

Inducing Sequentiality Using Grammatical Genetic Codes

Kei Ohnishi, Kumara Sastry, Ying-Ping Chen, and David E. Goldberg

Illinois Genetic Algorithms Laboratory (IlliGAL)
University of Illinois at Urbana-Champaign
104 S. Mathews Ave, Urbana, IL 61801, USA
{kei,kumara,ypchen,deg}@illigal.ge.uiuc.edu

Abstract. This paper studies the inducement of *sequentiality* in genetic algorithms (GAs) for uniformly-scaled problems. Sequentiality is a phenomenon in which sub-solutions converge sequentially in time in contrast to uniform convergence observed for uniformly-scaled problems. This study uses three different grammatical genetic codes to induce sequentiality. Genotypic genes in the grammatical codes are interpreted as phenotypes according to the grammar, and the grammar induces sequential interactions among phenotypic genes. The experimental results show that the grammatical codes can indeed induce sequentiality, but the GAs using them need exponential population sizes for a reliable search.

1 Introduction

Identification and exchange of important building blocks (BBs) is one of the key challenges in the design of genetic algorithms (GAs). Fixed recombination operators that do not adapt linkage of BBs have been shown to be inadequate and scale-up exponentially with the problem size [1]. Furthermore, GAs that adaptively identify and efficiently exchange BBs successfully solve boundedly difficult problems, usually requiring only polynomial number of function evaluations [2]. GAs that identify and exchange BBs and thereby solve difficult problems quickly, reliably, and accurately are called *competent* GAs [3].

One of the approaches to achieve competence is by means of linkage learning GA (LLGA) [4]. The LLGA takes the position that tightly linked BBs are evolutionally advantageous. The LLGA is designed to achieve tight linkage between interacting variables. While the LLGA has been successful in solving non-uniformly scaled problems, it can only solve uniformly scaled problems of limited size [5,6]. In non-uniformly-scaled problems, since a selection operator identifies BBs sequentially, it helps the LLGA achieve tight linkage. However, in uniformly-scaled problems, a selection operator identifies BBs simultaneously. Therefore, it is difficult for the LLGA to achieve tight linkage for all BBs in parallel [7].

Recently, a genetic algorithm using grammatical evolution (GAuGE) [8,9], which was inspired by grammatical evolution [10,11], has been proposed to solve problems through a process of getting salient phenotypic genes clustered in a

genotypic chromosome. The GAuGE relies on a grammatical genetic code in which genes are in a certain order interpreted according to the grammar, and the grammar induces sequential interactions among phenotypic genes corresponding to their determined order. In addition, the grammatical genetic code allows phenotypic genes to locate at any positions in a genotypic chromosome. If salient phenotypic genes get clustered on a specific part of a genotypic chromosome, they can be kept from their disruption due to a specific crossover operator as well as grammatical decoding.

We hypothesized that sequential interactions among phenotypic genes induced by grammar could induce prioritized phenotypic convergence for search problems including uniformly-scaled problems. Therefore, the objective of this paper is to investigate whether or not *sequentiality* can be induced in uniformly-scaled problem using grammatical genetic codes. Sequentiality is a phenomenon in which sub-solutions converge sequentially in time. The grammatical genetic codes used in this paper are based on similar principal as in the GAuGE.

This paper is organized as follows. Section 2 briefly describes studies on grammatical genetic codes and sequentiality. In section 3, three grammatical genetic codes used in this paper are explained. We empirically examine if GAs using the grammatical codes can induce sequentiality in section 4. Finally, we summarize our results and draw our conclusions.

2 Related Studies

Representation of the variables of a search problem play an important role in genetic and evolutionary search, and effects of a variety of genetic codes on the performance of GAs have been extensively studied. An exhaustive review of studies on genetic representations is beyond the scope of this paper and the reader is referred elsewhere [12,13,14] and to the references therein.

One of the motivations for this study came from [15,16], in which GAs with seemingly disruptive and highly epistatic genetic codes were successful in solving difficult combinatorial problems. Some researchers have also used grammar-based genetic codes, which are also highly epistatic, with reasonable GA success [17,8]. In [17], the genes encode production rules, which are in turn used in a fixed manner to generate a structured phenotype.

The grammar-based genetic code used in the GAuGE [8,9] allows phenotypic genes to locate at any positions in a genotype chromosome similar to the representation used by Goldberg and Lingle [18] and the representations used in messy GAs [14] and the LLGA [4]. The grammar in GAuGE also induces sequential interactions among phenotypic loci, which is determined by the genotype-to-phenotype decoding procedure. LINKGAUGE [19], which is a variant of GAuGE uses grammars that induce sequential interactions not only among phenotypic alleles, but also among phenotypic loci.

At the first glance, it looks like such highly epistatic genetic codes should yield poor results. However, based on their empirical success, we wondered if such genetic codes might be simplifying the search problem by implicitly focusing on

a single or few subproblems at a time. That is, we hypothesized that the genetic codes with high epistasis might be inducing sequentiality into search problems, which we investigate in this paper.

3 Grammatical Codes

Since we would like to verify if the strength of sequential interactions among phenotypic genes is directly related to inducing sequentiality, we employ three kinds of grammatical genetic codes which induce sequential interactions among phenotypic genes with different strength. The three codes are : (1) GAuGE code which is slight variant of [8], (2) complex grammatical code, and (3) cellular grammatical code. The codes (2) and (3) are meant to induce stronger interactions among phenotypic genes than the GAuGE code.

All the grammatical codes use integers as the genotypic genes, and all the genotypes are decoded from left to right. The grammatical codes (1) and (2) determine both the phenotypic loci and their alleles by applying modulus operation (%) to integers which are obtained in the decoding process. Interactions among phenotypic genes which are common to all the codes comes from relative phenotypic loci. All the phenotypic loci are labeled as integers, and they are relabeled every time one phenotypic locus is occupied. Those grammatical codes are in detail explained below, where a ℓ -bit optimization problem is assumed.

(1) GAuGE Code (Base 10)

The difference between the original GAuGE code proposed in [8] and the one used here is that in the original GAuGE, every integer, which is 0 to 255, is encoded into an eight-bit binary number. Here we directly use a base 10 integer from 1 to ℓ . That is, a GAuGE genotype used here can be written as $(p_1, v_1, p_2, v_2, \dots, p_\ell, v_\ell)$, where $p_q, v_q \in [1, \ell]$ and $q = 1, 2, \dots, \ell$. In this code, there are sequential interactions only among the phenotypic loci. The decoding procedure is as follows.

1. Let q be 1.
2. When $1 \leq q \leq \ell$, the unoccupied phenotypic loci are labeled as integers in $[1, \ell - q - 1]$ from left to right, which is as $(1, 2, \dots, \ell - q - 1)$. The locus and its allele are determined as $p_q \% (\ell - q - 1) \in [1, \ell - q - 1]$, which represents one of the labels of the unoccupied loci, and $v_q \% 2 \in \{0, 1\}$, respectively.
3. In the case of $q = \ell$, the whole decoding process ends. Otherwise q increases by one, and return to procedure 2.

(2) Complex Grammatical Code

A genotype in the complex grammatical code consists of $\ell + 1$ real and imaginary parts in complex numbers and ℓ operations applied to two complex numbers. The genotypic genes are arranged as $(r_1, i_1, o_1, \dots, r_\ell, i_\ell, o_\ell, r_{\ell+1}, i_{\ell+1})$, where $r_* \in [1, 3]$ represents the real part, $i_* \in [1, \ell]$ is the imaginary part, and $o_* \in \{\times, \times_\ell\}$ is the operation. In this code, there are sequential interactions among both the phenotypic loci and their alleles. This decoding procedure is described below. Since the decoding is done through ℓ iterations, the iteration number is denoted by $q = 1, 2, \dots, \ell$.

1. Let q be 1.
2. In the case of $q = 1$, a new complex number is calculated as $R_1 + I_1j = (r_1 + i_1j) \times (r_2 + i_2j)$ no matter what o_1 is, where j is a imaginary number. A phenotypic locus is obtained as $P_1 = |I_1|\% \ell + 1 \in [1, \ell]$, which points out one of the phenotypic loci labeled as 1 to ℓ from left to right. An allele at the locus is obtained as $V_1 = |R_1|\% 2 \in \{0, 1\}$. If $o_1 = \times$, a new complex number is defined as $rr_2 + ii_2j = (|R_1|\% 3 + 1) + (|I_1|\% \ell + 1)j$. If $o_1 = \times_t$, a new complex number is defined as $rr_2 + ii_2j = r_2 + i_2j$.
3. In the case of $2 \leq q \leq \ell$, a new complex number is calculated as $R_q + I_qj = (rr_q + ii_qj) \times (r_{q+1} + i_{q+1}j)$. A phenotypic locus is obtained as $P_q = |I_q|\% (\ell - q - 1) + 1 \in [1, \ell - q]$, which points out one of the unoccupied loci relabeled as 1 to $(\ell - q)$ from left to right. An allele at the locus is obtained as $V_q = |R_q|\% 2 \in \{0, 1\}$. If $o_q = \times$, a new complex number is defined as $rr_{q+1} + ii_{q+1}j = (|R_q|\% 3 + 1) + (|I_q|\% \ell + 1)j$. If $o_q = \times_t$, a new complex number is defined as $rr_{q+1} + ii_{q+1}j = r_q + i_qj$.
4. In the case of $q = \ell$, the whole decoding process ends. Otherwise q increases by one, and return to procedure 3.

(3) Cellular Grammatical Code

A genotype in this code is interpreted as a system which is composed of a series connection of simple cellular automata. Each cellular automaton, $C_q (q = 1, 2, \dots, \ell)$, is composed of four transition rules and an output timing. The inputs to the cellular automata, the outputs from them, and their inside states are represented by integers in a range of $[1, 4]$. The transition rules convert one integer ($\in [1, 4]$) into another one ($\in [1, 4]$). Therefore, integers ($\in [1, 4]$) are propagated among the cellular automata. The transition rules have not only their outputs but also information on a phenotypic locus and its allele, so that each cellular automaton can determine a phenotypic locus and its allele at its output timing. The output timing is also an integer ($\in [1, 8]$), which represents the number of the transitions. In this code, there are sequential interactions among both the phenotypic loci and their alleles. This decoding procedure is described below. Since the decoding is done through ℓ iterations, the iteration number is denoted by $q = 1, 2, \dots, \ell$.

1. In the case of $q = 1$, the initial input is given to the first cellular automaton C_1 . In the case of $2 \leq q \leq \ell$, the output of the $(q - 1)$ -th cellular automaton is give to the q -th one as its input.
2. When the input value to the cellular automaton is $i_q \in [1, 4]$, the i_q -th transition rule is activated. The state of the cell moves from i_q to $s_1 \in [1, 4]$ according to the i_q -th transition rule. This state transition is repeated until the number of times of the state transitions reaches the output timing o_t . After reaching o_t , the current state of the cell $s_{o_t} \in [1, 4]$ becomes the input value $i_{q+1} = s_{o_t}$ to the next cell C_{q+1} . Finally, one more the state transition is done according to the s_{o_t} -th transition rule, and the phenotypic locus and its allele are determined as $p_{o_t+1} \in [1, \ell - q]$, which represents one of the labels of the unoccupied loci labeled as 1 to $\ell - q$, and its allele $v_{o_t+1} \in [0, 1]$ that the s_{o_t+1} -th transition rule has, respectively.
3. In the case of $q = \ell$, the whole decoding process ends. Otherwise q increases by one, and return to procedure 1.

4 Experiments

4.1 Test Problems

We use three types of uniformly-scaled problems for investigating the GAs using the grammatical codes. Those are (1) OneMax problem with ℓ bits, (2) 4-bit trap deceptive function with tightly linked m BBs [20], and (3) 4-bit trap deceptive function with loosely linked m BBs. They are thereafter called OneMax- ℓ , $(m, 4)$ -Trap-T, and $(m, 4)$ -Trap-L, respectively.

(1) OneMax Problem with ℓ Bits (**OneMax- ℓ**)

This problem gives the number of ones in the phenotypes to their corresponding genotypes as their fitness values.

(2) 4-bit Trap Deceptive Function with Tightly Linked m BBs (**$(m, 4)$ -Trap-T**)

A BB in the phenotype consists of four bits, and each BB is close to one another like (B_1, B_2, \dots, B_m) , where B_q is the q -th BB. A fitness value of a genotype is the sum of fitness values that m BBs give. A fitness value of each BB is calculated in the same way. When the number of ones in a BB is u , the fitness value of the BB, $f_{BB}(u)$, is given by

$$f_{BB}(u) = \begin{cases} 4 & u = 4, \\ 3 - u & \textit{otherwise}. \end{cases}$$

(3) 4-bit Trap Deceptive Function with Loosely Linked m BBs (**$(m, 4)$ -Trap-L**)

A BB in the phenotype consists of four bits, and each BB is distant from one another. Concretely, the q -th BB is denoted by $(q, q + \ell/4, q + \ell/2, q + 3\ell/4)$, where $q = 1, 2, \dots, \ell/4$, $\ell (= 4m)$ is the length of the phenotype, and each element in that vector notation of the BB represents a phenotypic locus. A fitness value of a BB is calculated in the same way as done in $(m, 4)$ -Trap-T.

4.2 Genotype-Phenotype-Mapping Characteristics

First of all the experiments, the characteristics of genotype-phenotype mappings of the three grammatical codes are examined. We observe two things: (1) how many small perturbations in the genotypes change their phenotypes, which was called *locality* in [21], and (2) how many small perturbations in the genotypes change their fitness values when concrete optimization problems are assumed. We use OneMax-32, $(8, 4)$ -Trap-T, and $(8, 4)$ -Trap-L as test problems. The experimental procedure is as follows:

1. A genotype is randomly generated, and then its phenotype is obtained by the genotype-phenotype-mapping. The genotype and phenotype generated are called *original genotype* and *original phenotype*, respectively. Also, the fitness value of the original phenotype, which is called *original fitness value*, is calculated.
2. A new genotype is obtained by modifying an allele at a certain locus in the original genotype, and then its phenotype is obtained. Also, the fitness value of the new phenotypes is calculated. The difference between the original and the new genotypes is just one allele at the chosen locus.

3. A Hamming distance between the original and the new generated phenotypes is calculated. Absolute value of the difference between two fitness values that the original and the new phenotypes have is calculated.
4. Iterating the procedure 2 to 3 until all the genotypes that are adjacent to the original genotype are generated and compared with the original genotype.
5. Iterating the procedure 1 to 4 until 100 original genotypes are generated and compared with all the genotypes adjacent to them.

The experimental results are shown in Fig. 1, which represents two things: (1) the averaged Hamming distance between the original phenotype and each of the other phenotypes corresponding to all the genotypes adjacent to the original one, and (2) the averaged difference between the original fitness value and each of the other fitness ones that all the genotypes adjacent to the original one have. The two kinds of averaged values were calculated for each genotypic locus.

Figure 1(a)-1(c) show that small perturbations on the left parts of the genotypes in the three grammatical codes caused bigger changes in their phenotypes than small perturbation on the right parts of them.

However, the changes in the fitness values that result from the changes in the phenotypes were not always like the ones in the phenotypes. When we assumed OneMax-32 and (8, 4)-Trap-T, the changes in the fitness values were almost flat over all the genotypic loci for almost all the grammatical codes used (Figure 1(d)-1(i)). When we assumed (8, 4)-Trap-L, the more left the genotypic loci were, the bigger the changes in the fitness values became (Figure 1(j)-1(h)). In this case, we could say that uniformly-scaled problems became non-uniformly-scaled ones at least in the local regions of their genotype spaces.

4.3 Genetic Algorithm

The results shown in the previous section suggest that the three grammatical codes have genotype-phenotype-mappings that give low correspondences between their genotypic and phenotypic neighborhoods. In addition, fitness landscapes on their genotypic spaces have multi-modalities because the three codes are redundant genetic representations. As a result, the fitness landscapes on their genotypic spaces should be highly rugged ones. We now briefly describe the GA used in this paper to investigate the grammatical genetic codes to induce sequentiality.

Minimal generation gap model (MGG) [22] is used as a generation gap model. This model literally minimizes a generation gap. Since it seems that fitness landscapes on the grammatical genetic codes are highly rugged, this generation gap model should be better than ones which change GA populations drastically at generation gaps. We will thereafter regard generating genotypes amounting a population size as one generation.

We use a one-point crossover operator from the viewpoint of not exploiting genotypes but minimizing the disruption of good genetic materials in the left part of the genotype. Since there are sequential interactions among the genotypic genes from left to right, the genes in the left part of the genotype should be kept from their disruption. A one-point crossover operator should be suitable from this point. Mutation operator is not used.

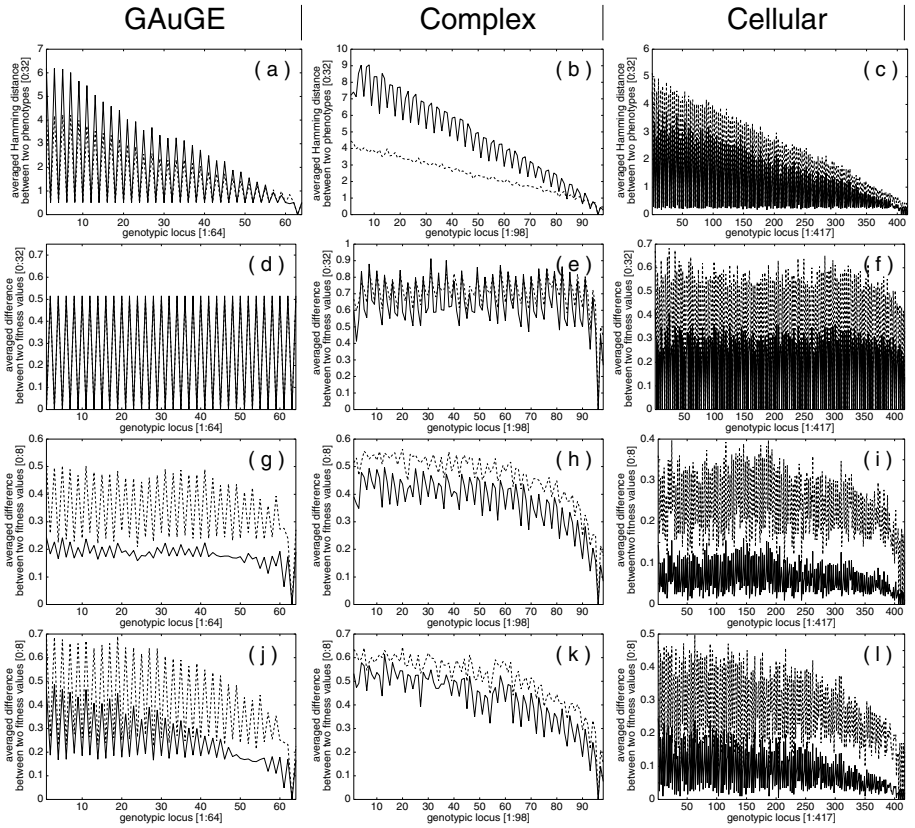


Fig. 1. Genotype-phenotype-mapping characteristics of the three grammatical codes. Labels of *GAuGE*, *Complex*, and *Cellular* represent the results for the GAuGE, the complex, and the cellular grammatical codes, respectively. The sub-figures (a)(b)(c) represent the averaged Hamming distance between the original phenotype and each of the other phenotypes corresponding to all the genotypes adjacent to the 100 original ones. The other sub-figures from (d) to (l) represent the averaged difference between the original fitness value and each of the other fitness ones that all the genotypes adjacent to the 100 original ones have. The sub-figures (d)(e)(f), (g)(h)(i), and (j)(k)(l) are for OneMax-32, (8, 4)-Trap-T8, and (8, 4)-Trap-L, respectively. Those two kinds of averaged values (**solid lines**) were calculated for each genotypic locus. When there are K genotypic alleles at a certain locus, $K - 1$ genotypes adjacent to an original genotype are obtained by modifying the allele at the locus in the original one. Then averaged value over $K - 1$ comparisons is obtained for the locus. The same procedure is applied to 100 original genotypes, so that the 100 averaged values are obtained. The final averaged value for the locus is obtained by averaging the 100 values. Standard deviations of the observed values are also plotted in all the sub-figures (**dash lines**).

4.4 Inducing Sequentiality

We examine if sequentiality is actually induced in uniformly-scaled problems by the GAs using the grammatical genetic codes. A GA using an identical map between a genotype and a phenotype spaces, which is called *standard GA* thereafter, is also used to compare with them. The population size is appropriately sized so that 95 out of 100 independent runs converge to the optimum. We observe convergence of both phenotypic alleles and loci. As for phenotypic alleles, we obtain averaged generations at which proportion of correct BBs or bits in the GA population are over 0.9. Since all the BBs or bits do not always converge in a fixed order, the generations at which the BBs or bits converged are sorted in ascending order and the sorted generations at the same order are averaged. As for phenotypic loci, we obtain averaged proportion of loci into which a set of genotypic genes at some order from the most left (1st) set in the GA population are mapped the most frequently. Those two averaged values are calculated using data of success runs out of 100. OneMax-80, (8, 4)-Trap-T, and (8, 4)-Trap-L are used here. The experimental results for the convergence of the phenotypic alleles and loci are shown in Table 1 and Fig. 2, respectively.

Table 1. Averaged generations over success runs out of 100 at which proportion of correct BBs or bits in the GA population were over 0.9. As for OneMax-80, the generations for 1st, 10th, 20th, 30th, 40th, 55th, 70th, and 80th converged bits are shown. As for (8, 4)-Trap-*, the generations for all the converged BBs are shown.

OneMax-80	pop. size	1st	10th	20th	30th	40th	55th	70th	80th
Standard GA	500	10.53	13.64	15.41	17.26	19.36	23.45	29.66	39.33
GAuGE	500	14.44	18.03	19.78	21.10	22.46	24.61	27.67	35.02
Complex	500	14.61	18.39	20.17	21.49	22.89	25.13	28.61	36.65
Cellular	2000	22.20	28.30	31.68	34.44	37.34	42.29	49.75	66.54
(8, 4)-Trap-T	pop. size	1st	2nd	3rd	4th	5th	6th	7th	8th
Standard GA	500	15.49	16.86	18.22	19.23	20.47	21.74	23.30	25.36
GAuGE	22000	33.70	35.44	36.54	38.09	39.29	42.08	44.65	48.94
Complex	22000	32.52	34.26	35.57	37.15	38.89	40.61	43.37	47.65
Cellular	50000	38.64	40.96	43.13	45.24	48.09	51.85	56.31	70.09
(8, 4)-Trap-L	pop. size	1st	2nd	3rd	4th	5th	6th	7th	8th
Standard GA	540000	39.26	40.98	42.65	43.94	45.59	48.03	51.41	57.94
GAuGE	32000	34.74	36.26	37.48	38.58	39.88	41.64	44.25	47.76
Complex	32000	34.08	35.89	37.26	38.30	39.46	41.15	43.58	46.90
Cellular	80000	38.84	41.08	43.29	45.03	46.97	50.18	54.52	63.73

Table 1 shows that the GAs using the grammatical code can induce sequentiality. However, since the standard GA also induced sequentiality, we can not conclude that the grammatical genetic code is the only factor to induce sequentiality. It is suggested that the genetic operators used, especially the generation gap model, could also be a possible factor.

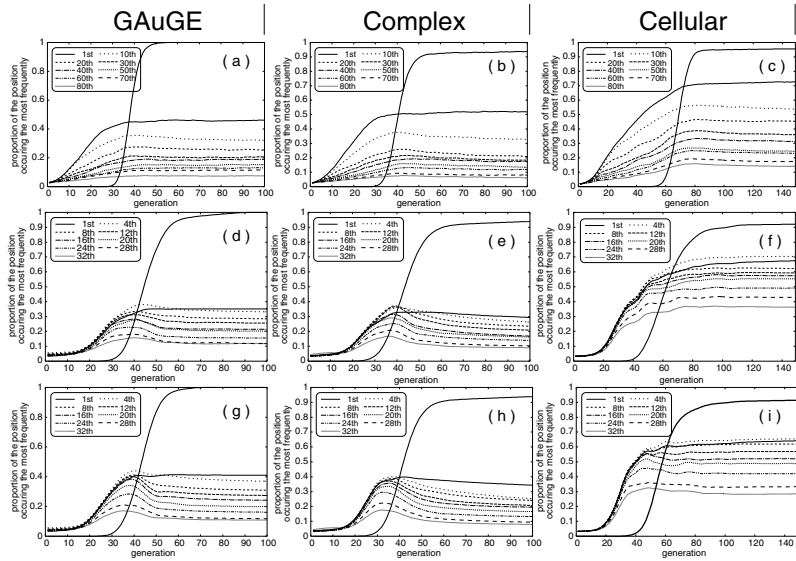


Fig. 2. Averaged proportion of loci into which the q -th set of genotypic genes from the most left (1st) set are mapped the most frequently over success runs out of 100. Labels of “GAuGE”, “Complex”, and “Cellular” mean the results for the GAuGE, the complex, and the cellular grammatical codes, respectively. As for OneMax-80 (the sub-figures (a)(b)(c)), the proportions for the 1st, 10th, 20th, 30th, 40th, 50th, 60th, 70th, and 80th sets of genotypic genes are plotted. As for (8,4)-Trap-T ((d)(e)(f)) and (8,4)-Trap-L ((g)(h)(i)), the proportions for the 1st, 4th, 8th, 12th, 16th, 20th, 24th, 28th, and 32th sets of genotypic genes are plotted. Also, the proportion of the genotypes with the optimal fitness value in the GA population is plotted in every sub-figure (**the thickest solid line**).

Figure 2 shows that the more left the genotypic genes are located in the genotypes, the more frequently they are mapped into the same locus. It is suggested that this fixations of the loci should be essential to induce sequentiality. However, as especially in the GAs using the GAuGE and complex codes for (8,4)-Trap-T and (8,4)-Trap-L, the degree of the fixations of the loci was low, and a variety of genotypes resided together in the GA populations even when the fitness values of all the genotypes almost converged. Considering the low degree of the loci fixations, it could be thought that the big reliable population sizes for (8,4)-Trap-T and (8,4)-Trap-L should result from the fact that the crossover operator used was not able to mix the genotypes effectively due to the lack of the mechanism to fix the loci properly.

4.5 Reliable Population Size

In the previous section, we verified that the GAs using the grammatical codes can induce sequentiality. However, the scalability of the GAs has not been revealed. Therefore, we examine population sizes with which the GAs using grammatical

codes can reliably find global optima for given optimization problems. The reliable population sizes are determined as minimal population ones with which the GAs succeed in finding global optima for given optimization problems over 95 times out of 100 runs. The experimental results are shown in Table 2.

Table 2. The reliable population sizes for OneMax-40,60,80, $(m, 4)$ -Trap-T ($m = 4, 6, 8$), and $(m, 4)$ -Trap-L ($m = 4, 6, 8$). The reliable population size is determined as a minimal population size with which each GA can find global optima for the given optimization problems over 95 times out of 100 runs.

	OneMax- ℓ			$(m, 4)$ -Trap-T			$(m, 4)$ -Trap-L		
	40 bits	60 bits	80 bits	4 BBs	6 BBs	8 BBs	4 BBs	6 BBs	8 BBs
Standard GA	150	280	460	120	250	400	6000	50000	540000
GAuGE	140	280	420	1300	6000	20000	1400	7000	30000
Complex	140	300	460	1400	5000	18000	1000	5000	26000
Cellular	380	900	1700	1800	11000	48000	1400	12000	76000

From Table 2, we can predict that the reliable population sizes of the GAs using the three grammatical genetic codes for $(m, 4)$ -Trap-T and $(m, 4)$ -Trap-L exponentially increase with problem size. The function evaluations that need to find the global optima can be predicted to increase exponentially as well, though those data are not shown in this paper. In terms of scalability, the GAs using the three grammatical codes are impractical for GA-hard uniformly-scaled problems. However, we should examine the performances of them when smaller cardinal numbers are used for representing their genotypic genes in the further work.

5 Summary and Conclusion

We empirically examined grammatical genetic codes as one of the factors that induce sequentiality in uniformly-scaled problems. The factors to induce sequentiality are manifold, such as optimization problems, genotype-phenotype-mapping, population size, and genetic operators. This work focused on genotype-phenotype-mapping, and empirically observed their effects on sequentiality. The observed effects are: (1) the grammatical codes get uniformly-scaled problems to be non-uniformly-scaled ones, and help GAs induce sequentiality, (2) the genetic operators used help GAs induce sequentiality, and (3) impractical population sizes are needed for a successful search with sequentiality.

The results suggest that while the grammatical codes help GAs induce sequentiality together with the genetic operators, they are not enough to cause strong fixations of the genotypic genes for a recombination operator to mix the genotypes effectively, so that the GAs using the grammatical codes scale-up exponentially with problem size. On the other hand, selectomutative GAs might be more economical for grammatical codes in which genotypic genes are represented by integers than selectorecombinative GAs. If we are waiting for

discovery of good genes block one after another by mutation, large population size might not really be needed. Therefore, by using mutation, we can do away with smaller populations, but might require longer time—in terms of number of generations—than in the case of crossover.

Our results are also useful in isolating some of the features of grammatical evolution (GE). One of the attributes for the success of GE might be a balanced mixture of inherent interactions among components of a program and interactions induced by grammar. Furthermore, unlike integer codes, the use of binary-coded genotypic genes in GE likely bring diversity and flexibility into search. Finally, the selectomutative part of GE might also be playing a more important role than it appears on a first glance.

Acknowledgments. This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129, and by the Technology Research Center (TRECC), a program of the University of Illinois at Urbana-Champaign, administered by the National Center for Supercomputing Applications (NCSA) and funded by the Office of Naval Research under grant N00014-01-1-0175. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, or the U.S. Government.

References

1. Thierens, D., Goldberg, D.E.: Mixing in genetic algorithms. In: Proceedings of the 5th International Conference on Genetic Algorithms (ICGA-93). (1993) 38–45
2. Goldberg, D.E.: The race, the hurdle, and the sweet spot: Lessons from genetic algorithms for the automation of design innovation and creativity. *Evolutionary Design by Computers* (1999) 105–118
3. Goldberg, D.E.: *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA (2002)
4. Harik, G.R., Goldberg, D.E.: Learning linkage. *Foundations of Genetic Algorithms 4* (1996) 247–262
5. Harik, G.R.: Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms. PhD thesis, University of Michigan, Ann Arbor (1997) Also IlliGAL Report No. 97005.
6. Chen, Y.P., Goldberg, D.E.: Introducing start expression genes to the linkage learning genetic algorithm. In: Proceedings of Parallel Problem Solving from Nature VII. (2002) 351–360
7. Chen, Y.P., Goldberg, D.E.: Convergence time for the linkage learning genetic algorithm. IlliGAL Report No. 2003025, Illinois Genetic Algorithms Lab., Univ. of Illinois, Urbana, IL (2003)
8. Ryan, C., Nicolau, M., O'Neill, M.: Genetic algorithms using grammatical evolution. In: Proceedings of the Fifth European Conference on Genetic Programming (EuroGP 2002). (2002) 278–287

9. Nicolau, M., Ryan, C.: How functional dependency adapts to salience hierarchy in the GAuGE system. In: Proceedings of the Sixth European Conference on Genetic Programming (EuroGP 2003). (2003) 153–163
10. Ryan, C., Collins, J., O'Neill, M.: Grammatical evolution: Evolving programs for an arbitrary language. In: Proceedings of the First European Conference on Genetic Programming. (1998) 83–96
11. O'Neill, M., Ryan, C.: Grammatical evolution. *IEEE Transactions on Evolutionary Computation* **5** (2001) 349–358
12. Rothlauf, F., Goldberg, D.E.: Representations for Genetic and Evolutionary Algorithms. Physica-Verg, Heidelberg, New York (2002)
13. Whitley, D., Rana, S., Heckendorn, R.: Representation issues in neighborhood search and evolutionary algorithms. In: Genetic Algorithms and Evolution Strategy in Engineering and Computer Science. John Wiley & Sons Ltd, West Sussex, England (1997) 39–58
14. Goldberg, D.E., Korb, B., Deb, K.: Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems* **3** (1989) 493–530
15. Anderson, P.G.: Ordered greed. In: Proceedings of Third International ICSC Symposium on Soft Computing. (1999)
16. Anderson, P.G.: Ordered greed, ii: Graph coloring. In: Proceedings of the International NAISO Congress on Information science innovations (ISI2001). (2001)
17. Kitano, H.: Designing neural networks using genetic algorithms with graph generation system. *Complex Systems* **4** (1990) 461–476
18. Goldberg, D.E., Lingle, Jr., R.: Alleles, loci, and the traveling salesman problem. In: Proceedings of an International Conference on Genetic Algorithms and Their Applications. (1985) 154–159
19. Nicolau, M., Ryan, C.: LINKGAUGE: Tackling hard deceptive problems with a new linkage learning genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference 2002 (GECCO 2002). (2002) 488–494
20. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. *Foundations of Genetic Algorithms* **2** (1993) 93–108
21. Rothlauf, F.: Towards a Theory of Representations for Genetic and Evolutionary Algorithms— Development of Basic Concepts and their Application to Binary and Tree Representations. Unpublished doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL (2001)
22. Satoh, H., Yamamura, M., Kobayashi, S.: Minimal generation gap model for GAs considering both exploration and exploitation. In: Proceedings of the International Conference on Fuzzy Systems, Neural Networks and Soft Computing (Iizuka'96). (1996) 494–497