ANALYZING OPTIMAL PERFORMANCE OF EVOLUTIONARY SEARCH WITH RESTART AS PROBLEM COMPLEXITY CHANGES

By

MICHAEL PAUL SOLANO

Bachelor of Science in Microbiology

Oklahoma State University

Stillwater, Oklahoma

2001

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2006

ANALYZING OPTIMAL PERFORMANCE OF

EVOLUTIONARY SEARCH WITH RESTART AS

PROBLEM COMPLEXITY CHANGES

Thesis Approved:

Dr. Istvan Jonyer

Dr. Douglas Heisterkamp

Dr. Johnson Thomas

Dr. Gordon Emslie

Dean of the Graduate College

ABSTRACT

This research explores how the complexity of a problem domain affects the performance of an evolutionary search using a performance-based restart policy. Previous research indicates that using a restart policy to avoid premature convergence can improve the performance of an evolutionary algorithm. One method for determining when to restart the search is to track the fitness of the population and to restart when no measurable improvement has been observed over a number of generations. Our empirical evaluation of such a restart policy confirms improved performance over evolutionary search without restart, regardless of problem complexity. Our work further indicates that as problems become increasingly complex a universal restart scheme begins to emerge.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1

Thesis Statement

Exploration versus exploitation is an inherent trade off of any search space algorithm. Exploration is necessary to traverse the search space in the hopes of finding a region where the global optimum exists but exploitation is necessary to actually move towards that global optimum. In evolutionary search algorithms which utilize a restart policy, the plateau length can determine the emphasis of the algorithm. By decreasing the plateau length, exploration becomes emphasized. By increasing the plateau length, the algorithm tends towards exploitation. Problems of greater complexity tend to have more complicated search spaces and as such, there should be some correlation between the complexity of the problem and the optimal plateau length to use in the restart policy. Specifically, as the search space of a problem becomes more complex, the optimal plateau length should decrease due to a greater need for search space exploration. This research attempts to confirm this correlation between problem complexity and the optimal plateau length.

Chapter 2

Introduction

Genetic programming belongs to a class of randomized search techniques that explore the search space of a problem domain using non-uniform randomized steps. It works in a manner very similar to genetic algorithms, the main difference being that with genetic programming individuals are represented as tree structures (which are essentially programs) and in genetic algorithms strings are used to represent individuals. Using evolutionary operators such as crossover, mutation, and reproduction, genetic programming automatically evolves these programs over a number of generations in an attempt to find a solution to the problem being explored (Koza, 1992). With each successive generation it is hoped that the fitness of the best individual and that of the entire population will be greater than in previous generations. This process usually continues until an ideal individual has been found or a specified number of generations have been processed.

The traditional approach to genetic programming attempts to find optimal individuals in a single run. While this technique has proven successful in a number of problem domains, it is not without its limits. This approach can suffer from phenomena known as *premature convergence* and *bloat*. Premature convergence occurs when the search reaches a local optimum and is unable to move to other parts of the fitness landscape (Goldberg, 1989). Bloating occurs when the size of an individual increases with no

noticeable change in the fitness of that individual (Koza, 1992; Banzhaf, 2001). With bloating, the functional component of a program typically remains small while nonfunctional components (those that never execute) grow without limit. Attempting to minimize these pitfalls continues to be an area of active research.

One method that has shown promise in combating these phenomena is the implementation of a restart policy. A restart involves re-initializing the population based on some event. A static policy restarts the population at predetermined intervals, while a dynamic approach analyzes the performance of the run to determine whether or not a re-initialization should occur (Fukunaga, 1997; Jansen, 2002). Determining what criteria should be used to decide whether or not a restart should occur can have a dramatic effect on the performance of the algorithm.

This work explores whether a relationship exists between the complexity of the problem and the optimal performance-based restart criteria. Does the optimal restart policy change with change in the problem complexity? Can the performance of an evolutionary search with a restart policy be optimized using a complexity metric? We will show that a correlation does indeed exist.

We begin with an exploration of previous work relevant to genetic programming and restart policies. We then describe the concepts on which we build our explorative study, which includes a discussion of a technique for the quantification of problem complexity, and an artificial problem domain the complexity of which can be customized. We describe our restart policy, and our experiments showing the correlation of optimal restart policies with problem complexity. We conclude with a discussion of the results and conclusions.

Chapter 3

Previous Work

Evolutionary search is widely considered to be a random-restart search technique. Individuals may explore one part of the search space (if the random steps happen to be small enough), or jump to an entirely different part of the search space (if the random step happens to be large enough) thereby "restarting" the search elsewhere. Generally speaking, however, large jumps occur increasingly rarely as the number of generations increase. As the population loses diversity it tends to mimic a hill climbing search rather than a true random-restart search (Goldberg, 1989).

Premature convergence highlights one of the fundamental trade-offs in evolutionary search: exploration versus exploitation. Should the search explore a larger part of the search space or should it exploit and refine the best solutions seen so far? The benefit of exploration is a higher probability of finding a better solution while exploitation is advantageous because it offers the possibility of finding the optimal solution more quickly. The essential trade-off involves balancing the need to find a solution fast with the need to find the solution at all. The danger of emphasizing the exploitation component is that the search may become stranded in a local optimum, leading to premature convergence. Overemphasizing exploration, on the other hand, runs the risk of not fully developing promising individuals.

Previous work on addressing premature convergence at the population level includes the utilization of a restart policy (Fukunaga, 1989; Jansen, 2002). While some policies involve restarting the run after a pre-determined number of generations (Jansen, 2002), others use past knowledge about the algorithm or implement a heuristic to dynamically track convergence (Fukunaga, 1997; Luke, 2001; Jansen, 2002). Tracking convergence usually involves measuring diversity within a population or analyzing the performance of the population as the run progresses.

In this work we investigate a restart policy which tracks the performance of a run by tracking the fitness of the best individual in each generation. If there is no measurable increase in performance of the population as the search progresses, a local optimum has been reached. Identifying local optima and determining when to initiate a restart is of pivotal importance.

By allowing a run to proceed without an increase in performance, we are emphasizing the exploitation component of the search hoping that the fitness will continue to increase. By restarting the search, we are emphasizing the exploration component of the algorithm hoping to find an area of the search space in which the optimal individual can be found.

We intend to empirically show that as the complexity of problems increase, an optimal restart scheme becomes evident. Our methodology requires the ability to measure and alter the complexity of a problem domain, which we discuss in the next section.

3.1 Problem Complexity

Defining and quantifying the complexity of a genetic programming problem domain has, until recently, been difficult. A domain independent methodology for determining the complexity of a problem domain was proposed by Tomassini et al (2005) which

defines the complexity of any problem in terms of a single value called the *fitness distance correlation*. The fitness distance correlation statistic was first used to quantify the complexity of GA problems (Jones and Forrest, 1995) by defining the relationship between the fitness of individuals and the distance of the genotypes of those individuals to the genotype of the ideal individual. This is a very straightforward technique when dealing with binary strings of fixed length. Adapting this technique to GP problems, however, requires a method for defining the distance between two program trees.

Structural distance (Ekart and Nemeth, 2002) provides a convenient method to calculate the distance between any two program trees. It requires that given a set of functions and terminals, a coding function must be devised to assign a value to any element in that set. Thus, the coding function takes the form

$$c : \{F \cup T\} \rightarrow N$$

where N is the coded value associated with the function or terminal. Many criteria such as the size or complexity of a function can assist in defining the coding function. For the problem domain explored herein, the arity of functions and terminals provide an ideal guideline for defining *c* (Tomassini et al 2005). The coding function will be described in further detail in the next section when describing the problem domain.

Computing the structural distance between two trees is a three-step process (Ekart and Nemeth, 2002; Tomassini et al, 2005). First, the trees are superimposed in a technique applied recursively starting with the leftmost subtree. Then, the difference of the coded values for each node pair at matching positions is calculated. Finally, the values

computed in previous recursions are combined into a weighted sum. The process to find the structural distance between trees T_1 and T_2 with roots R_1 and R_2 can be defined as

$$dist(T_1, T_2, k) = d(R_1, R_2) + k\sum_{i=1}^{m} dist(child_i(R_1), child_i(R_2), \frac{k}{2})$$

where $d(R_1, R_2) = \left(\left|c(R_1) - c(R_2)\right|\right)$ is the distance between R_1 and R_2, $child_i(X)$ is the i^{th} of the m possible children of some node X, k is a decreasing term for a weighted sum, and $c(Y)$ returns the coded value of the function at node Y.

Using the structural distance, it becomes possible to calculate the fitness distance correlation (*fdc*) for a particular problem domain. The importance of the *fdc* metric is that it provides a quantifiable relationship between the fitness of an individual and its distance from the ideal individual. This value can be used to gauge problem complexity (Tomassini et al, 2005). Computing this value requires generating a set of n individuals and computing the fitness of each individual and the structural distance of each individual from the global optimum. This generates a set $F = \{f_1, f_2, \ldots, f_n\}$ of fitness values and a set $D = \{d_1, d_2, \ldots, d_n\}$ of distances. It is then possible to calculate the *fdc* value as

$$fdc = \frac{C_{FD}}{\sigma_F \sigma_D}$$

where

$$C_{FD} = \frac{1}{n}\sum_{i=1}^{n}\left(f_i - \overline{f}\right)\left(d_i - \overline{d}\right)$$

is the covariance of F and D, and σ_F, σ_D, $\overline{f}$ and $\overline{d}$ are the standard deviations and means of F and D (Tomassini et al, 2005).

Previous research with genetic algorithms (Jones and Forrest, 1995) suggests that a problem may be classified into three categories based on the *fdc* coefficient. Research with genetic programming (Tomassini et al, 2005) has shown that the same qualification scheme employed by Jones for genetic algorithms can be extended to genetic programming. Problems classified as *Straightforward* are those with $fdc \leq -0.15$, where the fitness increases as the structural distance to the global optimum decreases. When $-0.15 < fdc < 0.15$, the problem is classified as *Difficult*. For these problems, there is no correlation between the structural distance of the individual and the fitness. When $fdc \geq 0.15$, the problem is classified as *Misleading*, and the fitness increases as the structural distance to the global optimum increases.

With these methods for calculating the structural distance between trees and the complexity metric of a problem, a suitable domain must be chosen to determine whether or not the complexity of a problem affects the criteria for an optimal restart policy. In the next section we describe our experimental set up, including a problem domain for which it is possible to artificially manipulate the complexity.

3.2 Trap Functions

The trap function makes it possible to define the fitness of an individual as a function of the distance of that individual to the global optimum (Deb and Goldberg, 1993). A language syntax (Punch et al, 1996; Tomassini et al, 2005) that easily lends itself for use with a trap function defines a set of functions with increasing arity and a single terminal. The terminal X has an arity of 0 (arity(X) = 0) while the function set (A, B, C, D, …) is

comprised of functions with increasing arity (arity(A) = 1, arity(B) = 2, arity(C) = 3, arity(D) = 4, …). For the purposes of this paper, the maximum arity is set to 5, which limits the function set to $F = \{A,B,C,D,E\}$ and the terminal set to $T = \{X\}$.

This language provides a convenient method for devising a coding function to assist in calculating structural distance. The arity of the function or terminal is used by the coding function c to assign a value to a given node (Tomassini et al, 2005). The coding function is formally described as

$$\forall x \in \{F \bigcup T\} \quad c(x) = arity(x) + 1.$$

A trap function defines the fitness of an individual by the distance of that individual to the global optimum. By manipulating the parameters of the trap function, it is possible to change the complexity of the function and the problem domain. Formally, the function allowing the conversion of distance into fitness is defined as

$$f(d) = \begin{cases} 1 - \dfrac{d}{b} & if\ d \le B \\ \dfrac{R*(d-b)}{1-B} & otherwise \end{cases}$$

where d is the distance of the individual to the global optimum and B and R are constants with values between 0 and 1 (Tomassini et al 2005).

In essence, the trap function creates a fitness landscape in which two peaks exist. One peak corresponds to the global optimum while the other peak is a globally sub-optimal local optimum. The B parameter in the trap function corresponds to the width of the

global optimum and, correspondingly, 1 – B defines the width of the local optimum. Increasing the value of B increases the width of the globally optimal peak thus decreasing the complexity of the problem. The R parameter defines the height of the sub-optimal peak with the global optimum having a corresponding value of 1. Increasing the value of R increases the height of the sub-optimal peak thus increasing the complexity of the problem.

Chapter 4

Experimental Methodology

4.1 Problem Domains

This work examines three separate problem domains. The trap function domain is chosen because the complexity of this problem can be manipulated easily. In addition, we explore two well-known problem domains: artificial ant and two-box. The artificial ant problem involves navigating a robotic ant to find all the food on the irregular sante fe trail. The two-box problem involves the symbolic regression of a mathematical expression for the difference between the volumes of two boxes using a data set comprised of independent and dependent variables within this domain.

4.2 Experimental Setup

For each domain, the results of 150 runs are averaged in both no-restart and performance-based restart approaches. When discussing the restart policy, we will use the term *plateau length* to refer to the number of generations to allow a run to continue without any fitness improvement. The term comes from the fact that in such situations a plateau can be observed on the plot depicting the fitness of the best individuals as the search progresses. Plateau lengths from 1 to 20 are utilized. In each domain, the population size is set to 1024 individuals and runs are allowed to continue for 100

generations. The probability of success for each run is calculated as the number of runs for which an ideal individual was found divided by the total number of runs.

We investigate several variations of the trap function by manipulating the B and R values to alter problem complexity. Specifically, eighty-one instances of the trap function are produced for B and R values ranging from 0.1 to 0.9 with increments of 0.1. A perfect-D tree is chosen as the ideal individual and is used when calculating the structural distance. Such a tree has the highest arity function, in this case D, at the root with each function in the tree having only functions of one less arity as children (Punch et al, 1996).

Chapter 5

Experimental Results

In this section we discuss the results obtained using the above described experimental procedure. First, we present the results using the artificial domain in which the correlation of problem complexity with our random restart scheme is examined. We then apply our scheme to two well-known domains.

5.1 Artificial Domain

The complexity of the fitness landscape of the unimodal trap function experiments is shown in Figure 1. The problem difficulty is gauged in terms of the *fdc* value which shows a steady transition from trivial to complex as the B values decrease and R values increase. Of greatest interest are the experiments in which the *fdc* values are high (most complex) and those in the transition from trivial to high.

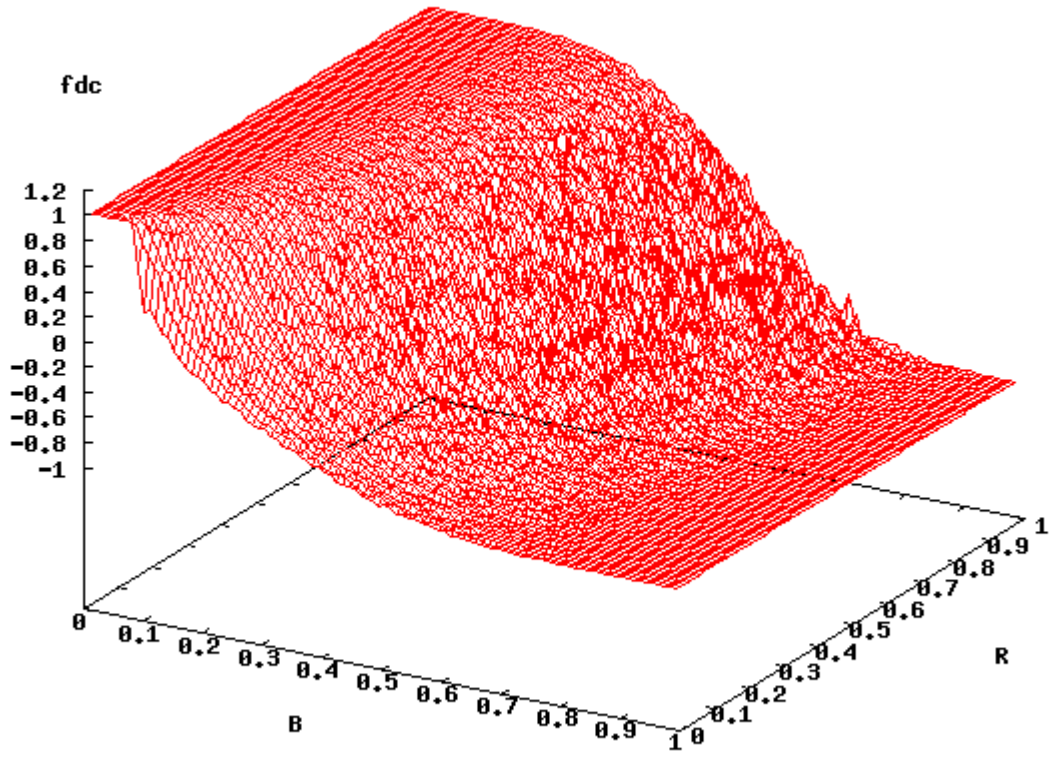

Figure 1: Unimodal Trap Function Complexity

The results from running the trap function experiment without a restart policy are displayed in Table 1. As expected, when parameter B decreases or parameter R increases the success probability of the experiment decreases indicating an increase in problem difficulty. In Table 2, the results from running the identical experiments with the best restart policy (determined by finding the optimal plateau length from all experiments run) are given. When comparing these to the no restart approach it becomes evident that using an optimized restart policy does markedly improve the performance of a genetic

programming run when solving non-trivial problems. Figure 2 contains a graph which compares the data found in these tables.

	Success Probability								
Function Parameters	R=0.1	R=0.2	R=0.3	R=0.4	R=0.5	R=0.6	R=0.7	R=0.8	R=0.9
B=0.1	0.09	0.03	0.01	0.01	0.00	0.00	0.00	0.00	0.00
B=0.2	1.00	0.98	0.88	0.65	0.37	0.16	0.01	0.00	0.00
B=0.3	0.99	0.99	0.99	0.99	0.97	0.83	0.45	0.11	0.00
B=0.4	0.99	0.98	0.98	0.99	1.00	1.00	0.96	0.45	0.04
B=0.5	0.99	0.98	0.99	0.99	0.99	1.00	0.99	0.94	0.24
B=0.6	0.99	0.99	0.99	0.98	1.00	0.99	0.97	0.98	0.74
B=0.7	0.99	1.00	0.99	0.99	0.97	0.99	0.99	0.99	0.93
B=0.8	1.00	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.94
B=0.9	0.97	0.99	0.98	0.99	0.98	0.98	0.99	0.99	0.94

Table 1: No Restart Success Probability

	Success Probability								
Function Parameters	R=0.1	R=0.2	R=0.3	R=0.4	R=0.5	R=0.6	R=0.7	R=0.8	R=0.9
B=0.1	0.61	0.47	0.27	0.11	0.04	0.01	0.00	0.00	0.00
B=0.2	1.00	1.00	1.00	1.00	0.99	0.77	0.27	0.02	0.00
B=0.3	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.44	0.01
B=0.4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.15
B=0.5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.69
B=0.6	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
B=0.7	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
B=0.8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
B=0.9	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 2: Optimized Performance-Base Restart Success Probability

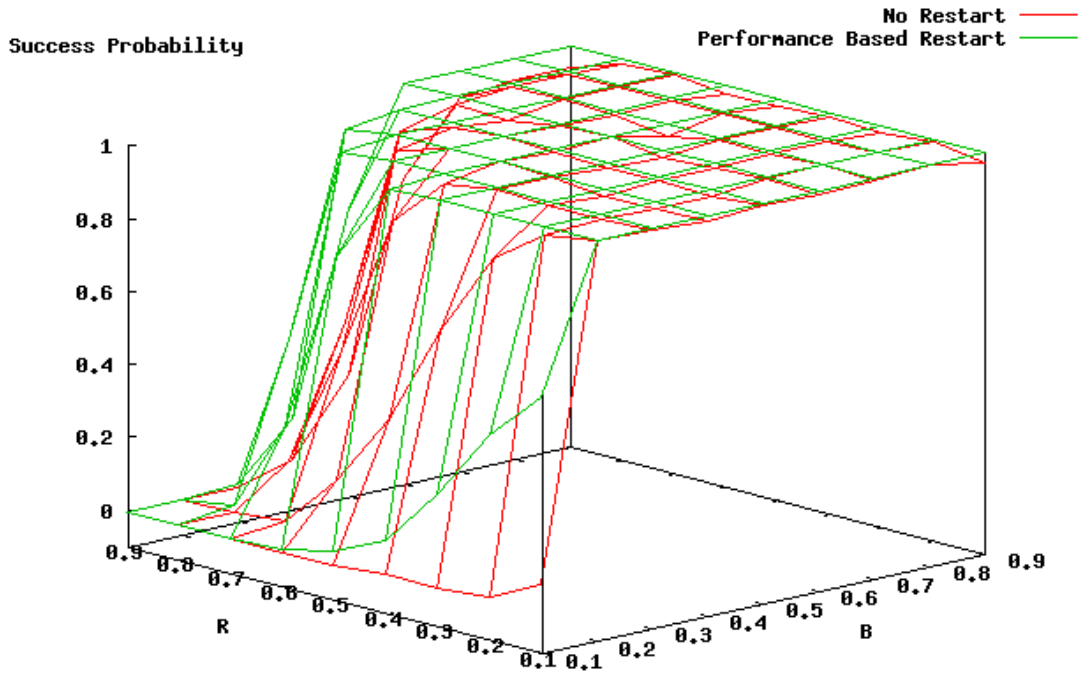

Figure 2: Performance of Evolutionary Search with and without Restart

As discussed earlier, maintaining population diversity is one of the major reasons for using a restart policy. The plots displayed in Figure 3 show two example runs (one with the performance-based restart policy and one without) using identical experimental parameters (B = 0.1 and R = 0.1). The run using a restart policy has its plateau length set to 5. The graph shows that the diversity of the experiment not using a restart policy quickly decreases as the run progresses while the run using the restart policy maintains diversity through restarts. Figure 4 shows that altering the plateau length effectively changes the frequency of restarts. This figure plots the number of restarts that occur as the plateau length increases for each of the trap function experiments. A trend emerges in which the number of restarts decrease as the plateau length increases.

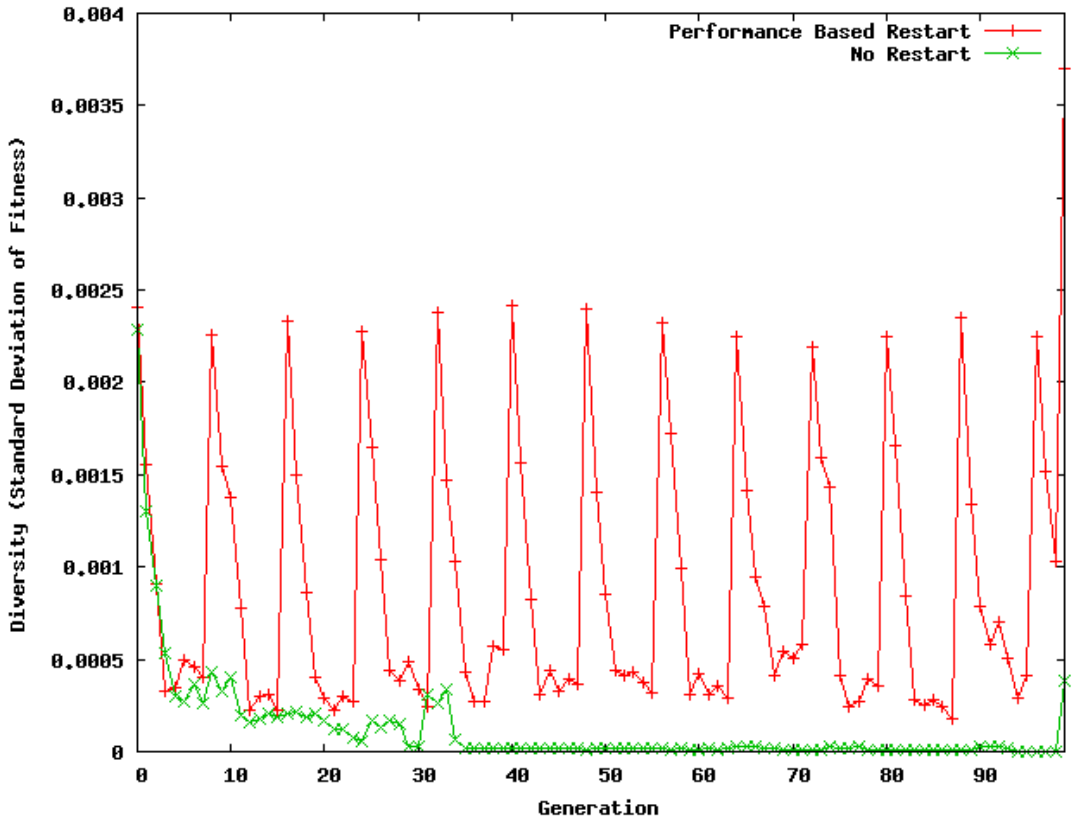

Figure 3: Comparison of a Single Run with and without Restart

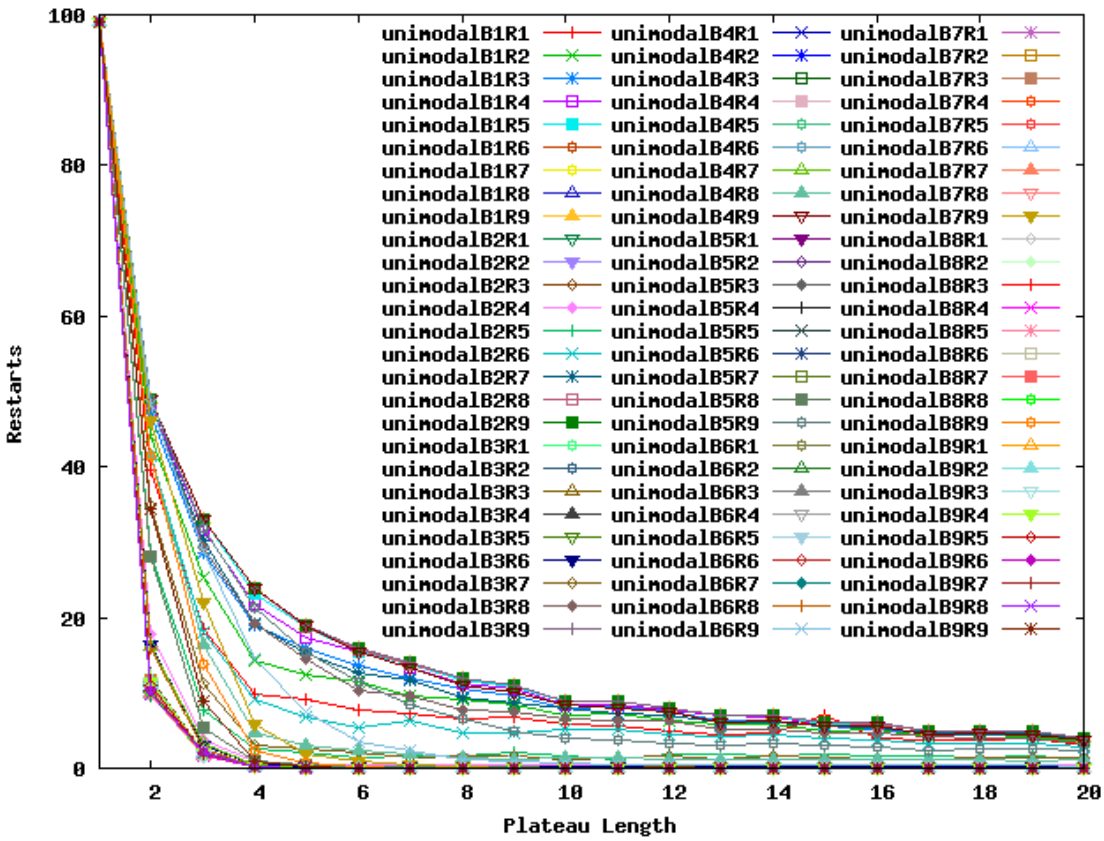

Figure 4: Restart Frequency

The results from running the trap function experiments with the performance-based restart policy are displayed in the plots below. For comparison, the success probability from the experiments without a restart policy (Table 1) is given for each plot. The *fdc* values computed for those experimental parameters (Figure 1) are given as well.

Figure 5 includes the plots for trap functions with B=0.1. For functions using higher R values (0.6 through 0.9) the probability of success remains low. For functions using lower R values (0.1 through 0.5), however, a curve emerges and optimal plateau lengths of 4 or 5 become apparent. At the highest complexity, little success is achieved regardless of the plateau length used. As problems begin to drop in complexity, an optimal plateau length emerges.

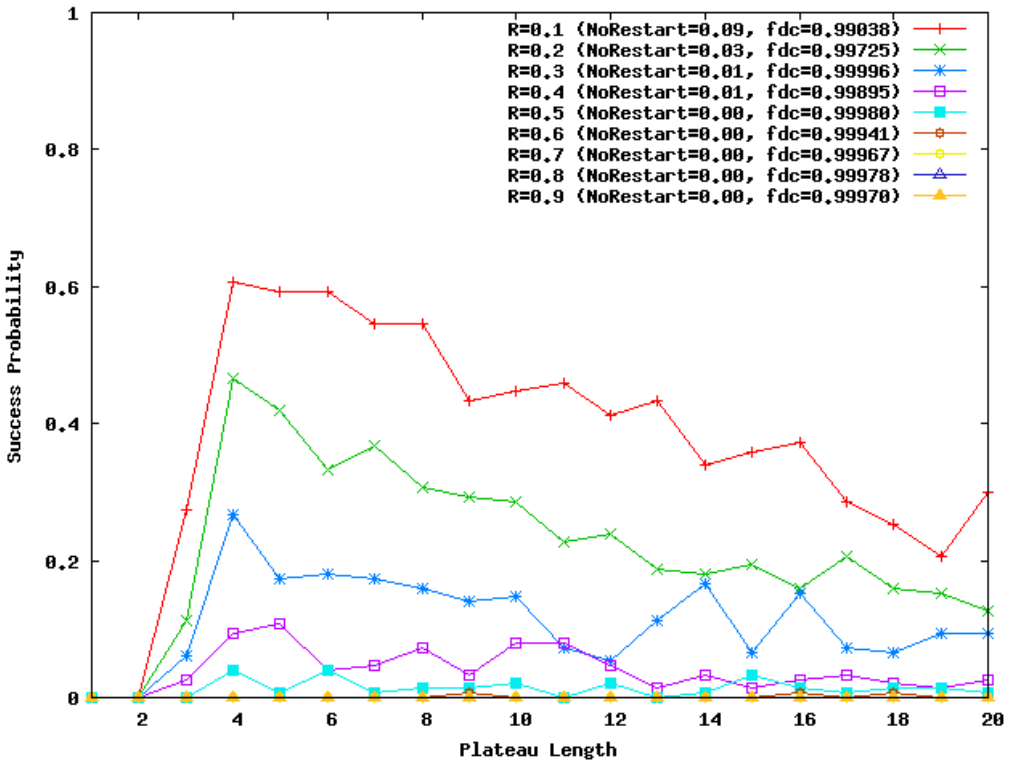

Figure 5: Trap Functions with B=0.1

Figure 6 includes the plots of trap functions for which B=0.2. The functions using the lowest R values (0.1 through 0.3) achieve 100% success for all plateau lengths greater than or equal to 6. When R=0.8 or 0.9, there is a low probability of success regardless of the plateau length. When R is between 0.4 and 0.7, however, a curve emerges for which the best performance occurs at a plateau length of either 6 or 8 depending on the value of R chosen. As in the previous results, with a high R value (and thus a higher complexity), the plateau length has a minimal effect on the performance. This is the case at low complexities as well where 100% success is achieved regardless of the plateau length. As the R value drops, however, and before the problem becomes trivial, a trend emerges where an optimal plateau length becomes apparent. It's interesting to note that the optimal plateau lengths that emerge when B=0.2 are slightly higher than when B=0.1. This may correspond with a decrease in complexity as B decreases.

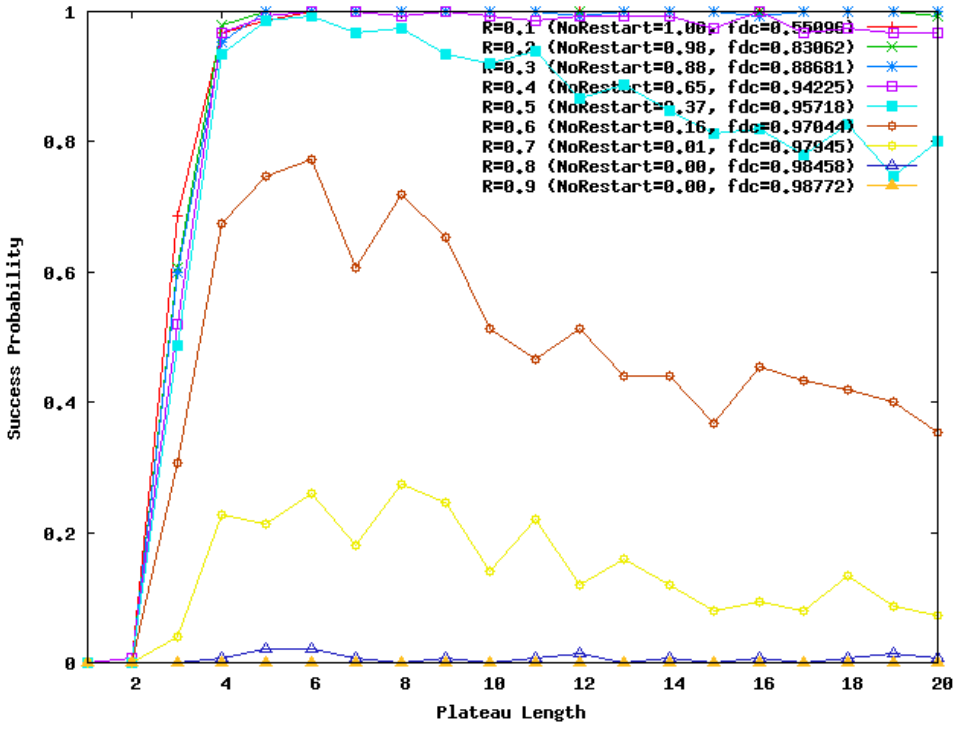

Figure 6: Trap Functions with B=0.2

Figure 7 includes the plots of trap functions for which B=0.3. The functions using the lowest R values (0.1 through 0.6) achieve 100% success for all plateau lengths greater than or equal to 4 with no decrease in performance as the plateau length increases. With R = 0.9, there is a low probability of success regardless of the plateau length. When R=0.7 and 0.8, however, a curve emerges for which the best performance occurs at a plateau lengths of 7 and 8 respectively. Again, for the most complex and most trivial problems, the plateau length has minimal effect on performance. As the problems transition from the highest complexity and before becoming trivial, a trend emerges where an optimal plateau length becomes apparent. As was the case earlier, the optimal plateau lengths increased as we increased the value of B.

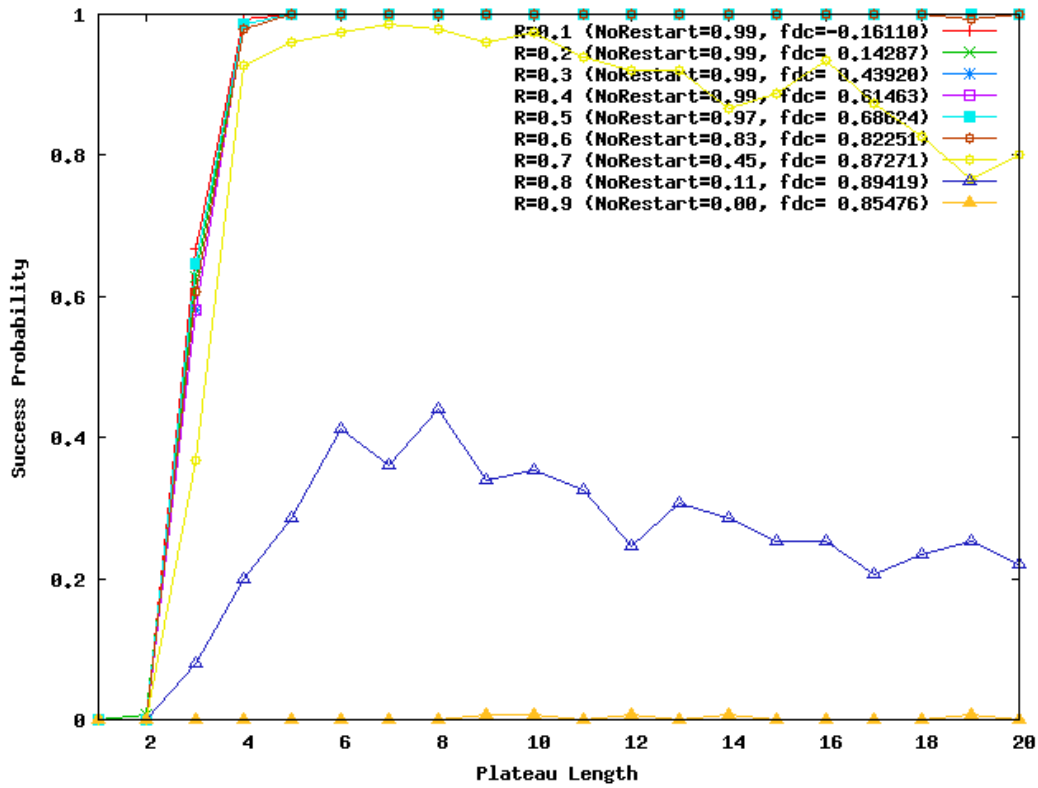

Figure 7: Trap Functions with B=0.3

The plots in Figure 8 show the performance of the trap functions when B=0.4. For the lowest values of R (0.1 thru 0.7), 100% success is achieved for instances with the plateau length set to 5 and higher. When R=0.8 and 0.9, however, a curve emerges in which the greatest performance occurs at a plateau lengths of 6 and 13 respectively. This set of experiments further indicate that while the plateau length has little effect on easier problems, as problems become more complex an optimal plateau length is evident. Again, there is evidence, at least in the R=0.9 case, that with the step from B=0.3 to B=0.4 the optimal plateau lengths increased. This isn't apparent for B=0.8 but that plot was atypical in that it showed optimal performance at two plateau lengths (6 and 9) and showed only a slight performance decrease until after a plateau length of 13.

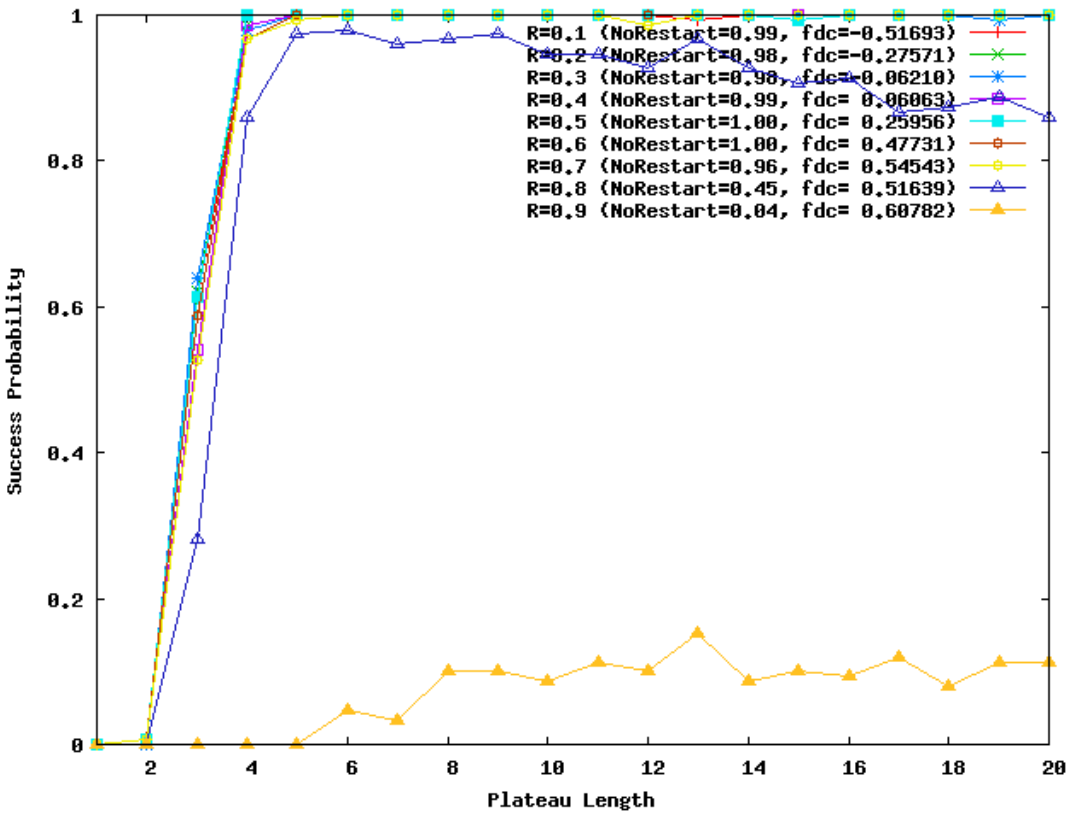

Figure 8: Trap Functions with B=0.4

The plots in Figure 9 show the performance of the trap functions when B=0.5. For the lowest values of R (0.1 thru 0.8), 100% success is achieved for instances with the plateau length set to 4 and higher. When R=0.9, however, a curve emerges in which the greatest performance occurs at a plateau length of 12. The trend continues in which the optimal plateau length observed in this problem is higher than problems with lower values of B.

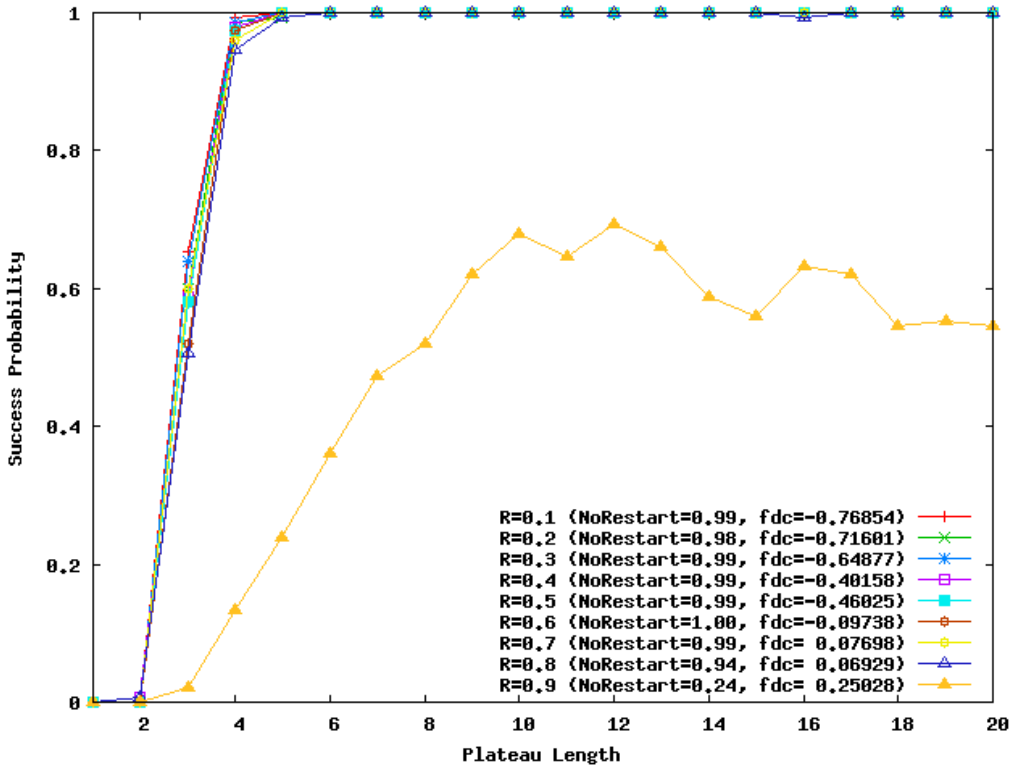

Figure 9: Trap Functions with B=0.5

The rest of the plots found below display the performance of trap functions when B >= 0.6. In all cases, a performance of 100% is obtained once the plateau length reaches an optimum and does not diminish as the plateau length increases. These experiments seem to fall in the trivial category where the problems are of such low complexity that the plateau length has minimal impact on the performance.

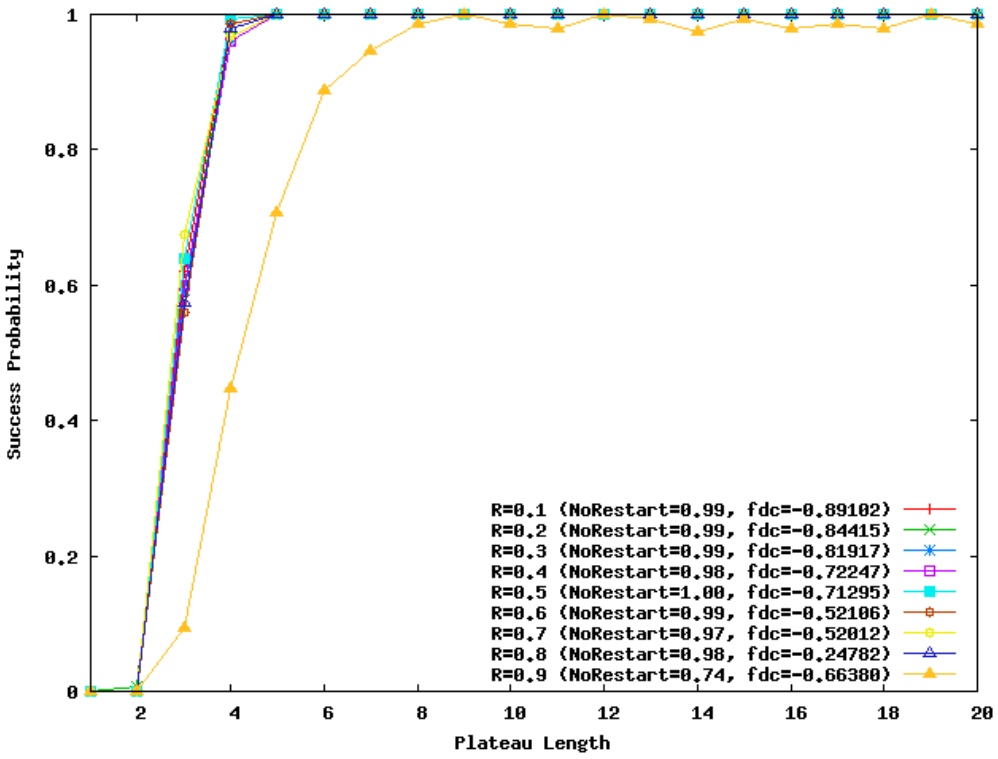

Figure 10: Trap Functions with B=0.6

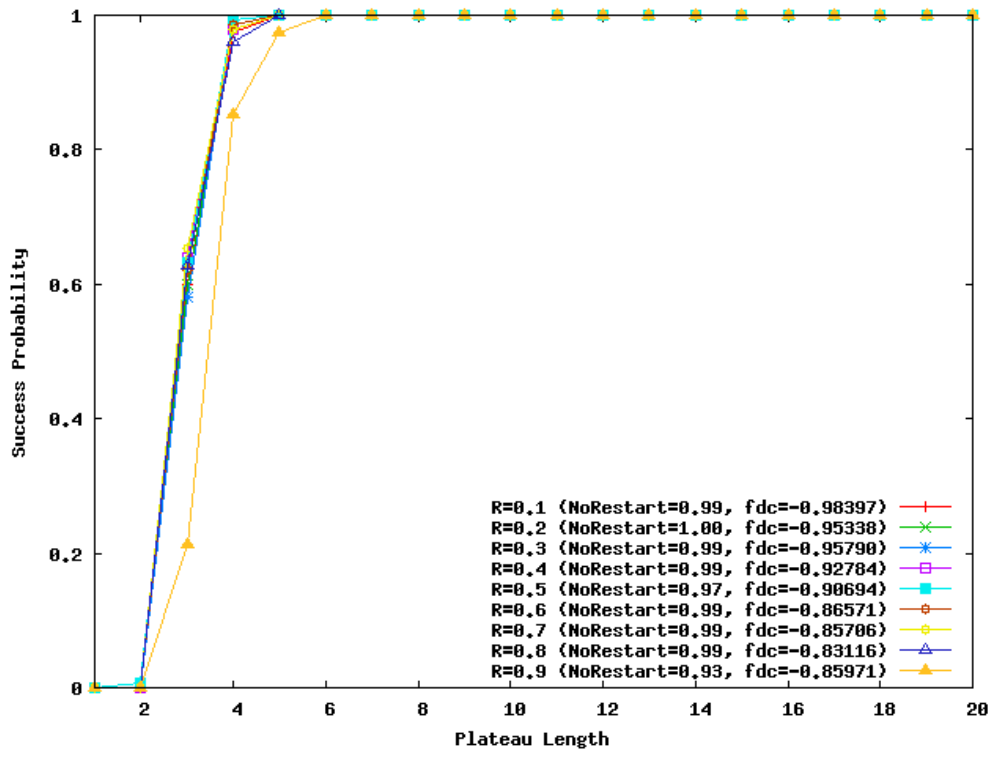

Figure 11: Trap Functions with B=0.7

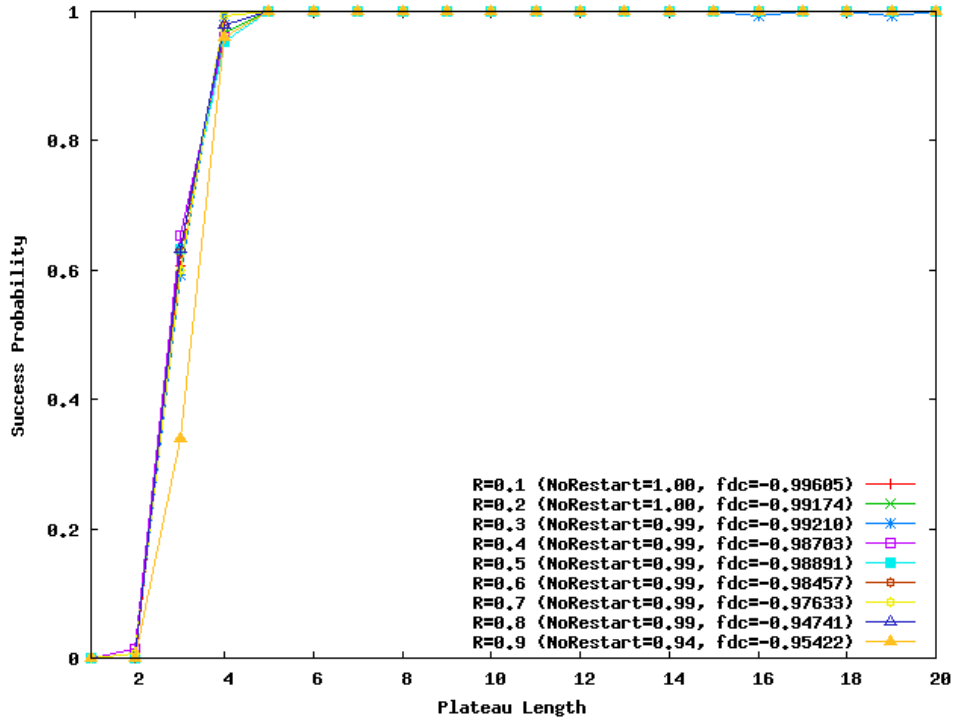

Figure 12: Trap Functions with B=0.8

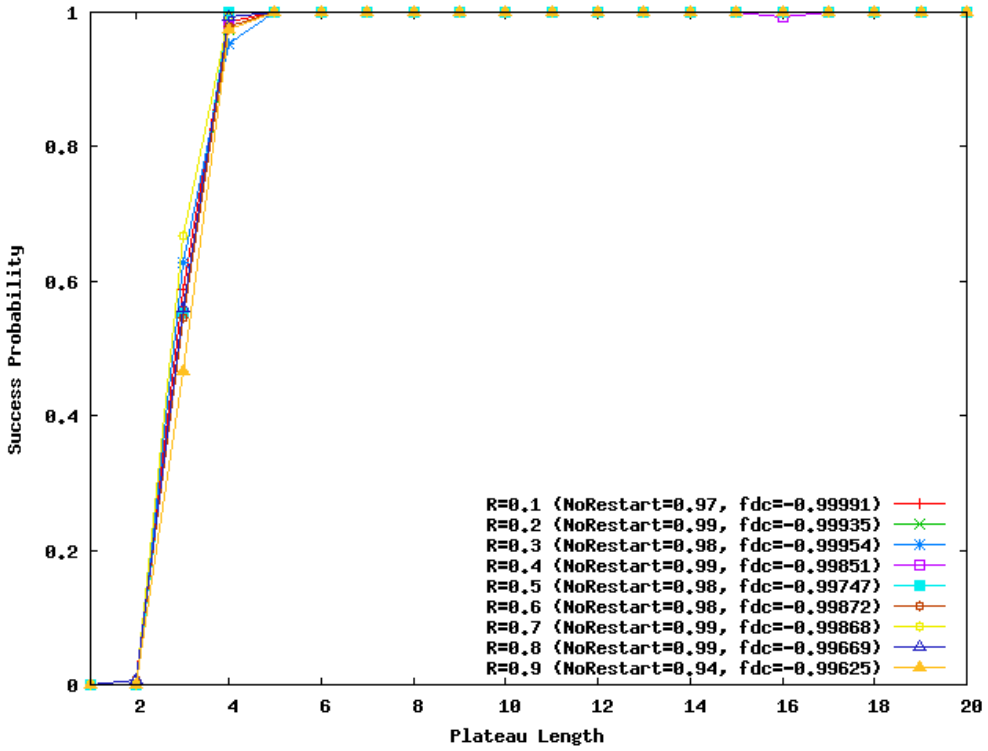

Figure 13: Trap Functions with B=0.9

The success rate of the experiments follow three basic patterns depending on the complexity of the problem. The first pattern involves a curve in which the success of the problem increases with the plateau length to an optimal peak after which the success steadily decreases as the plateau length continues to increase. For these problems, the chance of initializing the population in an area that will eventually lead to the global optimum is better than in the hardest problems but not high enough to negate the benefits of a restart policy.

An optimal plateau length emerges in 14 of the trap function variations. When we compare these instances, we see a correlation between the optimal plateau length and the complexity of the problem as indicated by the *fdc* value. Of these instances, 11 have relatively high *fdc* values ranging from 0.873 to 0.999. The best performance in these cases occurs at a plateau length between 4 and 8. The last instances have much lower *fdc* value (ranging from 0.250 to 0.608) and achieve optimal performance using longer plateau lengths between 6 and 13. The *fdc* values of these instances actually overlap the fdc values of some of the trivial cases which may indicate that the *fdc* value is not a precise indicator of complexity. Regardless, the B and R values for this problem domain are good indicators of complexity.

The second pattern involves trivial problems (lower R values and higher B values) which achieve a 100% success rate at a certain plateau length (usually between 4 and 6) and maintain this success as the plateau length increases. For these problems, there is a high chance that the population will initialize in an area of the fitness landscape where exploitation will eventually lead to the perfect individual. Thus, the restart policy has little effect since in most cases a restart will not be necessary.

The last pattern involves the hardest problems (higher R values and lower B values) which achieve minimal success regardless of the plateau length. In these cases, there is an extremely low chance that the population will initialize in an area of the fitness landscape that is close enough to the global optimum to where no major landscape features must be overcome. The probability of this is so low it negates the benefits of the restart policy.

As the evidence shows the plateau length has little effect on problems with low and very high complexities. There is, however, a complexity range in which the plateau length greatly influences the performance. For this range, it appears that problems on the lower end of the complexity scale perform better with a longer plateau length than problems at the higher end. This suggests that at a lower complexity it is best to put more emphasis on the hill climbing aspect of the algorithm but at higher complexities, it is best to emphasize exploration through restart.

5.2 Artificial Ant and Two-Box Domains

The artificial ant and two-box problems were chosen because they are among the harder of the popular toy problems. We wanted to verify our findings on the trap function on more realistic domains. The results shown below from running the artificial ant and two-box problem without restart indicate a relatively low probability of success. A comparison to the results without restart in the trap function domain indicates the ant and two-box problems are of relatively high complexity.

The plots in Figures 14 and 15 show the success probability of the artificial ant and two-box problems respectively. In the artificial ant problem, a curve emerges in which the best performance occurs when the plateau length is set to 6. The two-box problem indicates a curve in which the best performance occurs when the plateau length is set to

4. These results add evidence to the finding that for problems of higher complexity, setting the plateau length lower and emphasizing exploration through restart improves performance.

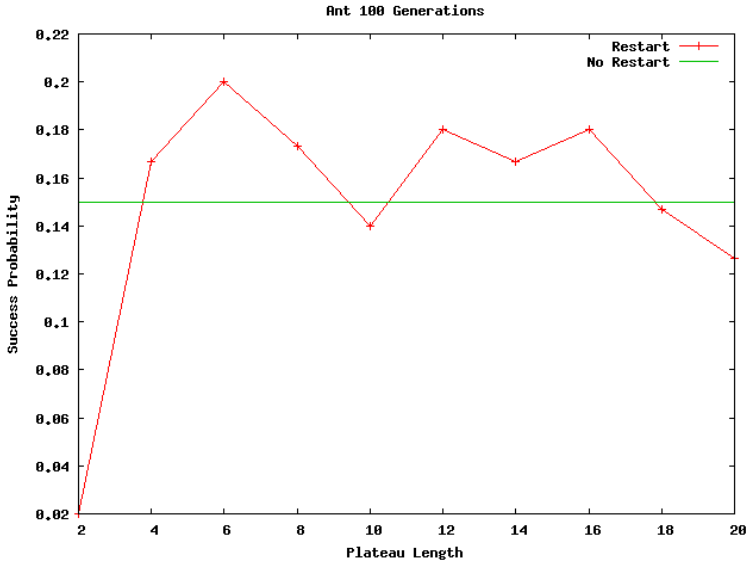

Figure 14: Artificial Ant with and without Restart

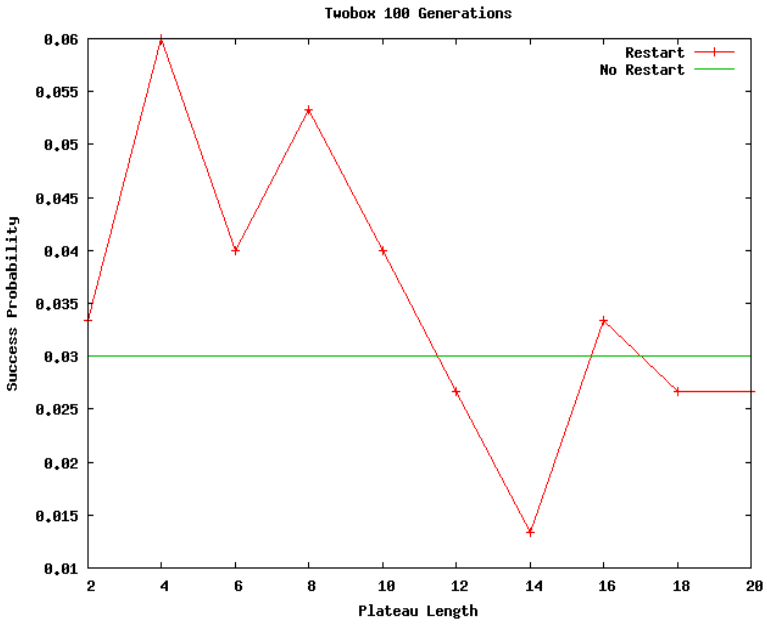

Figure15: Two-Box with and without Restart

Chapter 6

Conclusion

In this work, the performance of genetic programming with and without a dynamic random restart policy was compared in three problem domains. In the trap function domain, the complexity of the trap function was manipulated to show how a restart policy is affected by problem complexity. These experiments showed that for problems where an optimal plateau length existed, a correlation between the plateau length and the complexity of the problem was evident: for problems of higher complexity a shorter plateau length results in better performance, but for problems with lower complexity a higher plateau length tends to be optimal. In higher complexity problems it is easier to make progress along the many dimensions of the search space, although the progress is more likely to be small and towards a local optimum. A short plateau length is best here, promoting exploration, since local optima here are difficult to overcome. In simpler problems restarting the search often has a negative effect, because the search can overcome local optima relatively easily, and converge faster when the run is allowed to continue. The experiments show that exploration versus exploitation can be emphasized by adjusting the plateau length of our restart policy. Experiments on the artificial ant and two-box domains confirm this observation.

We plan to follow up this research by devising a restart policy that adjusts its own plateau length based on problem complexity. The major challenge will be to determine

problem complexity during the run. Also, even with averaging a large number of runs using the same parameters, some of the plots show jagged features, such as seen in Figure 6. We will investigate the underlying phenomena, which will hopefully lead to additional insight into dynamically determining and responding to problem complexity.

References

1) Banzhaf, W. and Langdon, W. B. 2002. Some Considerations on the Reason for Bloat. Genetic Programming and Evolvable Machines, Vol. 3. pp. 81-91.
2) Deb, K. and Goldberg, D. E. 1993. Analyzing deception in trap functions. In Foundations of Genetic Algorithms, 2. pp. 93-108. Morgan Kaufmann.
3) Ekart, A. and Nemeth, S. Z. 2002. Maintaining the diversity of genetic programs. In Genetic Programming, Proceedings of the 5th European Conference. Vol. 2278 of LNCS. pp. 162-171. Springer-Verlag.
4) Fukunaga, A. 1997. Restart scheduling for genetic algorithms. In Genetic Algorithms: Proceedings of the Seventh International Conference.
5) Goldberg, David E. 1989. Genetic Algorithms in Search, Optimization, & Machine Learning. Boston, Massachusetts.: Addison-Wesley.
6) Jansen, Thomas. 2002. On the analysis of dynamic restart strategies for evolutionary algorithms. In Proceedings of the 7th International Conference on Parallel Problem Solving From Nature (PPSN VII). Berlin, Germany.: Springer.
7) Jones, T. and Forrest, S. 1995. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In Proceedings of the Sixth International Conference on Genetic Algorithms. pp. 184-192. Morgan Kaufmann.
8) Koza, John. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, Mass.: The MIT Press,.

9) Luke, Sean. ECJ: A Java-based evolutionary computation and genetic programming system. Available at http://www.cs.umd.edu/projects/plus/ecj/, 2000.

10) Luke, Sean. 2001. When Short Runs Beat Long Runs. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001). pp. 74-80. San Francisco, CA.: Morgan Kaufman.

11) Punch, B., Zongker, D., and Goodman, E. 1996. The royal tree problem, a benchmark for single and multiple population genetic programming. In Advances in Genetic Programming 2. pp. 299-316. Cambridge, Mass.: The MIT Press.

12) Tomassini, M., Vanneschi, L., Collard, P., Clergue, M. 2005. A Study of Fitness Distance Correlation as a Difficulty Measure in Genetic Programming. Evolutionary Computation. Vol. 13. Issue 2. pp. 13-239. Cambridge, Mass.: The MIT Press.

13) Vanneschi, L. and Tomassini, M. 2002. A Study of Fitness Distance Correlation and Problem Difficulty for Genetic Programming. GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference. pp. 307-310

VITA

Michael Paul Solano

Candidate for the Degree of

Master of Science

Thesis: ANALYZING OPTIMAL PERFORMANCE OF EVOLUTIONARY SEARCH WITH RESTART AS PROBLEM COMPLEXITY CHANGES

Major Field: Computer Science

Biographical:

Personal Data: Born in Tulsa, Oklahoma on December 23, 1978, the son of Ambrose and Victoria Solano

Education: Graduation from Booker T. Washington High School, Tulsa, Oklahoma in May 1997; received Bachelor of Science degree in Microbiology from Oklahoma State University, Stillwater, Oklahoma in May 2001. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December, 2006.