文章编号: 1001-0920 (2016) 04-0586-07 DOI: 10.13195/j.kzyjc.2015.0126

一种求解旅行商问题的新型帝国竞争算法

张鑫龙, 陈秀万, 肖 汉, 李 伟

(北京大学 地球与空间科学学院，北京 100871)

摘 要: 帝国竞争算法是一种已在连续优化问题上取得较好效果的新型社会政治算法. 为了使该算法更好地应用于离散型组合优化问题, 提出一种求解旅行商问题的新型帝国竞争算法. 在传统算法的基础上, 改变初始帝国的生成方式; 同化过程采取替换重建方式, 以提升求解质量; 革命过程中引入自适应变异算子, 以增强搜索能力; 殖民竞争过程中调整了殖民地分配方式; 算法加入帝国增强过程, 以加快寻化速度. 实验结果表明, 新型帝国竞争算法求解质量高、收敛速度快.

关键词: 旅行商问题；帝国竞争算法；遗传算法

中图分类号: TP18 **文献标志码:** A

A new imperialist competitive algorithm for solving TSP problem

ZHANG Xin-long, *CHEN Xiu-wan*, *XIAO Han*, *LI Wei*

(School of Earth and Space Sciences，Peking University，Beijing 100871，China．Correspondent：ZHANG Xin-long, E-mail：mtxinlong@126.com)

Abstract: Imperialist competitive algorithm is a new socio-political motivated strategy, which has better performance in continuous optimization problems. In order to apply this method to complex discrete combinatorial optimization problems appropriately, an improved imperialist competitive algorithm for solving traveling salesman problem is proposed. Based on the traditional algorithm, the proposed algorithm changes the way of forming initial empires. In the assimilation process, solutions are improved by a replacement-reconstruction policy. Then a method of adaptive adjustment of mutation probabilities is introduced in the revolution process, which enhances the search capability of the proposed algorithm. The method of colony allocation is adjusted in the imperialist competition process. An imperialist improvement mechanism is presented in order to enhance the optimization speed. The results of a set of TSP instances demonstrate the superiority of the proposed algorithm in both solution quality and convergence speed.

Keywords: traveling salesman problem；imperialist competitive algorithm；genetic algorithm

0 引 言

旅行商问题 (TSP) 是组合优化领域中著名的 NP-hard 问题. 设有一个旅行商人要拜访 N 个城市, 从初始城市出发依次拜访每个城市, 且每个城市仅能拜访一次, 最终回到出发点. TSP 问题是在所有可能的路线中选择路长最短的路径. TSP 问题具有广泛的应用背景, 如飞机航线设计、物流配送、交通运输等. 因此, 快速高效求解 TSP 问题一直是组合优化领域研究的热点. 当前, 随着人工智能的发展, 许多智能优化算法及其改进算法被引入到求解 TSP 问题[1], 如免疫算法、蚁群算法、退火算法、遗传算法等.

随着研究的深入, 许多新型优化算法不断涌现. 其中, 帝国竞争算法 (ICA) 是近些年被提出的一种新颖的基于群体的元启发式优化算法. 该算法在创建初期主要用于解决连续型优化问题, 现已逐步应用于复杂的离散组合优化问题. Mohammadi 等[2]将 ICA 应用于枢纽站选址问题, 取得了较为理想的效果; Forouharfard 等[3]将该算法应用到转运仓库的物流安排中, 有效地提高了调度效率; Devika 等[4]将 ICA 与其他算法应用在供应链网络设计中, 对比发现 ICA 求解效果较优; Rahimi 等[5]在多项目调度问题中比较了 ICA 与进化算法, 发现前者比后者更为有效. 随着研究的深入, Seyedmohsen 等[6]指出 ICA 可以引入到求解 TSP 问题中, Ardalan 等[7]尝试利用 ICA 求解广义

收稿日期: 2015-01-27；**修回日期:** 2015-05-15.

基金项目: 国家科技支撑计划项目(2011BAH05B08).

作者简介: 张鑫龙(1990−), 男, 博士生, 从事智能算法、数字城市的研究；陈秀万(1964−), 男, 教授, 博士生导师, 从事智慧城市、智能导航等研究.

旅行商问题 (GTSP), 求解质量较以往算法高, 但仍有很大的优化空间. 为了充分发挥帝国竞争算法在组合优化领域的寻优能力, 本文在上述研究的基础上, 对传统帝国竞争算法加以改进, 提出一种求解 TSP 的新型帝国竞争算法. 实验结果表明, 所提算法在求解 TSP 问题时求解质量高, 收敛速度快.

1　传统帝国竞争算法

帝国竞争算法是 2007 年 Atashpaz-Gargari 等[8] 提出的一种社会政治进化算法, 是对人类社会殖民竞争过程的一种模拟. 帝国竞争算法的主要过程如下:

第一步是形成帝国. 帝国竞争算法首先通过随机方法生成初始国家, 每个国家代表所求问题的一个解. 这些国家依据权力大小 (求解质量) 分为殖民国家和殖民地两个类别. 权力最大的前若干国家成为殖民国家. 然后按照殖民国家权力大小依次将剩余国家作为殖民地分配给殖民国家, 权力越大的殖民国家所分配的殖民地越多. 殖民国家与其所属的殖民地统称为帝国.

第二步是同化与革命. 在帝国形成后, 必然伴随着殖民地的经济、文化、语言等属性趋向于所属殖民国家, 这一过程叫作同化. 同化的目的在于提升所有国家求解质量的同时, 能够增加殖民国家对殖民地的影响[9]. 为了与历史相符, 殖民地趋向帝国的过程总是有一定偏移, 极端情况下甚至会出现反向偏移, 即殖民地革命. 如果殖民地在同化与革命过程中权力超过了所属殖民国家, 则此时殖民地将取代殖民国家建立新的帝国.

最后一步是殖民竞争. 帝国之间存在着竞争, 帝国间势力的此消彼长使得弱小帝国的殖民地会被强大帝国所剥夺, 直至弱小帝国消失. 同时, 在竞争过程中会不断出现新的帝国. 经过数代的同化、革命、竞争后, 理想情况下只会留存一个帝国, 且所有国家都为该帝国的成员. 帝国竞争算法正是通过这样一系列的进化操作, 最终找到全局范围内的最优解.

2　求解 TSP 问题的新型帝国竞争算法

本文针对 TSP 问题, 对帝国竞争算法进行一定的改进, 提出一种新型帝国竞争算法. 改进内容具体如下:

第一, 传统帝国竞争算法中的初始国家是一个向量. 而在新型帝国竞争算法中, 在对每个城市编码后, 初始国家即为访问城市顺序组成的编码序列.

第二, 针对组合优化问题来说, 同化过程表现为殖民地解的结构与殖民国家解的结构更相似[10]. 由于殖民国家代表较优解, 在新型帝国竞争算法中, 同化作用是通过帝国内殖民国家与殖民地以替换重建的方式生成新的同化殖民地来表示. 替换重建过程可以使殖民国家解中包含的信息合理地扩散到它的殖民地中.

第三, 殖民地革命过程中, 借鉴了遗传算法中的变异算子[11-12], 通过殖民地随机突变来增加算法全局搜索空间. 在这一过程中, 从运算代数和殖民地所属帝国权力两个方面实现革命概率自适应调节.

第四, 殖民竞争过程前, 增加了帝国增强的步骤. 通过对殖民国家的优化操作, 可以使算法能够更快收敛. 同时, 帝国增强过程中还会伴随新帝国的产生, 这也可以有效避免算法过早收敛而陷入局部最优.

第五, 殖民竞争过程中, 调整了殖民地分配方式. 在分配最弱帝国的最弱殖民地时, 同时考虑帝国权力与帝国内拥有的殖民地数量两个因子, 使得权力较大且拥有较多殖民地的帝国获得待分配殖民地的概率变小, 以防止算法过早收敛.

2.1　建立初始帝国

在算法的初始化阶段, 帝国竞争算法随机生成 N_{pop} 个初始国家, 然后计算每个国家的权力. 这里权力与路径总长度有关

$$c_t = \frac{1}{\text{dis}(t)}. \tag{1}$$

其中: c_t 表示第 t 个国家的权力, $\text{dis}(t)$ 表示第 t 个国家按顺序遍历所有城市的总距离. 依据国家权力大小从高到低排序, 选择前 N_{imp} 个作为殖民国家, 其余作为殖民地, 则殖民地 N_{col} 的数量为

$$N_{\text{col}} = N_{\text{pop}} - N_{\text{imp}}. \tag{2}$$

为了计算方便, 通过下式对所有殖民国家权力进行标准化处理

$$p_n = \frac{c_n}{\sum_{i=1}^{N_{\text{imp}}} c_i}, \tag{3}$$

其中 c_n 和 p_n 分别表示第 n 个殖民国家的权力和标准化权力. 通过以下分配方式, 第 n 个殖民国家分配得到的殖民地个数 NC_n 为

$$\text{NC}_n = \text{round}\{p_n \cdot N_{\text{col}}\}, \tag{4}$$

其中 round 函数表示为四舍五入取整.

2.2　殖民地同化

同化作用是帝国内部殖民地逐渐趋向殖民国家的过程. 本文采取替换重建方式生成新的同化殖民地.

Step 1: 在殖民地每个城市编码位置上随机生成一个 0～1 之间的概率.

Step 2: 在殖民地内所有概率大于等于 ρ 的编码位置处, 用该位置的殖民国家城市编号作为同化殖民

地的编号.

Step 3: 对于殖民地内概率比 ρ 小的编码位置, 若该位置处的城市编号在同化殖民地其余位置处没有出现, 则以该位置殖民地的城市编号作为同化殖民地的编号. Step 2 和 Step 3 统称为替换过程.

Step 4: 重建过程. 对于同化殖民地中未出现的城市编号, 依次计算其插入在同化殖民地城市序列中所有可能出现位置的增加距离, 并把该城市编号插入在最小增加距离的位置处. 将编号为 i 的城市插入到城市编号为 a 与 b 两相邻位置间的增加距离 N 为

$$N = \mathrm{dis}(a,i) + \mathrm{dis}(i,b) - \mathrm{dis}(a,b), \tag{5}$$

其中 $\mathrm{dis}(a,i)$ 表示 a 城市到 i 城市的距离. 同化过程完成后, 用新生成的同化殖民地替换原有殖民地.

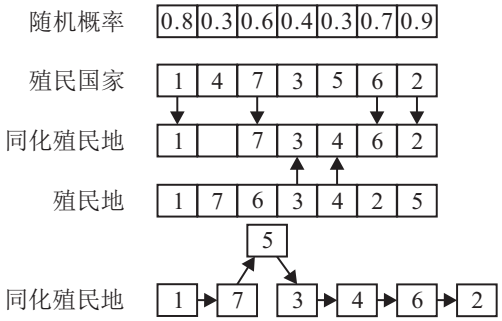

图 1 同化殖民地生成过程

具体操作方式如图 1 所示, 第 1 行是殖民地的随机概率序列, 第 2 行与第 4 行是殖民国家与殖民地的城市编号序列. 本文 ρ 取 0.5, 则可将殖民国家中的城市编号 1762 直接作为同化殖民地中的编号. 殖民地中 $\rho < 0.5$ 处的城市编号为 734, 殖民地中编号 7 在同化殖民地中已出现, 因而舍弃; 在同化殖民地中编号 3 和 4 还未出现, 因此可作为同化殖民地中对应位置的编号. 此时, 同化殖民地中未曾出现城市编号 5, 则计算编号 5 插入同化殖民地城市序列中所有位置的增加距离. 假设编号 5 插入城市编号 7 与 3 之间的增加距离最短, 则如图中第 5 行所示, 可确定同化殖民地的路径.

2.3 殖民地革命

本文中, 革命表现为随机选取殖民地两个位置的城市编号进行交换. 在避免无效随机搜索的同时, 能够充分发挥革命机制保持种群多样性功能, 从运算代数和所属帝国权力两方面对革命概率 p_m 建立调节公式

$$p_m = p_0\left(1 - \frac{p_n}{p_{\max}} \cdot \mathrm{e}^{-\frac{g}{G}}\right). \tag{6}$$

其中: p_n 为殖民地所属殖民国家的标准化权力, $p_{\max}$ 为所有殖民国家中最大的标准化权力, g 为当前运算迭代次数, G 为总共迭代次数, $p_0 = 0.3$. 对于权力越小的殖民国家, 该帝国内所有殖民地革命的概率越大, 这可以促进解的不断优化. 算法执行到后期, 帝国内部殖民地所代表的解结构类似, 此时很有可能陷入局部最优. 为了避免这种情况, 增加革命概率可以有效扩大整体的搜索范围. 在同化作用和革命完成之后, 重新计算帝国内殖民地和殖民国家的权力. 若殖民地权力比殖民国家大, 则替换该殖民国家形成新的帝国.

2.4 帝国增强

在帝国竞争算法中, 殖民地通过同化和革命机制来提高求解质量. 但是大多殖民国家求解质量并没有改变, 因此增加帝国增强步骤来提高殖民国家的求解质量.

帝国增强的思想为: 假设共有 m 个城市, 在殖民国家中随机选择两个位置, 将这两个位置之间的城市编码序列反转, 并从新生成的序列中随机移除 k $\left(k = \left[\frac{1}{10}m\right]\right)$ 个城市. 将这 k 个城市依次插入剩余 $m - k$ 个城市中所有可能的位置, 并计算增加距离 (见 2.2 节). 当这 k 个城市都插入到最短增加距离位置后, 即得到新的国家. 比较新生成国家与原殖民国家的权力, 将两者中权力较大的作为该帝国的殖民国家, 权力较小的放入备选帝国列表. 在对每一个帝国进行帝国增强运算后, 备选帝国列表个数与帝国个数相同. 在 0～1 之间随机生成一个概率, 若该概率大于 β ($\beta = 0.95$), 则在备选帝国列表中选择权力最大的国家作为新增帝国的殖民国家进行下一步运算.

通过帝国增强, 殖民国家会搜索到更优解, 同时殖民国家的信息也会通过同化作用扩散到它的殖民地上, 从而提高整个解集质量.

2.5 殖民竞争

帝国竞争是殖民地再分配的过程, 也是该算法收敛的关键步骤. 帝国的权力由组成该帝国的殖民国家的权力与殖民地平均权力加权和构成, 具体表示为

$$\mathrm{TC}_n = (1-\delta)p_n + \delta\frac{\sum_{j=1}^{\mathrm{NC}_n} p_{nj}}{\mathrm{NC}_n}. \tag{7}$$

其中: TC_n 为第 n 个帝国的权力; p_{nj} 为第 n 个帝国第 j 个殖民地的权力; NC_n 为当前该帝国殖民地个数; δ 为权系数, 本文取 0.1. 为了模拟帝国竞争过程, 对帝国权力进行标准化, 则第 n 个帝国的标准化权力 TP_n 由下式表示:

$$\mathrm{NTC}_n = \mathrm{TC}_n - \min_{i \leqslant N_{\mathrm{imp}}}\{\mathrm{TC}_i\}, \tag{8}$$

$$\mathrm{TP}_n = \frac{\mathrm{NTC}_n}{\sum_{i=1}^{N_{\mathrm{imp}}} \mathrm{NTC}_i}. \tag{9}$$

其中

$$\sum_{i=1}^{N_{\text{imp}}} \text{TP}_i = 1.$$

为了根据帝国权力来分配较弱帝国的殖民地, 建立帝国权力向量

$$P = [\text{TP}_1, \text{TP}_2, \cdots, \text{TP}_{N_{\text{imp}}}]. \tag{10}$$

再建立一个维数与帝国数量相同的, 由均匀分布随机数 $r_i\ (i \leqslant N_{\text{imp}})$ 组成的向量

$$R = [r_1, r_2, \cdots, r_{N_{\text{imp}}}]. \tag{11}$$

其中: $r_i \sim U\left(0, \dfrac{\text{NI}_i}{N_{\text{pop}}}\right)$, NI_i 表示第 i 个帝国内拥有国家的数量. 定义概率向量为

$$D = P - R = [d_1, d_2, \cdots, d_{N_{\text{imp}}}] = [\text{TP}_1 - r_1, \text{TP}_2 - r_2, \cdots, \text{TP}_{N_{\text{imp}}} - r_{N_{\text{imp}}}]. \tag{12}$$

假设 $d_i (i \leqslant N_{\text{imp}})$ 为 D 中所有元素的最大值, 则把最弱帝国的最弱殖民地分配给第 i 个帝国. 当较强帝国殖民地个数也较多时, 尽管考虑了帝国权力的作用, 仍有部分随机性使得非最强帝国获得一定数量的殖民地. 设置概率向量 D 可以有效避免运算前期出现权力过大的帝国导致算法较早收敛的情况. 通过殖民竞争的作用, 权力越弱的帝国将丢失越多的殖民地, 直至该帝国内没有任何国家, 该帝国消失.

2.6　算法计算流程

新型帝国竞争算法的流程如图 2 所示. 需要指出的是, 在理想情况下, 经过若干次的迭代后, 仅剩下一个帝国, 此时算法达到终止条件. 在实际应用中, 帝国竞争算法不断迭代运算, 达到了预先规定的次数也可作为终止条件.

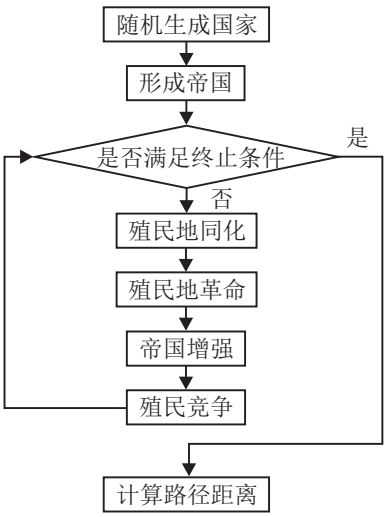

图 2　新型帝国竞争算法流程

3　算法实验与分析

为了验证新型帝国竞争算法(以下简称为 ICA)的有效性, 本文选取 30 组国际通用 TSPLIB 测试库实例进行测试, 并将测试所得结果与其他算法结果进行对比分析. 实验环境: 操作系统为 Windows 7, CPU 为 i5-3450, 内存 4G, 编程软件为 Microsoft Visual Studio 2010.

目前, 遗传算法 (GA) 在解决 TSP 问题上具有一定优势. 在本实验中, 将 ICA 与 GA 进行对比, 为保证两种算法对比的可靠性, ICA 初始国家个数与 GA 初始种群数量均设置为 100, 迭代次数均为 500. 帝国个数约占国家总数的 10 %～20 % 时, ICA 的求解质量较为理想[6-7], 因此本文帝国个数取 15. GA 交叉概率设置为 0.950, 变异概率设置为 0.005. 所有对比测试均采用全浮点数运算, 实验结果如表 1 所示.

表 1 中偏差率为两种算法所得的最短路径和已知最优解的差除以已知最优解的百分比比值, 若计算结果优于已知最优解, 则表示为 "—", 即找到了更优路径. 本文已知最优解部分来自 TSPLIB 标准库, 其余来自参考文献[13-14]. 已知最优解不全为浮点型, 因此偏差率在 1 % 以内的解, 实际上已接近已知最优解.

从表 1 可以看出, 在求解时间方面, 30 组实例中 ICA 比 GA 求解时间少的有 24 组, 平均用时少 3.61 s. 在种群规模小于 200 的情况下, 两种算法获得最优解的时间相差较小, ICA 平均用时较 GA 少 1.26 s; 在种群规模大于 200 的情况下, ICA 算法获得最优解所需要的运行时间明显较短, 平均用时较 GA 少 7.77 s. 在求解质量方面, ICA 求解质量优于 GA. GA 仅有 4 组实例偏差率低于 1 %, 比例仅为 13 %; 而 ICA 中有 22 组偏差率低于 1 %, 比例高达 73 %, 且其中有 7 组解的质量达到或超过当前已知最优值. 在实例 pr107、pr136 和 tsp225 中, 本文算法计算得到的解优于已知最优解.

TSPLIB 测试库中实例名称中的数字代表种群数量. 在求解种群数量 200 以下的中小型规模 TSP 问题上, GA 容易陷入局部最优, 求解平均误差约为 3.23 %; 而 ICA 得到结果非常接近已知最优解, 求解平均误差不到 0.34 %, 相差 9 倍以上. 随着种群数量增多, GA 所得结果偏差率也随之呈指数型增大, 在种群数量超过 400 时, 求解偏差率超过 10 %; 而 ICA 仍能够保证良好的求解质量, 偏差率维持在 6 % 以内, 表明算法具有很强的鲁棒性. 本文算法解决部分实例的最优路线如图 3～图 6 所示.

本文算法在与 GA 对比的基础上, 还与其他几种常见的算法在不同规模种群数量上进行了横向比较. 表 2 是本文算法所得到的结果与其他算法所得结果的比较, 其中包括 IMGRA[14]、GCGA[15]、ACS[16]和 OMACO[16].

表 1 ICA与GA实验结果对比

编号	实例	已知最优解	GA			ICA		
			最优解	偏差率/%	时间/s	最优解	偏差率/%	时间/s
1	a280	2 579.00	2 796.46	8.432	41.39	2 606.17	1.054	36.81
2	att48	33 523.71	33 523.71	0.000	2.13	33 523.71	0.000	2.65
3	att532	86 729.00	98 551.897	13.632	68.35	89 212.48	2.863	64.43
4	bier127	118 282.00	120 406.48	1.796	17.37	118 703.56	0.356	16.21
5	ch130	6 110.00	6 303.59	3.168	8.71	6 127.39	0.285	8.73
6	ch150	6 528.00	6 790.31	4.018	11.43	6 543.62	0.239	11.21
7	d198	15 780.00	16 149.24	2.340	30.22	15 872.19	0.584	27.10
8	d493	35 002.00	38 292.87	9.402	64.32	36 239.32	3.535	51.71
9	eil51	426.00	435.74	2.286	5.03	428.87	0.674	3.94
10	eil76	538.00	550.87	2.392	9.47	544.37	1.184	6.53
11	eil101	629.00	660.19	4.959	13.27	640.21	1.782	13.83
12	kroA100	21 282.00	22 353.41	5.034	12.03	21 285.44	0.016	11.28
13	kroA200	29 368.00	31 268.11	6.470	21.73	29 430.88	0.214	18.72
14	lin318	42 029.00	45 547.60	8.372	38.36	42 307.53	0.663	29.85
15	oliver30	423.74	423.74	0.000	0.44	423.74	0.000	0.51
16	pr107	44 303.00	46 674.54	5.353	16.69	44 301.68	—	17.90
17	pr136	96 772.00	101 188.67	4.564	18.71	96 770.92	—	16.36
18	pr152	73 682.00	77 969.70	5.819	19.72	73 683.64	0.002	18.62
19	pr299	48 191.00	51 951.83	7.804	48.22	48 640.22	0.932	40.40
20	pr439	107 217.00	118 084.52	10.136	63.23	107 917.10	0.653	59.36
21	rand50	5 553.00	5 555.52	0.045	2.10	5 555.52	0.045	1.95
22	rand75	7 053.90	7 053.90	0.000	2.63	7 053.90	0.000	1.99
23	rand100	7 891.44	8 130.23	3.026	15.41	7 891.44	0.000	10.54
24	rand200	10 649.00	11 312.22	6.228	28.27	10 685.66	0.344	24.53
25	rand300	11 865.00	12 949.94	9.144	40.94	11 917.58	0.443	43.12
26	rand400	14 722.00	16 638.64	13.019	66.61	15 076.37	2.407	52.33
27	rat99	1 211.00	1 257.69	3.855	9.36	1 219.24	0.680	9.41
28	rat575	6 773.00	8 016.25	18.356	78.26	7 036.84	3.895	71.68
29	rat783	8 806.00	10 469.57	18.891	122.46	9 247.50	5.014	103.97
30	tsp225	3 916.00	4 220.35	7.772	35.34	3 907.36	—	28.35

表 2 ICA 与其他算法所得最优解对比

实例 (已知最优解)	算法	所得最优解	实例 (已知最优解)	算法	所得最优解
	IMGRA	429.53		IMGRA	45 036.36
	ACS	440.79		ACS	45 815.46
eil 51(426.00)	OMACO	428.87	lin 318(42 029.00)	OMACO	42 371.29
	GCGA	427		GCGA	43.600
	ICA	428.87		ICA	42.307.53
	IMGRA	125 996.17		IMGRA	119 790.94
	ACS	123 365.27		ACS	113 797.02
bier 127(118 282.00)	OMACO	118 767.45	pr 439(107 217.00)	OMACO	108 651.12
	GCGA	120 516		GCGA	112 781
	ICA	118 703.56		ICA	107 917.10
	IMGRA	3 010.90		IMGRA	7 259.06
	ACS	3 005.50		ACS	7 865.38
a 280(2 579.00)	OMACO	2 606.32	rat 575(6 773.00)	OMACO	7 332.81
	GCGA	–		GCGA	–
	ICA	2 606.17		ICA	7 036.84

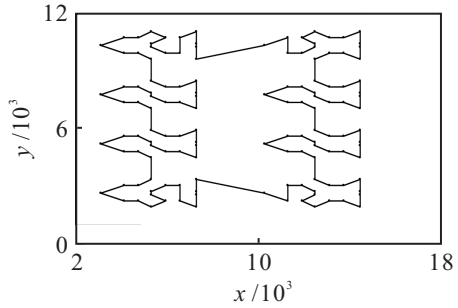

图 3　pr 136 最优路线图

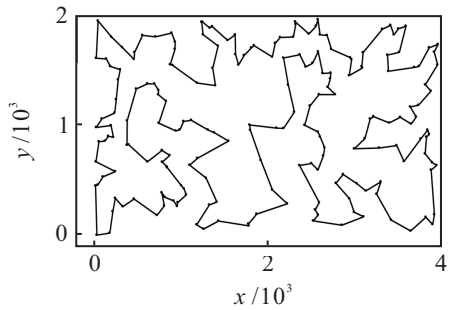

图 4　kroA 200 最优路线图

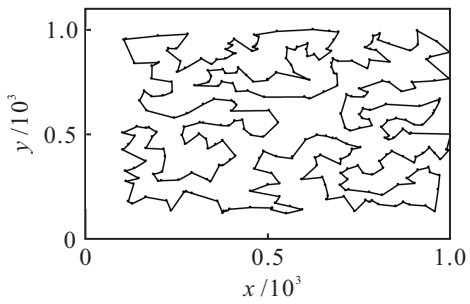

图 5　rand 300 最优路线图

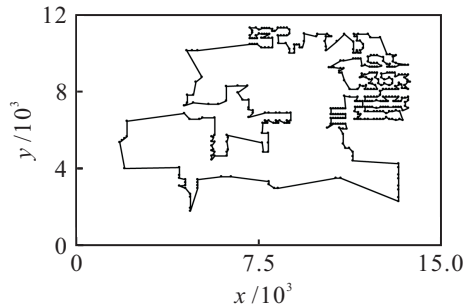

图 6　pr 439 最优路线图

从表 2 可以看出, 除了实例 eil51, ICA 在这 6 个实例中均取得比其他算法较优的解, 进一步说明了本文算法在不同规模问题上求解结果都较为可靠. 结合表 1 还可以看出, 在较大规模 TSP 问题上 (如 rat575), 虽然 ICA 求解结果没有达到已知最优解, 但所得结果均明显优于所对比的算法结果, 表明 ICA 全局搜索能力较强, 具有良好的寻优效果. 综合以上分析, 新型帝国竞争算法在 TSP 问题上求解质量较高.

为了评估 ICA 收敛速度, 分别用 ICA 和 GA 计算 TSPLIB 测试库中的 3 组实例, 并记录每一代两种算法求得的路径距离. 图 7～图 9 是 ICA 和 GA 计算得到迭代次数与路径距离的关系图.

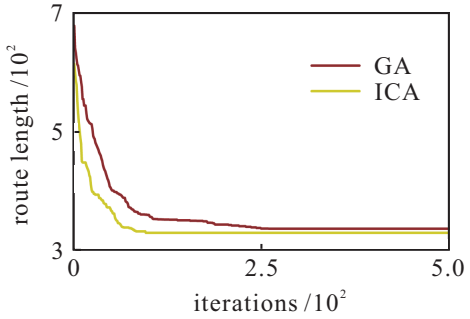

图 7　eil 51 实例 ICA 与 GA 路径随迭代次数变化曲线

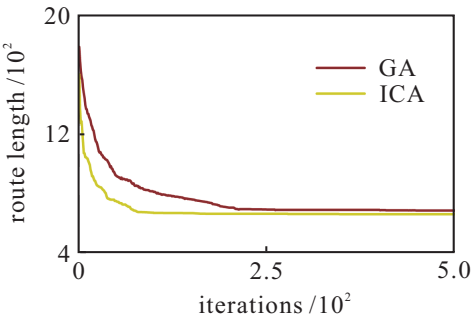

图 8　ch 150 实例 ICA 与 GA 路径随迭代次数变化曲线

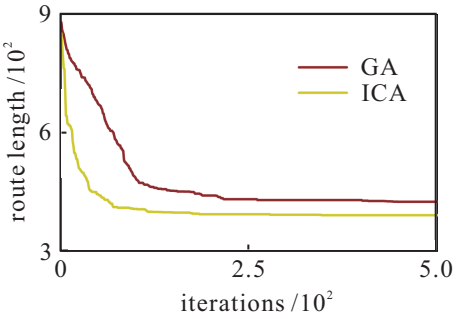

图 9　tsp 225 实例 ICA 与 GA 路径随迭代次数变化曲线

从图 7～图 9 可以看出, ICA 收敛速度明显较快, 能够在较早代数寻找到质量较优解. 随着种群数量的增加, GA 收敛速度明显放缓, 而本文算法收敛速度变化较小.

以实例 tsp225 (图 9) 为例, 虽然 ICA 在第 182 代时已趋于稳定, 但之后路径长度仍有 16 次变化, 最终在第 341 代找到最优值 3 907.36, 这表明 ICA 在达到较高质量解之前算法没有停滞; 而 GA 在第 261 代趋于稳定, 之后路径长度仅有 6 次变化, 在 446 代不再变化, 长度为 4 220.35. 在算法执行初期, ICA 通过同化和帝国增强过程加快寻优速度, 能够有效地高速收敛; 在后期, 又通过革命机制增加搜索空间, 避免陷入局部最优解. 由此可见, 本文算法收敛速度较快, 求解质量较高.

4　结　论

本文提出了一种求解 TSP 问题的新型帝国竞争算法, 拓展了帝国竞争算法在离散组合优化问题上的应用. 通过改变传统帝国竞争算法中初始帝国的建立、同化、革命、殖民竞争机制, 以及增加帝国增强过

程, 可以保证算法快速收敛的同时有效地提高全局搜索能力. 通过实验可以看出, 新型帝国竞争算法在求解质量和收敛速度方面都明显较优. 实验结果表明, 将帝国竞争算法应用于求解TSP问题是可行有效的, 将其应用于更多组合优化问题将是下一步的研究方向.

参考文献(References)

[1] 高海昌, 冯博琴, 朱利. 智能优化算法求解TSP问题[J]. 控制与决策, 2006, 21(3): 241-247.
(Gao H C, Feng B Q, Zhu L. Reviews of the meta-heuristic algorithm for TSP[J]. Control and Decision, 2006, 21(3): 241-247.)

[2] Mohammadi-ivatloo B, Rabiee A, Soroudi A. Imperialist competitive algorithm for solving non-convex dynamic economic power dispatch[J]. Energy, 2012, 44(1): 228-240.

[3] Forouharfard S, Zandieh M. An imperialist competitive algorithm to schedule of receiving and shipping trucks in cross-docking systems[J]. The Int J of Advanced Manufacturing Technology, 2010, 51(9): 1179-1193.

[4] Devika K, Jafarian A, Nourbakhsh V. Designing a sustainable closed-loop supply chain network based on triple bottom line approach: A comparison of metaheuristics hybridization techniques[J]. European J of Operational Research, 2014, 235(3): 594-615.

[5] Rahimi A, Karimi H, Afshar-Nadjafi B. Using meta-heuristics for project scheduling under mode identity constraints[J]. Applied Soft Computing, 2013, 13(4): 2124-2135.

[6] Hosseini S, Khaled A. A survey on the Imperialist Competitive Algorithm metaheuristic: Implementation in engineering domain and directions for future research[J]. Applied Soft Computing, 2014, 24(11): 1078-1094.

[7] Ardalan Z, Karimi S, Poursabzi O, et al. A novel imperialist competitive algorithm for generalized traveling salesman problems[J]. Applied Soft Computing, 2015, 26(1): 546-555.

[8] Atashpaz-Gargari E, Lucas C. Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition[C]. IEEE Congress on Evolutionary Computation. Singapore: IEEE, 2007: 4661-4667.

[9] Shabani H, Vahidi B, Ebrahimpour M. A robust PID controller based on imperialist competitive algorithm for load-frequency control of power systems[J]. ISA Transactions, 2013, 52(1): 88-95.

[10] 连坤雷. 殖民竞争算法在离散制造系统优化问题中的应用研究[D]. 武汉: 华中科技大学机械科学与工程学院, 2012.
(Lian K L. Colonial competitive algorithm and its applications in optimization of discrete manufacturing system[D]. Wuhan: School of Mechanical Science & Engineering of HUST, Huazhong University of Science and Technology, 2012.)

[11] 于莹莹, 陈燕, 李桃迎. 改进的遗传算法求解旅行商问题[J]. 控制与决策, 2014, 29(8): 1483-1488.
(Yu Y Y, Chen Y, Li T Y. Improved genetic algorithm for solving TSP[J]. Control and Decision, 2014, 29(8): 1483-1488.)

[12] Mirhoseini S H, Hosseini S M, Ghanbari M, et al. A new improved adaptive imperialist competitive algorithm to solve the reconfiguration problem of distribution systems for loss reduction and voltage profile improvement[J]. Int J of Electrical Power & Energy Systems, 2014, 55(2): 128-143.

[13] 冀俊忠, 黄振, 刘椿年. 基于多粒度的旅行商问题描述及其蚁群优化算法[J]. 计算机研究与发展, 2010, 47(3): 434-444.
(Ji J Z, Huang Z, Liu C N. An ant colony algorithm based on multiple-grain representation for the traveling salesman problems[J]. J of Computer Research and Development, 2010, 47(3): 434-444.)

[14] 饶卫振, 金淳, 陆林涛. 考虑边位置信息的求解ETSP问题改进贪婪算法[J]. 计算机学报, 2013, 36(4): 836-850.
(Rao W Z, Jin C, Lu L T. An improved greedy algorithm with information of edges' location for solving the Euclidean traveling salesman problem[J]. Chinese J of Computers, 2013, 36(4): 836-850.)

[15] Yang J H, Wu C G, Lee H P, et al. Solving traveling salesman problems using generalized chromosome genetic algorithm[J]. Progress in Natural Science, 2008, 18(7): 887-892.

[16] 周永权, 黄正新. 求解TSP的人工萤火虫群优化算法[J]. 控制与决策, 2012, 27(12): 1816-1821.
(Zhou Y Q, Huang Z X. Artificial glowworm swarm optimization algorithm for TSP[J]. Control and Decision, 2012, 27(12): 1816-1821.)

(责任编辑: 齐 霁)