

**Binary Representation in Gene Expression  
Programming: Towards a Better Scalability**

**Jose G. Moreno-Torres, Xavier Llorà  
and David E. Goldberg  
IlliGAL Report No. 2009003**

**January, 2009**

Illinois Genetic Algorithms Laboratory  
University of Illinois at Urbana-Champaign  
117 Transportation Building  
104 S. Mathews Avenue  
Urbana, IL 61801  
Office: (217) 333-2346  
Fax: (217) 244-5705

# Binary Representation in Gene Expression Programming: Towards a Better Scalability

Jose G. Moreno-Torres, Xavier Llorà and David E. Goldberg  
Illinois Genetic Algorithms Laboratory (IlliGAL)  
University of Illinois at Urbana-Champaign  
104 S. Mathews Ave, Urbana, IL 61801  
(josegmt, xllora, deg)@uiuc.edu

6 January 2009

## Abstract

One of the main problems that arises when using gene expression programming conditions in learning classifier systems is the increasing number of symbols present as the problem size grows. This issue severely limits the scalability of the technique, due to model building becoming untractable. We propose a binary representation of GEP chromosomes as a possible solution. We provide a theoretical reasoning behind our representation, and back it up with empirical results.

## 1 Introduction

There have been interesting results when attempting to use Gene Expression Programming (GEP) conditions in learning classifier systems. It has also been observed that there is a scalability limitation when doing so, which severely limits its usefulness. We propose a possible solution to this problem by using a different representation of the GEP chromosome.

We begin by providing some background about our problem. We continue by presenting our solution, followed by a theoretical analysis of the advantages of said solution. After that, we will construct a set of experiments to test our representation and confirm the hypothesis we have devised. Lastly, we will analyze the results from our experiments, and discuss their meaning and relevance.

## 2 Background

In 1998, Ryan et al. published the seminal paper on Grammatical Evolution (Ryan, Collins, & Neill, 1998). It introduced the idea of a separation between genome and phenotype, where the phenotype results from the translation of an underlying genome, a linear chromosome, which is the object of selection and genetic operators. It works with a user-defined BNF grammar where the genome encodes the choices between the different production rules in the grammar. Ferreira's Gene Expression Programming (Ferreira, 2001) extends this idea. In GEP, the information encoded in the genome is an expression tree similar to the ones used in Genetic Programming (Koza, 1992).

In 2007, Wilson (Wilson, 2007) suggested using GEP conditions in learning classifier systems (specifically, XCSF), successfully learning regularities in small problems. He later presented some

extended results, where he included the use of constants. It is remarkable that he used Ephemeral Random Constants (Koza, 1992) instead of the method proposed by Ferreira. In our analysis we will follow the same approach, so we will treat variables and constants the same way; distinguishing them from functions.

In the same year, Llorà et al (Llorà, Sastry, Lima, Lobo, & Goldberg, 2007) showed there is a scalability problem when attempting to perform model building in LCSs using GEP conditions. Basically, the problem comes down to the fact that the population size grows very quickly as a function of the problem size. This issue is one of the big obstacles for the use of GEP, and will be tackled here. We will approach it by representing the GEP chromosome as a binary string, consequently fixing the arity of the representation.

### 3 Design of the binary representation

We approach now the task of designing a new representation for GEP chromosomes. Our binary representation will provide

1. Arity  $\chi = 2$ .
2. The number of bits per symbol grows logarithmically with the problem size.
3. The Hamming distance between any two functions representation is bounded:

$$d(f_1, f_2) \leq \log_2(nFunctions), \forall f_1, f_2 \in F$$

The basic idea for the representation is to have a first bit to distinguish between functions and variables/constants. After that, there will be enough bits to represent variables and constants by their binary representation. Functions will work the same way, with the extra bits being filled at random. It would certainly have been possible to have each function represented by one bit (ie, the 2nd function would be encoded as 0100, the 3rd as 0010), but that would have made the existence of illegal strings possible, and the gain would be marginal.

It should be noted that a mechanism to deal with strings that do not represent a valid function or terminal could be necessary, for the cases where their numbers are not exact powers of 2.

Lastly, the number of bits per symbol can be calculated as

$$bPS = 1 + \max(\lceil \log_2(nFunctions) \rceil, \lceil \log_2(nConstants + nVariables) \rceil). \quad (1)$$

We now present an example for clarification. Let us have a problem with 30 variables, and let us assume we choose to limit the number of constants to 10, and to have 7 different possible functions. Substituting in (1),

$$\begin{aligned} bPS &= 1 + \max(\lceil \log_2(7) \rceil, \lceil \log_2(30 + 10) \rceil) \\ &= 1 + \max(3, 6) = 1 + 6 = 7. \end{aligned}$$

The representations of some different possible symbols are:

- 3rd function = 0011####. (0111#### would represent a non-valid function; we need a way to deal with these cases. A dummy function could be a viable alternative.)

- 23rd variable = 1010111.
- 4th constant = 1100010. (The 4th constant is the equivalent to the  $nVariables + 4$  variable, so it is the 34th variable in our example).

## 4 Theoretical analysis

We will now analyze the expected performance of our new representation as opposed to the commonly used  $\chi$ -ary one, focusing on population size needed to perform model building. We start by presenting a narrow example where the effectiveness of the new representation is high, and then generalize the analysis.

Goldberg (Goldberg, 2002) gives a theoretical bound for population size:

$$s = O(m \log(m) \cdot x^k), \quad (2)$$

where  $s$  is the population size,  $m$  is the number of building blocks,  $x$  is the arity of the alphabet in our representation and  $k$  is the size of the building block.

It is easy to see that, for the  $\chi$ -ary representation scheme, when working with the chromosome  $x_i \& x_i$  (where  $x_i$  could be any variable), we have  $m = 2$ ,  $x = N$ ,  $k = 2$  (& and  $x_i x_i$  are the building blocks). From these values, we can substitute in (2) and we obtain

$$s = O(N^2), \quad (3)$$

so the population size grows quadratically with the problem size ( $N = \text{dimensionality of the problem}$ ).

In our binary representation, however, the population size grows much more slowly. The reason for this is that the arity is fixed ( $\chi = 2$ ); as is the maximum size of a building block ( $k = 2$ ), due to each of the first variable bit's value depending only on the corresponding bit of the second variable; leaving only the number of building blocks to grow. Since this growth is logarithmic ( $m = \log_2(N)$ ), we have

$$s = O(\log(N) \log(\log(N))). \quad (4)$$

In a more general case, a building block of size two gives us a population size of  $s = O(N^2)$  for the  $\chi$ -ary representation (this is equation (3) again, since the analysis presented holds for the general case). With a binary representation, the building block size depends on the Hamming distance between the symbols we are learning:  $k = 2(d + 1)$ , where  $d$  is the Hamming distance. Substituting in our population size formula (2), we get

$$s = O(\log(N) \log(\log(N)) * 2^{2(d+1)}). \quad (5)$$

Since we have an upper bound for the Hamming distance ( $d = \log_2(N)$ ), we can solve (5) to get

$$\begin{aligned} s &= O(\log(N) \log(\log(N)) * 2^{2 \log(N)}) \\ &= O(\log(N) \log(\log(N)) * N^2). \end{aligned} \quad (6)$$

Our representation is designed to take advantage of the difficulty being in the Hamming distance. Since we chose to separate functions and terminals, we can bound the maximum Hamming distance when working with functions to  $d \leq \log_2(nFunctions)$ ; and since  $nFunctions$  is constant, we can solve (5) to get a new theoretical prediction for the binary representation:

$$s = O(\log(N) \log(\log(N))). \quad (7)$$

We have been able to replicate equation (4) when working with functions in general. Since functions usually make up the most interesting building blocks, this is an improvement compared to  $\chi$ -ary representations, even if we do not improve the cases where we use variables.

## 5 Experiment design

We want to study how the population size required for model building grows as a function of the problem size. We will test it using ECGA’s (Harik, Lobo, & Sastry, 2006) model building algorithm, probing the population size needed to extract the correct building blocks as the problem size increases. In order to consider a population size to be successful, it has to extract either all the correct building blocks or all of them but one in 19 out of 20 runs. To perform the testing, we artificially build a population of the desired population size to apply the model building to.

We will perform three independent experiments. In the first one, we try and confirm our initial intuition, working with the basic case  $x_i \& x_i$ . The second one is a more general case, in which we seek confirmation for the hypothesis stating that the Hamming distance is the real measure of difficulty when trying to model a building block composed by two symbols, doing it with variables:  $x \parallel y$ . For this experiment, in the binary case, we will do a worst-case study, by fixing  $x$  and  $y$  to be as far away from each other as possible, in terms of Hamming distance.

Our last experiment will probe the effectiveness of the designed representation when dealing with functions:  $\parallel \& !$ . In this experiment, we ignore the terminals that would need to be present in order to have a legal expression (it would be  $(x_i \parallel x_j) \& x_k$ ), and focus only on the BB formed by the functions.

If our analysis is correct, we expect to have equivalent results for all three tests when using a  $\chi$ -ary representation, following equation (3); while the binary one should follow equation (4) for the first and third experiment, and (6) for the second one. We present our results in the next section.

## 6 Results

We will now present the results obtained by performing the experiments. In this section we will focus on each experiment individually, leaving the more general conclusions for the discussion part.

Figure 1 shows the results for the first experiment, where the binary representation clearly outclasses the  $\chi$ -ary one. This result supports our theoretical analysis, since our experimental values follow very closely those predicted by equations (3) and (4). As we expected, the population size grows quadratically in the  $\chi$ -ary representation. For the binary one, the theory says it should grow according to  $s = O(\log(N) \log \log(N))$ ; but our empirical results show a constant value. This is probably just due to the problem being too easy. However, it is remarkable that our representation is much better equipped to take advantage of the problem’s simplicity than the  $\chi$ -ary one.

In figure 2, we present the data from a more general case, working with two arbitrary variables. In the binary case, the population size represents the square of the Hamming distance (ie, a population size of 128 actually means  $hDistance = \log_2(128) = 7$ ). For this example, both the  $\chi$ -ary and binary representation give a very similar performance. Again, they support the theory presented in equations (3) and (6), so there is a slight advantage for the  $\chi$ -ary representation.

Figure 3 corresponds to the third experiment, where we do linkage learning on functions. The tests, once again, accurately match the theoretical values that were predicted.

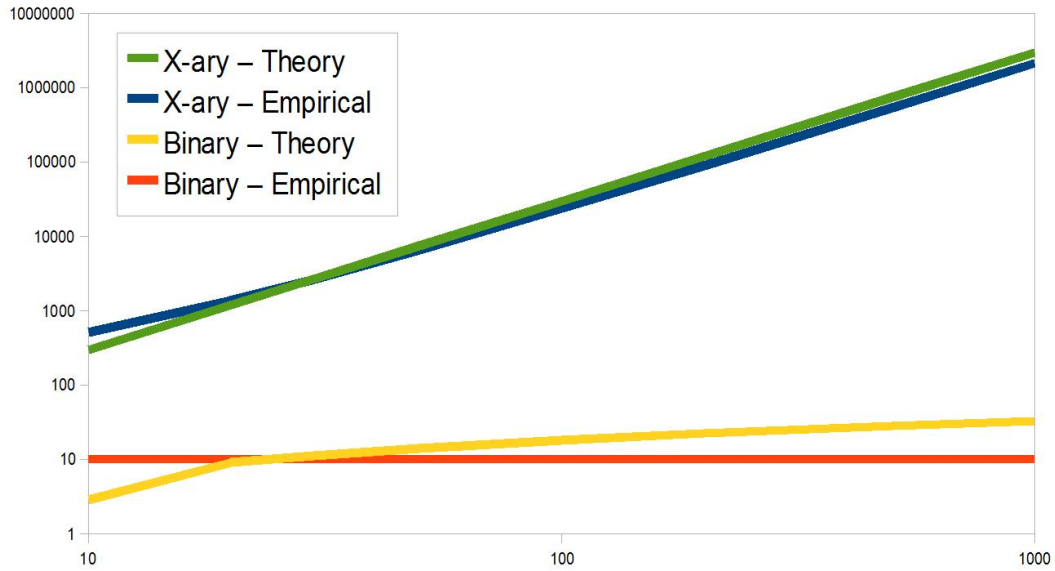


Figure 1: Population size as a function of problem size for binary vs  $\chi$ -ary representations when learning  $x_i$  &  $x_i$

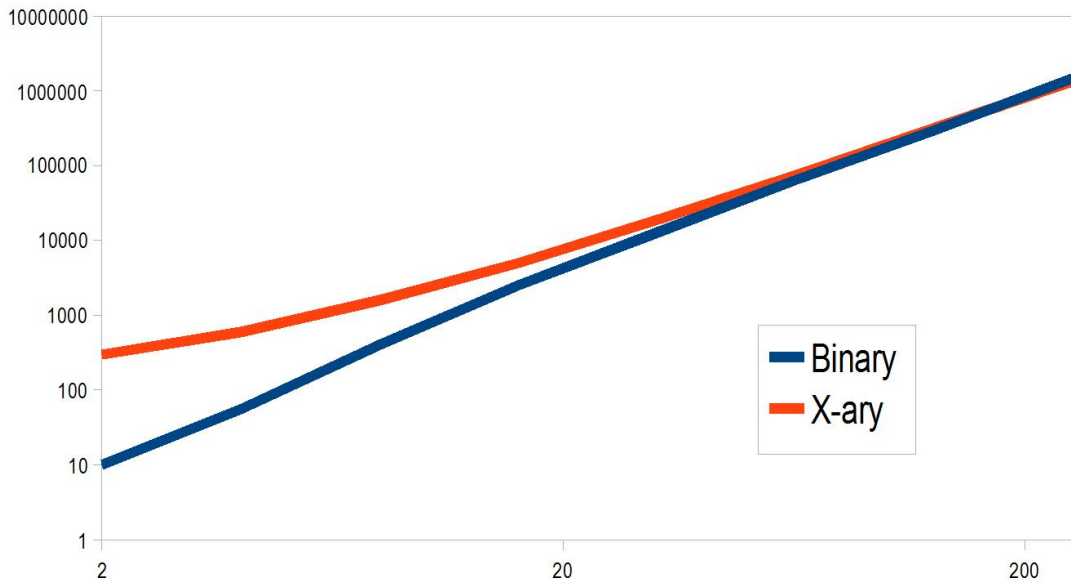


Figure 2: Population size as a function of problem size for binary vs  $\chi$ -ary representations when learning  $x || y$ . Empirical results only.

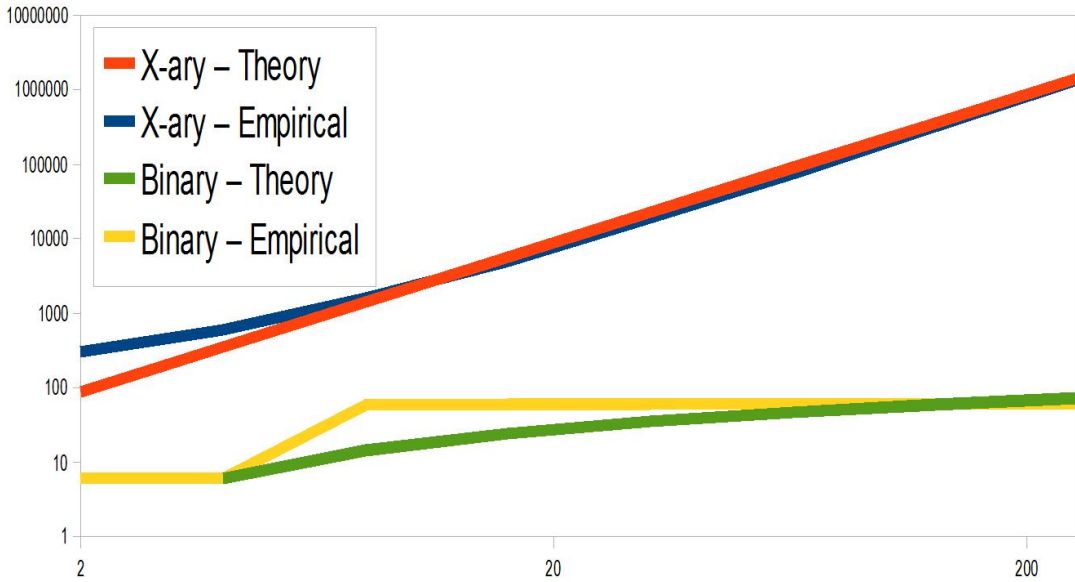


Figure 3: Population size as a function of problem size for binary vs  $\chi$ -ary representations when learning  $\|&!&$

## 7 Discussion and Conclusions

We have suggested the use of a binary representation to improve the scalability of GEP conditions in LCSs in terms of the number of variables in a problem. This is a new idea, and the first results we present are encouraging enough to warrant further study.

We have designed a new representation for GEP that, in theory, can outperform the commonly used  $\chi$ -ary representation when working with functions, while being comparable when modeling relationships between variables. The experiments performed strongly suggest that the theoretical analysis is indeed correct and that, for the cases studied, the proposed representation does provide a better scalability in terms of problem size when working with functions.

There are two positive conclusions here. The first is the improvement in performance, as it is the first step towards solving the scalability limitations to the effective use of GEP conditions in learning classifier systems when doing model building. The second is the accuracy of the theoretical analysis. The empirical data matches remarkably closely the one predicted by the equations derived from theory, which gives us confidence that we can develop solutions based on this theory.

However, there are still a lot of open questions. To begin with, a deeper analysis on the effect changing the representation has in the performance of gene expression programming is required. As a starting point, we will compare the  $\chi$ -ary and binary representations when dealing with the different genetic operators present in GEP, such as mutation, inversion, transposition and crossover.

Also, it is important to mention that size is only part of the difficulty of the problem. The complexity of the conditions necessary to accurately represent problem regularities gets reflected in the building block size, which by equation (2) induces exponential growth in the population size. This will be addressed in future research.

Lastly, it would be interesting to study the possibility of using this binary representation for genetic programming. As it was hinted previously, there is a strong parallelism between the way GEP and GP encode expression trees, more specifically the way the arity grows with the number

of variables in the problem; so the analysis performed here should also be relevant for GP. An empirical confirmation of this fact will be performed in future work.

## Acknowledgements

The authors would like to acknowledge the technical support provided by Thyago Duque. They would also like to thank Obra Social "la Caixa" for its financial support.

## References

- Ferreira, C. (2001). Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13(2), 87–129.
- Ferreira, C. (2006). *Gene expression programming: Mathematical modeling by an artificial intelligence*. Germany: Springer-Verlag.
- Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.
- Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Boston, MA: Kluwer Academic publishers.
- Harik, G. R., Lobo, F. G., & Sastry, K. (2006). Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga). *Scalable Optimization via Probabilistic Modeling*, 39–61.
- Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: The MIT Press.
- Llorà, X., Sastry, K., Lima, C. F., Lobo, F. G., & Goldberg, D. E. (2007). Linkage learning, rule representation, and the x-ary extended compact classifier system. In *IWLCS* pp. 189–205.
- Ryan, C., Collins, J., & Neill, M. O. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming* pp. 83–95. Springer-Verlag.
- Sastry, K., & Goldberg, D. E. (2003). Probabilistic model building and competent genetic programming. In *Genetic Programming Theory and Practise, chapter 13* pp. 205–220. Kluwer Academic Publishers.
- Wilson, S. W. (2007). Classifier conditions using gene expression programming. In *IWLCS* pp. 206–217.