Millipede, an Extended Representation for Genetic Algorithms

Clyde Meli

***Abstract*—In this paper, a proposal of a new extended binary representation ('Millipede') is made for genetic algorithms (GA). The GA representation may have an important effect on its performance. This paper looks at how single-valued encoding could be enhanced. Initial tests indicate this new solution encoding can be effective. Tests were made using a Genetic Algorithm (GA) package called GAGENES, written in object-oriented C++.**

***Index Terms*—Extended representation, solution encoding, genetic algorithms.**

I. Introduction

A new scheme is proposed in this paper to extend the classic binary representation as used in the original Genetic Algorithm (GA) by Holland [1]. The proposal involves extending using what the author calls the 'Millipede' solution encoding. The solution encoding being proposed in this paper is that if a point in the search space can be made to represent more than one point, it may be more efficient, much like a millipede can walk more efficiently by reaching more than a two-legged animal would.

Various other attempts at improving GA representation have been proposed in the past, including gray coding [2], adjusting population size on-the-fly[3], [4], the grouping genetic algorithm encoding structure [5], and the proportional GA. Banzhaf's GPM [6] looked at many genotypes mapping into one phenotype. In this paper we look at one genotype mapping into many phenotypes instead.

II. Encoding Scheme

The most straightforward encoding scheme typically used in a GA is the single-valued one, ie. one gene per object representation. As an example, chromosome 0110 would represent a value in the search space, eg x=6. Now if we could make every chromosome represent more than one value (or object) in the search space, this would make the size of the search space the GA has to search in smaller than with the default representation. As a result the GA's power is unimpaired and hypothetically might even increase.

While a Proportional GA (PGA) utilises individuals which are strings over an n-ary alphabet, the proposed Millipede GA utilises individuals which are bits just like the classical or canonical GA (CGA). However the mapping from the search space (genotype) onto the solution space (phenotype) is not one-to-one. It is instead, a one-to-many mapping defined as phenotype solution value = *f*0(genotype), f1(genotype)

Thus every space in the phenotype will correspond to more than one space in the genotype space.

In the most basic implementation of this concept, f may be defined as follows: *f*0(*x*)=2*x* and *f*1(*x*)=2*x*+1. This is the “two-legged” version, with two values. The corresponding fitness function would evaluate both values (using the traditional fitness function for a single value) and return the better of the two values.

Another implementation with three values could be as follows, on similar lines.

phenotype solution value = *f*0(genotype), *f*1(genotype), *f*2(genotype)

In this case f may be defined as follows: *f*0(*x*)=3*x*, *f*1(*x*)=3*x*+1, and *f*2(*x*)=3*x*+2

Generalising to n-values:

phenotype solution value = *f*0(genotype), *f*1(genotype), *f*2(genotype),.., *fn*-1(genotype)

In this case *f* may be defined as follows: *f*0(*x*)=*nx*, *f*1(*x*)=*nx*+1,... and *fn*-1(*x*)=*nx*+*n*-1

Once the GA finds the better solution from the population, one would have to evaluate the *n*-values to find the correct *x*-value from the n values represented.

III. Schemata

In the CGA, a population of p individuals of length m processes at least 2m and at most p.2m schemata [7]. In the n-value generalised Millipede GA, every individual represents *n* classic individuals, thus an equivalent population would process at least 2m and at most (np).2m schemata.

IV. Test Runs

GAGENES, a C++ GA implementation by the author, was run on an AMD Phenom(TM) II X6 processor computer running Windows 7 and Gentoo Linux in a VM.

The GAGENES program was used to solve the *X*4 problem which is the function single value minimisation problem [8] as well as the toy problem, maximize *x*2. Both cases involved a single integer parameter.

$$f(x) = x4\text{-}12x3+15x2+56x–60$$

The following process was performed. The Mersenne Twister PRNG [9] was used during the initialisation phase and afterwards. For every problem, the GA was run ten times, for a maximum of 500 generations. The runs were done once with the standard encoding, once with *x*2 extended encoding,

Manuscript received October 12, 2012; revised January 30, 2013.

Clyde Meli is with the CIS Department, University of Malta (e-mail: clyde.meli@um.edu.mt).

once with *x*4 extended encoding and once with x16 extended encoding. Population size was 100, crossover probability was 1.0 and the mutation probability was set to 0.2.

The tests were run once under Windows 7. All tests were run on an AMD Phenom(TM) II X6 processor computer running Windows 7.

The following diagrams illustrate the result of running the test runs.

V. Results of GA Testing

Fig. 1 and Fig. 2 show the standard GA (CGA) compared with the *x*2 extended encoding, plotting runs against best fitness value in Fig. 1 and best average fitness in Fig. 2 respectively, solving *X*4. The extended version is slightly better than the CGA, especially when it comes to fitness averages. Fig. 3 and Fig. 4 show the same problem (*X*4) but comparing CGA with *x*16 against best fitness and best average fitness in Fig. 3 and Fig. 4 respectively. *x*16 is much better than CGA, it has much higher fitness values. Fig. 5 and Fig. 6 show the *X*2 problem (*x*2) again against best fitness and best average fitness respectively, CGA compared to *x*2 encoding. *x*2 has better fitness values in all but one run, and the average fitness diagram shows the same story. Fig. 7 and Fig. 8 show the same problem comparing CGA to *x*16 again against best fitness value and best average fitness respectively. In all runs *x*16 shows much higher fitness values, and much higher average fitness values.

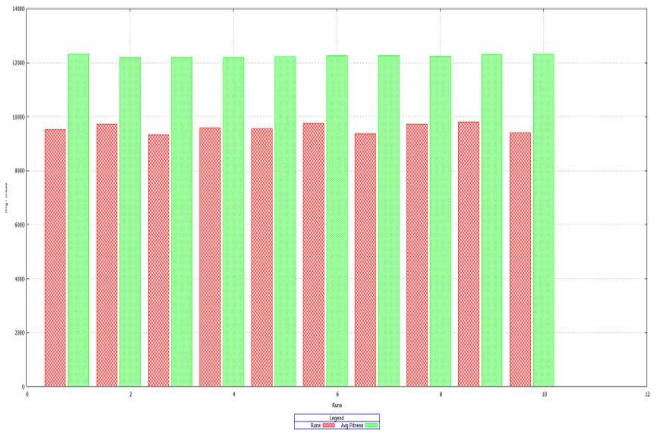

Fig. 3. X4-CGA vs EXT by 16.

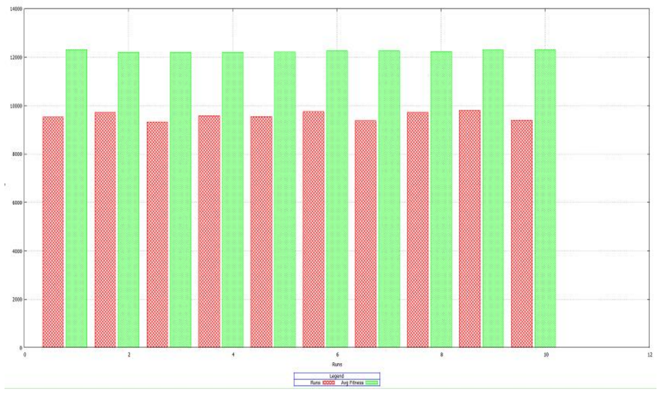

Fig. 4. X4-CGA vs EXT by 16 averages.

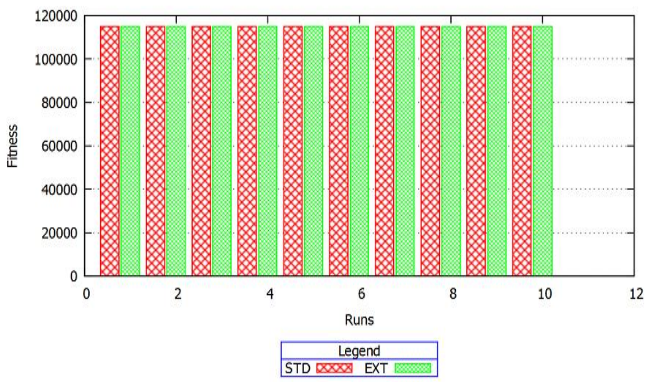

Fig. 1. X4-CGA vs EXT by 2.

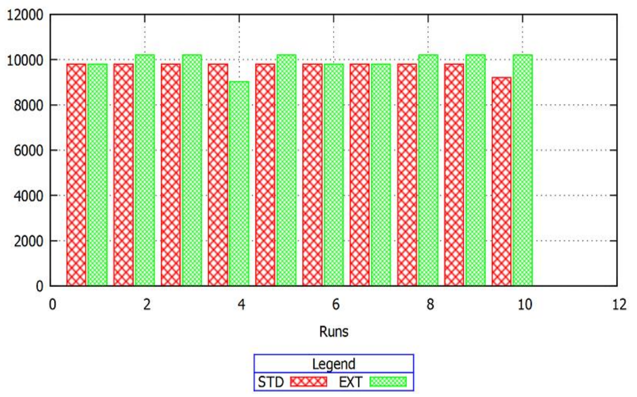

Fig. 5. XSQUARED-CGA vs EXT by 2.

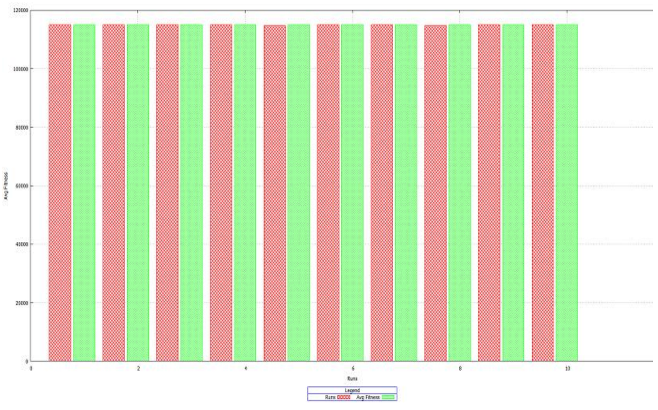

Fig. 2. X4-CGA vs EXT by 2 averages.

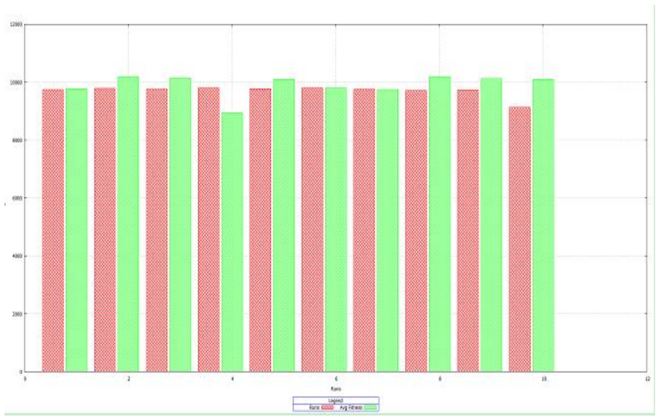

Fig. 6. XSQUARED-CGA vs EXT by 2 averages.

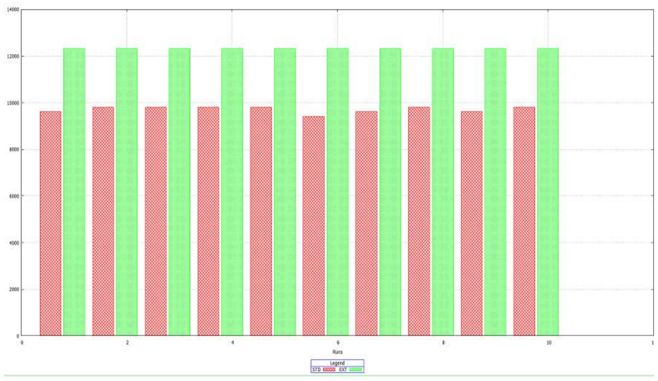

Fig. 7. XSQUARED-CGA vs EXT by 16

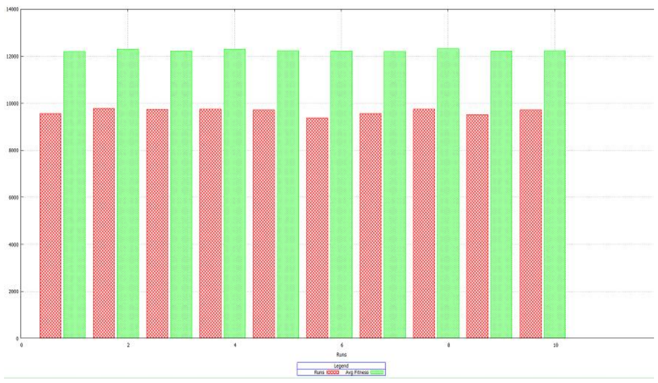

Fig. 8. XSQUARED-CGA vs EXT by 16 averages.

VI. Conclusion

The Millipede solution encoding appears to have promise. It is clearly dependent on the problem, thus for some problems it will be much more beneficial to use it than for some others.

One might worry that it would slow down the GA, however this has not been observed so far in these tests. It will probably depend upon the fitness function. If this is computationally complex, then calling it many times may slow down the GA.

There are further extensions to the encoding scheme as proposed in this paper which should be experimented with in future research. One could represent other numbers in a sequence, for instance.

References

[1] J. H. Holland, "Adaptation in natural and artificial systems: an introductory analysis with applications to biology," *Control and Artificial Intelligence (Complex Adaptive Systems S.)*, The MIT Press, 1975.

[2] D. E. Goldberg, "Genetic algorithms in search," *Optimization and Machine Learning*, Addison-Wesley, 1989.

[3] A. E. Eiben, E. Marchiori, and V. A. Valkó, "Evolutionary algorithms with on-the-fly population size adjustment," *Parallel Problem Solving from Nature PPSN VIII, LNCS 3242*, Springer, 2004, pp. 41-50.

[4] F. G. Lobo, I. Centro, and M. Ecológica, Revisiting Evolutionary Algorithms with On-the-Fly Population Size Adjustment.

[5] E. Falkenauer, "A new representation and operators for genetic algorithms applied to grouping problems," *Evol. Comput*, vol. 2, pp. 123-144, June 1994.

[6] W. Banzhaf, Y. Davidor, H.–.Schwefel, and R. M. (eds), *Genotype-Phenotype-Mapping and Neutral Variation - A case study in Genetic Programming*, 1994.

[7] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1999.

[8] A. Dolan. GA playground - java genetic algorithms toolkit. [Online]. Available: http://www.aridolan.com/ga/gaa/gaa.html#SingleVarMin [Accessed: Sep. 25 2012].

[9] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator," *ACM Trans. on Modeling and Computer Simulation*, vol. 8, 1998, pp. 3-30.

Clyde Meli was born in Sliema, Malta and has an M.Phil degree in computing from the University of Malta in Malta obtained in 1992 and is currently finalising his Ph.D. degree at the same university.

He has worked full-time for the University of Malta in Malta since graduating from his first B.Sc. degree in computing & mathematics at the same university in 1993. He has also worked part-time for REMPEC (UN) and is currently also working for the University's Gozo branch giving lectures in the neighbouring island of Gozo. Dr. Meli is a member of IEEE and ACM, and is currently on the IT Services Liason Committee at the same university. He was on the WICT (Workshop in ICT Malta) 2010 board. The author is interested in research in genetic algorithms, genetic programming and computer security including spam detection, the latter being the subject of his Ph.D.