优化的动态帧时隙 ALOHA 防碰撞算法*

郭来功，黄友锐，蔡　俊
（安徽理工大学 电气与信息工程学院，安徽 淮南 232001）

摘　要：对动态帧时隙 ALOHA 防碰撞算法中的标签个数估计进行改进，采用动态调整机制，使标签个数估计式系数自动调整，解决阅读器下一个查询周期应该使用的帧时隙长度的问题。针对实测中当帧长度与标签个数相等时冲突率较高的问题，将查询命令时间计入总的帧长度中以优化帧长度。仿真结果表明，优化后的 DFSA 算法延迟时间减少，冲突率降低，系统效率提高，在标签数量较少时效果尤为明显。

关键词：射频识别；动态帧时隙；防碰撞；冲突率；系统效率
中图分类号：TN911.23；TP301.6　　**文献标志码**：A　　**文章编号**：1001-3695(2012)11-4141-03
doi：10.3969/j.issn.1001-3695.2012.11.036

Optimized dynamic framed slotted ALOHA anti-collision algorithm

GUO Lai-gong, HUANG You-rui, CAI Jun
(*College of Electronically & Information Engineering*, *Anhui University of Science & Technology*, *Huainan Anhui* 232001, *China*)

Abstract: To improve the estimation of the number of tags in the dynamic framed slotted ALOHA anti-collision algorithm, this paper used dynamic adjustment mechanism to automatically adjust the formula coefficient of estimation of the number of tags for the problem of the length of the frame slot which the reader should be used in the next polling cycle. For the problem of higher collision rate when the frame length equal to the number of tags, it optimized the frame length by including the query command time in the total frame time. The simulation results show that the optimized DFSA algorithm reduces total delay time and the collision rate, improves system efficiency, especially in the number of tags are less.

Key words: RFID; dynamic framed slotted; anti-collision; collision rate; system efficiency

0　引言

射频识别(radio frequency identification，RFID)技术作为物联网(IOT)的关键技术之一，广泛地应用于物流、制造、管理等各种信息化系统中。RFID 包括阅读器和标签两大部分，在标签密集的情况下，常常存在标签冲突现象，阅读器必须采用防碰撞算法区别各个标签，从而顺利地读取标签内容。因此，防碰撞算法在多标签应用领域具有十分重要的地位。

无源标签在目前的应用环境下主要采用时分多路访问(TDMA)的方法，通常分为两大类：a)基于二进制树的确定性算法，其基本思想就是根据标签的 ID 号对标签进行分组，逐步缩小标签的范围直到最终完成对标签的阅读，然后再进行新的分组直到标签数量为 0；b)以 ALOHA 算法为代表的随机性算法，其基本思想是发现碰撞后，阅读器令所有标签延迟一段时间再进行查询。为了提高 ALOHA 算法的吞吐率和响应速度，通常将传输时间划分成若干时隙，规定若干个时隙为一帧，时隙数根据标签数量的多少自动变化的就被称为动态帧时隙 ALOHA(dynamic framed slotted ALOHA，DFSA)算法。两种算法各有其优缺点，随机性算法便于实现，适应能力较强，但响应时间不确定；确定性算法有确定的响应时间，系统吞吐率高，但识别时间较长，在有标签加入或者移出时，重新开始查询，适应性较差。

1　DFSA 算法分析

时隙 ALOHA 算法是将传输时间分为若干个时隙，标签在每个时隙开始时发出响应，算法的吞吐率为 $S = Ge^{-G}$，G 为标签数量，当 $G = 1$ 时，S 达到最大值，$S_{max} = e^{-1} = 0.368$。将多个时隙定为一帧，对于固定的帧长度，当标签数小于帧长度时出现时隙浪费，当标签数大于帧长度时冲突过于频繁。为了使标签数等于帧长度，需要在每次查询时，阅读器动态地估计标签数量，调整帧长度，从而使得吞吐率最高。

DFSA 算法实现过程如图 1 所示。

	下行	上行	标签1	标签2	标签3	标签4	标签5	标签6	标签7	标签8
第一帧	请求									
	时隙1	冲突	UID1			UID4				
	时隙2	空闲								
	时隙3	冲突					UID5		UID7	UID8
	时隙4	UID2		UID2						
	时隙5	UID3			UID3					
	时隙6	UID6						UID6		
	时隙7	空闲								
	时隙8	空闲								
第二帧	请求									
	时隙1	UID1	UID1							
	时隙2	UID4				UID4				
	时隙3	冲突					UID5		UID7	
	时隙4	空闲								
	时隙5	UID8								UID8

图1　DFSA算法实现过程

收稿日期：2012-04-15；**修回日期**：2012-05-26　　**基金项目**：安徽省高校省级自然科学研究资助项目(KJ2011Z091)

作者简介：郭来功(1980-)，男，河南邓州人，讲师，硕士，主要研究方向为无线通信(lgguo@aust.edu.cn)；黄友锐(1971-)，男，教授，主要研究方向为信号处理；蔡俊(1974-)，男，副教授，主要研究方向为微电子设计.

假设在读取期间,阅读器的读取范围内没有新的标签加入,同时也没有未识别的标签移出此范围。假定有 8 个标签,则可设帧长度为 8。首先阅读器发送查询命令,采用 8 个时隙为一帧。在第一次读取时,标签 1、4 在第一个时隙同时发送响应,标签 5、7、8 在第三个时隙同时发送其响应;它们在同一个时隙内有多个标签回应,产生了碰撞。标签 2、3 和 6 分别在第四、第五和第六个时隙分别发送响应,它们能够成功地被阅读器识别;另外三个时隙为空闲时隙。除了标签 2、3 和 6 被识别,还有 5 个标签没有读取,这样在下一帧时,阅读器设定新的帧长度为 5 个时隙,重新发送帧,未识别的标签重新响应。依此类推,直到在每一个时隙没有碰撞发生,所有的标签被阅读器识别。

由图 1 可以看出,DFSA 算法的性能优劣取决于帧的长度,即没有被阅读器正确识别的标签的个数。该精确值是很难确定的,DFSA 算法正是估算出未识别标签的数量,根据估算值动态调整帧长度,使算法性能最优。

设帧长度为 N,标签数量为 n,所有标签均匀分布在各个时隙,则一个时隙内具有 x 个标签的概率服从二项分布:

$$B(x)=\binom{n}{x}(\frac{1}{N})^x(1-\frac{1}{N})^{n-x} \tag{1}$$

具有 x 个标签时隙数的期望值为

$$P_x^{N,n}=NB(x) \tag{2}$$

在每一帧期间,每一个时隙的响应情况只有三种可能:冲突、空闲和成功读取一个标签,每种情况的时隙个数的估计值为

空闲: $$P_i=NB(0)=N(1-\frac{1}{N})^n \tag{3}$$

成功: $$P_s=NB(1)=n(1-\frac{1}{N})^{n-1} \tag{4}$$

冲突: $$P_c=NB(k)=N-P_i-P_s \quad k>1 \tag{5}$$

得到空闲时隙数、成功时隙数和冲突时隙数后,就可以得到算法的冲突率、系统效率等指标。

对于 n 值的估计算法,文献[1]中提出的算法为 $n=P_s+2P_c$,文献[2]对此算法进行优化,得到的估计值为 $n=P_s+2.3922P_c$。

2 DFSA 算法优化

在当前的研究中,标签个数的估计结果为通过已经发送的帧得到的该查询周期的标签数量,这个数值是已发送帧应该使用的长度;实际使用中应该确定下一次发送帧所使用的长度。另外,在实际的使用当中,当 N 的取值等于 n 时,并没有像期望的那样得到最大的吞吐率,此时的标签冲突频繁[3],因此应对 DFSA 算法进行分析,优化帧长度以得到更好的应用。

2.1 标签个数估计

为了确定下一次发送帧的长度,已知本次阅读器查询到的冲突时隙数为 P_c,假设冲突的标签个数与冲突时隙数呈线性关系,采用一个可调整的系数与 P_c 相乘:

$$n_y(i+1)=cP_c(i) \tag{6}$$

而第 $i+1$ 次查询完成后,得到的标签估计值为

$$n(i+1)=2.3922P_c(i+1) \tag{7}$$

要使两者之差最小,即

$$cP_c(i)-2.3922P_c(i+1)=0 \tag{8}$$

可以得到

$$c=\frac{2.3922P_c(i+1)}{P_c(i)} \tag{9}$$

这里系数 c 与 $i+1$ 次得到的结果有关,而 $i+1$ 次结果是下一次阅读器查询后才能得到,所以将其调整为

$$c=\frac{2.3922P_c(i)}{P_c(i-1)} \tag{10}$$

可得冲突标签个数估计为

$$n=cP_c(i)=\frac{2.3922P_c(i)}{P_c(i-1)}\times P_c(i) \tag{11}$$

这样,已知本次的冲突时隙数和上次的冲突时隙数,就可以得到剩余标签的个数,以确定下一次查询应该使用的帧时隙数。这里 c 为一可变系数,可以根据本次阅读器得到的查询结果自动调整。

2.2 帧长度优化

在实际标签识别中,当帧长度 N 与标签个数 n 相等时,频繁发生标签冲突现象,这是由于在帧发射时,并未考虑阅读器发射查询命令的时间。系统吞吐率为一次查询能够成功读取的标签个数,设一次读取的帧长度为 T,吞吐率可以表示为

$$S=\frac{P_s}{T}=\frac{P_s}{P_i+P_s+P_c+T_r}=\frac{n(1-\frac{1}{N})^{n-1}}{N+T_r} \tag{12}$$

其中:T_r 为一帧内阅读器发射的查询命令时间。显然,吞吐率是 N 的函数,两边对 N 求导,令其等于 0,化简后可得

$$(n-1)(N+T_r)(N-1)^{-1}-(N+T_r)(n-1)N^{-1}-1=$$
$$N^2-nN-nT_r+T_r=0 \tag{13}$$

解得

$$N=(n+\sqrt{n^2+4nT_r-4T_r})/2 \tag{14}$$

显然,当忽略 T_r 时,$N=n$。然而实际的阅读器发射信号时由于 T_r 的存在,使得 N 的取值必须大于 n 时才能使得标签冲突减小到最低。表 1 给出了在查询命令为 8 倍时隙长度的情况下应该使用的帧长度。

表 1 优化后的帧长度与标签个数的关系

n	帧长度 N	n	帧长度 N
50	57	150	158
100	107	200	208

3 优化后算法比较

为了评价优化后的算法性能,将本算法与 DFSA 算法进行对比。假定在当前的仿真环境下阅读器能够识别出所有的标签,而且在识别的过程中没有新的标签进入识别范围。设初始时隙数为 32(1 帧),时隙长为 4.9 ms,查询命令为 8 倍的时隙长度(39.2 ms)。

3.1 冲突率比较

冲突率定义为[4]

$$\eta=\frac{\text{总冲突时隙数}}{\text{总时隙数}}=\frac{\sum P_c}{\sum N} \tag{15}$$

其中:P_c 为每帧读取时的冲突时隙数,N 为每帧的时隙数时间(包含查询命令时间)。

图 2 给出了随着标签数量的增加冲突率的变化情况。由图可以看出,在标签数量较少时,本算法的冲突率比 DFSA 算法冲突率低很多,随着标签数量的增加,本算法的冲突率上升,

而 DFSA 算法的冲突率上升较慢。这是因为本算法在标签较少时,帧长度比标签数量大得多,随着标签数量的增加,帧长度与标签数量的比值越来越小。

3.2　系统效率比较

系统效率定义为

$$\beta = \frac{\text{成功通信时隙数}}{\text{总时隙数}} = \frac{\sum P_s}{\sum N} \tag{16}$$

其中:P_s 为每帧读取时成功读取标签的时隙数。

图 3 给出了随着标签数量的增加系统效率的变化情况。由图可以看出,本算法与 DFSA 算法相比有较高的系统效率。与图 2 类似,随着标签的数量增加,本算法中帧长度与标签数量的比值减小,系统效率的优势减小。

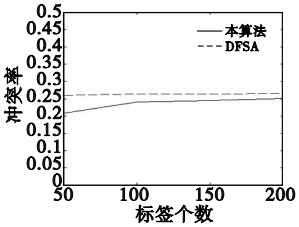

图2　本算法与DFSA算法冲突率的比较

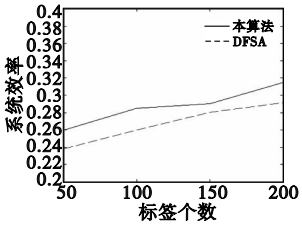

图3　本算法与DFSA算法系统效率的比较

3.3　延时时间

延时时间为在阅读器的识别范围内识别所有的标签所需要的时间。显然,随着标签数量的增加,所需时间也增加。图 4 给出了优化后的算法与 DFSA 算法所需时间的比较。虽然在标签数量较大时所需时间与优化前的算法相比优势不明显,但是在标签数量较少时,本算法所需时间明显减小。

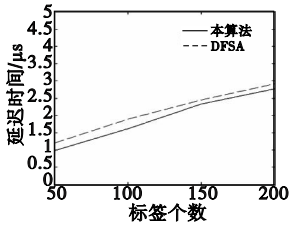

图4　本算法与DFSA算法延迟时间的比较

4　结束语

本文对射频识别中常用的 DFSA 防碰撞算法进行了分析,对标签个数的估计采用系数动态调整的方法,通过阅读器本次接收到的估计值预测下一次发送时隙数量。同时对当前 DFSA 算法常采用的帧长度等于标签数量时冲突率较高的现象进行分析,并根据标签数量调整帧长度。算法优化后的仿真结果表明本算法延迟时间减小,冲突率约有 10% 的降低,而系统效率有约 15% 的提高。

参考文献:

[1] VOGT H. Multiple object identification with passive RFID tags[C]// Proc of IEEE International Conference on Man and Cybemetics. 2002: 1-6.

[2] ZHEN Bin, KOBAYASHI M, SHIMIZU M. Framed ALOHA for multiple RFID objects identification[J]. IEICE Trans on Communications, 2005, E88-B(3): 991-999.

[3] 栗华. UHFRFID 多标签防碰撞算法的研究与性能分析[D]. 济南:山东大学, 2011.

[4] WANG Chun-yi, LEE C C, LEE M C. An enhanced dynamic framed slotted ALOHA anti-collision method for mobile RFID tag identification[J]. Journal of Convergence Information Technology, 2011, 6(4): 340-351.

[5] DENG D J, TSAO H W. Optimal dynamic framed slotted ALOHA based anti-collision algorithm for RFID systems[J]. Wireless Personal Communications, 2011, 59(1): 109-122.

[6] KNERR B, HOLZER M, ANGERER C, *et al*. Slot-wise maximum likelihood estimation of the tag population size in FSA protocols[J]. IEEE Trans on Communications, 2010, 58(2): 578-585.

[7] EOM J B, YIM S B, LEE T J. An efficient reader anti-collision algorithm in dense reader networks with mobile RFID readers[J]. IEEE Trans and Electron, 2009, 56(7): 2326-2336.

[8] 刘齐宏,李天德,周志斌,等. 基于射频识别系统 RFID 动态时隙算法的经济性研究[J]. 四川大学学报:工程科学版, 2009, 41(6): 183-186.

(上接第 4133 页)

4　结束语

本文研究了基于 GPSO 加速 BFA,不仅利用了 GPSO 算法的快速收敛能力,并保持了细菌觅食算法的全局搜索能力。测试结果显示,在解的精度和稳定性方面,GPSO-BFA 算法的整体性能比 BFA、PSO-BFO、BSO 和 BF-PSO 更优,且计算成本更低。细菌觅食算法与其他智能优化算法的结合还有多种方式,更多更细致的工作有待于进一步研究。

参考文献:

[1] PASSINO K M. Biomimicry of bacterial foraging for distributed optimization and control[J]. IEEE Control Systems Magazine, 2002, 22(3): 52-67.

[2] 周雅兰. 细菌觅食优化算法的研究与应用[J]. 计算机工程与应用, 2010, 46(20): 16-21.

[3] 李军军,吴燕翔,甘世红,等. 基于梯度 PSO 算法的 PID 参数整定[J]. 科学技术与工程, 2009, 9(9): 2463-2467.

[4] 肖健梅,李军军,王锡淮. 梯度微粒群优化算法及其收敛性分析[J]. 控制与决策, 2009, 24(4): 560-564.

[5] KENNEDY J, EBERHART R C. Particle swarm optimization[C]// Proc of IEEE International Conference on Neural Networks. Piscataway, NJ: IEEE Service Center, 1995: 1942-1948.

[6] BISWAS A, DASUPTA S, DAS S, *et al*. Synergy of PSO and bacterial foraging optimization: a comparative study on numerical benchmarks[C]//Proc of the 2nd International Symposium on Hybrid Artificial Intelligent Systems. Berlin: Springer-Verlag, 2007: 255-263.

[7] KORANI W M. Bacterial foraging oriented by particle swarm optimization strategy for PID tuning[C]//Proc of GECCO Conference Companion on Genetic and Evolutionary Computation. New York: ACM Press, 2008: 1823-1826.

[8] 刘小龙,李荣钧. 基于粒子群算法的细菌觅食全局优化算法[EB/OL]. (2010-04-28). http://www.paperedu.cn/paper_oo0b51.

[9] 胡海波,黄友锐. 混合粒子群算法优化分数阶 PID 控制参数研究[J]. 计算机应用, 2009, 29(9): 2483-2486.

[10] 储颖,糜华,纪震,等. 基于粒子群优化的快速细菌群游算法[J]. 数据采集与处理, 2010, 25(4): 442-447.

[11] 麦雄发,李玲. 混合 PSO 的快速细菌觅食算法[J]. 广西师范学院学报:自然科学版, 2010, 27(4): 91-94.

end

即对于式(5),w_1、w_3 取线性递减的正值,而 w_2 取线性递减的负值。

2.4　步长和向前移动的处理

BFA 中关键步骤是趋向操作,目前研究者对趋向性操作改进最多的是对步长单位 C 引入自适应机制[2],目的是在算法开始阶段有较强的开发能力,而在后期有较高的开采能力。而 GPSO-BFA 算法中通过线性递减的惯性权重,自然具有此特点。故在细菌的翻转动作中,GPSO-BFA 去除了步长 C 参数,这也避免了对 C 进行自适应控制所增加的复杂性。

另一方面,原始 BFA 中的趋向操作包括了翻转和向前移动两个步骤。而由前所述,GPSO-BFA 的翻转操作去除了步长 C,其翻转与粒子群中的粒子移动一样,步伐较大,故在 GPSO-BFA 中尝试去掉细菌的向前移动操作。初步实验表明,去掉或不去掉向前移动操作,对解的质量影响微小,但计算时间上却相差较大。

2.5　算法复杂度分析

若假设算法中变量维数为 p,细菌个数为 S,最大趋化代数为 Nc,最大前进代数为 Ns,最大繁殖代数为 Nre,最大驱散代数为 Ned,则 BFA 算法的算法复杂性为 $O(p \times S \times \mathrm{Nc} \times \mathrm{Ns} \times \mathrm{Nre} \times \mathrm{Ned})$,而 GPSO-BFA 算法的算法复杂性为 $O(p \times S \times \mathrm{Nc} \times \mathrm{Nre} \times \mathrm{Ned})$。

细菌觅食算法通过趋化、繁殖和驱散操作来更新细菌的位置和速度,具有很强的全局搜索能力。GPSO-BFA 算法将局部搜索能力强的 GPSO 算法与全局搜索能力强的 BFA 进行结合,以细菌的更新来带动粒子群的更新,更容易求解出全局最优值。

3　实验与分析

3.1　测试函数

为了考察 GPSO-BFA 算法的性能,选择 6 个常用的 Benchmark 函数进行数值实验,这些函数广泛应用于评价优化算法的性能。前 3 个是单峰函数,后 3 个是多峰函数。因为篇幅关系,函数的具体描述请参照文献[11]中的表 1。

3.2　参数设置

将 GPSO-BFA 与标准 BFA、PSO-BFO、BSO 和 BF-PSO 算法进行比较。为了保证算法的可比性,各相关算法取相同的参数,其中种群规模 $S=100$, 变量维数分别取 $p=50$ 和 $p=100$,最大驱散代数 Ned = 2,最大繁殖代数 Nre = 10,最大趋化代数 Nc = 10,最大前进步数 Ns = 4(GPSO-BFA 不需要此参数),驱散概率 Ped = 0.25;粒子群参数 $C_1=C_2=2$,w 从 0.9 线性递减到 0.4,ρ 取 0.1;每次实验重复进行 50 次。

3.3　实验结果及分析

评价的标准包括平均最优适应度和标准差。表 1 给出了五种算法对测试函数求解的结果对比。表 1 中统计结果表明,所对比的五种算法中,基本是 BFA 的性能最差,较好的是 PSO-BFO,其次是 BSO,然后是 BF-PSO,最好的是本文所建议的 GPSO-BFA 算法。不管是对于 50 维还是 100 维的问题,GPSO-BFA 在其中的 5 个函数中所得解的精度和标准差都优于其他算法,特别是在问题 f_5 和 f_6 中,50 次运算都找到了理论最优解,只是在 f_2 函数上稍差于 BF-PSO。因此,从表 1 中数据可以看出,不管单峰函数还是多峰函数,GPSO-BFA 在求解精度和稳定性方面,都明显优于其他三种方法,表明了本文提出的 GPSO-BFA 算法在解决高维函数问题上的可行性和有效性。

表 1　五种算法的平均最优适应度和标准差(平均值(标准差))

函数	维数	BFA	PSO-BFO	BSO	BF-PSO	GPSO-BFA
f_1	50	5.01E+01(7.82458)	7.06E-02(3.417E-03)	1.36E-01(0.082154)	1.26E-19(1.21E-19)	1.09E-30(2.1863E-30)
	100	3.30E+02(23.75462)	7.30E-02(0.003576)	5.91E-02(0.030129)	3.75E-17(9.32E-17)	2.96E-29(4.7233E-29)
f_2	50	2.36E+01(4.090789)	7.93E-03(0.004164)	6.38E-03(0.00295)	8.89E-05(7.9E-05)	1.17E-04(1.2484E-04)
	100	2.93E+02(43.01238)	1.39E-02(0.006872)	6.34E-02(0.039737)	1.08E-04(9.17E-05)	1.09E-04(8.3894E-05)
f_3	50	8.18E+01(2.214362)	2.18E+01(6.482329)	8.17E-02(0.029954)	7.53E-10(5.43E-10)	1.90E-13(1.3293E-13)
	100	9.01E+01(1.075389)	4.42E+01(3.634181)	1.48E-01(0.114484)	3.99E-09(1.96E-09)	1.14E-12(8.1039E-13)
f_4	50	5.05E+02(29.93833)	1.11E+01(0.69695)	6.78E+00(2.731808)	0(0)	0(0)
	100	1.20E+03(34.10532)	1.15E+01(0.593675)	1.94E+01(10.32486)	5.10E-16(5.02E-15)	0(0)
f_5	50	1.00E+03(84.74834)	6.75E+02(61.95324)	9.85E-01(0.460136)	4.88E-17(2.58E-16)	0(0)
	100	2.29E+03(113.2784)	1.76E+03(104.4716)	2.57E+00(1.257644)	1.52E-14(6.16E-14)	0(0)
f_6	50	2.01E+01(0.116055)	2.23E-01(0.011266)	1.28E+00(0.688637)	5.77E-10(3.84E-10)	4.44E-15(1.7581E-15)
	100	2.04E+01(0.079154)	1.61E-01(0.009737)	2.43E+00(0.750458)	6.08E-09(2.96E-09)	1.13E-14(5.5787E-15)

图 1~6 给出了 100 维下各函数的寻优进化曲线,图的纵轴取了平均最优适应度的对数值,横轴取繁殖代数。从图 1~6 可以看出,除了在 f_2 函数上取得与 BF-PSO 几乎相同的最优值外,在其余 5 个函数上,GPSO-BFA 获得了最好的最优值。其中 f_4 和 f_5 问题在 50 次运算中最差的一次也在第 12 代繁殖后到达最优值;在相同繁殖代数下,GPSO-BFA 下降最快,拥有最快的收敛速度。

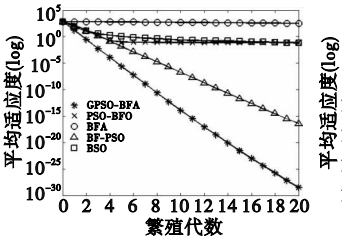

图1　f_1平均适应度的进化曲线

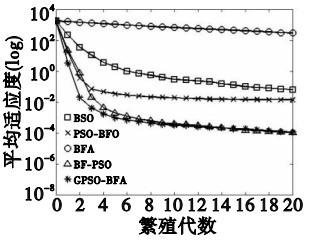

图2　f_2平均适应度的进化曲线

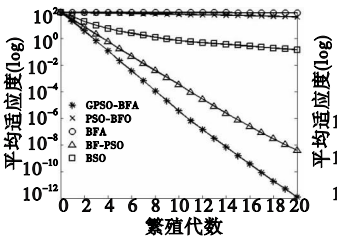

图3　f_3平均适应度的进化曲线

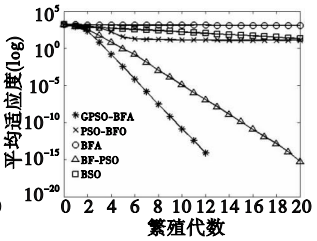

图4　f_4平均适应度的进化曲线

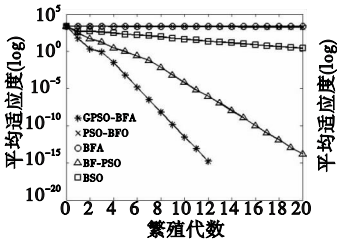

图5　f_5平均适应度的进化曲线

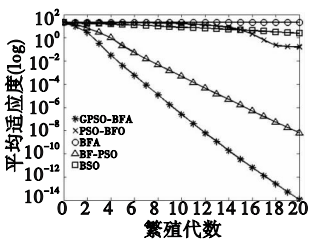

图6　f_6平均适应度的进化曲线

基于 50 维的运行时间对比如表 2 所示。从表中数据可以看出,因为 GPSO-BFA 省略了前进操作,速度比 BFA、PSO-BFO、BF-PSO 和 BSO 都要快很多。

表 2　五种算法的平均运行时间

函数	BFA	PSO-BFO	BSO	BF-PSO	GPSO-BFA
f_1	7.390 9	6.442 8	9.914 5	8.753 8	2.779 1
f_2	4.909 4	6.934 1	9.758 4	5.332 8	2.525 0
f_3	5.927 8	6.467 2	10.213 4	6.695 9	3.795 9
f_4	3.239 4	3.552 3	4.670 0	4.440 1	2.093 4
f_5	8.268 3	8.956 3	6.939 7	7.359 7	3.385 9
f_6	3.938 4	4.384 8	5.886 3	6.946 3	2.294 4

(下转第 4143 页)