

SHA-1 算法的高速 ASIC 实现

杜晓婧^{1,2}, 李树国^{1,2}

(1 清华大学 微电子学研究所, 北京 100084; 2 清华信息科学与技术国家实验室, 北京 100084)

摘 要: SHA-1 算法是一种国际标准的安全杂凑算法. 为提高 SHA-1 算法的吞吐率, 提出了一种新的五合一架构, 该架构使 SHA-1 算法的迭代压缩由原来的 80 轮变为 16 轮, 并可使每轮中某些 f 函数和部分加法移到关键路径外, 从而缩短了关键路径, 提高了吞吐率. 在 SMIC 65 nm 的工艺下, 吞吐率达到 12.68 Gb/s, 高于已发表的同类设计.

关键词: SHA-1 算法; 高吞吐率; ASIC 实现; 逻辑化简

中图分类号: TN49

文献标识码: A

文章编号: 1000-7180(2016)10-0019-05

The High-Throughput ASIC Implementation of SHA-1 Algorithm

DU Xiao-jing^{1,2}, LI Shu-guo^{1,2}

(1 Institute of Microelectronics, Tsinghua University, Beijing 100084, China;

2 Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China)

Abstract: SHA-1 algorithm is one of the national standard for Secure Hash Algorithm. For the sake of accelerating the throughput of SHA-1 algorithm, a new 5-in-1 structure is proposed in this paper. This structure reduces the compression function from original 80 rounds to 16 5-in-1 rounds and in each round some functions f and adders can be moved out of the critical path. Based on this, we can shorten the critical path and increase the throughput. In SMIC 65nm technology, the throughput of SHA-1 can achieve 12.68 Gb/s, which is higher than that of other reported designs and can meet the requirement of high throughput. This design also supports resuming transfer.

Key words: SHA-1 algorithm; high throughput; ASIC implementation; logical simplification

1 引言

Hash 函数将任意长度的消息输入映射为固定长度的杂凑值或消息摘要输出. 在安全应用中使用的 Hash 函数成为密码学 Hash 函数, 被广泛应用于各种不同的安全应用和网络协议中.

安全杂凑算法(SHA)是使用最广泛的 Hash 函数, 由美国国家标准与技术研究所(NIST)设计, 并作为联邦信息处理标准发布. 在消息认证、数字签名、产生单向口令文件、入侵检测、病毒检测等方面有广泛应用, 并可用于构建随机函数或用做伪随机数发生器.

SHA-1 算法是 SHA 系列算法的一种, 能够产生 160 比特的固定长度输出. 随着信息安全中数据

处理速度的不断增加, 传统的 SHA-1 算法渐渐不能满足人们在密码安全应用中对速度的需求, 因此增加 SHA-1 算法的运算速率, 提高其吞吐率是我们在设计该算法实现时的首要目标. 而由于 SHA-1 算法在迭代过程中存在反馈结构, 因此不能使用常用的流水线方法来达到吞吐率提高的目的. 为加快 SHA-1 算法的运算速率, 本文提出了一种新的五合一结构和迭代加法优化, 提高了算法吞吐率, 完成了该算法的高速 ASIC 设计和实现.

2 SHA-1 算法描述

文献[1]给出了 SHA-1 算法的标准. SHA-1 算法以任意长度小于 2^{64} 比特的消息作为输入, 并生成 160 比特的杂凑值输出. SHA-1 算法可由两部分构

成:预处理和哈希运算.如图1所示,预处理由消息填充、消息分组以及设初值三部分构成,而哈希运算则是对经由消息填充和消息分组后产生的消息块依次进行迭代压缩,当所有的消息块处理完毕之后,产生最终的杂凑值.

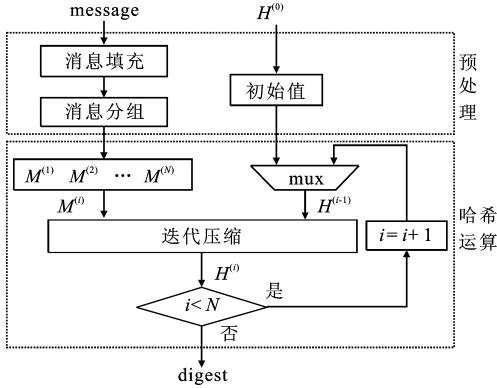


图1 SHA-1 算法结构

2.1 预处理

利用SHA-1算法对消息进行处理时首先要进行预处理.预处理包括三个部分:消息填充、消息分组以及设初值.

消息填充的目的是确保进入消息分组的消息长度为消息分组长度的倍数,即确保填充后的消息长度为512比特的倍数.假设消息 M 的长度为 l 比特,其填充由3部分构成:

(1)在消息的末尾添加比特“1”;

(2)在添加的比特“1”后再添加 k 个“0”,其中 k 是满足 $l+1+k \equiv 448 \pmod{512}$ 的最小非负整数;

(3)添加一个64位比特串,该比特串是长度 l 的二进制表示.

由此填充后消息的比特长度为512的倍数.

消息分组将填充后的消息按512比特进行分组,共分为 N 个消息块: $M^{(1)}, M^{(2)}, \dots, M^{(N)}$.这 N 个消息块依次进入哈希运算进行迭代压缩.

最后,在进行哈希运算前为本次运算设置初始值 $H^{(0)}$,SHA-1算法规定初始值 $H^{(0)}$ 的长度为160比特,也即最后输出杂凑值的长度,且:

$$H^{(0)} = 160'h67452301_efcdab89_98badcf_10325476_c3d2e1f0.$$

2.2 哈希运算

消息经过预处理后生成 N 个消息块 $M^{(1)}, M^{(2)}, \dots, M^{(N)}$ 进入哈希运算.哈希运算是整个SHA-1算法的核心,其中最关键的是每轮迭代压缩中的80轮轮函数.

以 $H^{(0)}$ 为初始向量,哈希运算对分组后的 N 个消息块 $M^{(1)}, M^{(2)}, \dots, M^{(N)}$ 依次进行迭代压缩,其定义为:

FOR $i = 1$ to N :

$$a = H_0^{(i-1)}$$

$$b = H_1^{(i-1)}$$

$$c = H_2^{(i-1)}$$

$$d = H_3^{(i-1)}$$

$$e = H_4^{(i-1)}$$

FOR $t = 0$ to 79:

$$T = a \lll 5 + f_t(b, c, d) + e + K_t + W_t$$

$$e = d$$

$$d = c$$

$$c = b \lll 30$$

$$b = a$$

$$a = T$$

ENDFOR

$$H_0^{(i)} = a + H_0^{(i-1)}$$

$$H_1^{(i)} = b + H_1^{(i-1)}$$

$$H_2^{(i)} = c + H_2^{(i-1)}$$

$$H_3^{(i)} = d + H_3^{(i-1)}$$

$$H_4^{(i)} = e + H_4^{(i-1)}$$

ENDFOR

其中, a, b, c, d, e 为字寄存器, T 为32比特中间值, $H^{(0)} = H_0^{(0)} || H_1^{(0)} || H_2^{(0)} || H_3^{(0)} || H_4^{(0)}$ 为160比特的初始向量, $H^{(i-1)}$ 为每轮迭代压缩的起始值并赋值给 a, b, c, d, e 这5个寄存器以开始本轮的80轮轮函数. $H^{(i)} = H_0^{(i)} || H_1^{(i)} || H_2^{(i)} || H_3^{(i)} || H_4^{(i)}$ 为迭代压缩的中间结果,当 N 个消息块依次完成迭代压缩后, $H^{(N)} = H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)}$ 即为最终的杂凑值输出.

在上述定义中, K_t 为常数, f_t 为函数, W_t 为消息块 $M^{(i)}$ 及其扩展得到的字,它们的表达式分别为:

$$K_t = \begin{cases} 5a827999 & 0 \leq t \leq 19 \\ 6ed9eba1 & 20 \leq t \leq 39 \\ 8f1bbcdc & 40 \leq t \leq 59 \\ ca62c1d6 & 60 \leq t \leq 79 \end{cases}$$

$$f_t(b, c, d)$$

$$= \begin{cases} (b \wedge c) \oplus (\neg b \wedge d) & 0 \leq t \leq 19 \\ b \oplus c \oplus d & 20 \leq t \leq 39 \\ (b \wedge c) \oplus (b \wedge d) \oplus (c \wedge d) & 40 \leq t \leq 59 \\ b \oplus c \oplus d & 60 \leq t \leq 79 \end{cases}$$

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1 & 16 \leq t \leq 79 \end{cases}$$

在传统 SHA-1 算法的哈希运算中,迭代压缩过程中有 80 轮轮函数,每个消息块的迭代压缩过程如图 2 所示,消息块 $M^{(i)}$ 扩展为 80 个 32 比特的字 $W_0 \sim W_{79}$ 分别作用于第 0 轮到第 79 轮运算中,每轮轮函数的结构如图 3 所示。

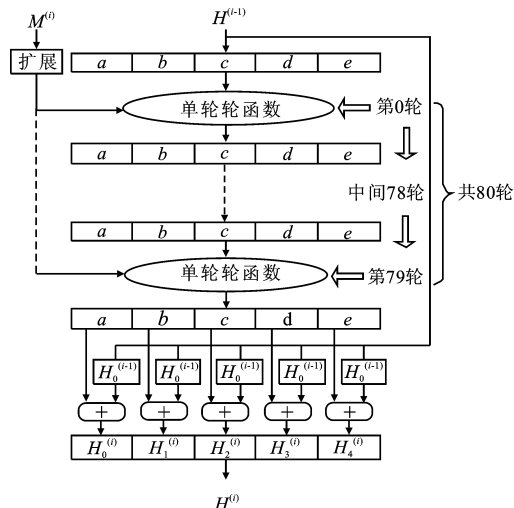


图 2 迭代压缩结构

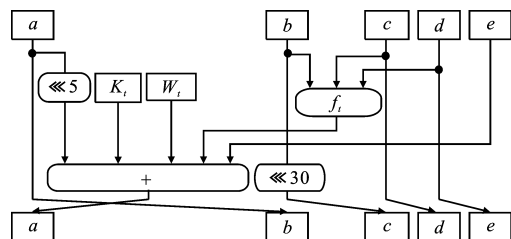


图 3 SHA-1 算法单轮轮函数结构

因此,SHA-1 算法在每处理一个 512 比特的消息块时,若每个周期完成单轮轮函数的运算则需要 80 个周期,且每个周期中的轮函数结构简单,可化简的逻辑较少,时钟频率较高,产生的功耗也较大。另外,根据现在常用的接口设计,相同时钟频率下,传统 SHA-1 算法处理数据的速度远低于数据传输的速度,使得数据的输入要等待算法的运算,造成资源浪费。因此,我们需要增大 SHA-1 算法的运算速率,尽可能的提高其吞吐率。

3 SHA-1 算法设计

3.1 五合一实现架构

我们将五轮轮函数在一个周期内实现的结构称为五合一结构。由 2.2 节中给出的 SHA-1 算法定义可知,该算法的核心在于迭代压缩中的 80 轮轮函数

的运算。由图 3 可以看出,SHA-1 算法的单轮轮函数结构简单,如果每个周期实现单轮轮函数则面积较小,但单轮中逻辑化简空间少,时钟频率高且算法功耗大。因此本文提出了一种在一个周期中实现多轮轮函数的结构,简称多合一结构。在这种情况下,一个时钟周期内要完成多轮轮函数的运算,降低了时钟周期数,每个周期延时增大,时钟频率降低功耗减小,且在多轮合一的过程中有更多的逻辑化简空间,为吞吐率的提高提供了优化空间。

多合一结构有多种实现方式,如五合一、十合一等。本文以五合一结构为例,给出了五合一结构的实现方式及其逻辑化简。图 4 为采用了五合一结构后 SHA-1 算法的迭代压缩过程,迭代压缩由原来的 80 轮变为现在的 16 轮,且在每轮中进行五轮单轮轮函数的运算。此时,每处理一个 512 比特的消息块所用的时钟周期数由原来的 80 个时钟周期减少到 16 个 (80/5),同时时钟周期延时增大,时钟频率降低,功耗减小。

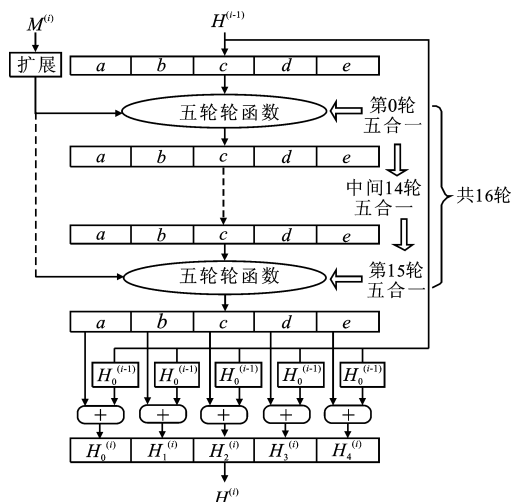


图 4 五合一迭代压缩结构

如图 5 所示为图 4 中由五轮轮函数组成的五合一结构。其中, a, b, c, d, e 对应于图 4 中每个周期期间的寄存器 a, b, c, d, e 。中间虚线框内表示的 $a_1, b_1, c_1, d_1, e_1, a_2, b_2, c_2, d_2, e_2, a_3, b_3, c_3, d_3, e_3, a_4, b_4, c_4, d_4, e_4$ 对应于图 2 单轮结构时相应轮函数得到的寄存器的值,此时在五合一结构中转化为中间值并参与逻辑化简。

3.2 五合一结构化简

由图 5 可以看出,五合一结构存在很大的化简空间。为加快算法的运算速率,提高算法吞吐率,需要对五合一结构进行优化。本文提出了一种提前计算的方式,将关键路径上的运算移到非关键路径。

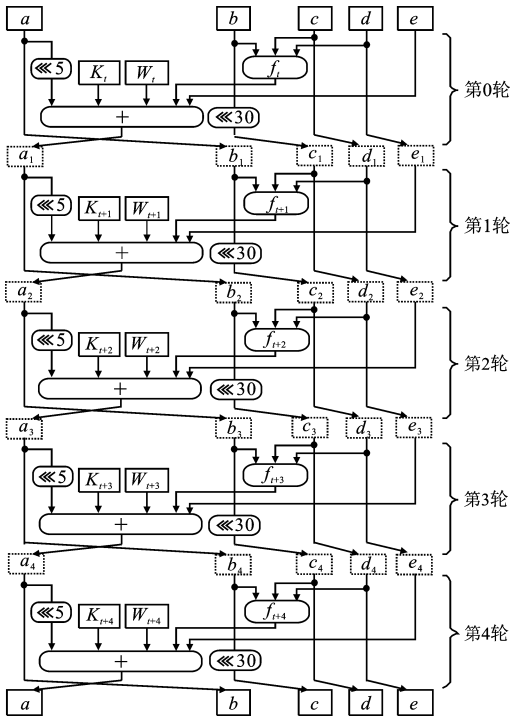


图 5 未化简的五合一结构

首先对函数 f 进行提前运算. f 是 b, c, d 以及 t 的函数, 由 2.2 节中哈希运算给出的定义以及图 5 所示的结构图可以看出, 从五合一结构的第 1 轮开始, 函数 f 的变量 b, c, d 从上一轮的运算开始前即可得到. 表 1 所示为五合一结构中函数 f 及其变量的值.

表 1 函数 f 的提前计算

第 n 轮	b_n	c_n	d_n	$f_{t+n}(b_n, c_n, d_n)$
0	b	c	d	$f_t(b, c, d)$
1	a	$b \lll 30$	c	$f_{t+1}(a, (b \lll 30), c)$
2	a_1	$b_1 \lll 30$	c_1	$f_{t+2}(a_1, (b_1 \lll 30), c_1)$
3	a_2	$b_2 \lll 30$	c_2	$f_{t+3}(a_2, (b_2 \lll 30), c_2)$
4	a_3	$b_3 \lll 30$	c_3	$f_{t+4}(a_3, (b_3 \lll 30), c_3)$

因此, 在五合一结构中, 后 4 轮每个单轮运算中的函数 f 可以在其前一轮提前计算得到, 从关键路径移到了非关键路径.

同理对五合一结构中五个单轮的部分加法进行提前计算. 表 2 给出了五合一结构每轮中变量 e 的值.

由于 K_t 为 t 决定的常数, W_t 为消息块 $M^{(i)}$ 扩展得到的字且在每个五合一结构进行运算前计算得到. 因此, 在五合一结构里后四轮中 e 与 K_t, W_t 的加

表 2 加法的提前计算

第 n 轮	e_n	$e_n + K_{t+n} + W_{t+n}$
0	e	$e + K_t + W_t$
1	d	$d + K_{t+1} + W_{t+1}$
2	c	$c + K_{t+2} + W_{t+2}$
3	$b \lll 30$	$(b \lll 30) + K_{t+3} + W_{t+3}$
4	$a \lll 30$	$(a \lll 30) + K_{t+4} + W_{t+4}$

法运算可以在第 0 轮中提前计算.

综上, 对于优化后的五合一结构, 第 0 轮保持不变, 后四轮中函数 f 在前一轮中提前运算, e 与 K, W 的加法在第 0 轮提前计算完成. 优化后的五合一结构如图 6 所示.

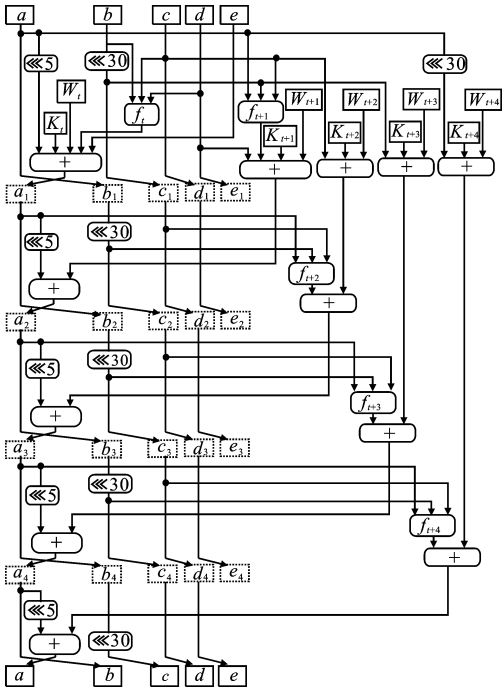


图 6 化简后的五合一结构

3.3 迭代加法的优化

SHA-1 算法对每个填充分组后的消息块进行迭代压缩, 如图 4 所示, 每个消息块进行压缩后以加法的形式进行迭代. 迭代压缩将每个消息块 $M^{(i)}$ 开始进行处理的起始值 $H^{(i-1)}$ 赋值到寄存器 a, b, c, d, e 中, $a || b || c || d || e = H^{(i-1)}$, 在经过 16 轮五合一结构后与自身分别做加法并拼接生成下一轮的起始值 $H^{(i)} = (a + H_0^{(i-1)}) || (b + H_0^{(i-1)}) || (c + H_0^{(i-1)}) || (d + H_0^{(i-1)}) || (e + H_0^{(i-1)})$.

因此, 在五合一结构中, 完成一个 512 比特消息的处理需要 16 个周期, 而在最后一个周期中, 除五

合一结构的运算外还需额外完成一个加法运算. 由于寄存器 a 的运算为关键路径,即相当于在最后一个周期中, a 需要额外运算一个与 $H_0^{(i-1)}$ 的加法,而 b, c, d, e 与 $H_1^{(i-1)}, H_2^{(i-1)}, H_3^{(i-1)}, H_4^{(i-1)}$ 的加法并不影响关键路径.

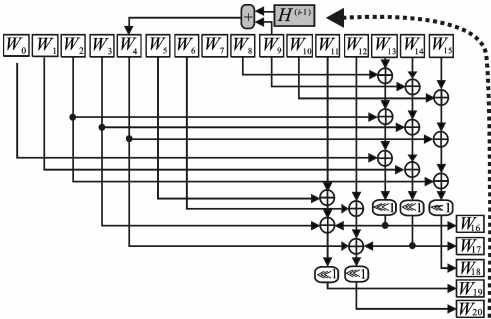


图 7 消息块的扩展

图 7 所示为采用了五合一结构时 SHA-1 中消息块的扩展过程. $W_0 \sim W_{20}$ 为寄存器,512 比特的消息块 $M^{(i)}$ 首先赋值到 $W_0 \sim W_{15}$ 这 16 个 32 比特的寄存器中,在迭代压缩过程中, $W_0 \sim W_4$ 作用于每轮的五合一结构中,每经过一个周期, $W_5 \sim W_{20}$ 分别赋值给 $W_0 \sim W_{15}$,因此在 16 轮中最后一轮用到的 $W_0 \sim W_4$ 即为 SHA-1 算法定义中的 $W_{75} \sim W_{79}$.

为改进最后一轮中迭代的加法,在第 14 轮 W 寄存器赋值过程中增加阴影的部分的计算,即 W_t 的表达式变为如下所示,此时在最后一轮中 a 的路径不变,与前 15 轮保持一致.

W_t =

$$\begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1 & 16 \leq t \leq 78 \\ (W_{79} \oplus W_{71} \oplus W_{65} \oplus W_{63}) \lll 1 + H_0^{(i-1)} & t = 79 \end{cases}$$

4 功能验证与性能分析

4.1 功能验证

根据以上方法,我们用 Verilog HDL 实现了 SHA-1 算法的高速 ASIC 设计,并利用 modelsim 仿真对其进行功能验证. 将输入消息作用于 Verilog 语言生成杂凑值,并使其结果与相同输入下经过 C 语言功能模型产生的结果进行比对. 为验证 SHA-1 算法功能的正确性,输入消息从 8 比特开始依次增加 8 比特进行遍历测试,经过大量数据测试,本设计功能正确.

4.2 性能分析

采用多合一的设计思想,除本文中采用的五合一结构,我们还分别完成了 SHA-1 算法采用十合

一、八十合一的 ASIC 设计与实现,并利用 DC 综合工具对电路进行综合,同时与传统的采用单轮结构下的算法设计进行比较. 在 SMIC 65 nm 的工艺下,综合结果如表 3 所示.

表 3 不同结构的综合结果

设计	吞吐量/ (Gb/s)	功耗/ mW	面积/ μm^2	延时/ ns
单轮	5.73	16.553 8	34 237	1.04
五合一	11.50	8.782 0	53 902	2.59
十合一	12.68	4.512 0	77 347	4.70
八十合一	14.44	6.043 0	333 727	32.95

根据本文提出的多合一结构以及逻辑化简思想,五合一结构时吞吐率为 11.50 Gb/s,十合一时吞吐率为 12.68 Gb/s,对比于其他同类设计以及 IP 设计公司 Helion 和 Cast 的同类设计,吞吐率有很大的提高. 如表 4 所示,本设计在性能上优于其他同类设计.

表 4 与其他设计性能对比

设计	工艺	吞吐量	面积
本设计	65 nm	12.68 Gb/s	77 347 μm^2
文献[2]	FPGA	5.9 Gb/s	2 894 Slices
本设计	0.18 μm	4.23 Gb/s	381 366 μm^2
文献[3]	0.18 μm	3.9 Gb/s	12 900 gates
CAST[4]	0.18 μm	350 Mb/s	14 500 gates
本设计	0.13 μm	7.02 Gb/s	198 280 μm^2
Helion[5]	0.13 μm	2.14 Gb/s	16 000 gates

5 结束语

针对 SHA-1 算法的高速 ASIC 实现,本文提出了多合一结构,并具体给出了一种五合一结构及其逻辑化简方法,减少了时钟周期数,提高了算法的吞吐率,降低了功耗. 在 SMIC 65 nm 工艺下达到了 12.68Gb/s,高于当前同类设计. 同时该多合一方法对于其他杂凑算法也具有较高的参考价值.

参考文献:

[1] FIPS. FIPS PUB 180-4. Secure hash standard[S]. USA, the National Instiute of Standards and Technology(NIST),2015.

运行的统计结果,可以看出,DFA 在较短的时间($Fes=10\ 000$)、在所有测试函数上的结果也都好于或者等于其他算法.除 Rosenbrock 尚未收敛到理论最小值点之外,其余函数都已求得理论最小值,这反映 DFA 的收敛速度较快.在两个很难优化的函数 Rosenbrock 和 Schwefel 上,DFA 比 FA 有显著的优势.

6 结束语

本文基于烟花进化速度改进了烟花算法中每次火花的更新维数 z ,使得算法在接近陷入局部最优的时候,通过动态的调整 z ,增大算法跳出局部极小值的机会.本文提出的改进算法没有改变原始烟花算法的结构,仅改变烟花算法的参数设置方式,因此没有增加原始烟花算法的计算复杂度.所以,对于基于原始烟花算法所做的改进算法,都可以移植到本文改进算法中.实验表明,改进算法提高了搜索精度.本文提出的动态更新维数的思想,也可以推广到其他进化算法中,例如粒子群、蜂群算法,期待更有效的改进.

参考文献:

[1] Tan Y,Zhu Y C. Fireworks algorithms for optimization [C]//Proceedings of Int. Conf. on Swarm Intelligence (ICSI2010), Part II. Beijing, China : LNCS 6145, 2010:355-364.

[2] Eun Hee Lee,Je Hoon Lee,Il Hwan Park,et al. Implementation of high -speed SHA-1 architecture[J]. IE-ICE Electronics Express,2009,16(6):1174-1179.

[3] 黄淳,白国强,陈弘毅. 快速实现 SHA-1 算法的硬件结构[J]. 清华大学学报,2005,45(1):123-125.

[4] SHA1. Secure hash algorithm core[EB/OL]. [2016-02-20]. <http://www.cast-inc.com/ip-cores/encryption/sha-1/index.html>.

[5] Helion Technology. Fast Hash IP cores for ASIC[EB/OL]. [2014-11-07]. http://www.heliontech.com/fast_hash.htm.

[6] William Stallings. 密码编码学与网络安全:原理与实践[M]. 5 版. 北京:电子工业出版社,2013.

[2] Pholdee N,Bureerat S. Comparative performance of meta-heuristic algorithm for mass minimission of trusses with dynamic constraints[J]. Advances in Engineering Software,2014,75(9):1-13.

[3] 杜振鑫. 烟花算法中爆炸半径的改进研究[J]. 计算机时代,2013,30(1):28-29.

[4] Zheng S,Andreas J, Tan Y. Enhanced fireworks algorithm[C]// IEEE International Conference on Evolutionary Computation. Cancun, MEXICO :IEEE,2013:2069-2077.

[5] Pei Y,Zheng S,Tan Y,et al. An empirical study on influence of approximation approaches on enhancing fireworks algorithm[C]// Proceeings of the 2012IEEE congress on System. Man and Cybernetics, Seoul, South Korea;IEEE,2012: 1322-1327.

[6] Tan Y, Shi Y, Mo H. The improvement on controlling exploration and exploitation of firework algorithm [C]// ICSI 2013, Part I, LNCS 7928. Berlin Heidelberg: Springer_Verlag,2013: 11-23.

[7] 谭莹. 烟花算法引论[M]. 北京:科学出版社,2015:3-12.

作者简介:

杜振鑫 男,(1976-),博士研究生,讲师.研究方向为数据挖掘、计算智能. E-mail:gladmail@163.com.

[7] He J,Chen H,Huang H. A compatible SHA series design based on FPGA[C]//Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON), 2010 International Conference on. Thailand,IEEE, 2010:380-384.

作者简介:

杜晓婧 女,(1990-),硕士研究生.研究方向为信息安全算法及其大规模集成电路设计与实现. E-mail:duxiaojing1727@163.com.

李树国 男,(1963-),教授,博士生导师.研究方向为信息安全算法及其大规模集成电路设计与实现.

(上接第 23 页)