文章编号:1671 - 6833(2015)01 - 0020 - 04

一种求解连续空间约束优化问题的蚁群算法

焦留成, 邵创创, 程志平

(郑州大学 电气工程学院,河南 郑州 450001)

摘 要: 借鉴蚁群算法和惩罚函数的思想提出了一种用于求解连续空间约束优化问题的蚁群算法. 应用自适应调整惩罚因子的惩罚函数法将约束优化问题转化为无约束优化问题,再结合自适应调整全局选择因子和信息素挥发系数的连续域蚁群算法,求解连续空间约束优化问题. 通过对基准测试函数进行编程求解,对比采用固定参数的蚁群算法求解结果,验证了所提改进算法的正确性和有效性.

关键词: 连续空间;约束优化;蚁群算法;惩罚函数

中图分类号: TP18 **文献标志码**: A doi:10.3969/j.issn.1671 - 6833.2015.01.005

0 引言

蚁群算法是一种采用分布式控制,通过群体搜索策略和群体之间的信息交换寻找最优路径的优化方法[1]. 并已在旅行商问题、二次分配问题和调度问题等一系列离散优化问题上获得了成功应用[2]. 该算法的寻优过程不依赖于优化问题本身的严格数学性质,对目标函数无解析性要求[3]. 因此,可以将本质上离散的蚁群算法应用于求解连续空间的约束优化问题,近年来也取得了一些成果[4]. 但这些研究大多是针对无约束或者约束条件仅为自变量取值范围的优化问题,而对约束条件一般的等式约束和不等式约束的研究很少. 笔者借鉴蚁群算法和惩罚函数的思想设计了一种可以求解约束优化问题的连续域蚁群算法. 并通过对基准测试函数的编程求解,验证了算法能以很高的概率寻找到最优解,误差率很低.

1 约束优化问题

约束优化问题一般可以表达如下.

$$\min f(\boldsymbol{X}). \tag{1}$$

$$\text{s.t.}\begin{cases} g_i(x)\leqslant 0, i=1,2,\cdots,q; \\ h_j(x)=0, j=q+1,q+2,\cdots,m; \\ l_t\leqslant \boldsymbol{X}_t\leqslant u_t, t=1,2,\cdots,n. \end{cases} \tag{2}$$

式中:$\boldsymbol{X}=(X_1,X_2,\cdots,X_n)\in \mathbf{R}^n$ 为 n 维向量;f 为待优化函数;g_i,h_j 为约束函数. 约束优化问题的本质是寻找到既满足约束条件又能使目标函数值达到最小的变量 X^* 及其所对应的函数值 $f(X^*)$.

2 连续域蚁群算法

(1)初始化. 依据具体问题设定蚂蚁数目 m、最大迭代次数 K,m 只蚂蚁随机分布在函数定义域内,并将各自位置作为其寻优的起点. 设各自变量 $\boldsymbol{X}(X_1,X_2,\cdots X_n)$ 的取值范围为 $[a,b]=\{[a_1,b_1],[a_2,b_2],\cdots,[a_n,b_n]\}$. 第 i 只蚂蚁的初始位置为 $X(i)=(X_1^i,X_2^i,\cdots,X_j^i,\cdots,X_n^i)$,其中:

$$X_j^i=a_j+(b_j-a_j)\cdot \text{rand}(1). \tag{3}$$

设定蚂蚁 i 所处位置对应的信息素强度为

$$T_0(i)=\exp(-f(X_i)). \tag{4}$$

(2)全局搜索. 所有蚂蚁在完成一次搜索后,将会有一只蚂蚁搜索到本次循环的最优解,然后其它蚂蚁在各自的下一轮搜索中便会以一定的概率和步长转向该处蚂蚁. 其转移概率和步长分别为

$$P(i,b)=\begin{cases}\dfrac{\exp(r)}{\exp(T(b))}, i\neq b; \\ 0, \text{else}.\end{cases} \tag{5}$$

$$X_i=\begin{cases} X_i+\lambda(X_b-X_i), P(i,b)<P_0; \\ X_i+\text{rand}(-1,1)\cdot L(i), \text{else}.\end{cases} \tag{6}$$

收稿日期:2014 - 09 - 18;**修订日期**:2014 - 11 - 20

基金项目:国家自然科学基金资助项目(61075071);河南省教育厅自然科学基金资助项目(14A413008);郑州市科技局资助项目(131PPTGG409 - 5)

通信作者:程志平(1974 -),男,河南商水人,郑州大学副教授,博士,主要从事直线电机建模与控制研究,E-mail:zpcheng@zzu.edu.cn.

$$P_0=\begin{cases}0.9\cdot\exp((k/K)\cdot 2\cdot\ln(1/2)), k\leqslant(K/2);\\ 0.225\cdot\exp((k/K)\cdot 2\cdot\ln(2)), \text{else}.\end{cases} \tag{7}$$

式中:$r=T(b)-T(i)$;$\lambda=1/k$ 为全局转移步长参数;k 表示当前迭代次数;P_0 为全局选择因子,将其设为迭代次数 k 的函数,迭代初期 P_0 取较大值,那么非最优解蚂蚁 $X(i)(i=1,2,\cdots,m,i\neq b)$ 向最优解蚂蚁 $X(b)$ 转移时,就会以较大概率选择步长 $X_i+\lambda(X_b-X_i)$,可加快算法寻优,迭代中期 P_0 取较小值,蚂蚁会以较大概率选择步长 $X_i+\text{rand}(-1,1)\cdot L(i)$,会增强步长选择的随机性,进而能扩大搜索的解空间,迭代后期为加快搜索速度,P_0 还取较大值.

(3)局部搜索. 上一轮循环中找到最优解的蚂蚁继续在一个小邻域内进行局部搜索,以期望能找到更优解. 假设蚂蚁新找到的解 $X(u)$ 其信息素强度为 $T(u)$,比较新解和原来解的信息素强度,若新解比原来解的信息素强度大,则用新解位置取代原来解位置;否则,仍保留原来最优解. 表达式如下.

$$X(b)=\begin{cases}X(u), T(u)>T(b);\\ X(b), \text{else}.\end{cases} \tag{8}$$

$$X(u)=\begin{cases}X(b)+w\cdot\text{step}, \text{rand}(1)<0.5;\\ X(b)-w\cdot\text{step}, \text{else}.\end{cases} \tag{9}$$

$$w=w_{\max}-(w_{\max}-w_{\min})\cdot\frac{k}{K}. \tag{10}$$

式中:*step* 为局部搜索的初始步长;w 为局部搜索步长,将其取为迭代次数的函数,随迭代次数增加而减小,这样在迭代后期步长就会变小以便能搜索到更精确解. 式中 $w_{\max}$ 和 $w_{\min}$ 分别为步长更新参数上下限,其值一般取范围为(1,1.4)和(0.2,0.8)的常数[5].

(4)信息素更新. 蚂蚁完成一次全局搜索和局部搜索后,将对其信息素进行更新,更新规则如下.

$$T(i)=(1-\rho)T(i)+\Delta T(i). \tag{11}$$

$$\rho=0.1\cdot\exp((k/K)\cdot\ln(9)). \tag{12}$$

式中:$\Delta T(i)=\exp(-f(X_i))$;$\rho$ 为信息素挥发系数,将 ρ 设定为迭代次数的函数,迭代早期,ρ 取较小值可以提高搜索的随机性和全局搜索能力,随着搜索的进行,蚁群会越来越向最优解蚂蚁聚集,这时需要减少搜索的随机性,加快收敛速度,那么,增大 ρ 的取值,增强信息的正反馈作用可以达到此目的.

3　惩罚函数法处理约束条件

3.1　经典惩罚函数法

常用的惩罚函数法表达形式如下.

$$p_i(X)=\sum_{i=1}^{L}g_i(X). \tag{13}$$

$$g_i(X)=(h_i(X))^2, i=1,2,\cdots,p. \tag{14}$$

$$g_{i+p}(X)=\begin{cases}0, & g_i(X)\leqslant 0;\\ (g_i(X))^2, & g_i(X)>0.\end{cases} \quad i=1,2,\cdots,m. \tag{15}$$

$$F(X,M_k)=f(X)+M_k\sum_{i=1}^{l}g_i(X). \tag{16}$$

其中,式(14)、(15)分别为等式和不等式约束条件,式(16)表示新构造的适应度函数,$l=p+m$,$M_k>0$ 为惩罚因子. 当约束条件被破坏时,至少会有一个 $i(1\leqslant i\leqslant l)$,使 $g_i(X)>0$,从而 $p(X)>0$. 且约束条件被破坏得越厉害,$p(X)$ 取值就会越大,在 M 取为一定值时,$F(X,M)=f(X)+Mp(X)$ 也就越大,惩罚也就越厉害,体现了对于约束条件被破坏时的惩罚. 反之,当 x 满足约束条件时,则 $g_i(X)=0$ $(i=1,2,\cdots,l)$,$p(X)=0$,这时不管 M 取值多大,都有 $F(X,M)=f(X)$,即惩罚函数对原目标函数的求解无影响.

3.2　自适应惩罚函数法

惩罚函数法因其方法简单,易于编程实现而得到广泛应用. 但也存在着一个主要缺陷:惩罚因子对收敛速度影响较大[6]. 若选取过小,则对不可行解的惩罚力度不够,算法不易进入可行域甚至可能收敛到不可行解;若选取过大,则惩罚力度过大,会降低算法的搜索效率. 为了解决该问题,这里将惩罚因子设置为自变量的函数. 构造惩罚函数如下.

$$p(X)=\sum_{i=1}^{m}S_i(X)|h_i(X)|+\sum_{j=1}^{p}M_j(X)\max\{0,g_j(X)\}. \tag{17}$$

$$S_i(X)=\frac{|h_i(X)|}{\sum_{i=1}^{p}|h_i(X)|+\sum_{j=1}^{m}\max\{0,g_j(X)\}}, \quad i=1,2,\cdots,p. \tag{18}$$

$$M_j(X)=\frac{\max\{0,g_j(X)\}}{\sum_{i=1}^{p}|h_i(X)|+\sum_{j=1}^{m}\max\{0,g_j(X)\}}, \quad j=1,2,\cdots,m. \tag{19}$$

构造适应度函数如下.

$$G(X)=f(X)+C(X)p(X). \tag{20}$$

$$C(X)=1+\frac{|f(X)|}{1+p(X)}. \tag{21}$$

惩罚因子 $S_i(X)$ 和 $M_j(X)$ 随着约束函数 $h_i(X)$ 和 $g_j(X)$ 的变化而变化,即惩罚因子会随着算法迭代的进行做自适应调整. 这能实现在违反约束程度大的段给予较大的惩罚,对违反程度小的段给予较小的惩罚,进而能提高算法的求解效率.

4 算法求解步骤

结合上文连续域蚁群算法和自适应惩罚函数法,求解连续空间约束优化问题的具体步骤如下.

(1)根据具体问题确定最大循环次数 K、蚂蚁数目 m、收敛精度 ε、步长更新参数上下限 $w_{\max}$ 和 $w_{\min}$、局部搜索时的初始步长 $step$、各变量取值范围,按式(3)初始化蚁群位置.

(2)根据式(17)~(21)将约束优化问题转化为无约束优化问题,并构造适应度函数.

(3)根据式(4)计算各蚂蚁所在位置对应的信息素强度、适应度值,找出信息素强度最大的蚂蚁 $X(b)$.

(4)根据式(5)~(7)各蚂蚁进行全局搜索,更新各个蚂蚁位置.

(5)最优解蚂蚁按式(8)~(10)进行局部搜索,更新当前最优解蚂蚁位置.

(6)所有蚂蚁在完成本次循环后按式(11)、(12)进行信息素更新,并保存当前最优解.

(7)如果所求的最优解满足要求或已经达到最大迭代次数,则算法迭代结束,否则转向步骤 2 继续迭代运算.

5 仿真实验及数据结果分析

笔者选取如下标准测试函数利用 MATLAB 软件进行算法的仿真实验.

$$f(x)=100(x_2-x_1^2)^2+(1-x_1)^2. \tag{22}$$

约束条件为 $x_1+x_2^2\geqslant 0$, $x_1^2+x_2\geqslant 0$, $x_1\in[-0.5, 0.5]$ 和 $x_2\leqslant 1.0$. 已知最优解为 $x^*=(0.5, 0.25)$,此时有 $f(x^*)=0.25$.

由于该函数为开口向上的曲面,如果直接求解该函数的最小值,则蚂蚁都会分布在曲面底部,被曲面遮盖,故笔者采用求解 $-f(x_1,x_2)$ 最大值的方式,这样蚂蚁的最初和最终位置分布都会覆盖在曲面上,对比更明晰. 将采用固定参数蚁群算法对比改进后自适应调整参数值的蚁群算法用 MATLAB 软件分别编程求解. 采用同样的蚂蚁个数 $m=100$,最大迭代次数 $K=1\ 000$. 改进前固定参数 $p_0=0.2$, $\rho=0.8$, $m=100$,改进后自适应调整参数的步长更新参数上下限分别取 $w_{\max}=1.2$ 和 $w_{\min}=0.6$,局部搜索的初始步长 $step=0.1\cdot\mathrm{rand}(1)$. 分别运行该算法程序 30 次,求得蚂蚁最初分布位置、最终分布位置、最优函数值随迭代次数的变化趋势如图 1~3 所示. 分别记录每次迭代时间和求得的解,取其中最优解、获得最优解次数、最差解、获得最差解次数和计算的平均解、平均时间以及误差率制作表 1.

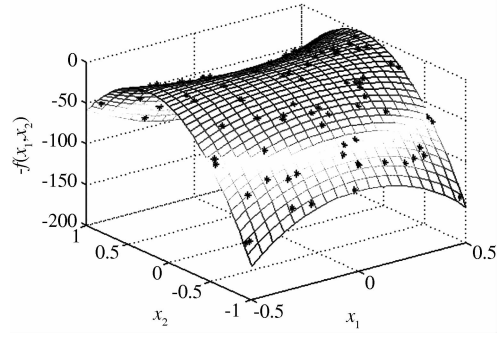

图 1 蚂蚁的最初分布位置

Fig. 1 Initial location of ants

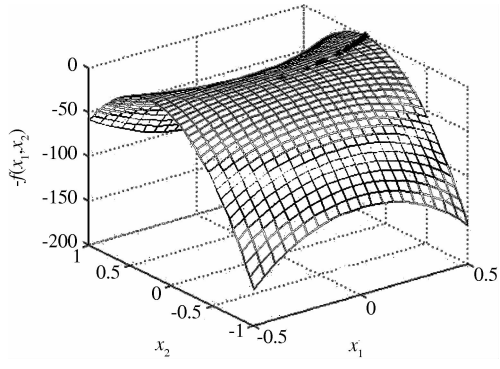

图 2 蚂蚁的最终分布位置

Fig. 2 Final location of ants

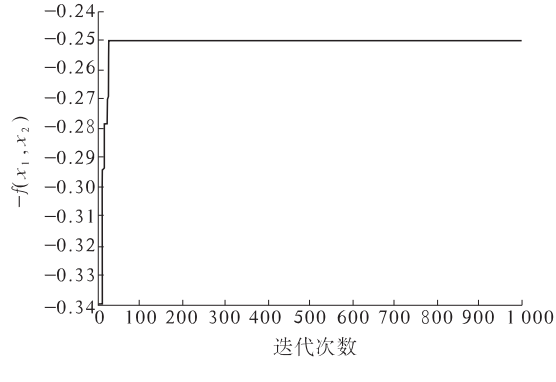

图 3 最优函数值随迭代次数变化趋势

Fig. 3 The optimal function values of chang trends with the iteration

表 1　蚁群算法优化结果

Tab. 1　Ant colony algorithm results

项目	改进前	改进后
最优解	$-f(0.5,0.25)=-0.250\ 0$	$-f(0.5,0.25)=-0.250\ 0$
最优解的个数	10	25
最差解	$-f(0.485\ 8,0.269\ 2)=-0.374\ 6$	$-f(0.498\ 6,0.251\ 2)=-0.251\ 4$
最差解的个数	1	1
平均解	$-f=-0.289\ 6$	$-f=-0.250\ 6$
平均时间	7.367 2	5.096 1
平均误差率	3.27%	0.24%

由上表可知,笔者设计的蚁群算法在求解基准测试函数所求各结果明显优于改进前采用固定参数的蚁群算法. 蚂蚁最初比较分散地分布在解空间内,程序运行结束后会比较明显地聚集在最优点附近,对比明晰. 改进后算法求得的平均解 $-f=-0.250\ 6$,即求得的满足约束条件的函数最小值为 $f=0.250\ 6$,平均误差率很低.

6　结论

在借鉴前人研究蚁群算法求解优化问题的基础上,设计了一种用于求解连续空间约束优化问题的蚁群算法,并给出了算法的较详细步骤. 在 MATLAB 软件上分别编程采用固定参数的蚁群算法和采用自适应调整参数的蚁群算法求解一个基准测试函数,对比求得的结果,验证了改进后的算法具有更强的寻优能力.

参考文献:

[1]　原思聪,刘道华,江祥奎,等. 基于蚁群算法的多维有约束函数优化研究[J]. 计算机应用研究,2008,25(6):1682 - 1684.

[2]　刘喜恩. 用于连续空间寻优的一种蚁群算法[J]. 计算机应用,2009,29(10):2744 - 2747.

[3]　李朝辉,梁昔明. 连续域蚁群算法的改进研究及在参数估计中的应用[D]. 长沙:中南大学信息科学与工程学院,2011.

[4]　SOCHA K, DORIGO M. Ant colony optimization for continuous domains [J]. European Journal of Operational Research (S0377 - 2217), 2008, 185(3): 1155 - 1173.

[5]　贾延臣,刘长良. 基于连续空间优化问题的蚁群算法及其应用研究[D]. 保定:华北电力大学控制与计算机工程学院,2008.

[6]　雷韵平,成良玉. 一种改进的蚁群算法及其在电机优化设计中的应用研究[D]. 广州:中山大学信息科学与技术学院,2010.

Ant Colony Algorithm for Solving Continuous Space Constrained Optimization Problems

JIAO Liu-cheng, SHAO Chuang-chuang, CHENG Zhi-ping

(School of Electrical Engineering, Zhengzhou University, Zhengzhou 450001, China)

Abstract: With ideas of ant colony algorithm and penalty function, an ant colony algorithm, which can solve continuous space constrained optimization problems, was proposed. We adopted the penalty function method of adjusting its value of adaptively to transform the constrained optimization problems into unconstrained optimization problems, and then combined with the continuous domain ant colony algorithm of adjusting its global selection factor and the value of the pheromone evaporation factor adaptively to solve the continuous space constrained optimization problems. And through programming solution of one benchmarking function, we compared the results with those of using fixed parameters ant colony algorithm, it was verified with correctness and effectiveness.

Key words: continuous space; constrained optimization; ant colony algorithm; penalty function method