禹博文,游晓明,刘 升. 基于空间聚焦机制的混沌随机蚁群算法[J]. 智能计算机与应用,2024,14(9):1-9. DOI:10.20169/j.issn.2095-2163.240901

基于空间聚焦机制的混沌随机蚁群算法

禹博文, 游晓明, 刘 升
(上海工程技术大学 电子电气工程学院, 上海 201620)

摘 要: 为了提高蚁群算法在大规模旅行商问题上的性能,本文提出了一种基于空间聚焦机制的混沌随机蚁群算法(CRACS)。首先,使用空间聚焦机制将旅行商问题划分为几个子类;其次,提出了一种改进混沌随机蚁群算法来求解子类;最后,将各个子类路线进行聚焦,选择最优连接路线动态连接各个子类,形成旅行商问题的最优解决方案。本文选取不同规模的典型旅行商问题进行了仿真实验,实验结果表明本文提出的算法在大规模问题上具有优秀的性能,相比于其他智能算法具有更优质的解和更快的搜索速度。

关键词: 蚁群算法; 旅行商问题; 空间聚焦机制; 混沌随机
中图分类号: TP391 **文献标志码**: A **文章编号**: 2095-2163(2024)09-0001-09

Chaotic random Ant Colony Algorithm based on spatial focusing mechanism

YU Bowen, YOU Xiaoming, LIU Sheng
(College of Electronic and Electrical Engineering, Shanghai University of Engineering Science, Shanghai 201620, China)

Abstract: In order to improve the performance of Ant Colony Algorithm in large-scale traveling salesman problems, this paper proposes a chaotic random Ant Colony Algorithm based on spatial focusing mechanism (CRACS). Firstly, the spatial focusing mechanism is used to divide the traveling salesman problem into several subcategories. Secondly, an improved chaotic random Ant Colony Algorithm is proposed to solve the subcategories. Finally, the routes of each subcategory are focused, and the optimal connecting route is dynamically connected to each subcategory to form the optimal solution to the traveling salesman problem. This article selects typical traveling salesman problems of different scales for simulation experiments, and the experimental results show that the algorithm proposed in this paper has excellent performance, especially in large-scale problems. Compared with other intelligent algorithms, the algorithm proposed in this paper has better solutions and faster search speed.

Key words: Ant Colony Algorithm; traveling salesman problem; spatial focusing mechanism; chaotic random

0 引 言

群智能算法是通过模拟自然界中个体的行为,在目标空间中搜索局部最优解并与其他个体进行交流,最终形成全局最优解的启发式算法。群智能算法具有适合分布式计算,计算速度快等优点,目前应用广泛的群智能算法包括粒子群算法(PSO)、模拟退化法(SA)、人工鱼群算法(FSS)、蚁群算法(ACS)等,其中蚁群算法作为群智能算法的代表应被广泛应用。

蚁群算法(Ant Colony System,ACS)是意大利学者 Dorigo [1] 于 1996 年提出的一种启发式全局搜索算法,通过模仿自然界中蚂蚁释放信息素寻路的过程实现路径搜索。蚁群算法具有鲁棒性强、参数简单等优点,并且具有很好的可移植性,能应用在不同种类的问题上,广泛应用于求解车间调度、路径规划、路由问题和旅行商问题上。由于蚁群算法具有众多优点,许多学者对蚁群算法进行了改进。Yu[2] 等将聚类算法与推荐算法引入蚁群算法,通过对旅行商问题中的城市节点聚类使蚁群能够更高效地对路径进行搜索,而推荐算法能够帮助避免蚁群信息素过于集中;Gülcü

基金项目: 国家自然科学基金(61673258,61075115); 上海市自然科学基金(19ZR1421600)。
作者简介: 禹博文(1997-),男,硕士研究生,主要研究方向:智能算法,机器学习,人工智能; 刘 升(1966-),男,博士,教授,硕士生导师,主要研究方向:量子启发式进化算法,分布式并行处理,进化算法。
通讯作者: 游晓明(1963-),女,博士,教授,硕士生导师,主要研究方向:群智能系统,分布式并行处理,进化算法。Email:yxm6301@163.com
收稿日期: 2023-04-22

等[3]将 3OPT 算法与蚁群算法相结合,通过 3OPT 算法提高蚁群算法的局部搜索能力;王原等[4]将神经网络与蚁群算法相结合,利用注意力机制选择蚁群算法中的启发因子,提高了蚁群算法的全局搜索能力;Tuani 等[5]在引入近邻搜索算法的同时,动态调整蚁群算法中的信息素启发因子和距离启发因子,提高了算法在不同规模问题上的搜索速度;陈佳等[6]引入信息熵理论,通过计算信息熵判断蚁群的离散度,根据系统的离散程度调整算法参数,提高算法的求解能力;陈颖杰等[7]首先使用贪心算法为蚁群算法赋予一个初始解,然后计算初始解的次优解,对次优解也给予一定的信息素奖励,加快了解的收敛速度,在求解后期加入变异操作,提高当前最优路径附近的路径上的信息素浓度;冯振辉等[8]将刺激响应机制加入蚁群算法,根据不同任务设置响应阈值,按照不同任务的刺激大小对蚂蚁进行分工,当任务刺激超过某只蚂蚁的相应阈值后,将该蚂蚁分配至对应任务,同时引入正负反馈调节机制,根据种群多样性按正负反馈调节蚂蚁数量,平衡了收敛速度与全局搜索能力;张鹏等[9]提出一种异构多种群蚁群算法,将蚁群分为多个种群,通过计算种群间的相似度,自适应选择互补程度最高的种群进行信息交换,提高了算法跳出局部最优解的能力;庞永杰等[10]在分种群搜索的基础上,将遗传算法中的排序策略引入蚁群算法中,有效缓解了因信息素浓度失衡而造成的局部收敛,提高了算法的搜索能力;Ebadinezhad 等[11]利用聚类算法改进蚂蚁初始点的原则策略,并根据聚类的大小动态调整信息素蒸发率,减小了算法在大规模问题上误差率;Yong 等[12]提出一种混合最大最小蚁群算法,使用最大最小蚁群算法搜索近似的最优哈密顿回路,通过四点三线不等式对最大最小蚁群算法的结果进行约束,使算法能够获得更好的近似解;Zhang[13]提出一种结合拥挤因子和协同进化机制的多种群蚁群算法,通过调整拥挤因子防止一条路径上集中过多信息素,利用协同进化机制定期交换不同种群的信息素,提高算法求解的多样性。

由于蚁群算法在处理大规模问题时存在收敛精度低、收敛速度慢的问题[14]。本文提出了一种基于空间聚焦机制的混沌随机蚁群算法(Chaotic Random Ant Colony algorithm based on pointer network and dynamic fusion Strategy, CRACS)。首先,使用空间聚焦机制将大规模旅行商问题划分为多个子集,使大规模问题变成求解多个小规模问题,降低大规模问题的复杂度,并对各个子问题的解进行聚焦,选择出最佳连接方式进行动态融合,形成最终解;其次,提出混沌随机蚁群算法来寻找每个子问题的最优路径,使信息素混沌释放以拓展蚁群搜索空间并加快算法的收敛速度;最后,使用不同规模的旅行商问题测试集评估了本文算法的有效性,结果表明本文算法的求解速度比传统蚁群算法最多提高了 5.7 倍,在大规模问题上求解精度也有较大的提升。

1 算法原理

1.1 蚁群算法

蚁群算法是一种模拟自然界蚁群觅食行为的启发式算法。蚂蚁在觅食过程中会留下信息素,其他蚂蚁在前进时会更大概率选择信息素浓度较高的路径,相同时间内在较短路径上积累的信息素变多,选择该路径的蚂蚁数量也变多,形成正反馈。每只蚂蚁走过一段路径后,其留下的信息素就会立即挥发一部分,$\Delta\tau(i,j)$ 为 i 与 j 之间的信息素增量:

$$\tau(i,j) = (1-\rho)\cdot\tau(i,j) + \Delta\tau(i,j) \tag{1}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m}\tau_{ij}^{k} \tag{2}$$

其中,ρ 是信息素的局部挥发率。

释放信息素后,蚂蚁按照下式选择下一个节点,其中 J 和 η 的计算如下:

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}(t)\cdot\eta_{ij}^{\beta}(t)}{\sum\tau_{is}(t)\cdot\eta_{is}^{\beta}(t)}, & j\in allowed_k \\ 0, & s\in allowed_k \\ & \text{else} \end{cases} \tag{3}$$

$$J = \begin{cases} \arg\max[\tau_{ij}\eta_{ij}^{\beta}], & q\le q_0 \\ p_{ij}^{k}, & q > q_0 \end{cases} \tag{4}$$

$$\eta = \frac{1}{d_{ij}} \tag{5}$$

其中,p_{ij}^{k} 表示蚂蚁 k 从节点 i 到节点 j 的概率;$\tau_{ij}(t)$ 表示节点 i 到 j 之间 t 时刻的信息素总量;η 为节点 i 到 j 之间距离的倒数,$allowed_k$ 表示蚂蚁 k 还未走过的节点。相比蚂蚁算法,蚁群算法改进了路径选择公式,增加了控制参数 q_0 和随机数 q。蚂蚁在选择路线时不完全按照信息素浓度选择,按照一定概率选择其他路线,提高了算法在陷入局部最优路线后跳出局部最优的能力。

所有蚂蚁遍历过一次后进行信息素的全局更新,对所有路径上的信息素进行更新,更新方式如下:

$$\tau(i,j) = (1-\alpha)\cdot\tau(i,j) + \alpha\cdot\Delta\tau(i,j) \tag{6}$$

$$\Delta\tau(i,j) = \frac{Q}{d_{\text{best}}} \tag{7}$$

其中，α 为全局信息素挥发系数，代表上一次迭代后信息素的残留比例；Q 为信息素更新权重；d_{best} 为全局最短路径。

蚁群算法在全局信息素更新时只留下全局最优蚂蚁的信息素，并且随着迭代次数的增加，最优路径上的信息素不断累加，会导致算法陷入局部最优。

1.2　近邻传播聚类

近邻传播聚类（AP Cluster）是 Frey 等[15]在 2007 年提出的一种聚类算法。AP 聚类算法主要根据两点之间的关系进行聚类，该算法计算点之间的适应度和吸引度，然后迭代更新适应度和吸引度矩阵，最后计算每个节点的吸引度和适应度之和。当聚类中心点没有改变或算法超过最大迭代次数时，聚类中心即被确定。吸引度的计算如下：

$$r(i,k) \leftarrow s(i,k) - \max_{k's.t.\,k'\neq k}\{a(i,k) + s(i,k)\} \quad (8)$$

其中，$r(i,k)$ 为节点 k 对节点 i 的吸引度；$s(i,k)$ 为节点 i 作为聚类中心的参考度，参考度越大则节点 i 更有可能成为聚类中心；$a(i,k)$ 为节点 i 对节点 k 的适应度，$a(i,k)$ 越大表明相比于其他节点，节点 i 更有可能选择节点 k 作为聚类中心，$a(i,k)$ 的计算方法如下：

$$a(i,k) \leftarrow \min\{0, r(k,k) + \sum_{i's.t.\,i'\notin\{i,k\}} \max\{0, r(i',k)\}\} \quad (9)$$

通过不断计算每个节点吸引度 $r(i,k)$ 与适应度 $a(i,k)$ 的值，最终吸引度与适应度之和最大的节点即被选为节点 i 的聚类中心，计算如下：

$$a(k,k) \leftarrow \sum_{i's.t.\,i'\neq k} \max\{0, r(i',k)\} \quad (10)$$

类的数量与每个点的参考度 $p(k)$ 的值有关：

$$r(k,k) = p(k) - \max\{a(i,k) + s(i,k)\} \quad (11)$$

$p(k)$ 的值越大，节点 k 就更有可能被选为聚类中心，并且类的数量也会增多，近邻传播聚类具有对初始值不敏感，无需指定类的数目等优点。

1.3　指针网络

近年来，用于解决组合优化问题的深度学习方法逐渐受到关注[16]。在组合优化问题中，例如旅行商问题，输入是城市的坐标序列，输出也是城市的坐标顺序，每次求解的旅行商问题的大小 n 不是固定的。传统的序列到序列方法不能解决字典长度可变的问题，因此 Vinyals 等[17]在 2015 年提出指针网络，将注意力机制作为指针，动态选择每次输出的解，解决了输入输出序列长度变化问题，其核心机制如式(12)和式(13)所示，u_j^i 表示解码器在第 i 步时，输入序列中第 j 个元素与当前解码状态的注意力分数，衡量了输入序列中第 j 个元素对于生成当前输出元素的重要性；$p(C_i \mid C_1,\cdots,C_{i-1},P)$ 表示在给定先前输出 $C_1,\cdots,C_{i-1}$ 和输入序列 P 的条件下，当前输出为 C_i 的概率。在指针网络中，C_i 是输入序列 P 中的一个索引，指向 P 中的一个元素作为当前输出。

$$u_j^i = v^{\mathrm{T}}\tanh(W_1 e_j + W_2 d_i),\ j \in (1,\cdots,n) \quad (12)$$

$$p(C_i \mid C_1,\cdots,C_{i-1},P) = \mathrm{Softmax}(u^i) \quad (13)$$

其中，v^{T}，W_1 和 W_2 是待训练的参数；e_j 为 j 时刻编码器隐藏层的输出；d_i 为解码器在 i 时刻的输出。指针网络将网络的输出作为输入城市序列中每个位置输出的概率，并将具有最高概率的点作为下一步的输出，将具有最高注意力概率的节点作为每次要前往的节点。

指针网络将注意力机制引入到旅行商问题中，增强了算法对旅行商问题的特征提取能力。然而，目前的指针网络无法处理大规模问题。随着问题规模的增加，网络训练的时间成本变得不可接受，效果也会迅速下降。

2　基于空间聚焦机制的混沌随机蚁群算法

2.1　空间聚焦机制

由于蚁群算法在求解大规模问题时存在收敛速度慢，解的精度低等问题，因此本文提出空间聚焦机制，降低目标空间的复杂度并提高连接精度。首先，使用近邻传播聚类算法将旅行商问题（TSP）划分为几个子集；其次，用改进的蚁群算法分别求解每个子类，降低目标空间的复杂度，大大加快蚁群算法的求解速度；最后，使用指针网络对目标空间中最优连接顺序进行聚焦，确定子集之间的连接顺序，通过比较不同的连接点，形成最佳连接方法，将各子集动态融合为最终解决方案。

首先，使用近邻传播聚类对目标空间进行划分，近邻传播聚类的聚类数目由参考度方程 $p(k)$ 决定，当 $p(k)$ 较小时，聚类数目也较少。对于旅行商问题来说，由于类间的连接需要损失一定的精度，当节点数较少时尽量减少聚类的数量；当节点数量增多时，聚类后的每一类中的节点数也相应增加，这时应该尽量增加聚类的数量以增加蚁群算法在每个子类上的求解效率。因此，设置规模参数 K 并根据不同规模的旅行商问题选择不同的参考度方程，选择公式(14)：

$$p = \begin{cases} \min(\boldsymbol{s}), & \text{otherwise} \\ -\inf, & \text{if } n < K \end{cases} \quad (14)$$

其中，K 为规模参数；n 为城市数量；$\boldsymbol{s}$ 为所有城市的相似度矩阵。

当城市数量 n 小于规模参数 K 的时候，参考度

方程设置为负无穷,即将所有节点聚为一类;当 n 大于 K 时,参考度方程设置为相似度矩阵中的最小值。通过规模参数 K 可以保证当数据集规模较小时避免因类间连接带来的精度损失,也可以缩短大规模旅行商问题的运行时间。

聚类完成后,使用混沌随机蚁群算法找到每个类的解。当所有子类路径构造完成时,需要确定每个类之间的连接方式。首先,将每个类抽象为由聚类中心表示的点,并将子类之间的连接顺序转换为点之间的连接次序;其次,将每个子类的聚类中心的坐标输入到指针网络中,通过指针网络依次聚焦于目标空间中概率最大的点,以确定连接顺序如图 1 所示,连接顺序的结果如图 2 所示。

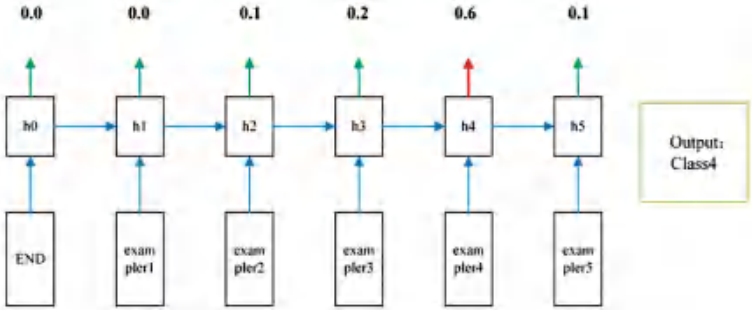

图 1 指针网络确定连接顺序

Fig. 1 Connection sequence chosen by pointer network

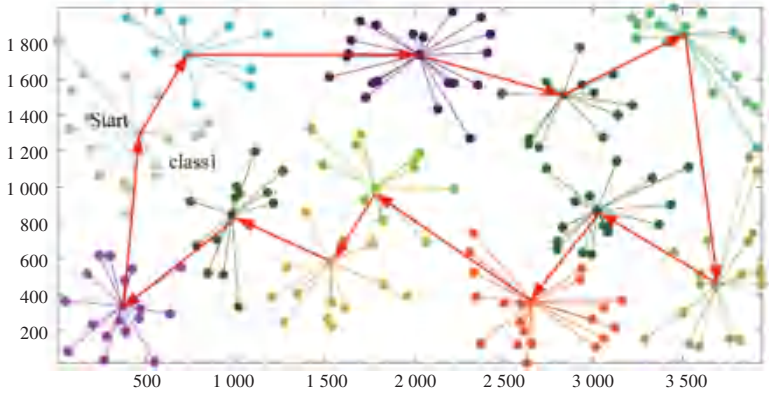

图 2 各类之间连接顺序

Fig. 2 Connection order of classes

确定连接顺序后,按顺序选择最优连接点。首先,找到两个类中最近的两个节点并将其连接起来;其次,在两个类的路径列表中找到与最近节点相邻的两个节点,并将每个类的这两个节点确定为相邻节点;将需要连接的两个类中第一个类的相邻节点与第二类的邻近节点连接,存在 4 个不同的连接方式,空间聚焦机制选择连接距离最短的路径作为两个类之间的连接路径;最后,把两个连接好的类合并为一个新类,并继续与下一个类连接,直到所有子类都合并为一个大类。子类之间连接路径的 4 种情况如图 3 所示,选择 4 种连接方式中总路径长度最短的路线作为两个类之间的连接路线。

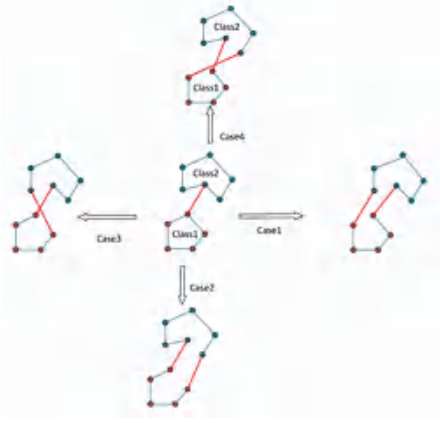

图 3 类间连接方式

Fig. 3 Connect cases between subclasses

2.2　混沌随机蚁群算法

聚类完成后，需要求出每个子类的解。由于每个类的大小并不固定，传统蚁群算法的固定参数无法在每个类中都有很好的表现，在多次迭代后容易陷入局部最优。为了解决这个问题，本文提出了一种混沌随机蚁群算法，根据不同类的大小设置不同的算法参数：

$$m = round(n/10) \tag{15}$$

其中，m 为每一类中的蚂蚁个数，n 为每一类中的城市数量。

在传统蚁群算法的初期阶段根据贪心算法进行初始化，各条路径上信息素为 0，算法只能以距离信息进行搜索，导致搜索速度过慢，容易在初期就陷入局部最优路径。由于传统蚁群算法在初始化时采用伪随机策略，在大规模复杂 TSP 问题上很容易降低种群的多样性，导致某些路径根本无法搜索。为了解决以上问题，在蚁群搜索的早期阶段，让信息素的量在每条路径上无序更新，使得随机量的信息素散布在各个路径，以提高算法初期的能见度。当算法陷入局部最优时，意味着信息素集中在局部最优路径上，导致其他路径根本无法被探索。为了解决这种问题，在算法陷入局部最优时打乱信息素矩阵，以在算法停滞后改变蚂蚁的路线。如果最优路径在多次迭代后保持不变，即假设该算法陷入了局部最优。公式(16)表示若算法经过 L 次迭代后，最优路径仍未改变，则认为算法陷入局部最优。

$$L = round(\frac{\max\ iter}{m}\lg(n)) \tag{16}$$

其中，m 为蚂蚁总数，n 为类内城市总数。

当算法陷入局部最优路径后，所有路径上的信息素将会随机蒸发或增加以增大搜索空间。

信息素更新公式如下：

$$\tau_{ij}(i+1) \leftarrow q \cdot \tau_{ij}(i) + (1-q) \cdot \Delta\tau_{ij}(i) \pm \mathrm{Logistic}(c) \tag{17}$$

其中，Logistic(c) 为 Logistic 混沌函数，c 为控制常数，使混沌随机数与信息素保持同一个数量级。

通过混沌函数可以产生完全随机的随机序列，x_{n+1} 和 x_n 是混沌序列的值[18]，表达式如下：

$$x_{n+1} = k \cdot x_n \cdot (1 - x_n) \tag{18}$$

其中，k 是混沌因子。

混沌序列的作用是产生一系列非线性和不可预测的随机数，当取 3.569 到 4 时，这些随机数是完全随机的。

2.3　基于空间聚焦机制的混沌随机蚁群算法(CRACS)算法流程

本文提出基于空间聚焦机制的混沌随机蚁群算法的流程：

CRACS 算法求解旅行商问题步骤
1. 输入：旅行商问题的城市坐标
2. 初始化各项参数
3. 空间聚焦机制将旅行商问题分为若干子类
4. for i = 1：子类数量 *do*
5.　　for j = 1：最大迭代次数 *do*
6.　　初始化混沌随机蚁群算法
7.　　路径构建
8.　　if 路径更新停滞 then
9.　　随机重组每条路径上的信息素
10.
11.　　end if
12.　　end for
13. 运用空间聚焦机制动态融合各个类别
14. end for
15. 检查是否达到终止条件
16. if True then
17.　　结束
18. Else
19.　　返回第 4 步
20. end if
21. End

3　仿真实验及分析

为了检验 CRACS 算法的性能，本文利用旅行商问题(TSP)测试集进行算法的性能测试。每个测试集都是一组城市的坐标序列，测试集后面的数字表示该测试集中有多少个城市节点，分别选择蚁群算法、最大最小蚁群算法和本文提出的算法进行对比实验。

3.1　与传统方法对比结果

为了验证基于空间聚焦机制的混沌随机蚁群算法(CRACS)的效果，选择不同规模的 TSP 测试集，将 CRACS 算法分别与蚁群算法(ACS)和最大最小蚁群算法(MMAS)[19]进行对比实验，最优解的精度、平均解的精度、收敛时间和平均误差率，其中平均误差率按照下式计算：

$$\mathrm{Avg}Er = \frac{1}{25}\sum_{i=1}^{25} |x_i - \bar{x}| \tag{19}$$

其中，x_i 为第 i 次迭代时的最短路径长度，$\bar{x}$ 为 25 次实验的平均路径长度。

CRACS、ACS 和 MMAS 算法的参数选择见表 1；各算法在 TSP 测试集上的运行结果见表 2。通过表 2 可以看出，与 ACS 和 MMAS 相比，CRACS 具有更快的

收敛速度和更高的精度。在小型 TSP 问题上如 Eil51、KroA100,ACS 算法、MMAS 算法和 CRACS 算法几乎都找到了最佳路径,但本文算法比 ACS 算法、MMAS 算法的搜索速度快得多。随着 TSP 规模增加到中等规模,ACS 算法、MMAS 算法的搜索能力不足,导致求解精度快速下降,收敛速度缓慢,搜索时间平均为 CRACS 的 2 到 3 倍;由于连接时用各个子类的中心点代替类,大幅降低了算法的计算量,提高了算法的运行速度。在大规模 TSP 测试集中,例如 U1060、U2319 测试集,ACS 算法、MMAS 算法搜索速度大幅下降,而由于空间聚焦机制,CRACS 算法可以将大规模问题分解为几个小规模问题,从而降低搜索难度并加快搜索速度,相对于 ACS 算法、MMAS 算法,CRACS 的搜索速度和解的精度都大幅度高于 ACS 算法、MMAS 算法,搜索速度为其它几种算法的 5 到 6 倍。总之,CRACS 在收敛速度和精度之间实现了良好的平衡,尤其在大规模测试集上 CRACS 最优解的质量与搜索速度远远高于 ACS 算法、MMAS 算法。

表 1 CRACS、ACS、MMAS 算法参数选择

Table 1 Parameters setting of CRACS、ACS and MMAS algorithms

算法	α	β	ρ	ξ	q_0
CRACS	3	4	0.3	0.3	0.9
ACS	2	4	0.2	0.3	0.9
MMAS	1	4	0.1	–	–

表 2 CRACS 与 MMAS、ACS 对比结果

Table 2 Comparison between CRACS MMAS and ACS

TSP 测试集	标准解	算法	最优解	平均解	平均误差率	收敛时间/s
Eil51	426	CRACS	**426**	**427**	**2.00**	3.25
		MMAS	427	431	2.53	23.8
		ACS	426	428	5.81	**2.36**
Eil76	538	CRACS	**538**	**541**	3.28	5.80
		MMAS	543	549	**2.66**	27.20
		ACS	538	544	11.36	**3.05**
Rat99	1 211	CRACS	**1 211**	**1 214**	18.08	9.24
		MMAS	1 212	1 235	**13.44**	36.40
		ACS	1 213	1 219	17.04	**4.92**
KroA100	21 282	CRACS	**21 282**	**21 326**	**30.26**	**9.47**
		MMAS	21 292	21 475	31.84	33.63
		ACS	21 282	23 441	179.36	9.84
Eil101	629	CRACS	**630**	**635**	**3.44**	3.49
		MMAS	631	640	4.69	25.23
		ACS	631	639	7.84	**1.68**
Pr107	44 303	CRACS	**44 303**	**44 494**	173.50	14.08
		MMAS	44 387	44 547	115.52	43.70
		ACS	44 555	44 623	**107.84**	**4.86**
Ch130	6 110	CRACS	**6 144**	**6 154**	40.54	25.20
		MMAS	6 145	6 219	**17.50**	39.91
		ACS	6 153	6 220	25.40	**15.77**
KroA200	29 368	CRACS	**29 394**	**29 501**	**51.62**	**36.96**
		MMAS	29 803	30 359	176.28	100.44
		ACS	29 498	29 501	130.96	37.05
Tsp225	3 916	CRACS	**3 940**	**3 953**	**22.88**	**39.06**
		MMAS	4 104	4 192	30.54	79.92
		ACS	3 963	3 997	24.64	41.39
Rd400	15 281	CRACS	**15 610**	**15 750**	**34.91**	**54.17**
		MMAS	15 870	16 206	144.44	311.08
		ACS	16 709	16 799	99.12	122.31
FL417	11 861	CRACS	**11 946**	**11 991**	**15.60**	**55.48**
		MMAS	12 010	12 183	47.41	117.25
		ACS	12 057	12 213	33.44	102.2
Pr439	107 217	CRACS	**108 605**	**109 279**	**980.04**	**55.62**
		MMAS	117 104	122 830	2 758.75	296.46
		ACS	109 037	110 788	1 573.02	101.78
U1 060	224 094	CRACS	**230 331**	**238 060**	**1 052.30**	**133.58**
		MMAS	313 876	334 780	3 321.95	270.83
		ACS	251 784	261 384	2 175.28	511.18
U2 319	234 256	CRACS	**236 322**	**237 590**	**570.45**	**312.95**
		MMAS	311 308	332 480	4 501.12	2 247.50
		ACS	269 928	276 427	1 157.98	1 784.70

CRACS 和 ACS 算法收敛曲线对比如图 4 所示。通过图 4 可以看出，在算法初期，CRACS 能够快速收敛，在不同规模城市集上收敛速度均高于传统算法。在算法后期，蚁群算法由于搜索能力下降和陷入局部最优路线等原因几乎已经停止搜索，而由于 CRACS 的混沌随机蚁群算法，使 CRACS 在迭代后期也能保持较高的搜索效率和跳出局部最优的能力。

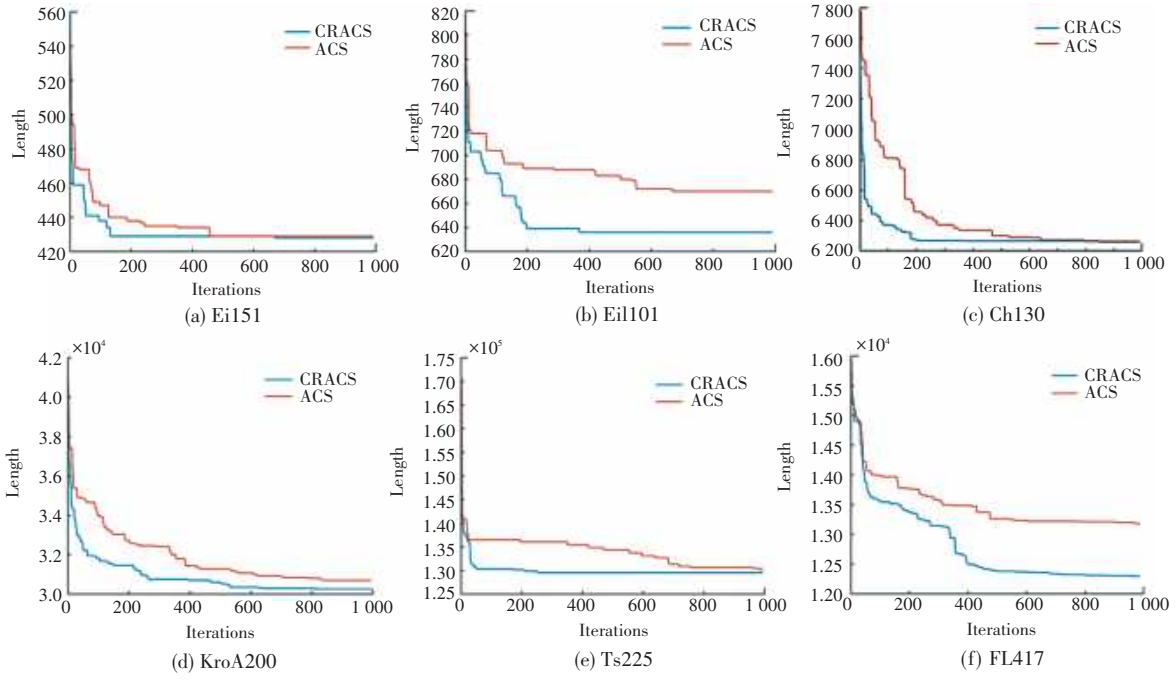

图 4　CRACS 与 ACS 收敛曲线对比图

Fig. 4　Convergence comparison of CRACS and ACS

3.2　与其他改进算法对比结果

为了进一步测试 CRACS 的性能，分别在大中小 3 种不同规模的测试集上进行测试并与基于蚁群优化和 3-OPT 算法的并行协同混合算法（Parallel operative hybrid method based on Ant Colony Optimization and 3-OPT algorithm，PACO-3OPT）和基于启发式信息的离散混洗蛙跳算法（Discrete Shuffled Frog-Leaping algorithm based on heuristic information，DSFL）[20] 进行对比实验。小规模测试集为城市规模小于 200 的数据集，大规模测试集为城市规模大于 500 的数据集。CRACS 与 PACO-3OPT 算法在中小规模旅行商问题参数设置见表 3，CRACS 与 PACO-3OPT 算法在大规模旅行商问题参数设置见表 4，其中 N 为蚂蚁种群数；m 为每个子类中的蚂蚁数，PACO-3OPT 中蚂蚁数为一常数，CRACS 中蚂蚁数与每一类的大小有关；I 为 PACO-3OPT 中的种群常数；Q 为 PACO-3OPT 的种群迁移间隔。“-”表示该方法未作此实验。

表 3　CRACS 与 PACO-3OPT 算法在中小规模旅行商问题参数设置

Table 3　Initialization of CRACS and PACO-3OPT parameters for middle-scale TSP and small-scale TSP

算法	N	m	I	ρ	Q	Max-iteration
PACO-3OPT	2	5	5	0.1	1	1 000
CRACS	1	round(size(*nodes*)/10)	-	0.1	-	1 000

表 4　CRACS 与 PACO-3OPT 算法在大规模旅行商问题参数设置

Table 4　Initialization of CRACS and PACO-3OPT parameters for large-scale TSP

算法	N	m	I	α	β	ρ	Q	Max-iteration
PACO-3OPT	4	5	5	3	2	0.1	30	1 000
CRACS	1	round(size(*nodes*)/10)	-	3	2	0.1	-	1 000

其中，N 为蚂蚁种群数，m 为每个子类中的蚂蚁数，PACO-3OPT 中蚂蚁数为一常数，CRACS 中蚂蚁数与每一类的大小有关；I 为 PACO-3OPT 中的种群常数；Q 为 PACO-3OPT 的种群迁移间隔。

“-”表示该方法未作此实验。

CRACS 与 PACO-3OPT、DSFL 算法中小规模 TSP 问题对比实验结果见表 5,当 TSP 测试集规模较小时,本文算法找到了 TSP 测试集的大部分最优路径,并且平均解接近最优。由于使用空间聚焦机制和混沌随机蚁群算法,城市数量增加到中等规模(200-500)时,本文的算法可以在 FL417 测试集和 Pr439 测试集上找到更好的解。通过表 5 可以看出,本文算法可以有效地扩展搜索空间并找到更好的解决方案,例如 KroA200 测试集,由于使用了空间聚焦机制,误差降低到 0.01%。

表 5 CRACS 与 PACO-3OPT、DSFL 在中小规模旅行商问题上的对比结果

Table 5 Results of CRACS PACO-3OPT and DSFL for middle-scale and small-scale TSP

TSP	标准解	CRACS			PACO-3OPT			DSFL		
		最优解	平均解	误差率	最优解	平均解	误差率	最优解	平均解	误差率
Eil51	426	**426**	427	0.00	**426**	427	0.00	**426**	**426**	0.00
Eil76	538	**538**	541	0.00	**538**	**538**	0.00	**538**	539	0.00
Rat99	1 211	**1 211**	1 214	0.00	1 213	1 217	0.50	**1 211**	1 216	0.00
KroA100	21 282	**21 282**	21 326	0.00	**21 282**	21 326	0.21	**21 282**	**21 312**	0.00
Eil101	629	630	635	0.01	**629**	**630**	0.25	**629**	632	0.00
KroA200	29 368	**29 394**	**29 501**	**0.01**	29 533	29 644	0.94	29 499	29 671	1.03
Rd400	15 281	15 610	15 750	2.10	15 578	15 614	1.91	-	-	-
FL417	11 861	**11 946**	**11 991**	0.71	11 972	11 987	0.94	-	-	-
Pr439	107 217	108 605	109 279	1.29	**108 482**	**108 702**	1.18	-	-	-

表 5 表明 CRACS 很好地提高了解的准确性。当 TSP 规模较小时,本文算法找到了 TSP 的大部分最优路径,并且平均解接近最优。由于使用空间聚焦机制和混沌随机蚁群算法,城市数量增加到中等规模(200-500)时,本文的算法可以在 FL417 和 Pr439 上找到更好的解,说明本文算法可以有效地扩展搜索空间并找到更好的解决方案,例如 KroA200,由于使用了空间聚焦机制,误差降低到 0.01%。

4 结束语

针对蚁群算法在求解大规模旅行商问题上存在的不足,本文提出了一种基于空间聚焦机制的混沌随机蚁群算法来解决旅行商问题。首先,设计空间聚焦机制将大规模旅行商问题分解成若干子问题以降低目标空间的复杂度并对各个子问题的解进行聚焦,选择出最佳连接方式进行动态融合,保证各子类路径连接为最优连接方式,提高优化解的精度;其次,为了提高算法的搜索能力,提出混沌随机蚁群算法,使信息素混沌释放以拓展蚁群搜索空间并加快算法的收敛速度;最后,通过在不同规模的旅行商问题上多次实验证明,改进算法在各种规模的旅行商问题上都有很好的表现,尤其在大规模问题上能够有效加快搜索速度并提升解的精度。

在研究过程中发现,聚类算法能够有效降低大规模旅行商问题的复杂度,提高算法效率,因此在接下来的工作中将进一步研究不同聚类算法在解决旅行商问题上的应用。

参考文献

[1] DORIGO M, MANIEZZO V, COLORNI A. Ant system: Optimization by a colony of cooperating agents[J]. Man, and Cybernetics Society, 1996, 26(1): 29-41.

[2] YU J, YOU X, LIU S. Dynamic density clustering ant colony algorithm with filtering recommendation backtracking mechanism [J]. IEEE Access, 2020, 8: 154471-154484.

[3] GÜLCÜ Ş, MAHI M, BAYKAN Ö K, et al. A parallel cooperative hybrid method based on ant colony optimization and 3-OPT algorithm for solving traveling salesman problem [J]. Soft Computing, 2018, 22(5): 1669-1685.

[4] 王原, 陈名, 邢立宁, 等. 用于求解旅行商问题的深度智慧型蚁群优化算法 [J]. 计算机研究与发展, 2021, 58(8): 1586-1598.

[5] TUANI A F, KEEDWELL E, COLLETT M. Heterogenous adaptive ant colony optimization with 3-OPT local search for the travelling salesman problem[J]. Applied Soft Computing, 2020, 97: 106720.

[6] 陈佳, 游晓明, 刘升, 等. 结合信息熵的多种群博弈蚁群算法[J]. 计算机工程与应用, 2019, 55(16): 9.

[7] 陈颖杰, 高茂庭. 基于信息素初始分配和动态更新的蚁群算法[J]. 计算机工程与应用, 2022, 58(2): 95-101.

[8] 冯振辉,肖人彬. 基于混合反馈机制的扩展蚁群算法[J]. 控制与决策,2022,37(12):3160-3170.

[9] 张鹏, 薛宏全, 原欣伟. 基于相似度的自适应异类多种群蚁群算法[J]. 计算机工程与应用, 2014 (19): 5.

[10] 庞永杰, 唐旭东, 李晔. 基于改进精英机制的双种群蚁群算法

[J]. 自动化技术与应用, 2008, 27(2): 5.
[11] EBADINEZHAD S. DEACO: Adopting dynamic evaporation strategy to enhance ACO algorithm for the traveling salesman problem[J]. Engineering Applications of Artificial Intelligence, 2020, 92: 103649.
[12] YONG W. Hybrid Max - Min ant system with four vertices and three lines inequality for traveling salesman problem[J]. Soft Computing, 2015, 19(3): 585-596.
[13] ZHANG H, YOU X. Multi-population ant colony optimization algorithm based on congestion factor and co-evolution mechanism [J]. IEEE Access, 2019, 7: 158160-158169.
[14] 卜冠南, 刘建华, 姜磊, 等. 一种自适应分组的蚁群算法[J]. 计算机工程与应用, 2021, 57(6): 67-73.
[15] FREY B J, DUECK D. Clustering by passing messages between data points[J]. Science, 2007, 315(5814): 972-976.
[16] KOOL W, VAN H H, WELLING M. Attention, learn to solve routing problems! [J]. arXiv preprint arXiv:1803.08475, 2018.
[17] VINYALS O, FORTUNATO M, JAITLY N. Pointer networks [J]. axXiv preprint arXiv:1506,03134, 2015. DOI: 10.48550/arXiv.1506.03134
[18] KANSO A, SMAOUI N. Logistic chaotic maps for binary numbers generations[J]. Chaos, Solitons & Fractals, 2009, 40(5): 2557-2568.
[19] STÜTZLE T, HOOS H H. MAX-MIN Ant System[J]. Future Generation Computer Systems, 2000, 16(8): 889-914.
[20] HUANG Y, SHEN X-N, YOU X. A discrete shuffled frog-leaping algorithm based on heuristic information for traveling salesman problem[J]. Applied Soft Computing, 2021, 102: 107085.