Searching for Quasigroups for Hash Functions with Genetic Algorithms

Václav Snášel, Jiří Dvorský, Eliška Ochodková, Pavel Krömer, Jan Platoš
Department of Computer Science,
VŠB - Technical University of Ostrava
17. listopadu 15, 708 33
Ostrava-Poruba, Czech Republic
{vaclav.snasel,jiri.dvorsky,eliska.ochodkova,pavel.kromer,jan.platos}@vsb.cz

Ajith Abraham
Center of Excellence for Quantifiable
Quality of Service, Norwegian
University of Science and Technology
O.S. Bragstads plass 2E,
N-7491 Trondheim, Norway
Email: ajith.abraham@ieee.org

***Abstract*—In this study we discuss a method for evolution of quasigroups with desired properties based on genetic algorithms. Quasigroups are a well-known combinatorial design equivalent to more familiar Latin squares. One of their most important properties is that all possible elements of certain quasigroup occur with equal probability. The quasigroups are evolved within a framework of a simple hash function. Prior implementations of quasigroups were based on look-up table of the quasigroup, on system of distinct representatives etc. which is infeasible for large quasigroups. In contrast, analytic quasigroup can be implemented easily. It allows the evaluation of hash function without storing large amount of data (look-up table) and the concept of isotopy enables consideration of many quasigropus.**

I. Introduction

The need for random and pseudorandom sequences arises in many applications, e.g. in modeling, simulations, and of course in cryptography. Pseudorandom sequences are the core of stream ciphers. They are popular due to their high encryption/decryption speed. Their simple and cheap hardware design is often preferred in real-world applications. The design goal in stream ciphers is to efficiently produce pseudorandom sequences - keystreams (i.e. sequences that possess properties common to truly random sequences and in some sense are "indistinguishable" from these sequences).

Hash functions map a large collection of messages into a small set of message digests and can be used for error detection, by appending the digest to the message during the transmission (the appended digest bits are also called parity bits). The error will be detected if the digest of the received message, in the receiving end, is not equal to the received message digest. This application of hash functions is only for random errors, since an active spoofer may intercept the transmitted message, modify it as he wishes, and resend it appended with the digest recalculated for the modified message. Informally, the cryptographic hash function H must have the following properties:

- H can be applied to a block of data of any size.
- H produces a fixed-length output.
- $H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
- For any given value y, it is computationally infeasible to find x such that $H(x) = h$ - one-way property.
- For any given block x_1, it is computationally infeasible to find $x_2 \neq x_1$ such that $H(x_2) = H(x_1)$ - weak collision resistance property.
- It is computationally infeasible to find any pair (x_1, x_2) such that $H(x_1) = H(x_2)$ - strong collision resistance property.

With the advent of public key cryptography and digital signature schemes, cryptographic hash functions gained much more prominence. Using hash functions, it is possible to produce a fixed length digital signature that depends on the whole message and ensures authenticity of the message. To produce digital signature for a message M, the digest of M, given by $H(M)$, is calculated and then encrypted with the secret key of the sender. Encryption may be done either by using a public key or a private key algorithm. Encryption of the digest prevents active intruders from modifying the message and recalculating its checksum accordingly. It effectively divides the universe of users into two groups: outsiders who do not have access to the key of the encryption algorithm and hence cannot effectively produce a valid checksum, and insiders who do have access to the key and hence can produce valid checksums. We note that in a public key algorithm, the group of insiders consists of only one member (the owner of the private key) and hence the encrypted hash value uniquely identifies the signer. In the case of symmetric key algorithms, both the transmitter and the receiver have access to the secret key and can produce a valid encrypted hash for an arbitrary message and therefore, unique identification based on the encrypted hash is not possible. However, an outsider cannot alter the message or the digest.

The use of quasigroups and quasigroup string transformations is a recent but successful tendency in cryptography and coding [13]. With quasigroups in the hearth of advanced cryptosystems and hash functions, a need to find good quasigroups arose.

Genetic algorithms are probably the most popular and wide spread member of the class of evolutionary algorithms (EA). EAs found a group of iterative stochastic search and optimization methods based on mimicking successful optimization

978-1-4244-5612-3/09/$26.00

strategies observed in nature [3], [4], [12], [17]. The essence of EAs lies in the emulation of Darwinian evolution, utilizing the concepts of Mendelian inheritance for use in computer science [3]. Together with fuzzy sets, neural networks, and fractals, evolutionary algorithms are among the fundamental members of the class of soft computing methods.

EAs operate with a population (also known as a pool) of artificial individuals (also referred to as items or chromosomes) encoding possible problem solutions. Encoded individuals are evaluated using a carefully selected objective function which assigns a fitness value to each individual. The fitness value represents the quality (ranking) of each individual as a solution to a given problem. Competing individuals explore the problem domain towards an optimal solution [12].

II. Construction of hash function based on quasigroup

Definition 2.1: A quasigroup is a pair $(Q, \circ)$, where $\circ$ is a binary operation on (finite) set Q such that for all not necessarily distinct $a, b \in Q$, the equations

$$a \circ x = b \text{ and } y \circ a = b.$$

have unique solutions.

The fact that the solutions are unique guarantees that no element occurs twice in any row or column of the table for $(\circ)$. However, in general, the operation $(\circ)$ is neither a commutative nor an associative operation.

Quasigroups are equivalent to more familiar Latin squares. The multiplication table of a quasigroup of order q is a Latin square of order q, and conversely, as it was indicated in [2], [5], [19], every Latin square of order q is the multiplication table of a quasigroup of order q.

Definition 2.2: Let $A = \{a_1, a_2, \ldots, a_n\}$ be a finite alphabet, a $n \times n$ Latin square L is a matrix with entries $l_{ij} \in A$, $i, j = 1, 2, \ldots, n$, such that each row and each column consists of different elements of A.

For $i, j, k \in A$ the ordered triple $(i, j; k)$ is used to represent the occurrence of element k in cell (i, j) of the Latin square. So a Latin square may be represented by the set $\{(i, j; k)|$ entry k occurs in cell (i, j) of the Latin square $L.\}$

All reduced Latin squares of order n are enumerated for $n \leq 11$ [15]. Let L_n be the number of Latin squares of order n, and let R_n be the number of reduced Latin squares of order n. It can be easily seen that

$$L_n = n!(n-1)!R_n.$$

Number of distinct Latin squares of a given order grows exceedingly quickly with the order, there is no known easily-computable formula for the number of distinct Latin squares. The problem of classification and exact enumeration of Latin squares of order greater than 11 probably still remains unsolved. Thus, there are more than 10^{90} quasigroups of order 16 and if we take an alphabet $L = \{0 \ldots 255\}$ (i.e. data are represented by 8 bits) there are at least $256!255! \ldots 2! > 10^{58000}$ quasigroups.

Multiplication in quasigroups has an important property; it is proved that each element occurs exactly q times among the products of two elements of Q, q^2 times among the products of three elements of Q and, generally q^{t-1} among the products of t elements of Q. Since there are q^t possible ordered products of t elements of Q, this shows that each element occurs equally often among these q^t products (see [6]).

Definition 2.3: Let $(Q, \circ)$ be a quasigroup and Q^+ be a set of all nonempty words formed by the elements $q_i \in Q$, $1 \leq i \leq n$.

For a fixed $a \in Q$ let the *hash function* $h_Q : Q \times Q^+ \to Q^+$ be defined as

$$h_Q(a, q_1 q_2 \ldots q_n) = (((a \circ q_1) \circ q_2 \ldots) \circ q_n.$$

Example 2.1: Quasigroup of modular subtraction has following table representation:

$\circ$	0	1	2	3
0	0	3	2	1
1	1	0	3	2
2	2	1	0	3
3	3	2	1	0

The table above defines quasigroup because it satisfies conditions to be Latin Square. Multiplication in the quasigroup is defined in following manner: $a \circ b = (a + 4 - b)\ mod 4$. It is obvious that the quasigroup is neither commutative ($1 \circ 2 = 3$, $2 \circ 1 = 1$) nor associative. Value of hash function is $H_2(0013) = (((2 \circ 0) \circ 0) \circ 1) \circ 3 = 2$.

A. Sketch of proof of resistance to brute-force attack

Hash function based on quasigroup is iterative process which computes hash value (digest) for message $X = x_1 x_2 \ldots x_n$. Suppose that $H_Q(X) = d$. Hash function is preimage resistant when it is "impossible" to compute from given digest source message X. The digest d should be factorized into message $Y = y_1 y_2 \ldots y_n$. In the first step we can divide digest d into two parts y_1 and α_1, where $d = y_1 \circ \alpha_1$. In the second step value α_1 needs to be divided into y_2 and α_2 ($\alpha_1 = y_2 \circ \alpha_2$) and so for each element $y_i, 1 \leq i \leq n$. Because each y_i has a same probability of occurrence among products of Q, $|Q|^n$ possible choices should be checked to obtain message Y.

However, proposed hash function is rather theoretical than practical construction, not suitable for cryptography. It is constructed very simply, e.g. it is not an iterated hash function [16], the most used structure of hash functions in use today. It was shown in [21] that one can create false messages that hash to the same value, too. It follows this hash function can be used rather for the hash table data structure.

Definition 2.4: Let $(G, \cdot)$, $(H, \circ)$ be two quasigroups. An ordered triple (π, ρ, ω) of bijections π, ρ, ω of the set G onto set H is called an *isotopism* of $(G, \cdot)$ upon $(H, \circ)$ if $\forall u, v \in G, \pi(u) \circ \rho(v) = \omega(u \cdot v)$. Quasigroups $(G, \cdot)$, $(H, \circ)$ are said to be *isotopic*.

We can imagine an isotopism of quasigroups as a permutation of rows and columns of quasigroup's multiplication table.

Example 2.2: Table of quasigroup, which is isotopic to the quasigroup of modular subtraction:

$\circ$	0	1	2	3
0	0	3	2	1
1	2	1	0	3
2	1	0	3	2
3	3	2	1	0

The table was created from table of modular subtraction. The second and the third row were exchanged. Permutations π, ρ are identities and $\omega = [0213]$. For example $1 \circ 0 = \omega(1) \circ 0 = 2 \circ 0 = 2$.

From the quasigroup of modular subtraction we can move to a large number of the quasigroup isotopic to quasigroup of modular subtraction [7], [18]. This allows us to use quasigroups with a very large number of elements without necessity of their storage. The multiplication in such an isotopic quasigroup is defined as follows:

$$a \circ b = \pi((\omega(a) + n - \rho(b)) \bmod n). \quad (1)$$

We call the quasigroup defined by its multiplication and three selected permutations an "analytic quasigroup".

This enables efficient work with large quasigroups. Works that are already known use mostly quasigroups of small order only [10], or only a small parts of certain quasigroup are utilized mainly as a key for Message Authentication Code. These are represented as a look-up table in main memory. Larger quasigroup of order 2^{256} is used by NIST's SHA-3 competition[1] candidate, hash function Edon$\mathcal{R}$ [9].

The properties of one analytic quasigroup isotopic to the quasigroup of modular subtraction were studied in [11]. The quasigroup was created using three static functions that divided the sequence of n elements of the quasigroup into several parts. The parts were rotated in various directions and exchanged among themselves. It was shown that the investigated quasigroup has some faults in its properties.

In this work, we study the effect of different permutations used to create isotopic quasigroups and the possibilities to search for analytic quasigroups isotopic to the quasigroup of modular subtraction by genetic algorithms.

[1]http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

B. Constructing quasigroups isotopic to the quasigroup of modular subtraction

Consider a quasigroup on the length n defined by multiplication $a \circ b = (a + n - b) \bmod n)$. Then three permutations π, ρ, ω must be chosen in order to implement isotopic quasigroup, whose multiplication will be defined as shown in (1).

Obviously, there is $n!$ different permutations of n elements. Because three distinctive permutations are used to define isotopic quasigroup, there are $n!n!n!$ possible choices of π, ρ and ω

Permutations of elements cannot be sought for an analytic quasigroup directly, because its elements are not stored in memory. Instead, the permutation needs to be implemented as a function of element of Q. One way to achieve this goal is the use of bit permutation.

A quasigroup over a set of n elements requires $log_2(n)$ bits to express every element. Each permutation of bits in the element representation is also a permutation of the elements of the quasigroup (if n is a power of 2). Bit permutation can be implemented easily as a function of $q \in Q$.

The bit permutation is a simple and straightforward way of implementing permutations over n elements of Q. Although it allows us to explore only a fragment $(log_2(n)!log_2(n)!log_2(n)!)$ of all possible permutation triples over the quasigroup of n elements, it is useful because it does not require all n elements in main memory and therefore fits into the framework of analytic quasigroups.

Bit permutations are more costly than the static functions used to implement permutation in [11], but there are ongoing efforts to implement bit permutation instructions in hardware, which would improve the performance of the proposed algorithm significantly [8].

III. Genetic algorithms

Genetic algorithms are generic and reusable population-based metaheuristic soft optimization method [1], [12], [17]. GAs operate with a population of chromosomes encoding potential problem solutions. Encoded individuals are evaluated using a carefully selected domain specific objective function which assigns a fitness value to each individual. The fitness value represents the quality of each candidate solution in context of the given problem. Competing individuals explore the problem domain towards an optimal solution [12]. The solutions might be encoded as binary strings, real vectors or more complex, often tree-like, hierarchical structures (subject of genetic programming [14]). The encoding selection is based on the needs of particular application area.

The emulated evolution is driven by iterative application of genetic operators. Genetic operators algoritmize principles observed in natural evolution. The crossover operator defines a strategy for the exchange of genetic information between parents (sexual reproduction of haploid organisms) while the mutation operator introduces the effect of environment and randomness (random perturbation of genetic information).

Other genetic operators define e.g. parent selection strategy or the strategy to form new population from the current one. Genetic operators and algorithm termination criteria are the most influential parameters of every evolutionary algorithm. The operators are subject to domain specific modifications and tuning [17]. The basic workflow of the standard generational GA (GGA) is shown in Fig. 1.

1 Define objective (fitness) function and problem encoding
2 Encode initial population P of possible solutions as fixed length strings
3 Evaluate chromosomes in initial population using objective function
4 **while** *Termination criteria not satisfied* **do**
5 Apply selection operator to select parent chromosomes for reproduction: $sel(P_i) \rightarrow parent1$, $sel(P_i) \rightarrow parent2$
6 Apply crossover operator on parents with respect to crossover probability to produce new chromosomes: $cross(pC, parent1, parent2) \rightarrow \{offspring1, offspring2\}$
7 Apply mutation operator on offspring chromosomes with respect to mutation probability: $mut(pM, offspring1) \rightarrow offspring1$, $mut(pM, offspring2) \rightarrow offspring2$
8 Create new population from current population and offspring chromosomes: $migrate(offspring1, offsprig2, P_i) \rightarrow P_{i+1}$
9 **end**

Fig. 1. A summary of genetic algorithm

Many variants of the standard generational GA have been proposed. The differences are mostly in particular selection, crossover, mutation and replacement strategy [12].

In the next section, we present genetic algorithm for the search for analytic quasigroups.

IV. Genetic search for analytic quasigroups

The genetic algorithm for the search for analytic quasigroup relies on encoding of the candidate solutions and fitness function to evaluate chromosomes.

A. Encoding

As identified in II-B, an analytic quasigroup isotopic to quasigroup of modular subtraction is defined by three permutations. Such permutation triple must be mapped to one chromosome. Permutations can be for the purpose of genetic algorithms encoded using several strategies. In this study, we use random key encoding.

Random key (RK) encoding is an encoding strategy available for problems involving permutation evolution [20]. In random key encoding, the permutation is represented as a string of real numbers (random keys), whose position changes after sorting corresponds to the permutation index. An example or random key encoding is shown in (2).

$$\Pi_5 = \begin{pmatrix} 0.2 & 0.3 & 0.1 & 0.5 & 0.4 \\ 2 & 3 & 1 & 5 & 4 \end{pmatrix} \tag{2}$$

To encode a quasigroup (isotopic to the quasigroup of modular subtraction) of the length $n = 2^l$, we use a vector of $3l$ real numbers $v = (v_1, \ldots, v_l - 1, v_l, \ldots v_{2l-1}, v_{2l}, \ldots, v_{3l})$. The vector is interpreted as three concatenated RK encoded permutations of the length l.

This encoding allows us to use traditional implementations of genetic operators, such as n-point crossover and mutation. Crossover was implemented as mutual exchange of genes between selected parents and mutation was implemented as a replacement of gene with a uniform random number from the interval $[0, 1]$.

B. Fitness function

Fitness function is used to rank candidate solutions among themselves. In this work, we have adopted one of the properties of hash function based on quasigroups investigated in [11], namely the distribution of lengths of slots of the hash function. The fitness function is defined in (3), where $slots$ represents a vector of the size n containing the number of occurences of each element of the quasigroup as a result of the hash procedure. The functions $\max(slots)$ and $\sigma(slots)$ are used to evaluate the maximum element of $slots$ and standard deviation of $slots$ respectively.

$$fit(v) = \frac{\max(slots) - \sigma(slots)}{\max(slots)} \tag{3}$$

The fitness function assigns higher value when there is low variance in the length of slots and lower value when there is high variance in the length of slots. The variance depends on the data processed by the hash function. We have used a training data collection created from the NIST correct answer testing data set that is used also in the competition for the SHA-3 algorithm. The collection contains 2048 messages that have different length and structure. Each message was processed by the hash function and its hash value was observed.

The second data collection contained distinctive words refined from the WebTREC [22] test collection. The collection consists of one million text/html pages downloaded from the .gov domain in the early 2002. It also includes text/plain and the extracted text of pdf, doc and ps documents. We have extracted a set of about 50 MBs of different terms from the parent corpus. The extracted words included valid English terms and erroneous strings such as *javacardforum* and others created during automatted text processing.

Two collections were used to speed up the evaluation of the chromosomes (the smaller collection was processed quickly) and observe the generality of found quasigroup.