
The Proportional Genetic Algorithm

Annie S. Wu
School of EECS
University of Central Florida
Orlando, FL 32816-2362
aswu@cs.ucf.edu

Ivan Garibay
School of EECS
University of Central Florida
Orlando, FL 32816-2362
igaribay@cs.ucf.edu

Abstract

We introduce a genetic algorithm (GA) with a new representation method which we call the proportional GA (PGA). The PGA is a multi-character GA that relies on the existence or non-existence of genes to determine the information that is expressed. The information represented by a PGA individual depends only on what is present on the individual and not on the order in which it is present. As a result, the order of the encoded information is free to evolve in response factors other than the value of the solution, for example, in response to the identification and formation of building blocks. The PGA is also able to dynamically evolve the resolution of encoded information.

1 Introduction

This paper summarizes the initial studies of a new genetic algorithm (GA) representation method which we call the proportional genetic algorithm (PGA). Additional details of this work may be found in [23].

A genetic algorithm (GA) works with a population of individuals each of which represents a potential solution to the problem to be solved. A typical individual is a binary string on which the problem solution is encoded. Problem representation is one of the key decisions to be made when applying a GA to a problem. How a problem is represented in a GA individual determines the shape of the solution space that a GA must search. As a result, different encodings of the same problem are essentially different problems for a GA [18]. Selecting a representation that correlates with a problem's fitness function can make that problem much easier for a GA to solve [13, 4]. In the canonical

GA (CGA) representation, most decisions regarding a problem's representation are decided and fixed prior to execution. Unfortunately, there is often not enough information about a problem to make effective decisions on arrangement of information.

We introduce a GA with a new representation format which we call the proportional GA (PGA). Like a CGA, a PGA encodes solutions as linear strings. A PGA, however, is a multi-character GA that relies on the existence or non-existence of genes to determine the information represented. The information represented by a PGA individual depends only on what is present on the individual and not on the order in which it is present. As a result, the order of the encoded information can evolve in response factors other than the value(s) of the solution, for example, in response to the search for tightly linked building blocks. The PGA representation achieves location independence without the additional overhead of "tags" or "location markers" that many existing location independent encodings must use. In addition, a PGA is able to dynamically adapt the resolution of encoded information.

2 Motivation

The typical CGA uses an order-based encoding. Information is encoded on an individual in a location dependent manner, e.g. bits 0 to 3 always represent parameter 1; bits 4 to 7, parameter 2; and so on. The ordering of the information on an individual is typically an arbitrary programmer decision. A difficulty with this type of representation arises due to crossover's positional bias [5]. Positional bias refers to the fact that bits that are relatively far apart on an individual will be more likely to be disrupted (separated) by crossover than bits that are close together. Conversely, bits that are close together are more likely to be treated as atomic units (not separated) by crossover than bits that are far apart. If there is epistasis between vari-

Value V	$positive_char(V)$	$negative_char(V)$	V_{min}	V_{max}
V_1	A	a	0	10
V_2	B	b	0	10
V_3	C	c	0	10
V_4	D	d	0	10
V_5	E	e	0	10

Table 3: PGA2 character assignment in a five value problem.

modifications must be made to the PGA representation to encode solutions for other types of problems.

We focus on the common problem format involving a search for a vector of values (either integer or floating point). Given a problem in which we are searching for $alpha = 5$ parameter values, $V_i, i = 0, \dots, alpha - 1$ and the range of each value is given by $V_{i,min}$ and $V_{i,max}$. We define a second PGA strategy called PGA2 in table 3. The number “positive” and “negative” characters on an individual are used to calculate

$$pct(V_i) = \frac{positive_char(V_i)}{positive_char(V_i) + negative_char(V_i)}$$

where $i = 0, \dots, alpha - 1$ and $0.0 \leq pct(V_i) \leq 1.0$. The value of each parameter is calculated by the equation

$$P_{PGA2}(V_i) = V_{i,min} + pct(V_i) \times (V_{i,max} - V_{i,min}).$$

A typical individual of length 50 such as

AccBdDeeEbAbBDEccaAAAEebbEEECDDbbbABCDEedcbaAAddbA

would generate the parameter values shown in table 4. As with PGA1, the evaluation of a PGA2 individual is completely independent of the arrangement of characters. Thus, the individual

AAAAAAAAaaBBBBbbbbbbbbbCCCccccDDDDdddEEEEEEeeee

also evaluates to the values shown in table 4.

4 The issue of resolution

Typically, CGA individuals are strings over a binary alphabet while PGA individuals are strings over a higher-arity alphabet. In order to perform a fair comparison between CGA and PGA, we need to find a relationship among their individuals lengths and alphabet

Value V	Number of $positive_char(V)$	Number of $negative_char(V)$	$pct(V)$	Expressed value
V_1	9 A’s	2 a’s	$9/(9+2)$	8.18
V_2	3 B’s	9 b’s	$3/(3+9)$	2.5
V_3	3 C’s	5 c’s	$3/(3+5)$	3.75
V_4	4 D’s	4 d’s	$4/(4+4)$	5.0
V_5	7 E’s	4 e’s	$7/(7+4)$	6.36

Table 4: Allocation of resources specified by example individual of length 50.

sizes to ensure that both algorithms are targeting solution spaces of comparable complexity. We call this the issue of resolution. Let us introduce the following notation: the *search space*, $\mathcal{G}_{(l,\Sigma)} = \{g_i\}$, is the set of all genotypes, g_i , of length l over alphabet Σ ; and the *solution space*, $\mathcal{P} = \{p_j\}$, is the set of phenotypes, p_j , which are strings of length l for a CGA and *multisets*¹ of length l for a PGA. The mapping between spaces is $\mathcal{M} : \mathcal{G}_{(l,\Sigma)} \mapsto \mathcal{P}$, hence $p_j = \mathcal{M}(g_i)$. Let \mathcal{A} be a CGA with individuals of length $l_{\mathcal{A}}$ over an alphabet $\Sigma_{\mathcal{A}}$ with cardinality $|\Sigma_{\mathcal{A}}| = n_{\mathcal{A}}$. Let \mathcal{A} ’s search space, solution space, and mapping be denoted by $\mathcal{G}_{(l_{\mathcal{A}},\Sigma_{\mathcal{A}})}$, $\mathcal{P}_{\mathcal{A}}$, and $\mathcal{M}_{\mathcal{A}}$, respectively. Similarly, let \mathcal{B} be a PGA with length $l_{\mathcal{B}}$ and alphabet $\Sigma_{\mathcal{B}}$ with cardinality of $|\Sigma_{\mathcal{B}}| = n_{\mathcal{B}}$. Let \mathcal{B} ’s search space, solution space and mapping be denoted accordingly by $\mathcal{G}_{(l_{\mathcal{B}},\Sigma_{\mathcal{B}})}$, $\mathcal{P}_{\mathcal{B}}$, and $\mathcal{M}_{\mathcal{B}}$. For the CGA, \mathcal{A} , the search space and the solution space are the same size, $(n_{\mathcal{A}})^{l_{\mathcal{A}}}$, and the genotype-phenotype mapping, $\mathcal{M}_{\mathcal{A}}$, is trivial—one-to-one and onto from strings to strings—in fact, the identity mapping. For the PGA, \mathcal{B} , the size of the search space is similarly $(n_{\mathcal{B}})^{l_{\mathcal{B}}}$, but the genotype-phenotype mapping, $\mathcal{M}_{\mathcal{B}}$, will map strings onto multisets instead of strings to strings. This mapping is, in general, many-to-one and onto, leading to a solution space of smaller size than its corresponding search space. The number of elements on each of these multisets is the same as the genotype length. The mapping, $\mathcal{M}_{\mathcal{B}}$, represents the essence of the PGA idea: every string is mapped according to the multiplicity of its elements regardless of the arrangement of the elements. We can use basic combinatorics to show that the total number of multisets produced by the mapping, $\mathcal{M}_{\mathcal{B}} : \mathcal{G}_{(l_{\mathcal{B}},\Sigma_{\mathcal{B}})} \mapsto \mathcal{P}_{\mathcal{B}}$, is $\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}$, which is the size of the solution space.

¹Multisets are analogous to sets, except that they may contain multiple copies of identical elements.

To fairly compare the CGA and PGA, both should search the same sized solution space. Therefore,

$$(n_{\mathcal{A}})^{l_{\mathcal{A}}} = \binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}} \quad (1)$$

Equation (1) describes the relationship between the lengths and alphabet sizes of an arbitrary CGA and PGA that would ensure resulting solution spaces of equal size. A complete discussion regarding the issue of resolution can be found in [23].

5 Test problems

Preliminary studies on the PGA were performed on three types of test problems: resource allocation, number matching, and symbolic regression. All three problems involve a search for five numeric values. A CGA individual encodes a solution in 40 bits. Those 40 bits are divided into five eight-bit fields, each encoding one value in binary format at a resolution of 256. For the resource allocation problem, the PGA1 encoding is described in section 3.1. For the number match and symbolic regression problems, the PGA2 encoding is described in section 3.2.

The fitness of the evolved solutions are evaluated as follows.

5.1 Resource allocation

The resource allocation problem involves the allocation of a fixed pool of resources among $alpha = 5$ users. Thus, a GA's goal is to find a set of values that sum to a predefined constant. The target allocation is randomly generated at the start of each GA run.

In the CGA encoding, each encoded value is divided by the sum of all of the values to produce a proportion between 0.0 and 1.0.

Given $alpha = 5$ target values or proportions, $T_i, i = 0, \dots, alpha - 1$ where $\sum T_i = 1.0$, and $alpha$ values $E_i, i = 0, \dots, alpha - 1$, encoded by either the CGA or PGA1, we first calculate the *ratio* of each corresponding pair:

$$ratio(i) = \begin{cases} \frac{T_i}{E_i} & \text{if } T_i < E_i \\ \frac{E_i}{T_i} & \text{otherwise} \end{cases}$$

This value gives an indication of how close each encoded value is to the corresponding target value. The fitness of an individual is the average of all ratios:

$$fitness = \frac{\sum_{i=0}^{alpha-1} ratio(i)}{alpha}$$

A perfect match gives the maximum score of 1.0.

5.2 Number match

The number match problem is a search for $alpha = 5$ independent values. The $alpha$ independent target values are randomly generated at the beginning of each run so that each run searches for a different set of values.

The fitness function for this problem is the same as the fitness function for the resource allocation problem. Given $alpha$ target values, T_i , and $alpha$ encoded values, $E_i, i = 0, \dots, alpha - 1$, we first calculate the ratio of the smaller value divided by the larger value.

$$ratio(i) = \begin{cases} \frac{T_i}{E_i} & \text{if } T_i < E_i \\ \frac{E_i}{T_i} & \text{otherwise} \end{cases}$$

The fitness of an individual is the average of all ratios:

$$fitness = \frac{\sum_{i=0}^{alpha-1} ratio(i)}{alpha}$$

Again, a perfect match gives the maximum score of 1.0.

5.3 Symbolic Regression

Given a set of p data points, $d_i, i = 0, \dots, p - 1$, the symbolic regression problem is a search for $alpha = 5$ coefficients that, when plugged into the equation

$$f(x) = \mathbf{a}x^2 + \mathbf{b}x + \mathbf{c} + \mathbf{d}\cos(x) + \mathbf{e}\sin(x),$$

most closely approximate the target equation. Instead of trying to match the encoded coefficient values to target values, the fitness function is calculated from the difference between the target data points and the function values generated using the encoded coefficient values. The *ratio*(i) at each data point is

$$ratio(i) = \begin{cases} \frac{d_i}{f(x)} & \text{if } d_i < f(x) \\ \frac{f(x)}{d_i} & \text{otherwise} \end{cases}$$

The fitness is the average of all ratios:

$$fitness = \frac{\sum_{i=0}^{a-1} ratio(i)}{a}$$

A perfect match gives a maximum score of 1.0. This problem differs from the number match problem in that the $alpha$ values are interdependent. Changes in a single value can affect the impact of other values.

Population size	: 200
Maximum number of generations	: 500
Selection method	: tournament
Crossover type	: two-point (fixed len), homologous (variable len)
Crossover rate	: 1.0
Mutation rate	: 0.01, 0.005

Table 5: GA parameter settings.

6 Experimental details

Table 5 gives the parameter setting used in these experiments. Mutation rate indicates the probability that a single character will be mutated. with equal probability to any other character in the alphabet. Homologous crossover is implemented as described in [3]. Each experiment was run 100 times and the results averaged over all runs.

In addition to comparing a PGA and a CGA, We test a total of six variations of each PGA:

PGA-40: Simple PGA with fixed length individuals. The length of the individuals is the same as the length of the individuals in the CGA (40 bits).

PGA-255: Simple PGA with fixed length of 255 bits. As the full resolution of the PGA requires a genome length of 1275, PGA-255 is still somewhat penalized with respect to resolution; however, this PGA at least has the resolution of a single CGA field. The location independence of the PGA allow us to overlap all five values on the same genome.

PGA-40-nc: PGA with fixed length individuals and non-coding regions. Length is same as in **PGA-40**. One or more non-coding characters are added to the PGA alphabet.

PGA-255-nc: Same as **PGA-40-nc** with length of 255.

PGA-var: PGA with variable lengthed individuals. The maximum allowed length is set at 2048 to ensure that the PGA would have as much resolution as it would need. Note that this value is bigger than the required length of 1275. No parsimony pressure is applied unless otherwise specified.

PGA-var-nc: PGA with variable lengthed individuals and non-coding regions.

Figures 1 to 3 show comparisons of a CGA with the PGA variations described above.

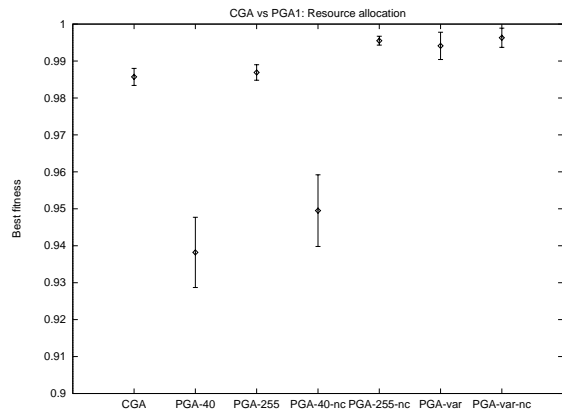


Figure 1: CGA -vs- PGA1 on resource allocation: Fitness of best solution found averaged over 100 run and 95% confidence intervals.

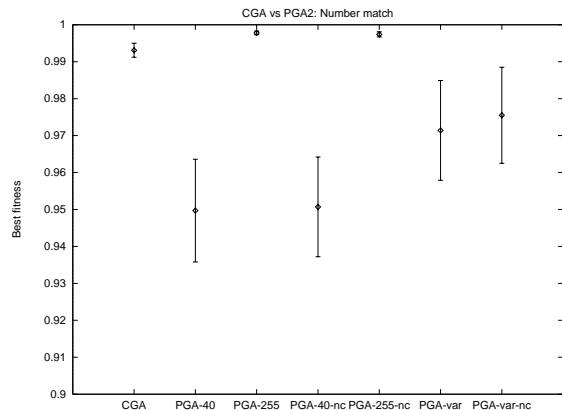


Figure 2: CGA -vs- PGA2 on number match: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

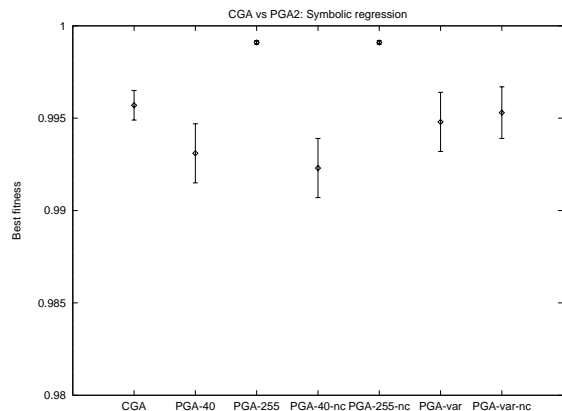


Figure 3: CGA -vs- PGA2 on symbolic regression: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

Results indicate that the PGA can perform as well or better than a CGA. The determining factor for PGA performance appears to be its resolution. PGA performance is comparable or better than the CGA, despite lowered resolution; however, extremely low resolution does appear to limit PGA performance. A measure of this limit is under investigation. Given the opportunity, a PGA will attempt to evolve the length of its individuals to accommodate the required resolutions.

Despite the lowered resolution of the PGA representation, PGA performance appears to be competitive with CGA performance and PGA runs appear to take advantage of the flexibility provided by variable length individuals. We have tested the PGA representation successfully on a number of types of problems involving the search for sets of both independent and interdependent values.

References

- [1] S. Bandyopadhyay, H. Kargupta, and G. Wang. Revisiting the gemga: Scalable evolutionary optimization through linkage learning. In *Proc. IEEE Int'l Conf. on Evolutionary Computation*, pages 603–608. IEEE Press, 1998.
- [2] W. Banzhaf. Genotype-phenotype mapping an neutral variation – a case study in genetic programming. In *PPSN 3*, pages 987–995, 1998.
- [3] D. S. Burke, K. A. De Jong, J. J. Grefenstette, C. L. Ramsey, and A. S. Wu. Putting more genetics into genetic algorithms. *Evolutionary Computation*, 6(4):387–410, 1998.
- [4] A. Clark and C. Thornton. Trading spaces: Computation, representation, and the limits of uninformed learning. *Behavioral and Brain Sciences*, 20:57–90, 1997.
- [5] L. J. Eshelman, R. A. Caruana, and J. D. Schaffer. Biases in the crossover landscape. In J. D. Schaffer, editor, *Proc. 3rd Int'l Conf. on Genetic Algorithms*, pages 10–19, 1989.
- [6] R. W. Franceschini, A. S. Wu, and A. Mukherjee. Computational strategies for disaggregation. In *Proc. 9th Conf. on Computer Generated Forces and Behavioral Representation*, 2000.
- [7] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison Wesley, 1989.
- [8] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid accurate optimization of difficult problems using fast messy genetic algorithms. In S. Forrest, editor, *Proc. 5th Int'l Conf. on Genetic Algorithms*, pages 56–64, 1993.
- [9] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.
- [10] J. Grefenstette, C. L. Ramsey, and A. C. Schultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5(4):355–381, 1990.
- [11] G. R. Harik. *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. PhD thesis, University of Michigan, 1997.
- [12] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [13] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman, editor, *Proc. 6th Int'l Conf. on Genetic Algorithms*, 1995.
- [14] H. Kargupta. The gene expression messy genetic algorithm. In *Proc. IEEE Int'l Conf. on Evolutionary Computation*, pages 814–819. IEEE Press, 1996.
- [15] H. Kargupta. A striking property of genetic code-like transformations. *Complex Systems*, 11, 2001.
- [16] H. Kargupta and B. H. Park. Gene expression and fast construction of distributed evolutionary representation. *Evolutionary Computation*, 9(1):43–69, 2001.
- [17] R. E. Keller and W. Banzhaf. Evolution of genetic code in genetic programming. In *Proc. Genetic and Evolutionary Computation Conf.*, 1999.
- [18] K. Mathias and L. D. Whitley. Transforming the search space with gray coding. In *Proc. IEEE Int'l Conf. on Evolutionary Computation*, pages 513–518, 1994.
- [19] M. Shackleton, R. Shipman, and M. Ebner. An investigation of redundant genotype-phenotype mappings and their role in evolutionary search. In *Proc. Congress on Evolutionary Computation*, pages 493–500, 2000.
- [20] R. E. Smith and D. E. Goldberg. Diploidy and dominance in artificial genetic search. *Complex Systems*, 6(3):251–285, 1992.
- [21] T. Soule and A. E. Ball. A genetic algorithm with multiple reading frames. In *Proc. Genetic and Evolutionary Computation Conf.*, 2001.
- [22] G. Syswerda. Uniform crossover in genetic algorithms. In *Proc. 3rd Int'l Conf. on Genetic Algorithms*, 1989.
- [23] A. S. Wu and I. Garibay. The proportional genetic algorithm: Gene expression in a genetic algorithm. *Genetic Programming and Evolvable Hardware*, 2002. In press.
- [24] A. S. Wu and R. K. Lindsay. A comparison of the fixed and floating building block representation in the genetic algorithm. *Evolutionary Computation*, 4(2):169–193, 1996.
- [25] A. S. Wu, A. C. Schultz, and A. Agah. Evolving control for distributed micro air vehicles. In *Proc. IEEE Int'l Symp. Computational Intelligence in Robotics and Automation*, 1999.