

一种改进的变步长的萤火虫算法

李 恒¹, 郭 星^{1,2}, 李 炜^{1,2}

(1 安徽大学 计算智能与信号处理重点实验室, 安徽 合肥 230039;

2 安徽大学 计算机科学与技术学院, 安徽 合肥 230601)

摘 要: 为了克服人工萤火虫算法(Glowworm swarm optimization, GSO)在求解全局优化问题时,存在收敛速度慢,求解精度不高、易陷入局部最优等问题,提出了一种改进的变步长自适应的萤火虫优化算法。采用该算法可在一定程度上避免因过早的成熟陷入局部最优,并且可在后期达到更高的精确度。实验仿真证明,该算法较明显的提高了其收敛速度和精确度。

关键词: 萤火虫算法;全局优化;变步长;函数优化

中图分类号: TP301.6

文献标识码: A

文章编号: 1000-7180(2016)10-0041-05

An Improved Glowworm Swarm Optimization Algorithm with Changing Step

LI Heng¹, GUO Xing^{1,2}, LI Wei^{1,2}

(1 Key Laboratory of ICSP, Ministry of Education, Anhui University, Hefei 230039, China;

2 School of Computer Science and Technology, Anhui University, Hefei 230601, China)

Abstract: In order to overcome the basic artificial firefly algorithm (Glowworm swarm optimization, GSO) in solving global optimization problems, slow convergence speed, precision is not high, easily falling into the master problem. So, The paper puts forward an improved variable step optimization algorithm. To some extent, adopt the algorithm can avoid premature mature into a local optimum, and can achieve a higher accuracy in the later. Through simulation experiment, proves that the algorithm on the convergence speed and precision are obviously improved.

Key words: GSO; global optimization; changing step; function optimization

1 引言

萤火虫优化算法^[1] (Glowworm swarm optimization, GSO). 是基于种群的在多个位置并行随机搜索的策略,根据萤火虫的个体的感知范围来决定其搜索路径及其范围。通过不断的搜索迭代计算逐渐使主群进化到各个局部最优和全局最优。

目前,该算法已经在函数优化最优解上得到了成功的应用,也被广泛用于装配序列规划方法^[2], PFSP 问题^[3], 零等待流水线调度优化^[4] 和 T-S 模型辨识^[5] 等。该算法捕捉效率较高,寻找最优值速度极快,并且通用性较强等特点。但也具有不少缺陷,

如前期收敛速度慢、易陷于局部最优解、易出现不稳定震荡等现象^[6]。针对此问题,提出了一种改进的变步长的人工萤火虫算法(CSGSO, Change the Step of GSO)。

2 基本萤火虫算法

2.1 基本 GSO 算法描述

在基本 GSO 算法中,首先随机的在目标函数的解集空间中初始化一群萤火虫,每只萤火虫的荧光素都被初始化为同一个值,荧光素值与萤火虫的亮度是正比例关系,而荧光素的值又和寻优运动过程中萤火虫所处的空间位置相关联,并且荧光素值与

其领域范围内的吸引力也呈正比例关系. 其领域范围又称作决策域范围, 其大小被称为 r_d . r_d 值在 r_s 和 0 之间变动, r_s 是其最大值, 也是初始值, 称为萤火虫的可视范围. 单只萤火虫会以一定的概率向其 r_d 范围内的邻居萤火虫移动. 通过算法的不断迭代, 萤火虫的不断移动, 结果会存在较多的萤火虫分布在具有较高的荧光素(适应度值)的萤火虫周围. 从而达到智能优化的目的.

2.2 萤火虫算法 GSO 的主要步骤

GSO 算法, 概括起来, 主要分为四个阶段^[7]: 萤火虫初始化阶段、荧光素更新变化阶段、萤火虫位置更新阶段以及决策域范围更新阶段.

(1) 萤火虫初始化阶段

对算法的各个萤火虫的位置及各个参数进行初始化. 每只萤火虫的初始荧光素和感知半径都是 l_0 和 r_0 .

(2) 荧光素更新变化阶段

萤火虫的荧光素大小与其所在的搜索空间的位置有直接关系, 其所在空间位置的适应度值越高, 则更新后的荧光素就越大, 因为荧光强度与其荧光素成正比关系, 所以发光强度也就越强. 另外, 萤火虫的发光强度还与其上一次迭代的荧光素的大小以及发散速度的快慢有关系, 萤火虫的位置更新完成后, 下一次算法迭代开始前, 所有的萤火虫的荧光素都会更新, 其更新依据的公式(1):

$$l_i(t+1) = (1-\rho)l_i(t) + \gamma J_i(t+1) \quad (1)$$

式中, $\rho(0 < \rho < 1)$ 是萤火虫的荧光素挥发速度参数, $l_i(t)$ 是算法在第 t 次迭代时萤火虫 i 的荧光素大小, γ 是萤火虫的更新率参数, $J_i(t+1)$ 是算法在 $t+1$ 迭代时荧光虫 i 的自适应值.

(3) 萤火虫位置更新阶段

萤火虫算法每次迭代后, 萤火虫会在搜索空间中移动自己的位置. 萤火虫 i 移动位置之前必须找到一个符合条件的萤火虫 j . 这个萤火虫 j 是萤火虫 i 的邻居. 萤火虫 j 称为萤火虫 i 的邻居前提条件包含以下两项: 一是萤火虫 j 的荧光素必须大于萤火虫 i ; 二是萤火虫 j 要在萤火虫 i 的决策域范围内; 二者缺一不可. 然后在全部满足以上条件的萤火虫邻居集合中, 根据式(2) 计算每个邻居萤火虫的概率:

$$p_{ij} = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)} \quad (2)$$

式(2) 中 $j \in N_i(t)$, $N_i(t) = \{j: d_{i,j}(t) < r_d^i(t); l_i(t) < l_j(t)\}$, $N_i(t)$ 是在第 t 此迭代中萤火

虫 i 的领域集合, $d_{i,j}(t)$ 是在第 t 次迭代中萤火虫 i 与萤火虫 j 的空间欧式距离. 根据轮盘赌法则, 选出邻居萤火虫 j , 然后根据式(3) 更新其空间位置:

$$x_i(t+1) = x_i(t) + s \left(\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|} \right) \quad (3)$$

式(3) 中: $\|X_j(t) - X_i(t)\|$ 是萤火虫 j 与萤火虫 i 的欧式空间距离. S 是萤火虫的移动步长.

(4) 萤火虫决策域范围更新阶段

萤火虫位置移动更新后, 会动态的更新其自身的决策域范围, 更新的大小是基于决策域范围内邻居的数量多少. 具体根据式(4) 进行调整更新:

$$r_d^i(t+1) = \min\{r_s, \max\{0, r_d^i(t) + \beta(n_i - |N_i(t)|)\}\} \quad (4)$$

式(4) 中 β 是调整萤火虫动态决策范围的常量参数, n_i 是萤火虫邻域集合大小的阈值.

3 本文算法

3.1 分析萤火虫算法缺陷

GSO 算法, 萤火虫移动时, 是采用固定的步长移动, 步长取值是一个关键. 若固定步长取较小值, 会使算法过早成熟, 陷入局部极值. 若固定步长取较大值, 则可能导致后期出现震荡或者跳离最优解等问题^[6,8]. 综上可知, 固定步长取值难以达到理想的效果. 因此, 本文算法根据不同阶段的具体需要, 动态的调整步长, 用来较好的协调全局搜索和局部搜索之间的平衡, 使算法的效率有较大的提高.

3.2 本文改进的变步长萤火虫优化算法

根据上述的基本萤火虫算法存在的不足, 本文采用式(5) 和式(6) 来动态的改变萤火虫的移动步长, 在算法迭代初期, 步长保持较大值, 避免算法过早的成熟, 陷入局部最优. 算法的迭代后期, 曲线性的使步长减小, 利于算法在后期快速收敛, 而不至于因为步长太大而跳过最优解, 从而获得更精确的解.

$$s(t) = s_{\max}(p)^c \quad (5)$$

$$c = 1 + \frac{t}{t_{\max}} - \cos\left(\frac{t}{t_{\max}}\right) \quad (6)$$

上述两个公式中, t 为当前的迭代次数, $t_{\max}$ 为算法的最大迭代次数, $t_{\max}$ 为算法中最大的步长, 取值为 1, p 为定值, 本文取值为 0.08. 图 1 分别给出了当 $p = 0.06$ 、 $p = 0.08$ 、 $p = 0.1$ 时对测试函数的测试情况:

由图 1 可以看出, 当 p 值小的时候, 容易陷入局部最优, 但当 p 值较大时, 并不能保证会提高算法的搜索精度. 从图中可以看出, 其中精度最高并且迭代

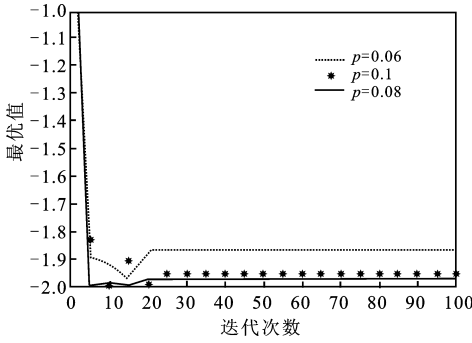


图1 p 取值对比情况

效率最高的是 $p = 0.08$, 所以本文选取 p 值为 0.08.

测试函数:

$$f_5(x) = x_1^2 + x_2^2 - \cos 18x_1 - \cos 18x_2$$

$$x_i \in (-10, 10), i = 1$$

3.3 改进变步长的 GSO 算法的步骤

综上所述,改进变步长的算法的流程图如图 2. 其算法步骤如下(本文的算法称为 CSGSO).

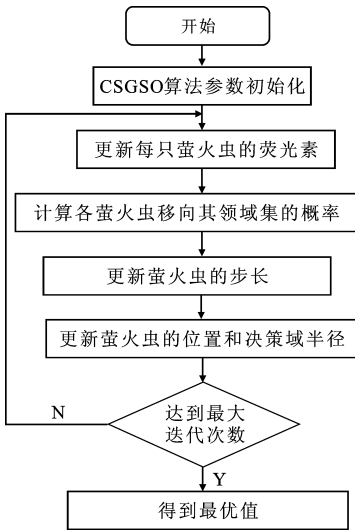


图2 改进变步长的算法的流程图

步骤 1 初始化问题的函数维数 m , 萤火虫群总数 n , 并随机的在问题的解空间中初始化萤火虫群, 初始化最大移动步长 $s_{\max}$, 最大的迭代次数 $t_{\max}$, 并且初始化 $\rho, \gamma, \beta, n_i, s, l_0, r_0, p$ 等参数.

步骤 2 对萤火虫群中的所有萤火虫按式(1)更新其荧光素值.

步骤 3 开始进入 CSGSO 算法迭代寻优.

步骤 4 计算每只萤火虫 i 的邻居萤火虫集合 $N_i(t)$.

步骤 5 根据式(2)计算萤火虫 i 的领域内的萤火虫移动概率, 用轮盘赌方法选择移动方向的萤火虫 j 属于 $N_i(t)$, 并且用式(3)进行位置的更新.

步骤 6 根据式(4)对萤火虫 i 的决策域半径进行更新.

步骤 7 判断当前迭代次数 t 是否大于最大值 $t_{\max}$, 如果小于, 则转向步骤 2, 如果大于则结束, 退出算法迭代, 得到最大值.

4 仿真实验分析

4.1 测试函数和算法参数设置

为验证 CSGSO 算法的有效性, 本文采用六个标准函数^[9-11] 对算法进行测试, 函数包含了高维和低维, 所有的函数都是求最小值问题.

(1) Sphere

$$f_1(x) = \sum_i^n x_i^2$$

$$x_i \in [-10, 10], i = 1, \dots, 10$$

该函数的全局最小值为 0;

(2)Becker and Lago

$$\min f_2(x) = (|x_1| - 5)^2 + (|x_2| + 5)^2$$

$$x_i \in [-10, 10], i = 1, 2$$

该函数的全局最小值为 25;

(3)Schaffer F6

$$\min f_3(x) = 0.5 + \frac{\sin^2 \sqrt{x_1^2 + x_2^2} - 0.5}{[1.0 + 0.001(x_2^2 + x_2^2)]^2}$$

$$x_i \in [-10, 10], i = 1, 2$$

该函数的全局最小值是 0

(4)BR-Branin

$$\min f_4(x) = (x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6)^2 + 10(1 - \frac{1}{8\pi}) \cos x_1 + 10$$

$$x_1 \in [-10, 10], x_2 \in [-10, 10]$$

该函数的全局最小值为 0.398

(5)RA-Rastrigin

$$\min f_5(x) = x_1^2 + x_2^2 - \cos 18x_1 - \cos 18x_2$$

$$x_i \in [-10, 10], i = 1, 2$$

该函数的全局最小值为 -2

(6)Rosenbrock

$$f_6(x) = 100(x_2 - x_1^2)^2 - (x_1 - 1)^2$$

$$x_i \in [-5, 5], i = 1, 2$$

该函数的全局最小值为 0;

4.2 测试平台

硬件配置: Intel(R) Core(TM) i3-3240 CPU @ 3.40GHZ, 内存 8 GB; 操作系统为 Windows 7, 仿真环境为 Matlab 2010b.

4.3 参数设置

GSO 算法和改进的变步长 GSO 算法参数基本相同,区别在于: GSO 算法采用的是固定的步长 $s=0.03^{[12]}$,而改进的变步长算法采用的步长是动态变化的,其最大值 $S_{\max}=1$. GSO 和本文的改进算法 CSGSO 的萤火虫群数 $n=100$,迭代最大值 $t_{\max}=100$. 算法的参数设置如表 1.

表 1 算法参数表						
算法	ρ	γ	N_t	l_0	r_s	$r_d(0)$
GSO	0.4	0.6	6	5	2.048	2.048
CSGSO	0.4	0.6	6	5	2.048	2.048

4.4 实验仿真结果分析

本文对函数 $f(1)\sim f(6)$ 分别独立运行 20 次,取其最大值、最小值和平均值,并计算了其标准偏差.

表 2 本文算法 CSGSO 与基本算法 GSO 性能对比

测试函数	算法	最优值	最差值	平均值	标准偏差
$f_1(x)$	GSOCSGSO	0.092 587 369 300 378	0.402 108 208 309 763	0.198 400 453 221 490	0.101 478 846 023 481
		0.042 835 881 301 151	0.146 298 257 041 541	0.087 833 712 549 783	0.037 601 130 366 615
$f_2(x)$	GSOCSGSO	25.012 500 665 642 417	25.824 525 312 306 619	25.396 878 874 057 229	0.6424 437 091 761 35
		25.000 016 751 291 483	25.090 718 761 299 5	25.004 747 417 213 899	0.0041 827 440 638 73
$f_3(x)$	GSOCSGSO	3.276 628 730 286 557e-004	0.097 159 098 775 57	0.030 891 906 047 7	0.002 690 985 626 271
		5.825 737 690 035 027e-005	0.009 715 910 019 954	0.008 100 600 364 569 8	0.002 602 701 127 650
$f_4(x)$	GSOCSGSO	0.397 892 583 872 517	0.413 797 392 373 521	0.400 478 008 123 630	0.005 112 199 564 405
		0.398 007 432 906 843	0.397 887 517 065 987	0.397 888 257 568 971	6.151 656 274 884 348e-007
$f_5(x)$	GSOCSGSO	-1.994 766 694 109 697	-1.657 385 298 219 274	-1.798 6857408 531 06	0.220 269 867 261 620
		-1.999 327 632 522 751	-1.753 153 993 897 536	-1.869 754 4492 062 43	0.111 993 367 259 544
$f_6(x)$	GSOCSGSO	3.244 082 250 007 794e-005	0.121 880 309 927 046	0.025 903 784 895 426	0.032 986 452 608 884
		4.867 775 308 528 369e-006	0.020 783 832 850 119	0.003 952 802 086 519	0.006 862 542 521 361

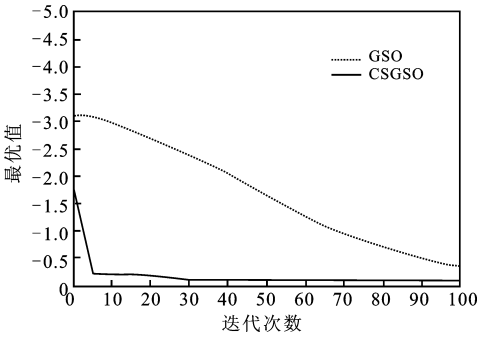


图 3 优化 $f_1(x)$ 的算法比较

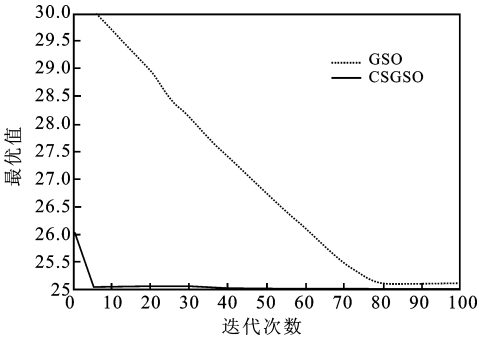


图 4 优化 $f_2(x)$ 函数的算法比

5 结束语

本文针对萤火虫算法 GSO 存在的不足,提出了一种新的变步长人工萤火虫 CSGSO 群优化算法,该步长根据算法的迭代次数的增加,呈规律性的递

与 GSO 算法进行对比. 数据中的平均值反应了算法的优劣,其标准偏差反应了算法的稳定性. 其结果如表 2 所示.

从表 2 可以看出改进的算法的平均值,要比 GSO 更优,其标准偏差要比其 GSO 更小. 所以改进算法的精度比 GSO 更高,改进算法的稳定性要比 GSO 更稳定.

对算法的收敛性情况,可以从求解目标函数的算法收敛图上分析. 从目标函数的算法收敛图如图 3~图 8 可以很直观的看出 CSGSO 算法的收敛速度很明显的比 GSO 算法的收敛速度快. 主要原因是 GSO 算法采用的是固定的步长,而本文采用的是动态自适应步长,在算法前期,采用较大的步长,提高了算法的搜索能力,在后期,采用较小的步长提高了算法的精度.

减. 使算法在迭代前期具有较高的全局搜索能力,后期快速收敛. 仿真实验证明,改进的萤火虫算法性能明显优于基本的萤火虫算法. 但对于高维函数的求解,效率不突出. 所以如何提高高维函数的寻优性能,是今后的研究方向.

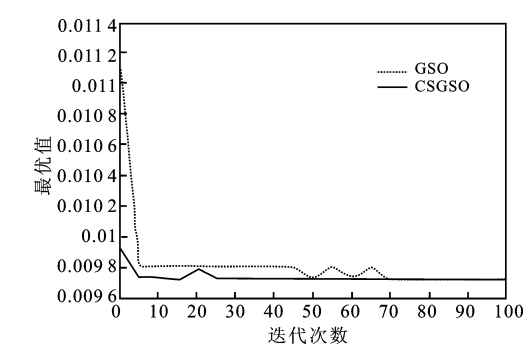


图 5 优化 $f_3(x)$ 函数的算法比

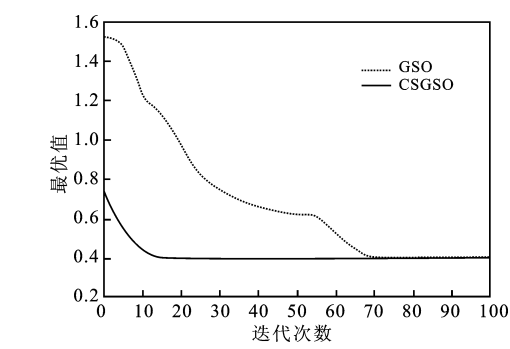


图 6 优化 $f_4(x)$ 的算法比较

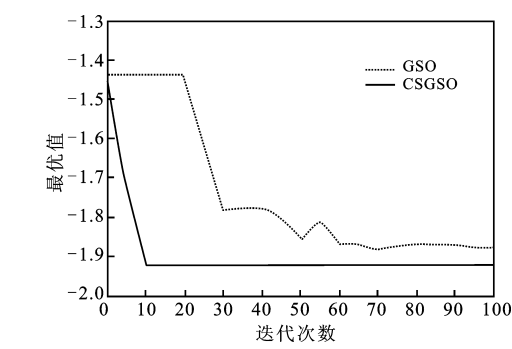


图 7 优化 $f_5(x)$ 函数的算法比

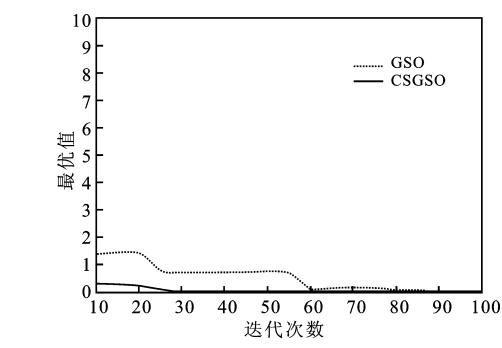


图 8 优化 $f_6(x)$ 函数的算法比

参考文献:

[1] Krishnanand K N, Ghose D. Detection of multiple source locations using a glowworm metaphor with applications to collective robotic[C]// IEEE Swarm Intelligence Symposium. Pasadena, California; IEEE Press, 2005; 84-91.

[2] 曾冰,李明富,张翼. 基于改进萤火虫算法的装配序列规划方法[J]. 计算机集成制造系统, 2014, 49(11): 799-806.

[3] 周季华,叶春明. 应用萤火虫算法求解 PFSP 问题[J] 计算机与信息技术, 2012, 19(3): 1-3.

[4] 李永林, 叶春明. 基于萤火虫算法的零等待流水线调度优化[J]. 机械设计与研究, 2013, 29(6): 50-54.

[5] 吴东周,丁学明. 基于改进萤火虫算法的 T-S 模型辨识[J]. 计算机仿真, 2013, 30(3): 327-330.

[6] 黄凯,周永权. 一种改进的变步长自适应 GSO 算法[J]. 计算机工程, 2014, 38(4): 185-187.

[7] 唐少虎,刘小明. 一种改进的自适应步长的人工萤火虫算法[J]. 智能系统学报, 2015, 10(3): 220-224/ 470-475.

[8] 黄正新,周永权. 自适应步长萤火虫种群多模态函数优化算法[J]. 计算机科学, 2011, 38(7): 220-224.

[9] Deep K, Bansal J C. Mean particle swarm optimization for function optimization[J]. International Journal of Computational Intelligence Studies, 2009, 1(1): 72-92.

[10] 贺春华,张湘伟,吕文阁. 竞选算法的参数设计与性能研究[J]. 计算机工程, 2010, 36(6): 201-203

[11] Simakov D, Caspi Y, Shechtman E, et al. Summarizing visual data using bidirectional similarity[C] // Proc. of CVPR'08. Anchorage, Alaska, USA: [s. n.], 2008.

[12] Krishnanand K N, Ghose D. Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications[J]. Multiagent and Grid Systems, 2006, 2(3): 209-222.

作者简介:

李 恒 男, (1992-), 硕士研究生. 研究方向为科研成果计算与信号处理. E-mail: 13063213016@163.com.

郭 星 男, (1983-), 博士, 讲师. 研究方向为模式识别与信号处理.

李 炜 男, (1969-), 教授. 研究方向为模式识别、智能计算与信号处理.