The Effect of Crossover on the Behavior of the GA in Dynamic Environments

A Case Study using the Shaky Ladder Hyperplane-Defined Functions

William Rand
Northwestern Institute on Complex Systems
220 Annenberg Hall
2120 N Campus Dr.
Evanston, IL, USA, 60208
wrand@northwestern.edu

Rick Riolo and John H. Holland
Center for the Study of Complex Systems
University of Michigan
4485 Randall Lab
Ann Arbor, MI, USA, 48109-1120
rlriolo@umich.edu, jholland@umich.edu

ABSTRACT

One argument as to why the hyperplane-defined functions (hdf's) are a good testbed for the genetic algorithm (GA) is that the hdf's are built in the same way that the GA works. In this paper we test that hypothesis in a new setting by exploring the GA on a subset of the hdf's which are dynamic—the shaky ladder hyperplane-defined functions (sl-hdf's). In doing so we gain insight into how the GA makes use of crossover during its traversal of the sl-hdf search space. We begin this paper by explaining the sl-hdf's. We then conduct a series of experiments with various crossover rates and various rates of environmental change. Our results show that the GA performs better with than without crossover in dynamic environments. Though these results have been shown on some static functions in the past, they are re-confirmed and expanded here for a new type of function (the hdf) and a new type of environment (dynamic environments). Moreover we show that crossover is even more beneficial in dynamic environments than it is in static environments. We discuss how these results can be used to develop a richer knowledge about the use of building blocks by the GA.

Categories and Subject Descriptors: F.2.m [Analysis of Algorithms] Misc. I.2.8 [Artificial Intelligence] Search

General Terms: Algorithms

Keywords: Crossover, Dynamic Environments, Hyperplane-Defined Functions, Genetic Algorithms

1. INTRODUCTION

Investigations into the use of crossover within the genetic algorithm (GA) have been carried out in the past [10]. Some of these investigations have been purely theoretical and some have been more empirical [12, 2]. However, most of this previous work examined the use of different crossover operators and did not investigate the effect of the crossover operator itself. This paper investigates the effect of the crossover operator and presents results on a new type of landscape (the shaky-ladder hyperplane-defined functions) in a new type of environment (dynamic environments). In addition, recent advances in the speed and power of computers allow us to now conduct a set of experiments located between theory and application. By treating the GA operating on a hypothetical and specially designed landscape as an empirical system upon which we are making observations, we can contribute to theory by collecting a set of regularities that need to be explained, and we can help practitioners by making recommendations to them.

In order to conduct controlled observations on the GA in dynamic environments, a test suite of problems is necessary, so that we can control the inputs to the system and define metrics for the outputs. Moreover, the more parameters of the system (e.g. time and severity of shakes, difficulty of the problem) that are controllable, the easier it is to develop explanations for the observed behavior.

Other test suites for EAs in dynamic environments exist, such as the dynamic knapsack problem, the moving peaks problem and more [1]. The test suite presented here is similar to the dynamic bit matching functions utilized by Stanhope and Daida [11] among others. The test functions that we have developed to explore the GA in dynamic environments—the shaky ladder hyperplane-defined functions (sl-hdf's)—are a subset of the hdf's [4]. Holland created the hdf's in part to meet criteria developed by Whitley [13]. The hdf's are designed to represent the way the GA searches by combining building blocks, hence they are appropriate for understanding the operation of the GA.

The sl-hdf's have been explored in previous work [8, 7, 6, 5, 9]. In this paper we explore three variations on the basic sl-hdf's and examine the effect of crossover on these variants. We begin by describing the sl-hdf's and the three variants utilized in this paper. We then describe the experiments that we carried out utilizing these functions, and examine the behavior of the GA with various rates of one-point crossover. Finally we discuss these results and draw some conclusions.

2. SHAKY LADDER HDF'S AND THE VARIANTS

In this section we briefly describe the sl-hdf's and the three variants we will be exploring. Due to space constraints it is impossible to fully describe the sl-hdf's within this paper; for a more detailed description the reader is referred to previous explanations [5]. To make the hdf's [4] usable as a test platform for the GA in dynamic environments we place three restrictions on the hdf's. The first restriction is the *Unique Position Condition* (UPC). It requires that all elementary schemata contain no conflicting bits. The second restriction we call the *Unified Solution Condition* (USC). This condition guarantees that all of the specified bits in the positive-valued elementary level schemata must be present in the highest level schema, and that all intermediate schemata are a composition of lower level schemata. 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 that matches the highest level schema must be optimally valued. Moreover it gives us an easy way to create a similar but different sl-hdf by changing the intermediate building blocks. This process is referred to as "shaking the ladder", i.e. the intermediate schemata are changed which alters the reward structure that describes the fitness landscape. It is through shaking the ladder that dynamism is introduced into the sl-hdf landscape; by controlling how many generations are allowed to occur between shakes of the ladder, the frequency of change in the sl-hdf's can be controlled. Thus in summary these restrictions allow us to transform the full class of hdf's into a class that can be used for exploring the behavior of the GA on dynamic environments.

In addition the sl-hdf's are representative of many real world problems in which there is a hierarchical reward structure with highly epistatic interactions, and where the dynamism in the environment contains regularities. For instance, job scheduling might be such a problem. The reward structure consists of assembling smaller tasks to solve larger and larger tasks, but these sub-tasks are highly interdependent. In addition in a dynamic job scheduling problem there may well be regular sub-tasks that need to be solved on a constant basis. The sl-hdf's feature these same qualities.

There are many parameters that control the construction of the sl-hdf's. A more detailed explanation of these variants can be found in other work [9]. Below we will explore three variants that we are utilizing in this paper and explain the parameter choices associated with them.

2.1 Variations on the sl-hdf

Though we explored many combinations of the basic parameters of the sl-hdf in order to examine the behavior of the GA in dynamic environments, here we concentrate our efforts on understanding three specific variations of the sl-hdf's which span the space of the parameters. These are usually referred to as *variants* (short for "variants of the sl-hdf's") but are occasionally also referred to as *landscapes* in reference to Wright's notion of a fitness landscape (as in the "Cliffs landscape") [14]. We will describe three variants in turn in the following sections. The differences between these variants are summarized in Table 1.

2.1.1 Cliffs Variant: Intermediate Schemata are cliffs

The base case for our experiments is the Cliffs variant, which derives its name from the fact that there are sharp "cliffs" in the landscape and the landscape shakes cause fitness values to change dramatically. This variant alone utilizes the unrestricted construction method of the sl-hdf's. When creating a new intermediate schema using the unrestricted method, all of the previous level schemata, plus the potholes, and the highest level schema can be used to generate the new schema. This is opposed to the restricted construction method, where the only material that is available to construct the next level of schemata is the previous level, and not the potholes or highest level schema. The unrestricted construction method has the property of introducing "cliffs" into the landscape, because the combination of any schemata and the highest level schema is the highest level schema. Thus many intermediate schemata are replaced by copies of the highest level schema. An effect of this is that any string which matches the highest level schema will have a much higher value relative to the other strings than it would in the restricted construction method. Moreover, the effect of having some intermediate schemata combining with potholes or potholes combining with potholes to create intermediate schemata is complicated. Essentially, this has the effect of smoothing transitions since some of the potholes will not be as detrimental as they could be. On the other hand, if there is one of these "bridges" in place and the ladder shakes removing the bridge, an individual who was on that bridge will suffer a sharp decline in fitness because it loses an intermediate schema and gains a pothole. In general the result of these effects is that the fitness landscape and its changes are more sharply defined.

In addition to the unrestricted construction method, the random construction method of the sl-hdf's is used, i.e., the two schemata to be combined to form the next level of schemata are chosen randomly from the pool of available schemata. This is opposed to the neighbor construction method in which only schemata which have centers located near each other can be combined to form the next level of schemata. All of the parameters of these construction methods are explained in more detail in [9].

2.1.2 Smooth Variant: An sl-hdf without cliffs

The case that we refer to as the Smooth variant is constructed using the restricted method, which is the only difference between it and the Cliffs variant. It differs from the Weight variant because the intermediate schemata weight are never changed, and it still uses the random method of intermediate schema construction. This variant is called the Smooth sl-hdf because unlike the sl-hdf with Cliffs, there are no sharp edges or transitions in the landscape. Instead elementary schemata are combined to form intermediate schemata, which then are combined to form the next level of intermediate schemata and so forth.

2.1.3 Weight Variant: Shaking the weights not the form

This variant most closely resembles the hdf's described by Holland. The first change from the previous variants is that the schemata have a fixed length of $50 = \frac{l_s}{10}$ (where l_s is the length of the string), and an order of 8. Second, the intermediate schemata are created using the restricted, neighbor intermediate construction routine. This gives the

Variant	*Construction Method*	*Elementary Schema Length*	*Shaking Method*
Cliffs	Unrestricted, Random	Not Specified	Form
Smooth	Restricted, Random	Not Specified	Form
Weight	Restricted, Neighbor	50	Weight

Table 1: Differences between sl-hdf variants.

sl-hdf a constant and regular form. Finally, when the ladder is shaken, the form of the intermediate schemata remain constant and only the weights of the schemata are changed. This variant has limited length building blocks, and a highly restricted intermediate building block construction routine.

3. CROSSOVER EXPERIMENTS

One of the principle arguments as to why the sl-hdf's are a good testbed for exploring the GA in a dynamic environment is that the sl-hdf's are built in the same way that the GA itself works. In other words, the sl-hdf's are built out of elementary schemata which are then combined together to create higher and higher levels of schemata. The Building Block hypothesis [3] states that the GA works by combining building blocks which are partial solutions to a problem in order to create better and more complex solutions to the problem. The way the GA does this is through the use of crossover. Crossover allows the GA to take partial solutions in good individuals in the population and combine them to create even better solutions.

Thus one hypothesis is that if the sl-hdf's are best solved through combining schemata, then by turning off crossover we should see a dramatic reduction in the performance of the GA. Moreover, a second hypothesis is that since the Weight variant uses short length, small order schemata for its elementary schemata, crossover should have the greatest effect on a GA performing in that environment. In order to test these two hypotheses, we carried out two sets of experiments. In the first experiment, we manipulated the variable t_δ which is the number of generations between times when the ladder is shaken. This experiment compared multiple t_δ values (1, 100, and 1801) for runs of the GA with no crossover and runs of the GA with the normal level of crossover (70% of the next generation created through crossover) and compared the results on all three variants. In the second experiment, we compared multiple crossover rates (0.0, 0.1, 0.3, 0.5, 0.7, 0.9, and 1.0) for one t_δ value (100) in all three variants. In both of these experiments the rest of the GA parameters were set to some standard values. Population size was 1000; the selection method was a tournament of size 3; the mutation rate was .001; the string length was 500, and the GA was run for 1800 generations. These results are discussed by variant below.

3.1 Crossover Results: Cliffs Variant

As mentioned in the previous section, the GA was run on sl-hdf's with three different t_δ values (1, 100, and 1801) with crossover turned off completely and crossover turned on. When crossover is turned on 70% of the individuals in each generation were created through crossover. Mutation still occurs at the standard rate with 1 out of every 1000 bits being flipped every generation. When crossover is turned off, the only evolutionary operator that is being used is mutation, again at a rate of 1 out of every 1000 bits. The results of the best individual's performance of the population (averaged over 30 runs) on the Cliffs variant for every tenth generation is presented in Figure 1.

As can be observed in the graph, crossover always improves the performance of the system. This is true even when comparing the results of the $t_\delta = 1801$ environment, where the GA performed the worst in the runs with crossover, to the no crossover results of the $t_\delta = 100$ environment, which is the environment that the GA with crossover performed best in. The dramatic effect of crossover supports the idea that the GA's primary method of search is through crossover, and thus supports the first hypothesis that the sl-hdf's are indeed a good test suite to examine the GA's performance.

In addition, the GA with crossover improves more rapidly early on in the run than the GA without crossover. This indicates that crossover is mostly used for combining multiple elementary schemata into one string, whereas later on in the runs it does not have as great an effect. This result will be more clear in the next two variants.

Also note that the changes in performance between different t_δ values is different for GAs with versus without crossover. As is illustrated by Figure 1, the GA with crossover performs better in the $t_\delta = 1$ environment than in the $t_\delta = 1801$ environment in the long run. This is not the case in the GA without crossover. Without crossover the GA is not able to adapt to the constant shaking as well, but the mutation operator alone is enough to continually make improvements in the static environment.

Moreover, in the GA without crossover, though the performance in the $t_\delta = 100$ environment does eventually surpass the performance in the $t_\delta = 1801$ environment, it does not do this until much later in the run of the GA than it does in the GA with crossover. The argument was made in previous work [8] that in the $t_\delta = 100$ environment, premature convergence is prevented from occurring because of the shaking of the ladder. However, with mutation being the only operator, it is difficult for the GA to move past local optima.

These results indicate that crossover is even more important for the functioning of the GA in dynamic environments than it is in static environments. The relative performance increase when going from the no crossover GA to the crossover GA is much larger for the dynamic environments than it is for the static environments. One hypothesis is that when a shake of the ladder dramatically changes the landscape the new elementary schemata that need to be recombined are already present in the population but are located in different individuals. Crossover provides a mechanism for the GA to recombine those elementary schemata to discover which combinations are now being rewarded. Without crossover, the GA must rely on mutation which has no ability to recombine existing material and must rediscover elementary schemata in other strings.

In the second experiment of this section, we examined the effect of various rates of crossover on the Best Performance

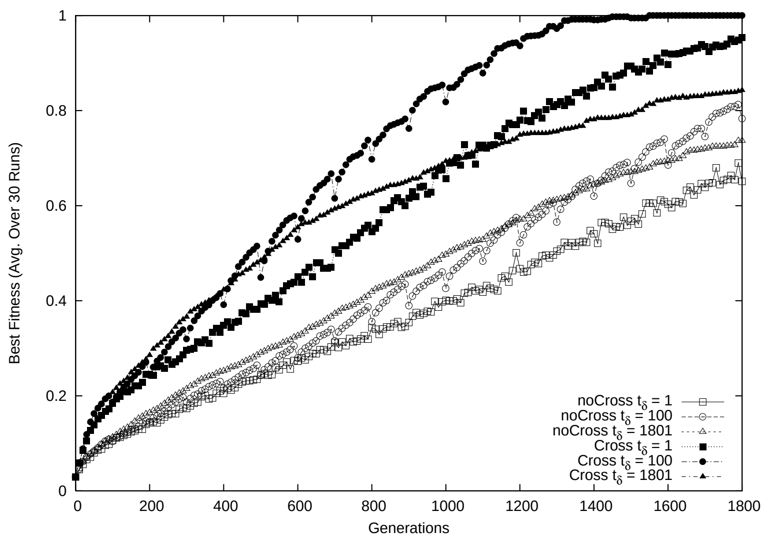

Figure 1: Cliffs Variant: Crossover vs. No Crossover

(averaged over 30 runs) of the population on the Cliffs variant. t_δ was held at 100 throughout these runs. We set the rates of crossover to the following values: 0.0, 0.1, 0.3, 0.5, 0.7, 0.9, and 1.0. The results of this experiment for every tenth generation are presented in Figure 2.

These results support the hypothesis mentioned above that crossover has a dramatic effect on the performance of the GA in the sl-hdf environment. They illustrate that as crossover increases so does the performance of the GA, though the results with crossover rates between 0.7 and 1.0 are roughly equivalent.

These results also illustrate that crossover is most important early on in the run as the GA is assembling elementary building blocks together. Later, as the GA is attempting to discover the last few building blocks that it needs, the performance of the system is roughly parallel for all levels of crossover since, at that point, mutation is the primary mechanism for exploration.

Moreover, these results support a common conception in GA practice that a crossover rate of 0.7 allows for good performance. On the other hand, these results indicate that even a crossover rate of 1.0, which would usually destroy too many schemata to be useful, can still have a positive effect in this environment. This is probably due to the fact that since so much of the landscape is dominated by zero fitness spaces, the positive fitness individuals who will be selected to proceed into the next generation share at least enough bits in common that crossover does not always destroy the schemata they contain.

3.2 Crossover Results: Smooth Variant

Figure 3 presents the results of every tenth generation for the no crossover experiment on the Smooth variant.

In these results, many of the same phenomenon described Section 3.1 can be seen. The performance of the GAs with crossover is better than the performance of the GAs without crossover. In addition, without crossover, the GA performs statistically significantly better (outside a double standard error bar) at the end of the run in the environment that is constantly changing than in the other two environments. In addition, the $t_\delta = 100$ environment is coming close to passing the static environment at the end of the run. The long run dynamics may in fact turn out to be similar to the results with crossover, but in the no crossover GA it may take more time for the dynamics to develop. This again supports the hypothesis that crossover is most important to "jump-start" the GA early on in the run and less important later.

Moreover, we also see support for the hypothesis that crossover is more important in a dynamic environment than a static one. Again, the relative performance increase from the no crossover GA to the crossover GA is much larger in the dynamic environment than in the static environment. Crossover helps the GA recover from shakes of the ladder by recombining elementary schemata in new ways.

Again, in the second experiment on this variant, the t_δ value is held constant while the crossover rate is varied. The results of this experiment for every tenth generation on the Smooth variant are presented in Figure 4.

These results are similar to those presented in Figure 2. More crossover is better, but crossover rates within the range of 0.7 to 1.0 have roughly equivalent performance. Crossover rates between 0.3 and 0.5 also have roughly equivalent performance. Since the GA makes faster progress early on in this variant, it may be that it gets past the point where crossover makes as much of a difference quicker and thus the differences between 0.3 and 0.5 disappear in the long term performance of the system. In fact earlier on in the run there is a difference, but this difference disappears over time. All of these observations support the hypothesis that crossover aids the GA working within the sl-hdf environment, allowing it to make quick progress early on and adapt to changes rapidly.

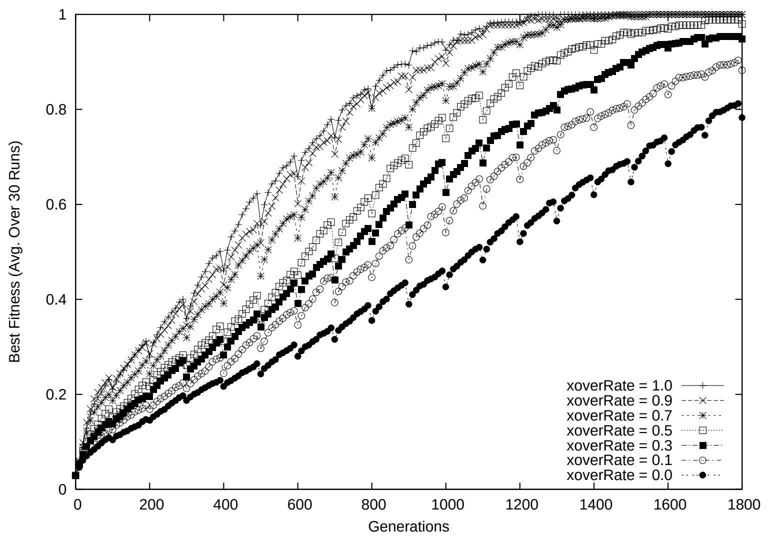

Figure 2: Cliffs Variant: Various Rates of Crossover

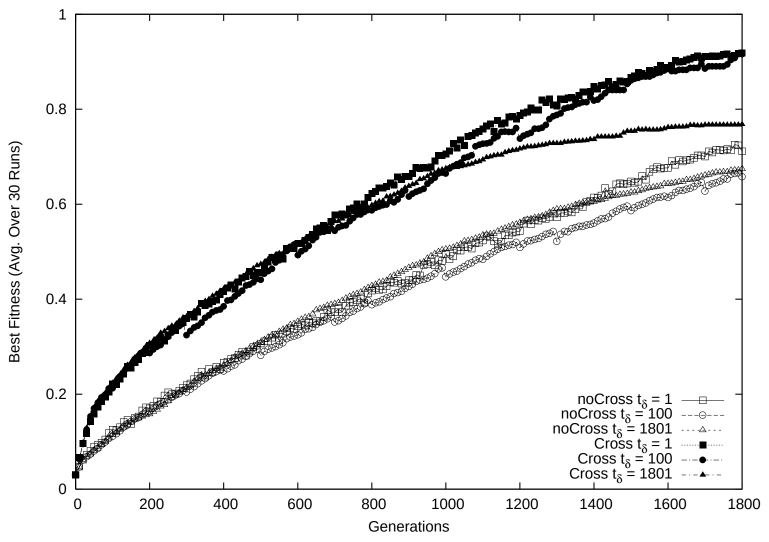

Figure 3: Smooth Variant: Crossover vs. No Crossover

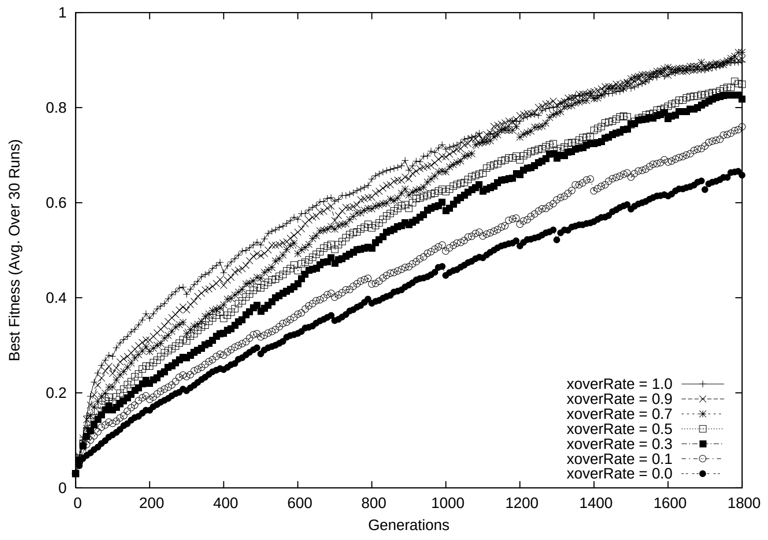

Figure 4: Smooth Variant: Various Rates of Crossover

3.3 Crossover Results: Weight Variant

Figure 5 presents the results of every tenth generation for the no crossover experiment on the Weight variant.

One of our hypotheses was that the GA operating in the Weight variant would be able to make better use of crossover than the GA did in the Cliffs and Smooth variants, and it does early on. Clearly the difference in performance between the crossover runs and the non-crossover runs of the GA in the first 800 generations is substantially greater in the Weight variant than it is in the other two variants. However, as the population nears the optima, crossover does not make as much of a difference. One explanation is that recombination of elementary schemata is very important early on to make progress. However, as the GA approaches the optima, crossover is not as successful since every crossover operation will potentially disrupt a schema, which is explained by the Schema Theorem [3]. Of course once the population has genomically converged then crossover has less of a disruptive effect since crossing over two similar individuals is less likely to disrupt schema. However at this point crossover really is not gaining anything either since all of the individuals look very similar. This supports an additional claim made by Holland that the GA is not really optimizing but instead explores the search space for areas of high fitness. Thus when the GA approaches the optimum it is mutation that really drives the search toward the final global optima. Though one factor that mitigates this effect is that if a long schema is highly concentrated in a few areas of the string then it is less likely to be disrupted. A long, concentrated schema is one where most of its fixed bits are located within a short distance of each other (the concentrated portion), but some of its fixed bits are located far away which makes the schema long. These type of schema are less likely to be disrupted by crossover because even if crossover breaks off a bit or two from the non-concentrated portion of the schema there is a better probability that the other parent may have that bit set anyway. However if the schema is long and not concentrated then the probability decreases that the other individual will have the necessary bits to prevent the schema from being destroyed. This means that our original hypothesis, that crossover would have the greatest effect on the Weight variant, was not incorrect but just incomplete. Crossover does have the greatest effect on the Weight variant early on in the run of the GA.

One final note on this experiment is that crossover does not increase the performance of the GA operating in the dynamic environment more than it does the GA operating in the static environment. This is different from the previous two variants where the results show that crossover is more important in the dynamic environments than it is in the static environments. This can be explained by the fact that since the form of the ladder remains constant in this variant, the GA is not forced to recombine material into new individuals to find those favored by the new environment after a change. Instead the result of the weight variant changes is to change the selection pressure on different combinations of the schemata that are already present in the population. Thus, though crossover does help initially in finding the proper combination of elementary building blocks, it is not as beneficial to adapt to changes in the Weight variant as it is in the previous two variants.

Again in the second experiment on this variant, the t_δ value is held constant while the crossover rate is varied. The results of this experiment for every tenth generation on the Weight variant are presented in Figure 6.

More evidence for the explanation given above for the effect of crossover in the Weight variant can be seen in this figure. Early on in the run, the same differences between the various rates of crossover can be seen, with a higher rate of crossover allowing the GA to perform better. However, as the GA progresses and comes closer to the global optima, the effect of different crossover rates starts to disappear. We observe from the results of the experiments on all three of these variants that crossover does have a dramatic effect on the performance of the GA within the sl-hdf environment.

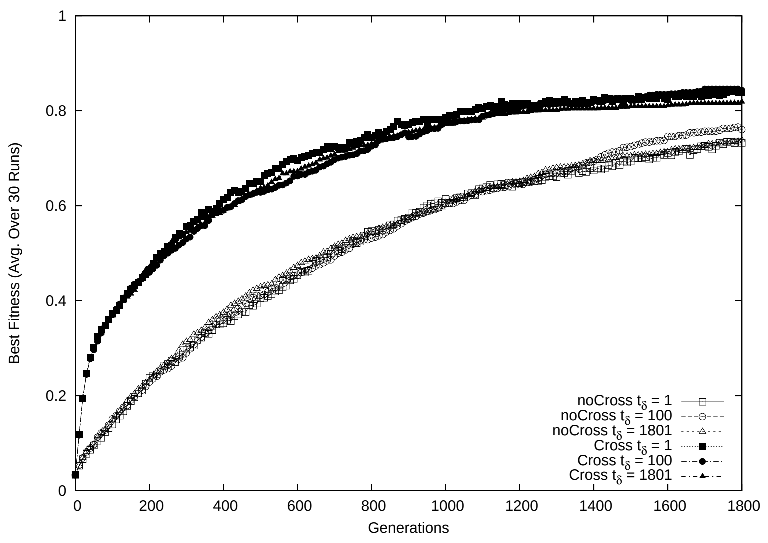

Figure 5: Weight Variant: Crossover vs. No Crossover

Moreover, these experiments also show that when the elementary schemata are short in length and small in order, crossover has the greatest effect. Thus problems with short length, small order building blocks are better suited for the GA than those with long, highly ordered building blocks.

4. CONCLUSION

This paper explores the effect of crossover on the GA operating in the sl-hdf environments. It is shown that crossover has a dramatic effect in the sl-hdf environments, with the GA not being able to perform nearly as well when crossover is only used to a limited degree or turned off completely. In addition, we have shown that crossover is even more important to the performance of the GA in some dynamic environments than it is in a static environment. Moreover, we have provided evidence that a crossover rate of 0.7 allows the GA to operate well. The experimental results support the hypothesis that the sl-hdf is a good environment to utilize in exploring the GA in dynamic environments. In addition, they lend credence to the Building Block Hypothesis by illustrating that without a mechanism to combine small building blocks the GA does not perform as well. Furthermore, the Schema Theorem states that longer schema are more likely to be disrupted by crossover; this phenomenon is illustrated by the fact that the GA operating on an environment with shorter schemata does better early on than the GA operating on an environment with longer schemata. Thus these results also provide further evidence for the Schema Theorem.

Some of these results we expected ahead of time. For instance, crossover as a required mechanism in the successful exploration of the hdf's was one of the designing principles of those test functions. Nonetheless the results of this paper are novel because it is the first time that this has empirically been shown to be true. In addition, there was no clear explanation or hypothesis with regards to the effect of crossover in a dynamic environments. A clearly novel result of this paper is the demonstration that in at least some classes of problems crossover improves the performance of the GA more dramatically in dynamic environments than it does in static environments. The current hypothesis is that this is because dynamic environments require the GA to quickly adapt and move large distances through the search space. Crossover facilitates this process more efficiently by recombining schemata in different individuals within the population, whereas mutation restricts the search to a local neighborhood around individuals. This new result needs to be explored more fully in future work.

These experiments are first steps toward a larger goal of more fully understanding the role of crossover in the GA. For instance, one mystery that arose from these results is that if mutation is more important near the end of the run (even with the crossover mechanism operating) then why do some of the variants make more progress than others toward the goal? We feel that answering questions like these will require studying the GA in different ways than just through the lens of performance. Thus we plan to carry out some of the experiments presented in this paper, but explore in more detail the particular schemata that are contained in the individuals during the course of the GA run. We plan to build a system that will allow us to investigate detailed histories of particular schemata or to examine aggregate counts of schemata across the population simultaneously. Such a system will allow us to more fully explore the interplay of crossover and building blocks.

Acknowledgements: We thank the University of Michigan's Center for the Study of Complex Systems (CSCS) for providing computational resources and support for RR. We also thank the Northwestern Institute on Complex Systems for providing support for WR.

5. REFERENCES

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

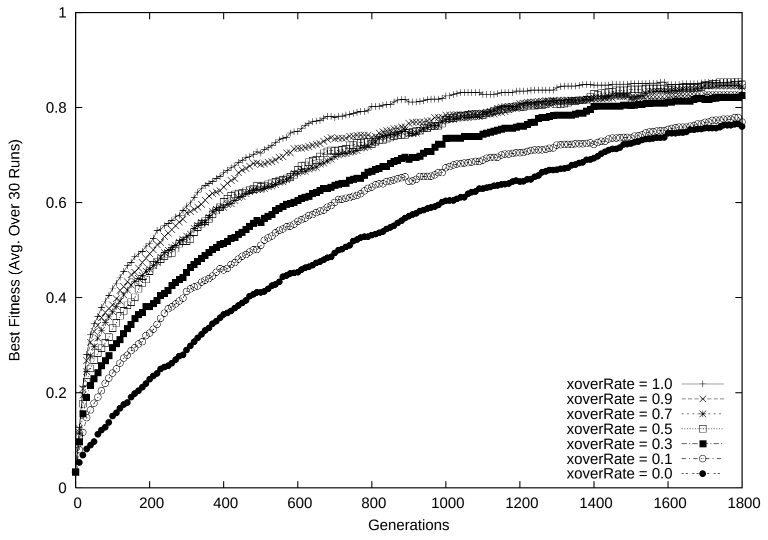

Figure 6: Weight Variant: Various Rates of Crossover

[2] DeJong, K. A., and Spears, W. M. An analysis of the interatcing roles of population-size and crossover in genetic algorithms. In *Lecture Notes in Computer Science* (1991), vol. 496, p. 38.

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

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

[5] Rand, W. *Controlled Observations of the Genetic Algorithm in a Changing Environment: Case Studies Using the Shaky Ladder Hyperplane-Defined Functions*. PhD thesis, University of Michigan, 2005.

[6] Rand, W., and Riolo, R. Measurements for understanding the behavior of the genetic algorithm in dynamic environments: A case study using the shaky ladder hyperplane-defined functions. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005* (New York, 2005), H.-G. Beyer et al., Eds., ACM Press.

[7] Rand, W., and Riolo, R. The problem with a self-adaptive mutation rate in some environments: A case study using the shaky ladder hyperplane-defined functions. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005* (New York, 2005), H.-G. Beyer et al., Eds., ACM Press.

[8] Rand, W., and Riolo, R. Shaky ladders, hyperplane-defined functions and genetic algorithms: Systematic controlled observation in dynamic environments. In *Applications of Evolutionary Computing, Evoworkshops* (2005), F. Rothlauf et al., Eds., vol. 3449 of *Lecture Notes In Computer Science*, Springer.

[9] Rand, W., and Riolo, R. The effect of building block construction on the behavior of the ga in dynamic environments: A case study using the shaky ladder hyperplane-defined functions. In *Applications of Evolutionary Computing, Evoworkshops* (2006), F. Rothlauf et al., Eds., vol. 3907 of *Lecture Notes In Computer Science*, Springer, pp. 776–787.

[10] Spears, W. M., and Jong, K. A. D. On the virtues of parameterized uniform crossover. In *4th Internationl Conference on Genetic Algorithms* (La Jolla, CA, 1991), pp. 230–6.

[11] 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.

[12] Sywerda, G. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic Algorithms* (George Mason University, United States, 1989), J. D. Schaffer, Ed., pp. 2–9.

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

[14] Wright, S. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings 6th Congress on Genetics* (1932), vol. 1, p. 356.