Bloat is Unnatural: An Analysis of Changes in Variable Chromosome Length Absent Selection Pressure

UCF Technical Report CS-TR-04-01

January 2004
Revised July 2004

Hal Stringer and Annie S. Wu

School of Computer Science, College of Engineering and Computer Science
University of Central Florida, Orlando, FL 32816
stringer@cs.ucf.edu, aswu@cs.ucf.edu

Abstract. In this work we provide empirical evidence that shows how a variable-length genetic algorithm (GA) can naturally evolve shorter average size populations. This reduction in chromosome length appears to occur in finite population GAs when 1) selection is absent from the GA (random) or 2) when selection focuses on some other property not influenced by the length of individuals within a population.

1 Introduction

In (Wu & Stringer, 2002), we investigated the use of a new form of genetic algorithm (GA) to develop rule sets for autonomous micro air vehicles (MAVs). This "Chunking GA" or ChGA combines a variable-length GA with a shared communal memory that allows evolution of both individual chromosomes and memory simultaneously. Memory can be accessed by individuals in the population to improve their solutions over the course of a GA run.

The ChGA exhibits a number of unique properties which warrant further investigation. One of these is the way in which a ChGA evolves its memory contents to contain an optimal or near optimal solution. The ChGA appears able to separate good genes from bad (or less good) and stores them in the shared memory structure. In (Stringer & Wu, 2004) we provide a more formal investigation of this "winnowing" effect and the relationship between memory capacity and solution size.

A second unique property of the ChGA is its ability to reduce the size of individual base chromosomes (e.g., MAV rule sets) over time. This reduction occurs without explicit parsimony pressure built into either the ChGA or the fitness function.

A series of experiments was conducted using a ChGA and various fitness functions to verify that reductions in base chromosome lengths were not the result of the MAV simulator or *mxn* MaxSum functions used in previous works. Although not reported here, the results of those experiments show that winnowing and reduction in base chromosome size occurs to varying degrees with most of our test functions. In addition it appears that reduction in base chromosome length always occurs after the contents of memory are filled with a near optimal solution.

We have come to believe that reduction in base chromosome length is related to reduction in selection pressure as the ChGA finds a near optimal solution and stores it in memory. To put this in proper perspective, we must look at what happens during a ChGA's execution.

As mentioned previously, the ChGA evolves the contents of its shared memory simultaneously with access to that memory by individuals within the population at large. At some point in the ChGA's execution, memory contains numerous high-value genes. These genes are in turn accessed by more and more individuals within the population. Over time, the contents of memory becomes the primary contributor to these individuals' fitness. The contents of their base chromosomes becomes less relevant. As the run continues further, the majority of individuals reference the same memory contents (or slots) and we begin to see similar fitness values for most of the population – the population converges on references to the same memory slots. As a result, selection among these like individuals within the population becomes essentially random in nature.

In this work we focus specifically on understanding and modeling how the absence of selection pressure (random selection) can cause reduction in chromosome sizes for a variable-length GA given 1) a finite population and 2) a uniform distribution of chromosome lengths within a population.

We begin in Section 2 with a general survey of variable-length genetic algorithms and a discussion of size increase ("bloat") in GA and GP systems. We test our ideas using a variety of experiments and models in Sections 3 and 4. Empirical data from models in these sections show that absent selection pressure related to chromosome size, a variable-length GA can evolve populations with shorter average-sized chromosomes. Our conclusions are contained in Section 5 along with suggestions for future work and investigations.

2 Variable-Length Genetic Algorithms and Other Related Work

Many different GAs have been developed over the years which incorporate variable length chromosomes. In this section we consider some of the different variable-length representations found in the GA literature.

2.1 The messyGA

The earliest and best known variable-length representation is the *messyGA* (Goldberg, Korb & Deb, 1989). This GA was developed to eliminate bit positional dependencies in a standard GA. The representation allows the GA to evolve both bit values and bit locations on a chromosome during a GA run. The goal of the messyGA is to find tight linkage between bits as well as good bit values simultaneously.

Each bit value in a messyGA chromosome is tagged with a name (an integer number indicating the position of the bit). An ordered name-value pair is referred to as a messy gene. Prior to fitness evaluation, the value bits are extracted from the chromosome and reordered based on their corresponding names. Goldberg, et. al. (1989) describe this approach as a "moving-locus scheme". Bits within a genotypic string are no longer in fixed positions and can move around on a chromosome to develop better building blocks.

The messyGA was implemented using a cut-and-splice operator rather than standard one-point crossover. The operator produces variable-length chromosomes. Reproduction can sometimes result in children with multiple bit-values for a given name (overspecification) or with an absence of bit values for a given name (underspecification). Instances of these two errors must be dealt with before a complete solution can be forwarded to the fitness function for evaluation.

Left-to-right precedence is used to eliminate overspecification due to duplicate or competing bit values. The bit value from the first messy gene with a given name is used during fitness evaluation. All other messy genes with the same name are then ignored and considered to be non-coding regions.

Underspecification occurs when no messy gene exists for a given bit required in the pre-test solution. This condition is handled by the use of competitive templates. Bit values not found in the genome are pulled from a template, a string containing locally optimal bits from the previous generation. The use of competitive templates allows the messy GA to build a complete solution for testing even when an individual chromosome does not encode a solution in its entirety.

The primary weakness of the messyGA is its focus on bits. A complete solution for an *n*-variable problem with each variable needing *l* bits requires $n2^{nl}$ additional bits in the genome to identify all its component bits.

2.2 The Virtual Virus

Another variable-length representation in the literature is the *Virtual Virus* or VIV (Burke et al., 1998). The goal of this work was to create a genetic algorithm which more closely mirrored biology. The authors describe a number of key features found in biological systems that may be absent from a standard GA including:

- Genomes which vary in length during evolution,
- Gene locations independent of position identified by stop and start codons,
- Presence of non-coding regions in a genome,
- Duplicative or competing genes on the same genome,
- Use of overlapping reading frames when retrieving genetic information,
- A multi-character alphabet (A, T, C & G representing nucleotides) and
- Degenerate mapping of nucleotide triplets (codons) to amino acids.

Each of these features was incorporated into VIV to some degree. Use of a multi-character alphabet and degenerative mapping allows for redundant genotype representations in VIV. More than one codon (three alphabet characters) can translate into the same phenotypic trait. Incorporation of intergenic regions (non-coding information) and redundant genes (the same information can appear more than once) also allow VIV to more closely follow its biological inspiration.

To remove location restrictions, each gene in VIV is marked by a START codon consisting of three characters. This start tag is analogous to a promoter region in DNA. The end of a gene is specified by a corresponding STOP codon. The use of START and STOP codons allows genes in VIV to be of variable length. In addition, non-coding regions can result if any genetic information is contained between a STOP and START codon.

Processing of a chromosome begins by reading left to right across the genome. If a START codon is found at position *p*, any subsequent characters are read as genes until a STOP codon is reached. Reading of the genome picks back up at position *p*+1 looking for the next START codon. The processing of chromosomes in this manner allows VIV to contain overlapping genes within a chromosome. Overlap makes possible evolution of solutions in a very compact genome.

2.3 SAMUEL System

The *SAMUEL system* (Grefenstette, Ramsey & Schultz, 1990) is a type of Learning Classifier System or LCS. SAMUEL uses a variable-length GA at the heart of its learning module to learn rules for sequential decision making. Each individual in the population represents a set of rules or a "tactical plan" to guide SAMUEL in its tasks. Plans consist of a variable number of rules that are loaded into SAMUEL's performance module for execution. Rules are expressed in a high-level if/then representation rather than binary strings. If a condition is matched, then an action is taken. In SAMUEL, rules are analogous to genes within a chromosome. Crossover occurs between rules and serves to combine good rules to form a better solution. Creation of new rules is the function of mutation and a new specialization operator.

2.4 GAs for MAV Rule Development

A number of researchers have used variable length GAs in conjunction with micro air vehicle simulators (Wu, Schultz & Agah, 1999; Bassett & De Jong, 2000; Wu & Stringer, 2002). GAs in these papers are used to evolve rule sets for control and navigation of MAVs. Rules are matched against sensor inputs from a simulated environment. Best matched rules are executed affecting the movement of the MAV.

An MAV has a total of eight sensors that are in either an "on" or "off" state. A total of 2^8 possible environmental states or conditions can exist. The largest possible solution for this problem would contain 256 condition/action rules, assuming we associate one action variable with each possible sensor state combination. The results of experiments show instead that a compact set of general rules (5-20) was sufficient to provide good results.

Over- and underspecification were not directly addressed by GAs working with this problem. Instead these situations were resolved by matching and rule selection algorithms within the MAV simulator. For underspecification (specific conditions not addressed), the MAV selects the rule that most closely matches the current environment. In the case of overspecification (multiple actions for the same condition), the MAV randomly chooses between competing rules to determine the action to take.

2.5 The Proportional Genetic Algorithm (PGA)

The most recent variable-length representation to appear in the literature is the *Proportional Genetic Algorithm* or *PGA*. This representation evolves solutions of fixed length while taking locus-independence to the extreme. The PGA uses a multi-character alphabet. Each character is associated with a specific variable to be found in the solution. The number of each character appearing in the genome is used to compute a value that is then assigned to its corresponding variable.

The PGA uses alphabet symbols as atomic units in its representation rather than bits or genes. To quote from (Wu & Garibay, 2002), "The PGA representation is based on this idea that it is the content rather than the order of the encoded information that matters." The PGA allows evolution of the genome at the atomic character level rather than the gene level.

The PGA can be used to look for solutions of varying sizes. The number of characters in the PGA alphabet is determined at the start based on the number of variables in a complete solution. The downside of this representation is the need for large alphabets if problems have large numbers of variables.

A major benefit of the PGA is that over- and underspecification are no longer issues. All assigned characters in the genome are used to calculate proportions for each variable. Underspecification (absence of a character) is handled simply by setting that value associated with a missing character to "0". Another major benefit of this representation is the lack of any "overhead" information in the genome – no identifying tags or start/end tags are required

2.6 Bloat and Bloat Control in GA/GP

This paper looks at reductions in genome length under random selection for variable length genetic algorithms. As one can imagine, the quantity of literature directly addressing this issue is extremely limited. What we have found is a wealth of information describing the problem of uncontrolled genome growth. As a matter of fact, most variable-length representations, regardless of the evolutionary computation (EC) paradigm suffer from "bloat" to one degree or another. In general, bloat is an increase in genome length from one generation to the next, primarily due to growth in extraneous genotypic information. This information is extraneous since it is not necessary to form a valid and/or optimal solution.

For GAs, bloat may include bits which make up non-coding regions, duplicate genes or competing genes. Suggested causes of bloat within a GA include, but are not limited to 1) high mutation rates (Ramsey et al., 1998) and 2) the idea that additional genetic material is "free" (does not directly impact fitness) and provides more space to explore for better genes (Burke et al., 1998). GA researchers have taken a variety of approaches to handling bloat in their works. The following control methods are taken from papers previously cited in this Section:

- **SAMUEL:** The crossover operator removes any duplicate rules from being passed to the same child.
- **PGA and Chunking GA**: Simple right truncation is applied after crossover to any child longer than a pre-specified limit. All characters/genes beyond that limit are deleted from the chromosome.
- **VIV**: A linear length penalty is added to the fitness function. This penalty alters the initial raw fitness of each chromosome so that shorter individuals are favored assuming all raw fitness values are equal.
- **messyGA**: No explicit bloat control mechanism. However the authors suggest that use of competitive templates discourages duplicate genes.

A final approach to bloat control is one advocated by Harvey (Harvey, 1992). He recommends use of a reproductive operator which carefully chooses a crossover point on the second parent that is complimentary to the crossover point chosen on the first parent. The *SAGA cross* causes a gradual increase in average genome size within a population from one generation to the next, yet allows small variations in individual length within the same generation.

Within EC, variable-length representations are most prominent in genetic programming or GP. This EC paradigm evolves parse trees represented by LISP programs which in turn are encoded as individual chromosomes. GP has been observed to evolve trees or programs that are much larger than necessary, containing one or more sections of unused code (Koza, 1992; Blickle & Thiele, 1994; Soule, Foster & Dickinson, 1996). Of all EC paradigms, GP has produced the largest quantity of bloat-related literature due to its abundant use of variable-length genomes.

Various causes have been espoused to account for increases in the size of GP chromosomes. Among these are theories based on the hitchhiking of non-coding regions (Tackett, 1994), defense against crossover (Blickle & Thiele, 1994; Nordin & Banzhaf, 1995), removal bias (Soule & Foster, 1998), and the idea that fitness causes bloat (Langdon & Poli, 1997). These theories are summarized in (Luke, 2003) along with the introduction of a new theory which correlates GP bloat to the depth of modification points within a GP parse tree.

Methods for controlling bloat are equally numerous in the GP literature. Most methods fall into two categories: 1) parsimony pressure methods which incorporate some size penalty into the fitness function and 2) reproduction restrictions which eliminate or prevent children which exceed a given parse tree depth limit. Recent additions to the arsenal of anti-bloat methods include use of *size fair* and *homologous crossover*s (Langdon, 2000), *double* and *proportional tournaments* (Luke & Panait, 2002), and the treatment of size and raw fitness as separate objectives in a *multi-objective optimization* scheme (Panait & Luke, 2004). Panait and Luke offer two additional methods (the *Waiting Room*, *Death by Size*) for use with steady-state GP systems.

Parsimony pressure appears to be the bloat control method common to both GAs and GP. Several of the newer GP methods just mentioned could easily be applied to variable-length GAs but to our knowledge have not yet been tried. Bloat control methods which take into account maximal tree depth are specific to GP and are not directly applicable to variable-length GAs.

As shown here, bloat control methods in GP are more numerous, varied, and often more complex than those described in the GA literature. This could be due to GP's popularity and larger body of work relative to that of variable-length GAs. Or this could be a result of the more complex nature of GP problem representations. Taken from the perspective of a GA practitioner, why use complicated control methods when a GA's locus-independent linear structure allows one to simply truncate genomes greater than a pre-determined size?

3 Simulating a Variable-Length GA

How do we verify our hypothesis that a GA without selection pressure causes the average size of a population to shrink. Since our motivation for this work is in part a desire to explain the shrinking genomes in (Wu & Stringer, 2002), we must first verify that this behavior is not problem specific. To do so we perform a series of empirical tests on a simple model of a variable-length GA with selection and crossover operators.

Our model starts with an integer array of 100 elements, each element representing a single individual in a 100-member GA population. The value of each array element represents the number of genes contained within an individual's chromosome (its length). Lengths can vary in the model from 0 to *c* where *c* is some size cap which invokes a truncation operator. For most of the experiments in this section, elements of an array are initialized to 20 to match the initial length used in prior ChGA works.

Before proceeding we should address the issue of "0" length chromosomes. In (Wu & Stringer, 2002; Stringer & Wu, 2004) our GA allowed for individuals of with base chromosomes of zero length. This type of "null" chromosome was possible since individuals could still access genes stored in the ChGA's communal memory structure. Without a memory structure, crossover is still possible between two parents, one being of 0 length. This scenario could be thought of as a form of asexual reproduction where the genetic material in one parent is divided between two children. Finally, for the experiments in this section the use of 0-valued array elements is a non-issue. If length is re-defined as the number of crossover points within a chromosome, then a 0-length individual is really a 1-gene chromosome, that gene being indivisible. Alternatively our experiments could also have been run using elements of size 1 to *c*+1 and achieved the same results.

Within our model, a `for_loop` contains code which simulates random selection, crossover, and reproduction. No fitness function exists in our initial model since we want to test the performance of a GA absent any selection pressure. We have also excluded mutation from our model as this operator normally affects the contents of a chromosome rather than its length.

The loop is executed 500 times to simulate 500 generations. During each "generation", a new child array of 100 elements representing the next generation is created.

A simulated 1-point crossover populates elements in a child array. Two parent elements are chosen (selected) from the first array at random. Two random integers between 0 and each parental element's value are chosen as the crossover points. Two children are produced and copied into the child array as follows: *Child1* equals the first crossover point plus the value of *Parent2* less the second crossover point; *Child2* equals the second crossover point and the difference between *Parent1* and the first crossover point. The following example illustrates our simulated crossover:

Parent1 = **25**	Crossover Point 1 = 6	Child1 = 6 + (7-5) or **8**
Parent2 = **7**	Crossover Point 2 = 5	Child2 = 5 + (25–6) or **24**

The above calculation serves to model the effect of crossover on the length of chromosomes from one generation to the next. Remember our concern here is size, not fitness of the individual or its genetic composition.

3.1 Results Using 1-Point Crossover and Random Selection

A number of experiments of 1000 runs each were performed. At the end of each run, the average length of the final population (average of 100 array elements) was computed and classified according to size ranges in Table 1. The experiments themselves varied by the size cap (*c*) applied to larger chromosomes. In past works, we have used a size cap of 100 genes with right truncation similar to Table 1(b).

As Table 1 shows, the vast majority of populations ended with an average length below the starting average of 20 genes per individual. This occurred even when a size cap of 1 billion (in effect no cap) was implemented. A smaller size cap (in this case below 250) may act as a dampening force causing more "shorter" populations. But a cap itself does not seem to be the force driving the reduction.

Ranges for Avg Population Size at End of Run	Number of Runs Within Given Ranges			
	(a) Cap = 50	(b) Cap = 100	(c) Cap = 250	(d) Cap = 1B
<10	895	740	673	665
(10 – 20]	98	163	140	133
(20 – 30]	7	63	73	67
(30 – 40]	0	28	36	30
(40 – 50]	0	6	25	22
(50 – 60]	n/a	0	15	22
(60 – 70]	n/a	0	17	13
(70 – 80]	n/a	0	12	10
(80 – 90]	n/a	0	4	5
(90 – 100]	n/a	0	4	3
>100	n/a	0	1	30
Avg. Size for All Runs	3	6	11	16

Table 1: Number of runs with avg. size of final population within various ranges for GA simulations using random selection, 1-pt crossover, and initial gene size of 20

Could our short populations be due to the initial array values (20)? To find out we conducted the same experiment with array elements initialized to 200 thus simulating longer genes per chromosome. Results are shown in Table 2. The distribution of population averages across ranges is similar to that in Table 1 (assuming larger ranges) and the average size for all runs varies approximately by a factor of 10.

Ranges for Avg Population Size at End of Run	Number of Runs Within Given Ranges			
	(a) Cap = 500	(b) Cap = 1000	(c) Cap = 2500	(d) Cap = 1B
<100	879	725	667	661
(100 – 200]	115	170	143	138
(200 – 300]	6	68	75	67
(300 – 400]	0	31	37	32
(400 – 500]	0	6	26	22
(500 – 600]	n/a	0	13	19
(600 – 700]	n/a	0	16	11
(700 – 800]	n/a	0	13	16
(800 – 900]	n/a	0	6	2
(900 – 1000]	n/a	0	3	3
>1000	n/a	0	1	29
Avg. Size for all Runs	44	77	122	167

Table 2: Number of runs with avg. size of final population within various ranges for GA simulations using random selection, 1-pt crossover, and initial gene size of 200

3.2 Adding Fitness Back Into the Simulation

The previous experiments indicate that, absent selection pressure, a GA can shorten the average size of a population over time. What if selection pressure is incorporated into our model? Experiments were conducted similar to those shown in Tables 1 & 2 using tournament selection with fitness maximization in lieu of random selection. A cap of 1 billion was set for all experiments in this group to allow near unlimited growth in chromosome size

In the first experiment (Table 3 column (a)), each parent and child element in the array is assigned a "fitness" equal to its size thus favoring longer individuals. Not surprisingly, the average sizes (and fitnesses) of the final populations are enormously large. All final populations are well over 100 genes in terms of average length. This experiment shows how fitness, tied in some way to length, increases the average size of the population dramatically – another illustration of Langdon and Poli's assertion that fitness causes bloat.

In the second experiment, fitness is equal to 100 minus the size of the individual (favoring smaller individuals). In every case, all individuals evolve to a size of 0 (no genes) hence all 1000 final populations average in the 0-10 range (see Table 3, column (b)).

The third experiment starts by assigning a random fitness value to each array element. During execution, the children of each mating are assigned fitness values equal to the sum of their parents' values divided by two. The result is a fitness assignment that does not depend on the length of the parents. Our simulated GA in this scenario evolves many populations with individuals of short average size (see Table 3, column (c)).

Ranges for Avg Population Size at End of Run	Number of Runs Within Given Ranges			
	(a) Longer is Fitter	(b) Shorter is Fitter	(c) Fitness = (F*p1*+F*p2*)/2	(d) Random Fitness
<10	0	1000	611	709
(10 – 20]	0	0	149	107
(20 – 30]	0	0	77	52
(30 – 40]	0	0	42	25
(40 – 50]	0	0	23	21
(50 – 60]	0	0	22	11
(60 – 70]	0	0	14	10
(70 – 80]	0	0	10	9
(80 – 90]	0	0	8	7
(90 – 100]	0	0	4	7
>100	1000	0	40	42
Avg. Size All Runs	799,722,254	0	21	17
Avg. Fitness All Runs	799,722,254	100	83.49	49.34

Table 3: Number of runs with avg. size of final population within various ranges for GA simulations using tournament selection, 1-pt crossover, and initial gene size of 20

In our last experiment, fitness values for all array elements (parents and children) are assigned randomly. Fitness is in no way tied to the size of the individual. Nor does fitness from prior generations have any bearing on selection in the next generation. Like the results in column (c) we see that populations are by far, smaller than the initial starting size. The fact that fitness is randomly assigned causes a complete absence of selection pressure associated with length and we see that our GA simulation naturally evolves shorter populations. Note that this column is very similar to Table 1, column (c) which used purely random selection without any fitness function.

3.3 Other Crossover Types

The three prior sets of experiments (Tables 1, 2 and 3) modeled a GA using 1-point crossover. Two more sets of experiments were conducted to see if reduction in average size across populations occurs using 2-point crossover. The results of these experiments are contained in Tables 4 and 5.

These tables show that reduction in average chromosome size for a population can occur with at least one other form of crossover. A comparison of results from Table 1 & 4 to those in Tables 3 & 5 shows however that the size reduction under 2-point does not appear to be as great (or fast) as that occurring with 1-point.

Avg Population Size at end of Run	Number of Runs Within Given Ranges			
	(a) Cap = 50	(b) Cap = 100	(c) Cap = 250	(d) Cap = 1B
<10	761	598	563	561
(10 – 20]	212	228	189	188
(20 – 30]	27	100	89	80
(30 – 40]	0	51	44	45
(40 – 50]	0	21	27	26
(50 – 60]	n/a	2	23	17
(60 – 70]	n/a	0	20	14
(70 – 80]	n/a	0	15	14
(80 – 90]	n/a	0	11	6
(90 – 100]	n/a	0	9	13
>100	n/a	0	10	36
Avg. Size for All Runs	5	10	16	19

Table 4: Number of runs with avg. size of final population within various ranges for GA simulations using random selection, 2-pt crossover, and initial gene size of 20

Ranges for Avg Population Size at End of Run	Number of Runs Within Given Ranges			
	(a) Longer is Fitter	(b) Shorter is Fitter	(c) Fitness = (F*p1*+F*p2*)/2	(d) Random Fitness
<10	0	1000	555	605
(10 – 20]	0	0	179	153
(20 – 30]	0	0	77	61
(30 – 40]	0	0	53	46
(40 – 50]	0	0	38	35
(50 – 60]	0	0	22	20
(60 – 70]	0	0	20	15
(70 – 80]	0	0	10	13
(80 – 90]	0	0	8	9
(90 – 100]	0	0	8	3
>100	1000	0	30	40
Avg. Size All Runs	838,601,988	0	20	18
Avg. Fitness All Runs	838,601,988	100	83.66	49.60

Table 5: Number of runs with avg. size of final population within various ranges for GA simulations using tournament selection, 2-pt crossover, and initial gene size of 20

3.4 Effect of Population Size and Length of Run

We have recently become aware of work coming out of the GP community which may impact our own investigations (personal conversations with W. Langdon and N. F. McPhee at GECCO 2004). An initial review of their papers (McPhee & Poli, 2001; Poli & McPhee, 2001) shows that, given an infinite population and a flat fitness landscape, there should be no change in the average length of individuals within a population over time. The model we have constructed in this section uses a finite population of 100 elements. Do our results change if larger populations are used?

To answer that question we conducted a series of additional experiments using our GA model but varied both the size of the population (from 100 to 1000 elements) and the number of generations (500-5000) per run. For each size/generation combination, a total of 500 runs were made. In each run, all members of the population began with a length of 50. The number of runs ending with a population of average length less than 50 (the average starting length) were counted, converted to a percentile, then plotted in Figure 2.

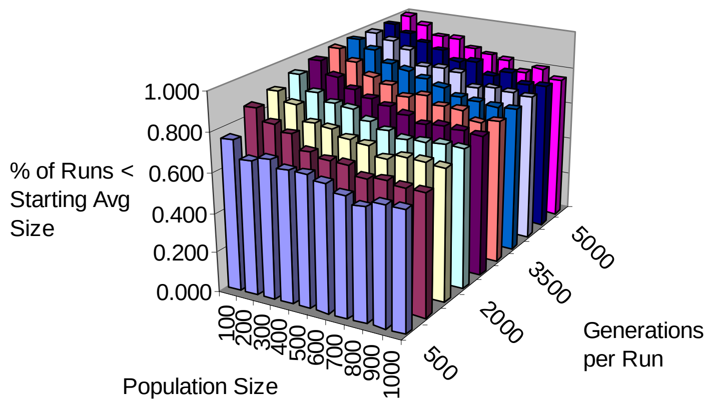


Figure 1: Graph showing effect of population size and run length on percentage of runs with ending average population sizes below the starting size.

Figure 1 shows that as population size increases, the reduction in mean size within a population slows. In other words, the phenomenon we have seen with our ChGA experiments and those conducted earlier in this section becomes less apparent with larger populations.

Also shown in Figure 1 is the effect of time on this reduction. The longer the GA runs, the greater the reduction in mean length. For example, when using an array of 100 elements and run length of 5000 generations, over 99% of all runs ended with a mean length less than 50 (the mean length for the starting population). At the opposite end of the time spectrum (100 elements, 500 generations) only 79% of runs ended in populations with mean lengths of less than 50.

The results in Figure 1 do not necessarily disagree with the conclusions in (Poli & McPhee, 2001). If we extended Figure 1 out indefinitely in terms of population size we would most likely never see a reduction in mean length within that population, even given an infinite number of generations.

3.5 Discussion

The simulations performed in this section yield several important bits of information. First, absent selection pressure our GA model prefers to evolve finite populations of shorter average size given sufficient time (generations). Without a size cap, roughly 75%-80% of all final populations have an average chromosome size less than the starting length regardless of initial starting size. Second, a size cap does exert downward pressure on chromosome size but does not appear to be the primary cause of reductions in average size. Last but not least, time is a factor in this reduction – the more individuals in a population, the more time required for reduction to take place.

Results clearly show that a finite population GA without any selection pressure tends to evolve populations with shorter average chromosome sizes. But as shown in Table 3 & 5, absence of selection is not the complete story. The GA evolves larger-size populations only when longer individuals are considered more fit. Shorter populations can evolve in the presence of selection as long as 1) fitness and size are not related to one another or 2) fitness favors more compact solutions.

These simulations seem to confirm at least that fitness can be the cause of bloat if longer individuals in a population are considered more fit. The reason for longer individuals being more fit is itself a subject for future investigation.

4 Inside, Outside and Equal Crossovers

This section explores the idea that a GA, absent selection pressure creates a diverse population in terms of size and does not inherently over-produce longer children. We conduct our study using empirical evidence from enumerative experiments that track the types of crossover events possible in a variable-length GA. Our experiments requires that we first define the different possible types of crossover events that can take place during a GA's reproduction cycle.

Assuming one-point crossover, there are three possibilities where the size of offspring is concerned. A crossover event can produce 1) children of size equal to the parents (*equal event*); 2) one child longer than both parents and one child shorter than the shortest parent (*outside event*); or 3) both children shorter than the longest parent and longer than the shortest parent (*inside event*). Figure 2 illustrates these three distinct events.

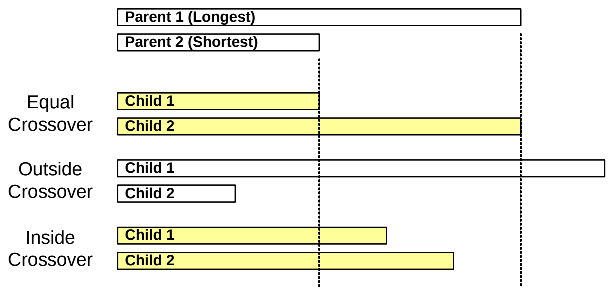


Figure 2: Illustration of crossover event categories for reproduction in variable-length GAs.

4.1 A Theoretical Population

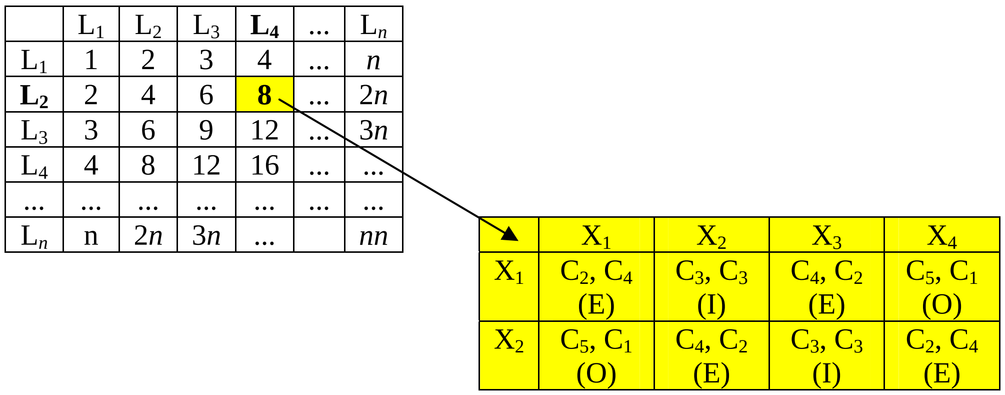
To set up our experiments we define a theoretical population as one occurring at some generation t consisting of $2n$ individuals. The individuals which make up this population consist of two of every size from 1 to n crossover points. We use the notation L_i to indicate the size of an individual where i is the number of crossover units (bits or genes) within the genome. The population in total makes up an nxn matrix with each row and column corresponding to one of the population members (see Figure 3).

The number of possible *crossover events* for any given *mating* is the multiplication of the number of crossover points in each of the parents. For example, a total of 8 unique crossover events is possible between a single mating of two parents, one with four crossover points and the other with two. We can fill each cell in our population matrix with the number of possible crossover events that can take place for any two chromosomes of size j and k where $j,k \leq n$ by simply multiplying the row and column indexes. Each element of the matrix in Figure 3 shows the number of possible crossover events between parents with crossover point lengths of *j* and *k* up to *n*.

	L_1	L_2	L_3	L_4	...	L_n
L_1	1	2	3	4	...	*n*
L_2	2	4	6	8	...	2*n*
L_3	3	6	9	12	...	3*n*
L_4	4	8	12	16	...	...
...	...	...	...	...	...	...
L_n	n	2*n*	3*n*	...		*nn*

Figure 3: Number of possible crossover events in *nxn* population matrix

Let us now extend our illustration from Figure 3 by expanding a single cell to contain an inner matrix representing each of the unique crossover events possible for a given mating. We use the notation X_i to indicate the different crossover points for both individuals where *i* is the location of the crossover point along the genome. In Figure 4 we see this expansion. Children produced by each possible crossover event are shown as C_l where *l* is the number of crossover points (length) in each child. For the mating of any L_2 and L_4 chromosome there are eight possible mating events which produce children of various sizes.



	L_1	L_2	L_3	**L_4**	...	L_n
L_1	1	2	3	4	...	*n*
L_2	2	4	6	**8**	...	2*n*
L_3	3	6	9	12	...	3*n*
L_4	4	8	12	16	...	...
...	...	...	...	...	...	...
L_n	n	2*n*	3*n*	...		*nn*

	X_1	X_2	X_3	X_4
X_1	C_2, C_4 (E)	C_3, C_3 (I)	C_4, C_2 (E)	C_5, C_1 (O)
X_2	C_5, C_1 (O)	C_4, C_2 (E)	C_3, C_3 (I)	C_2, C_4 (E)

Figure 4: Expansion of *nxn* population matrix cell to show length of children created by each possible crossover event. The number of rows and columns in the expanded matrix is determined by the number of crossover points in the parent genomes.

By comparing the size of the children with that of the two parents we can determine if the crossover event is of type inside (I), outside (O) or equal (E). As Figure 4 shows, the mating of an L2 and L4 parent can result in one of eight crossover events – four of which are equal events, two are inside and two are outside events.

4.2 Experiments and Results

Our first experiment consists of a number of nested loops which look at all possible crossover events for all possible matings for a series of *nxn* population matrices varying in size from 1 to 200. The events are categorized for each matrix and counted by type. The results of this enumerative counting are then converted into percentages of total crossover events and are shown in Figure 5.

It should be noted that our experiment is based on *n* being equal to a number of crossover points. Therefore our results should apply to any variable-length GA regardless of representation form (e.g., integers, floating point numbers, multi-character alphabets).

As *n* increase (the size of the population grows) the number of *Equal* crossover events diminishes rapidly. Equal events are dominated by growing numbers of *Inside* and *Outside* events which approach 33% and 67% of all possible crossovers respectively. Mathematical proof of these numbers are left for future work.

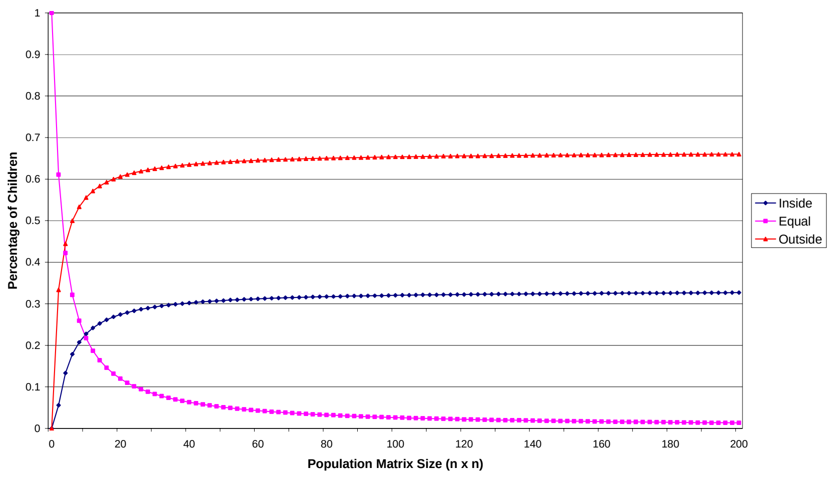


Figure 5: Percentage of crossover events by type for *nxn* population matrices varying in size from *n*=1 to 200 population matrix.

To understand these results more fully we must go back Figure 2. As that figure shows, only an outside crossover event can produce a child longer than both parents. In addition, only one of the two children is actually longer than the longest parent. As a result, we see that one half of the possible children produced by all outside crossovers will be longer than both parents. This leads to the conclusion that the probability of producing a child which is longer than both parents via outside crossover is .67 / 2 or 33.3%. Equally there is a 33.3% probability that a child will be shorter than the shortest parent and produced through an outside crossover event

4.3 Weighting Results for Reality

The empirical results discussed in Section 4.2 have one flaw. Our experiment assumes that all crossover events are weighted equally (or equally likely to occur). During actual execution of a GA, the probability of a given type of crossover is lessened or increased by the probability of a mating between two parents.

Going back to Figure 3 and assuming a 4x4 matrix we have a total of 100 separate possible crossover events. The graph in Figure 5 assumes that each of these events is equally weighted and equally likely to occur. But any single mating has only a 1 in 16 chance of occurring. Crossover event probabilities within each mating must be weighted differently since they result from a single mating event. The result is that crossover events between larger individuals (more events) are individually less likely than those that can occur between smaller parents.

Our previous experiment was re-run but with mating probability factored in to the program. The results of this weighted version are shown in Figure 6. They indicate that the probability of both inside and outside crossovers approaches 50% while equal crossover types drops to 0%. This leads us to the conclusion that the probability of producing a child longer than both parents is only 25% for our theoretical population. This is balanced by a 25% probability that any children produced by our theoretical population will also be shorter than both parents.

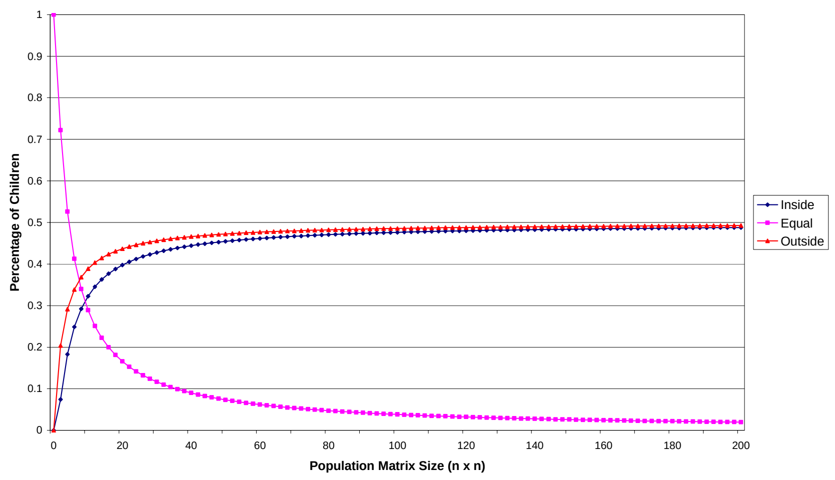


Figure 6: Percentage of crossover events by type for *nxn* population matrices varying in size from *n*=1 to 200 population matrix. Event probabilities weighted based on mating occurrences.

4.4 Discussion

Assuming the weighting of crossover events, we can summarize our results in the following way: Given a) two randomly selected individuals from an infinite population of variable-length chromosomes, uniformly[1] distributed in terms of size and b) two randomly chosen crossover points on those individuals, the probability of an inside or outside crossover event approaches approximately 50% each and the probability that the two resulting children will be equal to the parents in length approaches 0%.

These probabilities help us understand more about how the GA processes variable-length strings and creates size diversity within a population. As the maximum size of individuals within a uniformly distributed population increases, the probability of an inside and outside crossover approaches 50%. The equality of these percentages shows that the GA does not inherently favor one type of event over another. By extension, the GA does not inherently favor crossover types which might lead to bloat.

[1] The use of a *nxn* population matrix in our experiments gives us a perfect and uniform size distribution to work with. The true distribution of size within a GA population appears to be more Gaussian in nature. We do not believe this will change our results significantly, particularly those relating to populations with tightly packed or split distributions. In future work, we hope to investigate crossover types given non-uniform size distributions.

Another result found during these experiments relates to the probability of crossover types for populations with tightly packed or highly split sizes. When size variation between individuals is small (e.g., initial population set to same size), the probability of outside crossover is near 100%. This high probability causes the GA to increase the variance in the next generation dramatically. The opposite is true for populations with split distributions (many short and many long individuals with little in between). In these populations, the probability of inside crossover is greater, thus filling in the gaps between smaller and larger chromosomes.

5 Conclusions

In this work we have provided a general survey of variable-length genetic algorithms. Our experiments in Section 3 provide strong evidence that such GAs can naturally evolve populations of shorter mean length over time if selection is random or not associated with length in some way. We also show how a GA can maintain diversity through inside and outside crossover events. Assuming a population of uniform size distribution, the probability of an inside vs. outside crossover event is 50/50. By extension, the probability of producing a child longer or shorter than both parents in that same population is 25% each.

In terms of the ChGA, our results in this work are mixed. We hoped at the start to find the cause for reduction in base chromosomes. What we have seen from experiments and models in Section 3 is that reductions in chromosome length do occur. They are not the result of our ChGA implementation or prior fitness functions. Section 4 has not however, provided us with a specific reason for the reductions but does indicate that a variable length GA is neutral with respect to crossover event type. There does not appear to be any bias towards crossover events which would over-produce longer children.

The information presented in this Technical Report begins our investigation into the causes of reduction in chromosome length absent selection pressure. Much work needs to be done. Some of the question this work has raised include:

- Section 3 showed that reductions in average chromosome size for a population can happen but it doesn't occur 100% of the time. This indicates that the process which causes reduction is not deterministic. Why do our models show a preference for shorter populations rather than a consistent reduction in all runs?
- Can the empirical probabilities for event types found in Section 4 be proven mathematically? Can we also mathematically model the behavior when size distribution is non-uniform? What about tight or split variations in sizes?
- What effect does distribution in size caused by inside and outside crossovers have on reduction – more precisely, where does it fit in the process.
- Since the proportion of inside and outside crossover events appears equal is there something about the length distribution within these crossover types which drives reduction. Works by Poli and McPhee previously cited appear to be relevant here.

Future efforts to answer these questions may help us better understand how the reduction behavior in the chunking GA and provide insights into how variable-length GAs in general perform.

References

Bassett, J. K. & De Jong, K. A. (2000). Evolving behaviors for cooperating agents. In *International Syposium on Methodologies for Intelligent Systems*, pages 157-165.

Blickle, T. & Thiele, L. (1994). Genetic programming and redundancy. In *Proceedings of Genetic Algorithms within the Framework of Evolutionary Computation*, pages 33-38, Workshop at KI-94, Saarbrucken. Max-Planck-Institut fur Informatik (MPI-I-94-241).

Burke, D. S., De Jong, K. A., Grefenstette, J. J., Ramsey, C. L. & Wu, A. S. (1998). Putting more genetics into genetic algorithms. *Evolutionary Computation*, 6(4):387-410.

Goldberg, D. E., Korb, B. & Deb, K. (1989). Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, 3(5):493-530.

Grefenstette, J. J., Ramsey, C. L. & Schultz, A. C. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5:355-381.

Harvey, I. (1992). The SAGA Cross: The mechanics of recombination for species with variable-length genotypes. In *Proceedings of Parallel Problem Solving from Nature II*, pages 269-278. North-Holland.

Koza, J. R. (1992). *Genetic Programming*. MIT Press, Cambridge, MA.

Langdon, W. B. & Poli, R. (1997). Fitness causes bloat. In *Proceedings of the Second On-line World Conference on Soft Computing in Engineering Design and Manufacturing*, pages 13-22. Springer-Verlag, London.

Langdon, W. B. (2000). Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines*, 1(2):95-119.

Luke, S. & Panait, L. (2002). Fighting bloat with nonparametric parsimony pressure. In *Proceedings of Parallel Problem Solving from Nature VII*, LNCS 2439, pages 411-420, Granada, Spain. Springer-Verlag, Heidelberg.

Luke, S. (2003). Modification point depth and genome growth in genetic programming. *Evolutionary Computation*, 11(1):67-106.

McPhee, N. F. & Poli, R. (2001). A schema theory analysis of the evolution of size in genetic programming with linear representations. In *Genetic Programming, Proceedings of EuroGP'2001*, 2038, pages 108-125. Springer-Verlag.

Nordin, P. & Banzhaf, W. (1995). Complexity compression and evolution. In *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*, pages 310-317. Morgan Kaufmann.

Panait, L. & Luke, S. (2004). Alternative bloat control methods. In *GECCO 2004: Proceedings of the Genetic and Evolutionary Computation Conference*, LNCS 3103, pages 630-641, Seattle, Washington. Springer, Berlin, Germany.

Poli, R. & McPhee, N. F. (2001). Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In *Genetic Programming, Proceedings of EuroGP'2001*, 2038, pages 126-142. Springer-Verlag.

Ramsey, C. L., De Jong, K. A., Grefenstette, J. J., Wu, A. S. & Burke, D. S. (1998). Genome length as an evolutionary self-adaptation. In *Proceedings of Parallel Problem Solving from Nature V*, LNCS 1498, pages 345-353, Amsterdam, Netherlands. Springer-Verlag, Heidelberg.

Soule, T. & Foster, J. A. (1998). Removal Bias: a new cause of code growth in tree based evolutionary programming. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*, pages 781-786. IEEE Press.

Soule, T., Foster, J. A. & Dickinson, J. (1996). Code growth in genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 215-223. MIT Press.

Stringer, H. & Wu, A. S. (2004). Winnowing wheat from chaff: The Chunking GA. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, LNCS 3103, pages 198-209, Seattle, WA. Springer, Berlin.

Tackett, W. A.(1994). *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, University of Southern California, Department of Electrical Engineering Systems.

Wu, A. S. & Garibay, I. (2002). The proportional genetic algorithm: Gene expression in a genetic algorithm. *Genetic Programming and Evolvable Hardware*, 3(2).

Wu, A. S. & Stringer, H. (2002). Learning using chunking in evolutionary algorithms. In *Proceedings of the 11th Conference on Computer-Generated Forces and Behavior Representations*, pages 243-254, Orlando, FL.

Wu, A. S., Schultz, A. S. & Agah, A. (1999). Evolving control for distributed micro air vehicles. In *In Proceedings of IEEE Computational Intelligence in Robotics and Automation Engineers Conference, 1999*.