第 31 卷第 8 期 系统工程理论与实践 Vol.31, No.8
2011年 8 月 Systems Engineering — Theory & Practice Aug., 2011

文章编号: 1000-6788(2011)08-1419-10 中图分类号: F502 文献标志码: A

求解 TSP 问题的最近邻域与插入混合算法

饶卫振, 金 淳, 黄英艺

(大连理工大学 系统工程研究所, 大连 116024)

摘 要 研究了求解旅行商问题 (TSP) 的构建型启发式算法中的最近邻域算法和插入算法的特点, 集最近邻域算法求解速度快、插入算法求解质量高的优点, 提出了一种最近邻域与插入混合算法. 分析了混合算法的合理性、复杂度及参数取值, 并分别采用以上三种算法求解了 TSPLIB 标准库中多个算例, 结果表明混合算法的求解速度接近最近邻域算法, 对城市数量小于 1000 的小规模 TSP 问题的求解质量与插入算法相当, 而对大规模 TSP 问题的求解质量明显优于插入算法.

关键词 旅行商问题; 混合算法; 最近邻域算法; 插入算法

Hybrid algorithm of the nearest neighbor and insertion for the traveling salesman problem

RAO Wei-zhen, JIN Chun, HUANG Ying-yi

(Institute of Systems Engineering, Dalian University of Technology, Dalian 116024, China)

Abstract This paper presented a new hybrid construction heuristic algorithm (NN&IN) -combination of the nearest neighbor algorithm (NN) and the insertion algorithm (IN), based on analyzing properties of NN and IN algorithm for solving the traveling salesman problem (TSP). Moreover, we studied the rationality, complexity, parameter setting of the hybrid algorithm and solved many benchmark instances from TSPLIB using NN, NN&IN, IN algorithm respectively. Computational results show that NN&IN algorithm whose running time is close to NN can yield solutions as good as IN for small TSP (the number of city is less than one thousand) and better solutions for large TSP than IN, and prove that the efficiency and effectiveness of the proposed hybrid algorithm outperform both NN algorithm and IN algorithm.

Keywords traveling salesman problem; the hybrid algorithm; the nearest neighbor algorithm; the insertion algorithm

1 引言

TSP 问题 (traveling salesman problem) 是典型的容易描述难以求解的 NP-Hard 组合优化问题 [1], 因其求解的困难性而成为各种算法测试性能的最佳平台. 目前, 求解 TSP 的算法可分为精确算法 (exact algorithms) 和启发式算法 (heuristic algorithms) 两类. 精确算法能够确保得到精确最优解, 但要以巨大的甚至是不可行的时间为代价. 启发式算法又称为近似算法 (approximate algorithms), 是在考虑计算时间成本的基础上, 以牺牲找到最优解的保证为代价, 在合理时间内找到近优解甚至最优解的算法. 启发式算法可细分为构建型算法 (construction algorithms) 和改进型算法 [2](improvement algorithms). 构建型算法是从最初的空解开始, 迭代的添加解成分到解中, 直到完全构建解. 目前比较典型的构建型算法有随机构建生成法、随机贪婪算法、最近邻域算法、插入算法、生成树法、节省法等 [3]. 改进型算法是在构建型算法构造出的解基础上, 采用优化法则进一步改进解质量的算法. 比较常见的改进型算法有 k- 交换法 [3]、Lin-Kernighan 搜索法 [4−5]、Ejection-chain 算法 [6]、模拟退火法 [7]、禁忌搜索法 [8−9]、遗传算法 [10]、神经网络法 [11]、蚁群算法 [12−14]、粒子群算法 [15] 以及这些算法的混合算法或改进算法 [10,16] 等.

收稿日期: 2009-12-08
资助项目: 国家自然科学基金 (70571008); 辽宁省自然科学基金 (20062184)
作者简介: 饶卫振 (1981–), 男, 汉, 江西省丰城人, 博士研究生, 研究方向: 物流系统优化、算法分析; 金淳 (1963–), 男, 汉, 辽宁省大连人, 教授, 博士生导师, 研究方向: 物流系统优化、物流仿真.

改进型算法与构建型算法相比具有更高的求解质量, 但求解时间较长, 而且很多改进型算法涉及到众多的参数取值 [14], 或者具有复杂的执行规则, 这使改进型算法的应用受到一定限制. 在实践应用中, 有时选择规则明确、高效的构建型算法求解可能是更好的选择; 另外在部分改进型算法中, 使用性能良好的构建型算法产生高质量的初始解可以提高改进型算法的求解速度, 甚至改进求解的最终质量 [17]. 本文结合两种典型的构建型算法: 最近邻域算法求解速度快、插入算法求解质量高的优点, 提出了一种最近邻域与插入混合算法. 当前, 设法将不同算法的优点加以融合, 以产生综合特性更加优良的混合算法是人们关注的热点研究领域 [18].

2 TSP 问题的描述及两种构建型启发式算法

2.1 TSP 问题描述

TSP 问题可描述为: 一个推销员要旅行 n 个城市, 且任意两个城市之间距离已知, 求推销员旅行每个城市一次且仅一次, 并回到起点城市的最短路径. 该问题可以用多种数学模型描述, 下面是其中的一种:

设 n 个城市之间的距离矩阵为 $D=(d_{ij})_{n\times n}$, 其中, $\begin{cases} d_{ij}, & \text{代表城市 } i \text{ 至城市 } j \text{ 的距离}(i\neq j),\\ d_{ij}=M, & M\text{为足够大的数}(i=j). \end{cases}$

设 $n(n>3)$ 个城市分别标号为 $\{1,2,\cdots,n\}$, 一个 TSP 问题的可行解就是 n 个城市标号的环状排列. 令 $\pi[n]$ 为存放 n 个城市的任意排列的一维数组, 那么可行解的旅行总距离 (路径总长度) 可表示为:

$$f(\pi)=\left(\sum_{i=0}^{n-2} d_{\pi[i]\pi[i+1]}\right)+d_{\pi[n-1]\pi[0]} \tag{1}$$

显然, 使得 $f(\pi)$ 取得最小值的排列 π 为 TSP 问题的最优解 [2].

TSP 问题主要有三种分类方式 [5]: 1) 按距离矩阵 D 是否为对称矩阵, 分为对称 TSP 问题和非对称 TSP 问题; 2) 按任意三个城市间距离值是否满足三角不等式, 分为三角不等式 TSP 问题和非三角不等式 TSP 问题; 3) 按 n 个城市的坐标位置之间距离是否完全符合欧几里德平面规则, 即任意两个城市之间距离, 分为欧几里德 TSP 问题和非欧几里德 TSP 问题. 欧几里德 TSP 问题必为对称三角不等式 TSP 问题, 反之不成立.

相邻程度 r 的含义: 与城市 i 相邻程度为 r 的城市 j 是指城市 j 是城市 i 第 r 近的城市. 比如 j 是城市 i 相邻程度为 1 的城市, 那么 j 是离城市 i 最近的城市, 同理当 j 是离城市 i 最远的城市 $(i\neq j)$, 则称城市 j 是城市 i 相邻程度为 $n-1$ 的城市.

2.2 求解 TSP 问题的两种构建型启发式算法

在此介绍两种与本文相关的构建型启发式算法: 最近邻域算法 (the nearest neighbor algorithm, 简称 NN) 和插入算法 (the insertion algorithm, 简称 IN).

NN 简要步骤如下 [3]:

Step 1 任选一城市 j 为起点, 设 $k\leftarrow j$, 令 Unvisitedcity $=\{1,2,\cdots,n\}\backslash\{j\}$;

Step 2 设 $d_{km}=\min\{d_{ki}|i\in$Unvisitedcity$\}$, Unvisitedcity = Unvisitedcity $\backslash\{m\}$, 令 $k\leftarrow m$;

Step 3 如果 Unvisitedcity 为非空集, 重复 Step 2; 否则, 结束构造, 得到最终解.

NN 任意选择起点城市, 构造路径时每一次都选择最近的未旅行的城市作为下一个旅行城市, 直到旅行完所有的城市再回到起点城市, 得到一个解.

IN 算法求解步骤如下 [3]:

Step 1 任选三个相互较近城市 i、j、k 为形成圈 L: $i-j-k-i$, 令 Unvisitedcity $=\{1,2,\cdots,n\}\backslash\{i,j,k\}$;

Step 2 根据规则选取集合 Unvisitedcity 中元素城市 h, 确定适当位置 (致使圈 L 的长度增加最小的位置) 将 h 插入圈 L, 更新圈 L, Unvisitedcity = Unvisitedcity $\backslash\{h\}$;

Step 3 如果 Unvisitedcity 为非空集, 重复 Step 2; 否则, 结束构造, 得到最终解.

IN 算法从任意三个位置相互较近的城市组成的圈开始, 每步都在当前圈中选择使路径长度增加最少的位置插入一个新城市得到一个新圈, 重复此规则, 直到构成一完整的可行解. 根据 Step 2 中选择待插入城市 h 的不同规则, IN 算法可分为最远插入 (farthest insertion)、最近插入 (nearest insertion)、最节省插入 (cheapest insertion) 和随机插入算法 (random insertion).

3 最近邻域与插入混合算法设计

3.1 最近邻域与插入混合算法

NN 算法的优点是求解速度快, 缺点是求解质量低; 而 IN 算法的优点是求解质量高, 缺点是求解速度慢. 因此, 如果将 NN 和 IN 科学的结合, 就有可能构建一种不但求解速度快而且求解质量高的混合算法. 最近邻域与插入混合算法 (hybrid algorithm of the nearest neighbor and insertion, 简称 NN&IN) 正是基于该思路的一种混合构建型启发式算法.

NN 算法的缺点是由于每一步决策的短视行为, 导致在构造路径的后阶段很可能要相互连接距离较远的城市. 那么 NN 算法构建一个城市数量为 n 的 TSP 问题的可行解, 且路径中任意两相邻城市均在彼此相邻程度 r ($2\le r\le n-1$) 范围内 (包括相邻程度 r) 的概率 P 是多少? 设 NN 算法构造的第 k ($1\le k\le n$) 步的当前城市为 i (即第 $k-1$ 步选择旅行的城市为 i, 当 k=1 时, i 为起点城市), 城市 i 的最邻近未旅行城市 j 是城市 i 相邻程度 r 范围内的城市的概率表示为 p_k, p_k 等于在前 $k-1$ 步旅行了 $k-1$ 个城市后, 剩下的 $n-k$ 个未旅行城市 (当前城市 i 除外) 中至少有一个城市与当前城市 i 的相邻程度在 r 范围内的概率. 设 f 为当城市 j 在城市 i 的相邻程度 r 范围内时, 城市 i 也在城市的 j 相邻程度 r 范围内的概率. 则在 NN 算法的第 k 步将当前城市 i 连接至城市 j, 且使城市 i、j 均在彼此的相邻程度 r 范围内的概率 Δp_k 可由公式 (2) 计算.

$$\Delta p_k = p_k \cdot f \tag{2}$$

其中 p_k、f 的取值如公式 (3)、(4) 所示, 其中公式 (4) 中的三角不等式 TSP 问题 f 等于 1 为近似取值.

$$p_k = \begin{cases} 1, & k \le r \\ 1 - \dfrac{C_r^r \cdot C_{n-1-r}^{k-1-r}}{C_{n-1}^{k-1}}, & k \le n-2 \text{ 且 } k > r \\ \frac{r}{n-1}, & k = n \text{ 或 } n-1 \end{cases} \tag{3}$$

$$f = \begin{cases} 1, & \text{三角不等式 TSP 问题} \\ \dfrac{r}{n-1}, & \text{非三角不等式 TSP 问题} \end{cases} \tag{4}$$

因此概率 P 可由公式 (5) 计算:

$$P = \prod_{k=1}^{n} \Delta p_k \tag{5}$$

当 f=1, n=10 时, Δp_k 随 k、r 变化的取值如表 1 所示, 在 f=1 时 P 随 n、r 变化的取值见表 2. 显然, Δp_k 随 k 单调递减, 随 r 单调递增; P 随 n 单调递减, 随 r 单调递增.

表 1　f=1, n=10 时 Δp_k 随 k, r 变化表

Δp_k	n=10, r=2	n=10, r=3	n=10, r=4	n=10, r=5	n=10, r=6	n=10, r=7	n=10, r=8	n=10, r=9
Δp_1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Δp_2	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Δp_3	0.972	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Δp_4	0.917	0.988	1.000	1.000	1.000	1.000	1.000	1.000
Δp_5	0.833	0.952	0.992	1.000	1.000	1.000	1.000	1.000
Δp_6	0.722	0.881	0.960	0.992	1.000	1.000	1.000	1.000
Δp_7	0.583	0.762	0.881	0.952	0.988	1.000	1.000	1.000
Δp_8	0.417	0.583	0.722	0.833	0.917	0.972	1.000	1.000
Δp_9	0.222	0.333	0.444	0.556	0.667	0.778	0.889	1.000
Δp_{10}	0.222	0.333	0.444	0.556	0.667	0.778	0.889	1.000
P	0.006	0.041	0.120	0.243	0.403	0.588	0.790	1.000

表 2 f=1 时 P 值随 n、r 变化表

	10	50	100	500	1000
5	2.430E-01	1.047E-06	4.076E-13	1.579E-63	2.958E-126
10	-	1.212E-03	3.989E-07	3.914E-34	1.314E-67
15	-	1.680E-02	6.386E-05	1.917E-23	2.594E-46
20	-	6.775E-02	9.143E-04	6.703E-18	2.760E-35
50	-	-	1.531E-01	1.638E-07	1.056E-14

Helsgaun[5] 提到一个有 532 个城市的 TSP 问题, 其最优路径中的任意两相邻城市相邻程度最大值为 22, 并且平均相邻程度为 2.4(最完美的平均相邻程度为 1.5, 即每个城市的两相邻城市与其相邻程度分别为 1 和 2). 也就是说, 在最优路径中只有极少数相邻城市的相邻程度值较大. 通过表 2 可知当 $n \geq 500$ 时, 即使 r 取值较大时, 路径中任意两相邻城市均在相邻程度 r 范围内的概率 P 也几乎为零. 即采用 NN 算法求解较大规模的 TSP 问题, 几乎不可能得到最优解或近优解. 但注意到 Δp_k 随 k 的变动具有特殊规律性, 如图 1 是当 f=1, n=1000, r=20 时 Δp_k 随 k 变化的趋势图.

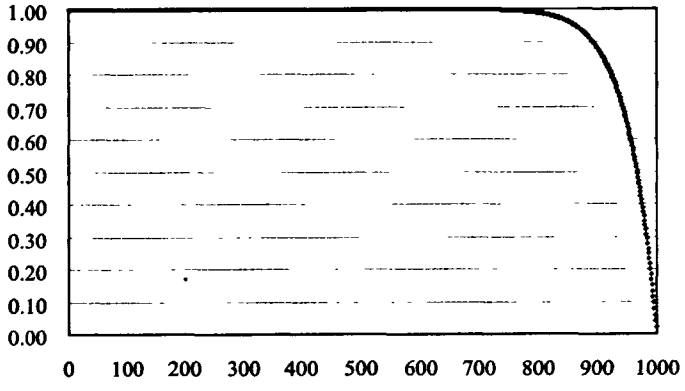

图 1 当 f=1, n=1000, r=20 条件下 Δp_k 随 k 变化的趋势图

由图 1 可以看出当 $1 \leq k \leq 800$ 时, Δp_k 的值几乎等于 1; 就是在 $800 < k \leq 900$ 区间里, Δp_k 的取值仍然大于 0.9; 只有在区间 $900 < k \leq 1000$ 中 Δp_k 才锐减至 Δp_{1000}=0.02002. 即采用 NN 算法求解城市数量为 1000 的三角不等式 TSP 问题, 在构造的前 800 步中, 几乎 100% 能够使任意两个相邻城市的相邻程度在 20 范围内, 只有在最后的 100 步构造中使得这一概率锐减至 0.02002. NN 算法求解其他 TSP 问题也存在这一规律, 如果能够采取措施克服 NN 算法在最后阶段的低质量构造, 则 NN 算法的性能可得到很大的改进.

然而 IN 算法在构造的后阶段连接相互邻近的城市的概率更大. 分析如下: 设 IN 算法在构造的第 k 步 ($4 \leq k \leq n$, 前三步为添加三个城市构成初始圈) 使待插入城市 i 插入的位置恰好是与其相邻程度在 r 范围内的两城市 j、h 之间, 且城市 i 也分别在城市 j、h 的相邻程度 r 范围内的概率为 q_k. g_s 表示在前 $k-1$ 步已旅行的城市中有 s 个城市与待插入城市 i 相邻程度在 r 范围内的概率, 其中 $0 \leq s \leq \min(r, k-1)$, g_s 的计算方法如公式 (6).

$$g_s = \frac{C_{n-1-r}^{k-1-s} \cdot C_r^s}{C_{n-1}^{k-1}} \tag{6}$$

显然 $\sum_{s=0}^{\min(r,k-1)} g_s = 1$, 设在 s 确定的条件下, 城市 i 插入的位置恰好是与其相邻程度在 r 范围内的两城市 j、h 之间的概率为 t_s, t_s 等于在 $k-1$ 个城市构成的圈排列中某 s 个城市中至少有两个城市相邻 (设优先选择这样的相邻城市为最佳插入位置) 的概率, 因此 t_s 的计算方法如公式 (7).

$$t_s = 1 - \frac{P_{k-1-s}^s \cdot (k-1-s-1)\,!/2}{(k-2)\,!/2} = 1 - \frac{P_{k-1-s}^s \cdot (k-2-s)\,!}{(k-2)\,!} \tag{7}$$

据概率知识可得 q_k 的计算公式 (8).

$$q_k = \left[\sum_{s=0}^{\min(r,k-1)} (t_s \cdot g_s) \right] \cdot f^2 \tag{8}$$

其中 f 含义与公式 (2) 中相同. 注意, 在公式 (3)、(6)、(7) 中当 $0\leq m<n$, $l\leq 0$ 时 $C_m^0=1, P_m^n=C_m^n=0, l!=1$.

假设 Q 表示 IN 构建的完整路径中任意两相邻城市均在彼此相邻程度 r 范围内的概率. 注意到 IN 算法每步插入一新城市等价于在旧圈中添加两条新边 (边是两城市的连线, 表示该两城市在路径中相邻) 并删除一条旧边得到新圈, 因此 IN 算法前阶段的低质量 (低质量: 新添加的城市未与其相邻程度在 r 范围内的城市相邻, 反之为高质量含义) 构造很可能被后阶段高质量的构造取代, 那么 Q 可按公式 (9) 计算.

$$Q=\prod_{k=1}^{n}\Delta q_k \tag{9}$$

$$\Delta q_k=\begin{cases}1, & \text{该步构造被后面步骤取代}\\ q_k, & \text{该步构造被保留至最后}\end{cases} \tag{10}$$

为了直观比较 NN 和 IN 算法求解质量的特点, 在参数 f=1, n=100, r=10, 且 $\Delta q_k=q_k$ 的条件下, 分析 Δp_k 和 Δq_k 随 k 变化的趋势, 如图 2 所示.

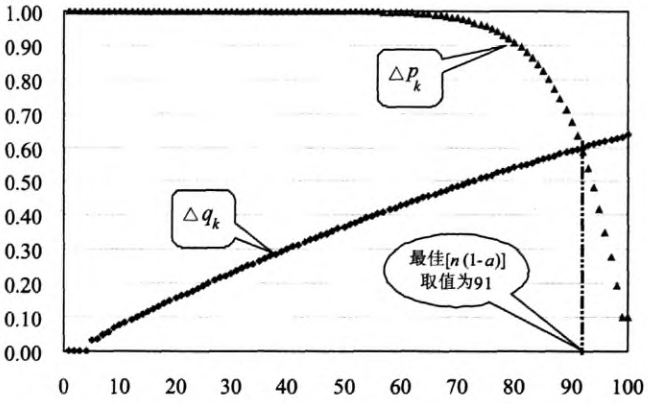

图 2　当 f=1, n=100, r=10, $\Delta q_k=q_k$ 条件下 Δp_k、Δq_k 随 k 变化的趋势图

由图 2 易见, NN 算法与 IN 算法的构造质量恰好互补, 在前一阶段 NN 算法构造质量非常高, 而后一阶段构造质量锐减, IN 算法的构造质量则逐渐增加. 因此, 如果在前 $[n(1-\alpha)]$ (表示取 $n(1-\alpha)$ 的整数部分, $0\leq\alpha\leq 1$), 步中采用 NN 算法, 后 $n-[n(1-\alpha)]$ 步使用 IN 算法, 形成的 NN&IN 混合算法的性能将大大提高. 那么在混合算法 NN&IN 构建的最终路径中, 任意两相邻城市均在彼此相邻程度 r 范围内的概率 W 及其下界的计算方法如公式 (11) 所示.

$$W=\left(\prod_{k=1}^{[n(1-\alpha)]}\Delta p_k\right)\cdot\left(\prod_{k=[n(1-\alpha)]+1}^{n}\Delta q_k\right)\geq\left(\prod_{k=1}^{[n(1-\alpha)]}\Delta p_k\right)\cdot\left(\prod_{k=[n(1-\alpha)]+1}^{n}q_k\right) \tag{11}$$

在图 2 所示的 Δp_k、Δq_k 散点曲线的相交处, 为当 f=1, n=100、r=10 时 $[n(1-\alpha)]$ 的最佳取值, 此处使 W 下界取值最大. 计算得 $[n(1-\alpha)]$ 等于 91, 参数 α 的的取值范围为 $0.08<\alpha\leq 0.09$, W 下界最大值为 4.345E-04, 是 NN 算法在表 2 中对应数据 3.989E-07 的 1089.232 倍, 可见算法 NN&IN 求解性能远远优于 NN 算法, 具有科学合理性. 以上数据分析中均假设为三角不等式 TSP 问题, 即 f=1, 对于非三角不等式 TSP 问题也有相同的规律性, 只是相应的概率数据更小.

3.2 NN&IN 算法的步骤

NN&IN 算法可以求解任何类型的 TSP 问题.

首先, 定义参数如下:

$\pi[n]$: 存放路径一维数组;

α: 参数, $0\leq\alpha\leq 1$;

V: 所有城市集合;

Unvisitedcity: 未旅行城市集合; 以下是通用的求解步骤:

Step 1　设定参数 α, 选择起点城市 i(按照城市标号由小到大选择起点城市), k=0,$\pi[k]\leftarrow i$, Unvisitedcity $=V\backslash\{i\}$;

Step 2 如果 $k+2 \leq [n(1-\alpha)]$ (采用最近邻域算法构造法则), $j \leftarrow \pi[k]$, $d_{jh}=\min\{d_{js}—s \in \text{Unvisitedcity}\}$, $k \leftarrow k+1$, $\pi[k] \leftarrow h$, Unvisitedcity = Unvisitedcity $\backslash\{h\}$;

否则 (采用插入算法构造法则), 选择 $t \in$ Unvisitedcity(本算法按集合 Unvisitedcity 中元素标号先小后大原则选择, 类似随机插入选择法则), $a=\pi[u]$, $b=\pi[u+1]$, $T(u, u+1)=d_{at}+d_{tb}-d_{ab}$, $T(v, v+1)=\min\{T(u, u+1)|0 \leq u \leq k$, 其中 $T(k, k+1)=T(k,0)\}$, 令 $\pi[k+1] \leftarrow \pi[k]$, $\pi[k] \leftarrow \pi[k-1]$, $\cdots$, $\pi[v+2] \leftarrow \pi[v+1]$, $\pi[v+1] \leftarrow t$, $k \leftarrow k+1$, Unvisitedcity = Unvisitedcity $\backslash\{t\}$;

Step 3 如果 Unvisitedcity 为非空集, 重复 Step 2; 否则, 结束构造, 得到最终解.

显然, 当参数 α=0 时, NN&IN 混合算法为 NN 算法, α=1 时, NN&IN 混合算法为 IN 算法.

3.3 NN&IN 算法复杂度分析

一个算法的计算复杂度由其计算量 (将一次判断、赋值、加、减、乘、除运算, 视为一次计算量) 决定, 求解 TSP 问题的计算量可以表示为城市数量 n 的函数, 记为 $f(n)$. NN&IN 的计算量由参数 α, NN 算法和 IN 算法的计算量确定.

设 NN 算法完成第 k 步构造的计算量记为 $\Phi(k)$. 第 k 步首先从其他 $n-1$ 个城市中判断出 $n-k$ 个未旅行城市, 然后在 $n-k$ 个未旅行城市中判断出离当前城市最近的城市, 将最近城市标号赋值给路径数组 π, 第 k 步完成. 因此 NN 算法的单步计算量 $\Phi(k)$ 和总计算量 $f(n)_{NN}$ 可以分别按公式 (12)、公式 (13) 计算.

$$\Phi(k)=(n-1)+(n-k)+1=2n-k \tag{12}$$

$$f(n)_{NN}=\sum_{k=1}^{n}\Phi(k)=\sum_{k=1}^{n}(2n-k)=\frac{3n^2-n}{2} \tag{13}$$

设 IN 算法完成第 k 步构造的计算量记为 $\Omega(k)$. 第 k 步首先从其他 $n-1$ 个城市中判断出 $n-k$ 个未旅行城市, 然后计算当前城市分别插入当前圈 $k-1$ 个位置使路径增加的长度 (i 插入 j、h 之间, 增加长度为: $d_{ji}+d_{ih}-d_{jh}$), 在 $k-1$ 个路径增加值中最小值对应的位置即为待插入城市的最佳插入位置, 最后更新存放圈路径数组 π, (设插入新城市后导致 $(k-1)/2$ 个路径数组元素重新赋值) 完成第 k 步. 因此 IN 算法的单步计算量 $\Omega(k)$ 和总计算量 $f(n)_{IN}$ 可以分别按公式 (14)、公式 (15) 计算.

$$\Omega(k)=(n-1)+2(k-1)+(k-1)+\frac{(k-1)}{2}=n+\frac{7}{2}k-\frac{9}{2} \tag{14}$$

$$f(n)_{IN}=\sum_{k=1}^{n}\Omega(k)=\sum_{k=1}^{n}\left(n+\frac{7}{2}k-\frac{9}{2}\right)=\frac{11\left(n^2-n\right)}{4} \tag{15}$$

令 $m=[n(1-\alpha)]$, 那么 NN&IN 算法的计算量可以由公式 (16) 计算.

$$f(m,n)_{NN\&IN}=\sum_{k=1}^{m}\Phi(k)+\sum_{k=m+1}^{n}\Omega(k)=\sum_{k=1}^{m}(2n-k)+\sum_{k=m+1}^{n}\left(n+\frac{7}{2}k-\frac{9}{2}\right) \tag{16}$$

将公式 (16) 化简得公式 (17).

$$f(m,n)_{NN\&IN}=mn+\frac{11\left(n^2-n\right)-9\left(m^2-m\right)}{4} \tag{17}$$

由 $0 \leq \alpha \leq 1$ 可得 $0 \leq m \leq n$, 因此 NN、NN&IN、IN 计算量的大小关系式如不等式 (18).

$$f(n)_{NN} \leq f(m,n)_{NN\&IN} \leq f(n)_{IN} \tag{18}$$

不等式 (18) 左右两边等号, 分别在 $m=n$ 和 m=0 处取得. 因复杂度仅由计算量多项式的最高次确定, 所以 NN、NN&IN、IN 三种算法的复杂度均为 $O(n^2)$.

4 实例及参数取值分析

4.1 实例

求解了 TSPLIB 标准库 [19] 中的 40 个对称 TSP 算例 (均为欧几里德 TSP 问题) 和 14 个非对称 TSP 算例 (均为三角不等式 TSP 问题), 以测试、比较 NN、NN&IN 及 IN 算法的性能. 求解环境为 Studio Visual C++6.0, C 语言编程; CPU 为 AMD turion64×2 Mobile TL-52, 内存为 1.5G, 主频为 1.61GHz. 三种算法求解算例的结果如表 3、表 4 所示.

表 3、表 4 关键字说明: 算例名称中包含的数字为算例的城市数量, 如 "pr2392" 为有 2392 个城市的 TSP 问题 (kro124p 含有 100 个城市, 属于个别例外情况); 最优路径长度: 算例的已知最优解路径长度; 求解

数量: 对城市数量 n <600 的算例, 求解数量和城市数量 n 相同, 600≤ n < 1000, 求解数量为 200, n ≥1000, 求解数量为 100; 三种算法分别对应的最优求解质量、平均求解质量: 求解质量用百分数表示, 计算方法为 100%×(解路径长度 − 最优路径长度)/ 最优路径长度; 单个解平均运行时间: 即总运行时间/求解数量, 由于某些算例城市数量较小, 导致求解速度太快超出计算机计时精度, 从而有些平均运行时间显示为 0; 参数 α 取值: 为测试 NN&IN 算法的鲁棒性, 在求解每个算例时参数 α 均设定为 0.2 (对于很多算例并不是理想取值, 详见论文 4.2 节的分析).

表 3　三种算法求解对称 TSP 算例数据比较表

序号	算例名称	最优路径长度	求解数量	最优解质量 (%)			平均解质量 (%)			单个解平均运行时间 (s)		
				NN	NN&IN	IN	NN	NN&IN	IN	NN	NN&IN	IN
1	a280	2579	280	19.98	8.99	8.67	28.54	20.23	18.91	0.110	0.125	0.239
2	berlin52	7542	52	8.49	1.08	2.03	24.28	13.38	10.43	0.000	0.000	0.000
3	bier127	118282	127	13.26	4.58	5.11	23.58	13.46	12.88	0.016	0.016	0.024
4	ch130*	6110	130	17.82	5.27	6.85	26.36	11.60	11.58	0.015	0.015	0.031
5	ch150	6528	150	8.43	7.24	8.13	17.91	14.10	13.84	0.020	0.027	0.040
6	d198*	15780	198	12.86	5.03	3.72	19.50	9.56	8.37	0.045	0.051	0.086
7	d493*	35002	493	18.56	8.76	10.61	25.23	15.67	15.99	0.633	0.684	1.613
8	d657*	48912	200	24.81	11.68	14.66	28.39	21.38	20.96	1.105	1.360	4.380
9	d1291	50801	100	16.22	10.78	17.40	20.21	21.42	23.62	8.250	9.315	35.810
10	eil51*	426	51	18.73	3.86	5.00	32.18	10.72	10.51	0.000	0.000	0.000
11	eil101*	629	101	17.07	6.91	7.59	26.42	12.47	12.64	0.010	0.010	0.020
12	gil262*	2378	262	21.27	9.23	9.22	29.75	15.23	15.28	0.095	0.115	0.240
13	gr431	1714.14	431	41.77	23.65	23.72	50.64	28.73	28.93	0.401	0.457	1.032
14	kroB100*	22141	100	16.90	5.03	4.83	26.04	10.73	8.75	0.010	0.010	0.010
15	kroE100*	22068	100	12.86	1.87	3.79	24.60	12.30	9.42	0.010	0.010	0.010
16	kroA150*	26524	150	18.69	5.54	5.80	26.83	14.94	11.60	0.027	0.027	0.040
17	kroA200*	29368	200	17.64	5.56	6.70	27.58	13.33	12.42	0.045	0.050	0.090
18	lin105	14379	105	17.81	4.45	3.04	30.19	18.22	9.66	0.010	0.010	0.010
19	lin318*	42029	318	17.10	11.05	10.03	25.15	16.75	15.51	0.176	0.182	0.387
20	pcb442	50778	442	16.10	9.94	10.33	22.55	18.72	18.38	0.423	0.507	1.208
21	pr76*	108159	76	21.04	3.72	3.37	36.08	9.70	8.79	0.000	0.000	0.013
22	pr144	58537	144	4.14	1.66	2.06	12.59	11.00	8.20	0.021	0.028	0.035
23	pr299*	48191	299	20.95	8.63	10.13	30.59	15.68	15.37	0.144	0.157	0.301
24	pr439*	107217	439	18.66	5.91	8.02	26.56	16.29	18.21	0.440	0.535	1.146
25	pr2392	378032	100	21.36	18.12	19.86	24.16	23.11	23.50	74.155	80.620	189.180
26	rat99	1211	99	13.09	8.15	7.28	21.88	17.57	14.06	0.010	0.020	0.010
27	rat195	2323	195	13.15	10.23	11.41	19.66	17.47	18.25	0.046	0.051	0.082
28	rat783*	8806	200	26.17	14.43	16.77	29.33	23.49	22.88	1.955	2.600	7.025
29	rd400*	15281	400	19.78	9.53	10.19	25.77	14.63	15.38	0.360	0.405	0.983
30	st70*	675	70	12.84	5.42	5.21	22.34	11.00	9.37	0.000	0.000	0.000
31	ts225	126643	225	10.93	10.35	12.87	17.33	21.71	21.82	0.067	0.076	0.133
32	tsp225*	3916	225	18.31	4.82	8.21	25.63	12.47	13.46	0.067	0.076	0.129
33	u159	42080	159	15.46	5.85	6.76	28.67	13.71	15.99	0.025	0.031	0.050
34	u574*	36905	574	20.62	9.74	14.08	27.78	16.63	18.59	0.704	0.828	2.409
35	u724*	41910	200	21.94	12.38	14.72	28.00	17.17	18.67	1.465	1.820	5.560
36	u1060*	224094	100	25.37	15.95	17.13	31.29	20.16	19.61	4.570	5.570	15.570
37	u1432*	152970	100	23.00	14.12	15.96	26.80	18.76	18.90	13.740	18.710	48.570
38	u2152	64253	100	21.65	18.24	20.92	23.28	22.74	24.08	38.030	47.070	133.410
39	vm1084*	239297	100	21.52	11.88	13.74	25.85	16.99	17.54	4.690	5.995	21.540
40	vm1748*	336556	100	20.65	11.08	13.94	26.16	17.54	17.82	16.960	22.560	100.920

表 4 三种算法求解非对称 TSP 算例数据比较表

序号	算例名称	最优路径长度	求解数量	最优解质量 (%)			平均解质量 (%)			单个解平均运行时间 (s)		
				NN	NN&IN	IN	NN	NN&IN	IN	NN	NN&IN	IN
1	br17	39	17	43.59	0.00	0.00	112.51	50.97	1.21	0.000	0.000	0.000
2	ft53*	6905	53	24.32	6.60	6.36	35.04	20.31	17.01	0.000	0.000	0.000
3	ftv35	1473	35	13.17	5.30	6.04	26.30	14.15	13.72	0.000	0.000	0.000
4	ftv47*	1776	47	22.35	0.79	3.15	39.16	13.13	10.76	0.000	0.000	0.000
5	ftv55*	1608	55	21.14	6.90	7.65	35.70	14.46	17.12	0.000	0.000	0.000
6	ftv64	1839	64	19.74	8.32	10.17	36.63	19.95	17.84	0.000	0.000	0.000
7	ftv70	1950	70	17.28	9.28	8.97	26.73	18.87	18.24	0.000	0.000	0.000
8	ftv170*	2755	170	30.02	19.31	16.99	44.10	27.26	29.80	0.035	0.035	0.080
9	kro124p*	36230	100	19.56	7.35	7.05	27.59	14.17	13.74	0.010	0.010	0.020
10	p43*	5620	43	1.14	0.14	0.05	2.34	0.51	0.42	0.000	0.000	0.000
11	rbg323*	1326	323	28.36	10.71	11.31	31.30	19.48	19.27	0.170	0.217	0.573
12	rbg358*	1163	358	50.21	13.59	12.98	54.39	22.66	22.91	0.232	0.302	0.818
13	rbg403*	2465	403	41.87	1.83	1.78	43.78	4.49	4.51	0.325	0.419	1.117
14	rbg443*	2720	443	41.84	1.76	1.88	43.49	4.74	4.85	0.429	0.555	1.519

在表 3、表 4 中包括了能够体现算法性能的求解速度和求解质量两方面的数据.

在求解速度方面: 所有算例中 NN&IN 算法求解时间仅略高于 NN 算法, 但 IN 算法几乎求解每个算例 (运行时间很小或忽略不计的算例除外) 的耗时均是 NN 算法的数倍. 通常有以下方法提高算法的求解速度[17]: 1) 距离矩阵中数据设定为整数型; 2) 对于大规模欧几里德 TSP($n \geq$1000) 算例, 使用最近邻域表, 且不预先计算距离矩阵, 只在用到某两城市间距离时才计算; 3) 使用更优化的数据结构. 实践证明这些方式能够大大提高算法运行速度. 本程序距离矩阵的数据类型为双精度型、没有使用最近邻域表和优化的数据结构. 因为本论文以在同等条件下比较 NN、NN&IN、IN 三种算法之间的相对求解速度为目的, 程序中未采用任何方式以提高求解速度并不影响该指标的比较.

在求解质量方面: NN 算法的求解质量明显低于 NN&IN 和 IN 算法, 在绝大部分算例中 (见表 3、表 4 中算例名称后带 "*" 的算例) 甚至后两种算法的平均解质量都高于 NN 算法求解的最优解质量. NN&IN 的求解质量与 IN 算法相比, 对求解小规模 TSP 问题 ($n < 1000$) IN 算法平均解质量略胜于 NN&IN, 后者的最优解质量高于前者; 而对于大规模 TSP 问题 ($n \geq$1000) 无论是最优解质量还是平均解质量 NN&IN 算法均明显优于 IN 算法. 一般来说, 由于 TSP 问题解空间非常大 (n 个城市数量的 TSP 问题有 $(n-1)!/2$ 个可行解), 对于城市数量稍大的 TSP 问题 ($n > 50$), 构建型启发式算法几乎不可能得到最优解, 目前算例最优解要么是采用改进型启发式算法以更大的求解时间为代价得到的已知最优解, 要么是采用精确型算法耗费巨大时间得到的确定最优解. 改进型启发式算法是在构建型启发式算法得到的解的基础上, 采用特定优化规则进一步改进解质量的算法, 因此改进型算法求解质量一般优于求解迅速的构建型启发式算法. 然而通过比较 Junger[3] 求解的数据发现, 对于某些算例 NN&IN 的求解质量甚至可以和改进型启发式算法 2- 交换法 (2-opt) 相媲美, 如表 5 所示.

表 5 NN&IN 算法与 NN+2-opt 法求解质量比较表

	d198	d493	gil262	lin105	pcb442	pr144	pr299	u159
NN&IN(%)	5.03	8.76	9.23	4.45	9.94	1.66	8.63	5.85
NN+2−opt[3] (%)	3.85	9.37	10.26	8.42	8.74	3.79	10.46	14.00

注: NN+2−opt 算法是在 NN 算法构造的解基础上, 采取替换解中 2 条边的优化方式的改进型启发式算法.

另外, 值得注意的是在很多算例中 α 值设定为更小的数时, NN&IN 的求解质量却更高, 如算例 pcb442, 取 α=0.03, 运算 442 个解, 得最优解为 53373.86(5.11%), 平均解为 56602.33(11.47%), 单解平均运行时间 0.446s, 类似例子不再赘述.

4.2 参数 α 取值分析

由公式 (17) 可知: NN&IN 算法的计算量随参数 α 单调递增, 所以参数 α 的取值原则是在保证 NN&IN

算法求解质量的条件下越小越好. 通过大量算例求解试验表明参数 α 的取值与求解质量具有一定的对应规律. 如算例 eil101, 参数 α 分别取 0.01、0.02、$\cdots$、0.99、1.00 共 100 个不同值, 在每个取值条件下, 采用 NN&IN 算法求解 eil101 生成 100 个解 (共计 10000 个解). 不同的参数 α 值对应的求解平均质量与最优解质量变动趋势如图 3 所示, 其中纵坐标为路径总长度, 横坐标为参数 α 值 (算例 eil101 最优路径长度为 629).

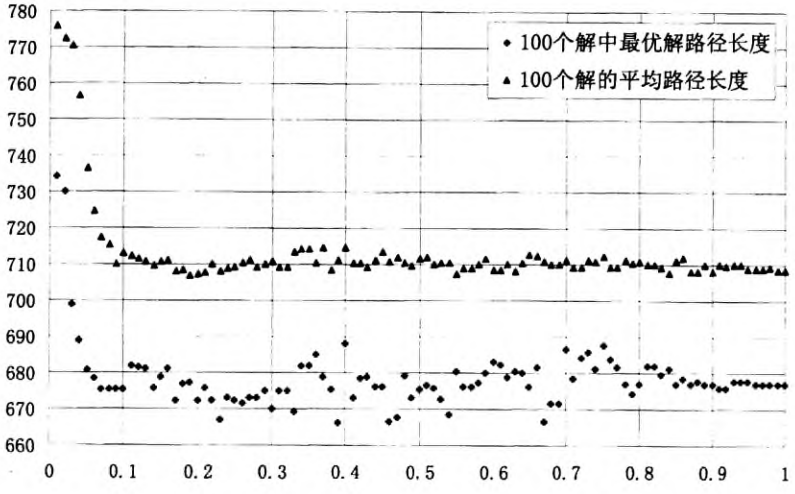

图 3　算例 eil101α 取不同值与 NN&IN 算法求解质量关系图

通过图 3 可以看出, 当 α 取值从 0.01 增加到 0.1 的过程中, NN&IN 求解的最优质量与平均质量均迅速提高, 而当 α 取值大于 0.1 时求解质量呈波动趋势. 因此, NN&IN 求解质量随 α 值的变动可以分为迅速提高、水平波动两个阶段, 这两阶段之间的 α 临界值 (求解质量首次变差的临界点) 正是最理想的 α 值. 图 3 中算例 eil101 平均质量的临界 α 值为 0.09, 最优质量的临界 α 值为 0.10, 这与图 2 中分析的当 f=1, n=100, r=10, $\Delta q_k = q_k$ 时的最佳 α 取值范围 $0.08 < \alpha \leq 0.09$ 基本一致. 因此, 通过分析 Δp_k、Δq_k 散点曲线交点确定 α 理想取值范围的方式具有较强的实用参考价值. 因计算 Δq_k 涉及阶乘的运算, 论文仅计算了当 n 取值较小时 α 的理想取值范围, 对于较大的 n 值根据求解试验数据提供了 α 的经验取值范围, 具体数据如表 6 所示.

表 6　参数 α 取值范围参考表

	计算数据			经验数据		
城市数量 n	n=50, r=5	n=100, r=10	n=150, r=10	$150 < n \leq 500$	$500 < n \leq 1000$	$n > 1000$
α 取值范围	$0.060 < \alpha \leq 0.080$	$0.080 < \alpha \leq 0.090$	$0.053 < \alpha \leq 0.060$	$0.050 < \alpha \leq 0.200$	$0.050 < \alpha \leq 0.150$	$0.050 < \alpha \leq 0.100$

5　结束语

在求解 TSP 问题的算法具有理论方面的重大突破前, 算法设计大多是在解空间中的求解深度和求解广度之间权衡, 即在求解质量和求解时间之间的折中 [20−21]. 因此如何以增加少量的运行时间为代价换取求解质量明显的提高, 是改进这类算法性能的核心问题. 本文提出的 NN&IN 算法集合了 NN 算法的求解速度快、IN 算法求解质量高的优点, 以略微高于 NN 算法的耗时为代价, 使得总体求解质量超过耗时为 NN 算法数倍的 IN 算法. 从理论分析 NN&IN 算法的合理性到算例求解结果的高效性表明: NN&IN 算法的综合性能优于 NN、IN 算法.

参考文献

[1] Ugur A, Aydin D. An interactive simulation and analysis software for sovling TSP using ant colony optimization algorithms[J]. Advances in Engineering software, 2009(5): 341–349.

[2] Dorigo M, Stutzle T. Ant Colony Optimization[M]. Massachusetts: MIT Press, 2004.
[3] Junger M, Reinelt G, Rinaldi G. Handbooks in OR&MS: Chapter 4 the Traveling Salesman Problem[M]. Elsevier Science, 1995.
[4] Lin S, Kernighan B W. An effective heuristic algorithm for the traveling salesman problem[J]. Operations Research, 1973, 21: 498–516.
[5] Helsgaun K. An effective implementation of the Lin-Kernighan traveling salesman heuristic[J]. European Journal of Operational Research, 2000, 126: 106–130.
[6] Pesch E, Glover F. TSP ejection chains[J]. Discrete Applied Mathematics, 1997, 76: 165–181.
[7] Kirkpatrick S, Gelatt C D, Vecchi M P. Optimization by simulated annealing[J]. Science, 1983, 220: 671–680.
[8] Glover F. Tabu search — Part I[J]. ORSA Journal on Computing, 1989, 1(3): 190–206.
[9] Glover F. Tabu search — Part II[J]. ORSA Journal on Computing, 1990, 2(1): 4–32.
[10] Xing L N. A hybrid approach combining an improved genetic algorithm and optimization strategies for the asymmetric traveling salesman problem[J]. Engineering Applications of Artificial Intelligence, 2008, 21: 1370–1380.
[11] Li M L, Yi Z, Zhu M. Solving TSP by using Lotka-Volterra neural networks[J]. Neurocomputing, 2009, 10: 3873–3880.
[12] Dorigo M, Di Caro G, Gambardella L M. Ant algorithms for distributed discrete optimization[J]. Artificial Life, 1999, 5: 137–172.
[13] Dorigo M, Blumb C. Ant colony optimization theory: A survey[J]. Theoretical Computer Science, 2005, 344: 243–278.
[14] Stutzle T, Hoos H. Max-min ant system[J]. Future Generation Computer System, 2000, 16: 889–914.
[15] Marinakis Y, Marinaki M. A hybrid multi-swarm particle swarm optimization algorithm for the probabilistic traveling salesman problem[J]. Computers & Operations Research, 2010(3): 432–442.
[16] 屈稳太, 丁伟. 一种改进的蚁群算法及其在 TSP 中的应用 [J]. 系统工程理论与实践, 2006, 26(5): 93–98.
Qu W T, Ding W. A improved ant colony algorithm and application in TSP[J]. Systems Engineering —— Theory & Practice, 2006, 26(5): 93–98.
[17] Gamboa D, Rego C, Glover F. Implementation analysis of efficient heuristic algorithms for the traveling salesman problem[J]. Computers & Operations Research, 2006, 33: 1154–1172.
[18] Lozano M, Garcia-Martinez C. Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report[J]. Computers & Operations Research, 2010, 3: 481–497.
[19] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/.
[20] Blum C, Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison[J]. ACM Computing Surveys, 2003, 35: 268–308.
[21] Ravindra K, James B, Punnen P. A survey of very large-scale neighborhood search techniques[J]. Discrete Applied Mathematics, 2002, 123: 75–102.