2009 Ninth International Conference on Intelligent Systems Design and Applications

Binary Representation in Gene Expression Programming: Towards a Better Scalability

Jose G. Moreno-Torres*, Xavier Llorà† and David E. Goldberg*
*Illinois Genetic Algorithms Lab (IlliGAL)
University of Illinois at Urbana-Champaign
104 S. Mathews Ave, Urbana, IL 61801, USA
(josegmt, deg)@illinois.edu
†National Center of Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign
1205 W. Clark Street, Urbana, IL 61801, USA
xllora@illinois.edu

***Abstract*—One of the main problems that arises when using gene expression programming (GEP) conditions in learning classifier systems is the increasing number of symbols present as the problem size grows. When doing model-building LCS, this issue limits the scalability of such a technique, due to the cost required. This paper proposes a binary representation of GEP chromosomes to palliate the computation requirements needed. A theoretical reasoning behind the proposed representation is provided, along with empirical validation.**

I. Introduction

There have been interesting results when attempting to use GEP-based[1] conditions in learning classifier systems [2]. Despite its flexibility, when used in model-building LCS, there is a scalability limitation [3]. This limitation comes from the arity of the individuals growing linearly with the problem size. Such a growth usually leads to the population sizes requiring an exponential growth. This paper presents a possible solution to this problem by using a different representation of the GEP chromosome.

We begin presenting an alternative representation, followed by a theoretical analysis of the advantages. Then, we construct a set of experiments to test the proposed representation and to confirm its merits. Finally, we analyze the results from our experiments, and discuss their meaning and relevance.

II. Background

In 1998, Ryan et al. published the seminal paper on Grammatical Evolution [4]. 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[5] extends this idea. In GEP, the information encoded in the genome is an expression tree similar to the ones used in Genetic Programming [6].

In 2007, Wilson [2] suggested using GEP conditions in Michigan LCSs (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 [6] instead of the method proposed by Ferreira. This paper follows the same approach, treating variables and constants the same way and distinguishing them from functions.

In the same year, Llorà et al [3] showed a scalability problem when attempting to perform model-building in LCSs using GEP conditions. In this work, a Pittsburgh learner was presented; but the same issue appears in Michigan-style models. The limitation showed is due to the population size needed to perform model-building growing very quickly as a function of the problem size. This issue is one of the big obstacles for the use of GEP conditions in model-building LCSs, and is tackled here. This paper proposes a representation of the GEP chromosome as a binary string, consequently fixing the arity of the representation.

III. Design of the binary representation

The proposed representation provides:

1) A fixed arity $\chi = 2$.
2) A logarithmic increase of the population size encoding requirements respect the number of symbols.
3) A Hamming distance bound between any two functions' representation: $d(f_1, f_2) <= log_2(nFunctions)$

This representation uses its first bit to distinguish between functions and variables/constants. If the bit string starts with a '1', this indicates a variable/constant, and the following bits encode what variable or constant it is (constants follow immediately after the last variable in the problem, as is shown in the example). Otherwise, it is a function, and again the following bits determine which one it is. It should be

978-0-7695-3872-3/09 $26.00 © 2009 IEEE
DOI 10.1109/ISDA.2009.33

noted that a mechanism to deal with strings that do not represent a valid function or terminal is necessary, for the cases where their numbers are not exact powers of 2. In this paper, no-ops were used for those cases. Non-coding bits are ignored.

For the proposed representation, the number of bits needed to encode a symbol can be calculated as

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

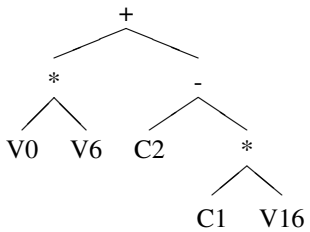
For instance, let us assume a problem with $nVariables = 30$ variables, and let the maximum number of constants $nConstants = 10$, and to have $nFunctions = 6$ different possible functions. Substituting in Equation 1,

$$bPS = 1 + \max(\lceil \log_2(7) \rceil, \lceil \log_2(30 + 10) \rceil) = 1 + \max(3, 6) = 1 + 6 = 7. \quad (2)$$

The representations of some different possible symbols in this example are:

- 3rd function = 0010XXX (X represents a non-coding bit).
- 23rd variable = 1010110.
- 4th constant = 1100001. (The 4th constant is the equivalent to the $nVariables + 4$ variable, so it is the 34th variable).

Here is an example of a full tree's representation, assuming the same setup presented above, and assuming the order of the functions is $(+, -, *, /, sin, cos)$. The tree to encode is the following:



Where V_i corresponds to the ith variable, and C_i corresponds to the ith constant. The encoding for this tree, in depth-first order, is:

0000XXX 0010XXX 1000000 1000110 0001XXX 1011111 0010XXX 1011110 1001111.

IV. Theoretical analysis

This section analyzes the expected performance of the proposed representation as opposed to the commonly used χ-ary one, focusing on the population size needed to perform Building Block (BB) signaling through model-building. It starts with a narrow example where the effectiveness of the new representation is high, and then presents a more general analysis.

Goldberg [7] gives a theoretical bound for the population size needed for BB signal:

$$s = O(m \log(m) \cdot \chi^k), \quad (3)$$

where s is the population size, m is the number of building blocks, χ 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 χ-ary representation scheme, when working with the chromosome $x_i \quad and \quad x_i$ (where x_i could be any variable), we have $m = 2$, $\chi = N$ (where N = dimensionality of the problem, in the worst case $nVariables + nConstants + nFunctions$), $k = 2$ (and and $x_i x_i$ are the building blocks). Substituting these values in Equation 3, we obtain obtain

$$s = O(N^2), \quad (4)$$

so the population size grows quadratically with the problem size.

For the proposed 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 bits' 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 ($m = \log_2(N)$), the population size follows

$$s = O(\log(N) \log(\log(N))). \quad (5)$$

In a more general case, a building block of size two means a population size of $s = O(N^2)$ for the χ-ary representation (this is Equation 4 again, since the analysis presented earlier holds for the general case). With a binary representation, the building block size depends on the Hamming distance between the symbols present in the example: $k = 2(d + 1)$, where d is the Hamming distance. Substituting in the population size formula presented above (Equation 3), we get

$$s = O(\log(N) \log(\log(N)) * 2^{2(d+1)}). \quad (6)$$

Since there is an upper bound for the Hamming distance ($d = \log_2(N)$), we can solve Equation 6 to get

$$s = O(\log(N) \log(\log(N)) * 2^{2 \log(N)}) = O(\log(N) \log(\log(N)) * N^2). \quad (7)$$

Since we chose to separate functions and terminals, we can bound the maximum Hamming distance when working with functions to $d <= \log_2(nFunctions)$; and since $nFunctions$ is constant, we can solve (6) to get a new theoretical prediction for the binary representation:

$$s = O(\log(N) \log(\log(N))). \quad (8)$$

We have been able to replicate Equation 5 when working with functions in general. Since functions usually make up

the most interesting building blocks, this is an improvement compared to χ-ary representations, even if we do not improve the cases where we use variables.

V. Experiment design

The aim of the paper was to study how the population size required for model building grows as a function of the problem size. The tests were run using ECGA's model-building algorithm [8], 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, an artificial population of the desired size was built to apply the model building to.

We performed three independent experiments. The first one tests the same narrow case the theoretical analysis started on, working with the specific case $(x_i \quad and \quad x_i)$, for a given i. The second experiment seeks 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: $(x_i \quad and \quad x_j)$, with $i \neq j$. For this experiment, in the binary case, we did a worst-case study, by fixing x_i and x_j to be as far away from each other as possible, in terms of Hamming distance (for example, a population size of 128 actually means $hDistance = \log_2(128) = 7$).

The last experiment was designed to probe the effectiveness of the designed representation when dealing with functions only. In this experiment, we ignored the terminals that would need to be present in order to have a legal expression, and focused only on the BB formed by the functions. What this means is that, for an expression like $((x_i \quad or \quad x_j) \quad and \quad not \quad x_k)$, this experiment would only focus on the functions $(or)and(not)$, ignoring the variables.

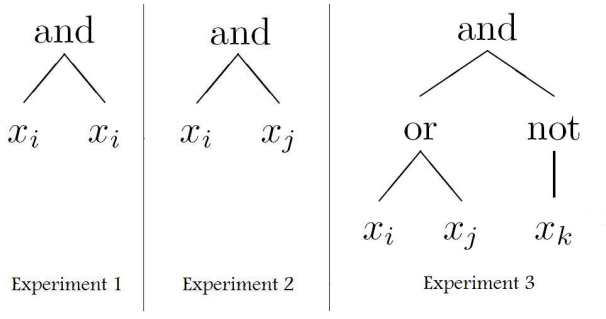


Figure 1. Tree representation of the expressions learned on each of the three experiments. For experiment 3, the whole tree is represented even though the terminals are ignored in the experiment.

According to the theoretical analysis presented previously, equivalent results for all three tests are to be expected when using a χ-ary representation, following Equation 4; while using the binary one should provide results following Equation 7 for the second experiment; and Equation 5 for the first and third ones, showing a significant improvement over the χ-ary representation in these two.

VI. Results

This section presents the results obtained by performing the experiments detailed above. Figure 2 presents the results from the first experiment, where the binary representation clearly outclasses the χ-ary one. This result supports the theoretical analysis presented, since the experimental values obtained follow very closely those predicted by Equations 4 and 5. As it was to be expected, the population size grows quadratically in the χ-ary representation. For the binary one, the theory says it should grow according to $s = O(\log(N)\log\log(N))$; but the empirical results show a constant value. This is probably just due to the problem being too easy. However, it is remarkable that the proposed representation is much better equipped to take advantage of the problem's simplicity than the χ-ary one.

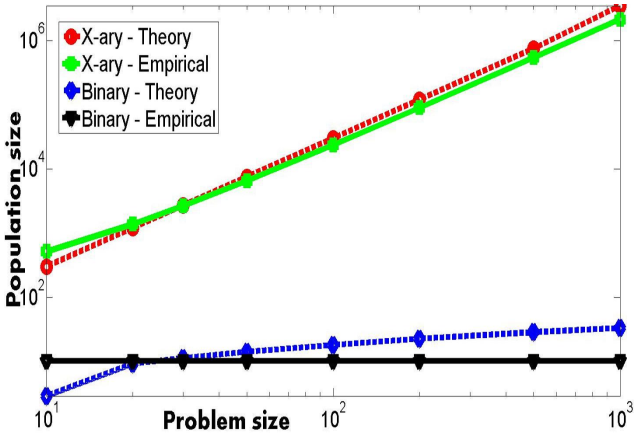


Figure 2. Population size as a function of problem size for binary vs χ-ary representations when learning $(x_i \quad and \quad x_i)$

In Figure 3, we present the data from a more general case, working with two arbitrary variables. In the binary case, the population size represents 2^d, where d is the Hamming distance (for example, a population size of 128 actually means $hDistance = \log_2(128) = 7$). For this experiment, the χ-ary and binary representation result in similar performances. Again, they support the theory presented in Equations 4 and 7, so there is a slight advantage for the χ-ary representation.

Figure 4 corresponds to the third experiment, where we do model-building on building blocks composed exclusively of functions. Results, once again, accurately match the theoretical model proposed. This a key result of this study, since it shows a real improvement in the population size needed for model building when dealing with relationships between functions. As can be seen in the figure, the quadratic

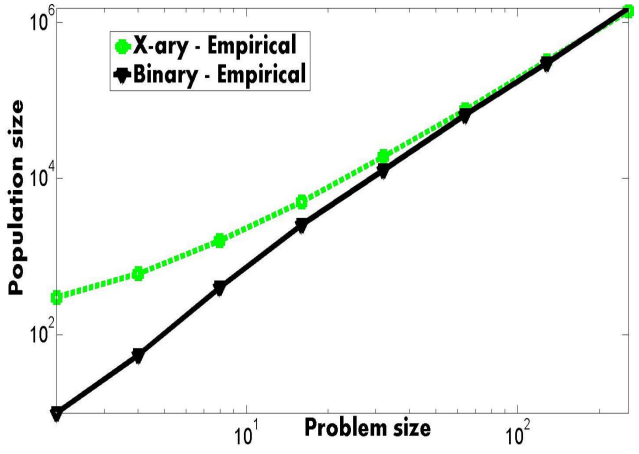


Figure 3. Population size as a function of problem size for binary vs χ-ary representations when learning $(x \quad and \quad y)$. Empirical results only.

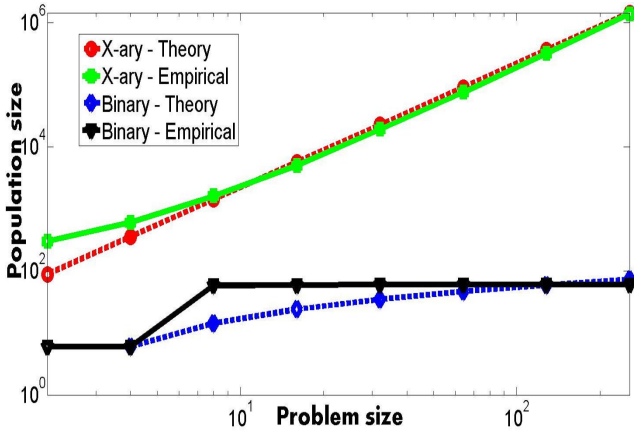


Figure 4. Population size as a function of problem size for binary vs χ-ary representations when learning $(or)and(not)$

growth present when using a χ-ary representation is reduced to a logarithmic growth for the binary representation.

VII. Discussion and Conclusions

This paper proposes a binary representation to improve the scalability of GEP conditions in model-building LCSs in terms of the number of variables in a problem. This is a new idea, and the presented results are encouraging enough to warrant further study.

We have designed a new representation for LCS using GEP that, in theory, can outperform the commonly used χ-ary representation when working with functions. However, the binary representation has a slightly worse performance than the χ-ary one when dealing with relationships between variables/constants.

The performance improvement is the first step towards solving the scalability limitations to effectively using GEP conditions model-building LCS. 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.

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 3 induces exponential growth in the population size. This issue will be addressed in future research.

Acknowledgments

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. They are also grateful for NCSA's support for this research.

References

[1] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. Germany: Springer-Verlag, 2006.

[2] S. W. Wilson, "Classifier conditions using gene expression programming," in *IWLCS*, 2007, pp. 206–217.

[3] X. Llorà, K. Sastry, C. F. Lima, F. G. Lobo, and D. E. Goldberg, "Linkage learning, rule representation, and the x-ary extended compact classifier system," in *IWLCS*, 2007, pp. 189–205.

[4] C. Ryan, J. Collins, and M. O. Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *Lecture Notes in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming*. Springer-Verlag, 1998, pp. 83–95.

[5] C. Ferreira, "Gene expression programming: a new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0102027

[6] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press, 1992.

[7] D. E. Goldberg, *The Design of Innovation: Lessons from and for competent genetic algorithms*. Boston, MA: Kluwer Academic publishers, 2002.

[8] G. R. Harik, F. G. Lobo, and K. Sastry, "Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga)," *Scalable Optimization via Probabilistic Modeling*, pp. 39–61, 2006.

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[10] K. Sastry and D. E. Goldberg, "Probabilistic model building and competent genetic programming," in *Genetic Programming Theory and Practise, chapter 13*. Kluwer Academic Publishers, 2003, pp. 205–220.