The Proportional Genetic Algorithm: Gene Expression in a Genetic Algorithm

Annie S. Wu (aswu@cs.ucf.edu) and Ivan Garibay (igaribay@cs.ucf.edu)
University of Central Florida, School of EECS, P. O. Box 162362, Orlando, FL 32816-2362

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. In this paper, we describe our motivations for developing this representation and provide a detailed description of a PGA along with discussion of its benefits and drawbacks. We compare the behavior of a PGA with that of a canonical GA (CGA) and discuss conclusions and future work based on these preliminary studies.

Keywords: genetic algorithm, representation, gene expression, proportional genetic algorithm

1. Introduction

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 [26]. Selecting a representation that correlates with a problem's fitness function can make that problem much easier for a GA to solve [18, 5]. In the traditional GA 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 traditional GA, a PGA encodes so-

© 2002 *Kluwer Academic Publishers. Printed in the Netherlands.*

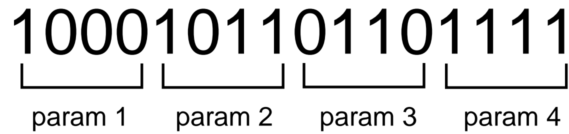

Figure 1. Example of location dependent problem representation. Each parameter value is encoded at a fixed location on each individual.

lutions 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.

This paper introduces this new representation format and discusses the motivations behind our work. We also provide details on how to implement a PGA, describe preliminary experimental comparisons of a PGA with a canonical GA (CGA), and discuss initial conclusions about the PGA.

2. Background

The importance of the arrangement of encoded information has been recognized since the introduction of the CGA. Holland [17] recommended the use of an inversion operator as a way to rearrange information to find tightly linked building blocks. A *building block* is a collection of bits or regions on an individual that work together to affect the fitness of that individual. Early GA studies successfully used a representation in which both the value and the ordering of the genes are dynamically evolved by a GA [1] and investigated the effects of gene order on a GA [11]. Nevertheless since then, much of the community has, in a sense, "converged prematurely" to the neat, orderly, fixed encodings that are so common today.

Figure 1 shows an example of a typical CGA problem representation. 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 [9]. 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 various components of a problem solution, encoding those components close together will reduce the chance that crossover will disrupt those components. As encoded in Figure 1, the probability that crossover will disrupt param1 and param2 is much lower than the probability that crossover will disrupt param1 and param4. The problem is that we do not always know which components are epistatic, and an arbitrary arrangement of the components of a solution on an individual may or may not elicit good performance from a CGA.

A number of options have been investigated to deal with this problem, including operators such as inversion [17, 12] and uniform crossover [33] which have little or no positional bias. A significant amount of research has also been devoted to examining ways in which problem representation may be designed to improve the GA search process. Numerous studies have focused on representations that allow a GA to dynamically evolve the arrangement of information on an individual. This class of representations provides flexibility at the expense of an increased search space. An alternative approach is the use of genotype-to-phenotype mappings to distinquish and transform between search space and solution space.

2.1. Bit level location independence

The basic unit of information on a typical GA individual is a bit. Thus, the bit level is the lowest at which one would need to consider the arrangement of information. In some cases, problem encoding naturally places the bits that make up a building block in close proximity on an individual, e.g. each of the groups of four bits that encode for a single parameter value in figure 1. In other cases, a building block may consist of bits that are spread out across an entire individual. In the latter example, the ability to dynamically evolve the order of the bits on an individual would improve linkage among related bits and help identify and preserve building blocks.

Goldberg et al. [14] developed the messy GA to investigate whether a GA is able to identify and recombine building blocks to form optimal solutions. The messy GA encodes individuals as a vector of pairs, where each pair specifies the location and value of a bit. Special rules deal with overspecified (duplicate) or underspecified (missing) bits. This encoding

allows a GA to evolve both the value and the ordering of information on an individual. The fact that the physical ordering of information on an individual can be different from the interpreted ordering means that a messy GA can arrange related bits into close proximity to decrease the chance of disruption by crossover. Thus, a messy GA can rearrange information to form tightly linked building blocks which may then be recombined to create a complete solution. Empirical tests indicate that a messy GA can solve difficult, deceptive problems, even when encodings are "loose". In comparison, a simple GA, on average, is only able to achieve about 25% of the correct solution on the problems tested.

The fast messy GA [13] improves upon the messy GA by reducing the computational efforts in the early phases of the messy GA. The gene expression messy GA [19] enhances the messy GA to actively search for linkage relationships within a search space. The linkage learning GA [16] is an elegant extension which uses a modified representation and new operators that alter the relative positioning of genes to create building blocks.

While the messy GA and linkage learning GA allow for bitwise location independence in the problem representation, the mapping from a GA individual to a problem solution still retains some aspects of order: (1) the order in which bits appear in an individual affects their expression, and (2) some sort of reordering of the bits is typically required to decode an individual into a solution. A side effect of the latter is that each bit within a list of ordered bits has a specific meaning or use, e.g. the 2nd binary digit. As a result, all encoding bits must exist in order for a solution to be formed. In the messy GA, missing or duplicate bits require special repair or selection mechanisms to determine what is expressed. These special mechanisms can increase the complexity of the system and potentially introduce biases into the search process. The linkage learning GA eliminates some of these biases by maintaining copies of all bits in each individual.

2.2. Group level location independence

Many problems such as the example in figure 1 use multiple bits to encode the component values of their solutions. These groups of bits are obvious building blocks which would likely benefit from a tight encoding. Combinations of these basic building blocks or genes may also create higher level building blocks. As a result, the ordering and arrangement of groups of bits (*basic building blocks*) has also been an active area of research.

Wu and Lindsay [37] developed the floating representation to study the maintenance of diversity in a GA. This representation assigns a tag

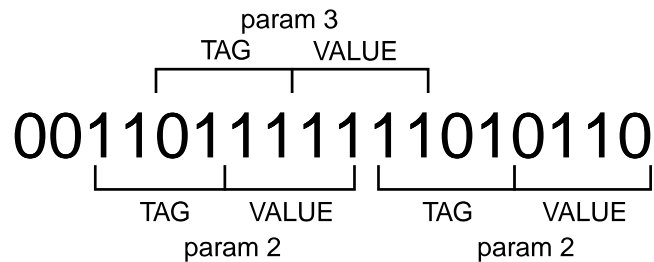

Figure 2. Example of the floating representation.

to each parameter or basic building block in a solution. The fitness function then searches each individual for all tags and retrieves the value of each parameter from the bits immediately following each tag. Figure 2 shows an example of how a problem may be encoded with the floating representation. In this example, the tag for `param2` is `1101` and the tag for `param3` is `0111`. The floating representation allows basic building blocks to be encoded anywhere on an individual while keeping the component bits of a basic building block together. This representation allows for overlapping building blocks which could reduce the amount of resources required to specify a solution. Like the messy GA, special rules are necessary to deal with overspecification and underspecification of values. Studies indicate that a GA using the floating representation is able to retain greater diversity in the population and that genetic operators in this system have more exploratory power [36].

Soule and Ball [32] investigated an encoding similar to the floating representation focusing on the idea of reading frames. Tags were used to delimit genes which could appear anywhere on an individual. These studies were conducted to investigate the likelihood and consequences of evolving overlapping genes. While performance varied from problem to problem, the GA was able to evolve genes much longer than the individuals encoding them.

Burke et al. [4] investigated the evolution of the Virtual Virus (VIV) which encoded genes as location independent character segments. This work focused on investigating the factors that affect the evolved size and fitness of individuals and on understanding the use of non-coding regions in a GA. Empirical studies produced clear evidence of regions switching between coding and non-coding status, adding support to the hypothesis that non-coding regions may serve as backups for coding regions by storing extra copies of useful information. Both coding and non-coding regions contained obvious building blocks (potential genes). In addition, the GA appeared to monitor the length of evolved individuals in response to parameter settings such as mutation rate. Variation in average length of individuals throughout a run suggested that the

GA exploits longer individuals early in a run when extra resources are helpful in searching for potential solutions then fine tunes to shorter individuals later in the run in response to parsimony pressure.

Group-wise location independence suffers from some of the same weaknesses as bit-wise location independence. There is again the notion that each building block has a specific meaning or contribution and, as a result, a value for every building block is needed in order for a solution to be formed. Missing and duplicate building blocks require special treatment. In addition, the final arrangement of building blocks can be sensitive to early decisions in a GA run. Bad decisions early in a GA run could result in arrangements in which it is impossible to encode all building blocks.

2.3. Alternative location independent atomic units

Both the bit-level and group-level use fairly general atomic units that can be used to encode more complex information. An alternate method is to make the atomic units themselves more complex. A number of applications have successfully used such representations. Each unit is a larger, more comprehensive idea or value than a bit, each variation of the unit may be considered a "character", and the alphabet size is typically much larger than two. The collection of units on an individuals are "expressed" and work together to create a solution. Information not encoded on an individual cannot be and is not expressed. The ordering of the units on a GA individual should not effect the fitness of the individual.

A common example of such a representation is the evolution of rule sets [15, 38]. Each individual represents an entire rule set for controlling robots and the basic atomic units are condition-action rules. Crossover exchanges groups of rules, but will not fall within a rule. Mutation and other related operators change individual rules. Each individual (rule set) is evaluated on the behavior of a robot using that rule set. Rules are selected to fire based on the similarity of the condition clause of a rule with the current conditions of the robot. Thus, the collection of rules in a rule set is expected to affect its performance. The ordering of the rules in a rule set has absolutely no effect on the interaction of the evolved rules when applied to a problem.

Another related representation is Franceschini et al.'s [10] work on disaggregation. A GA is used to evolved the configuration of a group of entities which satisfies pre-specifed rules. As there is no distinction among the entities, each GA individual simply encodes a vector of coordinates. The order in which these coordinates are arranged on an individual had no effect on the solution. The fitness evaluation simply

considers the content or collection of coordinates encoded. The existence of a coordinate affects the solution, the location of that coordinate does not.

Representations which use complex atomic units can be very effective at exploiting the advantages of location independence. Their implementation, however, will often require development of complex representations and problem specific operators.

2.4. Genotype to phenotype transformations

Genotype to phenotype transformations or mappings have also been studied as a way to improve GA problem representations. Such transformations distinguish between the search space and the solution space of a problem and allow for the development of a search space that is amenable to the GA search process when a solution space is not. One might take advantage of such a distinction to develop genotypes which, for example, simplify or otherwise modify the search space, include redundancies which could provide the GA with "backup copies" or a form of memory, include neutral classes (classes of multiple genotypes that map to a single phenotype) to increase connectivity in the solution space, or eliminate the generation of invalid solutions (and the need for repair or specialized operators). Some examples of approaches to studying genotype to phenotype transformations include the following.

Kargupta and Park [20, 21] examine transformations of a fitness function into Fourier space as a way distributing fitness evaluation and show that for a given class of functions, such evaluations may be performed in polynomial time. Redundant representations, which investigate mappings where only a subset of the encoded information on an individual is expressed, include diploidy and dominance mapping [25, 27, 31, 39], the structured GA [8], as well as many of the approaches discussed in the previous sections. A common side effect of redundant representations is the emergence of neutral variants. Studies of mappings that allow neutral variants indicate benefit in terms of diversity and efficiency [2, 30, 34]. Genetic programming studies have investigated dynamic evolution of the genetic code (which essentially dictates the genotype to phenotype mapping) [22, 23, 29]. A GA has also been used to develop simulations of gene expression and regulation in artificial cellular environments [24].

3. The proportional GA representation

If one considers the DNA level of encoding information in a genome, it appears to be very similar to a traditional CGA encoding. The system is looking for a desired sequence or pattern of "characters" whose encoding provides a desired value. Characters are chosen from a fixed alphabet. The biological alphabet size happens to be four; the computational alphabet size is typically two. A correctly ordered sequence contributes positively to an individual's fitness. An incorrect sequence contributes partial or no credit to an individual's fitness. Order-based encoding is an essential component of DNA information representation. All of the basic units of information that DNA can represent, i.e. genes, are encoded as ordered sequences.

If, however, we make the assumption that these basic units of information exist, and move up to a genome level viewpoint, we see an entirely different picture of what is being encoded. A biological genome encodes the collection of genes necessary to create a life form. A gene is expressed when it is translated into a protein product and can only be expressed if it exists on the genome. Multiple copies of a gene produces more of the protein for which it encodes. The interaction of the expressed proteins within a cell makes life possible. At this level, the key point is not the order of the encoded information, but the combination or content of the encoded information on a genome. This viewpoint requires, of course, that all of the "machinery" or encoding for the components (genes) that make up the content is already there. By focusing on a more abstract view of the genome as a description of what is available to interact within a cell, however, the existence or nonexistence of a gene has a much greater impact than the location of a gene.

The PGA representation is based on this idea that it is the content rather than the order of the encoded information that matters. We extend the idea of location independent atomic units from section 2.3 to a more generally usable encoding for integer and floating point numbers. Specifically, the PGA assigns one or more unique characters to each parameter or component of a solution. The value of a parameter is determined from the proportion of the characters assigned to that parameter as compared with the total number of characters in the individual, or from the relative proportions of the assigned characters of that parameter. Thus, characters that exist are "expressed" and, consequently, interact with other expressed characters. Characters that do not exist are "not expressed" and simply do not participate in the interactions of the expressed characters. This representation may be further extended with the addition of *non-coding* characters that are not

Table I. PGA1 character assignment in a five user resource allocation problem.

User U	Character assignment, $char(U)$
U_1	a
U_2	b
U_3	c
U_4	d
U_5	e

associated with any parameters. These non-coding regions are beneficial for fine tuning purposes.

3.1. PGA1

Initial examination of this representation suggests that it is a perfect fit for resource allocation problems. Suppose we have a common pool of resources that must be divided among n users. With a PGA, we would use an n-character alphabet with one character assigned to each user. Each individual, regardless of length represents the total available resources. The proportion of each user's character on an individual gives the percentage of resource allocated to each user. We will call this representation strategy PGA1.

For example, given a five user resource allocation problem, a PGA would require a minimum alphabet size of $a = 5$. We can assign characters as shown in table I. The percentage of total resource allocated to each user, $U_i, i = 0, ..., a-1$ is calculated by the equation

$$P_{PGA1}(U_i) = \frac{number\ of\ char(U_i)\ on\ individual}{length\ of\ individual}.$$

As a result, the sum of all of the allocated resources must equal 1.0 (100% of the resources). A typical individual such as

```
accaebbdbeeddbabbddebaabbdddebebadbecabebbedaacced
```

would generate the allocations shown in table II. This individual is 50 characters long. Note that the location of the characters play absolutely no role in the evaluation of the individual. Thus, multiple individuals may generate the same solution and the individual

```
aaaaaaaaabbbbbbbbbbbbbbbcccccdddddddddddeeeeeeeeee
```

encodes the exact same solution as the first individual above.

Table II. Allocation of resources specified by example individual of length 50.

User U	Number of $char(U)$	Allocation
U_1	9 a's	0.18
U_2	15 b's	0.3
U_3	5 c's	0.1
U_4	11 d's	0.22
U_5	10 e's	0.2

Table III. PGA2 character assignment in a five value problem.

Value V	$postive_char(V)$	$negative_char(V)$	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

3.2. PGA2

Unfortunately, many problems cannot be represented as pure resource allocation problems where the sum of the components is constrained to a fixed value. As a result, additional 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 $a = 5$ parameter values, $V_i, i = 0, ..., a-1$ 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 III. The number of "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 $i = 0, ..., a - 1$ 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

```
AccBdDeeEbAbBDEccaAAAEebbEEECCDbbbABCDEedcbaAAddbA
```

Table IV. Allocation of resources specified by example individual of length 50.

Value V	Number of $postive_char(V)$	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

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

```
AAAAAAAAAaaBBBbbbbbbbbbCCCcccccDDDDddddEEEEEEEeeee
```

also evaluates to the values shown in table IV.

3.3. PGA3

We attempt to simplify PGA2 by modifying the equation for $pct(V_i)$. In the new method called PGA3,

$$pct(V_i) = \begin{cases} \frac{positive_char(V_i)}{negative_char(V_i)} & \text{if } positive_char(V_i) < negative_char(V_i) \\ \frac{negative_char(V_i)}{positive_char(V_i)} & \text{otherwise} \end{cases}$$

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

$$P_{PGA3}(V_i) = V_{i,min} + pct(V_i) \times (V_{i,max} - V_{i,min}).$$

With PGA3, both individuals

```
AccBdDeeEbAbBDEccaAAAEebbEEECCDbbbABCDEedcbaAAddbA

AAAAAAAAAaaBBBbbbbbbbbbCCCcccccDDDDddddEEEEEEEeeee
```

generate the parameter values shown in table V. Initial experiments suggest that the behavior of PGA2 and PGA3 are very similar.

Table V. Allocation of resources specified by example individual of length 50.

Value V	Number of $postive_char(V)$	Number of $negative_char(V)$	$pct(V)$	Expressed value
V_1	9 A's	2 a's	2/9	2.22
V_2	3 B's	9 b's	3/9	3.33
V_3	3 C's	5 c's	3/5	6.0
V_4	4 D's	4 d's	4/4	1.0
V_5	7 E's	4 e's	4/7	5.71

3.4. Additional extensions

The adaptability of all three of the PGA variations described above can be further enhanced in two ways.

3.4.1. *Variable length individuals*

Because of the location independence of the PGA representation, it can easily be modified to work with variable length individuals. Variable length individuals allows a PGA to control the amount of computational effort it expends during a run. Using shorter individuals requires less computational effort. In addition, this modification would allow a PGA to control the resolution at which it encodes values. Longer individuals would allow finer resolution. The minimization of computational effort would likely need to be balanced against the need for adequate resolution.

3.4.2. *Non-coding regions*

Non-coding regions can easily be added by including, as part of a PGA's alphabet, characters that are not assigned to any parameter values. These characters would not be directly involved in encoding information on an individual; however, they may affect the values represented by an individual by changing the overall length of individuals. Non-coding regions are expected to help a PGA "fine-tune" the values encoded on its individuals. In a fixed length system where the length is unsuitable for the required resolution, non-coding regions may be introduced to compensate for suboptimal length. This effect is expected to be more prevalent in PGA2 and PGA3 than PGA1 because of the nature of the problems on which PGA1 can be applied.

Table VI. Possible solutions using a PGA with binary alphabet and length 8.

Number of 0's	Number of 1's
8	0
7	1
6	2
5	3
4	4
3	5
2	6
1	7
0	8

4. The issue of resolution

We would like to compare the performance of the PGA with that of a CGA; however, their fundamentally different representations make it difficult to determine what would be a fair comparison. For example, a CGA using a binary alphabet and individuals of length 8 can encode $2^8 = 256$ different solutions. Table VI shows that a PGA using the same alphabet and length can encode 9 unique solutions. Obviously it will be easier to find a solution with the PGA; however, the PGA is representing a much smaller and coarser solution space, i.e. trying to solve an easier problem. Increasing the alphabet size of the PGA to three increases the number of unique solutions to 45 (see Table VII). Obviously, the selected length and alphabet size affects the resolution of the solution spaces that each algorithm can represent.

We assert that, to fairly compare two search algorithms, they must search for a solution from among the same number of potential solutions, that is, within equal sized solution spaces. As a result, we will focus on determining metrics that would allow a CGA and a PGA to represent a problem at the same resolution. To do so, we analyze the effects of the multiset approach of the PGA on its genotype-phenotype mapping and on the formation of *neutral classes*; give an equation for determining the CGA and PGA solution structures required to represent the same solution space; and show that if we use those sizes in a comparative random sampling analysis, the probabilities of finding a solution, on average, are the same for a CGA and a PGA.

Table VII. Possible solutions using a PGA with alphabet size 3 and length 8.

Number of A's	B's	C's	Number of A's	B's	C's	Number of A's	B's	C's
8	0	0	3	5	0	1	6	1
7	1	0	3	0	5	1	1	6
7	0	1	3	4	1	1	5	2
6	2	0	3	1	4	1	2	5
6	0	2	3	3	2	1	4	3
6	1	1	3	2	3	1	3	4
5	3	0	2	6	0	0	8	0
5	0	3	2	0	6	0	0	8
5	2	1	2	5	1	0	7	1
5	1	2	2	1	5	0	1	7
4	4	0	2	4	2	0	6	2
4	0	4	2	2	4	0	2	6
4	3	1	2	3	3	0	5	3
4	1	3	1	7	0	0	3	5
4	2	2	1	0	7	0	4	4

4.1. Notation

We will use the terms *individual* and *genotype* interchangeably to refer to fixed length strings over an alphabet. Typically, CGA individuals are strings over a binary alphabet while PGA individuals are strings over a higher-arity alphabet.

We introduce the following notation: *search (genotype) space*, $\mathcal{G}_{\langle l,\Sigma\rangle} = \{g_i\}$, the set of all genotypes, g_i, of length l over alphabet Σ; and *solution (phenotype) space*, $\mathcal{P} = \{p_j\}$, 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 $\mathcal{M} : \mathcal{G}_{\langle l,\Sigma\rangle} \mapsto \mathcal{P}$, hence $p_j = \mathcal{M}(g_i)$

Let $\mathcal{A}$ be a CGA with individuals of length $l_{\mathcal{A}}$ over an alphabet $\Sigma_{\mathcal{A}} = \{\sigma_1, \sigma_2, \ldots, \sigma_{n_{\mathcal{A}}}\}$. Let the cardinality of the alphabet be $|\Sigma_{\mathcal{A}}| = n_{\mathcal{A}}$. Let $\mathcal{A}$'s search space, solution space, and mapping be denoted by $\mathcal{G}_{\langle l_{\mathcal{A}},\Sigma_{\mathcal{A}}\rangle}$, $\mathcal{P}_{\mathcal{A}}$, and $\mathcal{M}_{\mathcal{A}}$, respectively.

[1] Multisets—also known as *bags*—denoted on this paper by {|, , , |}, are analogous to sets, except that they may contain multiple copies of identical elements. The number of occurrences of an element in the multiset is called the *multiplicity* of the element.

Similarly, let $\mathcal{B}$ be a PGA with length $l_{\mathcal{B}}$ and alphabet $\Sigma_{\mathcal{B}} = \{\sigma_1, \sigma_2, \ldots, \sigma_{n_{\mathcal{B}}}\}$ with cardinality of $|\Sigma_{\mathcal{B}}| = n_{\mathcal{B}}$. Let $\mathcal{B}$'s search space, solution space and mapping be denoted accordingly by $\mathcal{G}_{\langle l_{\mathcal{B}}, \Sigma_{\mathcal{B}} \rangle}$, $\mathcal{P}_{\mathcal{B}}$, and $\mathcal{M}_{\mathcal{B}}$.

4.2. The PGA mapping

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, and leads to a solution space that is smaller than its corresponding search space. The number of elements in 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. In general, multiple strings will map onto the same multiset. The only exceptions are those strings that contain only one kind of element; i.e., for $l = 4$ and $\Sigma = \{a, b\}$, the string "*aaaa*" is the only one that maps onto the multiset $\{|a, a, a, a|\}$, while all of the following strings: "*aaab*", "*aaba*", "*abaa*", and "*baaa*", map onto the same multiset, $\{|a, a, a, b|\}$.

Using basic combinatorics, we can calculate the exact number of strings that map to each multiset as function of their multiplicities. We can also calculate the total number of such multisets for a given search space which is precisely the size of the PGA phenotype space.

Let $\eta(p_j)$ denote the number of strings on the genotype space which map to the multiset p_j. It is easy to show that:

$$\eta(p_j) = \frac{(l_{\mathcal{B}})!}{\prod_{i=1}^{n_{\mathcal{B}}} (|p_j|_{\sigma_i})!} \tag{1}$$

where, $p_j = \{|\varsigma_{j1}, \varsigma_{j2}, \ldots, \varsigma_{jk}, \ldots, \varsigma_{jl_{\mathcal{B}}}|\}$ is the jth multiset in $\mathcal{P}_{\mathcal{B}}$; $\varsigma_{jk} \in \Sigma_{\mathcal{B}}$ for $k = 1, \ldots, l_{\mathcal{B}}$; $|p_j|_{\sigma_i}$ denotes the multiplicity (number of occurrences) of σ_i in the multiset p_j; and $\sigma_i \in \Sigma_{\mathcal{B}} = \{\sigma_1, \sigma_2, \ldots, \sigma_{n_{\mathcal{B}}}\}$.

The total number of multisets produced by the mapping, $\mathcal{M}_{\mathcal{B}} : \mathcal{G}_{\langle l_{\mathcal{B}}, \Sigma_{\mathcal{B}} \rangle} \mapsto \mathcal{P}_{\mathcal{B}}$, is $\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}$.
As the set of all the multisets obtained by $\mathcal{M}_{\mathcal{B}}$ constitute the phenotype space $\mathcal{P}_{\mathcal{B}}$, the size of this space, denoted $|\mathcal{P}_{\mathcal{B}}|$, is given by:

$$|\mathcal{P}_{\mathcal{B}}| = \binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}} \tag{2}$$

4.3. Neutral classes

A *neutral class* is defined as the set of genotypes that maps to the same phenotype. The number of neutral classes as well as their corresponding sizes are determined by the mapping between spaces, $\mathcal{M}$. Each element of the phenotype space has a unique associated neutral class and vice versa. Note that the set of neutral classes is a partition of the genotype space.

For the PGA mapping, $\mathcal{M}_{\mathcal{B}}$, the size of the neutral class associated with the phenotype p_j is given by $\eta(p_j)$ on Equation (1), and the number of neutral classes formed on the genotype space by this mapping, equals the size of the phenotype space, $|\mathcal{P}_{\mathcal{B}}|$, given by Equation (2).

4.4. Resolution analysis

Let us consider a problem in which we are searching for a single parameter value with a given resolution, i.e., searching for a number between 0 and 100 with a resolution of 0.1. In this problem, the solution space would need to consist of at least $100 \times 10 = 1000$ elements to accomodate the required resolution of the search. As a result, an evolutionary algorithm applied to this problem must be able to represent at least 1000 elements in its solution space to find an accurate solution. In order to fairly compare two algorithms set to solve this problem, both of them should have the same solution space size.

Therefore, in order to compare CGA $\mathcal{A}$ and PGA $\mathcal{B}$, their solutions spaces should be of the same size, that is:

$$|\mathcal{P}_{\mathcal{A}}| \equiv |\mathcal{P}_{\mathcal{B}}| \tag{3}$$

The size of the phenotype space for CGA $\mathcal{A}$ is:

$$|\mathcal{P}_{\mathcal{A}}| \equiv (n_{\mathcal{A}})^{l_{\mathcal{A}}} \tag{4}$$

We use (2) and (4) to rewrite (3) and obtain:

$$(n_{\mathcal{A}})^{l_{\mathcal{A}}} = \binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}} \tag{5}$$

Equation (5) is important because it describes the relationship between the lengths and alphabet sizes of an arbitrary CGA and PGA that would ensure resulting solution spaces of equal size. We will use this equation to set up our comparative experiments on Section (6).

4.5. Statistical analysis

Consider the point $p_s \in \mathcal{P}$ to be a distinguished point on the phenotype space. Let us call it the solution point.[2]

Let $g_s \in \mathcal{G}$ be a point in the search space that maps to p_s. Let $\mathcal{N}(p_s)$ be the set of all such points, $g_{s_i} \in \mathcal{G}$, that map to p_s. Therefore, $\mathcal{N}(p_s)$ is the *neutral class* associated with p_s.

Since there is exactly one neutral class associated with each element of $\mathcal{P}$, we have $|\mathcal{P}|$ neutral classes. The number of elements in each neutral class, $|\mathcal{N}(p_j)|$, depends on the specifics of the mapping $\mathcal{M}$.

4.5.1. *Probabilities for finding a solution*

The probability of selecting a genotype that maps to p_s by random sampling is given by:

$$Pr(p_s) = \frac{Number\ of\ genotypes\ that\ map\ to\ p_s}{Total\ number\ of\ genotypes}$$

which can be rewritten as:

$$Pr(p_s) = \frac{|\mathcal{N}(p_s)|}{|\mathcal{G}_{\langle l,\Sigma\rangle}|} \tag{6}$$

where $|\mathcal{N}(p_s)|$ is the size of the neutral class associated with p_s.

Let us consider again CGA $\mathcal{A}$ and PGA $\mathcal{B}$. ¿From the previous discussion we know that the number of the neutral classes is the same as the number of elements in $\mathcal{P}$. As a result, $|\mathcal{P}_\mathcal{A}|$ and $|\mathcal{P}_\mathcal{B}|$ give us the number of neutral classes for $\mathcal{A}$ and $\mathcal{B}$.

The number of strings in each class for $\mathcal{A}$ is trivial to compute. Since the mapping, $\mathcal{M}_\mathcal{A}$, is one-to-one and onto, there is exactly one string in each class. For $\mathcal{B}$, the size of a neutral class is given by Equation (1) and depends on the multiplicity of the associated multiset. This multiplicity is the same as the multiplicities of each of the strings in the class defined by the multiset. Hence, we can expand out definition of η to account for multisets as well as for strings in the following way:

$$\eta(g_{s_k}) = \frac{(l_\mathcal{B})!}{\prod_{i=1}^{n_\mathcal{B}} (|g_{s_k}|_{\sigma_i})!} \tag{7}$$

where $g_{s_k} \in \mathcal{G}_{\langle l_\mathcal{B},\Sigma_\mathcal{B}\rangle}$.

The size of the neutral class associated with p_s is given by:

$$|\mathcal{N}(p_s)| = \eta(g_{s_k}) \tag{8}$$

[2] This point represents the phenotype with highest fitness, given a fitness function defined over the space $\mathcal{P}$

We use Equation (6) to obtain the probabilities $Pr_{\mathcal{A}}(p_s)$ and $Pr_{\mathcal{B}}(p_s)$:

$$Pr_{\mathcal{A}}(p_s) = \frac{1}{(n_{\mathcal{A}})^{l_{\mathcal{A}}}} \quad (9)$$

$$Pr_{\mathcal{B}}(p_s) = \frac{\eta(g_{s_k})}{(n_{\mathcal{B}})^{l_{\mathcal{B}}}} \quad (10)$$

4.5.2. *The PGA Hypothesis*
In order for PGA $\mathcal{B}$ to have a better or equal probability of finding a solution (p_s) than CGA $\mathcal{A}$, the following inequality must hold:

$$Pr_{\mathcal{B}}(p_s) \geq Pr_{\mathcal{A}}(p_s) \quad (11)$$

We call Equation (11) the *PGA hypothesis.* It simply states that the probability of randomly selecting a genotype that maps to the solution is equal or higher in a PGA than in a CGA.

Substituting (9) and (10) into (11) and rearranging the terms we get:

$$\eta(g_{s_k}) \geq \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{(n_{\mathcal{A}})^{l_{\mathcal{A}}}} \quad (12)$$

which is a general condition for the PGA hypothesis to hold. This equation depends on parameters such as length and alphabet cardinality, as well as depends on the choice of the desired solution p_s.

Observe that for the case in which $\mathcal{A}$ and $\mathcal{B}$ have the same sized search space, $|\mathcal{G}_{\langle l_{\mathcal{A}},\Sigma_{\mathcal{A}}\rangle}| \equiv |\mathcal{G}_{\langle l_{\mathcal{B}},\Sigma_{\mathcal{B}}\rangle}|$, Equation (12) reduces to:

$$\eta(g_{s_k}) \geq 1 \quad (13)$$

which always holds. Hence, in this case, subject to the choice of p_s, a PGA always has a better or equal chance of finding a solution than a CGA.

In the case where $|\mathcal{P}_{\mathcal{A}}| \equiv |\mathcal{P}_{\mathcal{A}}|$, Equation (12) reduces to:

$$\eta(g_{s_k}) \geq \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}} \quad (14)$$

In this case, either the PGA or the CGA could have the advantage, depending on the choice of p_s.

4.5.3. *Average probabilities over all solutions*
Since the previous results depend heavily on the choice of p_s, let us analyze the average of all possible choices of p_s. Note that p_s affects

the above equations by changing the size of the associated neutral class. Consequently, to analyze the average behavior of all possible choices of p_s, we will compute the average size of all of the neutral classes (all possible choices of p_s).

Let $\hat{\eta}$ denote the average size of all neutral classes for a given search space:

$$\hat{\eta} = \frac{\sum_{\{g_r \mid \mathcal{M}_{\mathcal{B}}(g_r) \neq \mathcal{M}_{\mathcal{B}}(g_t) \,\forall\, r \neq t\}} \Big[\eta(g_k)\Big]}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}}$$

Since the sets of all neutral classes are a partition of the search space, the summation of all of the neutral class sizes is the size of the space:

$$\sum_{\{g_r \mid \mathcal{M}_{\mathcal{B}}(g_r) \neq \mathcal{M}_{\mathcal{B}}(g_t) \,\forall\, r \neq t\}} \Big[\eta(g_k)\Big] = |\mathcal{G}_{\langle l_{\mathcal{B}}, \Sigma_{\mathcal{B}} \rangle}|$$

Thus, $\hat{\eta}$, the average size of all neutral classes over a search space, is given by the following equation as function of the parameters defining the space itself:

$$\hat{\eta} = \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}} \tag{15}$$

At this point, we can replace in our previous equations the specific size of a neutral class, η, with the average size over all possible neutral classes in a search space, $\hat{\eta}$, to yield results that are independent of p_s.

Replacing η with $\hat{\eta}$ on Equation (12) and rearranging terms we obtain the condition for the PGA Hypothesis to hold in the average case:

$$(n_{\mathcal{A}})^{l_{\mathcal{A}}} \geq \binom{n_{\mathcal{B}} + l_{\mathcal{B}} - 1}{l_{\mathcal{B}}} \tag{16}$$

Equation (16) is independent of the choice of p_s (solution point) because η has been averaged over all possible choices of p_s. Note from this equation that the PGA hypothesis will hold if and only if:

$$|\mathcal{P}_{\mathcal{B}}| \leq |\mathcal{P}_{\mathcal{A}}| \tag{17}$$

Equation (17) tells us that, in order for a PGA to have an advantage, the PGA solution space cannot be larger than the CGA counterpart. But equation (3) states that both solution spaces should be equal for a fair comparison. Hence, the probability that a PGA will find a solution is, on average, equal to the probability that a CGA will find a solution

when both solution spaces are equal in size. In order for a PGA to have a greater probability of finding a solution, the PGA must be searching in a solution space that is smaller than that of the competing CGA.

Equation (13) considers the case of $|\mathcal{G}_{\langle l_{\mathcal{A}},\Sigma_{\mathcal{A}}\rangle}| \equiv |\mathcal{G}_{\langle l_{\mathcal{B}},\Sigma_{\mathcal{B}}\rangle}|$. Using the average value of η, $\hat{\eta}$, we obtain:

$$\hat{\eta} \geq 1$$

Replacing $\hat{\eta}$ with its value from Equation (15), we get:

$$\frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}} \geq 1$$

This equation holds only for the inequality. Thus, the PGA hypothesis holds as an inequality. As a result, when a PGA and CGA have search spaces of same size, a PGA will have, on average, a better chance of finding a solution than a CGA.

Equation (14) examines the case where $|\mathcal{P}_{\mathcal{A}}| \equiv |\mathcal{P}_{\mathcal{A}}|$. Using $\hat{\eta}$ we can now obtain:

$$\hat{\eta} \geq \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}}$$

$$\frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}} \geq \frac{(n_{\mathcal{B}})^{l_{\mathcal{B}}}}{\binom{n_{\mathcal{B}}+l_{\mathcal{B}}-1}{l_{\mathcal{B}}}}$$

In this case, the equation holds only for the equality. Thus, the PGA hypothesis holds as equality. With solution spaces of the same size, a PGA has, on average, similar chances of finding a solution as a CGA.

5. Biological support

The primary inspiration for the PGA comes from the biological idea of gene expression. There is a fundamental difference in what and how information is encoded in biological evolutionary systems and that encoded in computational evolutionary systems. Computational encodings tend to focus on the order of information. Biological encodings also emphasize the existence of information.

Ordered encodings are, of course, an important method by which biological systems store and express information. Proteins are defined by the order of their component amino acids. Genes are defined by the order of their component DNA molecules. The order in which genes

appear on a genome can be important, for example, the mammalian β-globin gene cluster consists of one embryonic, two fetal, and two adult genes arranged on the chromosome in the order in which they are expressed during the development of the organism [35].

From the overall genome viewpoint, however, the job of a genome is to specify the collective and relative amounts of genes needed within a cell. The expressed genes produce proteins which interact to create life. The type and amount of protein available for interaction directly affects the expressed phenotype of a cell or organism. Biological genomes are not divided into fields, each of which is assigned to produce a specific protein. Rather, they are more open-ended, with genes appearing where ever a viable encoding is found. The number of genes that comprise a solution is bounded only by the genome size and the number of different "encodings" is virtually endless. This viewpoint suggests that the existence or nonexistence of a gene has as much or more impact than the location of a gene. After all, the location of a gene can only matter if it exists.

What is interesting to us is the fact that information is encodable purely as variations in the number of genes and completely independently of any ordering. Presumably, this is possible only if there are external rules that govern the interaction of the components produced. These rules need not be complex to produce interesting behavior, as is evident from our simple PGA equations. In our arguments here, we make the generalization that the actual locations of the genes do not affect their expression. In reality, the process is much more complicated, and gene location and gene regulation add a great deal of complexity to this system. We assert that the primary factor here is the existence or non-existence of a gene on a genome. The regulation of gene expression and gene location is secondary factor that extends the complexity of living systems to the next level, but whose analogy to GAs we will save for future work. For now, we will focus only on the analogy to simple gene expression.

There are clear examples in biology which indicate that the existence of a gene allows it to produce a required protein regardless of its location. Curtis [6] gives an example where a genetic defect (deletion) is masked by an extra copy of the missing gene on a different chromosome. The defect is expressed in the offspring which do not have an extra copy of the missing gene, possibly due to receiving that part of the chromosome from the other parent. Studies on the expression of X chromosome genes support the importance of accurate gene expression. Female mammals have two X chromosomes to male mammals' XY chromosomes. To prevent females from receiving a double dosage of

X chromosome genes, one X chromosome is suppressed in all female cells [3, 28].

With strong roots in genetic and evolutionary biology, the GA is essentially an abstract model of the process of evolution. As a result, GA researchers often look to biology for inspiration and direction. There have been concerns about random, haphazard addition of biologically-inspired features to a GA [7] ultimately producing a patchwork algorithm consisting of arbitrarily selected, sometimes misinterpreted components. Instead of a new addition or modification to the CGA, what we propose here is more of a complete revision of GA problem representation. In a sense, much of the GA community has "converged" on order-based encodings due to the success and straightforwardness of this encoding method. We are attempting to take a step back via "hypermutation" and begin exploring a new branch of representations that focus on gene expression rather than gene order.

6. Experimental details

We compare the three variations of the PGA described in section 3 with a CGA. Specific questions that we address in our experiments include:

1. How does a PGA compare to a traditional CGA? Can a PGA perform at least as well as a CGA?
2. Does a PGA use non-coding regions as a means of "fine-tuning" the values it encodes?
3. Can a PGA regulate the length of its individuals to minimize computational effort while maximizing performance?
4. Does a PGA encourage the formation of building blocks?

6.1. Test problems

We study three types of problems in the experiments described here.

1. Resource allocation
2. Number matching
3. Symbolic regression

All of these problems require a GA to find five values.

The resource allocation problem is a search for a series of values that sum to a predefined value. This type of problem appears to be a

Table VIII. GA parameter settings.

Parameter		Value
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

perfect fit for the PGA encoding. We compare the behavior of PGA1 with a CGA on this problem.

The second two problems search for a series of numbers that do not have to sum to a predefined value. Many real world problems can be mapped to one of these general problem types. In number matching, all values have equal weight. In symbolic regression, some values may have greater affect on fitness. We compare both PGA2 and PGA3 with a CGA on these two problems.

6.2. GA configuration

Table 6.2 gives the parameter setting used in these experiments. Each experiment was run 100 times and the results averaged over all runs.

Our CGA uses a binary representation consisting of a field of eight bits for each of the five values for a total length of 40 bits. We can use Equation (5) from Section 4 to calculate the appropriate PGA lengths for our comparisons. For each of our five values, we have $n_{\mathcal{A}} = 2$ and $l_{\mathcal{A}} = 8$, as our CGA is binary and eight bits are allocated per value. For the PGA we have $n_{\mathcal{B}} = 2$ as each parameter is represented with an alphabet size of two [3]. Entering these values into Equation (5) and solving for $l_{\mathcal{B}}$, we obtain $l_{\mathcal{B}} = 255$. Therefore, 255 is the equivalent length that a PGA would require to encode a single parameter value with the same resolution as the CGA. As a result, the PGA representation would need a genome length of $255 \times 5 = 1275$ to encode the same resolution as the CGA.

In addition to the simple PGA representation, we also test extensions which include non-coding regions and variable length individuals. We test a total of six variations of each PGA:

[3] This is clear for PGA2 and PGA3 since the alphabets are separate; however for PGA1 we also have two groups of characters for each value: one distinguished character and one group undistinguished characters—the rest of the alphabet.

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.

6.3. Resource allocation

The resource allocation problem involves the allocation of a fixed pool of resources among five users.

The PGA1 encoding is described in section 3.1. A CGA individual is decoded by first calculating the binary values encoded in each field. Each value is then divided by the sum of all of the values to produce a proportion between 0.0 and 1.0.

Given $a = 5$ target values or proportions, $T_i, i = 0, ..., a-1$ where $\sum T_i = 1.0$, and a users $U_i, i = 0, ..., a-1$, the encoded proportion assigned to each user is given by $P_{PGA1}(U_i)$ (see section 3.1). We first calculate the *ratio* of each corresponding pair:

$$ratio(i) = \begin{cases} \frac{T_i}{P_{PGA1}(U_i)} & \text{if } T_i < P_{PGA1}(U_i) \\ \frac{P_{PGA1}(U_i)}{T_i} & \text{otherwise} \end{cases}$$

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

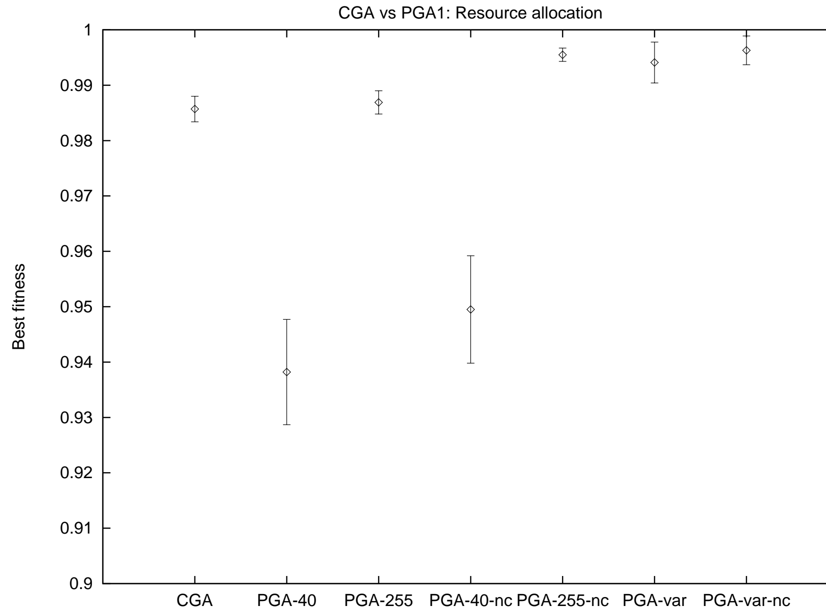

Figure 3. CGA -vs- PGA1 on resource allocation: Fitness of best solution found averaged over 100 run and 95% confidence intervals.

of all ratios:

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

A perfect match gives the maximum score of 1.0.

6.3.1. *Randomly generated target allocations*

Our first experiment simply compares the ability of a CGA and PGA1 to find a set of target proportions. To eliminate any unexpected bias, we randomly generate a set of target values, T_i where $\sum T_i = 1.0$, at the beginning of each run. As a result, each of the 100 runs that make up this experiment are searching for a different set of target values. Figure 3 shows the average best solutions obtained from each algorithm tested with 95% confidence intervals. The CGA performs well, with an average best fitness of 0.9857. All of the PGA variations except for the two using length 40 perform as well or significantly better than the CGA. PGA-255 performance is consistent with CGA performance despite its resolution penalty. Enhancing the PGA with either non-coding regions or variable length appears to significantly improve performance. Although the PGAs with length 40 performed significantly worse, they still provided respectable average fitnesses of 0.9382 and 0.9495. Table IX shows detailed results from this experiment. In addition to the best fitness value, we also track the number of

Table IX. Results of PGA1 versus CGA on finding randomly generated allocation values. Average (and standard deviation) over 100 runs.

Algorithm	Best fitness	Hits $\geq$ 0.99	Genome length	Time in sec.
CGA	0.9857 (0.0120)	53	40	13.96 (1.24)
PGA-40	0.9382 (0.0483)	2	40	12.08 (0.65)
PGA-255	0.9869 (0.0110)	59	255	15.75 (4.38)
PGA-40-nc	0.9495 (0.0496)	1	40	12.48 (1.02)
PGA-255-nc	0.9955 (0.0063)	94	255	16.12 (3.66)
PGA-var	0.9941 (0.0189)	91	1813.56 (585.68)	46.86 (16.77)
PGA-var-nc	0.9963 (0.0135)	95	1810.30 (606.37)	47.53 (13.72)

runs out of 100 in which a best fitness of 0.99 or higher was found — the number of "hits". The CGA finds fitness values above 0.99 in 53 out of 100 runs. Except for the PGAs of length 40, all other PGA runs are better able to achieve 0.99 fitness than the CGA. The variable length PGAs evolve significantly longer individuals than the fixed length values tested. These lengths, however, are not unreasonable given the actual required resolution of 1275. Applying parsimony pressure, as described in [4], results in average lengths that are about half of those shown above with essentally the same average best fitness value. All of our runs ran for 500 generations. The average clock time (in seconds) increases only slightly when genome length is increased from 40 to 255. Variable length PGAs, however, require significantly longer time to complete 500 generations of evolution.

6.3.2. *Formation of building blocks*

One of the expected advantages of the PGA encoding is that its complete location independence would encourage (or, at the very least, not impede) the formation of building blocks of related characters. Hypothetically, building blocks may be formed in two ways in the PGA. First, related characters (all of a particular character) are arranged in close proximity to minimize the chance of disruption by crossover. Second, all copies of each character are spread evenly across the entire individual making any particular segment of an individual essentially a building block (a miniature version of the full individual, albeit with coarser resolution). We examine individual PGA runs in detail to determine if such building block formation does occur.

We calculate the center of gravity and range for every character on every individual. The *center of gravity* (CoG) of a character is the average of the locations of all instances of that character on an

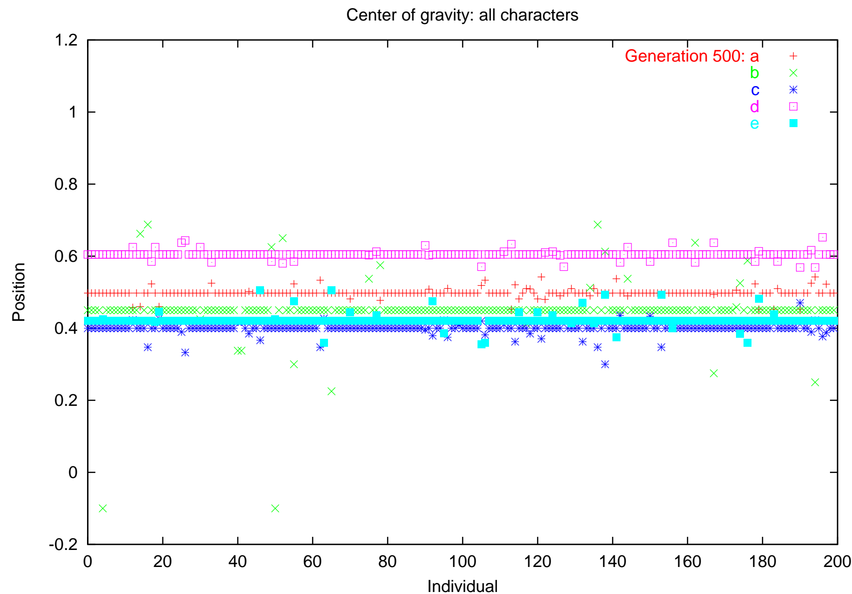

Figure 4. Center of gravity of characters on a population of PGA individuals.

individual. The *range* of a character is delimited by the two end-most copies of the character. We normalize the positions on an individual to range from 0.0 to 1.0 where 0.0 refers to one end of the individual and 1.0 refers to the opposite end. CoG values will then fall in between 0.0 and 1.0. Figures 4 and 5 show data from generation 500 of a PGA run. The x-axis of these plots indicate each individual from the generation. The y-axis indicates position on an individual with normalized position values (positions range from 0.0 to 1.0). The Figure 4 gives the CoGs of all five characters in this PGA. The Figure 5 shows the CoG and range of one specific character. By generation 500, all CoG values have evolved close to the center of the individuals. The range of the characters appears to be spread out over the entire individual. These results indicate that a PGA tends to form the second type of building block where characters are distributed across an entire individual. Examination of specific individuals supports this conclusion. Figure 6 shows an example individual from a PGA run. Similar, repeated sequences are aligned for easier recognition. A high degree of repetition and many similar building blocks are found throughout the individual. A shift in the alignment used in figure 6 by one character produces an entirely new set of building blocks containing approximately the same set of characters.

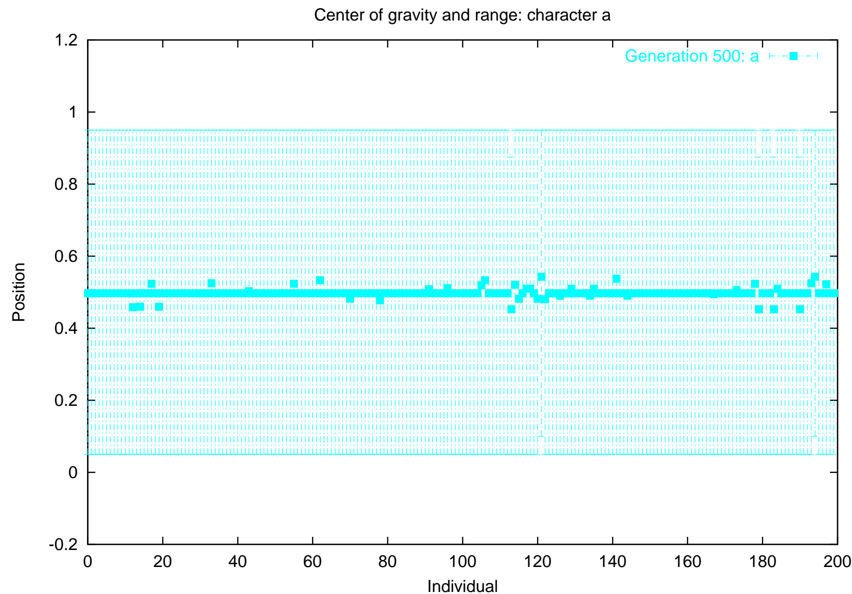

Figure 5. Range of characters on a population of PGA individuals.

6.3.3. *Regulation of length and resolution*

To further investigate the PGA's ability to regulate length and resolution, we next perform an experiment with specific target values where the target allocations have the ratio 1:2:4:2:1. The goals of this experiment are to test whether a PGA allocates an appropriate number of characters to account for the desired resolutions and to test whether a PGA will vary or minimize length in response to the resolution of the target values. The minimum number of characters required to encode this distribution in a PGA is 10 characters. As a result, the 40 length used in our fixed experiments should be able to find a perfect solution.

Table X shows the results from this experiment. The CGA is only able to find solutions above 0.99 fitness in 80% of its runs while all PGA runs were able to find solutions above 0.99 fitness. PGA-40 easily found the optimum distribution in all runs. Because a length that is a multiple of ten is required to achieve the optimum distribution, PGA-255 was not able to achieve a fitness of 1.0; A standard deviation of zero and examination of individual solutions, however, indicate that PGA-255 found the best possible solutions for its given length. Adding non-coding regions to PGA-255 alleviates this problem: PGA-255-nc finds all optimum solutions. Variable length PGAs, again, evolve significantly longer individuals but application of parsimony pressure reduces average lengths to around 240. Despite the large average length, two PGA-var runs generate solutions of length 10.

```
dcbdcfcdcfccbcbcccdgf
jbbcdhcddbcccdbdhgdbcccdbdhgdbcccbcdibbchhg
jacbccgcbdccdabiccdgdgbbcfecdchdcdcdcbbcchdgccdidhcdheccgcbhcjidhegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgicccbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgicccbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgicccbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgicccbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdeegccf
jcjbhgcdgcgbbdddcdbcdcddjhgiccgcbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgicccbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbcdccdcdebgccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbccgccccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbccgfcccgcbcdccdcdeegccf
jcbbhgcdgcgbbdddcdbcdcddjhgiccgcbccgccccgcbcdccdcdeegccd
jcbbhgcdgcgbbdddcdbcdcdijhgiccgebgcbccfbcdbdhgbhfcc
jcbacdcdhegccffcbbhdcdjdgbcccccjgjcifbe
```

Figure 6. Example of individual with repeated sequences.

Table X. Results of PGA1 versus CGA on finding fixed resolution allocation values. Average (and standard deviation) over 100 runs. A hit is a run that achieves the fitness of 0.99 or higher. All fitnesses are 0.95 or higher.

Algorithm	Best fitness	Hits ≥ 0.99	Genome length	Time in sec.
CGA	0.9911 (0.0009)	80	40	5.23 (0.45)
PGA-40	1.0 (0.0)	100	40	3.34 (0.48)
PGA-255	0.9922 (0.0)	100	255	8.27 (0.45)
PGA-40-nc	1.0 (0.0)	100	40	3.76 (0.43)
PGA-255-nc	1.0 (0.0)	100	255	8.36 (0.48)
PGA-var	0.9997 (0.0003)	100	1460.33 (849.38)	53.62 (8.87)
PGA-var-nc	1.0 (0.0)	100	1129.86 (933.11)	50.63 (14.96)

These results indicate that, given a problem whose resolution is achievable with the length used by a PGA, that PGA will able to find a perfect solution and will outperform a CGA. Furthermore, a PGA that has the ability to adjust its resolution, e.g. by varying length or amount of non-coding regions, will take advantage of these mechanisms to improve its solutions. Parsimony pressure can be used to minimize solution length with very little fitness penalty.

The suitability of the PGA to resource allocation problems is clearly demonstrated by the above studies. PGA is competitive and often outperforms a CGA on such problems. Most problems, however, cannot be encoded or interpreted as a resource allocation problem. Thus for the PGA to be useful, we must be able to apply it to other types of problems. In the next section we investigate PGA performance on two problems onto which many real world problems may be mapped.

6.4. Additional problems

There are many problems which may be described as a search for a vector of values. These values may be completely independent or may have dependencies among themselves. We now examine PGA performance on two representative problems. The first problem is a simple number matching problem in which we search for a set of five independent numbers. The second problem is a symbolic regression problem in which we search for a set interdependent numbers. We test two PGA encodings called PGA2 and PGA3 (described in section 3.2 and 3.3) on both of these problems.

6.4.1. *Number match*

The number match problem is a search for $a = 5$ independent values. The a 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 a target values, T_i, and a values decoded from a PGA individual, $P_{PGA}(V_i), i = 0, ..., a-1$, we first calculate the ratio of the smaller value divided by the larger value.

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

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

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

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

6.4.2. *Symbolic Regression*

Given a set of p data points, $d_i, i = 0, ..., p-1$, the symbolic regression problem is a search for $a = 5$ 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 a values are interdependent. Changes in a single value can affect the impact of other values.

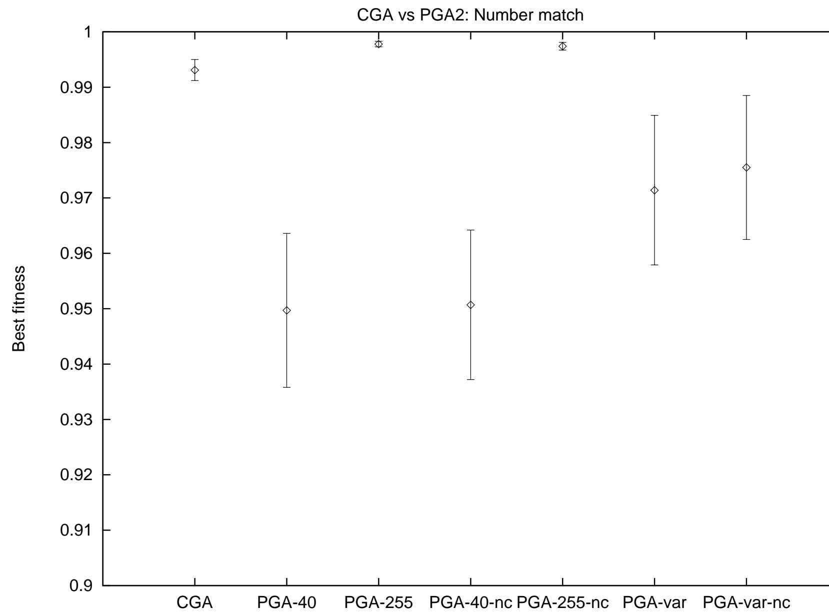

Figure 7. CGA -vs- PGA2 on number match: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

6.4.3. *Results*

Figure 7 and 8 compare the average best fitness and number of hits over 100 runs of a CGA and PGA2 variations on the number match problem. In both comparisons, PGA2 significantly outperforms the CGA only when it has length 255. A length of 40 results in a significant performance decrease; however, fitness values still reach well above 90%. The addition of non-coding regions has little effect. A variable length PGA2 performs somewhat better than PGA-40, but remains worse than the CGA. The fact that variable length is less beneficial with PGA2 than PGA1 is likely due to the more complex mapping from character proportions to function values in PGA2.

Figures 9 and 10 compare the average best fitness and number of hits over 100 runs of a CGA and PGA3 variations on the number match problem. Again, the best PGA performance occurs when PGA3 has length 255. Those PGA3 runs reach equivalent fitness values as the CGA and slightly outperform the CGA in hit count. PGA3 with length 40 performs significantly worse as do the variable length PGA3s.

Figures 11 to 14 show the corresponding data for the symbolic regression problem. The relative performances of the algorithms and the CGA performances are similar on both the symbolic regression and number match problems. All of the PGAs, however, perform significantly better on the symbolic regression problem. (Note the nar-

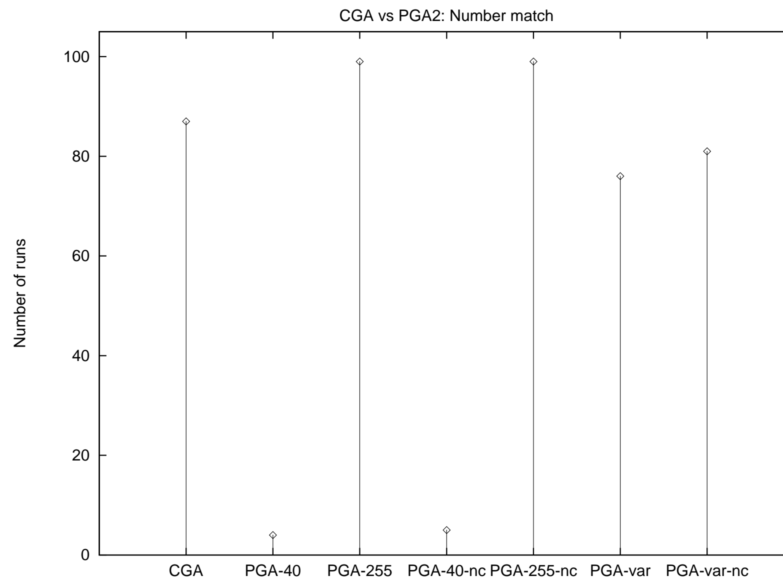

Figure 8. CGA -vs- PGA2 on number match: Number of runs out of 100 that find solutions with fitness 0.99 or higher.

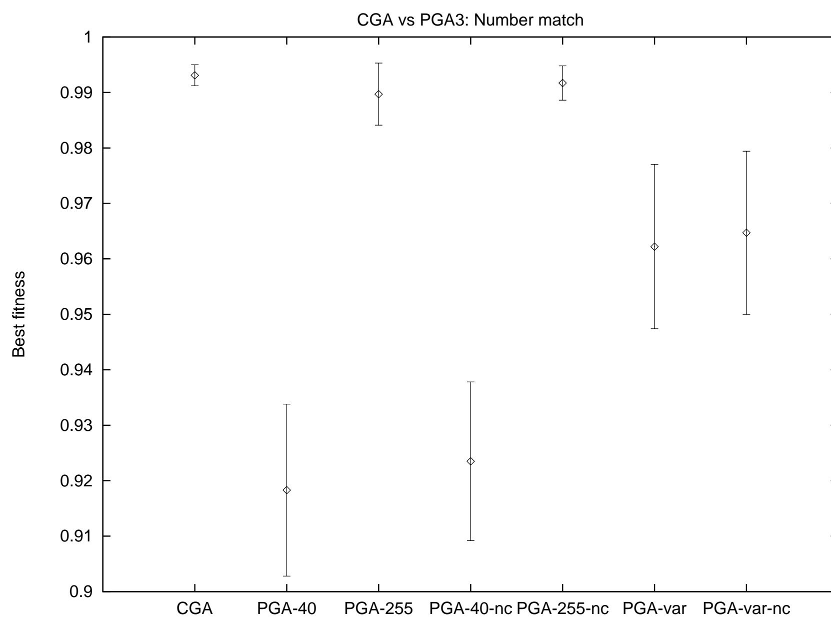

Figure 9. CGA -vs- PGA3 on number match: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

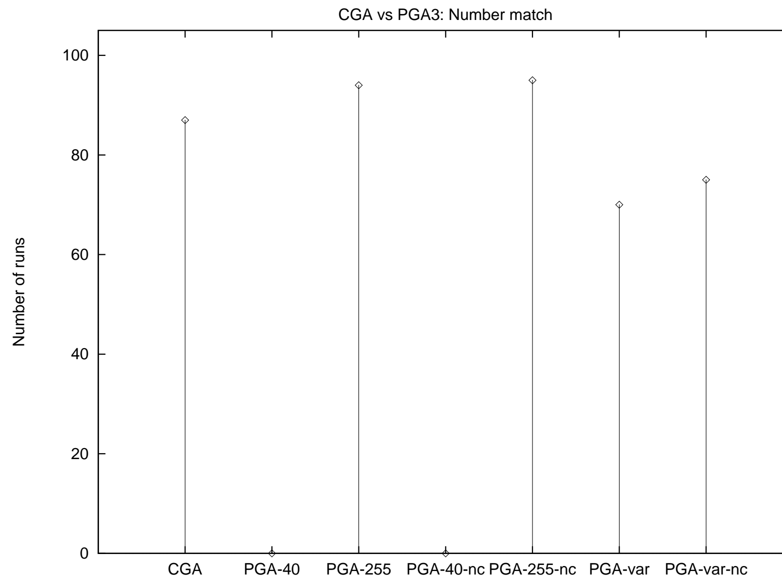

Figure 10. CGA -vs- PGA3 on number match: Number of runs out of 100 that find solutions with fitness 0.99 or higher.

rower y-axis range.) In particular, the PGAs with length 40 show a marked improvement in the number of hits. PGA-255 with and without non-coding regions is able to achieve a 100% hit rate in all but one experiment (which reaches 98).

In both the number match and symbolic regression problems, PGA2 performs slightly better than PGA3. Non-coding regions appear to have very little effect. Variable length individuals do give the PGA more flexibility, but appears to add too much complexity to the search space. Simply giving the PGA a fixed but reasonable amount of resource with which to work seems to be the best, and relatively simple, solution. It is important to note that, according the the calculations in section 4, comparing a CGA of length 40 with a PGA of length 255 still significantly penalizes the PGA. Presumably, increasing the length, and thereby the resolution, of the PGA would result in even better performance.

7. Conclusions

This paper introduces a GA with a new representation method which we call the proportional GA. The PGA is a multi-character GA that relies on the existence of genes rather than the order of genes to encode information. The inspiration for the PGA comes from the biological concept of gene expression: existing genes on a genome are expressed

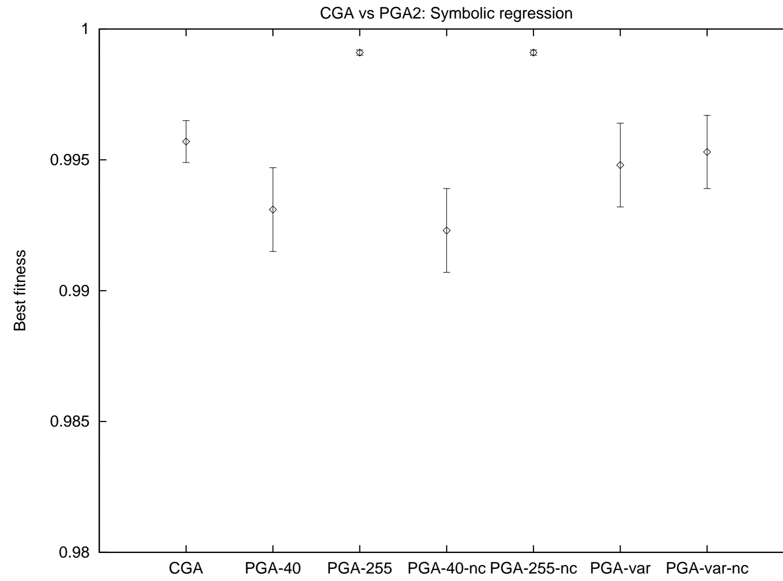

Figure 11. CGA -vs- PGA2 on symbolic regression: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

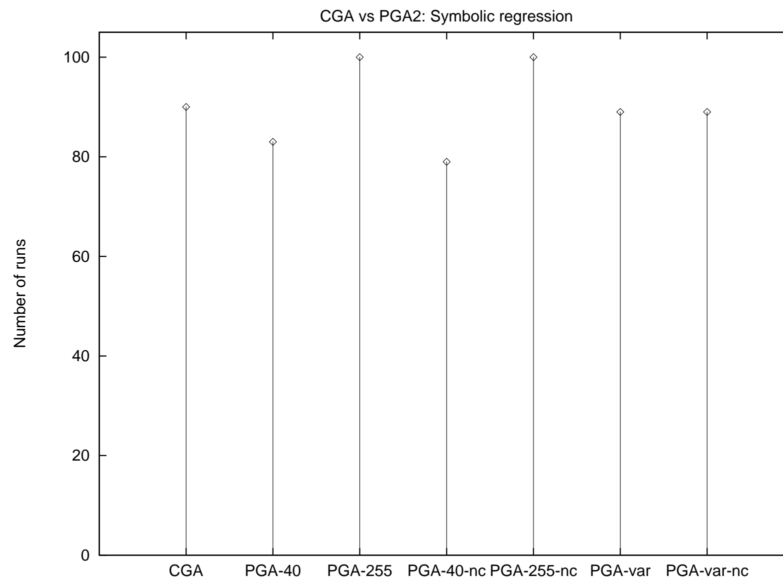

Figure 12. CGA -vs- PGA2 on symbolic regression: Number of runs out of 100 that find solutions with fitness 0.99 or higher.

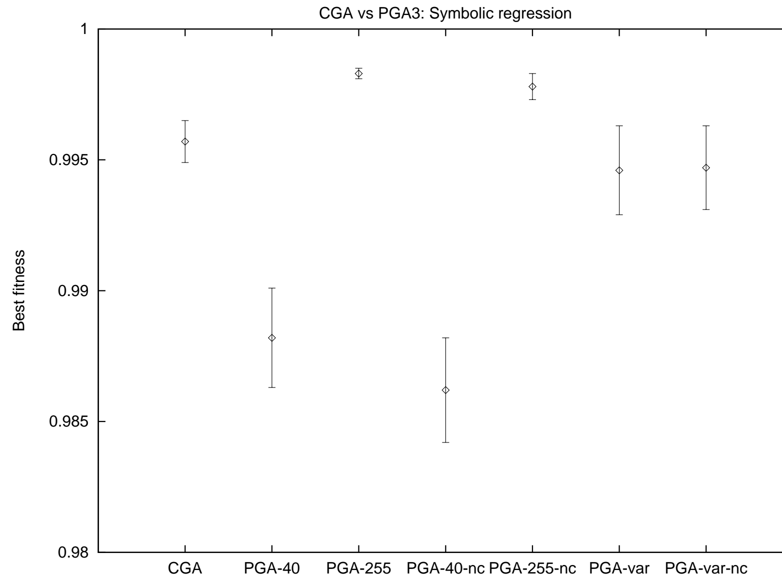

Figure 13. CGA -vs- PGA3 on symbolic regression: Fitness of best solution found averaged over 100 runs and 95% confidence intervals.

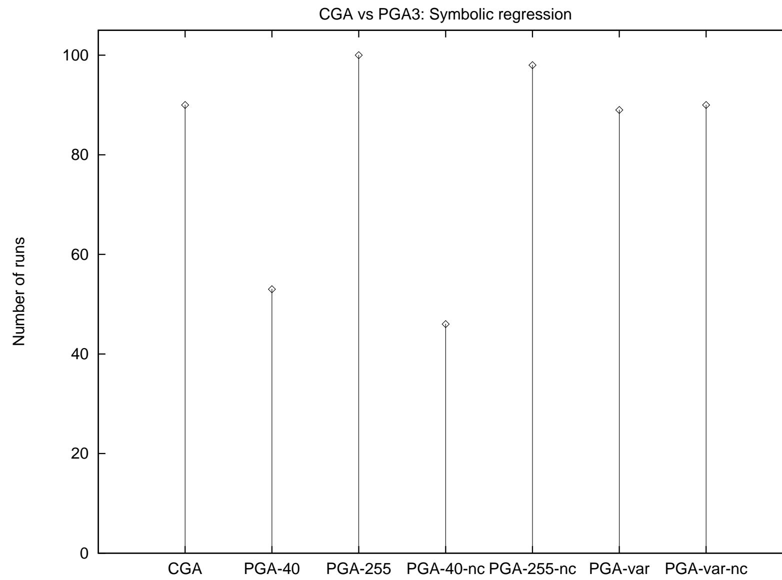

Figure 14. CGA -vs- PGA3 on symbolic regression: Number of runs out of 100 that find solutions with fitness 0.99 or higher.

when they produce protein products; the combination of expressed protein products interact to produce life. Similarly, existing characters on a PGA individual are expressed and interact with the other expressed characters from that individual to produce a candidate solution. Information is represented in terms of the proportions rather than the ordering of the characters on an individual. Statistically, a fairly matched CGA and PGA should have, on average, equal probabilities of finding a solution. Experimentally, the PGA appears to be able to generate comparable behavior even with lowered resolution of expression.

In section 6 we give a list of questions on which the work here is focused. We now discuss our conclusions with respect to those questions.

1. How does a PGA compare to a traditional CGA? Can a PGA perform at least as well as a CGA?

   Our initial studies indicate that the PGA can perform as well or better than a CGA. The determining factor for PGA performance appears to be its resolution. Given a reasonable resolution, our PGA performances were comparable or better than the CGA. "Reasonable", in this case, can still be significantly less that CGA resolution (in our experiments, five times less).

   The PGA appears to be particularly well suited for resource allocation problems. A PGA that has enough resolution to encode the target values will almost always find an optimal solution. A PGA that does not have enough resolution, finds the best possible solution for its given resolution. Given the opportunity, a PGA will adjust the length of its evolved individuals to more closely match the required resolution of a solution. The similarities between the PGA problem representation and the resource allocation problem itself is thought to be a reason for the excellent PGA performance.

   On the number match and symbolic regression problems, where the target values do not sum to 1.0, the PGA performs comparably or better than the CGA if it is given enough resolution. The ability to vary individual length is less useful in these problems, presumably due to the more complex mapping between genotype and phenotype. Higher resolution appears to be the best method for improving PGA performance. Interestingly, while CGA performance is similar for both problems, the PGA performs significantly better on the symbolic regression problem. The explanation for this different is unclear at this point, but we note that both the resource allocation and symbolic regression problems encode values that are dependent on each other.

2. Does a PGA use non-coding regions as a means of "fine-tuning" the values it encodes?

   The impact of non-coding regions is unclear in these experiments. In general, the addition of non-coding regions did not produce any noticeable difference in PGA behavior. The only instance in which non-coding regions improved performance significantly is in the resource allocation problem when a PGA length that does not match perfectly to form the target resolutions. In that instance, the PGA appears to use non-coding regions as "padding" to achieve better target values.

3. Can a PGA regulate the length of its individuals to minimize computational effort while maximizing performance?

   Given the opportunity, a PGA will attempt to evolve the length of its individuals to accommodate the required resolutions. Parsimony pressure can be used to minimize length with, apparently, very little impact on fitness. The PGA successfully uses variable lengths to solve the resource allocation problem. With the other problems, however, variable length appears to be less useful than simply providing sufficient resolution. As compared with insufficient resolution, however, variable length PGAs achieve significantly better fitness values. Overall, the addition of variable length individuals appears to be more beneficial than the addition of non-coding regions.

4. Does a PGA encourage the formation of building blocks?

   The PGA forms building blocks by distributing copies of each character across an entire individual. In a sense, any segment of an individual is a "building block" representing a coarser version of the solution represented by the entire individual. As a result, the solution represented by a single individual is extremely robust in response to changes in the length as well as to crossover. Close examination of individual genomes finds many repeated regions. The PGA does not form building blocks that consist of tightly linked regions of the same character.

Our preliminary studies on the PGA have yielded very promising results and we plan to continue development and analyses of the PGA representation. Despite the lowered resolution of the PGA representation, PGA performance appears to be competitive with CGA performance and PGA runs appear to be taking advantage of the flexibility provided by variable length individuals. Although well-suited to

resource allocation problems, the PGA representation can be modified to work successfully on other types of problems.

Acknowledgements

We would like to thank the reviewers and the members of the UCF Evolutionary Computation Laboratory for their careful reviews and many useful suggestions for improving this paper.

References

1. J. D. Bagley. *The behavior of adaptive systems which employ genetic and correlation algorithms.* PhD thesis, University of Michigan, 1967.
2. W. Banzhaf. Genotype-phenotype-mapping and neutral variation – a case study in genetic programming. In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, *Parallel Problem Solving from Nature 3*, pages 322–332, 1994.
3. E. Beutler, M. Yeh, and V. F. Fairbanks. The normal human female as a mosaic of X-chromosome activity: Studies using the gene for G-6-PD-Deficiency as a marker. *Proc. Nat'l Acad. Sci.*, 48(1):9–16, 1962.
4. 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.
5. A. Clark and C. Thornton. Trading spaces: Computation, representation, and the limits of uninformed learning. *Behavioral and Brain Sciences*, 20:57–90, 1997.
6. H. Curtis. *Biology.* Worth Publishers, 1983.
7. J. M. Daida, S. P. Yalcin, P. M. Litvak, G. A. Eickhoff, and J. A. Polito. Of metaphors and Darwinism: Deconstructing genetic programming's chimera. In *Proc. Congress on Evolutionary Computation*, pages 453–462, 1999.
8. D. Dasgupta and D. R. MacGregor. Nonstationary function optimization using the structured genetic algorithm. In R. Manner and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 145–154, 1992.
9. L. J. Eshelman, R. A. Caruana, and J. D. Schaffer. Biases in the crossover landscape. In J. D. Schaffer, editor, *Proc. 3rd Int'l Conference on Genetic Algorithms*, pages 10–19, 1989.
10. 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.
11. D. R. Frantz. *Non-linearities in genetic adaptive search.* PhD thesis, University of Michigan, 1972.
12. D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison Wesley, 1989.
13. 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 Conference on Genetic Algorithms*, pages 56–64, 1993.
14. D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

15. 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.
16. G. R. Harik. *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* PhD thesis, University of Michigan, 1997.
17. J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.
18. 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 Conference on Genetic Algorithms*, 1995.
19. H. Kargupta. The gene expression messy genetic algorithm. In *Proc. IEEE Int'l Conference on Evolutionary Computation*, pages 814–819. IEEE Press, 1996.
20. H. Kargupta. A striking property of genetic code-like transformations. *Complex Systems*, 11, 2001.
21. H. Kargupta and B. H. Park. Gene expression and fast construction of distributed evolutionary representation. *Evolutionary Computation*, 9(1):43–69, 2001.
22. R. E. Keller and W. Banzhaf. Evolution of genetic code in genetic programming. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proc. Genetic and Evolutionary Computation Conference*, 1999.
23. R. E. Keller and W. Banzhaf. Evolution of genetic code on a hard problem. In L. Spector, E. D. Goodman, A. S. Wu, W. B. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proc. Genetic and Evolutionary Computation Conference*, 2001.
24. P. J. Kennedy and T. R. Osborn. A model of gene expression and regulation in an artificial cellular organism. *Complex Systems*, 11, 1997.
25. J. Lewis. A comparative study of diploid and haploid binary genetic algorithms. Master's thesis, University of Edinburgh, 1997.
26. K. Mathias and L. D. Whitley. Transforming the search space with gray coding. In *Proc. IEEE Int'l Conference on Evolutionary Computation*, pages 513–518, 1994.
27. K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimization. In L. J. Eshelman, editor, *Proc. 6th Int'l Conference on Genetic Algorithms*, 1995.
28. S. Ohno. *Evolution by gene duplication.* Springer-Verlag, 1970.
29. M. O'Neill and C. Ryan. Grammar based function definition in grammatical evolution. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proc. Genetic and Evolutionary Computation Conference*, pages 485–490, 2000.
30. 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.
31. R. E. Smith and D. E. Goldberg. Diploidy and dominance in artificial genetic search. *Complex Systems*, 6(3):251–285, 1992.
32. T. Soule and A. E. Ball. A genetic algorithm with multiple reading frames. In L. Spector, E. D. Goodman, A. S. Wu, W. B. Langdon, H. M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proc. Genetic and Evolutionary Computation Conference*, 2001.

33. G. Syswerda. Uniform crossover in genetic algorithms. In *Proc. 3rd Int'l Conference on Genetic Algorithms*, 1989.
34. V. K. Vassilev and J. F. Miller. Embedding landscape neutrality to build a bridge from the conventional to a more efficient three-bit multiplier circuit. In D. Whitley, D. Goldberg, E. Cantu-Paz, L. Spector, I. Parmee, and H.-G. Beyer, editors, *Proc. Genetic and Evolutionary Computation Conference*, 2000.
35. J. D. Watson, M. Gilman, J. Witkowski, and M. Zoller. *Recombinant DNA*. Scientific American Books, 1992.
36. A. S. Wu and K. A. De Jong. An examination of building block dynamics in different representations. In *Proc. Congress on Evolutionary Computation*, pages 715–721, 1999.
37. 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.
38. 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.
39. K. Yoshida and N. Adachi. A diploid genetic algorithm for preserving population diversity. In Y. Davidor, H.-P. Schwefel, and R. Manner, editors, *Parallel Problem Solving from Nature 3*, 1994.