求解车辆路径问题的改进扰动机制的ILS算法

侯彦娥[1,2] 孔云峰[1] 党兰学[2]
(河南大学黄河中下游数字地理技术教育部重点实验室 开封 475004)[1]
(河南大学计算机与信息工程学院 开封 475004)[2]

摘 要 针对车辆路径问题,提出一种改进的迭代局部搜索(ILS)算法。该算法基于破坏再重建(Ruin and Recreate)的思想,设计了一种新的扰动机制。扰动过程包含破坏和重建两个阶段,即先使用一种兼顾随机性和相关性的破坏方法对解进行破坏,并引入扰动因子控制解的破坏强度,然后再随机选择基本贪婪插入和改进贪婪插入算法完成解的修复。利用国际标准测试案例对常规扰动机制和改进后的扰动机制进行了测试,并与量子进化算法、蜂群算法进行了比较,实验结果表明改进后的ILS算法更加有效。

关键词 车辆路径问题,迭代局部搜索,破坏再重建,元启发

中图法分类号 TP301.6 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.1.057

Iterated Local Search Algorithm with Improved Perturbation Mechanism for Vehicle Routing Problem

HOU Yan-e[1,2] KONG Yun-feng[1] DANG Lan-xue[2]
(Key Laboratory of Geospatial Technology for the Middle and Lower Yellow River Regions, Ministry of Education, Henan University, Kaifeng 475004, China)[1]
(College of Computer and Information Engineering, Henan University, Kaifeng 475004, China)[2]

Abstract This paper proposed an iterated local search (ILS) algorithm with a new perturbation mechanism for vehicle routing problem. The perturbation mechanism is based on ruin and recreating principle, which includes destroying and repair procedures. The best local neighborhood solution is firstly destroyed by a method which takes randomness and correlation into account. In the destroying process, a perturbation factor is also introduced into the perturbation mechanism to control the strength of destroying. And then the destroyed solution is repaired by randomly selecting basic greedy insertion and regret greedy insertion algorithms. The experiments on the benchmark instances prove that the developed algorithm is more effective when compared with quantum evolutionary algorithm and artificial bee colony.

Keywords Vehicle routing problem, Iterated local search, Ruin and recreate, Metaheuristic

1 引言

车辆路径问题(Vehicle Pouting Problem, VRP)自1959年由Dantzing和Ramser提出以来,其解决办法已经在邮政投递、物资运送、物流配送、公交车调度等多个领域得到广泛的应用。VRP属于NP难问题[1],也是组合优化领域的研究热点问题之一。VRP求解算法中,精确算法仅能求解小规模的案例,较大规模案例的求解则依赖于(元)启发式算法。求解VRP的元启发算法大致可以分为轨迹法和种群法两类。轨迹法主要基于邻域进行搜索,并且在搜索过程中维持单一解,也称为单解类算法。这类算法包括模拟退火及其变种、禁忌搜索、迭代局部搜索、变邻域搜索、大规模邻域搜索等。种群法在搜索过程维持多个解,主要包括遗传算法、蚁群算法、粒子群算法和蜂群算法等群智能算法。这些元启发算法在VRP中得到了广泛的应用[2-4]。

迭代局部搜索(Iterated Local Search, ILS)是一种基于邻域搜索的元启发算法,它具有鲁棒性强、寻优效果好的特点[5]。ILS从初始解出发经过一次局部搜索得到当前局部最优解,而后迭代地执行扰动、局部搜索和接受新解等步骤,完成整个寻优过程。为了避免陷入局部最优,ILS算法借助扰动机制对当前局部最优解进行扰动产生新解,而后在新解上执行局部搜索。因此,扰动机制和扰动强度对算法的性能有直接的影响。扰动强度过大,算法易变成随机启动算法,使得解与最优解的偏差太大;扰动太弱则算法会过早陷入局部最优。常见的ILS扰动策略中,大多采用同一条路径或不同路径间特定长度路径段的插入或交换的方法实现扰动。文献[6]随机选择两条路径,生成包含2~4个节点的路径段并进行交叉改变当前解的结构。文献[7,8]通过在两条路径间执

到稿日期:2015-03-15 返修日期:2015-06-23 本文受国家自然科学基金项目(41401461),河南省教育厅自然科学重点项目(15A520009)资助。

侯彦娥(1980—),女,博士生,讲师,主要研究方向为智能算法、空间优化,E-mail:houyane@foxmail.com;孔云峰(1967—),男,博士,教授,博士生导师,主要研究方向为空间分析与应用、GIS分析与设计,E-mail:yfkong@henu.edu.cn;党兰学(1980—),男,博士,讲师,CCF会员,主要研究方向为智能算法、空间分析与优化,E-mail:danglx@foxmail.com(通信作者)。

行多次单个点的插入或交换来扰动当前解。文献[9]针对带取送货的装卸一体化问题，设计了两条路径间单点交换、两点交换和抛射链方式等多种扰动机制，执行过程中随机选择一种扰动机制对解进行扰动。文献[10]针对多周期车辆路径问题，随机选择一条路径，在路径内随机选择两个节点交换。

破坏再重建(ruin and recreate)是 Schrimpf[11] 提出的一种增强解的多样性、提高搜索质量的方法。通过对解进行一定程度的破坏后再进行修复，扩大邻域搜索空间以获得更好的解[12,13]。鉴于此，本文基于破坏再重建的思想设计了一种 ILS 扰动机制。通过该机制对当前局部最优解扰动后，为后续局部搜索提供更多可能的候选解，从而增加搜索的多样性。破坏再重建会导致邻域搜索空间扩大，进而会使得算法收敛过慢，因此引入扰动因子控制解的扰动强度。在国际基准测试案例集上进行了实验，发现与常规的扰动方法相比，采用改进后的扰动机制的 ILS 算法的求解质量有了大幅提升；与量子进化算法和蜂群算法的比较结果表明，改进后的 ILS 算法质量更优。

2 问题描述及定义

VRP 的类型很多，本文以带容量约束的车辆路径问题(Capacitated Vehicle Routing Problem，CVRP)为研究对象。CVRP 可以描述为：某个场站有 K 辆载重容量为 Q 的车辆，从该场站出发为 n 个配送点送货，完成货物配送后再回到场站。假设场站编号记为 0，配送点的编号记为 $\{1,2,\cdots,n\}$，则 $V=\{0,1,2,\cdots,n\}$。每个配送点的容量需求为 $q_i(i=1,2,\cdots,n)$，并且 $q_i \leqslant Q$(即要求每个配送点的容量需求不超过车的总容量)、$q_0=0$。配送点 i 和 j 之间的距离(成本)表示为 c_{ij}。要求每个配送点能且只能被服务一次，车辆从车场出发完成货物的配送后回到车场。目标是以最少的成本完成所有配送点的服务。

定义决策变量 x_{ijk} 和 y_{ik}。其中 $x_{ijk}=1$ 时表示车辆 k 经过配送点 i 后直接访问配送点 j；否则 $x_{ijk}=0$。若配送点 i 由车辆 k 服务则 $y_{ik}=1$，否则 $y_{ik}=0$。由此，问题的数学模型定义如下：

$$\text{Min. } Z=\sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=1}^{K}c_{ij}x_{ijk} \tag{1}$$

$$\text{s. t. } \sum_{k=1}^{K}y_{ik}=1,\ \forall i\in\{1,2,\cdots,n\} \tag{2}$$

$$\sum_{i\in V}x_{ijk}=y_{ik},\ \forall j\in V,k\in\{1,2,\cdots,K\} \tag{3}$$

$$\sum_{j\in V}x_{ijk}=y_{ki},\ \forall i\in V,k\in\{1,2,\cdots,K\} \tag{4}$$

$$\sum_{i\in V}y_{ik}q_i\leqslant Q,\ \forall i\in V \tag{5}$$

$$x_{ijk}\in\{0,1\},\ \forall i,j\in V\mid i\neq j,k\in\{1,2,\cdots,K\} \tag{6}$$

$$y_{ik}\in\{0,1\},\ \forall i\in V,k\in\{1,2,\cdots K\} \tag{7}$$

模型中，目标函数(1)以最小化行驶里程(运营成本)为优化目标，约束(2)限定每一个配送点仅能有一辆车服务，约束(3)和(4)保证每一个配送点仅被允许访问一次，约束(5)保证每辆车的负载不超过车辆的容量，约束(6)和(7)表示决策变量 x_{ijk} 和 y_{ik} 的取值范围。

3 算法设计

3.1 算法描述

本文设计的算法(I-ILS)包括生成初始解、扰动、局部搜索和接受新解 4 个基本步骤。其执行流程如下：

(1)读取数据文件，并使用节约法[14]构造问题的初始解 S_0；设 S^* 为全局最优解，并设置 $S^*=S_0$。

(2)使用局部搜索算子(见 3.3 节)对初始解 S_0 进行改进，得到局部最优解 S_1，并置 $S^*=S_1$。

(3)使用扰动机制(见 3.2 节)对当前局部最优解 S_1 进行扰动，得到扰动之后的解 S_2。

(4)继续使用局部搜索算子对扰动后的解 S_2 进行局部提升，得到新的局部最优解 S_2^*。

(5)根据接受规则(见 3.4 节)判定是否接受新的局部最优解 S_2^*，若满足接受规则，则置 $S_1=S_2^*$。

(6)若 S_2^* 的目标值优于 S_1 的目标值，则更新全局最优解 $S^*=S_2^*$。

(7)转到步骤(3)继续执行，直到满足循环终止条件。

(8)输出全局最优解 S^*。

3.2 破坏再重建的扰动机制

破坏再重建是对解的某一部分先进行破坏，而后再进行修复的过程。破坏通常是选择解中的一些节点将它们移除。重建则是修复解的过程，通过贪婪的或者其他方法插入移除的节点，这样修复后能够得到与原始解不同的解。破坏时移除节点的个数即是解的破坏程度，邻域搜索空间的规模将会随着解的破坏程度增加而增大。本文使用扰动因子 ρ 表示对解的破坏程度，其值为 0～1 之间的一个小数。例如 ρ 值为 0.1时，表示对问题规模的 10%的节点进行扰动。ρ 值越大，表明解破坏程度越大，搜索空间越大，解的修复时间也越长。为了避免搜索时间增长和算法收敛过慢，合理地控制解的破坏程度至关重要。

此外，破坏方法和重建方法也会影响到扰动的效果。最简单的破坏再重建过程使用一个破坏方法和一个修复方法[12]，也可以设计多个破坏方法和修复方法[13]。本文设计的 ILS 扰动方法包含一个破坏方法和两种修复方法，每次迭代过程中随机选择修复方法。通过若干次尝试后，将较好的扰动结果返回作为扰动后的解。扰动的实现过程如算法 1 所示。

算法 1 扰动方法 RIPerturb

```
输入：待扰动的解 S、扰动因子 ρ、尝试次数 trails
输出：扰动后的解 Sr
(1)S* =S;
(2) for(int i=0;i<trails;i++){
(3)     List<int> nodes=new List<int> ();
(4)     R* =eject_neighborhood(S* ,ρ,nodes);
(5)     type=selectInsertType();
(6)     S1 =inject_set(R* ,nodes,type);
(7)     if f(S1)< f(S* )
(8)       S* =S1;
(9)  }
(10) Sr=S* ;
(11)Return Sr
```

第(1)行获得要扰动解 S 的副本 S^*，后续循环对 S^* 进行节点的移除和插入操作。第(3)行初始化泛型列表 nodes 保存选择的节点。第(4)行调用 eject_neighborhood 方法移除解 S^* 中的若干个节点，得到部分解 R^*，并将移除的节点存入 nodes 集合。第(5)行调用 selectInsertType 方法得到节点插入规则 type。第(6)行使用 inject_set 方法将移除的节点按

照插入规则插入到部分解 R^* 中，得到新解 S_1。第(7)、(8)行是判定新解 S_1 和解 S^* 的目标函数值，若 S_1 优于 S^* 则接受 S_1 并将其作为下一次循环扰动的解；否则仍对解 S^* 进行扰动。第(10)行循环结束时将尝试过程中的最好解 S^* 作为扰动后的解 S_r。

破坏方法涉及到移除点的选择和解的破坏程度。常见的破坏方法有随机移除和相关度移除[12,13]等。随机移除是随机选择若干个点进行移除，因可选择节点组合较多，所以能增强解的多样性；关联度移除能够充分利用节点间的关系，但是如何定义相关度以及确定相应的参数需要针对特定的问题专门进行设计。随机移除是全局性的破坏方法，而相关度移除倾向于破坏局部解。

本文设计了一种新的破坏方法，在选择待移除点时兼顾随机性和节点之间的相关性。过程如下：首先随机选择一个节点；然后选择距离该节点较近的点，点的个数取值为 $\min\{2*dn, N\}$，其中 dn 为待移除节点数，N 为问题规模；再从邻近点集合中随机选择若干点，将它们从当前解中移除。这样做不仅考虑了节点间的距离关系，同时增加了选点的随机性。其实现过程如算法 2 所示。

算法 2 破坏方法 eject_neighborhood

```
输入：待破坏的解 S、扰动因子 ρ、待移除点集合 nodes
输出：破坏后部分解 R
(1)dn=getdeletenum(S,ρ);
(2)p=getrandomNode(S);
(3)pneighborlist=CreateNeighborList(S,p,dn);
(4)nodes. Add(p);
(5)cnt=1;
(6)while (cnt<dn){
(7)   j=randomSelect(pneighborlist);
(8)   if(!nodes. Contains(j)){
(9)      nodes. Add(j);
(10)     cnt++;}
(11)}//end while
(12) R=eject_nodelist(S,nodes);
(13)return R
```

第(1)行 getdeletenum 方法确定移除点的个数，假设解 S 的问题规模为 N，则移除节点个数 dn 为 $\lceil N*\rho \rceil$，其中 $\lceil\ \rceil$ 表示上取整；第(2)行随机选择解 S 中某条路径上的节点 p，p 不能是场站；第(3)行使用 CreateNeighborList 方法选择距离节点 p 最近的 $\min\{2*dn, N\}$ 个点，记录在 pneighborlist 中。第(4)行将节点 p 放入 nodes 列表中。第(5)—(11)行循环地从集合 pneighborlist 中随机选取 $dn-1$ 个不重复的节点，并加入 nodes 列表中。第(12)行使用 eject_nodelist 方法将 nodes 列表中的节点从当前解 S 中移除，得到部分解 R。

重建方法提供基本贪婪插入和改进贪婪插入两种修复方法。修复解时，随机选择其中一种方式完成移除节点的插入操作。

1)基本贪婪插入：对于每一个待插入的节点，在当前解中寻找能够以最小成本(增加的路径长度最小)插入的可行位置，完成插入操作。选择插入位置时，首先在待插入节点的邻域内寻找能够插入的路径以及最佳位置，若找不到能够插入的路径，则表示要开辟新的路径。这种方式能够保证每次待插入节点的成本最小，但不能保证全局最优。

2)改进贪婪插入：首先计算每个节点在当前解的路径中的最好插入和次好插入之间的成本差值(*regret*)，然后选择 *regret* 值最大的节点插入到最好位置。这种方式每次选择最好位置和次好位置的 *regret* 值最大的节点插入。

重建方法 inject_set 的实现过程如算法 3 所示。

算法 3 重建方法 inject_set

```
输入：待修复的解 R、待修复点集合 nodes、修复类型 type
输出：完整解 S
(1)int [] edges;
(2)if(type=0){
(3)   for(int i=0;i<nodes. count;i++){
(4)      findcheapestpos(R,nodes[i],edges);
(5)      R=insert_node(R,nodes[i],edges);
(6)}else{
(7)   while (nodes. count!=0){
(8)      for(int i=0;i<nodes. count;i++){
(9)         RegretCost[] rglist=computecost(R,nodes);
(10)        SortAsc(rglist);
(11)        t=getregretmax(edges);} //end for
(12)     R=insert_node(R,t,edges);
(13)     nodes. remove(t);}//end while
(14)}//end else
(15)S=R;
(16)return S
```

第(1)行保存节点插入的边的两个端点信息；当插入方式为基本贪婪方式(即 type=0)时，执行(3)—(5)行，针对 nodes 集中的每一个元素，首先在当前解中寻找其能够以最小成本插入的可行位置；若任何路径上都不能插入该元素，则开辟一条新的路径，将 *edges* 边中的元素均置为 0。然后，将元素插入到最廉价的位置，并更新解 R。第(7)—(13)行表示改进贪婪插入方式的插入过程。对于 nodes 中的每一个元素，计算其插入到当前解 R 中每一条路径上的成本以及最佳位置信息，这些信息保存在 RegretCost 结构数组 rglist 中。第(10)行对 reglist 按照成本升序排列。第(11)行使用 getregretmax() 方法首先计算每个节点的 *regret* 值，从中找到 *regret* 值最大的节点 t 和其要插入的边 *edges*。若 *regret* 值为一个非常大的整数，则将 *edges* 的值均设为 0。第(12)行将节点 t 插入解 R 中选定的最好位置，并更新解 R。第(13)行将执行插入的节点 t 从 *nodes* 中移除。继续执行程序，直到 *nodes* 集合为空。第(15)、(16)行完成修复后的解 S 的赋值和返回。

3.3 局部搜索算子

局部搜索过程使用 OnePointMove、TwoPointMove、2-opt 和 2-opt*[15,16]4 种操作算子对扰动后的解进行局部提升，寻找其局部最优解。

1)OnePointMove：即单点移动算子，包括路径间和路径内两种操作。其中，路径内单点移动也称为重定位(Relocate)，将路径 r_1 上的点 i 移除，然后插入到路径 r_1 上的其他位置得到新路径 nr_1。路径间单点移动是将路径 r_1 上的点 i 移除，然后将点 i 插入到路径 r_2($r_1 \neq r_2$)中，形成两条新路径 nr_1 和 nr_2。图 1(a)表示将路径 r_1 上的站点 P 插入到路径 r_2 上的站点 Q 后路径间的操作，图 1(b)表示同一条路径内站点 P 在站点 Q 后的重新定位操作。当发生单点移动时，P、Q 不能为同一个站点，且 P 不能为场站。

2)TwoPointMove：即两点交换算子，将路径 r_1 上的节点

i 与路径 r_2 上的节点 j 进行交换，交换后得到两条新路径 nr_1 和 nr_2。其也包括路径间和路径内两种操作。图 2(a)演示将路径 r_1 上的点 P 和路径 r_2 上的点 Q 进行交换后的效果；图 2(b)表示同一条路径上的 P、Q 两个站点进行互换。两点交换时，P、Q 不能为同一个站点，且 P、Q 均不能是场站。

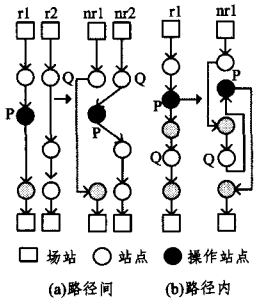

图 1 OnePointMove 操作

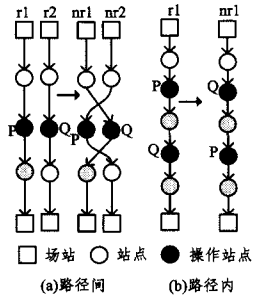

图 2 TwoPointMove 操作

3)2-opt：在一条路径内移除两条边，然后再增加两条边，同时线路的方向反转。图 3 中，路径 r_1 上的两条边 $P1P2$、$Q1Q2$ 被移除之后，将 $P2$ 到 $Q1$ 之间的节点序列方向反转，而后再增加两条新边 $P1Q1$、$P2Q2$ 形成一条新的路径 nr_1。

4)2-opt*：在两条不同的路径上分别移除一条边，而后通过增加两条新的边实现路径间的交叉。图 4 中，将路径 r_1 中的边 $P1P2$ 和 r_2 路径中的边 $Q1Q2$ 移除，然后增加两条新边 $P1Q2$、$Q1P2$ 完成两条路径上后半部分路径段的交叉操作，从而得到两条新路径 nr_1 和 nr_2。其操作如图 4 所示。

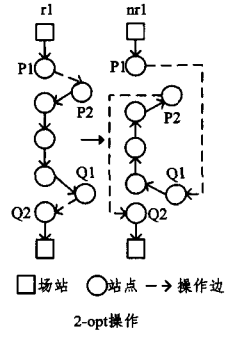

图 3 2-opt 操作

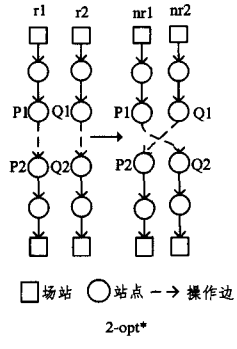

图 4 2-opt* 操作

这 4 个算子在搜索过程中，为了提高搜索效果，将搜索空间限定在节点的邻域范围内。计算与每一个节点距离较近的若干节点，这些节点的集合构成节点的邻域。在进行算子操作时，仅考虑在当前节点邻域范围内的节点之间的操作。例如路径间 P 和 Q 两点交换时，仅考虑 P 在 Q 的邻域范围或者 Q 在 P 的邻域范围中的情况，其他情况不予考虑，这样能够提高搜索效率。

3.4 接受规则

对于局部搜索产生的局部最优解，使用模拟退火的概率接受机制接受解。若经过局部搜索后得到的新局部最优解 S_l 优于当前局部最优解 S_b，则将 S_l 作为新的 S_b；若 S_l 差于 S_b，则以特定的概率接受解 S_l。接受规则定义如式(8)所示，其中 p 是一个[0,1)之间的随机小数，$T(T>0)$ 代表当前温度。

$$S_b=\begin{cases}S_l, & S_l<S_b\\ S_l, & e^{[-(S_l-S_b)/T]}>p, p\in[0,1)\end{cases} \tag{8}$$

4 实验验证

本文算法采用 Visual Studio 2010 环境中的 C# 来实现，测试环境为 PC 机，配置为 Intel R Core 2@3.06GHz CPU、4GB 内存、Windows 7 操作系统。算法的参数设置如下：迭代次数为 50，扰动尝试次数为 50，扰动因子 ρ 取值为 0.2，邻域大小设为 30，当问题规模不足 30 时，邻域大小为全邻域；模拟退火接受的初始温度为 2，降温系数为 0.9。测试案例集使用国际上公开的标准 CVRP 测试案例(http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/)。

4.1 算法测试

为了便于比较扰动机制改进后 ILS 算法的性能，使用基本 ILS 算法(BILS)和本文提出的改进算法(I-ILS)进行测试。BILS 和 I-ILS 具有相同的初始解、局部搜索和接受规则，二者仅在扰动策略上有所不同。其中，BILS 的扰动策略使用常规方法，即随机选择两条路径，分别在两条路径上随机选择长度为 $r\in[2,4]$的一段节点，进行交换得到一个可行解。

使用 BILS 和 I-ILS 分别求解标准案例集中 Set A、Set E 中具有小、中、大规模的 10 个案例，运行 10 次记录其最好解及其运算时间，比较结果如表 1 所列。其中，BKS 表示已知最好解的目标值，best 表示算法最好解的目标值，gap 是偏差值，表示算法最好解与已知最好解之间的偏差百分比，其计算公式为 $\frac{best-BKS}{BKS}\times 100\%$，黑体部分表示达到已知最好解。加星号(*)的已知最优解已被证明为最优解。后续表格中各列的含义同表 1。

表 1 10 个案例上改进前后的最好解和计算时间的比较

案例	BKS	BILS			I-ILS		
		best	time/s	gap/%	best	time/s	gap/%
A-n36-k5	799*	817	0.16	2.26	**799**	0.37	0.00
A-n45-k7	1146*	1183	0.18	3.23	**1146**	0.50	0.00
A-n55-k9	1073*	1098	0.23	2.33	**1073**	0.63	0.00
A-n65-k9	1177*	1201	0.36	2.04	**1177**	0.79	0.00
A-n80-k10	1763*	1826	0.44	3.57	1774	1.16	0.62
E-n23-k3	569*	**569**	0.10	0.00	**569**	0.19	0.00
E-n33-k4	835*	841	0.16	0.72	**835**	0.36	0.00
E-n51-k5	521*	544	0.58	4.41	**521**	0.63	0.00
E-n76-k7	682*	721	0.63	5.72	685	1.15	0.44
E-n101-k8	815	858	0.84	5.28	**815**	1.82	0.00
平均	—	—	0.37	2.96	—	0.76	0.106

从表 1 可以看出，在求解质量上 I-ILS 在 10 个案例上发现了 8 个最好解，其中 7 个为最优解，而 BILS 仅在 E-n23-k3 案例上找到了最优解；I-ILS 与已知最优解的平均 gap 值为 0.106%，而 BILS 与已知最优解的平均 gap 值为 2.96%。在求解时间上，I-ILS 在这 10 个案例上的平均求解时间比 BILS 增加了近 1 倍。尽管如此，对于规模最大的案例 E-n101-k8 而言，I-ILS 算法的求解时间仅有 1.82 s。虽然 I-ILS 平均求解时间比 BILS 有所增加，但是算法的求解质量得到了大幅度的提升。

进一步统计表 1 中 10 个案例的平均解和平均执行时间，并计算平均解与已知最好解的偏差值，结果如表 2 所列。其中，avg 代表案例运行 10 次的平均解，avgtime 代表平均运行时间，avggap 表示案例平均解与已知最好解的偏差值。

从表 2 可以看出，I-ILS 算法在 10 个案例上平均解的平均偏差值仅为 0.73%。其中，有 2 个案例的平均解达到已知

最好解，大部分案例的平均解与已知最好解的偏差值均在1%以下。尽管案例 A-n80-k10 和 E-n76-k7 的平均解偏差值大于1%，但仍在2%左右。从以上结果可以看出，I-ILS 算法整体具有较好的平均解，算法的稳定性较好。

表2　10个案例的平均解以及偏差值

案例	BKS	I-ILS		
		avg	avgtime/s	avggap/%
A-n36-k5	799*	803	0.38	0.50
A-n45-k7	1146*	1156.1	0.48	0.88
A-n55-k9	1073*	1078.1	0.62	0.48
A-n65-k9	1177*	1181.1	0.81	0.35
A-n80-k10	1763*	1793	1.17	1.70
E-n23-k3	569*	**569**	0.18	0.00
E-n33-k4	835*	**835**	0.35	0.00
E-n51-k5	521*	524.7	0.62	0.71
E-n76-k7	682*	696	1.14	2.05
E-n101-k8	815	820.3	1.79	0.65
平均	—	—	0.75	0.73

4.2　算法比较

为了进一步说明 I-ILS 算法的性能，使用 I-ILS 分别计算文献[17]和文献[18]的测试案例，并与文献[17]的量子进化算法(QEA)和文献[18]的蜂群算法(BCO)进行比较，结果如表3和表4所列。由于文献[17,18]并没有给出每个案例的运行时间，此处仅给出 I-ILS 算法的运行时间。

对于表3中的8个测试案例，QEA 发现了1个最优解，QEA 算法与已知最优解的平均偏差值为2.44%，而 I-ILS 算法发现了5个最优解，与已知最好解的平均偏差值仅有0.13%。针对表4中的8个案例，BCO 算法找到了2个最优解，与已知最优解的平均偏差值为4.64%；而 I-ILS 算法找到了7个最好解，其中6个最优解，与已知最好解的平均偏差值仅有0.0775%。可以看出 I-ILS 的寻优性能均显著优于 QEA 和 BCO。

表3　与量子进化算法[17]的对比

案例	BKS	QEA		I-ILS		
		best	gap/%	best	time/s	gap/%
A-n32-k5	784*	**784**	0	**784**	0.33	0.00
A-n33-k5	661*	667	0.91	**661**	0.31	0.00
A-n34-k5	778*	790	1.54	**778**	0.32	0.00
A-n39-k6	831*	837	0.72	833	0.41	0.24
A-n44-k6	937*	977	4.27	939	0.46	0.21
A-n46-k7	914*	942	3.06	**914**	0.49	0.00
A-n60-k9	1354*	1406	3.91	**1354**	0.74	0.00
A-n80-k10	1763*	1853	5.10	1774	1.16	0.62
平均	—	—	2.44	—	0.53	0.13

表4　与蜂群算法[18]的对比

案例	BKS	BCO		I-ILS		
		best	gap/%	best	time/s	gap/%
A-n33-k5	661*	678	2.57	**661**	0.31	0.00
A-n45-k7	1146*	1265	10.38	**1146**	0.50	0.00
A-n55-k9	1073*	1132	9.16	**1073**	0.63	0.00
A-n65-k9	1177*	1223	3.91	**1177**	0.79	0.00
A-n80-k10	1763*	1864	5.73	1774	1.16	0.62
E-n22-k4	375*	**375**	0.00	**375**	0.15	0.00
E-n33-k4	835*	**835**	0.00	**835**	0.36	0.00
E-101-k8	815	859	5.40	**815**	1.82	0.00
平均	—		4.64	—	0.715	0.0775

从表3和表4可以看出 I-ILS 能够充分利用破坏再重建的特点，扩大搜索的邻域空间，从而达到较好的搜索效果。

4.3　扰动因子 ρ 对算法的影响

扰动因子 ρ 的取值影响算法的求解质量和求解时间。ρ 值较小时，对解的扰动不明显；而 ρ 值过大时，对算法的求解时间将有影响，也可能导致破坏重建后解的质量变差。当 ρ 值在0.1，0.15，0.2，0.25，0.3 和 0.35 等6个值的情况下，测试 ρ 对算法的求解质量和求解时间的影响。

在标准案例集 A、E 中选择 A-n33-k5、A-n80-k10、E-n22-k4、E-n51-k5、E-n101-k8 等5个小、中、大规模的案例，每个案例从相同的初始解出发，其他参数保持相同，在不同的 ρ 值下分别运行10次程序，记录最好解及其计算时间。计算同一个案例在不同 ρ 值时的最好解与已知最好解之间的偏差值和计算时间，统计结果如图5和图6所示。

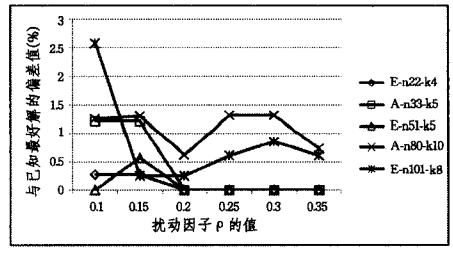

图5　不同扰动因子下案例最好解与已知最好解的偏差值

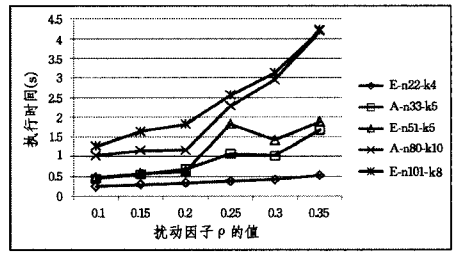

图6　不同扰动因子下5个案例的执行时间

从图5可以看出，当扰动因子 ρ 的值为0.2时，这5个案例与最好解的偏差均最小。对于 A-n33-k5、E-n22-k4 和 E-n51-k5 这3个中、小规模的案例，当 ρ 的取值继续增大时，算法仍能找到最优解。对于 A-n80-k10 和 E-n101-k8 这两个较大规模的案例，ρ 值为0.2时，偏差最小；继续增大 ρ 值，并没有使偏差值减少，究其原因在于 ρ 值增大时扰动时的可选择移除节点的组合增多，在有限次的迭代中并不能很有效地寻找到较好的移除节点组合。由此可以看出，当问题规模较大时，增加 ρ 的值并不能提高算法的求解质量。

图6中，随着 ρ 值的增加，案例的执行时间均呈现增加的趋势。当案例规模较小(比如 E-n22-k4)时，执行时间的增加趋势比较平缓；随着案例规模的增大，执行时间的增加趋势很快(比如 E-n101-k8)。案例 A-n33-k5 和 E-n51-k5 中，当 ρ 值为0.3时执行时间比 ρ 值为0.25时有所下降，也可以看出 ρ 值对算法执行时间的影响不仅与问题规模有关，而且还与案例中节点的分布有关。

综合考虑图5和图6，可以发现 ρ 值为0.2时，即扰动节点数为问题规模的20%时，算法基本能够在求解质量和求解时间两个方面达到平衡。

结束语　本文设计了一种基于破坏再重建的扰动机制对 ILS 算法进行改进。新的扰动机制在对解破坏时兼顾随机性和节点间的相关性，并通过扰动因子确定扰动的程度。分别使用改进的 ILS 算法和使用常规扰动机制的 ILS 算法测试

CVRP 的标准案例，测试结果表明改进后的 ILS 能够在增加少量计算时间的基础上大幅度提高求解的质量。将改进后的 ILS 算法与量子进化算法和蜂群算法进行了比较，进一步验证了改进后 ILS 算法的有效性。

参 考 文 献

[1] Toth P, Vigo D. The vehicle routing problem [M]// The vehicle routing problem: Society for Industrial and Applied Mathematic Philadelphia. 2002: 245-247

[2] Gendreau M, Potvin J Y. The Vehicle Routing Problem: Latest Advances and New Challenges [M]. New York: Springer, 2008: 143-169

[3] Laporte G. Fifty years of vehicle routing [J]. Transportation Science, 2009, 43(4): 408-416

[4] Vidal T, Crainic T G, Gendreau M, et al. Heuristics for multi-attribute vehicle routing problems: A survey [J]. European Journal of Operational Research, 2013, 231(1): 1-21

[5] Gendreau M, Potvin J Y. Handbook of Metaheuristics (2nd Edition) [M]. New York: Springer, 2010: 368-370

[6] Chen Ping, Huang Hou-kuan, Dong Xing-ye. A Multi-Operator Based Iterated Local Search Algorithm for the Capacitated Vehicle Routing Problem [J]. Journal of Beijing Jiaotong University (Natural Science), 2009, 33(2): 1-5 (in Chinese)
陈萍，黄厚宽，董兴业. 基于多邻域的车辆路径优化迭代局部搜索算法[J]. 北京交通大学学报(自然科学版)，2009，33(2)：1-5

[7] Penna P H V, Subramanian A, Ochi L S. An Iterated Local Search heuristic for the Heterogeneous Fleet Vehicle Routing Problem [J]. Journal of Heuristics, 2013, 19(2): 201-232

[8] Subramanian A, Penna P H V, Uchoa E, et al. A hybrid algorithm for the Heterogeneous Fleet Vehicle Routing Problem [J]. European Journal of Operational Research, 2012(221): 285-295

[9] Subramanian A, Drummond L M A, Bentes C, et al. A parallel heuristic for the Vehicle Routing Problem with Simultaneous Pickup and Delivery [J]. Computers & Operations Research, 2010, 37(11): 1899-1911

[10] Michallet J, Prins C, Amodeo L, et al. Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services [J]. Computers & Operations Research, 2014, 41(1): 196-207

[11] Schrimpf G, Scheider J, Stamm-Wilbrandt H, et al. Record breaking optimization results using the ruin and recreate principle [J]. Journal of Computational Physics, 2000, 159(2): 139-171

[12] Shaw P. Using constraint programming and local search methods to solve vehicle routing problems [C] // Principles and Practice of Constraint Programming — CP98. Springer, 1998: 417-431

[13] Pisinger D, Ropke S. A general heuristic for vehicle routing problems [J]. Computers & Operations Research, 2007, 34(8): 2403-2435

[14] Clarke G, Wright J. Scheduling of vehicles from a central depot to a number of delivery points [J]. Operations Research, 1964, 12(4): 568-581

[15] Groër C, Golden B, Wasil E. A library of local search heuristics for the vehicle routing problem [J]. Mathematical Programming Computation, 2010, 2(2): 79-101

[16] Hou Yan-e, Dang Lan-xue, Kong Yun-feng, et al. Design and Application of Metaheuristic Framework for School Bus Routing Problem [J]. Journal of Chinese Computer Systems, 2014, 35(7): 1625-1631 (in Chinese)
侯彦娥，党兰学，孔云峰，等. 校车路径问题元启发算法框架设计及应用[J]. 小型微型计算机系统，2014，35(7)：1625-1631

[17] Zhao Yan-wei, Peng Dian-jun, Zhang Jing-ling, et al. Quantum volutionary algorithm for capacitated vehicle routing problem [J]. System Engineering Theory and Practice, 2009, 29(2): 159-166 (in Chinese)
赵燕伟，彭典军，张景玲，等. 有能力约束车辆路径问题的量子进化算法[J]. 系统工程理论与实践，2009，29(2)：159-166

[18] Wang Zhi-gang, Xia Hui-ming. An artificial bee colony algorithm for the vehicle routing problem [J]. Computer Engineering and Science, 2014, 36(6): 1088-1095 (in Chinese)
王志刚，夏慧明. 求解车辆路径问题的人工蜂群算法[J]. 计算机工程与科学，2014，36(6)：1088-1095

(上接第 250 页)

[8] Bellot P, Bonnefoy L, Bouvier V, et al. Large Scale Text Mining Approaches for Information Retrieval and Extraction [M] // Innovations in Intelligent Machines. 2014: 3-45

[9] Zhu Ye-xing, Li Yan-ling, Cui Meng-tian. Clustering Algorithm CARDBK Improved from K-means Algorithm [J]. Computer Science, 2015, 42(3): 201-205 (in Chinese)
朱烨行，李艳玲，崔梦天. 一种改进 K-means 算法的聚类算法 CARDBK[J]. 计算机科学，2015，42(3)：201-205

[10] Brecheisen S, Krieegel H P, Kroger P, et al. Visually mining through cluster hierarchies [C] // International Conference on Data Mining. Lake Buena Vista, FL, 2004: 400-412

[11] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [C] // Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, 2004(6): 137-150

[12] Lee K, Lee Y, Choi H. Parallel Data Processing with Map-Reduce: A Survey [J]. ACM SIGMOD Record, 2011, 40(4): 11-20

[13] Kanungo T, Mount M D, Neanyahu N S, et al. A Local Search Approximation Algorithm for k-Means Clustering [J]. Computational Geometry Theory & Applications, 2004, 28(2): 89-112

[14] Xiong Zhong-yang, Chen Ruo-tian, Zhang Yu-fang. Effective method for cluster centers' initialization in K-means clustering [J]. Application Research of Computers, 2011(11): 4188-4189 (in Chinese)
熊忠阳，陈若田，张玉芳. 一种有效的 k-Means 聚类中心初始化方法[J]. 计算机应用研究，2011(11)：4188-4189

[15] Younis O, Fahmy S. HEED: A Hybrid, Energy-efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks [J]. IEEE Transactions on Mobile Computing, 2004, 3(4): 366-379