GEVOSH: A Grammatical Evolution System

Patrick Berarducci[1], Demetrius Jordan[1], David Martin[1], Jennifer Seitzer[1]

[1] Computer Science Department, University of Dayton,
Dayton, OH 45469-2160

{ berardpb, jordandt, martindw, seitzer @ notes.udayton.edu}

Abstract. In this paper, we present system GEVOSH, *Grammatically Evolved Hashing*. GEVOSH evolves hashing functions using grammatical evolution techniques. Hashing functions are used to expedite search in a wide number of domains. In our work, GEVOSH created hashing functions that, on average, perform better than many standard (human-generated) hash functions extracted from the literature. In this paper, we present the architecture of system GEVOSH, its main components and algorithms, and resultant generated hash functions along with comparisons to standard, human-generated functions.

1 Introduction

Grammatical Evolution is a method of machine generating computer programs of any arbitrary computer language, so long as that language has an associated BNF (Backus-Naur Form) grammar [Ryan, 1998]. Our system, *Grammatically Evolved Hashing* (GEVOSH), uses the method of Grammatical Evolution to generate hashing functions. The GEVOSH system produces hash functions by using a diminutive C++ grammar as its input.

1.1 GRAMMATICAL EVOLUTION

The precursors of Grammatical Evolution are Genetic Algorithms (GA) and Genetic Programming (GP). Genetic Algorithms is an area of artificial intelligence that can be described as “a search algorithm based on the mechanics of natural selection and natural genetics” [Stanford]. Genetic Programming is “the method of creating computer [Hyper Dictionary]. Grammatical Evolution (GE) can be defined as a grammar based genetic algorithm, the purpose of which is to generate executable programs. GE is clearly similar to GP. GE, however, has the distinction that its input is a BNF grammar which allows GE’s to generate programs in any language. The use of BNF also allows GE systems to more closely model real world DNA [Ryan, 1998] [Brabazon, 2002] [O'Neill, 2002]. Backus Naur Form or BNF is a context free meta-language. In particular, it is "a formal metasyntax used to express context-free grammars" [Unicode]. Thus, it is the notation used to specify programming languages.

1.2 HASHING

Hashing is a method of mapping a large range of keys into a smaller range of indices for compact storage and quick retrieval of information. A *collision* is the mapping of two or more keys to the same index; a *seek* is an attempt to locate an item when a collision has occurred; *linear probing* is the attempt to store a key sequentially after a collision has occurred [Flajolet, 1998]. Thus, the desirable hash function is one that generates few (if any) collisions, and provides the ability to find information without exhaustively searching a table of records.

2 The GEVOSH System

The GEVOSH system is designed to generate static hashing functions, and depicts each member as a unique hash function. The fitness of a member is determined by the percentage of keys hashed without collision. The fitness grade is used to rank population members, and ultimately determine which members of the population will survive through the next generation. GEVOSH is comprised of four modules, the interaction of which is shown below in Figure 1.

The *Load Grammar Module* opens a file specified by the user and reads in the BNF.

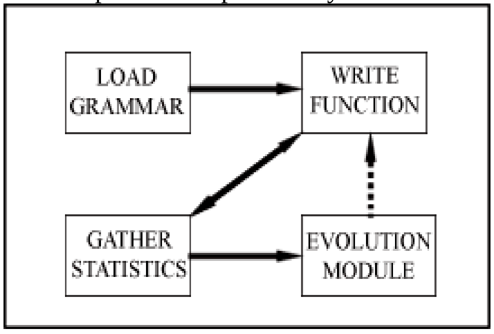


Figure 1: GEVOSH System Diagram

The *Write Function Module* decodes a member of the population to decide which grammar rules to follow, and ultimately, what code to write out for the function. The main activity involves the system resolving an identifier by following the BNF rule number that is given by taking the integer at position i modulo the number of rules for the given identifier. An example of this resolution can be seen below in *Figure 2.* The *Gather Statistics Module* ascertains the merit of the newly created hash function by using and measuring the hash function in a prewritten secondary program.

Member:	**19**	**26**	**13**	**39**	**48**	**6**

	Value	Number of Rules	Rule to Follow
<Statement-Line>	19	6	1
<var> = <expr> ;		1	0
value = <expr> ;	26	4	2
value = <expr> <op> <expr>;	13	4	1
value = <var> <op> <expr> ;		1	0
value = value <op> <expr> ;	39	7	4
value = value % <expr> ;	48	4	0
value = value % <num> ;	8	11	8
value = value % 7 ;			

Figure 2: Resolution of <statement-line>

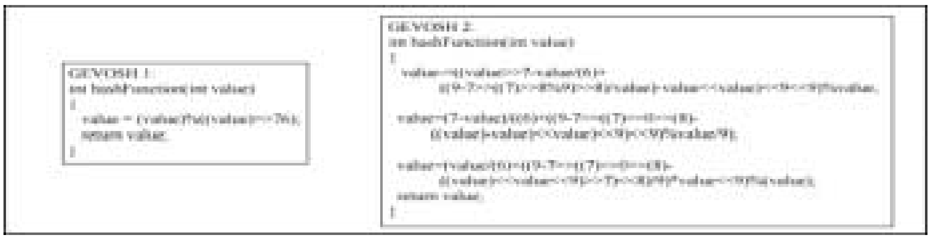
Lastly, the *Evolution Module* assigns fitness values to the members of the population. It then augments the population occur by spawning the next generation of hash functions through the use of genetic operators as shown below in Figure 3.

```
Begin GEVOSH-Algorithm()
  Load Grammar
  Randomly Generate N members
  Do Until fitness equals 100 or forced to quit
    For each member of the population
        Write function
        Compile secondary program
        Run secondary program
        Input statistics
    End For
    Compute fitness value
    Evolve population
  End Do
End GEVOSH
```

Figure 3: Main Algorithm of GEVOSH

3 Results

We are pleased to report that our system generated two hashing functions that show extremely promising potential as shown in Figure 4. That is, on average, these two functions in some ways are superior to six commonly used hashing functions that we extracted from the literature [Wang, 2002]. These hash functions produce collisions that are close in value with each other. Our testing involves comparing the number of seeks and collisions of our functions to these standard hash functions. As of now we have not simplified GEVOSH 2, but we have simplified GEVOSH 1 and found it to be a very interesting function. The number of shifts is actually 12 rather than 76 (which is far greater than the number of bits that are used to represent an integer); the function is quite successful in mapping and retrieving keys, Even though GEVOSH 1 out performs all other hash functions on average, it can be seen from below on the graphs that the function is not stable. That is, the generated function seems to be dependent on the data. The six other functions are not dependent on the data. The standard deviation of GEVOSH 1 is 0.186 where the standard deviation of the six testing functions are all approximately 0.01. Although GEVOSH 2's average percent hit rate without collisions is much lower than all of the other functions, its standard deviation is approximately 0.02, which implies that GEVOSH 2 is less dependent on the data. In general, we find these results to be promising, since the GEVOSH system can produce both functions with high averages and low standard deviations as shown below.



```
GEVOSH 1:
int hashFunction(int value)
{
  value = [illegible]
  return value;
}
```

```
GEVOSH 2:
int hashFunction(int value)
{
  value=[illegible]
        [illegible]

  value=[illegible]
        [illegible]

  value=[illegible]
        [illegible]
  return value;
}
```

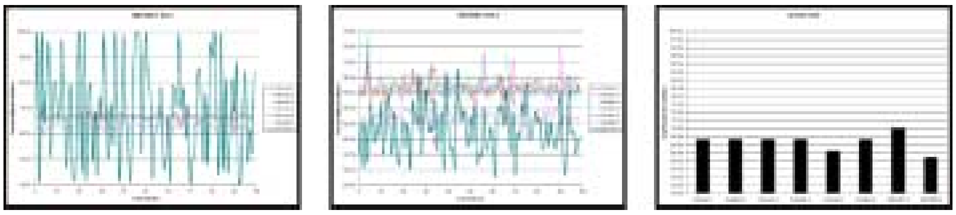


Figure 4: GEVOSH System Created Hash Function and Results

In summary, we believe that the GEVOSH system is exhibiting promising results and, as in most evolutionary computation systems, we are continuing to run GEVOSH through days and days of execution to evolve even stronger hash functions.

References

[Ryan, 1998] Ryan C., Collins J.J., O'Neill M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language; EuroGP 1998.

[Brabazon, 2002] Brabazon A., Matthews R., O'Neill M., Ryan C. Grammatical Evolution and Corporate Failure Prediction; GECCO 2002.

[Corman, 1990] T. Cormen, C. Leiserson, R. Rivest. *Introduction to Algorithms;* The MIT Press, 1990.

[Nicolau, 2002] Nicolau M., Ryan C. LINKGAUGE: Tackling hard deceptive problems with a new linkage learning genetic algorithm; GECCO 2002.

[O'Neill, 2002] O'Neill M., Ryan C. Investigations into Memory in Grammatical Evolution; GECCO 2002.

[Fogel, 2000] Fogel D. Evolutionary Computation *Toward a New Philosophy of Machine Intelligence;* IEEE Inc.

[Flajolet, 1998] Flajolet P. On the Analysis of Linear Probing Hashing; France Algorithms Seminar 1998.

[Wang, 2002] Wang T. Integer Hash Functions; http://www.concentric.net/~Ttwang/tech/inthash.htm.

[Stanford] www.stanford.edu/~buc/SPHINcsX/bkhm15.htm.

[Hyper Dictionary]http://www.hyperdictionary.com/computing/genetic+programming.

[Unicode] http://www.unicode.org/glossary/.