Measurements for Understanding the Behavior of the Genetic Algorithm in Dynamic Environments

A Case Study using the Shaky Ladder Hyperplane-Defined Functions

William Rand and Rick Riolo
Center for the Study of Complex Systems
University of Michigan
4485 Randall Lab
Ann Arbor, MI, USA, 48109-1120
wrand@umich.edu

ABSTRACT

We describe a set of measures to examine the behavior of the Genetic Algorithm (GA) in dynamic environments. We describe how to use both average and best measures to look at performance, satisficability, robustness, and diversity. We use these measures to examine GA behavior with a recently devised dynamic test suite, the Shaky Ladder Hyperplane-Defined Functions (sl-hdf's). This test suite can generate random problems with similar levels of difficulty and provides a platform allowing systematic controlled observations of the GA in dynamic environments. We examine the results of these measures in two different versions of the sl-hdf's, one static and one regularly-changing. We provide explanations for the observations in these two different environments, and give suggestions as to future work.

Categories and Subject Descriptors

F.2.m [**Analysis of Algorithms**]: Misc.; I.2.8 [**Artificial Intelligence**]: Search

General Terms

Algorithms, Measurement

Keywords

Genetic Algorithms, Measurement, Dynamic Environments, Hyperplane-Defined Functions, Genetic Algorithms

1. INTRODUCTION

The Genetic Algorithm (GA) has been shown to work successfully in many dynamic environments [1] [3], and while work has been done on trying to understand how the GA works in these environments, the behavior of the GA in dynamic environments is still not well understood. Part of the problem is that when examining a GA on a particular search problem often researchers are interested in the performance of the GA and not necessarily why the GA works and thus they only report performance measures. This does not always give a good indication of the overall behavior of the GA. Thus in this paper we present a suite of measures that are useful to measure beyond the standard performance metrics. By examining all of these measures together we hope to more fully understand the behavior of the GA in dynamic environments.

In a related area, Branke et al, among others, are interested in characterizing and measuring the dynamic landscape that evolutionary algorithms (EAs) are operating within [4]. While that work is related, that work does not directly address the goal we are addressing here, i.e., understanding the behavior of the EAs themselves, not the landscape they operate upon. Moreover, many researchers have developed benchmark dynamic landscapes, like the moving peaks function, that allow the comparison of different EAs to evaluate their performance [2] [13]. This work is also related but tangential to the work described here, since we do not want to compare different EAs but instead we are concerned with understanding the behavior of the simple GA.

In order to systematically examine the behavior of the GA we require a test suite of functions. The test suite presented here is similar to the dynamic bit matching functions utilized by Stanhope and Daida [17] among others. The difference between the test suite in this paper and other dynamic test functions is that the underlying representation of this test suite is schemata, which make it easier to examine how the GA is operating. By utilizing a test suite that reflects the way the GA searches, the performance of the GA can be easily observed. This test suite is a subset of John Holland's hyperplane-defined functions (hdfs) [9]. We have extended the hdfs to dynamic environments and we call this test suite, the Shaky Ladder Hyperplane-Defined Functions (sl-hdf's) [15].

In the rest of this paper, we examine the measures that we are interested in exploring, briefly describe the sl-hdf's, and finally present some results of these measures on the sl-hdf's in two different environments: a static and a regularly-changing environment. We conclude by discussing these re-

GECCO'05 June 25–29, 2005, Washington, DC, USA.

sults, providing some preliminary explanations and describing future work.

2. MEASURES

Though the performance of the GA is an important thing to measure, there may be other measures that can be useful along with performance in order to give a better description of how the system is behaving. This is especially true in dynamic environments where the behavior of the system over time can vary dramatically as the environment changes. Other measures may provide valuable explanations and allow researchers to better describe and characterize the dynamics of populations being changed by a GA and an associated (exogenous or endogenous) mechanism that assigns fitness ratings to the individuals.

In general researchers view the behavior of the GA from two different perspectives. Some are concerned with understanding extreme behaviors of the system, particularly the best that the system can do. These tend to be the measures preferred by application practitioners, who want to know what is the best result the system can possibly obtain. These measures tend to be of more interest to people who have particular problems that they are trying to solve. Other measures which may be useful characterize the population as a whole, i.e. average, standard deviations, distributions. Population distribution measures are often important to researchers trying to understand GAs as representations of evolutionary systems. They utilize the GA to model these systems and thus the behavior of the best individual in the system is not as important to them. Given that loose dichotomy, we examine all of the measures below through the context of both a best and an average measure.

Since a population evolving under the influence of a GA is, in general, a complex adaptive system, there are inherent stochastic and path-dependent processes such that each "run" of the GA, even given the same starting population, will almost always result in dynamics that differ in the details, and in some cases, the results will differ qualitatively from run to run as well. When the primary goal is to find a high performance solution for a real world problem, it often is most useful to track the best fitness individuals from each of a set of runs and then focus on the best of those best-of-run individuals. On the other hand, in other situations it is important to include measures that reflect the distribution of population histories, since that gives more information about both the complexity of the solution space and the dynamical properties of populations evolving in that problem space.

Standard measures of performance fall into a category we call *fitness-related* measures, which describe the system by examining the fitness ratings of the individuals in the population. Fitness in an evolutionary algorithm (EA) is defined to be the score that influences selection and hence determines the ability of the individual to replicate [8]. Performance, on the other hand, is a score that the individual receives on an objective function. Often fitness is performance in EAs; however, there are cases where other factors are included in the fitness score. In this paper the fitness score of an individual is defined as the performance score. Besides the standard measure which we call *performance*, we describe two additional fitness-related measures: *satisficability*, and *robustness*. Another category of measures is *composition-related* measures which attempt to describe the behavior of the system by describing the components that currently make up the system. In this paper we chose to look at only one composition-related measure, *diversity*.

We describe these four measures in detail below, but briefly, performance is the standard measure of how well the system performs the task presented it, satisficability is the ability of the system to maintain a certain level of fitness and to avoid egregious errors, robustness is a measure of how the system responds to changes in the environment, and finally diversity is a measure of how different the members of the population are. We do not think of this as a definitive list of all the possible measurements of a GA's behavior, but rather as a representative list of measures that are interesting to both those with a design or engineering perspective and those with a biological perspective.

2.1 Performance

Performance is the standard measure of how well the system solves an objective function. As mentioned, in many cases, this is the fitness function, but some GAs such as co-evolving systems, and systems with implicit fitness functions, do not have an absolute fitness function. In this paper we use the fitness function as a measure of performance. One use of performance is in evaluating the suitability of a GA-discovered solution to solving a design problem. The GA has proved to be particularly useful if there are epistatic interactions between variables. Examining the performance of the best individual in a population gives an estimate of the best solution the GA can find for a problem, given a set of GA parameters and the resulting total computational effort expended, in terms of the total number of fitness evaluations required [11]. For biological modelers it is more important to understand the performance of the whole population. The average fitness of the population provides a first-order representation of the overall fitness distribution, and thus is a good first measure. In both cases it usually makes sense when describing how the system works to aggregate the results of multiple runs by averaging the results across runs, since the average gives a good indication of how the system is expected to do on any given run.

In the experiments described below, we define *Best Performance* to be the average across n runs of the fitness of the best individual of the current generation in the population. One could also examine the best performance of any individual in any run, but we leave investigations of that measure for the future. *Average Performance*, on the other hand, is the average across n runs of the average fitness of the population. Since we know the optimal value a priori given the sl-hdf construction scheme, we express these measures as a fraction of the optimal fitness possible. This allows us to compare results where the optimal value changes.

2.2 Satisficability

Satisficability is a measure of when the system is able to achieve a predetermined criteria. The notion was first introduced by Simon who claimed that often humans do not optimize the solution to a problem but instead simply come up with a solution that satisfices some criteria they have previously defined [16]. In this paper, satisficability measures how well the system is able to maintain a certain level of fitness and not drop below a pre-set threshold. One application of this measure would be autonomous agent control. If a GA is in charge of steering a robot from one location to

another, it maybe not be necessary to get there as quickly as possible but the design specifications may require that it will get there in a reasonable amount of time.

Whether we are interested in the average satisficability or the best satisficability we set a threshold and count how many times the system is able to exceed that threshold. In the experiments described below, we define *Best Satisficability* to be the fraction of runs out of n that the GA's best solution in that generation exceeds $s = o\theta$ which is expressed as a fraction θ of the optimal o. This gives us a clear indication of how many times the system will return a result that is at least as good as our threshold. The fraction, θ (between 0 and 1), is a parameter to the measurement. *Average Satisficability* is defined to be the average across n runs of the fraction of individuals in the current population that exceed s. Of course it would be simple to extend this measure to a problem where the optimal was not known, in which case o could be set to 1 and $s = \theta$, which means that the measures would be based on the fraction of individuals or runs to exceed an absolute threshold.

2.3 Robustness

Of all the measures listed here, robustness is probably the most complicated and has the most varied definitions. There are many different notions of robustness [10], and thus defining it must always be done within the context of a particular question. Here we specifically address the idea of robustness as a measure of how a system's output changes in response to environmental changes. We want to know how much the fitness of the next generation of the GA can drop, given the current generation's fitness. The idea is that the performance of the system should never dramatically decrease since a dramatic decrease may upset other elements if the system is not isolated. In order to make this measure more useful in real-world systems it is important to define it so that it is not necessary to know when the environment changed, because changes are not always easily observable. One application of such a measure would be managing assets in a stock market portfolio. For instance a GA could be used to specify which stocks to hold at each time step, with the goal of never suffering a dramatic decreases in the total value of the portfolio. As long as the overall net value is increasing the owner is earning money, but it is important to make sure that this portfolio is robust to dramatic changes.

In the experiments below, we define *Best Robustness* to be the fitness of the best individual in the current generation divided by the fitness of the best individual in the previous generation. If this score is greater than 1 we set it equal to 1. For simplicity we set the robustness score of the first generation equal to 1. We then average this measure across n runs. We define *Average Robustness* to be the same measure but for the average fitness of the population.

2.4 Diversity

Diversity is a measure of the variance in the genomes in the population of solutions. Diversity captures the notion of how much of the search space the GA is currently exploring. Moreover having a diversity of solutions may mean the system is more able to adapt to changes in the environment. The diversity within the run of an EA has been studied many times before, including attempts to use it as an objective in a multi-objective EA fitness function [18]. A diversity of solutions is often needed to avoid premature convergence, which can cause an EA to get stuck at a local optima. In fact a whole variety of techniques have been used to maintain diversity throughout a run, like niching through fitness sharing [6]. Moreover similar techniques have been used to try and maintain diversity in dynamic environments throughout a run [7] as well as after a change in the environment[1] [5]. Besides the study of diversity within GAs for the purposes of studying GAs, it is also important to look at diversity within the context of biological modeling. Very often biological diversity is considered important to the success of a species and thus studying how different parameters of the GA affect the overall diversity of the system could be interesting to biological modelers [20].

The above concept of diversity is concerned with measuring how diverse solutions are within a run. Another measure of interest is how diverse solutions are across runs. This provides an idea of how different the various results of the GA will be between, as opposed to within, runs. In some cases diversity among runs would be a good thing, since it would indicate the system is able to find very different parts of the search space that may not have been obvious as potential areas for fruitful exploration. In other cases diversity may be a bad thing because it indicates that the system is very dependent on initial conditions.

In order to examine diversity we chose to use Hamming distance because it measures how many single bit mutations would be required to move from one string to another within the population. Likewise Hamming distance is also the Manhattan distance between two vertexes in an n-dimensional hypercube. Thus Hamming distance is a good approximation of how far apart two solutions are in the sl-hdf solution space.

In the experiments discussed below, we define *Best Diversity* to be the average Hamming distance between the genomes of the best individuals of each generation found in each of the n runs. We can also observe the diversity within a run and thus, we define *Average Diversity* to be the average Hamming distance between every member of the population averaged over n runs. To allow for comparisons of diversity where GA string lengths differ, we normalize these measures to the length of the string.

However, we do not present the results from the Best Diversity of the experiments. The reason is that in the case of the sl-hdf's the random number seed used to generate the population of the GA is also used to generate the particular problem to be solved. The result of this is that in every run the GA is facing a different problem, and thus the Hamming distance between different runs of the GA is always 0.5 since they are optimizing toward different fitness peaks.

3. SHAKY LADDER HYPERPLANE-DEFINED FUNCTIONS

The functions that we will be utilizing to explore the GA in dynamic environments are a subset of the hdfs [9]. The hdfs are designed to represent the way the GA searches by combining building blocks (through the use of schemata) hence they are appropriate for understanding the behavior of the GA. The hdfs were constructed to meet a set of criteria specified by Whitley [19]. The problem with the hdfs in the dynamic case is that the optimal set of strings is not easily known given the definition of the function, and thus the

[1]For a more thorough review please consult Bränke [3]

absolute performance can not easily be measured. Moreover, there is no way to take one hdf and create another that is similar to it, which would be useful when exploring dynamic environments.

Thus we impose three conditions on the full suite of hdfs and use a simple algorithm to these functions which solves these two problems. The process described below is more thoroughly explained in previous work [15] [14]. The first condition is the *Unique Position Condition* (UPC). It requires that all elementary schemata contain no conflicting bits. The second condition we call the *Unified Solution Condition* (USC). This condition guarantees that all of the specified bits in the elementary level schemata must be present in the highest level schema. The third condition is the *Limited Pothole Cost Condition* (LPCC), which states that the fitness contribution of any pothole plus the sum of the fitness contributions of all the building blocks in conflict with that pothole must be greater than zero. These three conditions guarantee that any string which matches the highest level schema must be a string with optimal fitness. By knowing the optimal set of strings we solve one of the problems with Holland's original hdfs.

Assuming these conditions, once a set of elementary schemata have been established we already know the highest level schema. If we hold the elementary and highest level schemata constant, we can generate new similar hdfs by creating new intermediate schemata, we call this *Shaking the Ladder*. Thus we have an easy way to create similar but different hdfs randomly, and solve the second problem with Holland's original hdfs.

For the experiments discussed below, we used sl-hdf's with a length of 500. There are 50 elementary schemata of order 8, 5 intermediate levels of schemata, and 1 highest level schemata. The length of the schemata is unknown since the location of the fixed bits are chosen randomly. Moreover the order of the higher level schemata is also unknown since the elementary schemata can "share" fixed loci. However we can place upper limits on the order. If all elementary schemata are disjoint, the maximum order of the highest level schemata is 400, which means there will always be at least 100 wildcards present in the highest level schemata.

4. EXPERIMENTS AND RESULTS

The basic setup for our experiments is a simple GA using the sl-hdf as its fitness function. The base GA presented here uses one-point crossover, per bit mutation, full population replacement, and is similar to the one described by Mitchell [12]. In this set of experiments we only change one variable, t_δ, which specifies the number of generations between shakes of the ladder. We only examine two values for t_δ, 1801 and 100. $t_\delta = 1801$ represents a static environment because the time between changes exceeds the run of the GA. $t_\delta = 100$ represents a regularly changing environment. The optimal value achievable by the sl-hdf is 1.0. All results below are presented at 10 generation increments in order to make the graphs easier to read.

Figure 1 illustrates both the fitness of the best individual in the population (Best Performance) and the average fitness of the whole population (Average Performance) for both t_δ values, averaged across 30 runs. These results have been normalized to 0 to 1; the non-normalized optimal value of these sl-hdf's is 191. Figure 2 illustrates (for both t_δ values) how many best of generation individuals out of 30 runs

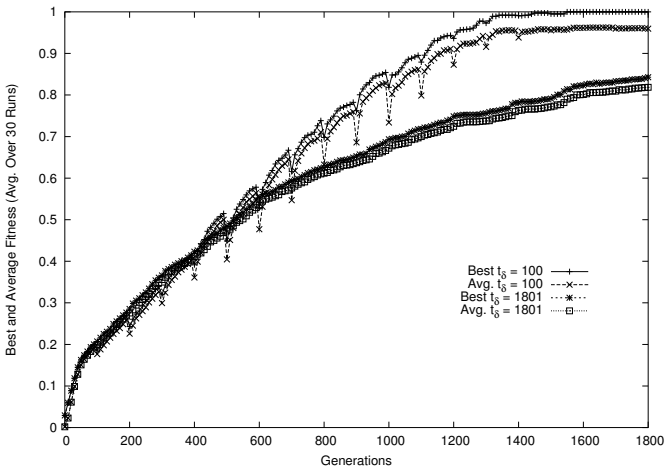

Figure 1: Performance Results

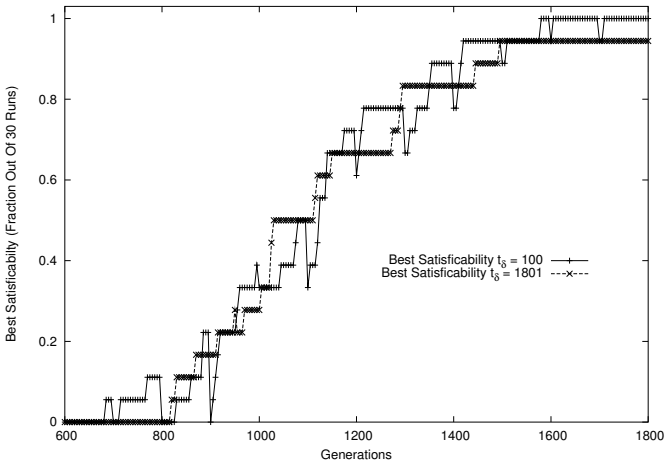

Figure 2: Satisficability Results

(Best Satisficability) were able to satisfice a goal of achieving $\theta = 0.5$, where $o = 191$, and thus $s = 0.5 \times 191 = 95.5$. The actual value of θ that is chosen in this case is arbitrary since the problem is abstract. Instead what is interesting is observing the dynamics of the measure over time which will be explored below. In Figure 2 only the last 1200 generations are presented to increase the resolution of the data. Figure 3 displays the robustness of the best individual in the population (Best Robustness) and the robustness of the average fitness of the population (Average Robustness) for $t_\delta = 100$ across the entire run (averaged across 30 runs). Figure 4 displays the average scaled hamming distance of the population (Average Diversity) for the last 1300 generations for both t_δ values.

5. DISCUSSION

The performance of the system has been more thoroughly

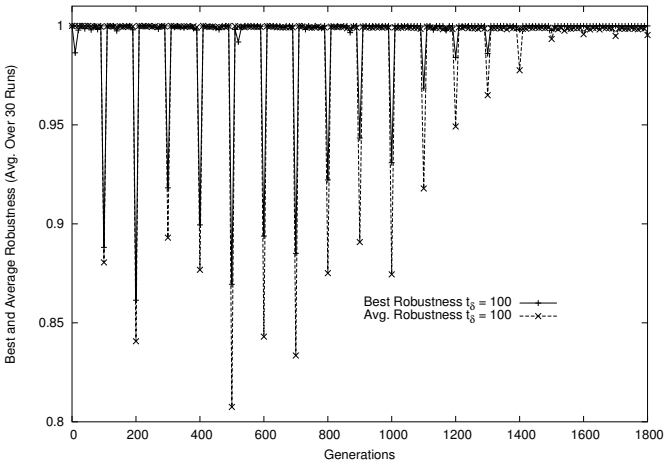

Figure 3: Robustness Results

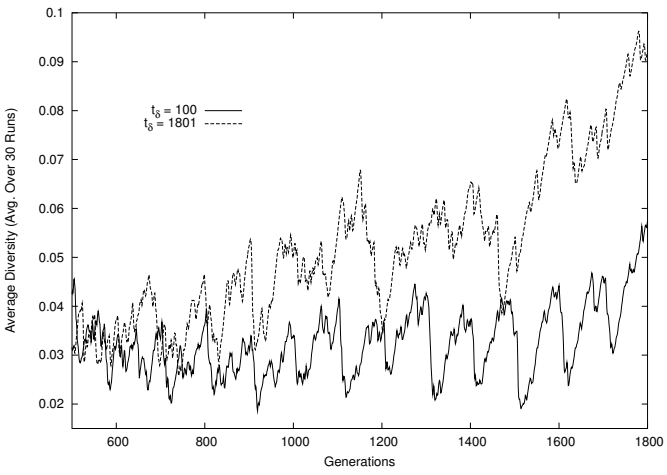

Figure 4: Diversity Results

explored in a previous paper [15]. However, it is clear from the results here that the regularly changing environment is able to outperform the static environment in the long run. Initially the dynamic environment under-performs the static environment but before halfway through the run the dynamic environment has achieved a superior fitness in both the best individual and the average fitness. We believe this is because the regularly changing environment prevents the GA from being locked into particular building blocks and forces it to explore a wider range of intermediate schemata. It is also interesting to note that when the ladder is shaken in the regularly changing environment, the Average Performance of the system falls farther than the Best Performance of the system (see Figure 3 for additional clarification). This makes sense– when the ladder is shaken many of the individuals that were being rewarded before lose those rewards and hence their fitness falls greatly; however it is reasonable to suppose that there are some individuals immediately after a shake that have a higher performance (the new best individuals) than they did before the shake and thus they mitigate the fall of the best performance.

Given the performance results the satisficability results are interesting as well. Since the satisficing results are measures of individuals above a certain threshold, it would make sense that the dynamic environment, which outperforms the static environment, would also have a higher level of satisficability. However, both the static and dynamic environments behave in a similar fashion. The satisficing threshold is set at 0.5 and the elementary building blocks constitute almost half of the fitness function reward. Thus, one hypothesis to explain why both systems are able to achieve similar levels of satisficability is that both the dynamic and static environments are finding basic building blocks at roughly the same rate. However the dynamic environment outperforms the static environment because it is better at finding the intermediate building blocks. The dynamic environment is getting rewarded for different intermediate building blocks and thus has a higher selection pressure to find them, whereas the static environment is mainly under pressure to find the elementary building blocks and the particular intermediate schemata that are rewarded in its sl-hdf instance. We do not present the results of Average Satisficability but it closely mirrors the Best Satisficability, which is interesting since it is a measure within a run instead of across runs, and thus there is no guarantee that it would be the same.

The robustness results are also interesting. The static environment results are not presented since it is almost always able to maintain its robustness, which means that its fitness is constantly improving. The one exception to this is that occasionally the best individual suffers some degradation, probably due to a deleterious mutation. However, for $t_\delta = 100$, every 100 generations the robustness decreases substantially, but then immediately recovers. Moreover, the robustness score at each shake changes as the run goes on. Basically there are three phases to the robustness score, early on (Generation 0 to Generation 400) the decreases are small, in the middle generations (Generation 400 to Generation 1000) they are larger, and in the final generations (Generation 1000 to Generation 1800) they are small again. The first phase is because the population has little fitness value to lose. The population is dominated by individuals who have a few elementary schemata and maybe one or two intermediate schemata, thus the ladder shakes have little ef-

fect on them since it only changes intermediate schemata. At the end of the run the GA has found most of the intermediate schemata, but has not assembled them into one individual, and thus it is not affected much by shakes of the ladder since there is some individual in the population that has the new intermediate schemata. In the middle is when the GA has the largest decreases in robustness, and this is because at this phase the GA has devoted lots of resources to exploring particular intermediate schemata. Also, the Best Robustness score never falls as much as the Average Robustness score. This is explained above when discussing the performance results.

The most interesting result is the diversity results. Our initial hypothesis was that the overall diversity of the static system would decrease as time went on, indicating that the population was converging, and that diversity in the regularly changing environment would increase immediately after a change but then decrease again. However that is not what happens. Instead it appears that for $t_\delta = 1801$ diversity (minus some noise) always increases, whereas for $t_\delta = 100$ diversity varies around some average, but decreases immediately after a ladder shake and then increases until the ladder is shaken again. Our new hypothesis is that the static environment's population is dominated by a few strong individuals but over time these individuals gain additional mutations that are neutral. This results in diversity increasing. In the regularly changing environment on the other hand, when the ladder is shaken, there are just a few individuals that contain the proper bit values to work well in the new environment, and thus these individuals quickly dominate the population. This results in the loss of diversity of all the previous explorations that were going on, and a very sharp founder's effect [9] which results in a quick convergence of the population around these new intermediate building blocks.

This explanation may seem to contradict the earlier statements that the dynamic environment prevents the premature convergence (loss of diversity) of the GA on particular intermediate building blocks. However the diversity measure that we present here is based on bits, not building blocks. There are some bits that never matter even in optimal strings, since they are wildcards in the highest level schema, we will call these highest level wildcards (as mentioned above the lower limit on this is 100 bits). There are other bits that are not as important given the current elementary schemata present in the population and the current intermediate schemata being rewarded, we will call these currently quasi-neutral bits. The problem with these bits is that only if an entire new elementary schemata (8 bits out of 500) is discovered and the proper combination of it with a few other elementary schemata occurs, is there enough of a selection pressure to allow these bits to go to fixation. Thus, diversity increases in the static environment because both highest level wildcards and the currently quasi-neutral bits can mutate without greatly affecting the fitness of an individual. In the dynamic environment diversity is kept lower because the shaking of the ladder causes a strong selection pressure that forces the population to move to a new set of intermediate schemata, and only after that can the currently quasi-neutral bits mutate. However the above explanation does not say anything about the diversity of schemata (building blocks). The dynamic environment has a larger diversity of schemata and that is how it is able to outperform the static environment, whereas the the static environment quickly converges on a group of intermediate schemata and does not search for additional combinations.

6. CONCLUSION

The overall goal of our project is to better understand how the GA works in dynamic environments. We have presented a set of measures that we feel will help us in better understanding the overall behavior of the GA in dynamic environments. Our observations and results here are on the sl-hdf's but we hope to show that these measurements are useful in a wide variety of environments. Though many of these measures have been presented before in different formats, by viewing them as a suite we are able to gain a deeper understanding of how the GA performs in dynamic environments. We plan to continue to conduct systematic controlled observations of the GA. We feel that this allows us to contribute to theory by providing a series of regular observations and to contribute to practice by providing suggestions for a rich set of environments.

Acknowledgments
We would like to thank the University of Michigan's Center for the Study of Complex Systems for the computer resources that they have provided to us without which this paper would not have been possible.

7. REFERENCES

[1] Branke, J. Evolutionary algorithms for dynamic optimization problems: A survey. Tech. Rep. 387, Institute AIFB, University of Karlsruhe, February 1999.

[2] Branke, J. Memory enhanced evolutionary algorithms for changing optimization problems. In *Congress on Evolutionary Computation CEC99* (1999), vol. 3, IEEE, pp. 1875–82.

[3] Branke, J. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2001.

[4] Branke, J., Salihoğlu, E., and Uyar, S. Towards an analysis of dynamic environments. In *To appear in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)* (2005).

[5] Cobb, H. G. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Tech. Rep. AIC-90-001, Naval Research Laboratory, Washington, D.C., 1990.

[6] Goldberg, D. E., and Richardson, J. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms* (Hillsdale, New Jersey, 1987), J. J. Grefenstette, Ed., Lawrence Erlbaum Associates, pp. 41–9.

[7] Grefenstette, J. J. Genetic algorithms for changing environments. In *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)* (Amsterdam, 1992), R. Männer and B. Manderick, Eds., Elsevier, pp. 137–144.

[8] Holland, J. *Adpatation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[9] Holland, J. H. Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary Computation 8*, 4 (2000), 373–391.

[10] Jen, E. Stable or robust? what's the difference? *Complexity 8*, 3 (January / February 2003), 12–18.

[11] Koza, J. R. *Genetic Programming*. MIT Press, Cambridge, Massachusetts, 1992, ch. 8.

[12] Mitchell, M. *An Introduction To Genetic Algorithms*. MIT Press, Cambridge, Massachusetts, 1997, ch. 1.

[13] Morrison, R. W., and DeJob, K. A. A test problem generator for non-stationary environments. In *Congress on Evolutionary Computation CEC99* (1999), vol. 3, IEEE, pp. 2047–53.

[14] Rand, W., and Riolo, R. The problem with a self-adaptative mutation rate in some environments: A case study using the shaky ladder hyperplane-defined functions. In *To appear in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)* (2005).

[15] Rand, W., and Riolo, R. Shaky ladders, hyperplane-defined functions and genetic algorithms: Systematic controlled observation in dynamic environments. In *EvoWorkshops 2005 Proceedings* (2005), R. et al., Ed., Lecture Notes In Computer Science, Springer. In Press.

[16] Simon, H. *Models of Man*. Wiley, New York, 1957.

[17] Stanhope, S. A., and Daida, J. M. Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment. In *Evolutionary Programming VII* (1998), no. 1447 in LNCS, Springer, pp. 693–702.

[18] Toffolo, A., and Benini, E. Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation 11*, 2 (2003), 151–67.

[19] Whitley, D., Rana, S. B., Dzubera, J., and Mathias, K. E. Evaluating evolutionary algorithms. *Artificial Intelligence 85*, 1-2 (1996), 245–276.

[20] Wilson, E. O. The current state of biological diversity. In *Biodiversity*, E. O. Wilson and F. M. Peter, Eds. National Academy Press, Washington, 1988, pp. 3–18.