文章编号:1001 - 9081(2006)04 - 0803 - 03

基于快速收敛遗传算法的 S 盒的优化算法

殷新春[1,2],杨 洁[1]
(1. 扬州大学 计算机科学与工程系,江苏 扬州 225009;
2. 南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093)
(yzgg_100@163.com)

摘 要:采用遗传算法来对 S 盒进行优化,并引入了启发式变异策略。实验表明,这种变异规则能够显著地提高算法的搜索效率,可以加快算法的收敛速度。此外,采用最佳个体保存法的选择策略可以减少额外的计算量。基于该方法,给出了 6×6 的 S 盒优化的完整程序描述,并获得了一批高非线性度和低差分均匀度的 S 盒。

关键词:S 盒;非线性度;差分均匀度;遗传算法
中图分类号: TP309.7 **文献标识码**:A

Optimum algorithm of S-boxes based on fast convergence speed genetic algorithm

YIN Xin-chun[1,2], YANG Jie[1]
(1. *Department of Computer Science and Engineering, Yangzhou University, Yangzhou Jiangsu* 225009, *China*;
2. *State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing Jiangsu* 210093, *China*)

Abstract: The optimization of S-boxes based on genetic algorithm was discussed, and the heuristic mutation strategy was used in this algorithm. Results of the experiments indicate that the mutation operation has high searching efficiency and fast convergence speed. Meanwhile, the selection strategy of preserving the best individuals can reduce the calculation time greatly. Based on the method, an effective genetic algorithm for 6×6 S-boxes was provided and a number of S-boxes with high nonlinearity and low difference uniformity were obtained.

Key words: S-boxes; nonlinearity; difference uniformity; genetic algorithm

0 引言

S 盒是许多密码算法中的唯一非线性部件,因此,它的密码强度决定了整个密码算法的安全强度。S 盒主要提供了分组密码算法所必须的混淆作用[1]。

一般地,可以将 S 盒的构造分成两大类:一类是用数学方法构造,另一类是随机生成构造。使用数学函数可以构造一些好的 S 盒,目前常用的此类 S 盒有:指数函数、对数函数、有限域 $GF(2^n)$ 上的逆映射以及有限域上的幂函数[2]。随机选取方法要求设计者有足够的时间和计算能力。本文提出了一种基于快速收敛遗传算法的双射 S 盒的优化方法。

1 遗传算法概述

遗传算法(Genetic Algorithm, GA)[3,4]是一种模拟生物种群进化的优化方法,它的思想来源于自然界中的进化过程。GA 本质上是一种群体迭代寻优过程,它的具体运算过程如下:①确定染色体编码方案和适应值计算方法;②随机生成一个初始群体;③计算群体中每个个体的目标函数和其他一些相关的函数,根据目标函数到适应值函数的映射关系计算出每个个体的适应值,并按适应值由大到小的顺序对个体进行排序;④采用某种选择方法选择群体中适应值较大的两个个体,并利用交叉和变异算子产生一个新个体,重复该过程直到产生一个新的解群;⑤如果迭代次数达到一定的值或其收敛指标已经满足,算法停止;否则,转③。图 1 给出了遗传算法的一般操作过程。

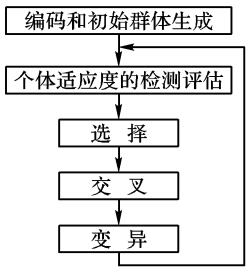

图 1 GA 求解流程

2 基于遗传算法的 S 盒的优化算法

2.1 S 盒的编码

对于任意 n 阶的双射 S 盒都可以看作是 0 到 $2n-1$ 的所有整数的一个全排列。因此,染色体的表达可以采用换位表达[3]。

例如:4 阶的双射 S 盒对应的输出依次为 2, 9, 13, 8, 15, 11, 7, 4, 12, 0, 14, 6, 10, 1, 3, 5,则 S 盒可以编码为如下所示的染色体,其中每一个数值表示染色体的一个基因:

[2 9 13 8 15 11 7 4 12 0 14 6 10 1 3 5]

2.2 产生初始种群

传统的遗传算法都是采用随机方法来生成初始种群,这就必然会增加额外的计算量。

我们将预先生成的一批 S 盒存放在某个文件夹内,执行遗传算法时,首先到指定的文件夹内随机选择 N 个 S 盒作为初始种群。这种方法在一定程度上减少了算法的运行时间。

2.3 适应值函数

自然界中,个体的适应值即是它繁殖的能力,它将直接关系到其后代的数量[5]。目前最有效的两种攻击方式是线性

收稿日期:2005 - 10 - 26;**修订日期**:2005 - 12 - 20 **基金项目**:国家自然科学基金 NSF 资助项目(60473012)
作者简介:殷新春(1962 -),男,江苏姜堰人,教授,博士,主要研究方向:并行与分布计算、信息安全; 杨洁(1981 -),女,江苏常州人,硕士研究生,主要研究方向:信息安全.

分析和差分分析,因此,在我们的适应值函数中将同时考虑 S 盒的非线性度和差分均匀度。

S 盒的非线性度和差分均匀度分别表示如下[1]:

$$N_s = \min_{l \in L_n, 0 \neq u \in F_2^m} d_H(u \cdot s(x), l(x))$$

$$\delta = \frac{1}{2^n} \max_{a \in F_2^n, a \neq 0} \max_{\beta \in F_2^m} |\{x \in F_2^n : S(x \oplus a) \oplus S(x) = \beta\}|$$

其中 L_n 表示全体 n 元线性和仿射函数之集。

本文将采用文献[6]中的适应值函数,其表示形式如下:

$$f(s) = a_s f_s(N_s) + a_d f_d(\delta)$$

其中,f_s,f_d 分别是 N_s 和 δ 的函数,a_s 和 a_d 是相应的系数。

本文将非线性度的函数定义为:$f_s(N_s) = N_s$;而差分均匀度的函数为:$f_s(\delta) = 2^n \times \delta$。

在确定 a_s 和 a_d 的值之前,非线性度和差分均匀度的不同组合应当根据 S 盒的密码学性能由高到低进行排列。排列的顺序不是唯一的,它主要依赖于使用 S 盒的算法的实际应用背景。

下面将给出 6×6 的双射 S 盒的适应值函数。首先,假设在某个特定的应用背景下,S 盒的性能由好到差的排列如下:(20,6),(20,8),(18,6),(18,8),(20,10),(18,10),(16,6)。适应值越大,组合越好,根据这一点我们将适应值函数定义如下:

$$F(x) = \begin{cases} N_s - \gamma & N_s < 18, \gamma > 18 \\ N_s - \frac{1}{2}\gamma & N_s < 18, \gamma \leqslant 8 \\ \frac{3}{2}N_s - \gamma & N_s \geqslant 18, \gamma > 8 \\ \frac{3}{2}N_s - \frac{1}{2}\gamma & N_s \geqslant 18, \gamma \leqslant 8 \end{cases}$$

表 1 显示了非线性度和差分均匀度的各种组合的适应值。从表 1 中可以看出适应值的次序与上文中组合的排列是一致的。

表 1 6×6 的 S 盒的适应值

(N_s, γ)	适应值
(20,6)	27
(20,8)	26
(18,6)	24
(18,8)	23
(20,10)	20
(18,10)	17
(16,6)	13

2.4 选择策略

算法的选择方法采用基于局部竞争机制的选择:$\mu + \lambda$ 选择,即是从 λ 个后代与 μ 个父体中选取 μ 个最优的后代。实验表明,这种最佳个体保存法的选择策略可以大大减少额外的计算量。

2.5 交叉算子

算法的交叉操作是:首先给当代群体中的所有个体分配杂交概率,满足概率 $< p_c$ 则选中,然后将所有选中的个体进行两两杂交操作。本文采用算法 1 对两个父体执行交叉操作。

算法 1:交叉算子

```
void crossover (int a, int b)
  { 随机选择 r, 0 ≤ r ≤ n;
    Child[r...r+l-1] = Parent[a][r...r+l-1];
    for(i=0,j=0; i<r; j++)
      if (Parent[b][j]没有在 Child 中出现过)
        { Child[i] = Parent[b][j];
          i++;
        }
    for(i=r+16; i<BOX_SIZE; j++)
      if (Parent[b][j]没有在 Child 中出现过)
        { Child[i] = Parent [b][j];
          i++;
        }
  }
```

2.6 变异规则

交叉完成后,新的子女个体并没有直接进行变异操作,而是首先根据 $\mu + \lambda$ 选择策略,从 N 个父体和所有子女个体中选出 N 个适应值最高的 S 盒作为后面进行变异的候选个体,然后以概率 1 将子女个体进行变异,其中 N 是群体的规模。这种方法既可以避免遗漏掉好的个体,又有利于加快算法的收敛速度。

变异算子具有补偿群体多样性损失的重要作用,一方面可以在当前解附近寻找更优解,另一方面可以保持群体的多样性,使群体能够继续进化。本文采用多次互换的变异技术,使得产生的子女个体有较大的顺序上的变化。为了防止优良的基因受到破坏,对于性质较好的染色体,只交换两次;为了保证群体中个体的多样性,对于性质较差的染色体,至少执行 5 次互换操作;为了体现遗传算法的随机性,用随机方法来产生互换的次数 m。实验表明,我们所采用的这种变异规则能够显著地提高算法的搜索效率,加快算法的收敛速度。变异操作的算法实现如下:

算法 2:变异算子

```
void mutation(int a)
  { 随机选择 r1, r2, 0≤r1≠r2≤n;
    n1 = abs(hwt(Child[a][r1]^ Child[a][r2-1]) - n/2);
    n2 = abs(hwt(Child[a][r2]^ Child[a][r2-1]) - n/2);
    m1 = abs(hwt(Child[a][r1]^ Child[a][r2+1]) - n/2);
    m2 = abs(hwt(Child[a][r2]^ Child[a][r2+1]) - n/2);
    if ((n1 <= n2) &&(m1 <= m2))
      Child[r1] ? Child[r2]
  }
```

2.7 算法的完整描述

上面讨论了算法的各个组成部分,下面给出针对6×6的 S 盒构造的完整程序描述:

算法 3:快速收敛遗传算法

```
CreateSbox_GA( )
  { /* Create initial population by random selection from a given
       folder; */
    opt = 0;
    while(opt < 24)
      { j = 0;
        /* generate crossover ratio randomly for each parent, select
           the ones with lower ratio than pc, and store them in an
           array temp_cross; */
        /* pare-wise crossover all the parents in the array temp_
           cross, the count of children is stored in a variable; */
        Getfit (k + N);
          /* k-the number of chidren; N-the number of parents */
        sortfit(k + N);
        /* select N better S-boxes according to the fitness value; */
        for(i = 0; i < N; i++)
          {                    /* generate random variable cur_p */
            if ((cur_p < pm) &&(fval[i] > =24))
              { mutation(i); mutation(i); }
            if ((cur_p < pm) &&(fval[i] <24))
              { r = rand() %7 +5;
                for(j = 0; j < r; j++)
                  mutation(i);
              }
          }                                          /* for i */
                    /* save the children as current population */
        getfit(N);
```

```
        /* opt ? the lowest fitness value of a child among the N
           children after crossover and Mutation; */
    }                                              /* while opt */
                                  /* output all the individuals; */
}                                                  /* CreateSbox_GA */
```

算法 4:不采用自适应变异规则的遗传算法

```
CreateSbox_GA( )
{ /* Create initial population by random selection from a given
     folder; */
  opt = 0;
  while( opt < 24)
    { …
      for( i = 0;  i < N; i ++ )
        {                    /* generate random variable cur_p */
          if ( ( cur_p < pm)
            { k = rand( ) % 10 + 2;
              for( j = 0; j < k; j ++ )
                mutation( i );
            }                                      /* if cur_p */
        }                                          /* for i */
      …
    }                                              /* while opt */
  /* output all the individuals; */
}                                                  /* CreateSbox_GA */
```

算法 5:采用部分最佳个体保存法的选择策略的遗传算法

```
CreateSbox_GA( )
{ /* Create initial population by random selection from a given
     folder; */
  opt = 0;
  while( opt < 24)
    { …     /* copy N′ better S-boxes to the next generation; */
      /* select ( N - N′) S-boxes from the remain S-boxes according
         to a selection strategy; */
      /* the mutation strategy is the same as the one described in
         Algorithm 3 */
    }                                              /* while opt */
                                  /* output all the individuals; */
}                                                  /* CreateSbox_GA */
```

3　算法分析及实验结果

3.1　算法的复杂性分析

设初始种群的个数 N,交叉率 p_c,变异率 p_m,染色体的长度 $Sbox\_num$。交叉运算的计算复杂性为 $N \times p_c \times N \times p_c \times Sbox\_num$;变异运算的计算复杂性为 $N \times p_m$;界于交叉运算和变异运算之间的选择运算的计算复杂性为 $N \times (N + childnum) = O(N^3)$,此外,在计算 S 盒适应值的函数中调用了两个子函数,分别是计算 S 盒的非线性度和差分均匀度,这两个函数的计算复杂度都是 $O(Sbox\_num^3)$,因此,计算适应值的复杂性为 $(N + childnum) \times Sbox\_num^3 = O(N^2 \times Sbox\_num^3)$,因此,整个算法的计算复杂性为 $O(C_1 \times N^2 \times Sbox\_num^3)$,$C_1$ 为遗传算法的进化代数。不难看出,整个算法的计算复杂性与迭代次数、种群大小及 S 盒的规模有关。

3.2　实验结果

在编程实现时将部分参数设置如下:$N = 20$, $Sbox\_num = 64$, $p_c = 0.7$, $p_m = 0.5$。

程序共执行了 5 次,则初始 S 盒与迭代后的 S 盒分别有 100 个。表 2 显示了迭代前后的不同性能 S 盒的分布情况(算法 3)。

表 2　迭代前后不同性能 S 盒的分布(算法 3)

γ	迭代前				迭代后			
	Ns				*Ns*			
	14	16	18	20	14	16	18	20
12	0	0	5	0	0	0	0	0
10	2	16	23	0	0	0	0	0
8	4	20	30	0	0	0	0	28
6	0	0	0	0	0	0	4	68

多次的实验都验证了本文提出的基于遗传算法的 S 盒构造的有效性。

表 3 显示了采用不同交叉概率和变异概率的迭代次数的对比。可见,交叉概率和变异概率对算法的执行效率有着很大的影响。

表 3　采用不同交叉概率和变异概率的迭代次数对比(算法 3)

	$p_c = 0.7$, $p_m = 0.5$	$p_c = 0.5$, $p_m = 0.5$	$p_c = 0.7$, $p_m = 0.4$
第一次运行	104 次	46 次	59 次
第二次运行	201 次	141 次	97 次
第三次运行	159 次	78 次	74 次
第四次运行	105 次	48 次	28 次
第五次运行	157 次	75 次	42 次

由表 4 可以看出,我们所采用的启发式变异策略和最佳个体保存法的选择策略,能够显著地提高算法的搜索效率,加快算法的收敛速度。

表 4　各种算法迭代次数的对比 ($p_c = 0.7$, $p_m = 0.4$)

	算法 3	算法 4	算法 5
第一次运行	59 次	255 次	232 次
第二次运行	97 次	401 次	883 次
第三次运行	74 次	190 次	186 次
第四次运行	28 次	115 次	463 次
第五次运行	42 次	160 次	227 次

4　结语

本文利用遗传算法来对 S 盒进行优化,算法中引入了启发式变异策略,该策略既可以防止优良的基因受到破坏,又可以保证群体中个体的多样性。同时,这种变异规则能够显著地提高算法的搜索效率,加快算法的收敛速度。此外,我们采用的最佳个体保存法的选择策略可以大大减少额外的计算量。基于该方法,我们给出了 6×6 的 S 盒优化的完整程序描述,并获得了一批能够有效抵抗线性分析和差分分析的 S 盒。

另外,如果将 S 盒的雪崩准则和扩散特性等其他性能亦作为演化的目标,那么可以对 S 盒的优化进行更为深入的研究。

参考文献:

[1] 冯登国, 吴文玲. 分组密码的设计与分析[M]. 北京: 清华大学出版社, 2000.

[2] 刘晓晨, 冯登国. 满足若干密码学性质的 S-盒的构造[J]. 软件学报, 2000, 11(10): 1299 - 1302.

[3] 玄光南, 程润伟. 遗传算法与工程设计[M]. 北京: 科学出版社, 2000.

[4] 陈国良, 王煦法, 庄镇泉. 遗传算法以及应用[M]. 北京: 人民邮电出版社, 1996.

[5] 潘正君, 康立山. 演化计算[M]. 北京: 清华大学出版社, 1998.

[6] Chen H, Feng D. An Effective Evolutionary Strategy for Bijective S-boxes[J]. Evolutionary Computation, 2004, 2: 2120 - 2123.