优化算法设计与应用
实践项目

名　　称：　基于改进 SA 求解 VCRP 问题

学员姓名：　张鑫航　　专业：　管理科学与工程

学员学号：　201902001008　　年级：　2019 级

所属学院：　系统工程学院

指导教员：　杨志伟　　职称：　副教授

国防科技大学系统工程学院

目 录

摘　要

车辆路径的优化是供应链优化中的重要环节。本文设计了一种改进的模拟退火算法用于求解有客户需求、车辆最大载重量的车辆路径问题。运用交换算子和逆转算子 2 种邻域生成算子，采用基本的指数降温方式控制降温过程。主要改进在于：

1. 编码方案采用客户编号的顺序编码，并设计专门的解码方法**wuyanqun**能够把 2 种约束全都纳入考虑。
2. 在优化算法的多水平参数优化试验中，将参数和对应解的关系看作高斯过程。
3. 设置了经验参数优化、贝叶斯参数优化两种参数优化试验，获取了对于结果改进较为明显的参数，（其中经验参数优化试验采用并行计算的方式提升了试验效率）。
4. 添加了每辆车行驶路径的 TSP 问题的优化。
5. 设置最大迭代次数提升了算法运行效率。

由参数优化试验得到 2 种共 21 组较优的参数，依据这些参数求出所有问题的解，最终筛选出 2 组结果相差不大参数：经验参数 $T_0 = 500,\ \alpha = 0.85,\ num_{Tk} = 7$ 和贝叶斯优化参数 $T_0 = 47.63,\ \alpha = 0.9173,\ num_{Tk} = 5.801$。

选择贝叶斯优化参数 $T_0 = 47.63,\ \alpha = 0.9173,\ num_{Tk} = 5.801$ 的结果得到文件中所有 VRP 问题的路径和最短距离。一些问题的解已经接近文件所给出的最优解。相距最少的 A-n33-k5、A-n33-k6、A-n34-k5、A-n44-k6、A-n45-k7 基本已经达到文件所给最优解见下表。其他问题的误差也控制在 3% 以内。所有问题的路径和最优解见附录 E。

VRP 问题	SA 最优解	文件最优解	VRP 问题	SA 最优解	文件最优解
A-n33-k5	662	661	A-n44-k7	938	937
A-n33-k6	744	742	A-n45-k7	1146	1146
A-n34-k5	780	778			

关键字： CVRP　模拟退火算法　并行计算　贝叶斯优化　分支定界法

小组分工如下：

姓名	学号		分工
张鑫航	201902001008	组长	建模、基本 SA 的编写、贝叶斯参数优化试验、画图、写文章、SA 的改进
胡兢	201902010006	组员	建模、经验参数优化试验、基本 SA

第 1 章 问题分析

车辆路径问题（Vehicle Routing Problem，VRP），如图 1-1，指存在一定数量的客户，各自有不同数量的货物需求，配送中心向客户提供货物，由一个车队负责运送货物，要求在满足一定的约束下，设计适当的配送路径，目标是使所有客户的需求都能得到满足，同时达到诸如路程最短、成本最小、耗费时间最少等目的。

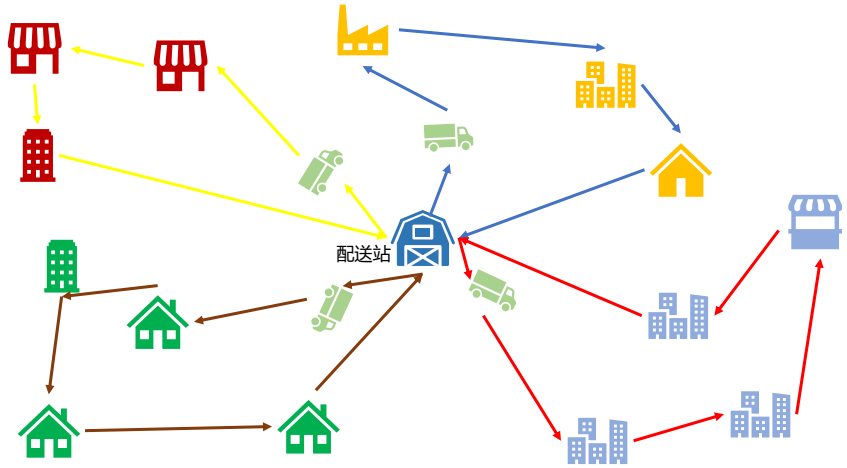


图 **1-1　VRP** 问题示意图

由此定义不难看出，旅行商问题（Traveling Saleman Problem,TSP）是 VRP 的特例。由于 Gaery 已证明 TSP 问题是 NP 难题[1997Heuristic]，因此，VRP 也属于 NP 难题[vrpNP]。

1.1 具有容量限制的路径优化问题（CVPR）

设有一配送中心（depot），配有多辆货车，有多位客户（customer），每位客户有其需求量。车辆从配送中心出发对客户进行配送服务最后返回配送中心，要求所有客户都被配送，每位客户一次配送完成，且不能违反车辆容量的限制，目的是所有车辆路线的总距离最小。

将上述要求归纳可以得到：

1. 每个车辆都要访问配送中心
2. 每个客户需要被一个车辆服务
3. 每个路径上，客户的总需求不要超过车辆的容量 Q
4. CVPR 的目标是最小化总路径成本

第 2 章 问题建模

2.1 符号说明

为了方便说明和求解，我们有如下符号说明

表 **2-1** 符号说明

符号	意义
Λ	客户顺序的一个排列
$\Lambda_i\,,(i=0,1,2,\cdots,M)$	Λ 的不相容子列加上 depot 节点，构成一个解
$\Lambda_i^{'}$	Λ 的不相容子列
$d_{\lambda_{i,j},\lambda_{i,j+1}}$	第 i 辆车经过边 $j\rightarrow j+1$ 的距离
$demand_{\lambda_{ij}}$	顾客 λ_{ij} 的需求量
W_max	货车最大载货量
M	需要的车的数量

2.2 问题模型

定义 1 $\Lambda=\langle\lambda_1,\lambda_2,\cdots,\lambda_N\rangle$ 为客户顺序的一个排列。**wuyanqun**

将 Λ 拆开分为 M 个互不相容的子列，并且加上节点 0，可构造一组问题的解。

$$\Lambda_i=\langle 0,\lambda_{i1},\lambda_{i2},\cdots,\lambda_{im_i},0\rangle\,,(i=0,1,2,\cdots,M) \tag{2.1}$$

其中 M 代表需要 M 辆车，记

$$\Lambda_i^{'}=\langle\lambda_{i1},\lambda_{i2},\cdots,\lambda_{im_i}\rangle$$

式 (2.1) 去除掉首尾的 0 满足

$$\Lambda_i^{'}\bigcap\Lambda_j^{'}=\varnothing \tag{2.2}$$

式 (2.2) 表示不同车辆经过的节点没有交集。

那么 M 辆车走过的子列的全程距离 z 有

$$z=\sum_{i=1}^{M}\sum_{j=0}^{m_i}d_{\lambda_{i,j},\lambda_{i,j+1}}$$

其中，$d_{\lambda_{i,j},\lambda_{i,j+1}}$ 代表节点第 i 辆车从节点 $\lambda_{i,j}$ 到 $\lambda_{i,j+1}$ 的距离。

对于某一条路径 Λ_i 要满足客户总需求不超过 W_max，有

$$\sum_{j=0}^{m_i} demand_{\lambda_{ij}} \leqslant W\_max$$

其中，$demand_{\lambda_{ij}}$ 代表节点 j 的需求量。那么有模型如下：

$$\min z = \sum_{i=1}^{M}\sum_{j=0}^{m_i} d_{\lambda_{i,j},\lambda_{i,j+1}}$$

$$s.t. \begin{cases} \Lambda_i' \bigcap \Lambda_j' = \varnothing \\ \sum\limits_{j=0}^{m_i} demand_{\lambda_{ij}} \leqslant W\_max \end{cases}$$

第 3 章 经典的模拟退火算法

物理退火流程如图 3-1，在升温-等温-降温过程中，粒子共做三种运动:

升温 ： 例子拜托非均匀运动状态

等温 ： 特定温度下粒子趋向平衡态的运动

降温 ： 粒子去向稳定态的运动

模拟退火算法就是将目标函数作为能量函数，当能量函数达到最小，目标函数也最小。

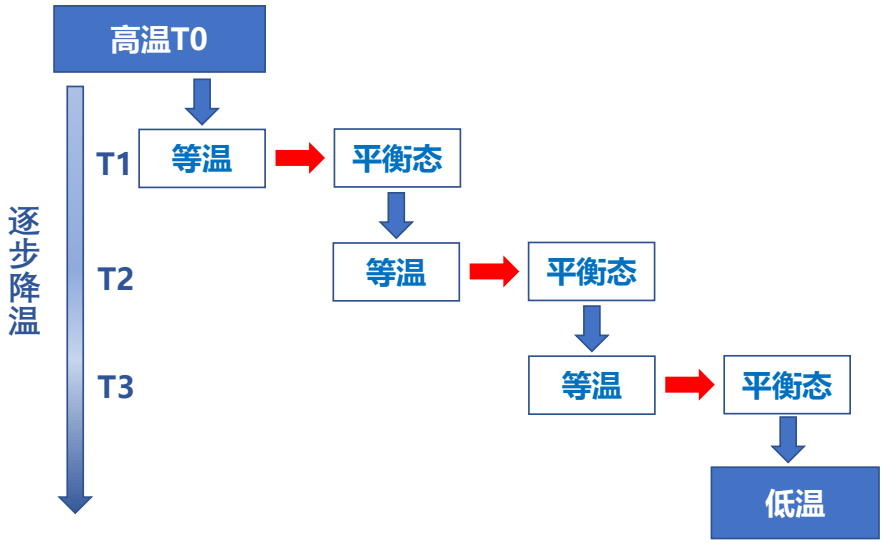


图 **3-1**　物理退火过程

3.1 算法流程和实现

物理退火各个步骤对应到算法如图 3-2，等温到平衡态相当于局部搜索，可以按照 Metropolis 准则接受劣解，最终降温循环得到粒子能量最低的状态。

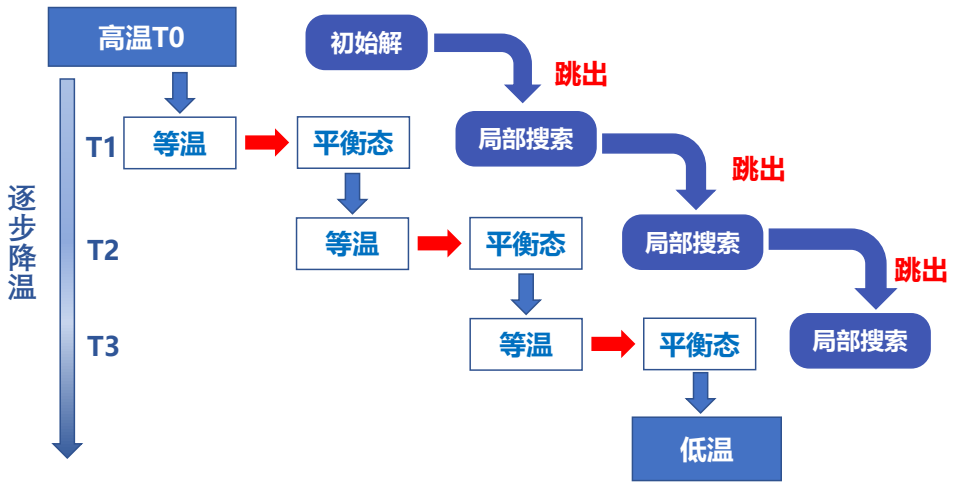


图 **3-2**　算法对应关系

3.1.1 实现流程

模拟退火算法的流程如图 3-3：按照 Metropolis 抽样准则如式 (3.1)

$$p\left(Tk,s,s^{'}\right)=\begin{cases}1 & if\ f\left(s^{'}\right)\leq f\left(s\right)\\ \exp\dfrac{f\left(s\right)-f\left(s^{'}\right)}{Tk} & others\end{cases} \tag{3.1}$$

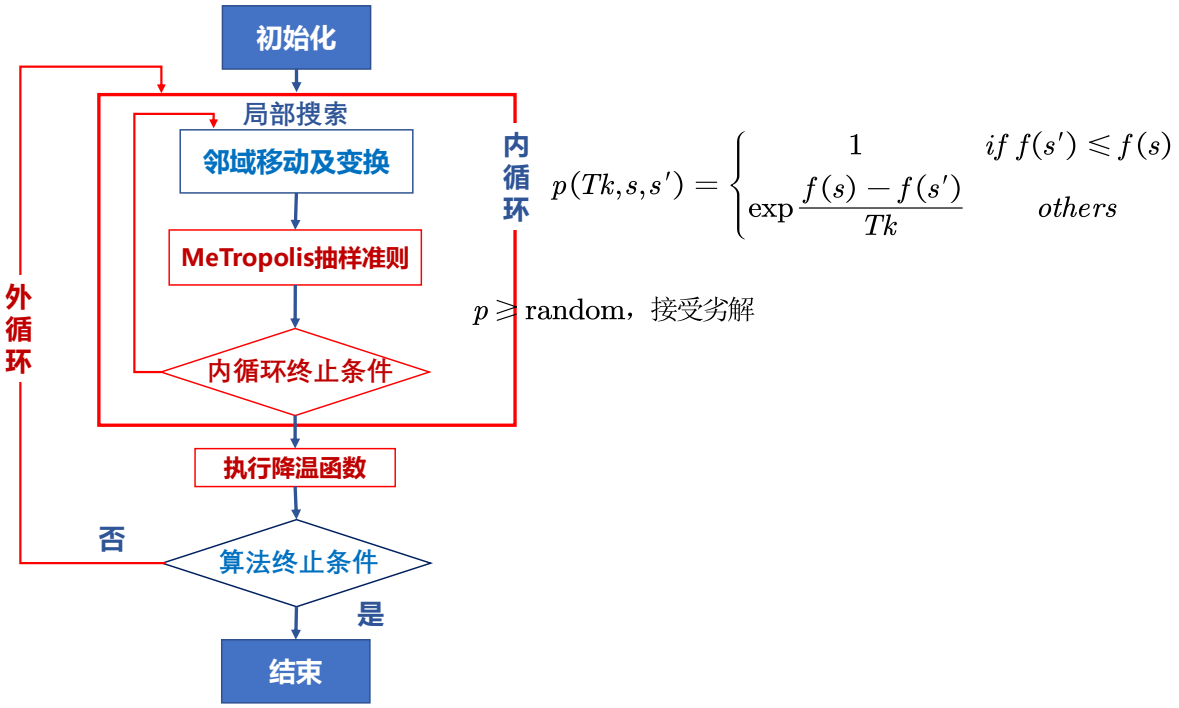


图 **3-3**　模拟退火算法流程图

3.1.2 实现的伪代码

算法 1 基本模拟退火算法

输入: $node\_id$ 节点 id 数组，$coord$ 节点坐标数组，$demand$ 需求数组，$vehicle\_cap$ 货车载量

输出: num_cars 车辆数目，$routes$ 车辆路径数组，obj 最终距离

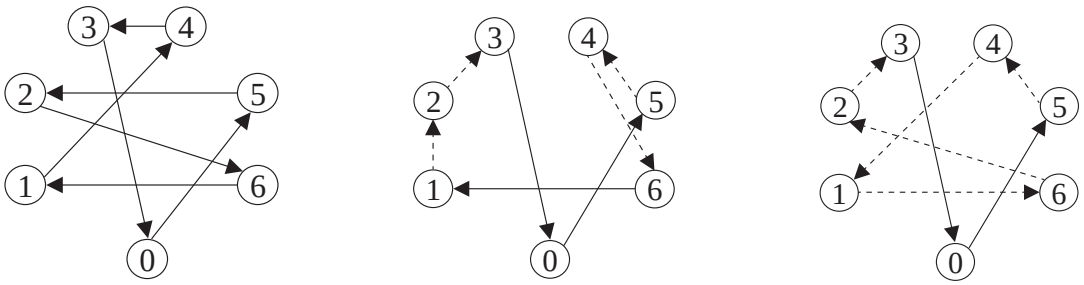
procedure SA($node\_id, coord, demand, vehicle\_cap$)
  $sol \leftarrow Sol()$
  while $Tk \geqslant T_{end}$ **and** $iter \leqslant max\_iter$ **do**
    for $i = 0 \rightarrow num\_T\_k$ **do**
      $new\_sol \leftarrow Sol()$
      $new\_sol.nodes\_seq \leftarrow Do\_Action(sol.nodes\_seq, action\_list)$
      $new\_sol.obj, new\_sol.routes \leftarrow Cal\_Obj(new\_sol.nodes\_seq)$
      $\Delta f \leftarrow new\_sol.obj - sol.obj$
      if $\Delta f < 0$ **or** $e^{-\frac{\Delta f}{Tk}} > random$ **then**
        $sol \leftarrow new\_sol$
      end if
      if $sol.obj < model.best\_sol.obj$ **then**
        $sol \leftarrow new\_sol$
        $sa.best\_sol \leftarrow sol$
      end if
    end for
    if $\alpha < 1$ **then** $Tk \leftarrow Tk \times \alpha$
    end if
    if $\alpha > 1$ **then** $Tk \leftarrow Tk - \alpha$
    end if
  end while
end procedure

其中产生的邻域的算子，参考文献[wuling]我们选取交换算子和逆转算子如图 3-4。

采用两种降温方式，线性降温和指数降温如式 (3.2)，以方便我们之后进行试验选取最优降温策略和初始温度。

$$\begin{cases} T_{k+1} = \alpha T_k, & \alpha < 1 \\ T_{k+1} = T_k - \alpha, & \alpha > 1 \end{cases} \tag{3.2}$$

(a) 例子



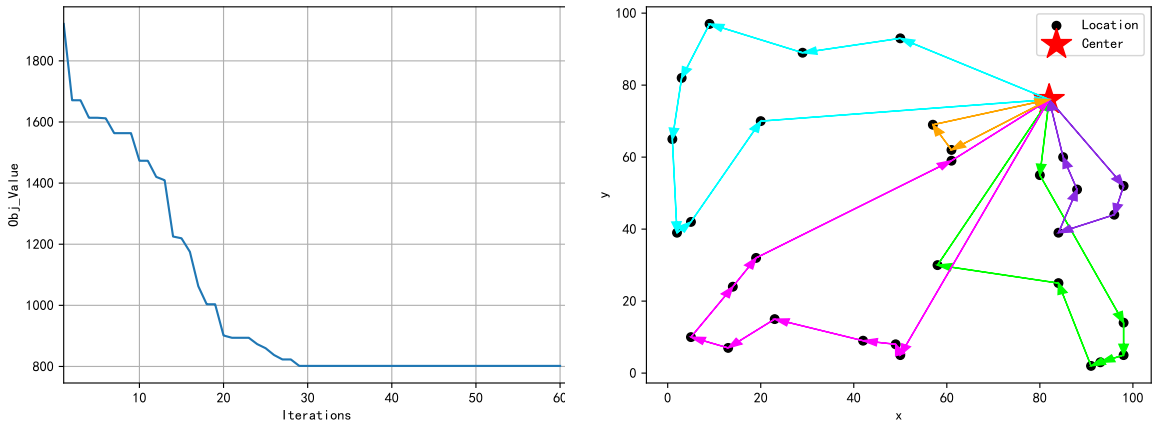
(b) 原始　　**(c)** 交换算子　　**(d)** 逆转算子

图 **3-4**　采用的邻域算子

随机化初始解，利用算法1计算 A-n32-k5 和 A-n80-k10，代码见附录 A，得到如表 3-1、图 3-5 和图 3-6。

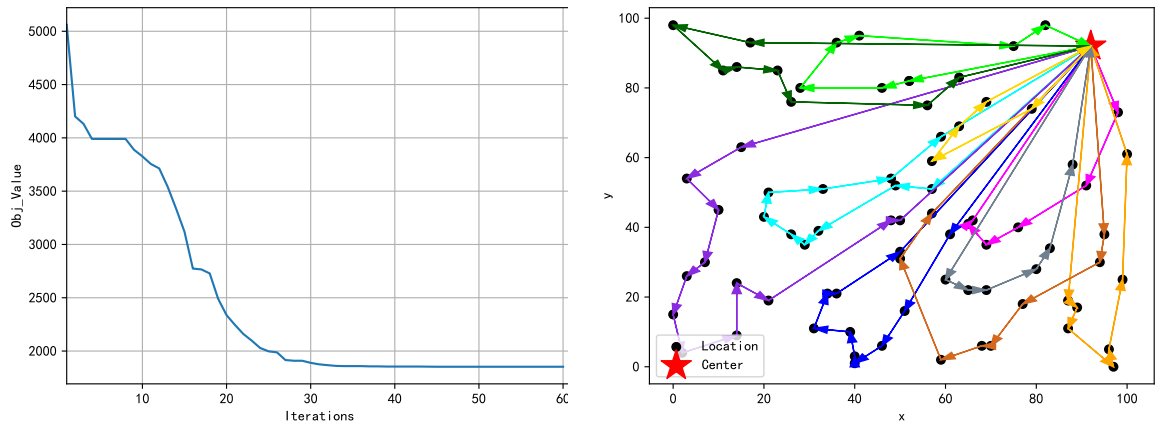
表 **3-1**　基本 **SA** 算法解决 **A-n32-k5** 和 **A-80-k10**

A-n32-k5	SA	vrp	A-80-k10	SA	vrp
num_cars	10	10	num_cars	10	10
best_obj	801	784	best_obj	1852	1764



(a) history_best-iter　　**(b)** A-n32-k5-routes

图 **3-5**　A-n32-k5 运行结果



(a) history_best-iter **(b) A-n80-k10-routes**

图 **3-6** **A-n80-k10** 运行结果

3.2 参数优化试验

SA 算法的参数包括初始温度 T_0，Metropolis 抽样次数 L 以及降温率 α，令其分别有 P_T, P_L 及 P_α 种选择，则参数组合共有 $P_T \times P_L \times P_\alpha$ 种。

一种参数组合 (P_T, P_L, P_α) 为问题的一个可行解，所有的参数组合构成问题的整个解空间，大小为 $P_T \times P_L \times P_\alpha$。

我们在实际过程中采用了两种参数优化的策略

基于经验的参数优化 利用经验给出几组可调的参数值，遍历选取其中最好的结果。

基于贝叶斯的参数优化 利用 python 集成的 bayes_opt 库，得到最好的参数值。

3.2.1 依据经验的参数优化

选择参数如式 (3.3)：

$$\begin{cases} T_0 = & \{50, 100, 200, 300, 500\} \\ \alpha = & \{0.8, 0.85, 0.9, 0.95\} \\ num_{Tk} = & \{1, 2, 3, 4, 5, 6, 7\} \end{cases} \tag{3.3}$$

其中抽样次数 L 为邻域大小的倍数 $\texttt{len}(actionlist) \times num_{Tk}$。如果要串行运算所有可能的情况，需要 $5 \times 4 \times 7 = 140$ 次，故而我们将其可能的参数组合划分，并行计算，代码如附录 B。

得到的较优的结果的参数组合为表 3-2，选取表 3-2中不同参数组合测试所有 vrp 问题数据，得到图 3-7。

表 **3-2**　经验参数优化

T_0	α	num_{Tk}	A-n32-k5 的结果	T_0	α	num_{Tk}	A-n32-k5 的结果
300	0.8	1	797.45129	200	0.8	6	797.4513
300	0.8	4	797.45129	200	0.9	4	797.4513
500	0.85	7	797.45129	50	0.8	6	798.9434
200	0.85	2	797.45129	100	0.85	2	798.9434

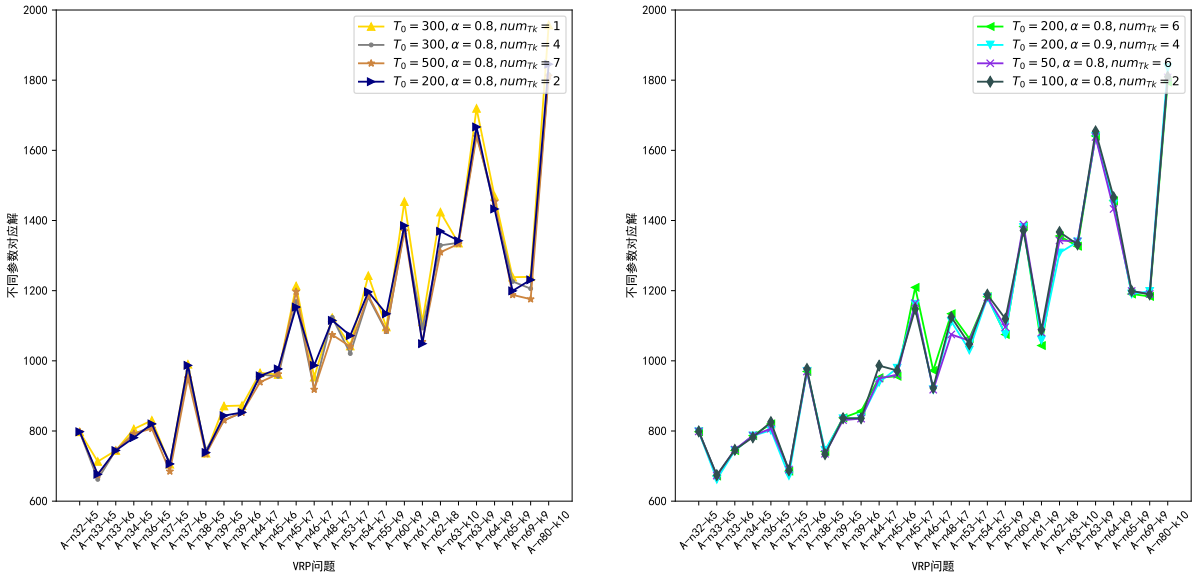


图 **3-7**　较优的几组经验参数对应 **vrp** 的解

3.2.2 依据贝叶斯的参数优化

由于 SA 算法本身存在随机因素，因此也可以将算法参数设定问题视为随机组合优化问题。这类问题往往不知道自变量与因变量的关系，但是只知道最后函数大概什么走向（也就是自变量对因变量大概的影响）。

而贝叶斯优化的好处在于只需要不断取样**2020Surrogates**，来推测函数的最大值。并且采样的点也不多。因此，模拟退火算法（SA）的参数设定非常适合利用贝叶斯优化的方法。

具体思路如下：

由于我们要优化的这个函数计算量太大，一个自然的想法就是用一个简单点的模型来近似，这个替代原始函数的模型也叫做代理模型，贝叶斯优化中的代理模型为**高斯过程**。假设我们对待优化函数的先验（prior）为高斯过程，经过一定的试验我们有了数据

（也就是 evidence），然后根据贝叶斯定理就可以得到这个函数的后验分布。

有了这个后验分布后，我们需要考虑下一次试验点在哪里进一步收集数据，因此就会需要构造一个 acquisition 函数用于指导搜索方向（选择下一个试验点），然后再去进行试验，得到数据后更新代理模型的后验分布，反复进行。

综上所述，贝叶斯优化的流程为：

算法 2 贝叶斯优化流程

输入: 迭代次数 $max\_iter = 30$，初始抽样点数 $init\_points = 5$

输出: 最优解对应参数 **x**，最优解 min z

1: **procedure** Bayesian Optimization($max\_iter = 30, init\_points = 5$)
2: **for** $t = 1 \rightarrow max\_iter$ **do**
3: Find x_t by **optimizing the acquisition function** over the GP: $\mathbf{x}_t = \arg\min_{\mathbf{x}} u(\mathbf{x}|D_{1:t-1})$
4: **Sample** the objective function:$y_t = f(\mathbf{x}_t) + \varepsilon_t$
5: Augment the data and update the GP:$D_{1:t} = \{D_{1:t-1}, (\mathbf{x}_t; y_t)\}$ and update the GP
6: **end for**
7: **end procedure**

高斯过程和 acquisition 函数见附录 D。

选择初始抽样点数目为 5，迭代次数为 30 次，按照距离从小到大排序，挑选较好的结果如表 3-3，代码见附录 C。

表 **3-3** 贝叶斯的参数优化

iter	target	T_0	α	num_{Tk}	iter	target	T_0	α	num_{Tk}
34	797.4482	85.34	0.9237	1.506	22	804.5052	47.63	0.9173	5.801
21	798.0846	29.35	0.8012	2.342	13	805.153	43.91	0.7872	4.06
16	801.2821	41.19	0.8072	3.08	26	805.153	20.51	0.8309	2.868
20	801.9246	36.39	0.9185	4.715	31	806.4516	43.55	0.8832	1.998
10	803.2129	27.1	0.86	4.974	35	807.7544	22.37	0.8769	5.267
14	803.2129	94.34	0.9004	4.89	23	808.4074	94.02	0.8189	2.402
3	803.8585	77.96	0.912	1.362					

选取表 3-3中不同参数组合测试所有 vrp 问题数据，得到图 3-8。

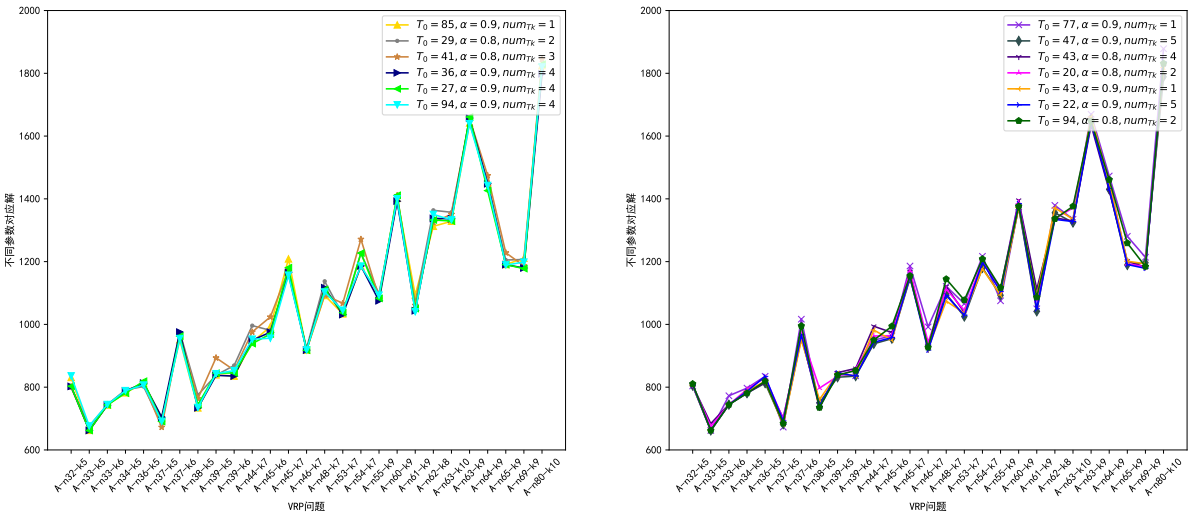


图 **3-8**　较优的几组贝叶斯优化参数对应 **vrp** 的解

3.3 实验结果

计算依据3.2.1节和3.2.2节得到的所有第 i 组参数对应的最优解 opt_{ij} 与 vrp 问题现已知最优解 opt_now_{ij} 的差值之和

$$minus = \sum_{j=1}^{N} opt_{ij} - opt\_now_{ij}$$

我们所求的最优参数组合为

$$\mathbf{x} = \arg\min_{\mathbf{x}} \ minus = \arg\min_{\mathbf{x}} \sum_{j=1}^{N} (opt_{ij} - opt\_now_{ij})$$

由上选取的最优组合如表 3-4：

表 **3-4**　两种参数优化结果

	T_0	α	num_{Tk}	所有路径总和	已经得到的的最优解
经验参数	500	0.85	7	28609.29953	28222
贝叶斯优化参数	47.63	0.9173	5.801	28582.5199	

可以看出贝叶斯优化的参数整体更优，但是总的距离分担到每一个 vrp 问题其实差距不大。

运行得到的结果如图 3-9，每一个 vrp 问题对应的路径长度如表 3-5 和表 3-6，每一个 VRP 问题的路径见附录 E。

表 **3-5** 经验参数 $T_0 = 500,\ \alpha = 0.85,\ num_{Tk} = 7$ 的 **VRP** 问题距离

x_list	y	x_list	y	x_list	y	x_list	y
A-n32-k5	797.4513	A-n38-k5	734.1847	A-n48-k7	1074.338	A-n63-k10	1333.255
A-n33-k5	674.1165	A-n39-k5	830.7456	A-n53-k7	1040.617	A-n63-k9	1639.088
A-n33-k6	744.2418	A-n39-k6	852.6008	A-n54-k7	1186.567	A-n64-k9	1454.067
A-n34-k5	792.5381	A-n44-k7	939.2543	A-n55-k9	1086.692	A-n65-k9	1187.933
A-n36-k5	806.7829	A-n45-k6	962.2644	A-n60-k9	1369.716	A-n69-k9	1175.815
A-n37-k5	684.9976	A-n45-k7	1197.484	A-n61-k9	1054.617	A-n80-k10	1811.23
A-n37-k6	950.8523	A-n46-k7	918.2779	A-n62-k8	1309.574		

表 **3-6** 贝叶斯优化参数 $T_0 = 47.63,\ \alpha = 0.9173,\ num_{Tk} = 5.801$

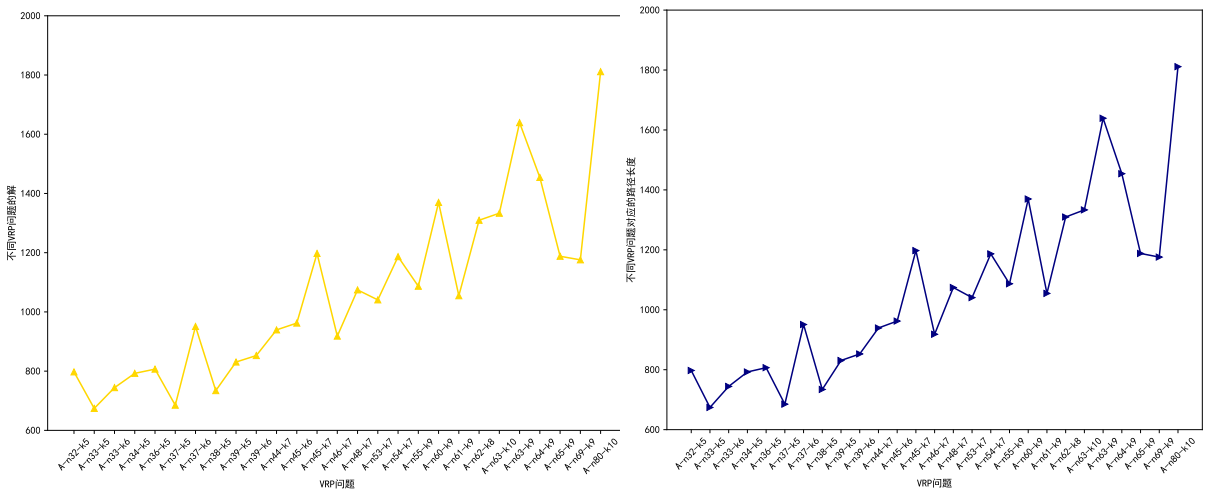
x_list	y	x_list	y	x_list	y	x_list	y
A-n32-k5	804.7916	A-n38-k5	749.889	A-n48-k7	1097.042	A-n63-k10	1324.492
A-n33-k5	662.1101	A-n39-k5	831.6498	A-n53-k7	1024.978	A-n63-k9	1652.328
A-n33-k6	744.2418	A-n39-k6	835.2518	A-n54-k7	1180.096	A-n64-k9	1430.108
A-n34-k5	780.9361	A-n44-k7	938.1813	A-n55-k9	1092.752	A-n65-k9	1189.281
A-n36-k5	812.9845	A-n45-k6	954.4739	A-n60-k9	1373.748	A-n69-k9	1196.691
A-n37-k5	682.3335	A-n45-k7	1146.909	A-n61-k9	1041.58	A-n80-k10	1788.94
A-n37-k6	956.8075	A-n46-k7	934.2195	A-n62-k8	1355.704		

第 4 章 改进的模拟退火算法

4.1 添加每条路径 TSP 优化的改进

我们构造的解是符合容量限制的，但是每一条路径对应的 TSP 问题并不一定是最短的，故而对与每一条路径的优化是我们改进的重要方向之一。

因为每一辆车走过的路径点数比较少，我们选择利用分支定界法求得每一辆车对应 TSP 问题的最优解来改进总的 VRP 问题。但是我们发现，在参数改进后，每条路径的



(a) 最优经验参数对应解　　**(b)** 最优贝叶斯参数对应解

图 **3-9**　两种最优参数对应解

TSP 都已经是最优解了，方法改进作用不大。

4.2 减小最大迭代次数的改进

如图 3-5a和图 3-6a，可以看出在只设置终止温度、不设置最大迭代次数情况下，只需要一半的迭代次数就收敛了。

故而，为了减少运行时间，最好设置最大迭代次数，在已经到达算法能达到的最优解时就停止降温。将最大迭代次数 max_iter 设置为所有 VRP 问题中需要迭代次数最长的收敛的次数，减少运行时间的同时保证收敛到当前的最优解。

运行时长如表 4-1和表 4-2：

表 **4-1**　$max\_iter = 60$ 的运行花费时间

VRP 问题	time	VRP 问题	time	VRP 问题	time	VRP 问题	time
A-n32-k5	7.21	A-n38-k5	12.61	A-n48-k7	26.04	A-n63-k10	54.12
A-n33-k5	8.35	A-n39-k5	14.24	A-n53-k7	33.33	A-n63-k9	48.25
A-n33-k6	8.15	A-n39-k6	12.93	A-n54-k7	34.90	A-n64-k9	55.93
A-n34-k5	8.36	A-n44-k7	20.17	A-n55-k9	34.13	A-n65-k9	56.61
A-n36-k5	10.63	A-n45-k6	20.36	A-n60-k9	44.54	A-n69-k9	61.42
A-n37-k5	11.46	A-n45-k7	21.13	A-n61-k9	51.47	A-n80-k10	92.57
A-n37-k6	12.33	A-n46-k7	23.17	A-n62-k8	43.62		

表 **4-2**　不设置迭代次数的运行时间

VRP 问题	time	VRP 问题	time	VRP 问题	time	VRP 问题	time
A-n32-k5	13.22	A-n38-k5	24.39	A-n48-k7	45.06	A-n63-k10	96.18
A-n33-k5	15.66	A-n39-k5	25.10	A-n53-k7	59.58	A-n63-k9	93.43
A-n33-k6	15.50	A-n39-k6	24.79	A-n54-k7	63.41	A-n64-k9	98.69
A-n34-k5	16.77	A-n44-k7	36.79	A-n55-k9	68.16	A-n65-k9	109.26
A-n36-k5	20.55	A-n45-k6	37.57	A-n60-k9	81.15	A-n69-k9	135.42
A-n37-k5	20.99	A-n45-k7	39.07	A-n61-k9	87.31	A-n80-k10	203.49
A-n37-k6	22.97	A-n46-k7	40.89	A-n62-k8	85.86		

二者对比如图 4-1，可见在增加最大迭代次数的限制后运行时间几乎缩短了一半。

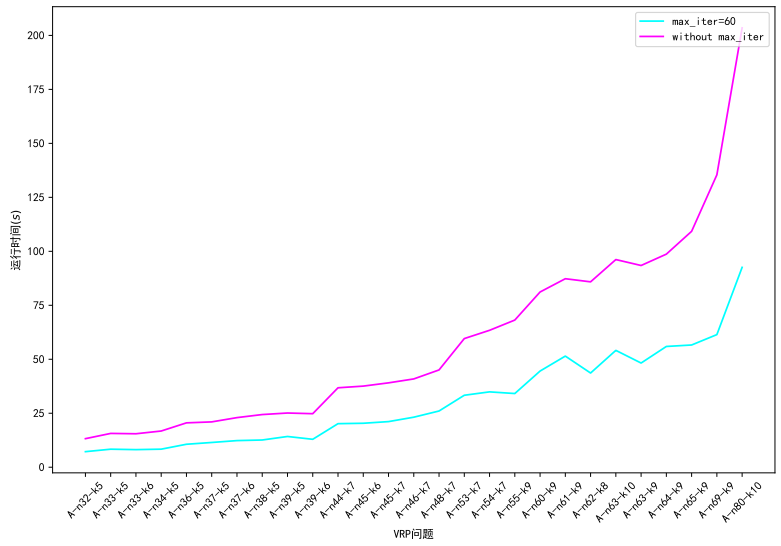


图 **4-1**　有无最大迭代次数 max_iter

附录 A 经典模拟退火算法代码

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 29 20:38:00 2022

@author: Hang
"""
#--------导入库--------#
import time
import math
import random
import numpy as np
import copy
import matplotlib.pyplot as plt
#--------定义类--------#
class Sol():
    '''
    nodes_seq    需求节点seq_no有序排列集合，对应TSP的解
    obj    目标值
    routes    车辆路径集合，对应CVRP的解
    '''
    def __init__(self):
        self.nodes_seq = None
        self.obj = None
        self.routes = None
class SA():
    '''
    best_sol    全局最优解，值类型为Sol()
    history_best_obj    每一代历史最优解
    T0  初始温度
    Te  终止温度
    '''
    def __init__(self ,T0=500, Te=0.001, delta_T=0.8):
        self.best_sol = None
        self.history_best_obj = []# 历史最优解
        self.T0 = T0
        self.Te = Te
        self.delta_T = delta_T
```

```
def ReadVrp(path):
    '''

    Parameters
    ----------
    path : str
        vrp文件路径

    Returns
    -------
    coord : list within array
        节点坐标
    demand : list
        需求量
    vehicle_cap : int
        最大载荷
    num_customer : int
        服务者数量

    '''
    f = open(path, "r", encoding='utf-8')
    vrp_list = [word for line in f for word in line.split()]
    f.close()
    vrp_coord = vrp_list[vrp_list.index('NODE_COORD_SECTION') + 1
                     : vrp_list.index('DEMAND_SECTION')]
    vrp_demand = vrp_list[vrp_list.index('DEMAND_SECTION') + 1
                     : vrp_list.index('DEPOT_SECTION')]
    vehicle_cap = int(vrp_list[vrp_list.index('CAPACITY')+2])
    best_obj = float(vrp_list[vrp_list.index('value:')+1][:-1])
    coord, demand = [], []
    for i in range( int(len(vrp_coord)/3) ):
        coord.append(np.array( [int(vrp_coord[3*i+1]), int(vrp_coord
            [3*i+2])] ))
        demand.append( int(vrp_demand[2*i+1]) )
    num_customer = i
    return coord, demand, vehicle_cap, num_customer

# 计算距离矩阵
def getDistance(coord):
    n = len(coord)
    distance = np.zeros((n, n))
```

```
    for i in range(n - 1):
        for j in range(i, n):
            distance[i, j] = np.linalg.norm(coord[i]-coord[j])
    distance += distance.T - np.diag(distance.diagonal())
    return distance

# 随机生成初始解
def Initial_Sol(node_seq):
    node_seq=copy.deepcopy(node_seq)
    random.seed(1)
    random.shuffle(node_seq)# random.shuffle():打乱顺序
    return node_seq

def Cal_Obj(nodes_seq, vehicle_cap, demand, Distance):
    num_vehicle = 0
    vehicle_routes = []# 车辆行驶路线
    route = []
    remained_cap = vehicle_cap
    distance = 0
    # 得到车辆行驶路线
    for node_no in nodes_seq:
        if remained_cap - demand[node_no] >= 0:
            route.append(node_no)
            remained_cap -= demand[node_no]
        else:
            vehicle_routes.append(route)
            route = [node_no]
            num_vehicle = num_vehicle + 1# 切割后所需车辆数
            remained_cap = vehicle_cap - demand[node_no]
    num_vehicle+=1# 切割后所需车辆数
    vehicle_routes.append(route)# 车辆行驶路线
    # 计算总距离
    for route in vehicle_routes:
        for i in range(len(route)-1):
            distance+=Distance[ route[i], route[i+1] ]
        distance+=(Distance[0,route[0]] + Distance[0,route[-1]])
    return num_vehicle, vehicle_routes, distance


# 邻域动作(算子)
def Create_Actions(n):
```

```
    actionList = []
    for i in range(n - 1):
        for j in range(i + 1, n):
            actionList.append([1, i, j])# 1-交换算子
    for i in range(n - 1):
        for j in range(i + 1, n):
            if abs(i - j) > 2:
                actionList.append([2, i, j])# 2-逆转算子
    return actionList

# 生成邻域
def Do_Action(nodes_seq,action):
    p = nodes_seq
    if action[0] == 1:  # 执行交换操作
        q = p.copy()
        q[action[1]] = p[action[2]]
        q[action[2]] = p[action[1]]
    if action[0] == 2:  # 执行逆转操作
        q = p.copy()
        if action[1] < action[2]:
            reversion = p[action[1]:action[2] + 1]
            reversion.reverse()
            q[action[1]:action[2] + 1] = reversion
        else:
            reversion = p[action[2]:action[1] + 1]
            reversion.reverse()
            q[action[2]:action[1] + 1] = reversion
    return q

#--------可视化--------#
# 画箭线
def Plt_Arrow(x_begin, y_begin, x_end, y_end, color):
    plt.arrow(x_begin, y_begin, x_end - x_begin, y_end - y_begin,
              length_includes_head = True,      # 增加的长度包含箭头部
                 分
              head_width = 2, head_length = 3, fc = color, ec = color)
# 连接节点
def Route(x0, y0, car, plt, color):
    Plt_Arrow(x0, y0, car[0][0], car[0][1], color)# 服务中心指向路径
        “起点”
    for i in range(len(car)-1):# 路径节点连接
```

```
            Plt_Arrow(car[i][0], car[i][1], car[i+1][0], car[i+1][1],
                color)
        Plt_Arrow(car[-1][0], car[-1][1], x0, y0, color)# 路径“终点”指向
            服务中心
# 退火过程可视化
def Plot_Obj(obj_list, path):
    plt.rcParams['font.sans-serif'] = ['SimHei'] # 中文显示
    plt.rcParams['axes.unicode_minus'] = False   # 显示('-')
    plt.plot(np.arange(1, len(obj_list) + 1), obj_list)
    plt.xlabel('Iterations')# 坐标轴
    plt.ylabel('Obj_Value')
    plt.grid()# 显示网格
    plt.xlim(1, len(obj_list)+1)
    plt.savefig(path+'iter.pdf')# 保存图片
    plt.show()
# 路径图可视化
def Draw_Map(coord, cars, path):
    x = [i[0] for i in coord]
    y = [i[1] for i in coord]
    colors = ['lime','cyan','blueviolet','fuchsia','orange','blue',
              'darkgreen','chocolate','slategray','gold','grey','peru'
                 ,
              'darkslategray','indigo','navy']
    plt.figure()
    plt.xlabel('x')
    plt.ylabel('y')
    for m,a in enumerate(cars):
        b = []
        for i in range(len(a)):
            b.append([])
        k = 0
        for j in range(len(a)):
            b[k].append(x[a[j]])
            b[k].append(y[a[j]])
            k += 1
        Route(x[0], y[0], b, plt, colors[m])
    plt.scatter(x, y, color = 'k', marker = 'o', label = 'Location')
    # 标出节点、服务中心
    plt.scatter(x[0], y[0], s = 500, color = 'r',marker='*', label = '
        Center')
    plt.legend()# 图例
```

```
    plt.savefig(path+'routes.pdf')# 保存图片
    plt.show()

def Run(path):
    # 读取数据
    coord, demand, vehicle_cap, num_customer = ReadVrp(path+'.vrp')
    Distance =  getDistance(coord)
    # 参数设置
    sa = SA(T0=500, Te=0.001, delta_T=0.8)
    Tk = sa.T0
    Te = sa.Te
    delta_T = sa.delta_T
    # 初始解
    sol = Sol()
    sol.nodes_seq = list(range(1,num_customer+1))
    sol.nodes_seq = Initial_Sol(sol.nodes_seq)# 随机初始解
    num_vehicle, sol.routes, sol.obj = Cal_Obj(sol.nodes_seq,
       vehicle_cap, demand, Distance)
    sa.best_sol = sol
    sa.history_best_obj = [sol.obj]
    # 邻域算子
    actionList = Create_Actions(num_customer)
    # 内循环邻域长度
    num_Tk = len(actionList)
    while Tk>=Te:
        for i in range(num_Tk):
            new_sol = Sol()# 当前解为初始解
            new_sol.nodes_seq = Do_Action(sol.nodes_seq, actionList[
               random.randint(0,num_Tk - 1)]) # random.randint(0,
               num_Tk - 1)
            num_vehicle, new_sol.routes, new_sol.obj = Cal_Obj(new_sol
               .nodes_seq, vehicle_cap, demand, Distance)
            delta_f = new_sol.obj-sol.obj
            if delta_f<0 or math.exp(- delta_f / Tk) > random.random()
               :
                sol = copy.deepcopy(new_sol)
            if sol.obj < sa.best_sol.obj:
                sa.best_sol = copy.deepcopy(sol)
        if delta_T < 1:# 定比例退火
            Tk = Tk * delta_T
        else:# detaT>=1时, 定步长退火
```

```
            Tk = Tk - delta_T
        sa.history_best_obj.append(sa.best_sol.obj)# 更新历史最优解
        print("temperature: %s, local_obj:%s best_obj: %s"
                  % (Tk,sol.obj,sa.best_sol.obj))
    Plot_Obj(sa.history_best_obj, path)
    Draw_Map(coord,sa.best_sol.routes , path)


start = time.time()
Run("data/A-n80-k10")
end = time.time()
print("运行时长%f"%(end-start))
```

附录 B 并行计算经验参数优化

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct  1 23:07:09 2022

@author: Hang
"""
import numpy as np
import math
from multiprocessing import cpu_count
from multiprocessing import Pool
from thread_SA import Run
T0 = [50, 100, 200, 300, 500] # 温度
alpha = [0.8, 0.85 ,0.9, 0.95] # 降温系数
num_Tk = [1, 2, 3, 4, 5 ,6, 7]  # Metropolis抽样次数

# 初始化参数解空间
P_t, P_alpha, P_num_Tk = len(T0), len(alpha), len(num_Tk)
S = []
for i in range(P_t):
    for j in range(P_alpha):
        for k in range(P_num_Tk):
            S.append(  [T0[i], alpha[j], num_Tk[k]] )


def howMany(T):
```

```
    res = []
    for i in range(T[0], T[1]+1 ):
        res.append( Run(S[i][0], S[i][1], S[i][2]) )
    return res

# 对整个数字空间N进行分段CPU_COUNT
def separateNum(N, CPU_COUNT):
    list = [[i * (N // CPU_COUNT), (i + 1) * (N // CPU_COUNT)-1] for i
        in range(0, CPU_COUNT)]
    return list


if __name__ == '__main__':
    N = len(S)
    # 多进程
    CPU_COUNT = cpu_count()  ##CPU内核数 本机为12
    # print(CPU_COUNT)
    pool = Pool(CPU_COUNT)
    sepList = separateNum(N, CPU_COUNT)
    print(sepList)
    result = []
    for i in range(CPU_COUNT):
        result.append( pool.apply_async(howMany, (sepList[i], )) )
    pool.close()
    pool.join()
    ans = [r for res in result for r in res.get()]
    best_index = S[ans.index(min(ans))]
```

附录 C 贝叶斯参数优化

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct  3 22:09:14 2022

@author: Hang
"""
from bayes_opt import BayesianOptimization
from thread_SA import Run

opt = BayesianOptimization(Run, {'T0': (20, 100),
```

```
                'delta_T': (0.75, 0.95),
                'num_Tk': (1, 6) } )

opt.maximize(n_iter=30, init_points=5)
print(opt.max)
```

附录 D 高斯过程和 acquisition 函数

高斯过程是多元高斯分布向无穷维的扩展，如果说高斯分布是随机变量的分布，则高斯过程是函数的分布，它可以由均值函数和协方差函数组成。

$$f(x) \sim \mathcal{GP}(m(x), k(x, x'))$$

在第 t 次试验后，我们有了数据 $\{x_{1:t}, f_{1:t}\}$，由于高斯过程上任意点 f_{t+1} 与之前的观测数据服从联合高斯分布，进一步可以得到预测分布

$$P(f_{t+1}|D_{1:t}, x_{t+1}) = \mathcal{N}(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1}))$$

我们已经可以根据高斯过程的后验分布对这个未知函数在任意位置的值做出预测，均值包括方差。

贝叶斯优化选择的搜索方向为预测值大的位置或者不确定性大的位置，这样才有可能搜到目标函数的最优解。

$$\arg\max_x u(x|D)$$

附录 E $T_0 = 47.63,\ \alpha = 0.9173,\ num_{Tk} = 5.801$ 的 VRP 结果

A-n32-k5

最优距离804.7916023618877
第1条路0->26->6->3->2->23->28->4->11->29->0
第2条路0->20->5->25->10->15->22->9->8->18->14->0
第3条路0->21->31->19->17->13->0
第4条路0->12->1->7->16->30->0
第5条路0->27->24->0

A-n33-k5

最优距离662.1100976492222
第1条路0->15->17->9->3->16->29->0
第2条路0->12->5->26->7->8->13->32->2->0
第3条路0->24->6->19->14->21->1->31->11->0
第4条路0->20->4->27->25->30->10->0
第5条路0->23->18->28->22->0

A-n33-k6

最优距离744.2417504053848
第1条路0->6->20->9->15->2->3->8->4->0
第2条路0->12->21->0
第3条路0->14->19->7->18->1->0
第4条路0->13->5->22->26->24->23->31->0
第5条路0->17->29->11->10->32->0
第6条路0->28->27->30->16->25->0

A-n34-k5

最优距离780.9360751738252
第1条路0->2->9->12->3->22->16->33->20->0
第2条路0->13->25->31->28->32->21->18->0
第3条路0->10->17->19->11->23->1->27->0
第4条路0->14->29->8->15->6->7->0
第5条路0->24->30->5->26->4->0

A-n36-k5

最优距离812.9845372769693
第1条路0->25->27->18->22->21->24->11->0
第2条路0->23->4->19->31->33->29->30->17->13->32->1->16->0
第3条路0->20->14->34->2->35->8->15->0
第4条路0->12->3->6->28->9->5->0
第5条路0->10->7->26->0

A-n37-k5

最优距离682.3334967199116
第1条路0->12->10->6->2->19->20->14->0
第2条路0->30->25->35->18->26->31->28->32->29->0
第3条路0->16->22->13->5->33->4->7->0
第4条路0->21->1->17->23->24->9->11->27->8->3->0
第5条路0->36->34->15->0

A-n37-k6

最优距离956.8074580710825
第1条路0->14->6->36->29->24->0
第2条路0->10->11->12->22->23->28->2->33->0

第3条路0->7->25->35->16->0
第4条路0->20->8->5->3->1->9->21->0
第5条路0->27->32->15->30->13->0
第6条路0->18->17->34->31->19->26->4->0

A-n38-k5

最优距离749.8889747660721
第1条路0->22->27->11->5->7->0
第2条路0->9->17->36->13->15->2->14->24->0
第3条路0->8->33->35->23->3->1->12->26->21->0
第4条路0->18->19->34->29->30->10->0
第5条路0->32->20->28->31->37->25->16->4->6->0

A-n39-k5

最优距离831.6498238065241
第1条路0->8->26->27->34->37->35->24->17->0
第2条路0->11->6->36->28->13->30->21->0
第3条路0->14->12->9->18->4->0
第4条路0->38->15->2->22->3->7->16->20->33->0
第5条路0->19->25->23->29->5->32->10->1->31->0

A-n39-k6

最优距离835.2517645944944
第1条路0->26->5->9->28->29->24->0
第2条路0->13->15->0
第3条路0->18->22->34->16->10->27->32->20->0
第4条路0->14->25->35->31->37->38->12->3->0
第5条路0->6->1->21->23->17->36->11->0
第6条路0->2->33->19->4->8->7->30->0

A-n44-k7

最优距离938.1812806893465
第1条路0->2->41->14->13->38->9->36->22->0
第2条路0->4->34->39->12->3->25->6->0
第3条路0->7->28->27->15->8->31->0
第4条路0->29->43->40->23->30->17->0
第5条路0->26->10->11->16->20->18->35->1->0
第6条路0->19->24->5->21->32->42->37->33->0

A-n45-k6

最优距离954.4739407506167
第1条路0->37->34->19->30->40->11->43->23->0
第2条路0->28->7->13->17->18->27->0
第3条路0->15->25->2->38->31->35->14->0
第4条路0->3->10->8->41->33->21->5->20->32->0
第5条路0->12->39->36->42->4->16->22->9->0
第6条路0->26->29->24->6->44->1->0

A-n45-k7

最优距离1146.9089683647403
第1条路0->35->34->26->4->12->0
第2条路0->10->15->25->23->17->38->13->0
第3条路0->9->22->30->37->1->42->8->0
第4条路0->2->6->28->3->11->43->41->27->21->0
第5条路0->32->5->36->19->29->31->0
第6条路0->39->14->33->44->24->0
第7条路0->18->7->16->20->40->0

A-n46-k7

最优距离934.2195100041765
第1条路0->38->4->30->2->37->32->36->0
第2条路0->45->6->43->13->26->14->23->0
第3条路0->8->33->16->24->15->20->21->17->0
第4条路0->28->41->29->31->34->40->1->27->19->0
第5条路0->12->10->7->22->39->0
第6条路0->11->42->35->18->25->44->3->5->0
第7条路0->9->0

A-n48-k7

最优距离1097.0422243198907
第1条路0->16->47->10->2->41->0
第2条路0->40->7->39->26->20->3->37->32->35->0
第3条路0->45->27->8->15->11->42->9->34->0
第4条路0->28->29->24->4->13->46->30->21->33->0
第5条路0->36->38->19->25->22->6->0
第6条路0->12->5->1->31->43->23->0
第7条路0->14->17->18->44->0

A-n53-k7

最优距离1024.9780330829383
第1条路0->38->18->40->26->10->49->29->44->30->1->0
第2条路0->37->2->36->50->43->23->19->15->32->16->9->0
第3条路0->3->34->11->24->41->17->22->28->0
第4条路0->39->5->14->13->52->21->25->0
第5条路0->47->7->12->48->42->45->4->0
第6条路0->51->46->8->35->27->0
第7条路0->31->20->6->33->0

A-n54-k7

最优距离1180.0960316145388
第1条路0->35->15->10->17->26->45->1->29->23->0
第2条路0->12->37->48->40->31->8->19->0
第3条路0->20->49->36->21->33->9->38->11->0
第4条路0->52->34->41->46->42->24->51->47->25->30->0
第5条路0->16->6->27->2->14->32->0
第6条路0->43->4->28->7->39->50->5->18->0
第7条路0->44->53->3->22->13->0

A-n55-k9

最优距离1092.7523049542342
第1条路0->29->50->44->24->52->23->54->13->0
第2条路0->4->7->42->31->20->26->0
第3条路0->14->46->17->34->3->37->0
第4条路0->41->28->27->19->22->30->0
第5条路0->12->10->5->53->40->16->38->32->0
第6条路0->43->35->9->49->39->47->0
第7条路0->25->48->18->21->8->0
第8条路0->6->45->1->0
第9条路0->15->11->51->2->33->36->0

A-n60-k9

最优距离1373.7483130879357
第1条路0->7->29->37->57->17->27->19->0
第2条路0->13->8->26->15->39->55->35->0
第3条路0->32->9->51->12->56->43->50->0
第4条路0->58->48->22->36->1->2->0

第5条路0->41->33->38->59->52->18->0
第6条路0->42->45->5->54->10->0
第7条路0->6->31->28->23->47->14->0
第8条路0->34->24->44->49->30->53->21->4->46->0
第9条路0->16->20->3->11->40->25->0

A-n61-k9

最优距离1041.5798867939454
第1条路0->24->10->59->45->37->30->42->33->0
第2条路0->39->44->1->16->48->18->35->34->0
第3条路0->58->29->36->21->27->56->47->51->0
第4条路0->50->55->7->23->14->0
第5条路0->43->32->26->4->22->12->13->0
第6条路0->15->38->3->0
第7条路0->17->57->52->31->11->60->28->0
第8条路0->49->2->46->54->5->6->25->0
第9条路0->19->53->8->41->20->40->0
第10条路0->9->0

A-n62-k8

最优距离1355.7038830753265
第1条路0->28->56->34->31->35->49->25->60->14->48->0
第2条路0->53->5->9->11->4->21->41->23->27->39->42->0
第3条路0->47->24->3->54->1->18->13->12->0
第4条路0->22->44->2->45->46->30->19->38->0
第5条路0->59->55->26->33->58->17->50->0
第6条路0->20->40->57->32->6->61->7->0
第7条路0->16->37->36->52->15->0
第8条路0->51->10->8->43->29->0

A-n63-k10

最优距离1324.4921667088724
第1条路0->31->51->58->22->11->34->47->44->50->0
第2条路0->40->53->16->37->54->56->0
第3条路0->24->55->33->60->21->0
第4条路0->61->46->23->7->0
第5条路0->20->28->32->57->62->17->0
第6条路0->12->6->27->35->8->29->0
第7条路0->15->49->45->39->1->41->9->2->14->0

第8条路0->25->18->5->3->42->19->10->59->0
第9条路0->13->26->4->38->30->0
第10条路0->48->36->43->52->0

A-n63-k9

最优距离1652.3281232245936
第1条路0->42->9->38->41->3->0
第2条路0->28->22->55->32->58->14->40->21->0
第3条路0->53->50->57->51->35->17->6->0
第4条路0->26->25->36->48->60->0
第5条路0->7->61->23->39->10->12->15->0
第6条路0->16->43->31->46->1->18->4->47->62->52->37->0
第7条路0->5->45->27->59->24->8->49->13->0
第8条路0->56->44->33->11->0
第9条路0->20->34->2->30->29->54->19->0

A-n64-k9

最优距离1430.1075485949573
第1条路0->17->1->35->26->21->7->40->36->39->44->14->27->0
第2条路0->48->42->32->61->13->52->56->41->10->0
第3条路0->15->11->57->30->3->53->45->0
第4条路0->6->25->29->33->0
第5条路0->24->60->16->8->46->47->63->2->58->0
第6条路0->9->4->54->5->34->23->0
第7条路0->19->37->59->31->50->49->55->0
第8条路0->38->43->28->12->22->18->0
第9条路0->20->51->62->0

A-n65-k9

最优距离1189.2812387356832
第1条路0->47->31->26->6->64->46->39->51->0
第2条路0->28->23->57->48->54->63->11->7->0
第3条路0->29->18->19->24->52->8->10->40->0
第4条路0->55->25->13->12->1->33->62->0
第5条路0->5->45->16->50->60->30->37->0
第6条路0->21->56->44->59->53->0
第7条路0->49->4->35->36->3->14->27->15->0
第8条路0->43->22->9->34->17->0
第9条路0->32->20->58->61->42->38->2->41->0

A-n69-k9

最优距离1196.690711991144
第1条路0->43->16->48->1->36->10->32->17->26->0
第2条路0->7->67->21->61->64->53->18->0
第3条路0->24->29->40->39->56->62->0
第4条路0->27->65->55->60->4->47->34->0
第5条路0->28->14->25->63->11->42->57->0
第6条路0->23->51->3->6->30->9->49->33->2->31->0
第7条路0->22->13->35->45->15->44->19->0
第8条路0->52->50->37->5->46->54->0
第9条路0->12->38->8->59->20->41->68->66->58->0

A-n80-k10

最优距离1788.9401505841874
第1条路0
->31->20->75->57->19->26->35->65->69->56->47->33->64->77->51->0
第2条路0->71->14->48->18->79->28->52->0
第3条路0->58->32->4->22->45->50->76->0
第4条路0->17->46->25->41->15->55->9->54->72->0
第5条路0->13->12->23->11->63->10->0
第6条路0->62->24->6->30->59->27->5->44->0
第7条路0->74->60->39->3->42->0
第8条路0->29->78->61->16->43->68->8->37->2->34->0
第9条路0->36->53->66->67->70->38->73->49->0
第10条路0->1->7->21->40->0

附录 F 代码清单

文件	作用
MySA.py	经典模拟退火算法
index_optimize.py	并行的经验参数优化
BayesianOptimization.py	贝叶斯参数优化
test_exp.py	经验参数优化试验
test_bayes.py	贝叶斯参数优化试验
time_improve.py	最大迭代次数条件 SA