

连续域问题的 CSO 算法性能研究

程 乐, 朱旦晨, 王志勃, 钱兆楼, 潘永安

(淮安信息职业技术学院 计算机科学与工程系, 江苏 淮安 223003)

摘 要: 介绍了基本蟑螂算法(CSO)的算法思想, 并通过改进基本的 CSO 算法得到可应用于连续优化问题的 CSO 算法. 分析了改进后算法的性能, 实验证明: CSO 算法在种群规模较小的情况下, 可以快速收敛到一个高精度的解.

关键词: 蟑螂算法; 连续域; 种群规模

中图分类号: TP751

文献标识码: A

文章编号: 1000-7180(2011)02-0193-04

The Research in CSO of Continuous Optimization Problem

CHENG Le, ZHU Dan-chen, WANG Zhi-bo, QIAN Zhao-lou, PAN Yong-an

(Department of Computer Science and Engineering, Huaian College of Information Technology, Huaian 223003, China)

Abstract: This article introduces algorithm thinking of cockroach swarm optimization (CSO). An improved CSO was presented and applied to continuous optimization problem. This paper analyses of performances of the improved CSO. The experiment results show that CSO can get a relative fast convergence solutions in small population size.

Key words: cockroach swarm optimization (CSO); continuous optimization problem; population size

1 引言

目前, 在优化领域中一类群智算法越来越受到人们的重视^[1-3]. 蟑螂算法(Cockroach Swarm Optimization, CSO)^[4]就是一种较新的群智能优化算法, 该算法主要通过模拟蟑螂群居、依靠嗅觉觅食等生物行为完成寻优. CSO 算法第一次被提出主要用于解决 TSP 问题. CSO 算法主要利用了启发信息, 其指导思想是“在当前较优解附近完成更优解的查找”. 但是这样的指导思想很容易使算法陷入局部最优, 为解决该问题文献[5]中提出了一种引入大变异策略的蟑螂算法.

文献[4]和文献[5]证明了 CSO 算法可以应用于求解 TSP 等组合优化问题, 但其求解的 TSP 问题城市规模较小, 求解速度较慢. 而 CSO 算法的启发式寻优策略更适合求连续域优化问题. 因此, 文中对 CSO 算法进行了改进, 得到了一种可应用于连续域问题的 CSO 算法.

2 应用于 TSP 问题的 CSO 算法

为了便于描述, 下面以符号 C 代表 CSO 算法中的蟑螂.

CSO 算法可通过如下公式描述^[4]:

$$Fp - C = \text{Road} \quad (1)$$

$$Fg - C = \text{Road} \quad (2)$$

$$Fg + \text{Rand}() \text{Step} = Fp \quad (3)$$

其中, Fg 为最好食物位置, 即算法当前找到的最优解; Fp 为每只 C 自身记录的食物位置; Road 为 C 爬向 Fp 和 Fg 的路径; Step 为 C 爬行步长. CSO 算法为每一个 C 分配一个随机解, 作为 C 爬行的初始位置(巢穴), 而且该初始位置在算法执行过程中不更新. 式(1)表示 C 从初始位置爬向 Fp; 式(2)表示 C 从初始位置爬向 Fg; 式(3)表示由 Fg 生成 Fp.

文献[5]中将 CSO 算法应用于 TSP 问题. 如 TSP 问题中两个解的坐标分别为: A(4, 2, 1, 6, 5, 3) 和 D(1, 2, 3, 4, 5, 6).

则算法中 C 从 A 解爬到 D 解的爬行行为可通

过图 1 表示.

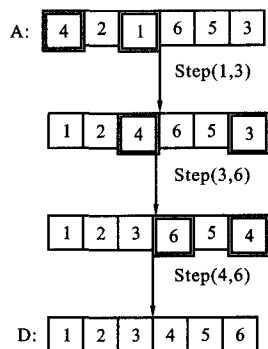


图 1 TSP 问题中 C 的爬行行为

图 1 中 Step(1,3)表示 A 中排序为 1 的城市和排序为 3 的城市位置交换,同时也表示 C 向 D 爬行一步. C 通过 Step(1,3)、Step(3,6) 和 Step(4,6) 三步完成从 A 解到 D 解的爬行. 每一步爬行,CSO 算法都将对当前所得解进行评价. 得到到更优的解则更新 Fg.

为了在当前较优解附近完成查找,基本的 CSO 算法中还运用了“食物再分配策略”(式(3)),其目的是生成与当前较优解(Fg)相近的解(Fp),用于吸引 C 爬行,完成较优解附近查找更优解的算法策略.

3 应用于连续域问题的 CSO 算法

根据连续域问题的特点,对基本的 CSO 算法稍加改进,便可得到应用于连续域问题的 CSO 算法. 文中主要从 CSO 的 C 爬行方式、食物再分配策略、回巢策略几个方面进行改进.

3.1 蟑螂爬行方式的改进

在 D 维的连续域问题中,CSO 中的 Fg、C、Fp 均可表示为向量 $x = (x_1, x_2, \dots, x_D)$. 其中向量分量 $x_i (i = 1, 2, \dots, D)$ 为 D 维函数中欲寻优的变量.

假设算法中有 m 个 C. 爬行时 $C_j (c_{j1}, c_{j2}, \dots, c_{jD}) (j = 1, 2, \dots, m)$ 的每个分量每次前进一个 Step(根据情况设定值,本实验 Step = 0.002). C 向 F 或者 Fg 爬行的一步可表示为向量 $\eta = (\eta_1, \eta_2, \dots, \eta_D)$. 分量 $\eta_i (i = 1, 2, \dots, D)$ 的取值可以是一 Step、0 和 Step. 以 C 向 $Fp_i (f_{i1}, f_{i2}, \dots, f_{iD}) (i = 1, 2, \dots, k)$ 爬行为例, η_i 的取值情况描述如下:

(1) 初始 $c_i = f_i$, 则 $\eta_i = 0$;

(2) 初始 $c_i < f_i$, 本次爬行后 $c_i < f_i$, 则 $\eta_i = \text{Step}$;

(3) 初始 $c_i < f_i$, 本次爬行后 $c_i > f_i$, 则 $\eta_i = 0$;

(4) 初始 $c_i > f_i$, 本次爬行后 $c_i > f_i$, 则 $\eta_i = -\text{Step}$;

(5) 初始 $c_i > f_i$, 本次爬行后 $c_i < f_i$, 则 $\eta_i = 0$.

由此可见,当 η 的分量 η_i 全部为零时,个体 C 向 F 爬行结束. 当所有 F 搜索完成后,群 C 将向 Fg (T 的坐标为 Ts) 爬行一次. 根据以上描述, C 向 F 或者 Fg 爬行的公式为:

$$C_{i+1} = C_i + \eta \quad (4)$$

3.2 食物分配策略的改进

每次迭代开始前,都 CSO 由 Fg 随机生成一定数量的 Fp 到预寻优问题的解空间定义域内. 若接下来的搜索过程中,找到了优于当前 Fg 的更优解,则立即更新 Fg.

在连续域问题中,假设 Fg 一次生成 k 个 Fp. 则每个 $Fp_i (f_{i1}, f_{i2}, \dots, f_{iD}) (i = 1, 2, \dots, k)$ 是由 Fg ($fg_1, fg_2, \dots, fg_D$) 与向量 $\delta (\delta_1, \delta_2, \dots, \delta_D)$ 相加得到. δ 的分量生成如下:

$$\delta_i = \text{Rand} \times \text{Step} \quad (5)$$

其中 Rand 为一个随机整数. Step 为步长,其具体数值可调. Fg 生成 Fp 如下:

$$Fp = Fg + \delta \quad (6)$$

为兼顾连续域问题中的全局搜索和局部搜索,在 Rand 的控制下 Fg 有两种生成食物方法:近端和全局. 两种生成 Fp 的 Rand 的取值范围如下:

(1) 近端: $\text{Rand} \in [-(\text{Max} - \text{Min})/(\tau \times \text{Step}), (\text{Max} - \text{Min})/(\tau \times \text{Step})]$

(2) 全局: $\text{Rand} \in [\text{Min}/\text{Step}, \text{Max}/\text{Step}]$

其中 τ 为域划分值,表示近端生成 Fp 时, Fp 距离 Fg 的最远距离为 τ 个步长距离, Max 和 Min 表示预求解问题某一维上取值的最大、最小值.

CSO 可以通过调整参与近端和全局生成的 Fp 的数量,调节算法的全局和局部搜索能力.

3.3 回巢策略的改进

基本 CSO 的初始解不进化. 即每个 C 每次都从同一个解(位置坐标)爬向 Fp 或者 Fg. 此方法限定寻优过程中 C 的巢穴位置为固定,不符合蟑螂巢穴不固定的生物特性.

在连续域问题的 CSO 算法中,采取每次 C 爬行开始前,重新随机初始化一个解作为 C 巢穴的方法. 此方法不仅符合自然界 C 的生物特性,同时增加了算法搜索空间的范围,提高了初始解的多样性,增加了算法收敛到全局最优解的概率.

3.4 算法步骤

以全局优化中最小化问题为例,描述算法流程如下:

Step1: 初始化 C 群: 定义种群规模 m, 初始化

$C_i(i=1,2,\dots,m)$ 的初始位置坐标(巢穴).

初始化 F_g : 选出 C 群中适应值最高的 C^* , 令 $F_g = C^*$.

初始化 k : F_g 一次生成 F_p 的个数.

初始化 M : 算法的总迭代次数.

Step2: F_g 生成 F_p (式(5)、式(6)).

Step3: C 向所有 F_p 爬行, 找到更优解更新 F_g (式(4)).

Step4: C 向当前相对优的解 F_g 爬行, 找到更优解更新 F_g (式(4)).

Step5: 所有的 C 完成巢穴迁移.

Step6: 满足算法结束条件 (F_g 达到最优, 或者算法总迭代次数达到 M), 则算法结束. 否则转 Step2.

4 仿真实验

文中选取几个复杂的 Benchmark 函数, 完成对改进的 CSO 算法性能测试 (见表 1). 实验设备为一般 PC 机, CPU 为 Intel(R) Pentium(R) Dual CPU 1.6GHz, 1014MB Memory, 实验仿真软件 VC++ 6.0.

表 1 测试函数

Symbol	Interval of x_i	Function(F)	Extreme value
f_1	$[-10, 10]$	$\sum_{i=1}^5 [i \cos((i+1)x_1 + i)] \sum_{j=1}^5 [j \cos((j+1)x_2 + j)] + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2$	-176.1375
f_2	$[-2.048, 2.048]$	$\sum_{i=1}^D [100 \times (x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	0
f_3	$[-32.768, 32.768]$	$20 + e - 20 \times \exp\left[-0.2 \sqrt{\frac{1}{D} \sum_{i=1}^D x_i^2}\right] - \exp\left[\frac{1}{D} \sum_{i=1}^D \cos(2\pi x_i)\right]$	0
f_4	$[-5.12, 5.12]$	$\sum_{i=1}^D [x_i^2 - 10 \times \cos(2\pi x_i) + 10]$	0

4.1 CSO 收敛性和跳出局部最优能力测试

表 2 中展示了针对表 1 的 Benchmark 函数测试结果. 实验参数: C 的数量为 $m=10$; 步长 $Step=0.002$; 每次迭代投放 F_p 数量为 8. 实验中迭代限 20 次. 表 2 对表 1 中每个函数测试 10 次, 求解精度: $1e$

-10 , 表 2 展示数据包括求得最优解的成功率 (G%); 平均、最优和最差解 (Value); 平均、最好和最差的评价次数 (Evals); 平均、最快和最慢的收敛时间 (Time), 单位为秒 (s).

表 2 Benchmark 函数测试结果

Function	G%	Value	Evals	Time(s)
		Eve/Best/Worst	Eve/Min/Max	Eve/Min/Max
f_1	100%	-183.5202/-186.0369/-179.0264	192/67/353	0.417/0.332/0.633
f_2	100%	1.9344e-27/2.0875e-28/6.0594e-27	162/55/232	0.513/0.242/0.712
f_3	100%	2.8899e-14/2.4280e-14/7.0466e-14	97/73/221	0.436/0.369/0.623
f_4	100%	0.0000e+00/0.0000e+00/0.0000e+00	95/74/182	0.394/0.298/0.593

从表 2 数据分析可知, 算法在较小的评价次数内能够稳定的收敛到精度较高的解.

4.2 CSO 与 AFS 对比实验

人工鱼群算法 (Artificial Fish-Swarm, AFS) 作为一种较新的群智能算法具有较快的收敛速度, 当问题要求解精度不高时, 用它可以快速的得到一个可行解^[2].

通过式(7)的测试函数完成 CSO 算法与 AFS 算法的比较试验.

$$\max f(x, y) = \frac{\sin(x)}{x} \times \frac{\sin(y)}{y} (x \in [-10, 10], y \in [-10, 10]) \quad (7)$$

式(7)函数的解空间分布如图 2 所示.

可以看出, 该非线性函数在全局极大值的周围密布着许多局部极值, 通常的寻优算法极易陷入局部极值或在各局部极值间震荡, 比较适用于验证算法的性能.

实验中采用两种算法每种运行 5 次, 限迭代 200 次, 取 5 次中收敛最快的一次, 生成收敛曲线.

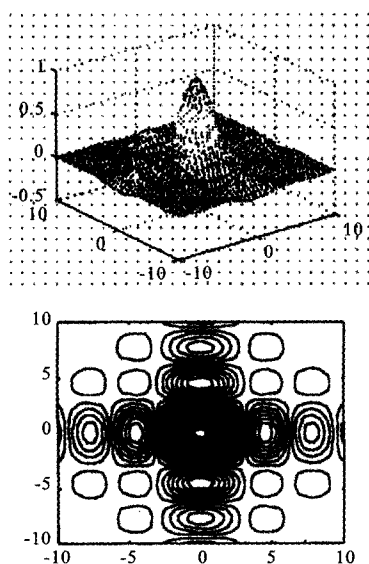


图2 式(7)函数的解空间分布

横坐标为迭代次数,纵坐标为每次迭代算法搜索到的最优解。

AFS算法参数设置参考文献[2],两种算法具体参数设置如下:

(1) AFS:感知距离为 25,移动步长为 0.3,拥挤度因子为 0.168. 种群规模为 10.

(2) CSO:种群规模为 10,步长 $Step=0.002$,每次迭代投放 Fp 数量为 8. 种群规模为 50.

从图 3 中容易比较 CSO 与 AFS 的收敛速度. CSO 在种群规模为 10 的情况下收所需要的迭代次数仍远低于种群规模为 50 的 AFS. 展示了 CSO 收敛速度.

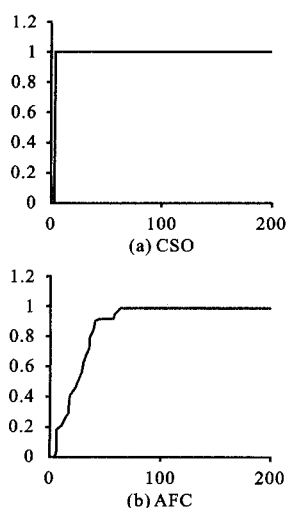


图3 CSO 与 AFS 收敛图

表 3 记录了本组五次实验的相关数据,数据包括每次实验的初始 Fg 坐标、初始 Fg 值以及前八次

迭代得到的最优解。

表3 图3(a)中的实验数据

一	二	三	四	五
(-9.822,10)	(-9.978,-10)	(-9.984,-10)	(-9.964,-10)	(-9.944,-10)
0.0021427	0.00286476	0.00289080	0.0028034	0.002714
0.99724238	0.99992467	0.99985268	0.99919959	0.99971936
0.99753576	0.99997000	0.99994600	0.99964737	0.99980668
0.99770544	0.99998067	0.99996733	0.99973069	0.99982335
0.99781458	0.99998867	0.99998867	0.99979668	0.99985268
0.99797231	0.99999400	0.99999867	0.99985267	0.99986667
0.99812607	0.99999733	1.00000000	0.99988734	0.99989200
0.99822460	0.99999933	1.00000000	0.99990934	0.99991334
0.99836773	1.00000000	1.00000000	0.99993467	0.99992467

分析表 3 的数据可知,初始 Fg 坐标所对应的初始解一般在 0.002 数量级. 经过一次迭代 Fg 对应的解除实验一为 0.99724238 以外,其余实验都在 0.999 以上数量级. 因此图 3 中 CSO 显示的收敛曲线平直,无抖动。

5 结束语

文中提出 CSO 算法更适合求解连续域优化问题. 从蟑螂爬行方式、食物再分配策略、回巢策略等几个方面对 CSO 算法做了改进. 通过实验证明,改进后的 CSO 算法性能优越,且对算法的种群规模依赖较小。

参考文献:

- [1] 徐文龙,孙俊,须文波. 基于干扰因子的 QPSO 算法改进[J]. 微电子学与计算机,2008,25(4):129-130.
- [2] 李晓磊,邵之江,钱积新. 一种基于动物自治体的寻优模式:鱼群算法[J]. 系统工程理论与实践,2002,22(11):33-38.
- [3] Eusuffm M, Lansey K E. Optimization of water distribution network design using shuffled frog leaping algorithm[J]. Journal of Water Resources Planning and Management,2003,129(3):210-225.
- [4] 程乐. 新的仿生算法:蟑螂算法[J]. 计算机工程与应用,2008,44(34):44-46.
- [5] 程乐. 引入大变异策略的蟑螂算法研究[J]. 微电子学与计算机,2009,26(5):13-20.

作者简介:

程 乐 男,(1979-),双硕士,讲师. 研究方向为人工智能、智能控制、数据挖掘、机器学习等.

朱旦晨 男,(1973-),硕士,讲师,工程师. 研究方向为人工智能.