542 523 120 499 528 101 789 124 999 } }
  - 客户的需求量依次为：0, 48, 43, 52, 38, 54, 74, 63, 60, 32
  - 车辆容量为180
- 问题要求：使用顺序和并行节约里程法计算车辆路径结果
  - 【节约里程表计算辅助工具： http://www.iescm.com/logisticOpt/index.html 】

节点	0	1	2	3	4	5	6	7	8	9
0	0	372	241	364	54	266	367	505	350	539
1	372	0	131	482	402	620	623	786	495	598
2	241	131	0	416	271	490	524	671	422	540
3	364	482	416	0	414	562	208	843	33	892
4	54	402	271	414	0	222	400	451	398	500
5	266	620	490	562	222	0	463	290	537	526
6	367	623	524	208	400	463	0	753	176	900
7	505	786	671	843	451	290	753	0	820	417
8	350	495	422	33	398	537	176	820	0	883
9	539	598	540	892	500	526	900	417	883	0

二、CW节约里程法+改进节约里程法

基于easyopt.jar的算法比较实验

```
**********already optimal result:   375.0
    saving.....sequence    458.0
    saving.....parallel    387.0
    modified saving.....sequence    448.0
    modified saving.....parallel   399.0
```



```
import easyopt.instances.VRPinstance;
import java.util.ArrayList;
/**用于测试使用VRP类中的节约里程法和改进型节约里程法求解CVRP的性能 * */
public class TestSavingAlgorithm {
   public static void main(String[] args) {
      VRPinstance instance = new VRPinstance();
      instance.initEn22K4();//使用VRPinstance类中的En22K4算例数据进行测试

      int truckCap = instance.capacity;
      int[] dmd = instance.dmdQty;
      double[][] posXY = instance.posXY;
      System.out.println("**********already optimal result:   "+ instance.optResult);
      double sumCost =0;
      ArrayList<VRP.Route> rList = VRP.optBySavingCWSeq(truckCap, dmd, posXY);
      for(VRP.Route r:rList){       sumCost+=r.sumCost;       }
      System.out.println("    saving.....sequence    "+sumCost);

      double sumCost2 =0;
      ArrayList<VRP.Route> rList2 = VRP.optBySavingCWParallel(truckCap, dmd,
posXY);
      for(VRP.Route r:rList2){          sumCost2+=r.sumCost;       }
      System.out.println("    saving.....parallel    " + sumCost2);
 //   System.out.println(rList2.toString());

      sumCost =0;
      double lamda=0.6;
      rList = VRP.optByModSavingSeq(truckCap, dmd, posXY,lamda);
      for(VRP.Route r:rList){          sumCost+=r.sumCost;       }
      System.out.println("    modified saving.....sequence    "+sumCost);

      sumCost2 =0;
      rList2 = VRP.optByModSavingParallel(truckCap, dmd, posXY,lamda);
      for(VRP.Route r:rList2){          sumCost2+=r.sumCost;       }
      System.out.println("    modified saving.....parallel   "+ sumCost2);
   }
}
```

二、CW节约里程法+改进节约里程法

基于easyopt.jar的算法比较实验

以En22K4、En30K3、En51K5、En76K10和En101K8五个算例为运算数据，λ分别设为0.2、0.4、0.6、0.8四个数值，进行节约里程法和改进节约里程法优化实验

```
import easyopt.common.EasyArray;
import easyopt.instances.VRPinstance;
import java.util.ArrayList;

/**用于测试使用VRP类中的节约里程法和改进型节约里程法求解CVRP的性能 * */
public class TestSavingAlgorithm {
   public static void main(String[] args) {
      manyExperiment();
   }

   public static void manyExperiment(){
      double[][] results = new double[5][10];
      for(int i=0;i<5;i++){
         VRPinstance instance = new VRPinstance();
         if(i==0)   instance.initEn22K4();
         if(i==1)   instance.initEn30K3();
         if(i==2)   instance.initEn51K5();
         if(i==3)   instance.initEn76K10();
         if(i==4)   instance.initEn101K8();

         int truckCap = instance.capacity;
         int[] dmd = instance.dmdQty;
         double[][] posXY = instance.posXY;
```

初始化算例数据

提出算例数据

```
         double sumCost =0;
         ArrayList<VRP.Route> rList = VRP.optBySavingCWSeq(truckCap, dmd, posXY);
         for(VRP.Route r:rList){          sumCost+=r.sumCost;          }
         results[i][0] = sumCost;

         double sumCost2 =0;
         ArrayList<VRP.Route> rList2 = VRP.optBySavingCWParallel(truckCap, dmd, posXY);
         for(VRP.Route r:rList2){          sumCost2+=r.sumCost;          }
         results[i][1] = sumCost2;
         for(int j=0;j<4;j++){
            sumCost =0;
            double lamda=0.2+0.2*j;
            rList = VRP.optByModSavingSeq(truckCap, dmd, posXY,lamda);
            for(VRP.Route r:rList){          sumCost+=r.sumCost;          }
            results[i][2 + 2*j ] = sumCost;

            sumCost2 =0;
            rList2 = VRP.optByModSavingParallel(truckCap, dmd, posXY,lamda);
            for(VRP.Route r:rList2){          sumCost2+=r.sumCost;          }
            results[i][2 + 2*j + 1  ] = sumCost2;
         }
      }
      EasyArray.printArray(results,0);
   }
}
```

序贯节约里程法

并行节约里程法

λ不同值下改进节约里程法

二、CW节约里程法+改进节约里程法

基于easyopt.jar的算法比较实验

以En22K4、En30K3、En51K5、En76K10和En101K8五个算例为运算数据，λ分别设为0.2、0.4、0.6、0.8四个数值，进行节约里程法和改进节约里程法优化实验

节约里程法和改进节约里程法实验结果表

算例	SeqCW	PrlCW	λ=0.2		λ=0.4		λ=0.6		λ=0.8	
			MSeqCW	MPrlCW	MSeqCW	MPrlCW	MSeqCW	MPrlCW	MSeqCW	MPrlCW
En22K4	458	387	518	469	427	444	448	399	458	387
En30K3	817	586	969	599	754	571	783	594	812	594
En51K5	734	591	1045	595	824	568	720	579	689	579
En76K10	1061	907	1355	1014	1143	931	1078	897	1066	878
En101K8	1197	999	1633	1041	1299	959	1275	940	1277	938

从表中可以看出，并行算法通常要优于序贯算法，对于改进节约里程法求解结果受到参数λ的影响，且当参数λ在某些取值时可以获得比普通节约里程法要好的结果。

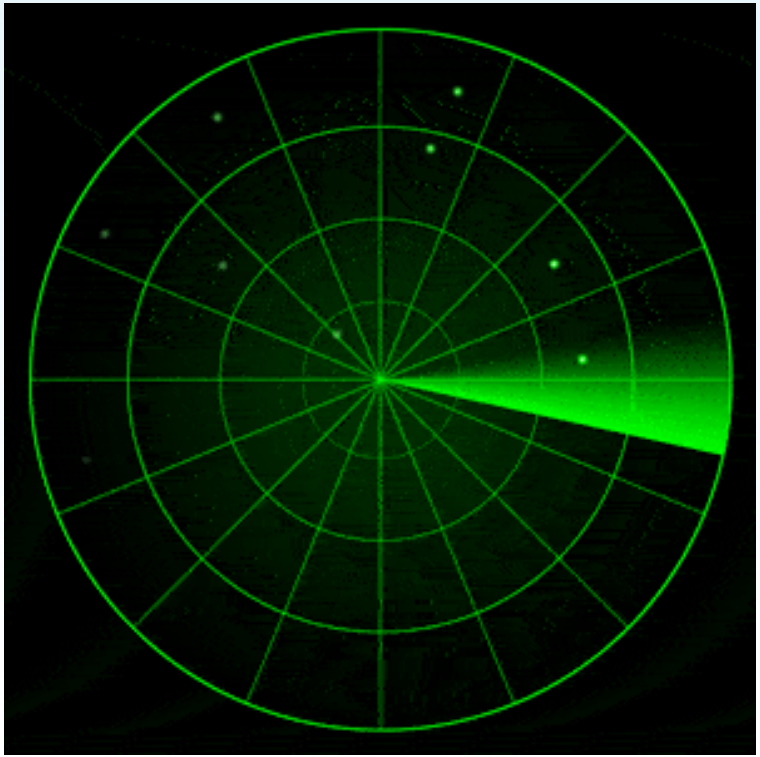
作业题

- 将前面的运算程序在IDE中实现
- 将每种算法结果的具体路径找出来
- 撰写作业报告

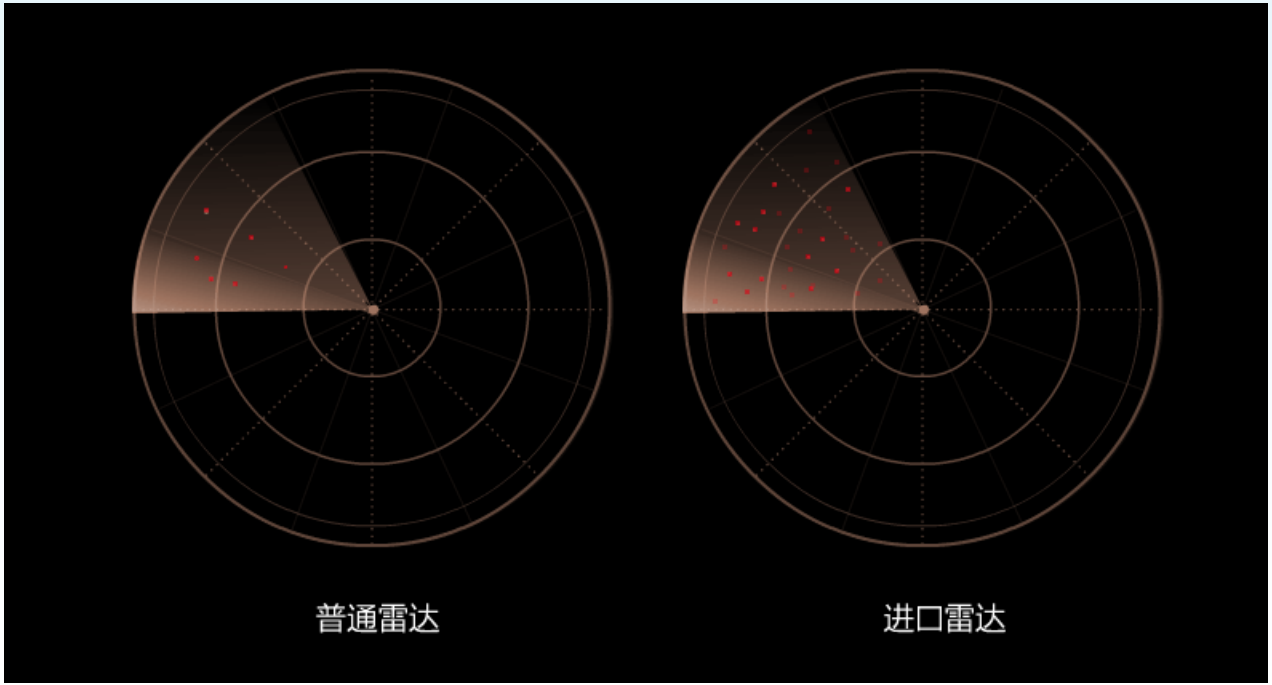
CVRP启发式优化算法

1. • CVRP定义
2. • CW节约里程法+改进节约里程法
3. • sweep扫描算法
4. • λ互换下降法

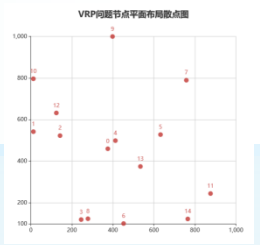
三、sweep扫描算法



三、sweep扫描算法



三、sweep扫描算法



扫描算法概述

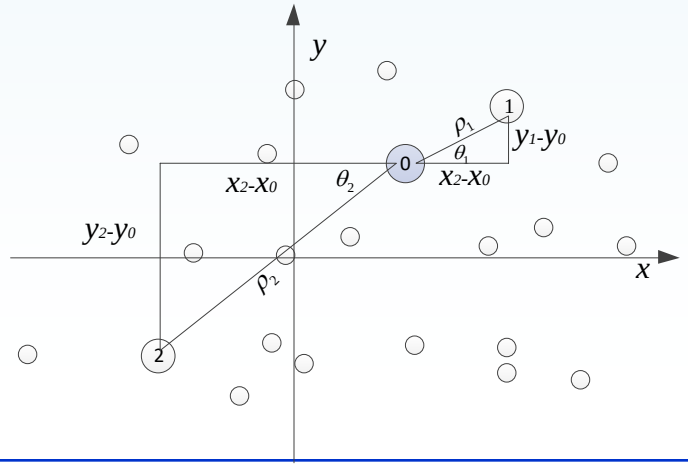
- 扫描算法（Sweep Algorithm）是以车场为极点进行扇形扫描的方式对客户进行分组，每一组客户需求由一辆车负责服务，组中的具体路径使用TSP相关算法进行优化求解以获得每一组的最短路径。该算法的基本思想是极角和极径相近的客户安排到同一辆车有较大可能获得最小的总里程。扫描算法开始由Wren在1971年最先提出，后来由Gillett在1974年对其进行具体描述和寻优性能分析，使其广为人知。
- 当给出车场和各个节点的二维平面坐标之后，可以算出按照下式算出各个节点*i*的极坐标：

$$\theta_i = a\tan\left(\frac{y_i - y_0}{x_i - x_0}\right)$$

$$\rho_i = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2}$$

车场二维平面坐标为(x_0, y_0)

节点 i 的二维平面坐标为(x_i, y_i)

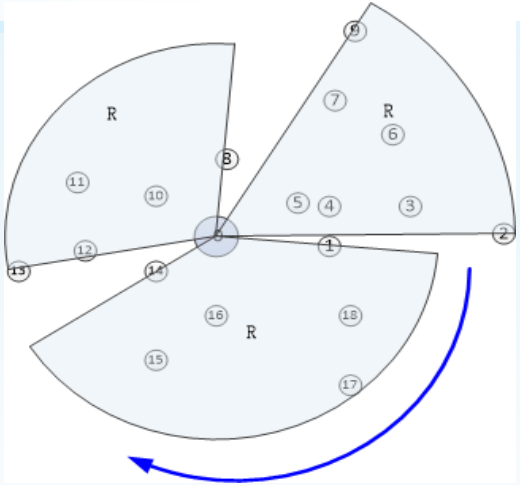


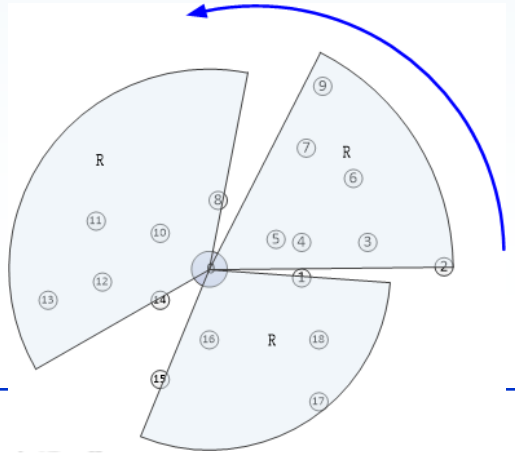
三、sweep扫描算法

扫描算法基本步骤：

- **Step1**（初始化路径）：选择一个还没有使用的车辆*k*。
- **Step2**（路径构建）：对未分配车辆的客户节点按照极角由小到大分配给车辆*k*，直至车辆*k*相关约束条件不满足时为止，然后对车辆*k*的具体行驶路径进行优化，获得一辆车的具体路径安排。如果还有节点未安排路径，则继续执行Step1，直至全部客户被安排了车辆。
- **Step3**（坐标旋转）：将X和Y坐标互换，即重新计算极角，然后重复Step1和Step2，获得多组解。这一步本质上是重新选择和设定极角最小的节点，然后以该节点为起点，逐步向极角大的节点扫描，分配车辆。
- **Step4**（逆排序求解）：重复Step2和Step3，只是在Step2中各个客户
得问题的
- **Step5**（

行输出。





示例中：

Step2-Step3：获得多少个解？

Step4：获得多少个解？

Step5：近似最优解是否唯一？

近似最优解的总里程是否唯一？

三、sweep扫描算法

- 扫描算法基本步骤：
  - Step1（初始化路径）：选择一个还没有使用的车辆k。
  - Step2（路径构建）：对未分配车辆的客户节点按照极角由小到大分配给车辆k，直至车辆k相关约束条件不满足时为止，然后对车辆k的具体行驶路径进行优化，获得一辆车的具体路径安排。如果还有节点未安排路径，则继续执行Step1，直至全部客户被安排了车辆。
- 示例
  - 例如在一个CVRP问题中，有18个客户需要配送中心进行送货。客户根据各自极坐标由小到大排序后进行编号，各自的需求量依次为：14、15、13、18、4、19、8、13、7、18、21、11、14、5、23、16、17、21，车辆容量50，试采用扫描算法的步骤来获得车辆路径方案。
  - 分别以节点1、3和9作为起始节点进行扫描分配

表1： 以节点1为起点进行任务分配结果示意图

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Q_i	14	15	13	18	4	19	8	13	7	18	21	11	14	5	23	16	17	21
k	1			2				3			4			5			6	

表2： 以节点3为起点进行任务分配结果示意图

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Q_i	14	15	13	18	4	19	8	13	7	18	21	11	14	5	23	16	17	21
k	6		1			2				3			4			5		6

表3： 以节点9为起点进行任务分配结果示意图

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Qi	14	15	13	18	4	19	8	13	7	18	21	11	14	5	23	16	17	21
k	5			6				7	1			2			3		4	

三、sweep扫描算法

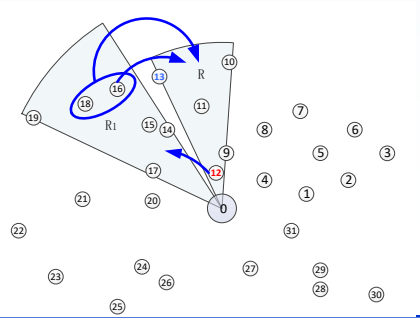
扫描算法基本步骤：

- Step2（路径构建）：对未分配车辆的客户节点按照极角由小到大分配给车辆*k*，直至车辆*k*相关约束条件不满足时为止，然后对车辆*k*的具体行驶路径进行优化，获得一辆车的具体路径安排。如果还有节点未安排路径，则继续执行Step1，直至全部客户被安排了车辆。
- 如何优化？
  - 查看当前正在构建的路径R是否可以同其后续路径R1进行节点互换，进而改善解的质量。
  - 后续解包括哪些节点？ --以后续的六个节点为限作为一个解
  - 换出的点可能是哪个？换入的点可能是哪个（哪两个）？
  - 换出点OI按照下列表达式获取
  - 后续路径中同当前路径最后一个客户最近的点II判断是否被换入，同II最近的点II2也判断是否被换入；
    - 判断标准：容量约束或里程约束是否满足；当前路径和后续路径总里程是否有改善

$$OI = \min_{h \in R} \left\{ \rho_{K[h]} - \theta_{K[h]} \overline{\rho} \right\}$$

距离极点越近的点可能被换出

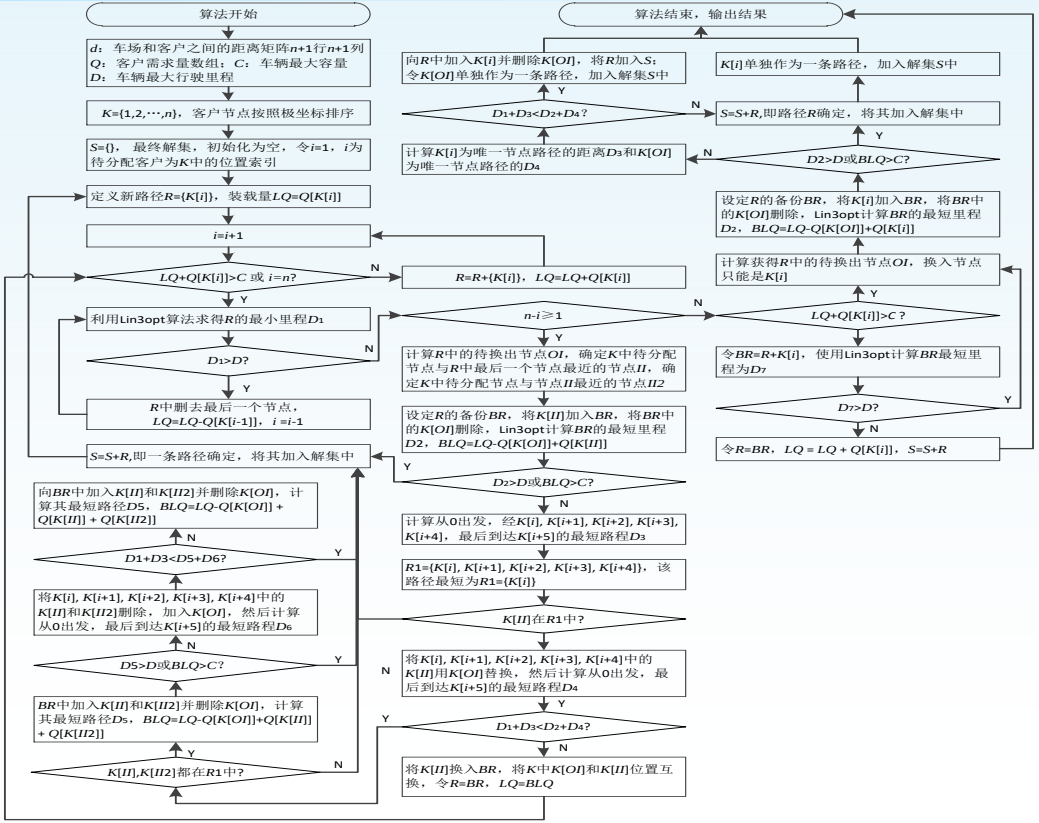
极角越大的点可能被换出；越靠近下个扇区的点可能被换出



三、sweep扫描算法

扫描算法基本步骤：

结合pdf文档和visio文档说明



作业题

- 问题参数：
  - 一个车场和14个客户的地理坐标如下：
    - {{375 12 142 245 412 632 452 758 278 398 12 876 124 534 765 } ;{460 542 523 120 499 528 101 789 124 999 796 245 632 375 123 } }
  - 客户的需求量依次为：0, 48, 43, 52, 38, 54, 74, 63, 60, 32, 52, 33, 35, 45, 61
  - 车辆容量为180
- 问题要求：使用sweep算法步骤获取最优车辆路径结果
- 简化：
  - 顺时针随机选择4个起点、逆时针随机选择4个起点，然后找出最优解
  - 不考虑相邻路径之间的优化

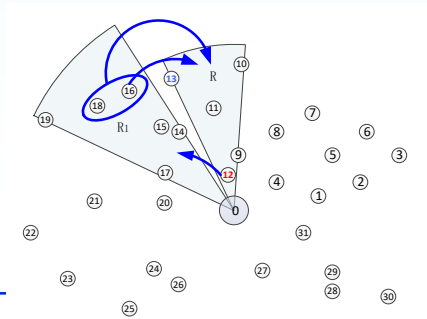
CVRP启发式优化算法

1. • CVRP定义
2. • CW节约里程法+改进节约里程法
3. • sweep扫描算法
4. • λ互换下降法

四、λ互换下降法

λ互换下降法概述

- 前述节约里程法属于构造法求解，即每次构造一条或多条路径，最后形成了问题的解；扫描算法属于先对客户进行分组，然后对组内客户路径进行优化，进而最后形成问题的解。这两种方法形成的VRP问题解可能还存在着改善空间，Osman等1993年提出了可以进行路径之间节点的交换来改善解质量，并设计了详细的求解步骤。
- Osman等将其算法称为λ互换下降法，基本思想是不断对路径对之间进行λ个节点互换，如果互换后形成的解质量有所提升则保留，然后重复该过程，直至解质量无法改善为止。
- λ互换过程
  - 随机选择两条路径，例如Rp和Rq；
  - 找出两条路径中的全部子集S1和S2，S1∈Rp，S2∈Rq，且|S1|≤λ，|S2|≤λ，一般λ为1或2
  - 依次从S1和S2中选择一对元素互换到另一个路径，从而形成两条新路径，新路径必须满足约束条件，然后计算新路径的总里程；
  - 对全部新里程中好的结果保留下来，再继续互换



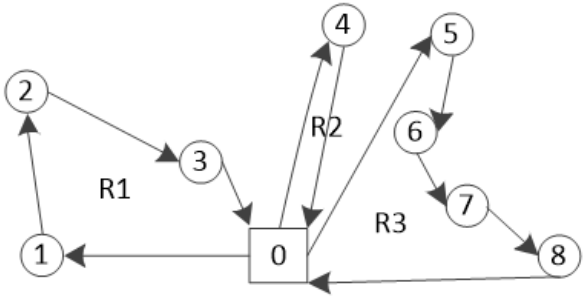
四、λ互换下降法

λ=2有几种互换形式？

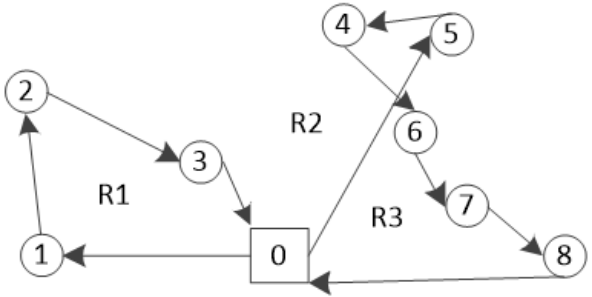
{1,0}{0,1}{1,1}{1,2}{2,1}{2,2}

λ互换形式

- 以λ=1为例说明两条路径的互换形式，该互换共有{1,0},{0,1},{1,1}三种形式
- 其中：
  - {1,0}和{0,1}是将一个路径中的一个点移动到另一条路径中；
  - {1,1}是将一个路径中的一个点同另一条路径中的一个点进行互换；



（a） （1,0）操作前



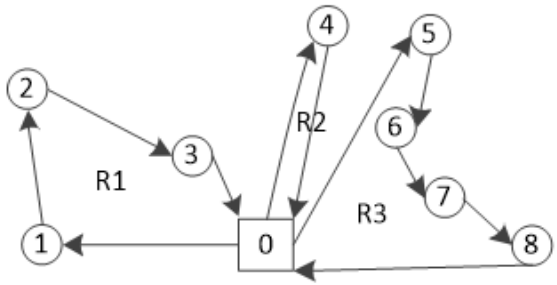
（b） （1,0）操作后

新移进的点4应该放到什么位置？

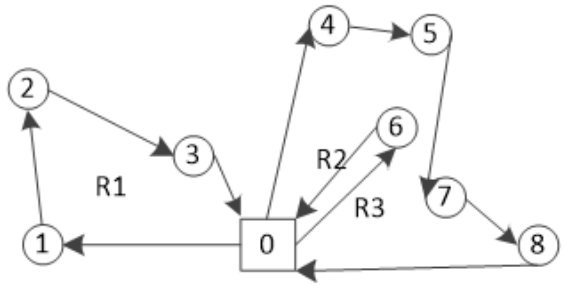
四、λ互换下降法

λ互换形式

- 以λ=1为例说明两条路径的互换形式，互换形式共有{1,0},{0,1},{1,1}三种
- 其中：
  - {1,0}和{0,1}是将一个路径中的一个点移动到另一条路径中；
  - {1,1}是将一个路径中的一个点同另一条路径中的一个点进行互换；



（a） （1, 1）操作前



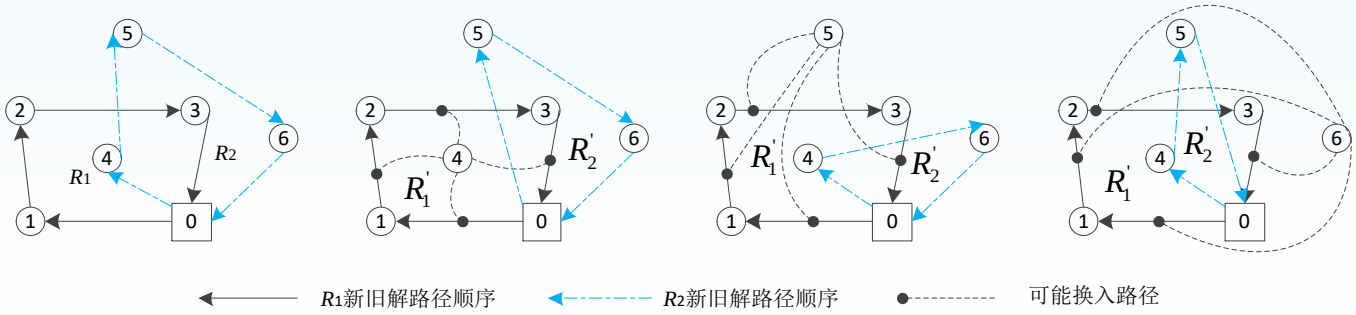
（b） （1, 1）操作后

4和6互换

四、λ互换下降法

λ互换降低寻优运算耗时策略

- 以λ=1为例说明，两条路径分别为Rp和Rq，各自里程为fp和fq
- {1,0}挪移操作将Rp中一个节点移入Rq：
  - 将移入节点{*i*}放入到Rq中各个位置计算出Rq∪{*i*}的最小总里程f1，将Rp/{*i*}其他节点位置不变计算总里程f2；
  - 如果f1+f2≤fp+fq，则利用Lin2opt进行新路径的优化

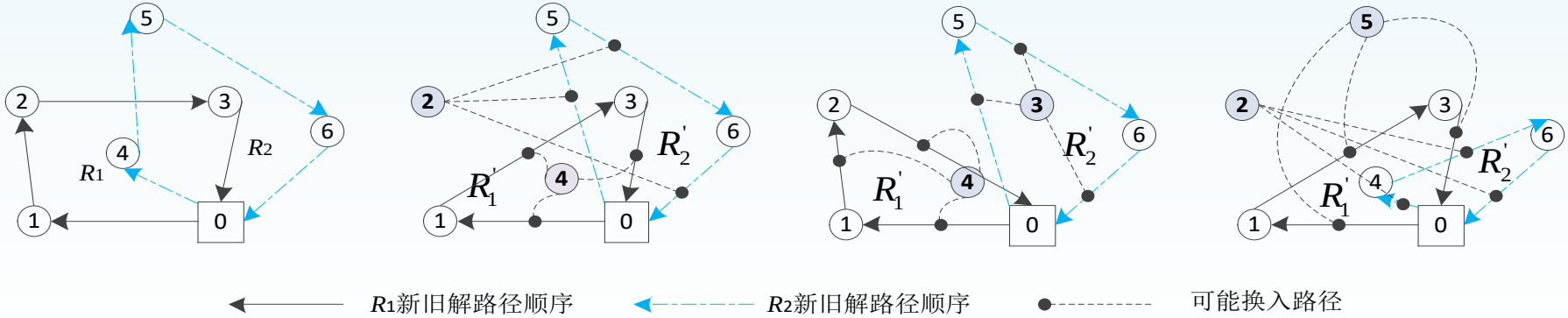


R2中的各个节点挪移到R1后的全部结果示意图

四、λ互换下降法

λ互换降低寻优运算耗时策略

- {1,1}互换操作将Rp中节点{*i*}和Rq中节点{*j*}进行互换：
  - 将节点{*i*}放入到Rq /{*j*}中各个位置计算出Rq/{*j*}∪ {*i*}的最小总里程f1，将节点{*j*}放入Rp/ {*i*}中各个位置计算出Rp/ {*i*} ∪ {*j*}的最小总里程f2；
  - 如果f1+f2≤fp+fq，则利用Lin2opt或Lin3opt进行新路径的优化



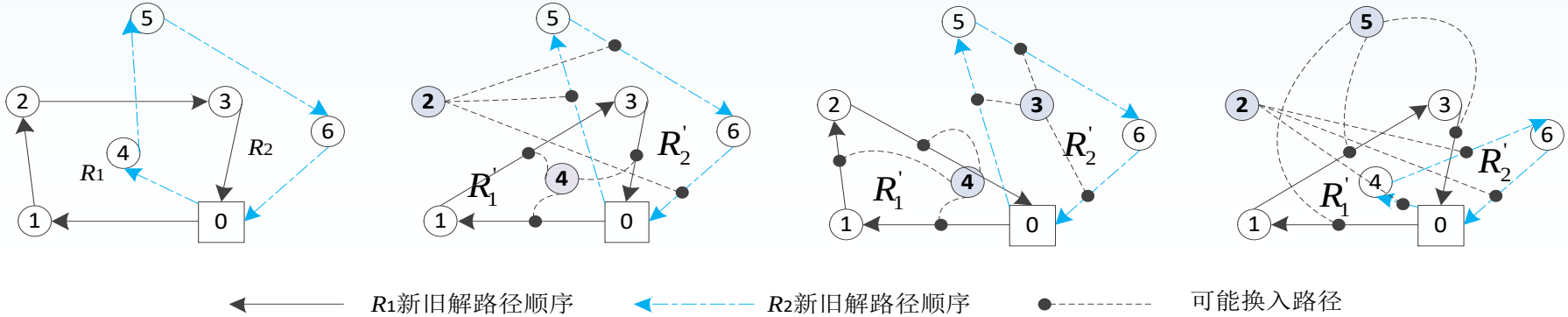
（1，1）互换近似估计新解生成示意图

每个点对互换都会生成很多可行解？如何选择一个作为新解？

四、λ互换下降法

λ互换下降法选择新解策略：

- （1）最优改善选择策略（Best-improve，BI）：从邻域解集中选择改善效果最好的可行解替换当前解；
- （2）最先改善选择策略（First-improve，FI）：以互换操作过程中最先出现改善的可行解替换当前解。



Sweep扫描算法和λ互换下降法程序实现

程序实现

- easyopt.jar中VRP类中的方法
  - optBySweep
  - optByLamdaExchange

表 5.7 算法最优解总里程对比表

算例	CWS	CWP	SWEEP	LAMDA	λ =0.3		λ =0.6		λ =0.9	
					MCWS	MCWP	MCWS	MCWP	MCWS	MCWP
En22K4	458	387	397	383	427	444	448	399	458	387
En30K3	817	586	515	562	900	575	783	594	812	590
En51K5	734	591	544	579	833	568	720	579	697	578
En76K10	1061	907	899	880	1276	953	1078	897	1030	903
En101K8	1197	999	854	996	1555	936	1313	940	1294	960

算例	SeqCW	PrlCW	λ=0.2		λ=0.4		λ=0.6		λ=0.8	
			MSeqCW	MPrlCW	MSeqCW	MPrlCW	MSeqCW	MPrlCW	MSeqCW	MPrlCW
En22K4	458	387	518	469	427	444	448	399	458	387
En30K3	817	586	969	599	754	571	783	594	812	594
En51K5	734	591	1045	595	824	568	720	579	689	579
En76K10	1061	907	1355	1014	1143	931	1078	897	1066	878
En101K8	1197	999	1633	1041	1299	959	1275	940	1277	938

作业题

- 根据各自小组问题的示例数据，或相关文献中的数据，使用这四种算法程序进行计算并进行结果比较分析。
- 制作PPT，下次课进行演示说明。