A Memory and Search Hybrid Genetic Algorithm for non-Stationary Environments with Repetitive Natures

David B. Bracewell and Fuji REN

Department of Information Science and Intelligent Systems
University of Tokushima
Minami-Josanjima-Cho Tokushima-shi 770, JAPAN
email: {davidb, ren}@is.tokushima-u.ac.jp

Abstract. We look at combining a search-based and memory-based approach creating a hybrid GA to solve problems with non-stationary environments. In particular, the memory search hybrid GA (MSHGA) we present is well suited to deal with non-stationary environments that are repetitive in nature, i.e. different problem landscapes are repeatedly seen. The MSHGA is capable of recalling candidate solutions for previously seen problem landscapes. This ability coupled with the search based technique of random immigrants causes the MSHGA to outperform the SGA and random immigrants GA in our experiments.

Keywords
Soft Computing, Evolutionary Computation, Genetic Algorithms, Non-Stationary Environments

1 Introduction

The genetic algorithm has been widely used for many problems that have stationary environments. With its population of candidate solutions it should also provide a good mechanism for solving problems having non-stationary environments. Recently, there has been much work on adapting the genetic algorithm to deal with non-stationary environments. Out of this research two main approaches have emerged, the search-based approach and the memory-based approach [10].

Diversity in a population is what secures its survival in changing environments. The strong selection pressure of the simple genetic algorithm (SGA) quickly eliminates diversity in the population. The search-based approach attempts to keep some level of diversity in the population. This added diversity helps the GA to cope with the changes in environment. There have been many studies on ways of keeping the population diverse. Cobb introduced the idea of triggered hypermutation that is activated when a degradation of fitness is monitored for the best candidates in the population [2]. Grefenstette introduced the idea of random immigrants that uses a partial hypermutation to replace a percentage of the population determined by the replacement rate [8]. Ghosh *et al.*

© *A. Gelbukh, C. Yáñez Márquez, O. Camacho Nieto (Eds.)*
Advances in Artificial Intelligence and Computer Science
Research on Computing Science 14, 2005, pp. 35-46

use aging of individuals that limits the dominance one individual can have over the population [6]. Cobb and Grefenstette did a comparison between the SGA, random immigrant GA, and GA with hypermutation. They found that diversity "represents a natural source of power in adapting to changing environments" [3].

The memory-based approach extends the memory of the GA to learn previous adaptations. Perhaps the most well known is diploidy and dominance maps, introduced by Goldberg and Smith [7]. They showed that this technique worked in certain non-stationary environments, notably ones having only two optima. However, this technique does not seem to scale up to the more general case. Kita and Sano proposed a memory-based fitness approach, which stores sample fitness values in memory for estimation of fitness values of points of interest [9]. While it looks promising only an outline of how to use it for non-stationary environments was given as the main focus was on noisy fitness functions.

In this paper we look at combining both the search-based and the memory-based technique to create a hybrid GA to deal with non-stationary environments. The Memory Search Hybrid GA (MSHGA) we will present is particularly well suited for environments that are repetitive in nature. This means the different problem landscapes brought about by the changes in environment are repeatedly seen. One can very well imagine a situation in which the variables for a problem are continuously changing and that the same variable values are seen over and over again. An example would be optimizing the temperature of a room where the variables would be outside weather conditions. This is the type of problem that the MSHGA is designed for and excels at. Typical techniques for non-stationary environments, especially search-based techniques, do not show this characteristic.

The ability to recall candidate solutions to previous variations of a problem also leads us in the direction of creating a GA that is capable of dealing with any possible instance of a problem. Imagine a GA that is always processing and different instances of a problem are continuously fed to it. Because of the memory if any instance has been seen before we can either not process and give the answer we have or continue to process from that point hoping to refine the answer even further. In a sense the MSHGA becomes an expert system that is capable of handling any instance of a certain problem.

The paper will proceed as follows, in section 2 the MSHGA will be explained. In section 3, we will present the experiments we performed and their results. Finally, in section 4 we will conclude the paper with some final thoughts and considerations for future work.

2 The Memory Search Hybrid GA

The MSHGA is made up of a generational GA [5] with 3 added parts, as seen in figure 1. Elitism allows the MSHGA to always refine solutions. It keeps the most fit individual always in the population. The search technique keeps diversity in the population allowing for a better chance of finding a good solution

to an unseen problem landscape. The candidate population keeps track of the environments and their best answer seen so far.

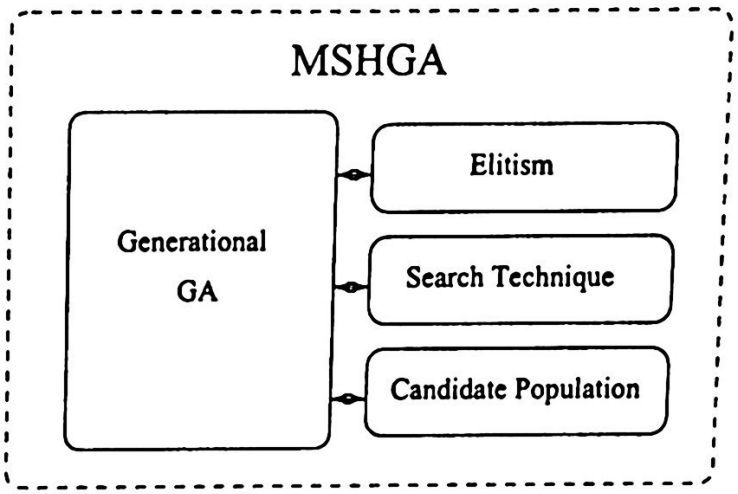

Fig. 1. MSHGA Diagram

The search-based techniques do well in keeping the population diverse. The added diversity allows the GA to keep a variety of building blocks available. However, a search-based technique has no way of remembering the candidate solutions for the different optima caused by the non-stationary environment. In order to handle this we need to add a memory that is specifically for the different optima. The MSHGA has an extra storage area, which is called the candidate population. The size of the candidate population is dependent on the number of changes that will take place in the environment. Each problem landscape, brought about by a change in environment, will have an individual in the candidate population. The individual can be simply a copy of the best fit candidate solution that has been seen for that problem landscape so far and its fitness. The size of the of the candidate population can be dynamic or fixed. If memory is at a premium then a fixed candidate population can be used. Using a fixed candidate population does require some mechanism for retiring candidate solutions from the population to make room for new ones. Since typically problems call for population sizes in the hundreds the memory needed for the candidate population will generally be much smaller. In our current implementation we use a dynamically sized candidate population.

The search-based technique and the underlying generational GA can be chosen to suite the needs of a particular problem. The MSHGA we designed for our experiments used random immigrants as the search technique and used a SGA base. For certain problems a non-fixed length encoding, dynamic population size, etc. may be useful.

As figure 2 shows, the MSHGA works like any other generational GA. For more information on the generational GA please see [5]. The main difference is that it needs some way of detecting a change in environment. This can be

achieved in a variety of ways. One such way is to keep track of the parameters for each problem landscape and link those parameters to an individual in the candidate population. This technique has the added benefit that at the end of the run each problem landscape can be shown with its corresponding answer and set of parameters. Of course, this technique does have the downside of requiring more memory. We chose to use this approach for our experimentations. This method checks current conditions to determine if the environment has changed, an example would be checking the temperature and determining if the environment has changed based on the difference between the current and previous temperatures. A second approach, which works well with permutation problems, is to modify the fitness function so that no individual can have the same fitness in different problem landscapes. In most cases, using this method will require encoding parameter information into the fitness of the individual. Both methods were tested for validity and the performance was the same.

Before the start of the MSHGA the population is initialized. As with the SGA this can be done randomly or using some heuristics about the problem. At the beginning of each generation a check is made to determine if the environment has changed. If it has then we first save the current best individual into the candidate population associating it with the environment is was in (overwriting any older saved individuals). We then, look into the candidate population to see if there is a candidate solution for the new problem landscape. If there is, we retrieve it and replace the least fit individual in the population with it. The MSHGA then goes on normally, performing selection, recombination, and mutation. Any selection, crossover, or mutation function can be used. At the end of each generation random immigrants are introduced into the population. Finally, the best fit individual from the previous generation is copied into the new child population replacing the least fit child.

```
Initialize Population
while( not done ) do
    If Enviroment has changed Then
        Store best individual
        Retrieve the new environment's best individual
    End
    Selection
    Crossover
    Mutation
    Random Immigrants
    If Enviroment has NOT changed Then
        Elitism
    End
End
```

Fig. 2. MSHGA Pseudocode

3 Experimentation

For experimentation we looked at two problems: x-max and the non-stationary knapsack problem. In each problem we compared the performance of the MSHGA to that of the SGA and the RIGA (random immigrants GA). Two tests were ran for each problem. In the first test, the parameters are changed on a schedule of every 50 generations. In the second test, we looked at having a random change in parameters. In this test each generation had a 15% chance that the parameters would be changed. In both experiments the RIGA and MSHGA used a replacement rate of 10%. The SGA and RIGA were augmented with elitism to keep them comparable to the MSHGA. In addition, each GA for each problem was run for 100 runs and the results shown are the average best fitness per generation over those 100 runs.

3.1 The X-Max Problem

The one-max problem is often introduced when learning about genetic algorithms, because it is easy to understand and to implement. The goal is to maximize the number of ones in a bit string. The x-max problem is the non-stationary version of the one-max problem. In the x-max problem the goal is to maximize the number of x in the bit string, where the value of x changes over time. In the version of x-max that we chose, the x value can range from 0 to 9. We chose a bit string size of 100. Thus, the maximum fitness value is 100 and is reached when the bit string has all 100 bits set to x.

For mutation we randomly chose a number from 0 to 9. The mutation rate was 0.005. For crossover we used uniform crossover with a crossover rate of 0.8. We used tournament selection with a tournament size of 2 and an 80% chance the individual with the higher fitness would be chosen. The population size was 100 and one run was for 4000 generations. This allowed for each of the x values to be seen 8 times on the scheduled change test. The MSHGA parameters were chosen by experimentation, where we tried to optimize the SGA for the 1-max problem.

In figures 3, 4, and 5 we can see the results for the SGA, RIGA, and MSHGA when there is a scheduled change in the value of x. We can see that the SGA and RIGA have to keep rebuilding their solutions each time the value of x changes. This extra effort causes the SGA and RIGA to spend so much of its time trying to rebuild good solutions that is never able to improve on them. This wasted effort is avoided by the MSHGA due to its ability to use its memory to retrieve the previous best candidate solution for an x value it has seen before. If we were to expand the number of generations we would expect to see that the SGA and RIGA would continue to hover around 60 and 80 respectively while the MSHGA would continue to improve upon its solutions.

In figure 6, we can see the results for the SGA, RIGA, and MSHGA when there is a random change in the value of x. The graph depicts how well each type of GA is able to cope with a rapidly changing environment. As with the scheduled change we can see that the SGA and RIGA are both stagnant in their

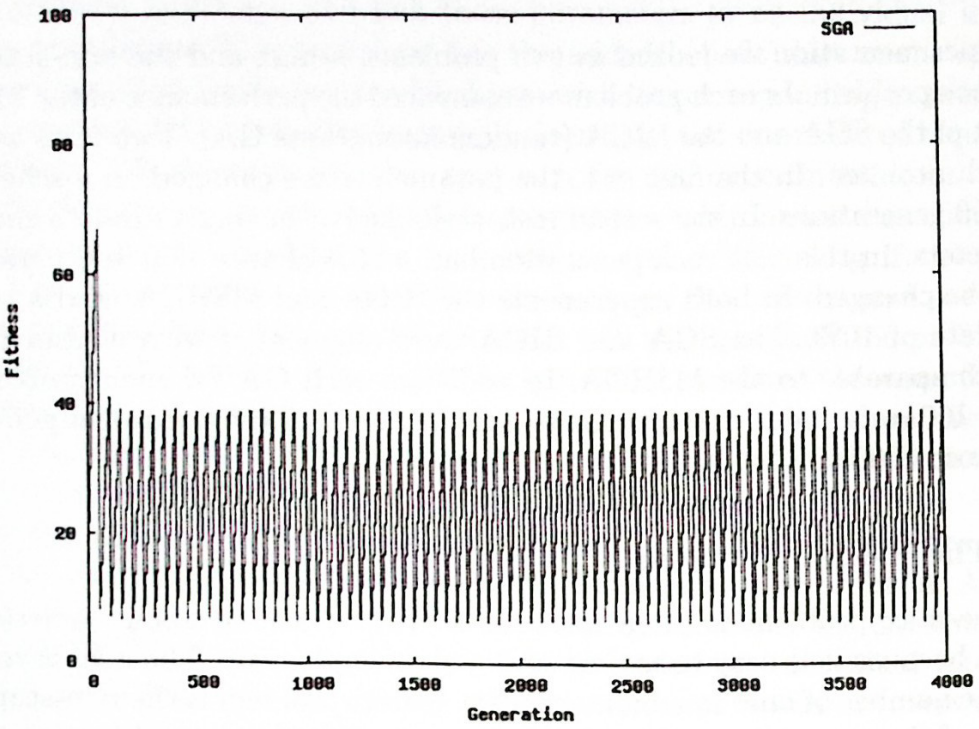

Fig. 3. Scheduled Change of X Value (SGA)

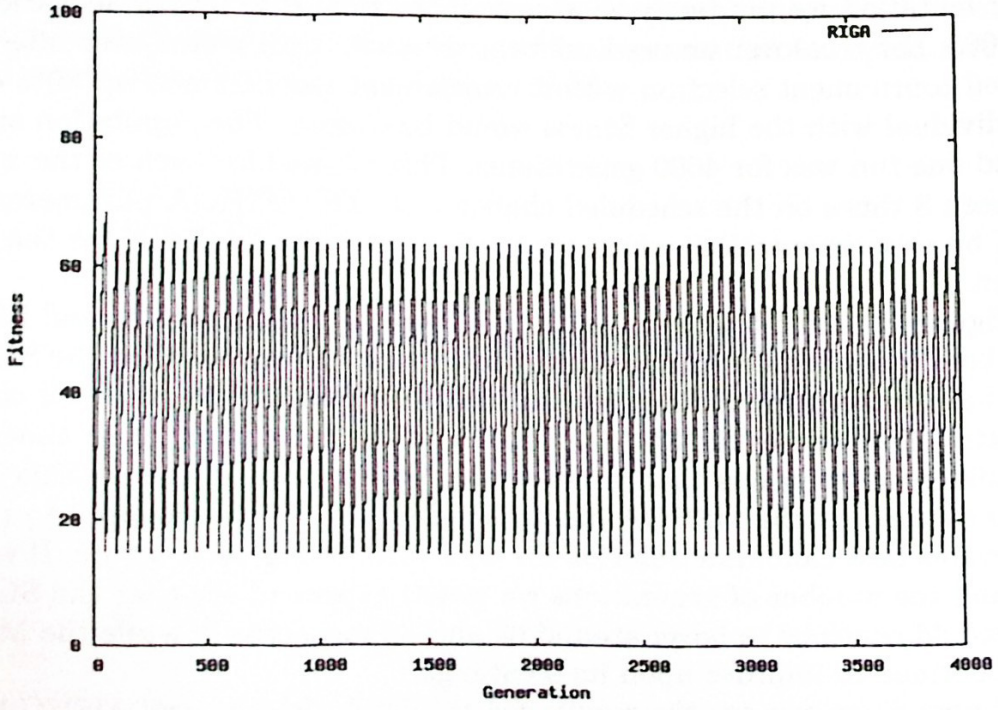

Fig. 4. Scheduled Change of X Value (RIGA)

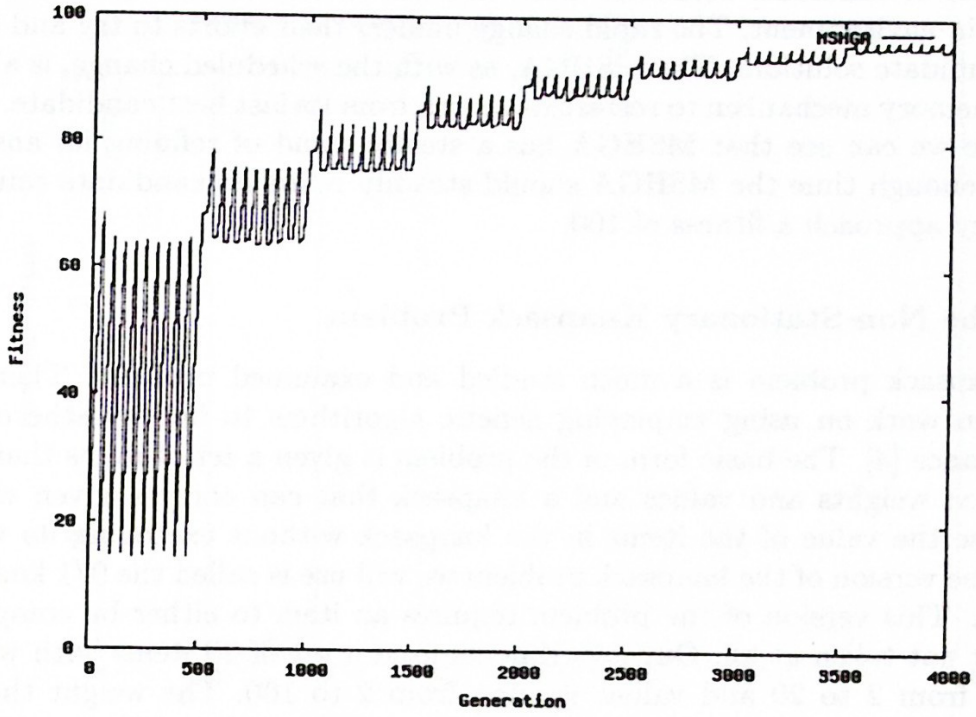

Fig. 5. Scheduled Change of X Value (MSHGA)

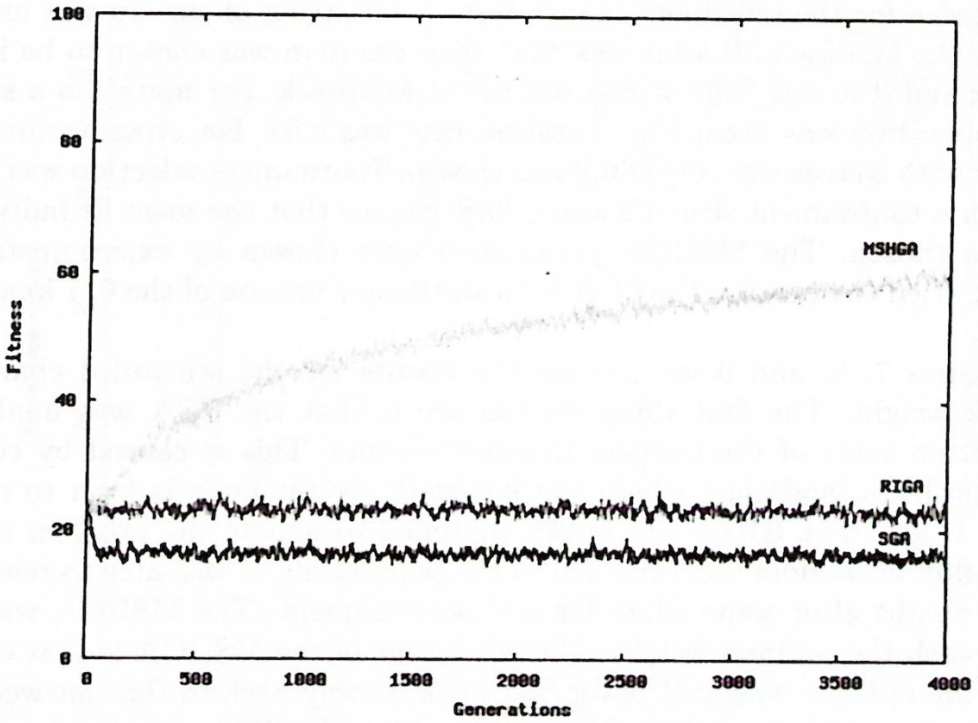

Fig. 6. Random Change of X Value

refinement of candidate solutions. They are not able to handle the more rapid changes in environment. The rapid change hinders their efforts to try and refine good candidate solutions. The MSHGA, as with the scheduled change, is able to use its memory mechanism to restart its search from its last best candidate. Also, as before we can see that MSHGA has a steady trend of refining its answers. If given enough time the MSHGA should steadily refine its candidate solutions until they approach a fitness of 100.

3.2 The Non-Stationary Knapsack Problem

The knapsack problem is a much studied and examined problem. There has also been work on using employing genetic algorithms to improve the overall performance [4]. The basic form of the problem is given a set of items that have associated weights and values and a knapsack that can carry a given weight, maximize the value of the items in the knapsack without exceeding its weight limit. The version of the knapsack problem we will use is called the 0/1 knapsack problem. This version of the problem requires an item to either be completely taken or not taken at all. Our experiments used a set of 20 items with weights varying from 2 to 20 and values varying from 2 to 100. The weight that the knapsack could carry was varied over time from 20% to 90% of the total weight of the items in 10% increments. All the items were randomly generated and the order of change in the knapsack's weight limit was randomly chosen.

The population size was chosen to be 500 and each run consisted of 800 generations. 800 generations allows each of the 8 different knapsack weights to be seen twice for the scheduled change test. A bit string of size 20 was used to represent the knapsack. If a bit was "on" then the item was chosen to be in the knapsack and if it was "off" it was not in the knapsack. For mutation a simple bit flip operation was used. The mutation rate was 0.01. For crossover,uniform crossover with a crossover rate of 0.8 was chosen. Tournament selection was again used with a tournament size of 2 and a 98% chance that the most fit individual would be chosen. The MSHGA parameters were chosen by experimentation, where we tried to optimize the SGA for a stationary version of the 0/1 knapsack problem.

In figures 7, 8, and 9 we can see the results for the scheduled change of knapsack weight. The first thing we can see is that the SGA was unable to recover from some of the changes in environments. This is caused by coming from a problem landscape where the knapsack weight limit is high to one in which it is low. The RIGA was better equipped to handle the problem thanks to the influx of random individuals into the population. It was able to reach the optimal weight after some effort for each environment. The MSHGA was also able to reach the optimal weight. The advantage of the MSHGA is that once it reached the optimal weight it never had work its way back to that answer. In a situation where time is critical this would make a big difference as once we find a good enough solution we would not have to do further computation.

In figure 10, we can see the results of all three GAs when the knapsack weight is randomly changed. As with the x-max problem, we again see that the SGA

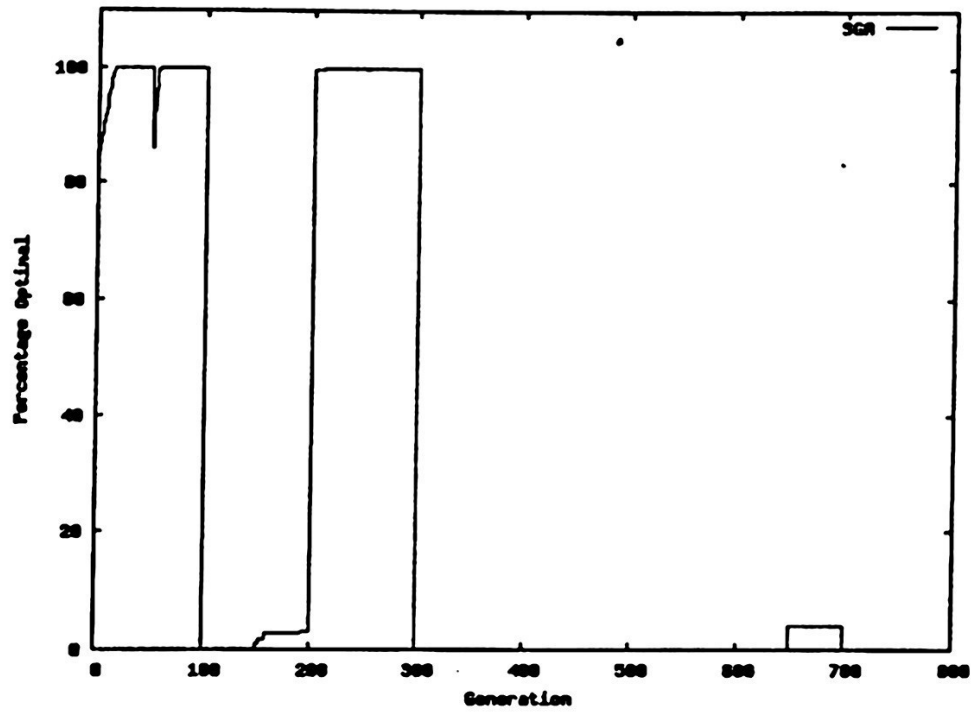

Fig. 7. Scheduled Change of Knapsack Weight (SGA)

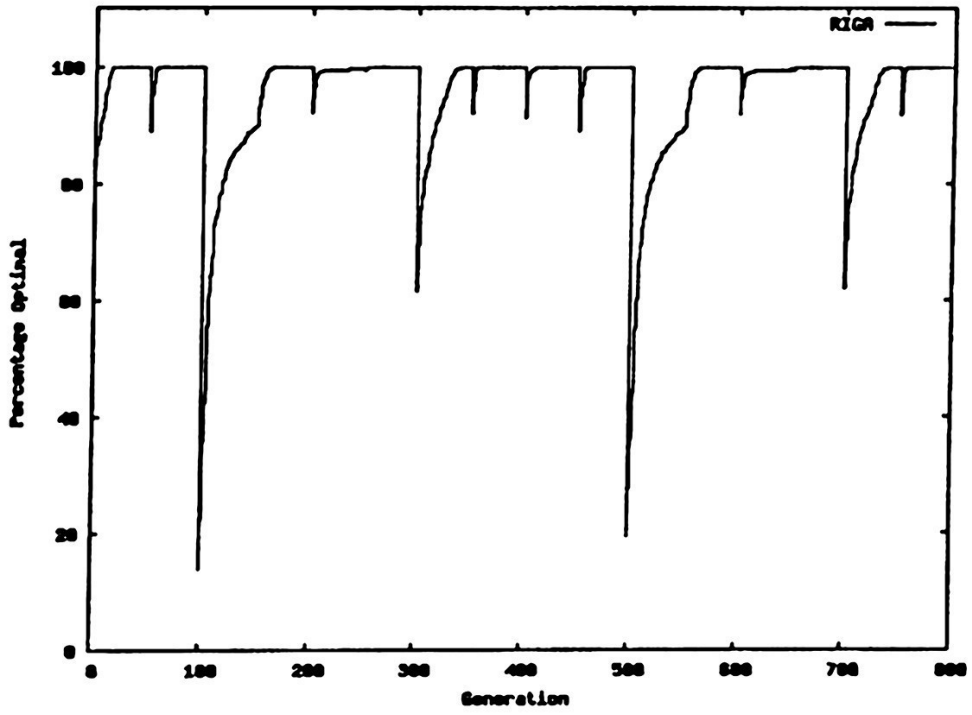

Fig. 8. Scheduled Change of Knapsack Weight (RIGA)

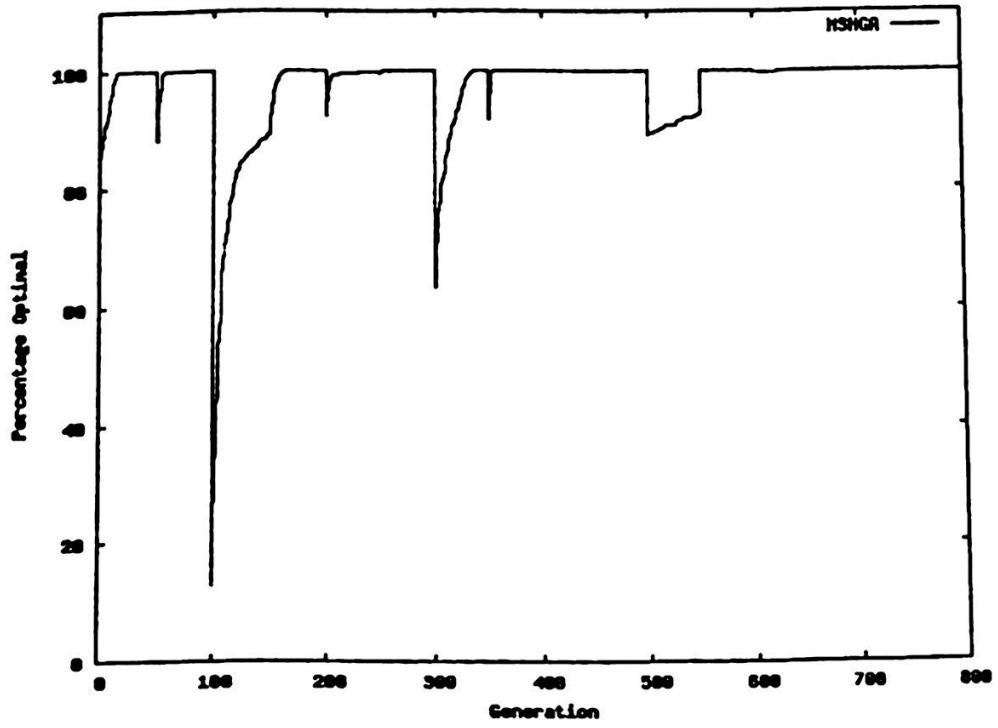

Fig. 9. Scheduled Change of Knapsack Weight (MSHGA)

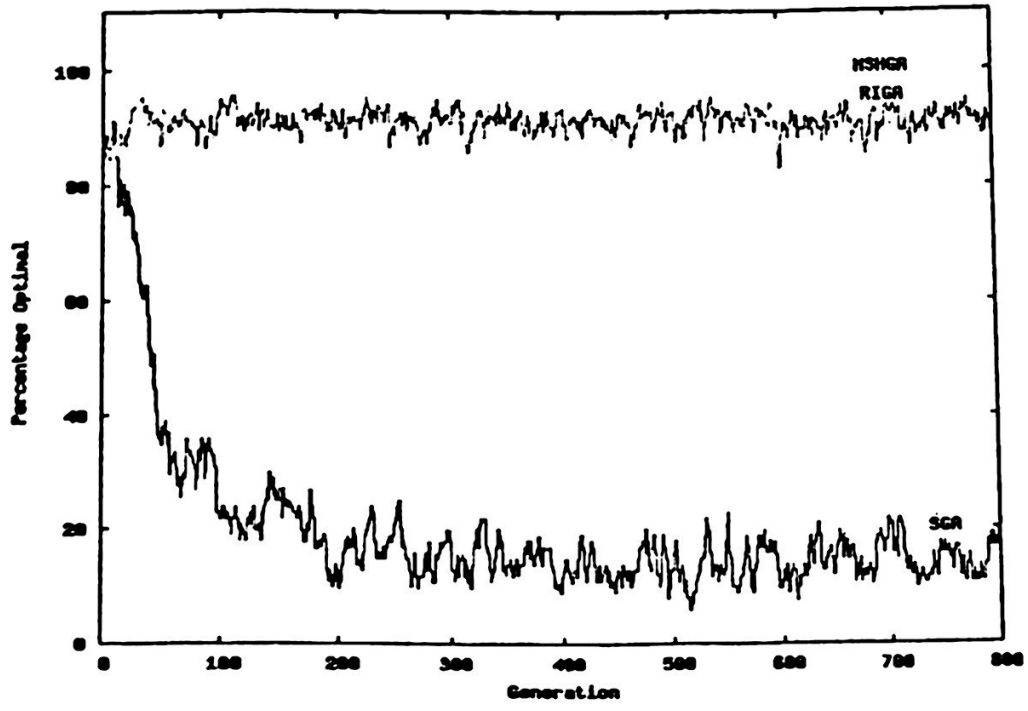

Fig. 10. Random Change of Knapsack Weight

is not capable of dealing with the rapidly changing environment. However, it performed better with the random change than it did with a scheduled change. The reason for this is that the with a rapid change in environment the SGA's selectional pressure was a bit a lower allowing a more diverse population. For the knapsack problem this added diversity helped to keep solutions that had high weights from dominating the population. The problem is that the SGA did not know which optimum to go toward and instead settled on a suboptimal answer somewhere in between. The RIGA performed well, but the extra effort of having to regain the optimal answer each time caused its performance to be lower than that of the MSHGA's. Over a longer duration the MSHGA will remain at 100% optimal while the RIGA will probably stay around 90% optimal.

4 Conclusion and Future Work

In this paper the Memory Search Hybrid Genetic Algorithm was introduced. It is designed to handle non-stationary environments that are repetitive in nature. Through experimentation it was shown that the MSHGA outperforms both the SGA and the RIGA in this task. The MSHGA is made up of a generational GA with elitism, random immigrants, and an extra set of memory called the candidate population.

The random immigrants part of the MSHGA is what keeps the population diverse and drives the search when encountering new problem landscapes. The candidate population is what ensures that the MSHGA does not have to waste rebuilding candidate solutions to previously seen problem landscapes. This ability to recall is what allows the MSHGA to outperform the RIGA over time. Much research has been done for non-stationary environments, but as far as we know, little to none has been done for non-stationary environments with repetitive natures. It seems that in everyday tasks often the same problems are encountered time and time again. For this purpose the MSHGA was created. Over time the MSHGA will be able to give instantaneous answers to problems that it has previously seen. For typical genetic algorithms for designed for non-stationary environments this would be a daunting task.

The candidate population and the ability to recall previously seen environment changes also points us in the direction of creating an always "on" MSHGA that can be continuously fed new instances of a problem. For example, imagine a system that must make choices based on external variables. If the external variables are finite in nature than over time the MSHGA can learn good solutions for each set of variables. It then can either bypass continued exploration and return a candidate solution instantly or it can continue to work and refine on the candidate solution. In essence the MSHGA memorizes answers to questions and is able to instantly recall them in the future. This is partly what we hope to look at in the future.

Also, we hope to look at not linking exact instances with candidate solutions, but instead to link abstract settings to candidate solutions. We then may be able to expand the MSHGA to handle many new situations. For example, in path

finding instead of linking a grid of terrain to a candidate solution we could instead link general information about the environment. This may allow the MSHGA to learn interesting information about the terrain and use this information for unseen terrain grids.

One final future application is to use the MSHGA to look at problem in several different ways and then vote on a best solution overall. For example, Part-of-Speech tagging, see [1], could be done using a combination of rule-based, Hidden Markov Model, and other approaches. Each approach would simply be a new environment for the MSHGA. At the end of the run of the MSHGA, a voting mechanism would chose which candidate solution of the different techniques should be the final answer. The hope is that information from the different problem landscapes will be shared causing better solutions to be found.

There are many possibilities that the MSHGA allots us. By changing out the underlying generational GA we can tailor the MSHGA to the problem. The MSHGA works as well as the RIGA in non-stationary environments. However, the MSHGA is more capable at dealing with non-stationary environments with repetitive natures. In the future we hope to expand the experimentation of the MSHGA to show off its ability at these types of problems.

References

1. E. Brill, "A Simple Rule-Based Part-of-Speech Tagger," in *Proceedings of 3rd Applied Natural Language Processing,* 1992, 152-155.
2. H. Cobb, "An Investigation into the user of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments," U.S. Naval Laboratory Memorandum Report 6760, 1990.
3. H. Cobb and J. Grefenstette, "Genetic Algorithms for Tracking Changing Environments," in *Proceedings of 5th ICGA,* 5, 1993, 523-529.
4. C. Cotta and J.M. Troya, "A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack Problem," *Artificial Neural Nets and Genetic Algorithms 3,* 1998, 251-255.
5. L. Davis (ed.), "Handbook of Genetic Algorithms," Van Nostrand Reinhold Computer Library, NY, 1991.
6. A. Ghosh, S. Tsutsui, and H. Tanaka, "Function Optimization in Nonstationary Environment using Steady State Genetic Algorithms with Aging of Individuals," in *IEEE International Conference on Evolutionary Computation,* 1999, 666-671.
7. D.E. Goldberg and R.E. Smith, "Nonstationary function optimization using genetic algorithms with dominance and diploidy," in *International Conference on Genetic Algorithms,* 1987, 59-69.
8. J. Grefenstette, "Genetic Algorithms for Changing Environments," in R. Maenner and B. Manderick, editors, *Parallel Problem Solving from Nature 2,* North Holland, 1992, 137-144.
9. H. Kita and Y. Sano, "Genetic Algorithms for Optimization of Noisy Fitness Functions and Adaption to Changing Environments," in *Statistical Mechanical Approach to Probabilistic Information Processing,* 2003.
10. N. Mori and H. Kita, "Genetic Algorithms for Adaptation to Dynamic Environments - A Survey," in *Proceedings of SEAL 2000,* 2000, 2947-2952.