Adaptation of length in a nonstationary environment

Han Yu[1], Annie S. Wu[1], Kuo-Chi Lin[2], and Guy Schiavone[2]

[1] School of EECS, University of Central Florida, Orlando, FL 32816-2362
{hyu, aswu}@cs.ucf.edu
[2] Inst. for Simulation & Training, Univ. of Central Florida, Orlando, FL 32826-0544
{klin, guy}@pegasus.cc.ucf.edu

Abstract. In this paper, we examine the behavior of a variable length GA in a nonstationary problem environment. Results indicate that a variable length GA is better able to adapt to changes than a fixed length GA. Closer examination of the evolutionary dynamics reveals that a variable length GA can in fact take advantage of its variable length representation to exploit good quality building blocks after a change in the problem environment.

1 Introduction

A typical genetic algorithm (GA) tends to use problem representations that are orderly, fixed, and somewhat arbitrary. Individuals are of a fixed length with information encoded at fixed, programmer-defined locations on the individuals. These representations tend to be very efficient, well organized, and encoded in ways that are very logical or easy for humans to interpret. Extending a GA to use a variable length problem representation brings about a host of new issues that must be addressed, including how to encode information and modifications to traditional genetic operators. Nevertheless, the advantages of a more adaptable and evolvable problem representation are thought to outweigh the additional effort.

In this paper, we explore the adaptability of a variable length representation in a nonstationary environment. Previous work has suggested that in periods of heavy search, e.g. those periods immediately following a change in the environment or target solution, a variable length GA will tend to favor longer individuals because longer individuals provide more resources to the search process [1, 2]. We test this theory on a variable length GA applied to the problem of multiprocessor task scheduling [3]. Although our initial results are somewhat surprising, a detailed analysis of the evolutionary dynamics provide an interesting and positive explanation.

2 Related work

Early work on variable length representations includes Smith's LS-1 learning system [4], Goldberg's messy GA [5], Koza's genetic programming (GP) [6],

and Harvey's SAGA [7]. These studies laid much of the groundwork in terms of defining the issues that need to be addressed with variable length representations and exploring potential solutions.

Since then, much research has been conducted on the variation of individual length during evolution. Langdon and Poli [2] explore the cause of bloat in the variable length genetic programming (GP) representation. They conclude that longer individuals are favored in the selection process because they have more ways to encode solutions than shorter individuals. Soule et al. [8, 9] perform a detailed investigation on code growth in GP and conclude that code growth is dominated by non-functional code. As a result, parsimony pressure can affect the search quality in GP. The relationship between size and fitness in a population is useful in predicting the GP's search performance in a long run. Burke et al. [1] study the adaptation of length in a variable length GA. They found that, without parsimony pressure, a GA tends to generate individuals of arbitrary length. If parsimony pressure is applied, the average individual length increases quickly in early evolution, followed by a gradual decrease until stablization. The early increase was thought to be a period of growth of resources: increasing individual length increases the probability of finding building blocks. All of the above studies examine variable length representation in a stable problem environment.

A variety of studies have looked at GA behavior in changing environments. Some of these approaches focus on maintaining the population diversity during GA search, such as the use of random immigrants [10], hypermutation [11], adaptive GA operators [12], and the TDGA [13, 14]. Other strategies attempt to improve the search by storing duplicate information with redundant representation schemes [15–17], using alternative memory systems [18], or encouraging the maintenance of multiple "species" within a GA population [19–21].

3 Problem description

We perform our experiments on a variable length GA applied to the problem of multiprocessor task scheduling [22]. The task scheduling problem begins with a task dependency graph which specifies the dependencies among a number of tasks that together compose a larger complete task. Figure 1 shows two example task dependency graphs. The goal of the GA is to assign tasks to a set of available parallel processors such that all tasks can be completed and total execution time is minimized. The data dependencies that exist among tasks place restrictions on the order in which tasks can be assigned to processors. Dependent tasks that are assigned to different processors may incur additional communication delays.

4 Implementation details

Our GA is based on a traditional generational GA, but extended to use a flexible variable length representation. Our extensions beyond a basic GA are described below along with the fitness function.

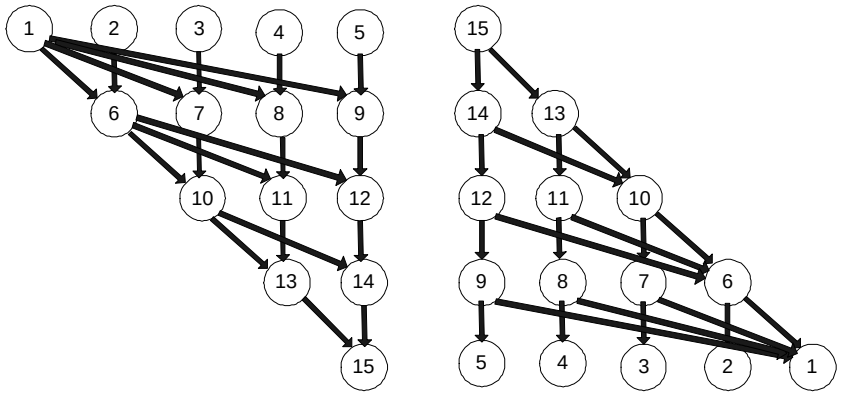

Fig. 1. Task dependency graph for problems G1 (left) and G2 (right).

```
(3,0)(1,3)(2,2)(1,2)(3,0)(4,3)(5,2)(0,0)(2,1)
```

Fig. 2. An example individual.

4.1 Variable length problem representation

Each individual in the population consists of a vector of cells or genes. A *cell* is defined as a task-processor pair, (t, p), which indicates that a task t is assigned to processor p. The number of cells in an individual may be fixed or may vary during evolution. The order of the cells of an individual determines the order in which tasks are assigned to processors: cells are read from left to right and tasks are assigned to corresponding processors as long as the same task is not assigned to the same processor twice. If the same task-processor pair appears multiple times in an individual, only the first cell contributes to the fitness evaluation. Any additional identical cells are ignored by the fitness function but still subject to action by genetic operations. The same task may be assigned multiple processors. In variable length evolution, the number of cells in an individual is limited to ten times the number of tasks in the problem. Figure 2 shows an example individual. Figure 3 shows the corresponding task assignment.

4.2 Modified genetic operators

Crossover is performed on the cell level and crossover points are restricted to falling only in between cells. In variable length runs, we use random one-point crossover which randomly selects one crossover point from each parent and exchanges the segments to the left of crossover points to form two offsprings. As a result, the length of offsprings may be different from their parents. In fixed length runs, we use simple one point crossover.

Processor 0	Task 3	Task 0	
Processor 1	Task 2		
Processor 2	Task 2	Task 1	Task 5
Processor 3	Task 1	Task 4	

Fig. 3. Assignment of tasks from individual in figure 2.

Each cell has equal probability to take part in mutation. If a cell is selected to be mutated, then either the task number or the processor number of that cell will be randomly changed.

4.3 Fitness function

The fitness function is a weighted sum of two components, the $task\_fitness$ and the $processor\_fitness$:

$$fitness = (1 - b) * task\_fitness + b * processor\_fitness.$$

The value of b ranges from 0.0 to 1.0.

The *task_fitness* evaluates whether tasks have been scheduled in valid orders and whether all tasks have been included in a solution. Calculation of $task\_fitness$ consists of three steps.

1. We use an incremental function to check the relative order of the task assignments. Suppose that the problem involves P processors and T tasks. Our GA evolution occurs in eras, $era = 0, 1, 2, \ldots, T$. Initially, era is set to zero. For all tasks assigned to the same processor, we check the sequence of every pair of adjacent tasks. A $raw\_fitness$ is calculated with the following equation:

$$raw\_fitness = \frac{number\ of\ valid\ task\ groups\ in\ an\ assignment}{total\ number\ of\ task\ groups\ in\ an\ assignment.} \quad (1)$$

   The era counter increases when the average population fitness exceeds a user defined threshold value and a fixed percentage of the population consist of valid individuals. A valid individual is one which encodes a valid assignment containing at least one copy of every task. Each time the era is increased, we increase the number of tasks in the sequences checked to obtain Equation 1. Thus, the length of the sequences checked equals $era + 2$.
   The size of the search space makes random generation of valid solutions unlikely. The goal of the era component is to provide a way for the GA to reward for partial solutions of increasing difficulty as a run progresses, eventually focusing only on full solutions.
2. We check the number of distinct tasks appearing in a solution. The $task\_ratio$ is calculated with the following equation:

$$task\_ratio = \frac{number\ of\ distinct\ tasks\ specified\ on\ an\ individual}{total\ number\ of\ tasks\ in\ the\ problem.} \quad (2)$$

Parameter	Value
Population size	200
Number of generations	1500
Crossover type	random one-point
Crossover rate	1.0
Mutation rate	0.005
Selection scheme	Tournament (2)
b	0.2
Fitness threshold	0.75

Table 1. Parameter settings used in our experiments.

3. Finally, $task\_fitness = raw\_fitness \times task\_ration$.

The *processor_fitness* evaluates the execution time of a valid task schedule, favoring schedules that minimize execution time.

$$processor\_fitness = \frac{P * serial\_len - t}{P * serial\_len}.$$

where P is the number of processors in the problem, t is the execution time of a solution, and *serial_len* is the execution time of all the tasks if they are assigned serially to a single processor.

Additional details regarding the fitness function are available in [3].

4.4 System parameter settings

Table 1 gives the default parameter settings used in our experiments. Unless otherwise specified, these values are used for all experiments. In a variable length GA, the initial population of individuals are initialized to length fifteen (the total number of tasks in the problem), and the maximum allowed length is 150. In the fixed length GA, we use individuals of length 150.

5 Experimental results

We compare the behavior of fixed and variable length GAs in a nonstationary environment. In these experiments, the target solution oscillates between two different problems, G1 and G2, shown in figure 1 every 100 generations. To emphasize the differences between the two target solutions and increase the difficulty of finding and maintaining solutions for both targets, we designed G2 to have the completely opposite set of task dependencies as G1. The results presented are from individual runs that are representative of overall GA behavior.

We compare four experimental scenarios:

1. A fixed length GA in which *era* is not reset.
2. A fixed length GA in which *era* is reset after each target change.

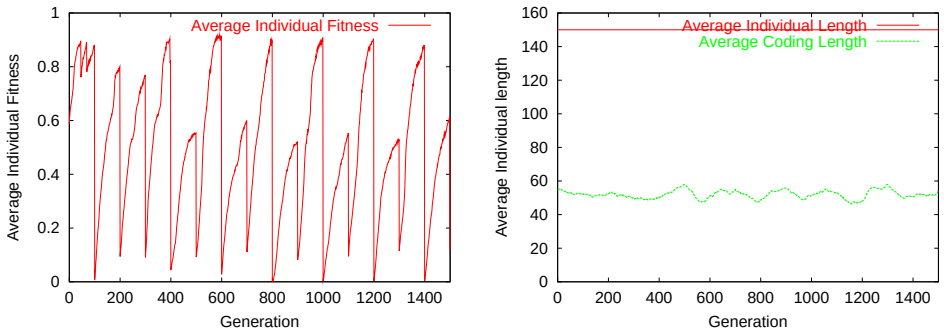

Fig. 4. Typical plots of average population fitness (left) and average coding length (right) for a fixed length (150) GA with no resetting *era* in a nonstationary environment.

3. A variable length GA in which *era* is not reset.
4. A variable length GA in which *era* is reset after each target change.

In the runs where *era* is not reset, the *era* counter increases uninterruptedly throughout a run. In the runs where *era* is reset, the value of *era* is reset to zero after each target change; *era* increases normally in between two consecutive target changes.

5.1 Fixed versus variable length

Figure 4 shows the typical variation in average population fitness and average coding length during a fixed length GA run where *era* is not reset. A sharp drop in fitness occurs after each target change indicating that the GA has difficulty retaining both solutions in its population. The alternating high and low peaks suggest that the GA population has converged primarily on one of the two target solutions. When the converged solution is the target solution, the average population fitness reaches above 0.8. When the non-converged solution is the target solution, the average population fitness is unable to exceed 0.6.

Figure 5 shows the typical variation in average population fitness and average coding length during a fixed length GA run where *era* is reset after each target change. The result is very similar to the run shown in Figure 4. We see sharp drops in fitness after every target change. The high and low peaks in fitness indicate that GA is evolving solutions toward only one target. The primary difference is the existence of smaller oscillations in fitness within the larger peaks due to increases in the *era* counter.

Figure 6 shows typical plots of the average population fitness and average population length of a variable length GA where *era* is not reset. We again see sharp drops in fitness after each target change indicating difficulty in retaining multiple solutions in the population. These data exhibit two notable differences from the corresponding fixed length results in Figure 4. First, there is little evidence of alternating peaks; the GA appears to be able to find equally strong

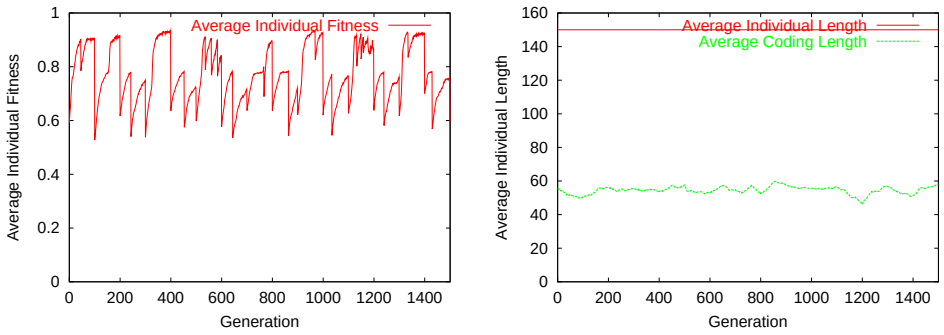

Fig. 5. Typical plots of average population fitness (left) and average coding length (right) for a fixed length (150) GA with resetting *era* in a nonstationary environment.

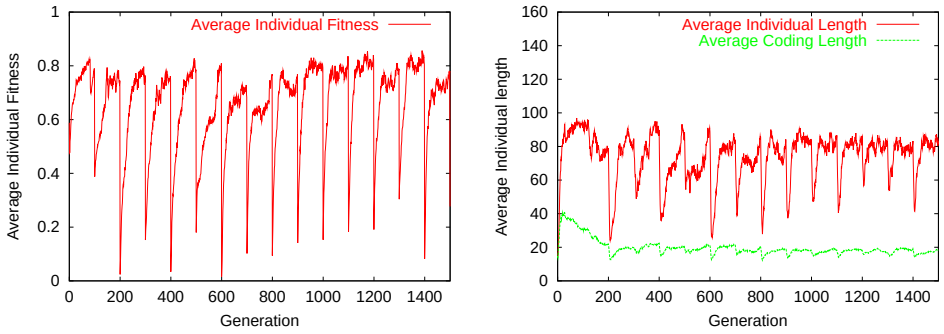

Fig. 6. Typical plots of average population fitness (left) and average length (right) for a variable length GA with no resetting *era* in a nonstationary environment.

solutions for both targets. Average fitness values are slightly lower, but comparable, to those achieved by the fixed length GA. Second, the rise in fitness following each drop is much faster and sharper. These differences suggest that a variable length GA is better able to adapt to changes in the target solutions.

The variation in average population length shows interesting and unexpected behavior. Instead of increasing after each target change (as one would predict based on previous studies [1, 2]), the average population length drops sharply after each target change. The average population length immediately increases after each drop, and reaches almost 90 for both target solutions. The average coding length shows similar behavior to a smaller degree, stabilizing at about 20. With respect to coding length, the variable length GA appears to be able to evolve more compact solutions than the fixed length GA, albeit with slightly lower fitness.

Figure 7 shows typical plots of the average population fitness and average population length of a variable length GA where *era* is reset after each target change. We observe much smaller drops in individual fitness after target changes

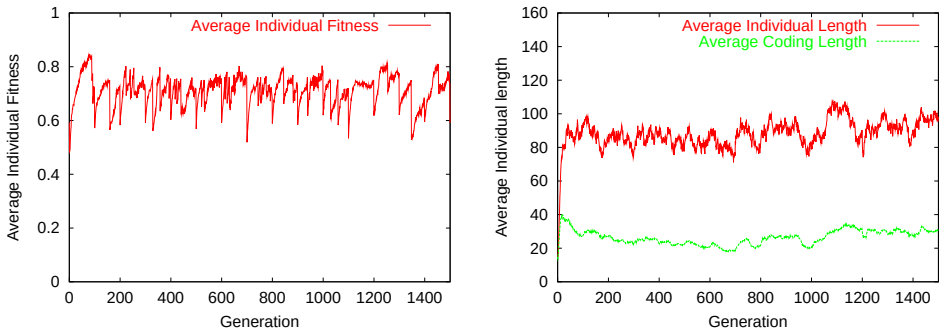

Fig. 7. Typical plots of average population fitness (left) and average length (right) for a variable length GA with resetting *era* in a nonstationary environment.

Test Cases	# generations to 1st valid solution (average/standard deviation)
Variable length GA (*era* not reset)	19.69/ 22.30
Variable length GA (*era* not reset)	33.94 / 34.05
Fixed length GA (*era* reset)	50.33 / 38.54
Fixed length GA (*era* reset)	52.29 / 44.66

Table 2. Average and standard deviation on the number of generations to find the first valid solution after every target change.

and variations in individual length that are less clearly correlated with target changes. Average fitness remains comparable to the fitness achieved in the other three experiments.

We further evaluate the adaptability of our GAs to changing environments by examining the average number of generations that a GA requires to find the first valid solution after each target change. The results, given in table 2, are collected from twenty runs in each test case. Results show that variable length GAs, as expected, are more adaptable than fixed length GAs. Surprisingly, not resetting *era* actually results in much quicker adaptation for the variable length GA and somewhat quicker adaptation for the fixed length GA. This result was unexpected as we expected the *era* counter to guide the GA in finding a solution by rewarding for partial solutions.

5.2 Variable length dynamics

Our original hypothesis was that average length would increase following each target change to provide the GA search process with more resources in the form of longer individuals. Our variable length GA runs with no resetting *era*, however, exhibit completely opposite behavior. Closer examination reveals that the GA is, in fact, selecting for *better* resources instead of *more* resources.

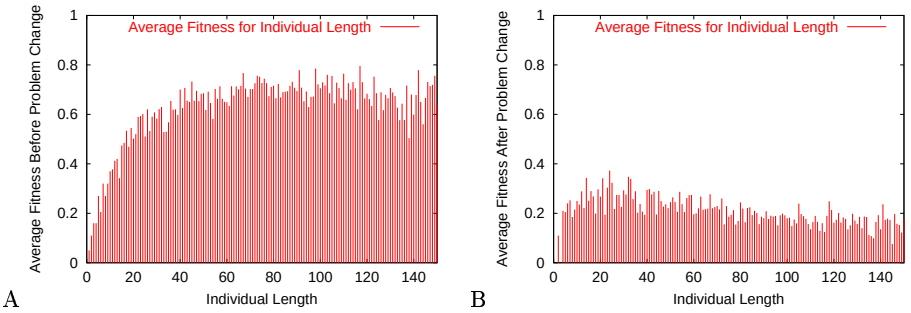

Fig. 8. Distribution of average fitness of individuals of equal length before (A) and after (B) a target change.

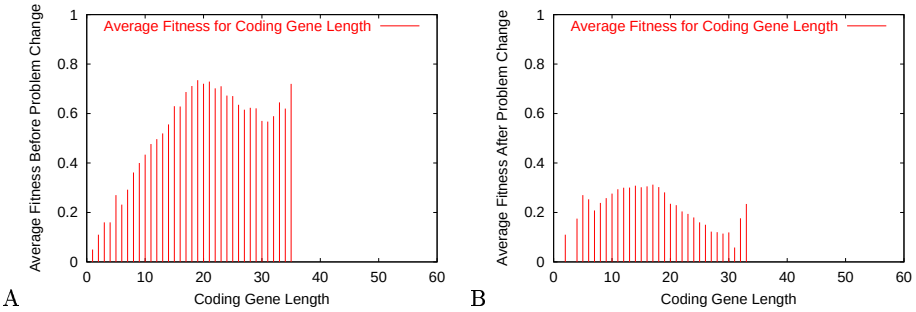

Fig. 9. Distribution of average fitness of individuals of equal coding length before (A) and after (B) a target change.

We examine the distribution of fitness with respect to length in the generations immediately preceeding and following a target change. Figure 8 shows this data for a sample run. Within a GA run, we take all individuals that are in populations immediately preceeding a target change (in our case, generations 99, 199, 299, etc.), group those individuals by length, and plot the average fitness of each group. Similarly, we plot the fitness distribution of all individuals that are in generations immediately following a target change (generations 100, 200, 300, etc.). Immediately before a target change (generation "X99"), longer individuals have higher average fitness than shorter individuals. Immediately after a target change (generation "X00"), shorter individuals appear to be more fit than longer individuals. This same dynamic is seen to a stronger degree in similar plots of the coding length of a population. Figure 9 show the distribution of fitness with respect to coding length.

Examination of individual members of the population provides an explanation. Both target problems used here consist of fifteen tasks. Thus, individuals must have coding lengths of at least fifteen to encode a complete solution. As expected, the data in figure 9.A indicates that the average fitness peaks at cod-

Proc 1	5	8
Proc 2	3	5
Proc 3	7	11
Proc 4	4	

Fig. 10. Small solution from generation 299 of sample run.

Proc 1	5	14	13	15					
Proc 2	1	3	2	6	4	5			
Proc 3	7	6	9	8	12	11	10	13	
Proc 4	4	9	12						

Fig. 11. Long solution from generation 299 of sample run.

ing lengths that are just slightly longer than fifteen and levels off with increasing length. For lengths shorter than fifteen, there is a steady linear decrease in fitness. Immediately before a target change, the GA has had time (in the runs here, 99 generations) to evolve towards the current target. Most individuals in the population will have some characteristics of the current target. Longer individuals are more likely to contain complete solutions, and therefore, are also more likely to be specialized towards the current target.

Figure 9.B shows the fitness distribution in the generations immediately following a target change. Individuals with coding lengths less than fifteen tend to have higher relative fitness while individuals with longer coding lengths exihibit steadily decreasing fitness values. Immediately after a target change, much of the population still consists of individuals that have been evolved towards the previous target. Individuals with long coding lengths are more likely to be specialized to the previous target and, thus, more likely to have low fitness with respect to a new target. Individuals with short coding lengths, on the other hand, are more likely to contain less specific building blocks that may be applicable to more target solutions.

Examination of specific individuals from a population supports the above explanation. Figures 10 and 11 show a small and a large solution, respectively, from generation 299 of a sample run. The target solution for generations 200 to 299 is problem G1. Accordingly, all task sequences in both solutions are valid with respect to G1. With respect to problem G2, all task sequences in the smaller solution are valid, but only the following subset of task sequences (18 out of 52 total) from the longer solution are valid:

[14-13] [1-3] [3-2] [1-3-2] [6-4] [4-5] [6-4-5] [7-6] [6-9] [9-8] [7-6-9] [6-9-8] [7-6-9-8]
[8-12] [12-11] [11-10] [12-11-10] [4-9]

The longer solution is a complete solution to problem G1. It is very specialized for G1, and consequently, noticeably less fit for G2. The shorter solution does not

specify a complete solution for either G1 or G2, but is equally fit for both. With respect to the percent of valid sequences, the shorter solution actually scores higher than the longer solution and consequently appears more fit immediately after a target change.

Thus, the sharp drops in average population length seen in figure 6 appear to be due to selection for more general building blocks following a target change. In this particular problem encoding, shorter individuals tend to consist of more general building blocks, longer individuals tend to consist of problem specific building blocks. By selecting for shorter individuals immediately after a target change, the GA increases its population resources by increasing the number of more general building blocks.

The above study also explains why there are no evident drops in individual length due to a problem change in Figure 7. In GAs where *era* is reset, the fitness function is again checking the order of every two adjacent tasks after each target change. Long individuals, though more specialized to the previous target, are still likely to contain short task sequences that are valid. As a result, long individuals drop less in fitness than they do in GAs where *era* is not reset and remain competitive with shorter, more general individuals. Surprisingly, resetting *era* is not beneficial in this problem as it actually retards a GA's adaptation to a new target.

6 Conclusions

In this paper, we investigate the behavior of a variable length GA in a nonstationary problem environment. We examine how variations in length can help a GA adapt to new environments.

We perform our experiments on a task scheduling problem under an oscillating environment. Experiment results indicate that a variable length GA has better and quicker adaptation to a new environment than a fixed length GA.

An interesting and somewhat surprising result showed the variable length GA undergoing sharp drops in length after each target change. This behavior is the opposite of what was expected based on previous work. Closer analysis reveals that the fitness function favors short individuals after a target change because short individuals contain more general building blocks. Long individuals, on the other hand, are more likely to contain very specific solutions adapted to the previous target. Our GA successfully exploits the flexibility of its variable length representation to better recognize and retain good building blocks.

Our study indicates that variable length representation provides a flexibile way for GA to reorgainze building blocks after problem changes. A good fitness function, where building blocks are properly defined, is able to reinforce this flexibility and improve a GA's adapation to changing environments.

References

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

2. Langdon, W.B., Poli, R.: Fitness causes bloat. Soft Computing in Engineering Design and Manufacturing (1997) 13–22
3. Authors: Authors' previous work (Year)
4. Smith, S.F.: A learning system based on genetic adaptive algorithms. In: PhD thesis, Dept. Computer Science, University of Pittsburgh. (1980)
5. Goldberg, D.E., Korb, B., Deb, K.: Messy genetic algorithms: Motivation, analysis, and first results. Complex Systems **3** (1989) 493–530
6. Koza, J.R.: Genetic programming. MIT Press (1992)
7. Harvey, I.: Species adaptation genetic algorithms: A basis for a continuing saga. In: Proceedings of the First European Conference on Artificial Life. (1992) 346–354
8. Soule, T., Foster, J.A., Dickinson, J.: Code growth in genetic programming. In: Proc. Genetic Programming(GP). (1996) 400–405
9. Soule, T., Foster, J.A.: Effects of code growth and parsimony pressure on populations in genetic programming. Evolutionary Computation **6** (1998) 293–309
10. Grefenstette, J.J.: Genetic algorihtms for changing environments. Parallel Problem Solving from Nature (PPSN) **2** (1992) 137–144
11. Cobb, H.G.: An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report AIC-90-001, Naval Research Laboratory (1990)
12. Grefenstette, J.J.: Evolvability in dynamic fitness lanscapes: a genetic algorithm approach. In: Congress on Evolutionary Computation. (1999) 2031–2038
13. Mori, N., Kita, H., Nishikawa, Y.: Adaptation to a changing environment by means of the thermodynamical genetic algorithm. In: PPSN. (1996) 513–522
14. Kita, H., Yabumoto, Y., Mori, N., Nishikawa, Y.: Multi-objective optimization by means of the thermodynamical genetic algorithm. In: PPSN IV. (1996) 504–512
15. Goldberg, D.E., Smith, R.E.: Nonstationary function optimization using genetic algorithms with dominance and diploidy. In: Intl. Conf. on Genetic Algorithms (ICGA). (1987) 59–68
16. Ng, K.P., Wang, K.C.: A new diploid scheme and dominance change mechanism for non-stationary function optimization. In: Proc. 6th ICGA. (1995) 159–166
17. Smith, R.E.: Diploidy genetic algorithms for search in time varying environments. In: Annual Southeast Regional Conference of the ACM. (1987) 175–179
18. Branke, J.: Memory enhanced evolutionary algorithms for changing optimization problems. In: CEC. (1999) 1875–1882
19. De Jong, K.A.: An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan (1975)
20. Deb, K., Goldberg, D.E.: An investigation of niche and species formation in genetic function optimization. In: Proc. ICGA. (1989) 42–50
21. Liles, W., De Jong, K.: The usefulness of tag bits in changing envrionments. In: CEC. (1999) 2054–2060
22. El-Rewini, H., Lewis, T.G., Ali, H.H.: Task scheduling in parallel and distributed systems. Prentice Hall (1994)