RESEARCH-ARTICLE

An efficient distance metric for linear genetic programming

MARCO GAUDESI, Polytechnic of Turin, Turin, TO, Italy

GIOVANNI SQUILLERO, Polytechnic of Turin, Turin, TO, Italy

ALBERTO PAOLO TONDA, National Research Institute for Agriculture, Food and Environment, Paris, Ile-de-France, France

Open Access Support provided by:

Polytechnic of Turin

National Research Institute for Agriculture, Food and Environment

Published: 06 July 2013

Citation in BibTeX format

GECCO '13: Genetic and Evolutionary Computation Conference
July 6 - 10, 2013
Amsterdam, The Netherlands

Conference Sponsors:
SIGEVO

An Efficient Distance Metric for Linear Genetic Programming

Marco Gaudesi
Politecnico di Torino
Corso Duca degli Abruzzi, 24
10129 Torino, Italy
marco.gaudesi@polito.it

Giovanni Squillero
Politecnico di Torino
Corso Duca degli Abruzzi, 24
10129 Torino, Italy
giovanni.squillero@polito.it

Alberto Tonda
INRA UMR 782, MALICES
1 Avenue Lucien Brétignères
78850 Thiverval-Grignon, France
alberto.tonda@grignon.inra.fr

ABSTRACT

Defining a distance measure over the individuals in the population of an Evolutionary Algorithm can be exploited for several applications, ranging from diversity preservation to balancing exploration and exploitation. When individuals are encoded as strings of bits or sets of real values, computing the distance between any two can be a straightforward process; when individuals are represented as trees or linear graphs, however, quite often the user must resort to phenotype-level problem-specific distance metrics. This paper presents a generic genotype-level distance metric for Linear Genetic Programming: the information contained by an individual is represented as a set of symbols, using n-grams to capture significant recurring structures inside the genome. The difference in information between two individuals is evaluated resorting to a symmetric difference. Experimental evaluations show that the proposed metric has a strong correlation with phenotype-level problem-specific distance measures in two problems where individuals represent string of bits and Assembly-language programs, respectively.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Algorithms; Measurements; Linear Genetic Programming; Diversity Preservation; Distance Metric; Fitness Sharing; Experimental Analysis; Individual Encoding; Assembly; NK-Landscapes

1. INTRODUCTION

Defining a *distance metric* in an Evolutionary Algorithm (EA) is both theoretically sound and practically challenging – and ultimately useful. Being able to quantify the similarity of two individuals can be used to promote diversity inside the population's gene pool, to avoid the over-exploitation of niches in the fitness landscape, to balance exploration and exploitation, and ultimately to ease the premature convergence problem. Not surprisingly, the topic has been actively investigated by the evolutionary community for many years.

From the theoretical point of view, two different aspects must be examined when a distance is defined: the level at which it is calculated; and the purpose for calculating it. On the other hand, for the practitioner the complexity involved in the calculation is the key point.

The level at which a distance is defined may be: genotype, phenotype, or fitness. The first and the last are probably the most easily definable: the genotype corresponds to the internal representation of the candidate solution; the fitness is ultimately the number, or numbers, returned by its evaluation. In biology, the phenotype is the sum of all the observable characteristics of an organism that result from the interaction of its genotype with the environment. It is hard to translate the concept in Evolutionary Computation since the environment is missing, being indirectly defined by its effects through the fitness function. Yet, in several classical problems – where an individual is a fixed-length bit string, for instance – the need to distinguish between genotype and phenotype is reduced. As a consequence, several works assimilate the fitness to the phenotype.

In many other cases identifying phenotype and fitness is not an option. The fitness is a synthetic information, and may not be able to convey the necessary data to separate individuals. Even in the simplistic one-max problem two solutions may have the same fitness without sharing a single gene (e.g., "0011" and "1100"). Moreover, the very same solution can be encoded in different ways. If the individual is the movement of a robot, for instance, a single 90° turn could also be represented as two consecutive 45° ones. More generally, whenever the genotype cannot be evaluated directly by the fitness function, but needs to be transformed into something else, fitness and genotype should be distinguished. In such scenarios, the phenotype could be easily defined as the "something else" in which the genotype needs to be transformed into.

The final goal for measuring the distance between individuals plays an important role. If the distance metric is used to avoid that a region of the search space becomes overly populous, then it should be defined at the level of phenotype. However, phenotype-level distances are often difficult

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

to define or practically impossible to calculate. Remarkably, NSGA-II, the widely used multi-objective evolutionary optimizer, adopts a sharp and computationally efficient mechanism called *crowding distance* to scatter individuals [6]. Here, the crowding distance may rely exclusively on information from the fitness because the genotypes are fixed-length arrays of real numbers, requiring no transformation; and the fitness is composed of several different values, reducing the loss of information.

Conversely, if the distance metric is used to promote diversity in the gene pool, balancing exploration and exploitation, it could be based on the genotype. For example, in [13] solutions are encoded as fixed-length bit strings and a metric based on the hamming distance is used to assess the global diversity inside the population. When the phenotypes are sets of real values of fixed size, computing the distance between them is relatively straightforward, albeit not trivial [5]. However, phenotypes in Genetic Programming (GP) [11], Linear Genetic Programming (LGP) [3] and other complex EAs pose a harder challenge: calculating the similarity between two binary trees, linear graphs, or generic compound structures is an open problem.

This paper proposes a new distance metric easily usable in different types of LGPs. The distance is calculated quite efficiently at the level of genotype, yet it is able to convey a considerable amount of information about the individual. Thus, it may be used to reduce crowding in place of a phenotype-level distance. The proposed approach computes the *symmetric difference* [1] between the global information contained in two individuals; while the global information itself is evaluated resorting to the concept of n-grams [18].

Experimental results demonstrate that the proposed distance is highly correlated with other phenotype-level problem-specific distance measures, both where individuals are string of bits and Assembly language programs. Further experiments show that exploiting the proposed metric to perform fitness sharing in a sample problem produces results comparable to using a phenotype-level metric.

2. BACKGROUND

2.1 Linear Genetic Programming

LGP is a variant of GP that evolves computer programs as sequences of instructions. It was introduced by Markus Brameier and Wolfgang Banzhaf between the late 90s and the early 2000s [2] [3], after the seminal work of Friedberg [7]. A traditional, *Koza-style* GP encodes individuals as trees. Such tree GPs – or TGPs, as they are sometimes called – are commonly used to evolve mathematical expressions: leaves correspond to *terminals*, such as input values, constants or variables; inner nodes represent *functions*. Quite differently, LGP evolves a simplified list structure that represents a sequence of instructions in an imperative programming language. The resulting individuals represent real programs, although in a simplified language, that can grow to a significant complexity. Since their appearance, LGPs have been widely used to solve both practical problems and perform theoretical studies.

In LGP the difference between genotype and phenotype becomes fully apparent. The *genotype* is the internal, list-based representation; the *phenotype* is the actual program resulting from the interpretation of the genotype; the *fitness* is the final result of the evaluation of the program (Figure 1).

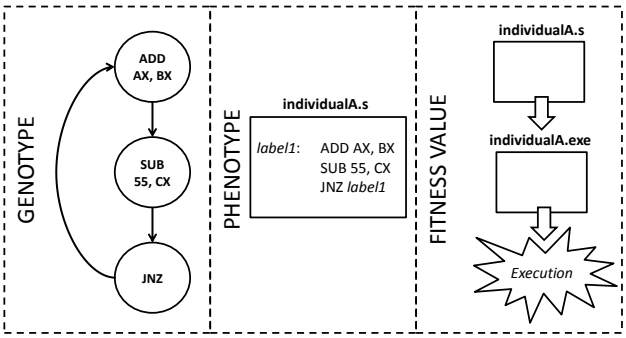

Figure 1: Distinction between genotype, phenotype and fitness value in an example with LGP used for Assembly language generation.

2.2 Symmetric Difference

In set theory, the symmetric difference [1] of two sets A and B is defined as

$$A \triangle B = A \cup B - A \cap B \tag{1}$$

In practice, the symmetric difference contains all elements which are in either of the sets and not in their intersection. The Venn diagram of the symmetric difference is reported in Figure 2.

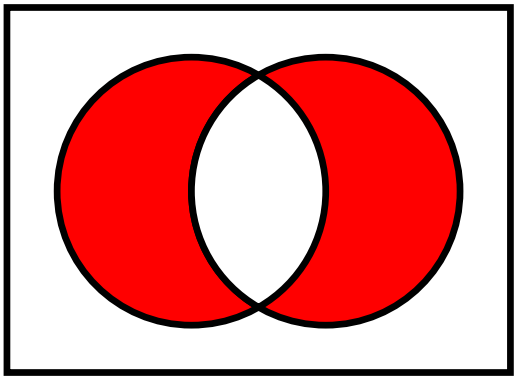

Figure 2: Venn diagram of the symmetric difference. The area corresponding to $A \triangle B$ is depicted in grey.

Considering the set as the *information* carried by an individual, the symmetric difference appears a plausible formalization of the intuitive idea of distance: when two sets are almost completely overlapping, their symmetric difference is very small; when they are completely separated, it is big.

Moreover, the symmetric difference exhibits useful properties for a distance: it is commutative ($A \triangle B = B \triangle A$); and the empty set is neutral ($A \triangle \emptyset = A$ and $A \triangle A = \emptyset$). The symmetric distance is also associative, but this fact is negligible in this application.

2.3 Fitness Sharing

When a reliable distance metric is defined, one useful application is to exploit it for *fitness sharing*, one of many methods to enforce diversity inside the population of an EA [16] [14].

The general idea of fitness sharing is to artificially decrease the fitness of individuals in crowded areas of the search space. The fitness f_i of an individual I_i is modified in a fitness $f'_i = f/m_i$, where m_i is dependent upon the number of individuals in a given radius σ_s from individual I_i, and their distance from the individual itself. In particular,

$$m_i = \sum_{j=0}^{N} sh(I_i, I_j) \quad (2)$$

where N is the number of individuals in the population, and $sh(I_i, I_j)$ is defined as

$$sh(I_i, I_j) = \begin{cases} 1 - (\frac{d(I_i,I_j)}{\sigma_s})^{\alpha} & d(I_i, I_j) < \sigma_s \\ 0 & d(I_i, I_j) \geq \sigma_s \end{cases} \quad (3)$$

where σ_s is once again a user-defined radius, and $d(I_i, I_j)$ is a distance measure applicable to the individuals' representation. α is a constant parameter that regulates the shape of the sharing function. In many practical cases $\alpha = 1$, with the resulting sharing function referred to as the triangular sharing function [8].

3. PROPOSED APPROACH

In LGP, Shannon entropy can be effectively used as a metric to assess the diversity in a population at a given generation [4]. The entire population is considered a message, and each allele appearing in an individual is a symbol: entropy is then computed on the basis of the number of different symbols and their occurrences.

In a preliminary work [17], a variant of this approach is sketched. Instead of considering just the alleles of each gene, their disposition inside the individual is also taken into account. A symbol is no longer considered equivalent to a single allele, but to the allele and its position inside the individual instead. Following the idea that recurring structures might possess meaning, *n-grams* of nodes are also regarded as symbols. An n-gram is a group of n items from a longer sequence. For example, `a b`, `b c` and `c d` are all 2-grams from the sequence `a b c d`, while `a b c` and `b c d` are 3-grams. Very often n-grams are used for the purpose of modelling the statistical properties of sequences, particularly natural language [18].

Building on the same principles, a generic genotypic *Universal Information Distance* (UID) for individuals in LGP is proposed. Considering two individuals I_i and I_j, the UID is defined as

$$UID(I_i, I_j) = |S(I_i) \triangle S(I_j)| \quad (4)$$

where $S(I)$ represents the set of symbols in individual I, $\triangle$ is the symmetric difference as defined in Equation 1, and the operator $|S|$ denotes the cardinality of set S.

In other words, the UID between two individuals is the number of distinct symbols they do not have in common. Intuitively, when two individuals share many common symbols, the UID will be small; on the contrary, if they have no symbols in common, their UID will be high. An example is reported in Figure 3.

When used in practice, symbols for each individual are computed resorting to a hash function of the n-grams and alleles. It is interesting to notice how the proposed UID, that acts at genotype level and is quite straightforward to compute, could provide the same information of more computationally intensive distance metrics that are evaluated at phenotype level: thus, UID could be used for fitness sharing, delivering the same results as problem-specific metrics.

4. EXPERIMENTAL EVALUATION

The correlation between the proposed UID and two phenotypic distance metrics is examined, in two problems where individuals are encoded as strings of bits, and as Assembly language programs, respectively. Experiments with an evolutionary toolkit that supports LGP [15] are then performed for the two problems, and the effectiveness of UID for fitness sharing is compared to the previously considered phenotypic distance metrics.

In the all the experiments, the computation of UID is limited to n-grams of order 2 and 3, as a trade-off between computational efficiency and thoroughness of the approach. Symbols are computed resorting to the DJB[1] hash function.

4.1 Considered Problems

The proposed approach is tested on two benchmarks: NK-landscapes, a NP-complete problem where individuals are represented a strings of bits, and a simple Assembly-language generation task.

4.1.1 NK-Landscapes

In the NK-landscapes problem [10], the individual is a string of bits of fixed length: both the overall size of the fitness landscape and the number of its local optima can be adjusted by tuning the two parameters, N and K. Generally speaking, values of K close to N create more irregular landscapes. Albeit simple, this benchmark is widely studied in the optimization community, because it is proven to be NP-complete [19]. In the following experiments, values of N and K are very close, in order to obtain a fairly rugged fitness landscape.

4.1.2 Assembly Language Generation

This second set of experiments targets a simple Assembly language generation problem: the fitness of a program is the number of bits set to 1 in registry *%eax* at the end of its execution.

During the Assembly-generation problem, the minimum length of the variable part of a program is set to 1 instruction, the while the maximum length is set to 1500 instructions. For the initial population, individuals are generated in order to possess an average of 70 instructions, with a large standard deviation of 60 in order to efficiently sample the search space. Table 1 recapitulates the possible genes (instructions) appearing in the individuals.

The fitness function used for this experiment is based on both the result of a candidate program's execution and the length of its code, and it is defined as

$$f(I) = 10^4 \cdot (\sum_{i=0}^{N=31} \%eax[i]) + max(0, 10^4 - length(I)) \quad (5)$$

where $\%eax[i]$ is the value of the ith bit of register $\%eax$, while $length(I)$ represents the number of instructions of candidate program I. Thus, the most important objective is to

[1] http://cr.yp.to/djb.html

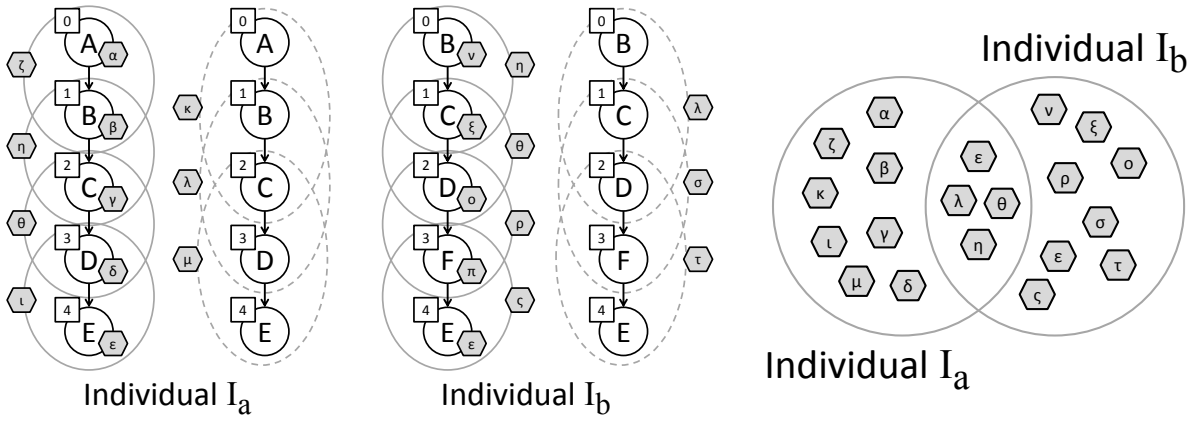

Figure 3: Example of symbols computed for alleles and $(2,3)$-grams for two individuals. Symbols are represented as Greek letters inside hexagons, alleles as Roman letters inside circles, while their position in the individual is reported in a square. The symbols common to the two individuals are ϵ (corresponding to allele E in position 4), η (2-gram $B-C$), θ (2-gram $C-D$) and λ (3-gram $B-C-D$). The UID between the two individuals is thus $|S(A) \triangle S(B)| = |S(A) \cup S(B) - S(A) \cap S(B)| = 16$

Gene	Parameters	Prob.
<ins> <sreg>, <dreg>	**ins**={addl, subl, movl, andl, orl, xorl, compl}, **sreg**={%eax, %ebx, %ecx, %edx}, **dreg**={%eax, %ebx, %ecx, %edx}	0.33
<ins> <scon>,<dreg>	**ins**={addl, subl, movl, andl, orl, xorl, compl}, **scon**={integer in (0,255)}, **dreg**={%eax, %ebx, %ecx, %edx}	0.33
<ins> <reg>	**ins**={incl, decl, notl}, **reg**={%eax, %ebx, %ecx, %edx}	0.33

Table 1: Possible genes appearing inside the individuals during the Assembly generation problem. For each gene, all variables and corresponding values are listed, as well as the probability of occurrence.

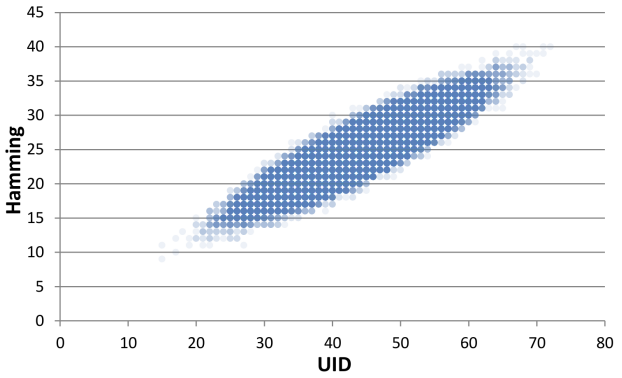

Figure 4: Correlation between the proposed UID distance and hamming distance in the standard OneMax problem (50 bits)– Sample of 500 random individuals.

set to 1 bits in register %*eax*, while a small bonus is assigned to individuals that perform the task with a moderate number of instructions.

4.2 Correlation

An important result that it is possible to immediately esteem looking at the figures is how much the proposed UID distance is well correlated with problem-specific phenotype-level distances. Figure 4 plots the UID against the standard Hamming distance for 500 random 50-bit individuals[2]. The cloud of points does not stretch down to the origin, nor up to the maximum because it is quite unlikely to find two identical strings, or two completely different ones, in a random pool.

Figure 5, on the other hand, plots the same data for all individuals generated during a run, until the optimal solution is reached. Since there is a strong similarity between all individuals in the same parental group, the cloud stretches down to distance zero. The correlation is even more evident than in the preceding example.

Figure 6 plots the proposed distance against the Levenshtein, or *edit*, distance for 500 random programs on the Assembly OneMax problem. Here, differences are more subtle and the number of overlapping values is reduced compared to the previous case. The triangular shape of the cloud is indicative: the two distances are better correlated for low values – that is, exactly when they are more useful: distinguishing between closely related individuals is in fact quite harder than discriminating between very different ones. The Levenshtein distance is a computationally-expensive metric that can be computed only at the level of phenotype. The proposed UID, on the contrary, can be efficiently calculated at the level of genotype and it is effective in estimating the distance between *similar* individuals.

[2]Several values are overlapping.

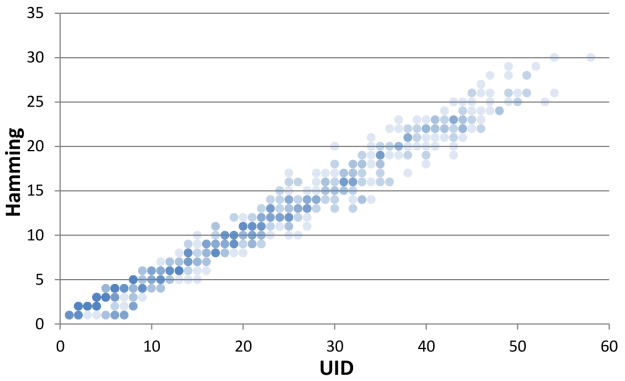

Figure 5: Correlation between the proposed UID distance and hamming distance in the standard OneMax problem (50 bits) – Individuals generated during a run.

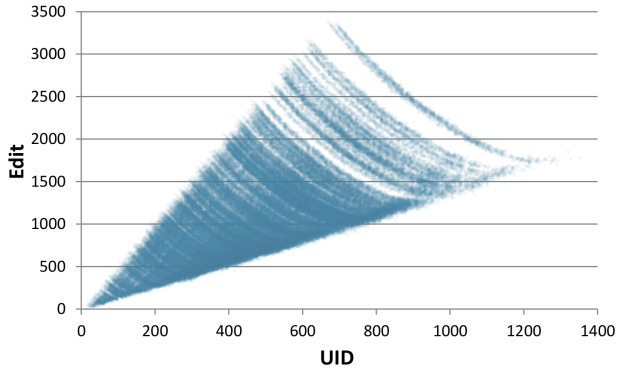

Figure 6: Correlation between the proposed UID distance and the Levenshtein distance in the Assembly OneMax problem (32 bits) – Sample of 500 random individuals.

4.3 Fitness Sharing

Since the proposed metric shows a heavy correlation with phenotypic distance metrics, its effectiveness can now be tested in multi-modal problems where a fitness sharing might help to spread the individuals over the search space. The aim of the following experiments is to show how using the proposed UID for fitness sharing delivers the same results as employing more rigorous problem-specific distance metrics, that are also more computationally expensive.

The first experiments are performed on the NK-landscapes (or NK-model) benchmark, tuned to obtain a rugged fitness landscape: the UID is compared to a classical Hamming distance [9]. A second set of experiments is then executed on a simple Assembly-language generation problem, where the objective is to obtain a program able to set all bits of register *%eax* to 1. In this latter tests, the UID is weighted against the Levenshtein distance [12].

In the LGP tool used for the experiments [15], μ is the population size; λ represents the number of genetic operators applied at each generation, rather than the offspring size; and σ is the *strength* of the mutation operators. Each time a mutation operator is applied to an individual, a random number r in the interval $(0, 1)$ is generated: if $r < \sigma$, the operator creates another mutation in the same individual, and a new r is generated.

It is important to notice that the parameters of the tool used in the trials have not been tuned to optimality, since the main objective of this experimental evaluation is to assess whether the fitness sharing mechanism behaves comparably when different distance metrics are applied with an equivalent radius, even with sub-optimal settings.

4.3.1 NK-landscapes

Parameters of the LGP tool used in the experiments are reported in Table 2. The mutation operator in this case simply changes the value of a random gene, while the one-point crossover selects a random cut point.

Parameter	Value	Parameter	Value
μ	32	P(one-point crossover)	0.5
λ	32	P(mutation)	0.5
σ	0.5	Max generations	50

Table 2: Parameters used during the experiments with fitness sharing in the NK-landscapes benchmark.

For each landscape, 10 experiments are run with the Hamming distance, and 10 with the UID, respectively, using equivalent radius measures derived from the correlation described in Subsection 4.2. At the end of the evolution, the fitness of the best individual (*online fitness*) and the average fitness of the final population (*offline fitness*) are compared.

From results in Table 3 it is noticeable how, for the same NK-landscape, the final online and offline fitness values are very close, as well as the average distance value between individuals in the population. In fact, running a two-sample Kolmogorov-Smirnov test on corresponding distributions for an equivalent radius reveals that the distributions are undistinguishable with $p < 0.01$.

4.3.2 Assembly OneMax

This second set of experiments targets a simple Assembly language generation problem: the fitness of a program is the number of bits set to 1 in registry *%eax* at the end of its execution. Table 4 summarizes the parameters used for the LGP tool in this experiment.

Parameter	Value	Parameter	Value
μ	10	P(crossover)	0.25
λ	7	P(mutation)	0.75
σ	0.7	Max generations	10

Table 4: Parameters used during the experiments with fitness sharing in the Assembly language generation problem.

The mutation operator, in this case, can add a random instruction; remove a random instruction; or change one or more parameters inside a random instruction, with equal probability. The crossover can operate on one or two cut points, with equal probability.

Table 5 shows the results over 10 experiments with each parameter configuration. At the end of the evolution, the fitness of the best individual (*online fitness*) and the average fitness of the final population (*offline fitness*) are compared.

Fitness sharing with Hamming distance					Fitness sharing with UID				
Radius (Hamming)	Online fitness (avg)	StDev	Offline fitness (avg)	StDev	Radius (UID)	Online fitness (avg)	StDev	Offline fitness (avg)	StDev
N=16, K=14, seed=1234567890									
5	0.6710	0.0167	0.3655	0.0256	10	0.6739	0.0326	0.3798	0.0391
N=16, K=14, seed=4242424242									
5	0.6774	0.0318	0.4094	0.0142	10	0.6948	0.0304	0.4099	0.0235
N=16, K=15, seed=1234567890									
5	0.6543	0.0228	0.3819	0.0124	10	0.6468	0.0109	0.3901	0.0137
N=16, K=15, seed=4242424242									
5	0.6770	0.0209	0.3912	0.0352	10	0.6671	0.0256	0.4067	0.0316

Table 3: Results for the set of experiments on the NK-landscapes benchmark. Experiments with fitness sharing with the Hamming distance (left) and the UID (right); experiments with a corresponding radius are reported on the same line.

Fitness sharing with Levenshtein distance					Fitness sharing with UID				
Radius (Levenshtein)	Online fitness (avg)	StDev	Offline fitness (avg)	StDev	Radius (UID)	Online fitness (avg)	StDev	Offline fitness (avg)	StDev
3	325,927	7,205.14	312,903	19,800.8	2	320,919	12,608.4	297,899	32,800.8
5	324,939	7,992.28	309,902	27,989.6	3	324,949	8,008.23	315,909	17,616.6
10	318,909	11,422.8	292,901	20,998.7	5	314,930	15,015.5	285,923	32014.3

Table 5: Results for the set of experiments on the Assembly-language generation benchmark. Experiments using fitness sharing with the Levenshtein distance (left) and the UID (right); experiments with a corresponding radius are reported on the same line.

Again, the results for an equivalent radius are indistinguishable through a two-sample Kolmogorov-Smirnov test with $p < 0.01$.

5. CONCLUSIONS

This paper presents a new distance metric for Linear Genetic Programming, based on the symmetric difference between the *information* contained in the genome, represented as symbols based upon n-grams and alleles. Experiments show a prominent correlation between the proposed method and phenotypic problem-specific distance measurements in two samples where individuals are radically different, NK-landscapes and Assembly language generation. The methodology is then successfully applied to two experiments with fitness sharing, showing again results comparable to more complex and computationally more demanding phenotypic fitness metrics.

Following the same general principles, a similar distance metric could also be defined for classical Genetic Programming, using for example a node and its children in place of n-grams. However, further studies are needed to assess the general validity of the proposed approach. Variations of the methodology need to be conceived in order to tackle individuals composed of both real values and constants, or individuals in combinatorial problems. Also, since the proposed approach relies upon the number of symbols encoded in each individual, its use for diversity preservation might implicitly benefit larger individuals, thus possibly contributing to the known issue of *bloating*.

6. ACKNOWLEDGEMENTS

The authors would like to thank Davide Barbieri (École des Hautes Études en Sciences Sociales, France) for his invaluable advice and stimulating conversation.

7. REFERENCES

[1] Symmetric difference. In E. J. Borowski and J. M. Borwein, editors, *The HarperCollins Dictionary of Mathematics*. HarperCollins, 1991.

[2] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. Genetic programming: An introduction: On the automatic evolution of computer programs and its applications (the morgan kaufmann series in artificial intelligence). 1997.

[3] M. Brameier and W. Banzhaf. *Linear Genetic Programming*, volume 117. Springer, 2007.

[4] F. Corno, E. Sánchez, and G. Squillero. Evolving assembly programs: how games help microprocessor validation. *Evolutionary Computation, IEEE Transactions on*, 9(6):695–706, 2005.

[5] G. Corriveau, R. Guilbault, A. Tahan, and R. Sabourin. Review and Study of Genotypic Diversity Measures for Real-Coded Representations. *IEEE Transactions on Evolutionary Computation*, 16(5):695–710, Oct. 2012.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182 –197, apr 2002.

[7] R. M. Friedberg. A learning machine: Part i. *IBM Journal of Research and Development*, 2(1):2–13, 1958.

[8] D. Goldberg. Genetic algorithms in search, optimization, and machine learning. 1989.

[9] R. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[10] S. Kauffman and E. Weinberger. The nk model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of theoretical biology*, 141(2):211–245, 1989.

[11] J. R. Koza. *Genetic Programming, On the Programming of Computers by Means of Natural Selection. A Bradford Book*. MIT Press, Cambridge, MA, USA, 1992.

[12] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics-Doklady*, volume 10, 1966.

[13] M. Mauldin. Maintaining diversity in genetic search. In *Proceedings of the national conference on artificial intelligence (AAAI conference on artificial intelligence)*, volume 247, page 250, 1984.

[14] C. D. Rosin and R. K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[15] E. Sanchez, M. Schillaci, and G. Squillero. *Evolutionary Optimization: the μGP toolkit*. Springer, 2011.

[16] B. Sareni and L. Krahenbuhl. Fitness sharing and niching methods revisited. *Evolutionary Computation, IEEE Transactions on*, 2(3):97–106, 1998.

[17] G. Squillero and A. Tonda. A novel methodology for diversity preservation in evolutionary algorithms. In *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pages 2223–2226. ACM, 2008.

[18] C. Suen. N-gram statistics for natural language understanding and text processing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):164–172, 1979.

[19] E. Weinberger. NP completeness of Kauffman nk model, a tuneably rugged fitness landscape. *Santa Fe Institute Technical Reports*, 1996.