DISSERTATION

EXAMINING THE ROLE OF LOCAL OPTIMA AND SCHEMA PROCESSING IN GENETIC SEARCH

Submitted by
Soraya Rana
Department of Computer Science

In partial fulfillment of the requirements
for the Degree of Doctor of Philosophy
Colorado State University
Fort Collins, Colorado
Summer 1999

COLORADO STATE UNIVERSITY

July 1, 1999

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY SORAYA RANA ENTITLED EXAMINING THE ROLE OF LOCAL OPTIMA AND SCHEMA PROCESSING IN GENETIC SEARCH BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Adviser

Department Head

ABSTRACT OF DISSERTATION

EXAMINING THE ROLE OF LOCAL OPTIMA AND SCHEMA PROCESSING IN GENETIC SEARCH

Several factors contribute to making search problems easy or difficult. One of these factors is the modality of the fitness landscape. Quite often, when local search algorithms fail to locate the global optimum, it is because the algorithm converged to a local optimum. The majority of local search methods in use today maneuver through the search space using local neighborhood information around a single point to guide the search. In that search paradigm, the number of local optima that occur in the search space has a tremendous effect on search performance.

Genetic algorithms are a population based search algorithm that use an ever changing neighborhood structure, based on the population mixture and genetic operators, to sample points in the search space. Genetic algorithms are believed to process schemata, where a schema is a subpartition of the search space, rather than individual points. When genetic algorithms fail to locate the global optimum, the typical analysis is that the schema information in the optimization problem was misleading. Consequently, it is unclear how local optima can affect such algorithms. While misleading schema information may be one source of problem difficulty for genetic algorithms, the existence of multiple *high quality* local optima may also be responsible for misleading the genetic algorithm. This research explores the relationship between local optima, schema processing and genetic algorithm behavior from a variety of perspectives.

Soraya Rana
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523

Summer 1999

TABLE OF CONTENTS

Chapter 1

Introduction

Genetic algorithms are population based optimization algorithms inspired by evolution (Holland 1975). From an evolutionary perspective, the continued survival of an organism is based on its ability to adapt to its environment. New organisms are created by either asexual or sexual reproduction and compete with other organisms in order to survive and propagate. It is through reproduction and/or mutation that adaptation occurs. If the adaptation is beneficial, the organism will thrive; otherwise it will become extinct. Genetic algorithms were designed to mimic this evolutionary process in the digital realm. Given that evolution was the inspiration for genetic algorithms and that evolution does not proceed rapidly, it is questionable whether or not the evolutionary model is, in fact, an efficient model for optimization.

Genetic algorithms have proven to be effective optimization techniques for many applications in engineering, economics, manufacturing, artificial intelligence and operations research. As with any search algorithm, there are advantages and disadvantages of using genetic algorithms for optimization purposes. The amount of *a priori* knowledge required to use genetic algorithms is minimal. One can apply an "off the shelf" genetic algorithm to an optimization problem by mapping all of the function inputs to a pre-specified representation such as binary strings. However, to obtain superior performance, one often specializes a genetic algorithm to a specific problem domain by using additional heuristics, specialized representations or specialized operators, which can be argued to constitute *a priori* information. If one opts to use a genetic algorithm without expending effort to customize the genetic algorithm to the problem at hand, the tradeoff is typically that the non-specialized genetic algorithm may produce poor results, which may include computationally inefficient optimization or convergence to a suboptimal solution. The effectiveness and efficiency of a genetic algorithm for optimization depends upon the amount of customization done to the algorithm for the specific problem. Customization can be a change in representation, the use of special genetic operators or non-standard operations such as hybridization with local search or the use of random restarts.

Despite the fact that most genetic algorithms in use for optimization are customized to the particular domain, most theoretical results for genetic algorithms are based on Holland's canonical genetic algorithm (Holland 1975). This model of a genetic algorithm utilizes a single, finite population of individuals represented as bit strings. The genetic operators, which are fitness proportionate **selection**, one point **crossover** and uniform random **mutation** for the canonical genetic algorithm model, are applied to the population to produce a new population of individuals. The genetic operators are described in detail in Chapter 2. An application of the genetic operators is called a **generation**. Initially, a genetic algorithm population consists of a random sample of points in the search space. Over many generations, the genetic algorithm population becomes increasingly uniform until it ultimately converges to a single point in the search space.

Genetic algorithms are widely believed to search by processing schemata. The schema processing view of genetic algorithm behavior is not typically considered to be related to local optima. Genetic algorithms are usually applied to bit represented optimization problems. For bit represented search problems, a schema is a subset of points defined by bit-wise similarities between the points. Since the genetic algorithm is theoretically manipulating several subsets of points, it is unclear that a single point can directly affect the performance of the genetic algorithm. Local search algorithms, by definition, terminate search at local optima, where a **local optimum** is a point in the search space for which there are no immediate improvements in a local neighborhood defined around that point. For bit represented problems, local optima are defined with respect to single bit flips (e.g., a Hamming distance one neighborhood). For local search algorithms, the existence of multiple local optima is known to be problematic.

This research shows that schema relationships are strongly influenced by the size of basins of attraction of local optima and that local optima are likely to be genetic algorithm convergence points for finite-population genetic algorithms using mutation. Showing that local optima affect schema relationships and that genetic algorithm performance is, in turn, affected by local optima illustrates that both genetic algorithms and traditional local search algorithms are adversely affected by the same landscape feature: local optima with large basins of attraction.

1.1 Schema Processing

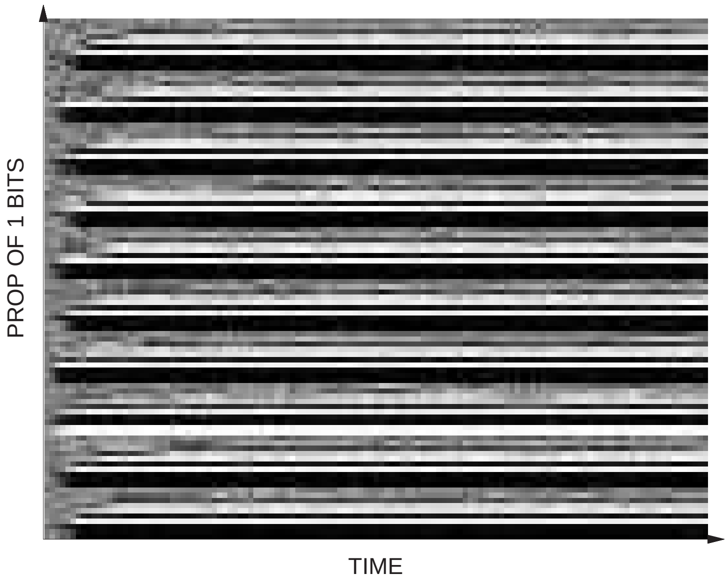


Figure 1.1: Population mixture of a Simple Genetic Algorithm changing over time.

Theoretically, the computational power of genetic algorithms lies in their ability to process **schemata** simultaneously in every generation (Holland 1975; Goldberg 1989c). Schemata are subsets of bit strings that form subpartitions of the search space. A **schema** is denoted by a concatenation of {0,1,*}, and the **order** of the schema is the number of 0 or 1 bits occurring in the string. For instance, 1**** is an order-1 schema defining the subpartition of the search space in which all five-bit strings begin with a 1 bit. Genetic algorithms process schemata by using the population to hold competitions between schemata implicitly and in parallel. The result of the competitions is that the low order schemata with an observed higher fitness (based on the population) will be allocated exponentially larger numbers of trials over time.

Figure 1.1 graphically illustrates the convergence behavior of a traditional genetic algorithm applied to a parameter optimization problem. The search problem consists of 100 bits and the vertical axis represents the proportion of the population having that bit position set to a `1`. When a square is gray, the population has an even mix of `0` and `1` bits. When a square is white, the population consists of strings with mostly `1` bits at that position. The horizontal axis in the image represents the generations (time). On the left side of the image, the image consists of large regions of gray squares, which indicates that the genetic algorithm population has reasonable variation. The image quickly becomes striated with black and white stripes. These are low order schema that the genetic algorithm has found to be highly fit. Over time, the population becomes even more striated. It is clear that the schemata having a high representation early in the search propagate throughout the entire search.

Despite the fact that many genetic algorithms used in practice are very different from the canonical genetic algorithm, the belief is still prevalent that all genetic algorithms are processing schemata. The modifications that practitioners make to improve genetic algorithm performance may include the use of uniform rather than one point crossover, hybridization with local search operators, such as the 2-opt operator used for the Traveling Salesman problem, the use of small populations with high mutation rates or random restart mechanisms. These modifications may vastly improve the performance of genetic algorithms for optimization, but they wreak havoc from a schema processing standpoint.

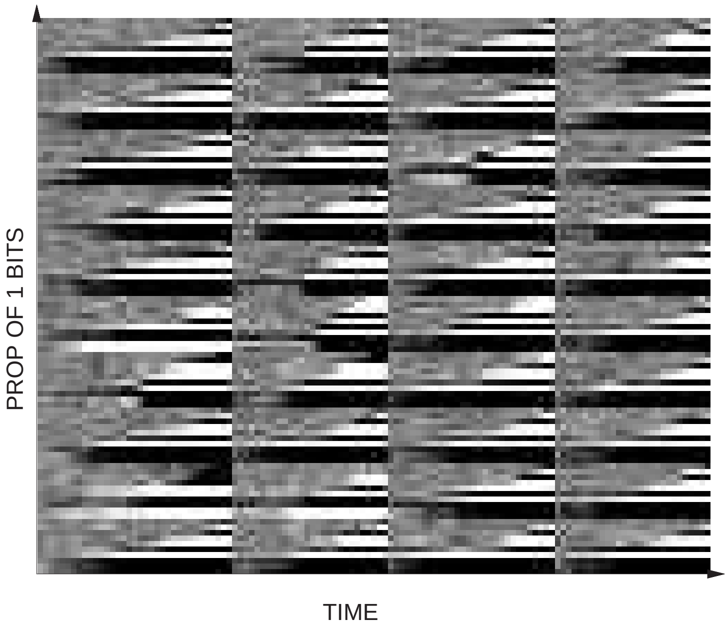


Figure 1.2: Population mixture of the CHC algorithm changing over time.

Figure 1.2 is the convergence behavior of an unusual form of genetic algorithm, CHC, described in depth in Chapter 3. CHC utilizes random restarts, which cause the discontinuous appearance in the image. Like the traditional genetic algorithm, CHC begins searching from an initial population of strings. However, the population quickly and aggressively converges to a single point in the search space. Upon convergence, the population is re-initialized. The image illustrates that the convergence behavior of CHC is very different from the traditional form of a genetic algorithm.

While it is difficult to argue that CHC is processing schemata, CHC is incrementally solving the problem by solving for subsets of bits at different times during search, as is illustrated by the striping effect in Figure 1.2. If CHC were truly processing schemata, then the image would appear to have continuous stripes that became more defined as the number of generations increased. The behavior of CHC is like a combination between a traditional genetic algorithm and a local search algorithm that climbs to a local optimum and restarts. However, because CHC explores the search space using a population of points rather than a single point, the low order schema relationships can still influence its convergence behavior.

Genetic algorithms, regardless of their specific form, tend to perform a large amount of sampling (i.e., exploration) during the first few generations. This exploration can be thought of as "global" sampling of the search space. For this reason, it is reasonable to associate low order schemata with genetic algorithm behavior. Low order schema relationships are believed to be the driving force behind genetic algorithms – assuming that useful low order schema relationships exist in the optimization problem itself. Intuitively speaking, low order schemata are large subpartitions of the search space. For binary optimization problems, an order-1 **partition** contains two complementary order-1 schemata that bisect the search space, such as `0****` and `1****`. Even when the genetic operators are disruptive from a schema processing perspective, genetic algorithms can still be seen to process low order schemata. This also implies that genetic algorithms are processing information from a globally oriented (coarse-grained) rather than locally oriented (fine-grained) perspective. For this reason, the relationship between genetic algorithm behavior and local problem structure is unclear.

A widely accepted view of local search algorithms is that highly multimodal problems (i.e., problems with many local optima) pose a greater challenge than problems with low multimodality; however, since genetic algorithms process schemata rather than individual points in the population, they tend to be more robust with respect to local optima (Goldberg 1989c). While genetic algorithms may not be vulnerable to all local optima within a single problem, when problems are deemed difficult for a genetic algorithm it is because the genetic algorithm is not reliably converging to the global optimum. As will be shown in this dissertation, when the genetic algorithm is not converging to the global optimum, it is usually converging to *some* suboptimal local optimum, assuming mutation is applied with a low, but non-zero, probability. While the relationships between schemata may be driving the behavior of a genetic algorithm, local optima are also affecting the genetic algorithm's behavior by trapping it during search. The goal of this research is to show that genetic algorithm performance is affected by local optima and thus, to show that problem difficulty for genetic algorithms is, in part, related to the multimodality of the problem.

1.2 Dissertation Overview

The effects of local optima on genetic search will be studied from four perspectives:

1) How encoding and representation affect the number of local optima.

2) How the genetic operators impact the ability of genetic algorithms to escape local optima.

3) How schema relationships and local problem structure jointly affect genetic algorithm behavior.

4) How schema relationships relate to local optima.

Many empirical studies have been performed touting one encoding or specific search operator over another (Caruana and Schaffer 1988; Syswerda 1989; Spears 1992; Whitley and Rana 1997;

Mathias and Whitley 1994). Different encodings and problem representations alter the connectivity of search spaces. This change in connectivity alters the number of local optima in the resulting landscape (Rana and Whitley 1998; Whitley and Rana 1997). In this thesis, results are presented to illustrate that changing the size of the search neighborhood, which is often a side effect of changing a problem representation, reduces the expected number of local optima in discrete optimization problems.

For example, special encoding techniques, such as Gray encodings, tend to maintain or reduce the number of local optima in numeric optimization problems. While the number of local optima alone is not an accurate measure of problem difficulty for genetic algorithms (Horn and Goldberg 1995), empirical results have illustrated that a Gray encoding can often improve genetic algorithm performance (Whitley, Mathias, Rana, and Dzubera 1995; Caruana and Schaffer 1988). The analytical results presented in this thesis can explain why some problems are (or are not) made easier by the choice of a Gray encoding.

Because they process schemata rather than single points, genetic algorithms have been thought to be robust with respect to local optima (Goldberg 1989c); however, in practice, genetic algorithms can still converge to local optima. The choice of genetic operators affects how genetic algorithms sample points in the search space. Crossover is the exploratory operator that allows the genetic algorithm to take "large jumps" during search. In general, the exploratory power of crossover is limited by the mixture of the genetic algorithm population. As the population converges, the exploratory power of crossover diminishes. Even when the population is sufficiently random, different crossover operators have different distributions (biases) of these "large jumps", and these "large jumps" enable genetic algorithms to escape local optima. Not surprisingly, the crossover operators that take the largest "large jumps" have been shown to be very effective search operators (Syswerda 1989; Eshelman and Schaffer 1991). Empirical results are presented to illustrate how local optima affect genetic algorithms and analytical arguments are made to explain why the genetic operators should be expected to move the genetic algorithm population towards highly fit local optima (i.e., high quality local optima with respect to the particular problem being optimized by the genetic algorithm).

Although the existence of highly fit local optima can be problematic for genetic algorithms, the underlying schema relationships in particular functions can also be problematic for genetic algorithms. A case study of genetic algorithms applied to the Boolean Satisfiability (MAXSAT) domain illustrates that both local landscape features, such as local optima and plateaus, and schema relationships can be seen as the cause of problem difficulty for genetic algorithms. This study includes analytical results that allow exact schema averages to be computed for large MAXSAT problems. The schema averages in this class of problems can be related to the local structure of the MAXSAT landscapes. The genetic algorithm convergence behavior is examined to determine whether the genetic algorithm is following the low order schema relationships any more or less than a traditional local search algorithm. Ultimately, the added complexity induced by using a population of points to search rather than a single point does not result in a dramatic difference in the regions explored during search. For this class of optimization problems, the low order schema relationships do not offer an inherent advantage to the genetic algorithm, and the structure of the local surface structure is more effectively exploited by the local search algorithm.

Finally, several problem difficulty measures have been proposed to attempt to statically analyze search spaces to determine when problems will be amenable to genetic search (Whitley, Mathias, and Pyeatt 1995; Jones and Forrest 1995; Weinberger 1990; Manderick, de Weger, and Spiessens 1991). These measures are based on underlying assumptions of how genetic algorithms are searching. Some of these measures are based solely on a local view of the problem landscape, while others rely on schema fitness relationships, which provide a more global view of the problem landscape.

The last portion of this thesis analyzes several different problem difficulty measures for genetic algorithms to assess how well they can predict the convergence behavior of genetic algorithms. The analysis illustrates that the schema based measure provides the most accurate predictions both of problem difficulty and of genetic algorithm convergence behavior. The schema based measure is also shown to predict the existence of highly fit local optima. Specifically, the schema based measure is inherently biased to support highly fit local optima with large basins of attraction. Formal arguments are made which show that the fitness and distance relationship between points in the basin of attraction for a local optimum directly influence the schema fitness relationships. This last analysis establishes that the schema relationships, which are theoretically the driving force behind genetic search, are related to the size and shape of the basins of attraction for local optima.

1.3 Genetic Algorithm Design and Local Optima

When constructing a genetic algorithm, the first step in the design process is to choose a problem representation. Choosing a representation can change neighborhood relationships, which can be thought of as the connectivity between points in the search space. This change directly affects the number of local optima that occur under the new neighborhood structure. The next step in the design process is to choose the genetic operators. There are numerous forms of selection, crossover and mutation operators. Each of the operators samples points in the search space subject to a bias. That bias can limit the amount of exploration that the genetic algorithm performs and, thus, limit the ability of the genetic algorithm to escape from local optima.

1.3.1 Representation, Local Optima and Genetic Algorithms

The first topic that will be explored in this work is the issue of representation for genetic algorithms that are applied to encoded forms of numeric parameter optimization problems. If we have a numeric optimization problem, we can discretize the problem to a desired precision and apply a bit encoding method to each of the evaluation function parameters. The most commonly used forms of bit encodings are a standard Binary encoding method and a Binary Reflected Gray encoding method (Whitley, Mathias, Rana, and Dzubera 1996).

Previously, researchers found that some test problems encoded using a Gray encoding were more easily optimized by genetic algorithms and other local search techniques (Caruana and Schaffer 1988; Mathias and Whitley 1994; Whitley, Mathias, Rana, and Dzubera 1996); thus, the choice of representation has been empirically shown to affect problem difficulty for genetic algorithms. An intuitive reason for this effect is that applying an encoding scheme remaps the entire search space and alters the connectivity of neighboring points. This change in connectivity often affects the multimodality of the resulting search landscape. This modality change helps explain why an algorithm's application to an encoded problem results in poor performance, even though the problem may appear smooth in numeric space. Chapter 3 examines the effects of applying encoding schemes to numeric parameter optimization problems.

1.3.2 Genetic Operators and Local Optima

A second design issue relating to genetic algorithms and local optima is the choice of genetic operators. Three types of genetic operators are selection, crossover and mutation. The selection operator exploits **fitness** information, where fitness measures the relative quality of points in a population. Selection is important because it biases the genetic algorithm to search in regions of higher fitness by allocating more mating opportunities to relatively highly fit strings in the population. However, selection cannot introduce new strings into the population, and therefore, it has no exploratory capabilities. The crossover and mutation operators are the means for exploration.

Crossover operators are designed to exchange and propagate bit patterns between pairs of bit strings. The points sampled due to crossover are restricted by the Hamming distance between pairs of strings. If two parent strings are identical in several bit positions, the offspring strings will also be identical in those bit positions. If two parent strings differ in specific bit positions, then the offspring will also differ in those positions. Crossover can only *exchange* information between parents; its exploratory power depends on the differences between strings in the population. Genetic algorithms that use only crossover converge to a point in the set of local optima that are defined with respect to the specific form of crossover operator (Jones 1995b), while a traditional local search algorithm converges to local optima defined with respect to a single-bit flip neighborhood.

Unlike crossover, mutation is a random operator that arbitrarily flips bits, typically with a low probability. Although mutation is a random operator, it is needed to maintain diversity within a population (Goldberg 1989c; Holland 1975). Mutation is needed to ensure that the genetic algorithm can climb to an optimum while crossover is needed to take large steps in the search space to avoid inferior local optima. The ability of a genetic algorithm to avoid or escape local optima depends on the "large jumps" taken by crossover during search; however, the genetic algorithm must also be able to make local improvements at the single-bit flip level to ensure that the global optimum can be reached.

Although all crossover operators are designed to take potentially large steps in the search space, different forms of crossover contain different biases in how they sample distant points. The analytical results presented in Chapter 4 show the distribution of step-sizes taken by commonly used crossover operators when they are applied to random populations. During the execution of a genetic algorithm, crossover tends to make its largest jumps during the first few generations when the population is most diverse. These distributions model how crossover samples points during the first generation of genetic search and thereby illustrate the exploratory limits of crossover for genetic search. These limits on exploration by crossover, in turn, limit the ability of a genetic algorithm to escape local optima.

Chapter 3 shows that the exploratory power of crossover is limited at best and that the utility of crossover quickly diminishes as search progresses. This observation implies that the convergence behavior of the genetic algorithm is, at some point, governed by the selection and mutation operators. The mutation operator and crossover operator differ in their underlying neighborhood descriptions; therefore, once crossover becomes ineffective at exploring the search space, the mutation operator becomes the primary mechanism for exploration. Assuming that search continues for a reasonable amount of time, the convergence points of the genetic algorithm must be defined with respect to the set of local optima defined under the mutation operator rather than the crossover operator. This work establishes that genetic algorithm behavior is affected by local optima because local optima are the most likely candidates for genetic algorithm convergence points when mutation is used.

1.4 Local Optima and Genetic Algorithm Difficulty

In order to learn more about how genetic algorithms behave as local optimizers, it is useful to study what problem features make search easy or hard for a genetic algorithm. For local search algorithms, the existence of many local optima is problematic. While there is a relationship between local optima and genetic algorithm performance, just counting local optima is insufficient to show that a problem is easy or difficult for the genetic algorithm (Horn and Goldberg 1995) or local search (Tovey 1985).

Deciding whether or not a problem is difficult for a genetic algorithm depends on how genetic algorithms are actually searching. If one believes that a genetic algorithm behaves like a local search algorithm, then perhaps studying features of the local structure of the search space is the best way to measure problem difficulty for genetic algorithms. If one believes that a genetic algorithm is processing schemata, then studying the fitness relationships between competing schemata is the best way to measure problem difficulty for genetic algorithms.

1.4.1 Genetic Algorithm Behavior in the MAXSAT Domain

A case study of genetic algorithm behavior in the MAXSAT domain is described in Chapter 5 to illustrate that both schema information and local surface structure are sources of difficulty for genetic algorithms. This case study is performed using Walsh analysis to examine the types of schema relationships that can exist in this very specific problem domain. Empirical results are presented to substantiate that genetic algorithms do not perform well in this domain. Based on the study, it appears that the genetic algorithm's poor performance is due in part to the misleading schema information, but also to the plateau phenomenon that plagues the landscape of MAXSAT problems. This case study illustrates that schema fitnesses and the plateaus that exist in the MAXSAT landscape are simultaneously affecting search performance.

1.4.2 Problem Difficulty Measures for Genetic Algorithms

To further examine the relationship between local landscape structure and schema fitness, several metrics are examined in Chapter 6 to determine which measure most accurately predicts problem difficulty for genetic algorithms. These metrics are all applied to specific problems to determine whether or not the problem is amenable to genetic search. The approaches taken to characterize problem difficulty often depend on the underlying assumption about how genetic algorithms work.

Two of the measures commonly used for analyzing search spaces are correlation length (Weinberger 1990; Manderick, de Weger, and Spiessens 1991) and fitness distance correlation (Jones and Forrest 1995). These measures rely on relatively local information; neither metric uses explicit schema information in its calculations. A schema based approach to characterizing problem difficulty for genetic algorithms is **static-ϕ** (Whitley, Mathias, and Pyeatt 1995), which is a metric that measures the consistency across partitions of the search space to determine how much support is available to a particular point in the search space. In Chapter 6, static-ϕ is shown empirically to accurately predict convergence points for genetic algorithms by assigning those points a high score. Problems that are difficult for genetic algorithms tend to be problems in which static-ϕ ranks a local (rather than global) optimum most consistent with the schema information.

The theoretical and empirical results in Chapter 6 demonstrate that the amount of consistency between a set of schemata and a local optimum is influenced by the size of the basin of attraction for that local optimum. Static-ϕ attempts to measure the degree of consistency between the fitness rankings of schemata with respect to a particular bit string. When static-ϕ assigns a high score to a particular local optimum, it implies that a local optimum resides in a large, highly fit, basin of attraction. This result illustrates that the global view as seen by the set of consistent schemata is actually indicative of local structure of the search space. Finally, this chapter establishes that the local surface structure of a fitness landscape is an important factor in gauging problem difficulty for genetic algorithms.

1.5 Summary

Genetic algorithms are believed to be less vulnerable to local optima than local search algorithms because they use a collection of points rather than individual data points (Goldberg 1989c). Genetic algorithms' use of multiple points in parallel, rather than a single point, allows them to bypass many local optima that might otherwise slow down traditional local search methods. However, there are many search problems where genetic algorithms do get trapped in a local optimum. Because genetic algorithms are widely believed to use more global information than local information to drive the search, it is unclear how local landscape information can relate to problem difficulty for genetic algorithms.

This research explores several aspects of problem difficulty for genetic algorithms as they pertain to local optima and the shape of the fitness landscape. The first issue addressed in this research is the effect of encoding and representation on the resulting fitness landscape. The choice of an encoding and/or representation affects the connectivity of the search space, which can potentially alter the number of local optima in the resulting landscape. This multimodality can adversely affect the performance of both local search algorithms and genetic algorithms. Secondly, the choice of genetic operators, particularly crossover operators, affects the sizes of search neighborhoods sampled during genetic search. The crossover operators enable the genetic algorithm to take "large steps" throughout the course of search. However, the exploratory ability of crossover wanes as the population converges, which allows the genetic algorithm to become trapped at local optima. The choice of representation and encoding affects the number of local optima and the choice of genetic operators affects the ability of the genetic algorithm to escape local optima.

Understanding what makes search problems difficult for specific search algorithms could potentially be used to customize the algorithm for a particular domain. For instance, if an algorithm is applied to a class of problems in which plateaus are a common occurrence, then the use of special exploratory mechanisms can be used for efficient exploration of plateaus. This research ultimately shows that the highly fit local optima are problematic for genetic algorithms and that the basins of attraction for local optima, particularly highly fit local optima with large basins of attraction, directly influence schema relationships.

Chapter 2

Background and Related Work

A core problem in Computer Science is searching efficiently and effectively. The general problem of search involves methodically looking for a solution to a specific problem. A search **algorithm** is the mechanism that sifts through the possible solutions. The two types of search problems are decision problems and optimization problems (Papadimitriou and Steiglitz 1982). **Decision** problems seek to find a *yes* or *no* answer to a question while **optimization** problems require that we locate the best of many solutions.

As a simple example of a search problem, consider the problem of searching through a sorted list for a particular value. This might be thought of as a decision problem because we need to determine whether or not the number actually occurs in the list. Optimization problems are usually looking for a solution that meets a desired criterion. An optimization analog to the sorted list example is to locate the smallest (or largest) value in the list.

This research considers optimization problems where the goal is to minimize or maximize functions that take l-dimensional bit vectors as their arguments and whose outputs are numeric values. The search algorithms considered in this research are all stochastic search techniques. The two classes of stochastic search algorithms that are studied are single-point search algorithms and genetic algorithms.

2.1 Local Search Algorithms

Three basic components to any single-point search algorithm are:

Neighborhood
: What are the subsets of points immediately reachable from any point?

Generate Operator
: How do we select the next point to visit from the set of neighboring points?

Accept Function
: What is our policy for accepting moves suggested by the generate operator?

For any search algorithm we must define a neighborhood, $N(p)$, around any point p in the search space. Given this neighborhood description, the function $\mathsf{Generate}(p)$ selects a point, q, from the set of neighbors defined by $N(p)$. The $\mathsf{Accept}(q)$ function decides whether or not the move suggested by the $\mathsf{Generate}$ function should be kept. For instance, suppose we have a search algorithm that is attempting to maximize some function f based on a random walk through the search space. This algorithm will be called **Random Walk** search and is defined by:

Step 1: Pick a single starting point p and set $best = p$.

Step 2: Set q to a randomly chosen point in $N(p)$.

Step 3: If $f(q) > f(best)$, $best = q$.

Step 4: Set $p = q$ and repeat Step 2 until stopping criterion met.

Given a starting point p, the random walk has a generate operator that randomly chooses a point q from $N(p)$. All moves suggested by this generate rule are accepted regardless of the difference between $f(p)$ and $f(q)$. The random walk search attempts to locate a maximum by saving the overall best solution encountered during the random walk. The search terminates when a user defined stopping criterion is met. For example, the stopping criterion could be a limit on the number of steps taken or a specific evaluation for the best solution found.

In contrast to the random walk algorithm, the steepest-ascent **local search** algorithm conducts search by iteratively improving a solution until it reaches a **local maximum**. The steps in steepest-ascent local search are:

Step 1: Pick a single starting point p.

Step 2: Set q to the neighbor of p in $N(p)$ with the maximum evaluation.

Step 3: If $f(q) < f(p)$ then terminate otherwise set $p = q$ and repeat from Step 2.

Starting from a single point p, the steepest-ascent local search algorithm evaluates all possible neighbors of p and chooses q such that $f(q)$ has the maximum evaluation of all neighbors. Unlike the random walk algorithm, the generate operator for steepest-ascent local search uses information about the neighbors of p in order to choose the next point to visit during the search. The accept operator in steepest-ascent local search is also based on the evaluation of q. If there are no improving moves within the defined neighborhood around p, then search is terminated otherwise the search continues from the point q.

This local search algorithm terminates search at a **local optimum**. A **local optimum** is a point p at which there are no improving moves available amongst the set of neighbors defined by $N(p)$. Since steepest-ascent local search explores all possible neighbors of p before choosing a new point q and the point q is the best of all neighbors, if q has an evaluation that is worse than the evaluation of p, there are no improving moves left to make subject to the neighborhood N. When the neighborhood N includes all points other than p, the neighborhood is said to be **exact** (Papadimitriou and Steiglitz 1982). Under an exact neighborhood, any local optimum found is also a **global optimum**. Since most neighborhoods are not exact, we often encounter local optima during search. Since steepest-ascent local search is maximizing f, it terminates at a **local maximum**. The minimization analog to steepest-ascent local search is the steepest-descent local search algorithm which terminates at a **local minimum**.

The scope of this research is restricted to optimization problems that take a bit vector as input; the simplest local search neighborhood in that context is a **Hamming distance one**, HD_1, neighborhood, where the size of the HD_1 neighborhood is the length, L, of the bit vector. Therefore, a **local optimum** as it relates to this research is any point at which all single-bit flip neighbors result in a non-improving move.

2.1.1 Variants of Local Search

The strict definition of local search is used in theoretical work to draw conclusions about the complexity of specific classes of search problems (Johnson, Papadimitriou, and Yannakikis 1988; Tovey 1985). However, the single-point search algorithms used in practice are often variations on this strict model of local search. The variations on local search can happen at different phases of the search: generating neighbors, accepting moves and/or changing stopping criterion.

The simplest variant of local search is local search with random restarts (Lin and Kernighan 1973; Johnson, Papadimitriou, and Yannakikis 1988). Rather than terminating the entire search at a local optimum, the search is repeated from a new starting point. Choosing a new starting point allows the search to continue and locate a potentially different local optimum. Each restart can be seen as another run of the local search algorithm, but the result of the search is the best local optimum encountered over all runs of the algorithm. In this case, the actual stopping criterion of the algorithm might be relaxed so that the search continues until a global optimum, assuming that information is available, is reached or until a certain maximum number of evaluations has been performed.

Three other variants of local search are tabu search (Glover 1994), simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983), and Walksat (Selman, Kautz, and Cohen 1993). These algorithms avoid local optima by using heuristic methods that allow non-improving moves to the taken. When these algorithms expand and examine the neighborhood around a single point, the ultimate choice of the neighbor to visit in the next iteration of the search is not necessarily the neighbor with the best evaluation, which allows these algorithms to continue to explore even when local optima are encountered. The use of non-improving moves allows a search algorithm to sample local optima without becoming trapped by local optima. A feature of some simulated annealing algorithms is the use of an adaptive neighborhood rather than a fixed neighborhood; other versions of simulated annealing may take random steps rather than taking *large* jumps. While these meta-heuristics can often improve the performance of search, the performance of all forms of single-point search are adversely affected by the existence of multiple local optima.

2.2 Genetic Algorithms

An alternative to single-point search is a population-based form of search. In his 1975 book, **Adaptation in Natural and Artificial Systems**, John Holland introduced a model for a search algorithm based on evolution. Genetic plans as Holland defined them, later called genetic algorithms (DeJong 1975), utilized a population of individuals represented by bit strings. The algorithm proceeds like a discrete event simulation: given a population at time t, genetic operators are applied to produce a new population at time $t+1$. This step of evolving the population from time t to $t+1$ is called a **generation**. Figure 2.1 illustrates a single generation of a traditional genetic algorithm. The three genetic operators are:

Selection

Choose members of the population who are eligible for mating based on their fitness.

Crossover

Take pairwise combinations of parents and exchange some of their bits to produce new *offspring* strings for the next generation.

Mutation

Randomly flip a small number of bits in the offspring strings.

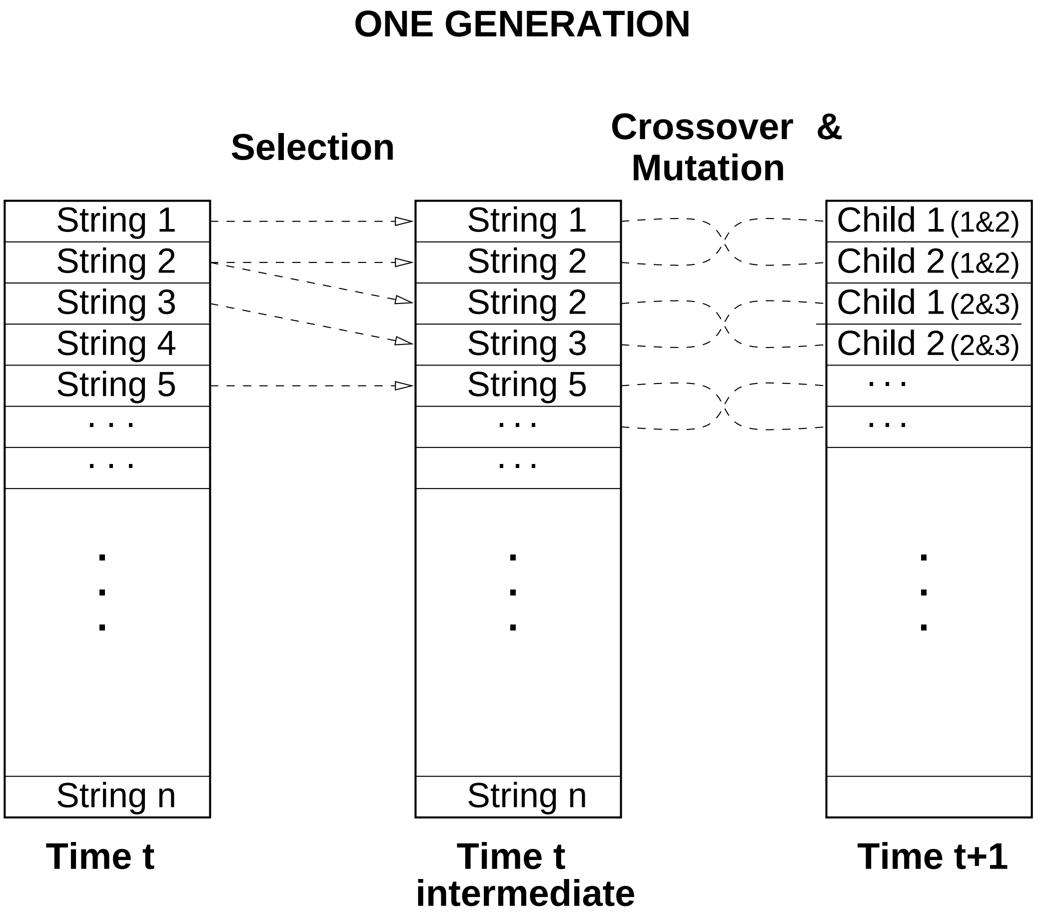


Figure 2.1: A single generation for a traditional genetic algorithm.

The term **fitness** is often used interchangeably with evaluation; however, evaluation is typically well-defined, while fitness is an abstract term. Evaluation of a single point p can be determined using a function f. The fitness of p is based on its evaluation $f(p)$ relative to other points in the search space *or* genetic algorithm population. In the context of genetic algorithm populations, if a point is highly fit, that means that its evaluation is better (i.e. lower if minimizing or higher if maximizing) than other points in the population. In the context of the entire search space, a point is highly fit if its evaluation is better than the majority of the points in the search space. Given a subset of points, if the points in the subset are ranked based on their evaluations, then highly fit points are those ranked the highest in the list.

Selection chooses individuals that will be eligible for mating from a population. Different selection operators can be employed, but the selection scheme used in the canonical genetic algorithm is **fitness proportionate selection**, where the probability that any string in the population is chosen for mating is dependent on its fitness relative to the other strings in the population. If a string has a higher than average fitness (based on the population), then that string has a better than average chance of being selected for mating.

All strings selected for mating are stored in an intermediate population; at this point, only copies of existing strings have been made, and no new strings have been created. The strings in the intermediate population are considered to be **parents** for the next generation. The parent strings in the intermediate population are randomly paired and probabilistically mated to generate

offspring.

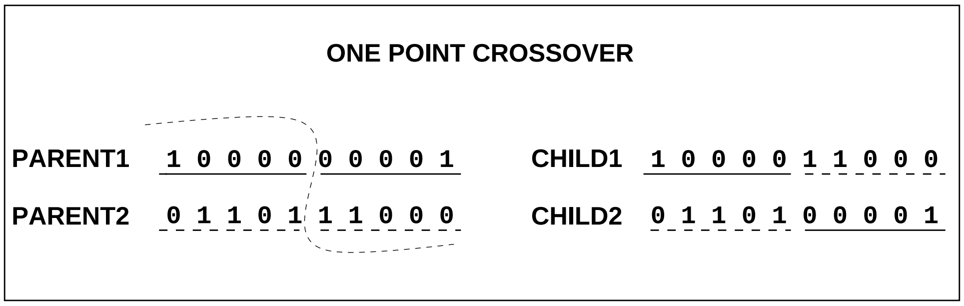


Figure 2.2: An example of one point crossover.

Offspring are generated by probabilistically applying crossover to each pair of parents, meaning that with some probability, p_c, the parent strings may crossover; otherwise, they are passed along to the next generation unchanged. There are several different types of crossover operators. An example of **one-point crossover** is given in Figure 2.2. In this example, the fifth crossover point has been randomly chosen, where a crossover point is any point between adjacent bits. The strings are divided into two segments at the crossover location. The first segment of one parent is combined with the second segment of the other parent to produce a child; crossover simply exchanges segments of both parents to create offspring.

The final operator is mutation, which is applied to each of the offspring independently. With some small probability p_m, each bit in each offspring string can be toggled. In the context of the Generate function, mutation is a random operator that chooses a neighboring point, q, according to a probability density function (p.d.f.) defined by the parameter p_m. This probabilistic neighborhood definition has a nonzero probability that *any* point in the search space can be reached; so in the limit, this neighborhood is *exact*. Thus, mutation guarantees that, given an infinite number of sampling opportunities, the global optimum will be visited infinitely often.

After crossover and mutation have been applied, the next generation of individuals has been produced. The entire process is repeated until some convergence criterion or cutoff criterion (i.e., solution quality, minimum error, or time limit) has been met.

2.2.1 The Schema Theorem

While each genetic operator is simple to understand independently, the resulting behavior of a genetic algorithm can be quite complex. Traditionally, the notion of **schema processing** is used to explain the behavior of a genetic algorithm. Consider the leftmost cube in Figure 2.3. Each corner of the cube is labeled using a three-bit string. Starting with the string `000`, each connection moves to a point that contains only one `1` bit. Any two points are connected if and only if they differ by exactly one bit. A traditional bitwise hillclimber traverses the edges of the hypercube by flipping one bit at a time. At any given time, a single-point search algorithm is sampling only one corner of the hypercube. Since a genetic algorithm uses a population of points, it is simultaneously visiting several corners of the hypercube, which can also be interpreted as sampling different schemata. In this work, hyperplanes and schemata may be used interchangeably.

Schemata are sets of bit strings that share similar characteristics, namely the same values in certain specified bit positions. A schema is described by a concatenation of L values drawn from a trinary alphabet `0,1,*`. Each schema describes a subset of the 2^L possible L length bit strings in the search space. The $*$ indicates a wildcard position, meaning that strings with either `0` or `1`

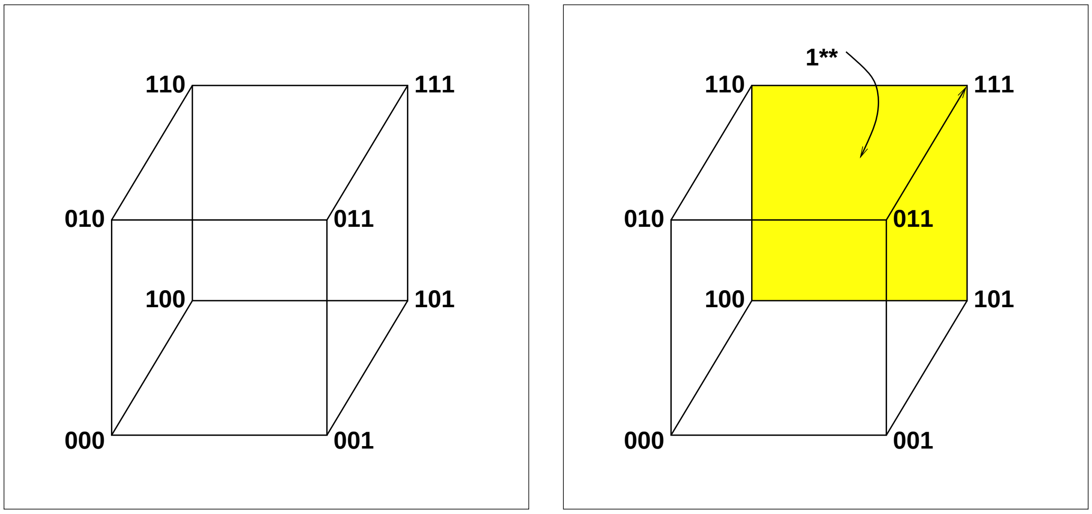


Figure 2.3: A three dimensional hypercube and a highlighted hyperplane.

in that position may be a member of that particular schema. When there is a 0 or 1 in a specific position of the schema, then only strings with a matching value in that position are an element of that schema. For instance, the schema *11* would contain four strings 0110, 0111, 1110, 1111 with all strings in the schema sharing the common bit settings in the second and third bit positions. The **order** of a schema is the number of fixed bit values that schema. The schema *11* is an order-2 schema. The rightmost cube in Figure 2.3 contains a shaded area corresponding to the schema 1**. It is easy to see that the side of the cube opposite to the 1** schema is the 0**. These two schemata together form a **partition** of the search space, which can be represented b** to indicate that the first bit is fixed.

There are 3^L possible schemata and every string is a member of 2^L schemata. For instance, the string 000 is an element of the schemata: 000, 00*, 0*0, *00, 0**, *0*, **0, ***. The schema consisting of all * symbols is usually not considered since it represents the entire search space. Since each schema is a subset of bit strings and we have a fitness function that assigns each bit string a fitness value, the schema can be assigned a fitness by averaging the fitnesses of all member strings.

When we have a population of strings, we are simultaneously sampling or not sampling from all 3^L schemata. Each string in the population is a sample from 2^L schemata; thus, each of the 3^L schemata has some "representation" in the population of a genetic algorithm. As we progress through several generations, we are actually changing the mixture of the population and biasing the population to sample from regions that contain better than average points. Consequently, schemata with higher than average fitnesses *as estimated by the current population* receive more samples in the next generation. The notion that many schemata are being processed simultaneously during one generation is known as **implicit parallelism**.

Holland derived a model for how genetic algorithms process schemata from one time step to the next. This model is known as the **Schema Theorem** and is a key to understanding how a genetic algorithm actually samples schemata.

The Schema Theorem is given by the formula (Schaffer 1987):

$$P(H,t+1) \geq P(H,t)\frac{f(H,t)}{\bar{f}}\left[1 - p_c\frac{\Delta(H)}{L-1}(1 - P(H,t)\frac{f(H,t)}{\bar{f}})\right](1-p_m)^{o(H)}$$

The formula is a lower bound for computing the proportional representation of a single schema in the next population, $P(H, t+1)$ where the function P stands for the proportional representation,

H is the schema, and $t+1$ is the time step. To derive this equation, the effects of each of the genetic operators on the schema H at time t must be modeled. The term:

$$P(H,t)\frac{f(H,t)}{\bar{f}}$$

corresponds to the effects of selection. Fitness proportionate selection is modeled by the current proportional representation of the schema multiplied by its relative fitness, so $f(H,t)$ is the average fitness of schema H and $\bar{f}$ is the average fitness of the population at time t. This term models the proportion of samples from H that will occur in the intermediate population. This number will either increase or decrease based upon crossover or mutation. Since the schema theorem is a lower bound on the representation of a schema in the next generation, consideration is given only to the case where crossover and mutation disrupt schema H.

The ability for one point crossover to disrupt a schema H depends on the number of crossover points that lie between the outermost bits of the schema. The distance between the outermost bits in a schema is called the **defining length** of the schema, which is represented by $\Delta(H)$. For instance, the defining length for the schema `**1*01*` is three because there are three points separating the outermost fixed bits (i.e. the two bits set to `1`). The term:

$$1 - p_c\frac{\Delta(H)}{L-1}(1 - P(H,t)\frac{f(H,t)}{\bar{f}})$$

corresponds to the effects of one point crossover. Crossover can disrupt H in the following way. First, crossover must occur subject to the probability p_c. Second, crossover must occur between a point that samples H and a point that does not sample H. The proportion of strings not sampling H in the intermediate population is simply one minus the proportion of H that is in the intermediate population. Finally, disruption under crossover can occur by picking one of the $L-1$ crossover points that is in between the outermost fixed bits in the schema.

To recap, the proportional representation of H after selection is weighted by the probability that the schema H is *not* disrupted by crossover. This probability is one minus the probability that crossover occurs at a crossover point lying between outermost bits of H with a point not sampling H.

Finally, the last term in the schema theorem:

$$(1-p_m)^{o(H)}$$

accounts for the effects of mutation. Each of the L bits in any string is mutated independently with probability p_m. The probability that mutation does not cause a disruption is simply the probability that each of the fixed bits in the schema are *not* mutated, $(1-p_m)^{o(H)}$, where $o(H)$ is the order of the hyperplane.

The schema theorem as described here models the *worst case* representation of schema H at time $t+1$ because it accounts only for the probability that a crossover event causes a disruption. However, the schema theorem does mathematically illustrate how a schema is allocated samples based upon its fitness in the population and how the schema is resampled through the use of crossover and mutation. In some sense, the schema theorem implies that the genetic algorithm is actually searching the set of schemata to find the single-most fit schema. However, the schema theorem cannot be used directly to predict what kinds of functions will be difficult for a genetic algorithm nor does it mean that all functions will have schema relationships that are amenable to such a search algorithm.

2.2.2 Fixed-Points

Unlike the traditional single-point search algorithms, the genetic algorithm is searching by sampling points in the search space using a population of points. The use of a population makes the convergence behavior of a genetic algorithm much more complex than traditional single-point search algorithms. When a genetic algorithm "converges", it is moving the entire population potentially towards mixture of points rather than a single point in the search space. From a dynamical systems perspective, genetic algorithm convergence behavior can be modeled using **fixed-points** of the infinite population model genetic algorithm (Liepins and Vose 1991; Vose 1993; Vose 1995).

The convergence behavior of a genetic algorithm with an infinite population can be modeled by a Markov chain (Vose 1995). Any genetic algorithm population can be represented by a vector $\vec{x} \in \mathbb{R}^{2^L}$ such that the x_i^{th} entry in the vector indicates the proportional representation of the i^{th} string occurring in the genetic algorithm population. For instance, suppose that there are four strings, $\{\mathtt{00}, \mathtt{01}, \mathtt{10}, \mathtt{11}\}$, occurring the search space. These four bit strings are labeled 0,1,2, and 3 respectively. Then a genetic algorithm population might sample these points in the following proportions: $\{0.75, 0.20, 0.05, 0.00\}$. Note that the sum of all entries of $\vec{x}$ must sum to one.

Let a function $\mathcal{G}$ represent the genetic operators in an infinite population genetic algorithm. Since the genetic algorithm population can be represented by the vector $\vec{x}$, the population at each generation of the genetic algorithm can be represented by a set of vectors $\vec{x}_t$, where t is the generation. The transition from a population at time t to time $t+1$ is performed using the function $\mathcal{G}$ such that $\vec{x}_{t+1} = \mathcal{G}(\vec{x}_t)$ (Vose 1995). A **fixed-point** for an infinite population genetic algorithm is a vector $\vec{x'}$ such that, through the repeated application of $\mathcal{G}$, $x'_t = x'_{t+1}$ as $t \to \infty$ (Vose 1995).

For local search algorithms and other single-point local search algorithms, the potential set of convergence points are restricted to local optima. For a genetic algorithm, the true convergence points are actually particular population mixtures that approximate the fixed-points for the infinite population model genetic algorithm. From a practical standpoint, when genetic algorithms are used for optimization, a single solution is typically the desired result. In an actual finite population genetic algorithm run, only the single "best" solution will be considered. Throughout this dissertation, the term **convergence point** refers to the overall "best" solution found by a finite population genetic algorithm *or* the single point with the highest proportional representation in the infinite population model genetic algorithm. It is only in this context that local optima, as defined by a steepest ascent, Hamming distance one local search neighborhood can be compared to the convergence point of a genetic algorithm.

2.2.3 Genetic Algorithms, Local Search and Operator Neighborhoods

When genetic algorithms are used for optimization, they are often not in the form of a canonical genetic algorithm. The canonical genetic algorithm might be viewed simply as a framework for a population-based search. The genetic algorithms used for optimization are usually modified in some way to improve their performance both in terms of efficiency and effectiveness. For instance, Ulder et al. (1990) introduced an algorithm called Genetic Local Search. This algorithm was essentially the traditional genetic algorithm framework with an additional local search step applied to all members of the population at every generation. The Genetic Local Search algorithm is what is more commonly known as a **hybrid genetic algorithm** meaning it is a genetic algorithm combined with some other algorithm.

Hybrid genetic algorithms are not the only variants of genetic algorithms that have a local search component. The use of elitism, or keeping one or more of the best solutions around from one generation to the next, is a departure from the traditional genetic algorithm. Elitism increases the amount of exploitation over that which would normally occur in the genetic algorithm because it explicitly keeps the most fit member of the population in the new population (DeJong 1992). Perhaps a more extreme view of elitism is that it causes the genetic algorithm to behave more like a steepest ascent local search algorithm by guiding the genetic algorithm in the direction of the best-so-far solution. This bias in the direction of the exploration can serve to drive the genetic algorithm up the hill that contains the best-so-far solution. Furthermore, the use of small populations, high selection pressure, high mutation rates and restarts may work to arrive at a *good* solution quickly, but this is certainly a departure from the canonical model of a genetic algorithm.

When genetic algorithms work in practice as optimizers, they are usually customized to make them behave more like domain-specific hillclimbers. However, it is usually the case that these specialized genetic algorithms do not behave like traditional single-point search algorithms either. It is difficult to apply a schema-based argument for the genetic algorithm performance when the genetic algorithm being applied is vastly different from a canonical genetic algorithm upon which the schema theorem is based. An alternative is to analyze genetic algorithms indirectly by studying the fitness landscape upon which it is searching.

2.2.4 Fitness Landscapes

The genetic algorithm has a neighborhood structure associated with each of the genetic operators, and each of these neighborhood structures induces a separate landscape upon which genetic algorithms search (Culberson 1992; Jones 1995a; Reeves 1994). Rather than studying the relationships between schemata to determine whether or not problems are amenable to genetic search, the landscape can be studied to determine whether or not the genetic operators used actually create a landscape that is amenable to search in general. This approach to studying genetic algorithms distinguishes between the search strategy applied and the landscape being searched, and this distinction allows tests to be performed to determine whether or not the genetic algorithm is an effective search strategy on that specific landscape.

Culberson (1992) introduces the explicit crossover and mutation neighborhoods in graph form. However, the crossover neighborhoods are defined solely for complementary parents. This use of crossover departs from a traditional genetic algorithm perspective where, at least theoretically, any two parents can potentially mate. Culberson's work addresses the question: "Which is more important: crossover or mutation?" His work proves that, subject to some assumptions and algorithm restrictions, search problems that are easily solved using mutation can be transformed into search problems that are easily solved by crossover. Likewise, problems that are difficult for mutation can be transformed to problems that are difficult for crossover.

An interesting observation of Culberson's work was that different local optima are produced by the different neighborhood structures formed by the complementary crossover operator and the single-bit mutation operator applied to a five-bit **ones counting problem**, $\mathsf{onemax} : \mathcal{B}^L \to \mathbb{R}$ where $\mathcal{B} = \{0, 1\}$. The ones counting problem can be posed as the following linear function:

$$\mathsf{onemax}(x) = \sum_{i=1}^{L} x_i$$

where $x \in \mathcal{B}^L$ and x_i references the i^{th} bit in x, and the goal is to maximize the number of 1 bits.

Figure 2.4 is a reproduction of his complementary one point crossover graph. The graph enumerates all possible mating pairs since only complementary pairs are allowed to mate. Since there are 32 strings in a five-bit space, there are 16 pairs of complementary strings. The fitness for each pair is assigned as the maximum number of ones in one of the parents. The edges connect two

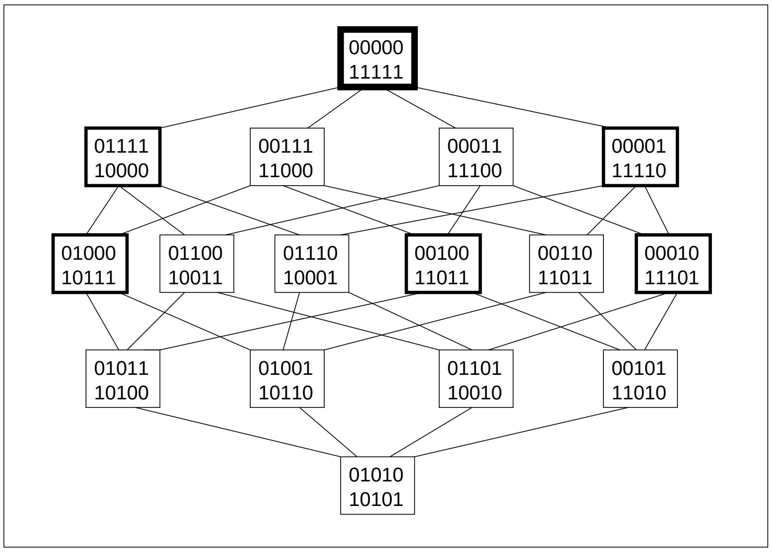


Figure 2.4: Complementary one point crossover graph for a ones counting problem.

complementary pairs that can produce one another under one point crossover. The thickness of the boxes around each pair is an indication of fitness. The topmost and thickest box surrounds the string with the maximum fitness of five. Since these are complementary pairs and fitness is assigned as the maximum number of ones over each string, the minimum fitness is three. Under this crossover operator, there is a local optimum for the string pair: $\{\texttt{00100}, \texttt{11011}\}$. This pair is connected only to pairs of strings with lower fitness. If we consider a single-bit flip neighborhood, then the global optimum is the only peak in that landscape.

Jones (1995b) also studied the neighborhood structures used by the genetic operators, but his work focused on the traditional forms of crossover rather than complementary crossover. His work also emphasized the importance of coupling the neighborhood used by the search operator when discussing landscape features such as local optima. Furthermore, Jones performed numerous comparisons between a *crossover hillclimber* and a genetic algorithm to determine whether or not the choice of operator produced a climbable landscape. These kinds of experiments help to answer the question of whether or not a more expensive, population-based algorithm was really necessary to search under a landscape that included crossover. For test problems designed to be easy for genetic algorithms based upon schema relationships and existence of building blocks, it was usually the case that a hillclimber in a crossover landscape will outperform the genetic algorithm.

2.3 What Makes Search Hard for Genetic Algorithms?

We have seen two perspectives on how genetic algorithms work: they work as schema processors and they work like local search algorithms on a combination of neighborhoods. These two perspectives on genetic algorithm behavior can result in different interpretations of convergence to a local optimum: it is either due to misleading schemata or the landscape induced by the genetic operators. That is not to say that both interpretations cannot be correct simultaneously.

Artificial test problems with specific characteristics have been created, and these problems have been used to study the complex behavior of genetic algorithms. However, the different perspectives on how genetic algorithms work can result in different approaches to creating challenging test problems for genetic algorithms. The first approach to creating test problems, presented in section

2.3.1, is driven by the schema theorem. An alternate approach to creating difficult test problems is to produce highly multimodal functions through the use of function generators that explicitly control for multimodality (section 2.3.2) or tunable ruggedness (section 2.3.3).

2.3.1 Generating Functions Based on Schema Relationships

A strong conjecture about how genetic algorithms behave as optimizers is the **building block hypothesis** (Goldberg 1989c), which suggests that the genetic algorithm combines short, low-order, highly-fit building blocks to produce higher-order, highly fit building blocks. The Royal Road Functions are a form of test function based on the building-block hypothesis (Forrest and Mitchell 1992). Royal Road Functions are designed by choosing an optimum, fully specified, string and dividing it up into a set of low-order schemata (building-blocks). These low order schemata are assigned a fitness value. For instance, the $R1$ function is given by:

$$
\begin{aligned}
&f(11111111**) = 8\\
&f(********11111111**) = 8\\
&f(****************11111111**) = 8\\
&f(************************11111111********************************) = 8\\
&f(********************************11111111************************) = 8\\
&f(**11111111****************) = 8\\
&f(**11111111********) = 8\\
&f(**11111111) = 8
\end{aligned}
$$

Then the evaluation of any string is the summation over the number of building blocks it contains. In this case, the string containing all `1` bits has an evaluation of 64 and is the global maximum.

The Royal Road class of functions explicitly creates the building blocks for the genetic algorithm and, in effect, creates a golden path for the genetic algorithm to follow while making search difficult for simple local search algorithms that do not assemble building blocks. However, the results were surprising in as much as a simple mutation-based hillclimber could consistently outperform the genetic algorithm (Forrest and Mitchell 1992). While these problems were easy for the genetic algorithm, they proved to be easy for a simple algorithm as well.

Just as the Royal Road Functions were designed to be GA-friendly problems, the deceptive functions were designed to be GA-unfriendly. The idea behind deception is to create functions where the low order schemata have average fitnesses that do not support the global optimum (Goldberg 1989c). In other words, the low order schemata with the highest fitnesses do not contain the global optimum. Problems can exhibit varying amounts of deception; there may be only a few schemata with high fitnesses that do not contain the global optimum. One of the most notorious deceptive problems is the **fully deceptive problem** (Whitley 1991), where all schemata less than order L (the length of the bit strings themselves) lead away from the global optimum. Suppose that a three-bit function has a global optimum of `111`. A three-bit fully deceptive problem implies the following schema fitness relationships:

```
f(0**) > f(1**)        f(00*) > f(11*), f(01*), f(10*)
f(*0*) > f(*1*)        f(0*0) > f(1*1), f(0*1), f(1*0)
f(**0) > f(**1)        f(*00) > f(*11), f(*01), f(*10)
```

In this case, the string `000` is called the **deceptive attractor** (Whitley 1991) because all of the schema fitnesses lead towards `000` when the global optimum is `111`. Although deception may be sufficient to make search difficult for genetic algorithms, deception is not a necessary condition for a problem to be difficult for a genetic algorithm (Grefenstette 1992).

2.3.2 Multimodality and Problem Difficulty

Local optima tend to adversely affect the performance of single-point search algorithms. Single-point search algorithms tend to become trapped very easily by local optima and need to have some method for restarting or moving off of the local optimum so search can progress. Since genetic algorithms use a population of points rather than a single-point, they tend to be more robust with respect to local optima (Goldberg 1989c). However, local optima also adversely affect the performance of genetic algorithms. For example, although the traditional deceptive problems are based on schema relationships, they are constructed so that a locally optimal solution is given more support by schemata than a globally optimal solution.

Horn and Goldberg (1995) examine the effects of modality on genetic algorithm performance. They provide two extreme examples using bit represented functions that are unimodal with an exponentially long path and functions that are maximally multimodal. It turns out that the unimodal long path function is difficult for the genetic algorithm while the maximally multimodal function is easy. They go on to describe a methodology for constructing misleading functions with a tunable number of optima. The conclusion from this work is that the number of local optima is not an adequate measure of problem difficulty.

Recently, Spears (1998) used a multimodal function generator to illustrate that the number of local optima can affect the performance of genetic algorithms using recombination, while the performance of the genetic algorithms using mutation alone remains fairly constant regardless of modality. These results do not contradict the conclusions made by Horn and Goldberg, but instead illustrate that multimodality can be problematic for genetic algorithms. While it is clear that local optima themselves can play a role in problem difficulty, more sophisticated analysis of these optimal points is needed to determine more precisely when they do or do not pose a problem.

In both studies, the genetic algorithms use recombination (one point crossover) as the sole means of exploration. A practical genetic algorithm used for optimization will tend to use both crossover and mutation during search, which can dramatically affect the performance. Unfortunately, the problem with drawing conclusions about genetic algorithm behavior, in the absence of mutation, based upon its performance on randomly generated multimodal problems is that the recombination operators induce a different landscape from mutation and should not be compared to a mutation-only search algorithm (Jones 1995b). Since the multimodal problems tested previously are defined in terms of a mutation landscape rather than a crossover landscape, and the algorithms tested do not use mutation in combination with crossover, the relationship between the number of local optima and genetic algorithm performance for practical genetic algorithms is still unclear.

2.3.3 Ruggedness and Problem Difficulty

Autocorrelation has also been used to analyze Kauffman's **NK landscapes** (Kauffman 1989). An NK landscape is a bit-represented landscape of size 2^N that has interactions between N subsets of $K+1$ bits. As K increases, the resulting landscape becomes increasingly complex and, thus, more difficult to optimize. Weinberger (1990) was among the first to examine the use of autocorrelation to characterize these search spaces. The autocorrelation is computed based on a random walk taken over NK landscapes with varying K. As the number of bits of interaction increases, the autocorrelation decreases, which is an indication of increased modality and increased ruggedness.

Manderick (1991) expands this idea and uses it to analyze the behavior of genetic algorithms. He uses a measure of autocorrelation, **correlation length**, to examine the behavior of a genetic algorithm on NK landscapes and the Traveling Salesman problem. This analysis utilizes a correlation length measure that indicates how many steps in the future result in an autocorrelation lower than 0.5. This measure is useful because it provides some information as to what the path length might be for local search. Recently, Hordijk (1996) has built upon the autocorrelation work and has developed a method for predicting the behavior of a random walk in an NK landscape.

The use of autocorrelation to predict problem difficulty is not without its problems. Altenberg (1995) points out that the autocorrelation of a random walk does not provide any information about the likelihood that parents in a genetic algorithm population will produce offspring with *higher* fitness. The landscape that is searched by a genetic algorithm is inherently different than that of a random walk, even when the underlying evaluation function is the same. Fitness is a relative term; the fitness of any string is dependent on the mixture of the population at a given generation. Suppose we have a string, x, at generation one and generation 100. Initially, x may have a high fitness because the population consists of a random sample of points. However, by generation 100, the genetic algorithm may be sampling only points with very high quality evaluations, so the relative fitness of x may not be as high as it was in generation one. For this reason, we can optimize a static evaluation function, but the genetic algorithm is still searching an adaptive fitness landscape. The view of the search space that is seen from a random walk does not necessarily indicate the density of improvements that might be seen in an adapting fitness landscape.

The utility of autocorrelation as a measure of problem difficulty may also be affected by the modality of the landscape itself. Tovey (1985) analytically examines the set of all discrete search problems with the assumption that there is a positive correlation with each point and its neighbors. In his paper, he computes upper bounds on expected path lengths for a steepest ascent strategy (called the Optimal Adjacency algorithm) and a next-ascent strategy (called the Better Adjacency algorithm). The path length is simply the number of steps an algorithm is likely to take before it reaches a local optimum. These expected path lengths are computed for functions that contain only one optimum with respect to the neighborhood definition. The unimodal functions were chosen because they will maximize the correlation between the fitness of a point and neighbors. As the number of optima increases, the path lengths will become shorter, on average; so this value can be seen as an upper bound on the expected path length over the set of all possible discrete functions. However, Tovey points out that the number of local optima is not a reliable indicator of the path lengths of search on specific problem instances. The sizes of the basins of attraction for local optima can widely vary from problem to problem, which will affect the path lengths of hillclimbing search algorithms.

Although Tovey's work used variants of hillclimbing algorithms rather than random walks, the path length of the hillclimbing algorithms, the Better Adjacency algorithm in particular, are related to the autocorrelation. If there are few local optima, then the search space will tend to be smooth and continuous with each optimum having a large basin of attraction. In such an environment, autocorrelation will be high. Conversely, when there are many local optima, basins will tend to be smaller and the autocorrelation of a random walk will be low. Consequently, the autocorrelation measure as a predictor of problem difficulty for genetic algorithms may be a more accurate indicator of the modality of the landscape and expected path lengths for local search algorithms.

2.4 Summary

Genetic algorithms are evolution-inspired algorithms that move through the search space using populations of points rather than a single point. These algorithms are nondeterministic and stochastic and are often applied using very domain-specific forms. Consequently, it is difficult to predict genetic algorithm performance in any general sense.

From an optimization perspective, it is important to know how accurately and how efficiently the genetic algorithm performs compared to other search algorithms applied to the same problem. When the genetic algorithms and less complex local search algorithms are applied to a suite of test problems, the conclusion is often: sometimes the genetic algorithm performs better and sometimes it performs worse than traditional local search.

The question is: Is there a class of problems where genetic algorithms perform well? That question is extremely difficult to answer in general since the types of algorithms that can still be called genetic algorithms can be very different. The majority of the time, the customizations that are made to genetic algorithms are the means for exploiting domain-specific information for a particular problem. Just as local search algorithms can be tailored to work well on one problem class, genetic algorithms can also be customized to perform well on one problem class. But a key to understanding when and why genetic algorithms perform well or perform poorly is to understand what makes search hard for a traditional genetic algorithm that uses mutation and crossover. If we know what makes search hard for the traditional genetic algorithm, then we can have a more solid basis for designing domain specific genetic algorithms that overcome these problems.

Numerous researchers have addressed the question of "What makes search hard for genetic algorithms?" However, the approaches to answering this question often depend on the underlying belief of how genetic algorithms work. There are two perspectives on how genetic algorithms work:

- The schema processing perspective.
- The landscape perspective.

The two views on how genetic algorithms work are actually not so different. Radcliffe (1992) has already shown that any search algorithm, even random search, can be described in terms of schema processing. So it is reasonable to think of genetic algorithms both as local search algorithms and schema processors. The primary goal of this research is to illustrate that the relationships that exists between schemata are strongly related to the local surface structure of the fitness landscape. So, just as local optima are problematic for local search algorithms, they can also be problematic for genetic algorithms.

Chapter 3

Neighborhood Size, Representation and Local Optima

Local optima occur in a variety of forms: peaks, plateaus and ridges (Ginsberg 1993). A peak is a point at which all neighboring points have a higher fitness (i.e., better function evaluation). Plateaus are formed by neighboring points that all share the same evaluation, making the surface appear flat. Ridges occur in multi-dimensional optimization problems when there are dependencies between problem parameters; that is, multiple parameters must be changed simultaneously to generate an improving move (Ginsberg 1993). When a local search algorithm reaches a local optimum, then all neighboring points have an equal or worse evaluation. In the broadest sense, local optima are points where there is no information in the immediate neighborhood to reliably direct search. Thus, these landscape features are known to be the source of problem difficulty for hill-climbing local search algorithms.

When we think of peaks, plateaus and ridges, we typically visualize these objects as one dimensional or two dimensional functions. In doing so, we make assumptions about the connectivity of the search space. If several points form a single peak with a large basin of attraction, then those points must be "neighbors" in order to accurately describe those points as forming a large peak. Consider the eight points in Figure 3.1. Figure 3.1a represents the most intuitive way to connect the eight points: connecting the points moving left to right. This particular method of connecting points creates four local minima (colored gray) and four local maxima (colored black) in the resulting search space. Figure 3.1b represents a second approach to connecting the points in the search space. If we had a search algorithm that maneuvered through the function in Figure 3.1b, then its' view of the landscape would be one peak with a large basin rather than four peaks with small basins. So, are there four peaks or just one? Is it fair to compare the performance of an algorithm that navigates a landscape with four peaks to an algorithm that navigates a landscape with one peak?

3.1 No Free Lunch

Wolpert and Macready's No Free Lunch theorem (1995) proves that no search algorithm is better on average, over the set of all possible functions, than any other search algorithm. While this theorem is simple, it makes it clear (subject to certain assumptions) that there is no general purpose search algorithm that can solve all search problems better than any other algorithm. First, we must assume the optimization problem is discrete; however, this describes all combinatorial optimization problems and really all optimization problems being solved on computers since computers have

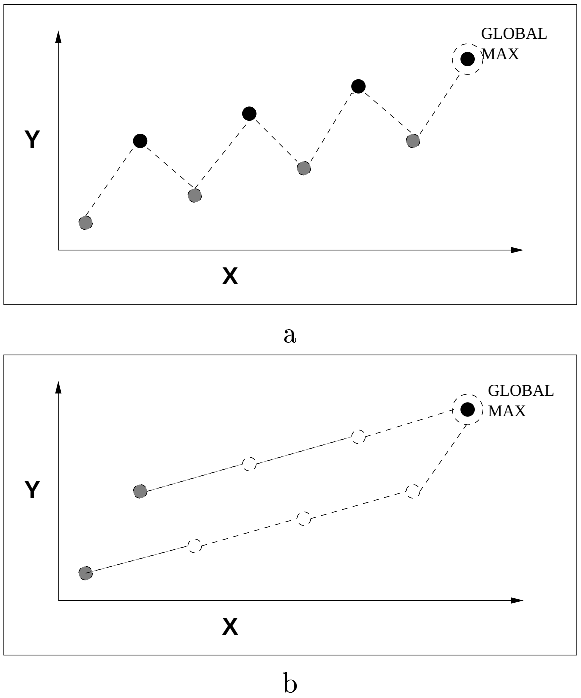


Figure 3.1: Effects of the connectivity of the search space.

finite precision. Second, we must ignore the fact that points may be resampled by an algorithm.

The "No Free Lunch" result can be stated as follows:

> The performance of all possible search algorithms is exactly the same when averaged over all possible functions.

Abstractly, an algorithm can be represented as an ordering, or permutation, over the points in the search space. In this context, the neighborhood definition is considered part of the algorithm itself. In practice, all search algorithms can be considered deterministic; "stochastic" algorithms can be modeled as a stochastic search operator coupled with a specific random seed. Thus, for any particular problem with a fixed search space of size N, an algorithm is just one of N! ordering of points in the search space.

In Figure 3.1, we can think of the solid lines and dashed lines as deterministic paths taken by two different search algorithms. We can also think of the solid lines and dashed lines as the path taken by the same search algorithm, but using different neighborhood definitions. The No Free Lunch theorem also applies to any neighborhood descriptions with the same cardinality (Radcliffe 1991). Given any two neighborhood definitions of size k, then the number of locally optimal points over the set of all functions must be equal under both neighborhood definitions.

This chapter will first present an analytical result that relates the neighborhood size to the number of local optima (Rana and Whitley 1998). Although the no Free Lunch Theorem holds, the total number of local optima possible over the set of all functions varies with k. This result is important for the comparison between traditional local search algorithms and genetic algorithms because comparative studies must be performed using the same form of the test problems, which includes the representation. A common practice in comparative studies of algorithms is to apply different search algorithms to optimization problems using different underlying representations.

Furthermore, even when the same representation is used, such as a bit representation, different encoding schemes might be used for different algorithms. Analytical results will be presented to illustrate how different encodings can affect the number of local optima. Finally, empirical results will be presented to illustrate how different encodings may affect genetic search.

3.2 Neighborhood Size and the Number of Local Optima

The algorithms that will be used throughout this dissertation will be applied to discrete optimization problems defined over $\mathcal{B}^L \rightarrow \mathbb{R}$, where $\mathcal{B} = \{0, 1\}$. When genetic algorithms are applied to numeric optimization problems, encoding methods are often used to remap the search space to a bit representation. Remapping the search space by changing the problem representation affects the connectivity of the space, thereby potentially affecting the number of local optima. This is one reason why changing representation may actually be beneficial: choosing a good representation may improve search performance because the remapped search space is more amenable to genetic search. However, for every search problem that is made easier due to a change in representation, there is another search problem that is made more difficult due to the same change. While specific encodings on specific problem instances may result in an easier or harder landscape, encoding methods that enlarge the neighborhood size categorically reduce the average number of local optima across all possible discrete functions.

This section formally analyzes the effects of neighborhood size on the number of local optima in discrete functions. Given any discrete function of size N, we can compute the average number of optima that will occur over all local search operators using a fixed neighborhood of size k. We can also compute the number of points that remain local optima with a specific probability p.

Suppose we have N unique points in our search space and a search operator that explores k points before making its next move. A point is considered a local optimum from a steepest ascent perspective if its value is better than all of its k-neighbors. Further suppose that the N points in our search space each have unique values. We can sort those points to create a ranking, $R = r_1, r_2, ..., r_n$, in terms of their fitness (where r_1 is the best point in the space and r_n is the worst point in the space). Using this ranking, we can compute the probability that a point ranked in the i-th position in r is a local optimum under an arbitrary representation of the search space (Rana and Whitley 1998).

<u>Theorem</u> 1

The probability, $P(i)$, that a point ranked i-th out of N possible unique fitnesses is a local optimum under any representation resulting in a neighborhood of size k is:

$$P(i) = \frac{\binom{N-i}{k}}{\binom{N-1}{k}} \qquad [1 \leq i \leq (N-k)] \tag{3.1}$$

Proof:

For any point in the search space, there are $\binom{N-1}{k}$ possible neighbors for that point. If the point is ranked in position r_1, then there are $\binom{N-1}{k}$ sets of neighbors that do not contain a point of higher fitness than the point r_1. Therefore, the point ranked in position r_1 will always be a local optimum under all representations of the function. In the general case, a point in position r_i has only $\binom{N-i}{k}$ sets of neighbors that do not contain a point of higher evaluation than the point r_i. Therefore the probability that the point in position r_i remains a local optimum under an arbitrary representation is $\binom{N-i}{k} / \binom{N-1}{k}$.

□

As N increases, the probabilities quickly become difficult to compute. For implementation purposes, it is easier to represent the probabilities as a recurrence relation. The recurrence relation is given as:

$$P(i) = P(i-1)\frac{N-k-(i-1)}{N-(i-1)} \quad \text{where} \quad [2 \leq i \leq (N-k)], \qquad P(1) = 1.0 \tag{3.2}$$

These probabilities enable us to count the number of local optima that should occur over the set of all functions of size N. The formula for computing the expected number of times a particular point will be a local optimum is simply $N! \times P(i)$. Therefore the total number of optima over the set of all functions is:

$$E(N,k) = \sum_{i=1}^{N-k} P(i) \times N! \tag{3.3}$$

If we want to find the expected number of local optima[1] over the set of all functions of size N, we divide $E(N,k)$ by $N!$, which yields:

$$\mu(N,k) = \sum_{i=1}^{N-k} P(i) = \sum_{i=1}^{N-k} \frac{\binom{N-i}{k}}{\binom{N-1}{k}} = \frac{N}{k+1} \tag{3.4}$$

In addition to computing the expected number of local optima for an arbitrary function, we can compute the ranking index i for r_i for any specific probability. This information can be used to indicate how many points are likely to occur as optima over the set of all functions.

Suppose we are interested in knowing which r_i has a probability $P(i) \geq p$. Let $p = \frac{1}{X}$ and assume $P(i) = p$. We now need to solve the following equation:

$$\frac{\binom{N-i}{k}}{\binom{N-1}{k}} = \frac{1}{X}$$

It follows that:

$$X = \frac{\binom{N-1}{k}}{\binom{N-i}{k}} = \prod_{j=1}^{i-1} \frac{N-j}{N-k-j}$$

Note that we can compute bounds for this equation.

$$\left(\frac{N}{N-k}\right)^{(i-1)} \leq X \leq \left(\frac{N-(i-1)}{N-k-(i-1)}\right)^{(i-1)} \tag{3.5}$$

Let the lower bound estimate of i be represented by i_{LB}, which is computed as follows.

$$i_{LB} = \frac{\ln(X)}{\ln(\frac{N}{N-k})} - 1 \tag{3.6}$$

[1] A restricted form of this result has been proven previously by Tovey (1985); however, his derivation was different and assumed that the neighborhood size was k and search space size was always 2^k.

Now compute i_{UB} using the following:

$$i_{UB} = \left(\frac{N - (i_{LB} - 1)}{N - k - (i_{LB} - 1)} \right)^{(i_{LB}-1)} \tag{3.7}$$

Using these bounds, a search between the bounds can quickly produce the exact point i at which $P(i) \geq p$ even for very large search spaces.

3.2.1 Functions with Reoccurring Values

Most real-world functions are not restricted to producing unique function values. So, rather than having a set of unique function evaluations, suppose we now have a multiset of function evaluations (i.e., a set that can contain duplicate elements). The calculations from the previous section can be adapted to the case with duplicate function evaluations.

Once again, we start with the ranking function R where all function evaluations are sorted (with duplicates). Now we create a set of disjoint subsets, G. Each subset $g \in G$ contains indices into R such that for all $i \in g$, $r_i = y$ where y is a particular function evaluation. In other words, all elements of a subset are indexed into R, where all of those points have an equal evaluation. The groups that make up G can then be ordered based on evaluation, with the ranking of elements in g_k being better (less than for a minimization problem) than the evaluation of members of g_{k+i} for all positive integers i. The function G serves as a meta-ranking of the function R.

We must first tackle the problem of computing the probability that a set of points occur as local optima. The function P that was defined in the previous section is a one-to-one function with R. We can use P to define a new set of probabilities $\mathcal{P}$ that can be associated with the subsets in G. Using those new probabilities, we can compute the expected number of optima that will occur over the set of all functions (denoted $\mu(N, k)$). Finally, we can adapt the calculation that a point in a given position in R will occur as a local optimum with probability of at least p.

Case 1: Define locally optimal points as points that are better than and not equal to any of their neighbors (i.e., assume that points that occur on plateaus are not considered locally optimal).

The probability, $\mathcal{P}(k)$, that any point indexed by subset g_k occurs as a local optimum is:

$$\mathcal{P}(k) = P(\max(g_k)) \tag{3.8}$$

where the function $\max(g)$ returns the maximum index contained in the set g. So, $\mu(N, k) = \sum_{k=1}^{m} |g_k| \times \mathcal{P}(k)$ where $m = |G|$.

Now we need to find the index i' such that the point $r_{i'}$ occurs as a local optimum with at least probability p. We already know how to compute the value i where $P(i) \geq p$ when there are no duplicates in R. We can use G and i to compute the new value i' that takes into account the duplicate function evaluations. First, locate g_k such that $i \in g_k$. Then we know the following information about $P(i)$: either $P(i) = \mathcal{P}(k)$ or $P(i) > \mathcal{P}(k)$.

$$i' = \begin{cases} i & if \;\; \mathcal{P}(k) = P(i) \\ \max(g_{k-1}) & otherwise \end{cases} \tag{3.9}$$

Case 2: Define locally optimal to mean that a point must be better than or equal to any of its k-neighbors. This is the case where a plateau is treated as locally optimal.

The probability $\mathcal{P}(k)$ of a subset of points indexed by the subset g_k occurring as local optima is:

$$\mathcal{P}(k) = P(\min(g_k)) \tag{3.10}$$

where the function $\min(g)$ returns the minimum index contained in the set g. The expected number of local optima in an arbitrary representation of the function is still $\mu(N,k) = \sum_{k=1}^{m} |g_k| \times \mathcal{P}(k)$ where $m = |G|$.

Once again we need to find the position i' in R such that the point $r_{i'}$ occurs as a local optimum with at least probability p. Starting with the value i computed assuming no duplicates, we can generate the new index i'. First we locate g_k such that $i \in g_k$. Then we know that $P(i) \leq \mathcal{P}(k)$ by definition. So the new index i' must be:

$$i' = \min(g_k) \tag{3.11}$$

3.3 Genetic Algorithms, Encoding Schemes and Number of Local Optima

The previous sections show that the size of the search neighborhood affects the expected number of local optima in discrete optimization problems. This observation relates to genetic algorithms through the choice of representation for optimization problems. Quite often, optimization problems that are specified using numeric values, either real or integer, are remapped to a bit representation.

For many traditional search algorithms, such as gradient methods, the search optimizes over the numeric space rather than the remapped bit space. In Holland's seminal work on genetic algorithms, problem representation for genetic algorithms is assumed to be a bit representation. The rationale for the use of a bit representation is that the alphabet $\{0,1\}$ is a minimal alphabet and results in a "richer" set of schemata for the genetic algorithm to process (Holland 1992). When a problem is not posed in terms of binary valued variables, some encoding procedure is used to remap the search space to bit strings rather than numeric values. However, by forcing a particular encoding onto a numeric optimization problem, the neighborhood size and connectivity of the space changes.

A common mistake that is made in comparative studies of search algorithms is to apply each algorithm, which may be using its own internal problem representation, to a search problem and conclude that one algorithm is better than another. However, drawing conclusions about algorithm performance based on a study of two algorithms searching over two different problems is of little value. To illustrate the problem, let p refer to the dimensionality of a numeric function. When an algorithm searches in numeric space, it has p parameters to alter at any time step. When that same function is remapped into bit space, the dimensionality of the problem is increased to some $l > p$ based on the number of bits used to encode each parameter. Using this representation, there are l parameters to change at any time step. So, if we assume the algorithms are using fixed neighborhoods defined as a change to a single parameter, then the number of local optima can differ in the two search problems. Furthermore, the number of local optima averaged over all possible discrete functions of size N will be $\frac{N}{l+1} < \frac{N}{p+1}$ by Equation 3.4 since $l > p$. For this reason, all algorithms used in a comparative study should use the same representation.

The use of encoding schemes for optimization by genetic algorithms or other search algorithms implicitly changes the structure of the search space. For this reason, the effects of encoding schemes should be understood before embarking on comparative study of algorithms using different encodings or representations. Since bit encoding schemes increase the dimensionality of the search problem to the length of the encoded string, it may seem that this transformation should be categorically beneficial to any search algorithm because it can reduce the number of local optima appearing in a function. However, keep in mind that increasing the size of the search neighborhood only reduces the *expected* number of local optima over all possible discrete functions of size N. Using an encoding method on specific problem instances may not reduce the number of local optima in the remapped problems. The next section illustrates how two different encoding schemes can affect the modality of the resulting search landscape.

3.3.1 Gray Coding

Suppose we are given a numeric optimization problem and a specific encoding scheme. If the numeric and encoded versions of the function are compared with one another, there may be cases where the numeric version of the function had fewer optima than the bit represented version of the function. So, when we increase the dimensionality of the problem, we actually may introduce optima into the search space for a specific problem. This section will examine two commonly used encoding schemes, Standard Binary encoding and Binary Reflected Gray encoding, to determine how these encoding schemes change the number of local optima in a given discrete numeric optimization problem.

The first encoding method, Standard Binary encoding, is the simplest and most commonly used encoding method. Given a bit string, $x \in \mathcal{B}^L$, label all bits in x as $b_l...b_2b_1$ so that the bit indices range from $1..l$ and the bits are labeled in decreasing order. Then given a specific string x, the Standard Binary decoding of x is:

$$\mathsf{B}(x) = \sum_{k=l}^{1} b_k 2^{k-1}$$

Every bit string can be evaluated as the summation of several offsets that are powers of two. Unlike a Standard Binary encoding, a Binary Reflected Gray encoding is a nonlinear transformation that involves parity calculations. Given x, a Binary Reflected Gray decoding, also referred to as the Degray function, has the form:

$$\mathsf{G}(x) = \sum_{k=l}^{1} -b_k(-1)^{\sum_{i=l}^{k} b_i}(2^k - 1)$$

In this case, the Degray function uses offsets that are conditionally added and subtracted from one another. For instance, decoding `1110` is $\mathsf{G}(\texttt{1111}) = 15 - 7 + 3 - 0 = 11$.

This example illustrates how the Gray and Binary encodings alter the connectivity of numeric functions. Assume a discrete numeric domain zero to $2^L - 1$ for an arbitrary function. Any point in the domain can be represented using an L-bit string. A Gray code is defined as a bit encoding scheme that guarantees that points that are adjacent in the numeric space have a Hamming distance of one. Figure 3.2 illustrates the connectivity between four-bit strings and their decoded numeric values according to a Standard Binary encoding and a Binary Reflected Gray encoding. The lines in the graphs illustrate the Hamming distance one connections that connect points that are adjacent in numeric space. For instance, two and three are adjacent in numeric space and differ by one bit under the Binary encoding as well as the Gray encoding. The Gray encoding, by definition, ensures that there is a walk using single-bit flips that corresponds to a walk in the numeric space. On the other hand, the Binary encoding offers no such guarantee and will *only* preserve the connection between one of the two possible neighbors from the numeric space.

Since Gray encodings preserve the adjacency in numeric space, the upper bound on the number of local optima under a Gray encoding is the number of local optima that occur under a numeric

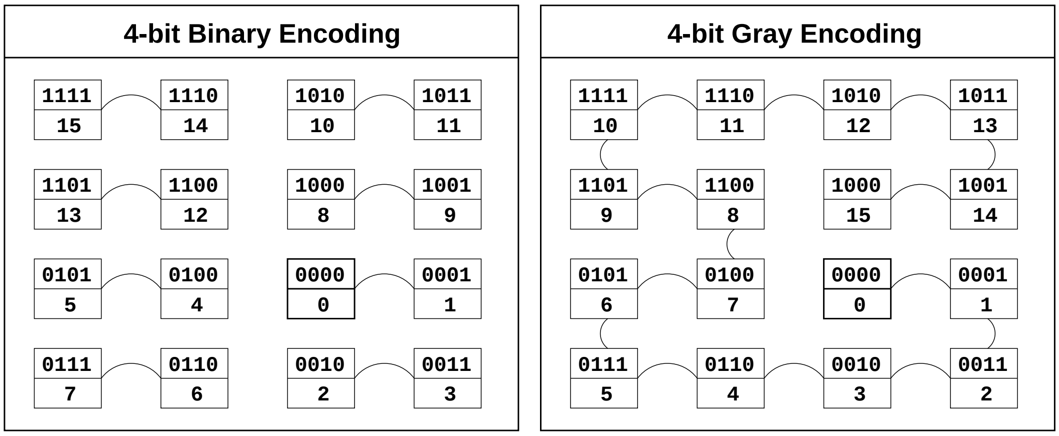


Figure 3.2: A diagram of the connections between points under Gray and Binary.

space. The numeric neighborhood is such that adjacent integers are neighbors and the space wraps around so that zero is adjacent to $2^L - 1$. A value is locally optimal if no neighboring points are better than the current value (for simplicity, assume no duplicates). Suppose we have K locally optimal points in a numeric neighborhood representation of the domain. Since Gray Coding preserves the numeric adjacency of the domain, there are no more than K locally optimal points under Gray encoding. This is referred to as the *Gray Compactness Theorem* (Whitley and Rana 1997):

Theorem 2 *The Gray-Compactness Theorem*
Let X be the number of local optima induced by a Hamming distance one (HD_1) neighborhood search operator over the bits in any Gray coded representation of a function in F. Let Y be the number of local optima induced under the numeric neighborhood topology, where each point has two neighbors which are the points immediately to the left and right of it, of the same function in F, then $X \leq Y$.

A formal proof is given by Whitley and Rana (1997). Simply put, Gray coding results in a function landscape that is *at worst* as multimodal as the landscape induced under a numeric neighborhood search operator. A Binary encoding offers no such guarantee. In fact, for simple functions, Binary encodings usually increase the number of optima compared to the numeric neighborhood. When problems are *highly structured* (i.e., problems containing few optima in the numeric space) then a Gray encoding ensures that the optima that occur in the numeric space will either remain or disappear in the resulting bit representation. Most importantly, the Gray encoding will not introduce new optima into the resulting bit function.

3.3.2 Counting Optima: An Example

Table 3.1 compares Gray and Binary representations over the set of all possible three-bit functions with unique evaluations. First, the set of evaluations for each set of functions can be ranked so that we need only consider all permutations of the function evaluations; thus, enumerating all three-bit functions over the set of ranks is the set of 40,320 permutations of eight indices. This set of permutations, representing the set of all possible numeric functions with $N = 8$, is called F. We partition the set F according to how many local optima occur in the function, where K is the number of optima (column one). The number of functions falling into each partition is given in column two of the table. All of the functions in F are then mapped onto the corresponding Gray and

K	# F with K optima	#Opts in F	# Opts in Gray	# Opts in Binary
1	512	512	512	960
2	14,592	29,184	23,040	27,344
3	23,040	69,120	49,152	49,392
4	2,176	8,704	7,936	2,944
ALL	40,320	107,520	80,640	80,640

Table 3.1: Counting optima over all three-bit functions.

Binary representations. The total number of optima (i.e., minima or maxima) that occur under the bit representations for each partition are given in the fourth and fifth columns. When K is small, the Gray representation consistently produces fewer total optima; however, as K approaches the maximum number of possible optima, the number of optima in the Binary representation quickly drops.

For higher dimensional functions, enumeration quickly becomes impossible. To estimate how the number of optima under Gray and Binary scale with problem size, data was collected for $N = 16$, $N = 32$ and $N = 64$ by randomly sampling 1,000,000 permutations for each problem size. In each case, Gray was better than Binary for all subsets of functions with *exactly* K minima in the numeric neighborhood, when K was less than $\frac{2}{3}MAX$, where $MAX = \frac{N}{2}$. However, when K is between $\frac{2}{3}MAX$ and MAX, Binary begins to dominate in terms of inducing fewer minima in the resulting bit representations than Gray.

Recently, Whitley (1999) has proven that numeric problems with the maximal number of local optima ($\frac{N}{2}$) are a subclass of functions for which there is a "free lunch" with respect to Binary and Gray encodings. For the problems in which there is a maximal number of local optima, Whitley has proven that Binary coalesces more minima than a Standard Binary Reflected Gray encoding. Thus, there is "A Free Lunch" for Binary encodings on the maximally multimodal problems. As a consequence of this result, there is a Free Lunch for the Standard Binary Reflected Gray encoding on all of the remaining functions.

If we assume that the set of functions we are interested in solving have a small, bounded number of optima, then Gray Coding would appear to be the better encoding choice, at least for traditional local search algorithms. The next section empirically examines the effects of choosing a Binary versus a Gray encoding for search for a bit-climbing search algorithm and two forms of genetic algorithms.

3.4 Representation, Hill Climbing and Genetic Algorithm Performance

Horn has previously shown that an extreme example of a maximally multimodal problem was trivial for a genetic algorithm to solve, while a unimodal problem (with an exponentially long path) was found to be difficult (Horn and Goldberg 1995). However, the genetic algorithm that was used in the experiment was a recombination-only genetic algorithm. These extreme and artificial functions simply show that the number of local optima is not always an accurate measure of problem difficulty for genetic algorithms using only recombination. To be thorough, genetic algorithms should be studied on functions that are not extremes and the genetic algorithms should include mutation in combination with crossover.

The following experiments use three search algorithms and several different test functions that were encoded using Binary Reflected Gray encoding and Standard Binary encoding to show the potential advantages and disadvantages associated with each encoding. The problems used as a testbed for the algorithms are numeric parameter optimization problems. One set of problems is a suite of test problems commonly found in the literature (Whitley, Mathias, Rana, and Dzubera 1996), while the other is set of artificially generated, multimodal problems with varying number of local optima. Since Gray and Binary encodings alter the connectivity of the search space and induce or coalesce optima, the behavior of search algorithms can differ on problems encoded in one encoding or the other. The purpose of this experiment is to determine when and why algorithms perform differently under the two encodings.

The algorithms tested include a local search algorithm and two genetic algorithms. Since local search is known to be adversely affected by local optima, and Gray and Binary encodings affect the number of local optima, then one should expect the performance of local search to be better under one encoding or the other. Since genetic algorithms are thought to be more robust with respect to local optima, the performance of the genetic algorithm should be less affected by the number of local optima in the Gray or Binary encoded problems.

3.4.1 Algorithms

Three algorithms were tested on several different test functions. The first algorithm, the Random Bit Climber (RBC) (Davis 1991), serves as a baseline for the experiments. The remaining two algorithms are genetic algorithms: the Simple Genetic Algorithm (SGA) (Goldberg 1989c) and the CHC algorithm (Eshelman and Schaffer 1991). The next three sections describe the algorithms in more detail.

3.4.2 RBC

The simplest algorithm tested is the random bit climber introduced by Davis (1991). Davis' random bit climber (RBC) starts by changing one bit at a time using a random permutation of the bit indices as a priority queue for flipping the bits in the string. The random permutation also serves as an implicit *tabu* mechanism (Glover 1994), which ensures that some time must pass before toggling the same bit twice. When an improvement is found, it is accepted. After the climber has flipped every bit in the string, a new random permutation is generated for testing the bits, and RBC again tests every bit for an improvement. If RBC has checked every bit in the string and no improvement is found, a local optimum has been found, and RBC is restarted from a new random point in the search space.

3.4.3 SGA

A model of the Simple Genetic Algorithm (SGA), as described by Goldberg (1989), is given in Figure 3.3. SGA is a generational algorithm that is similar to the canonical genetic algorithm described by Holland (1975). SGA is typically described using fitness proportionate (roulette wheel) selection and one point crossover; however, the SGA that will be used here uses tournament selection and two point crossover with reduced surrogates. This genetic algorithm begins with a population of strings and applies tournament selection to choose an intermediate population of parents. Tournament selection takes place by randomly choosing some specified (tournament size) number of individuals from the population and then choosing the best member of that set to copy to the intermediate population.

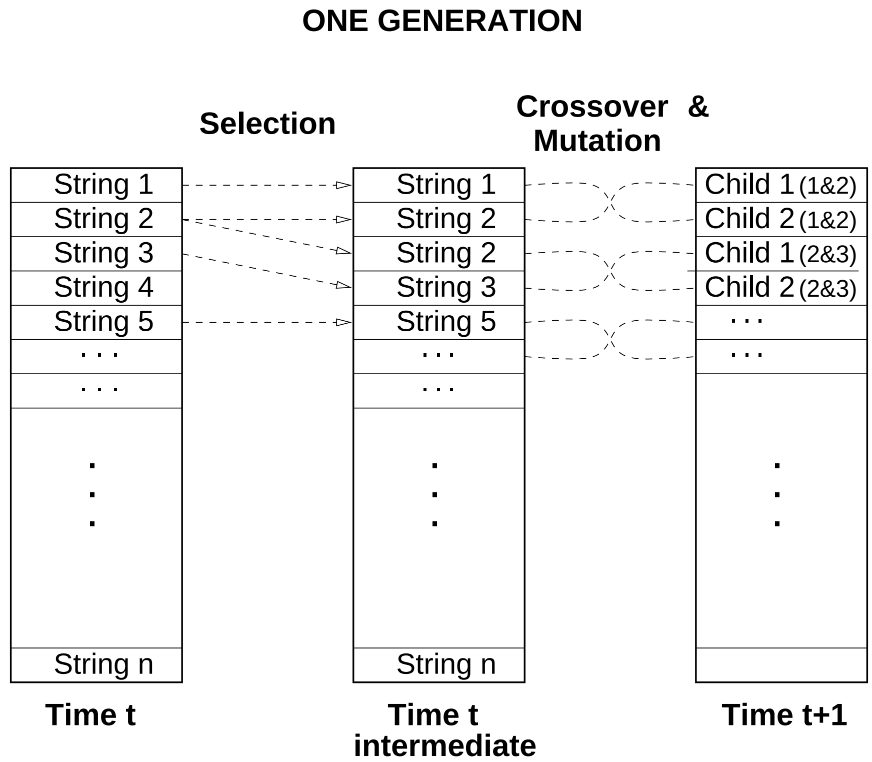


Figure 3.3: A model of the Simple Genetic Algorithm.

```
Parents
1 0 1 1 0 1 1 0 1
0 0 1 0 0 1 0 0 0

Reduced Surrogates

1 - - 1 - - 1 - 1
0 - - 0 - - 0 - 0

Offspring
1 0 1 0 0 1 0 0 1
0 0 1 1 0 1 1 0 0
```

Figure 3.4: An example of two point crossover with reduced surrogates.

After the intermediate population has been chosen, parents are randomly mated using two point crossover with reduced surrogates. Two point crossover is similar to one point crossover except that two crossover points are chosen per parent, the parents are divided into three segments, and the center segment is swapped to produce new strings. In effect, two point crossover treats strings as though they were circular so that the first and the last bit are adjacent to one another. Reduced surrogates means that the crossover points occur where there are differences between the parents.

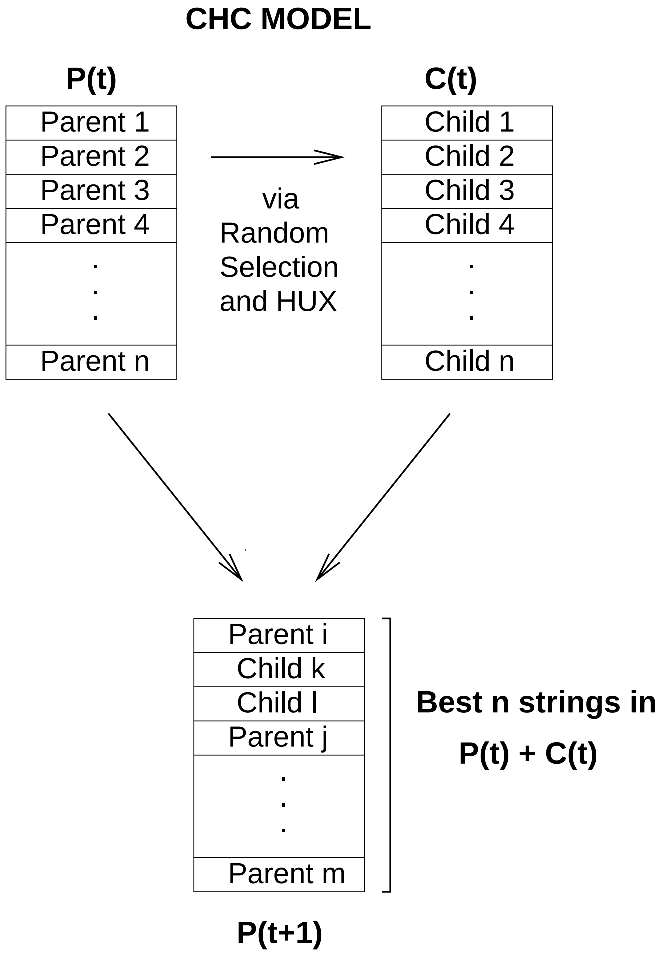


Figure 3.5: A model of CHC.

For example, Figure 3.4 illustrates two point crossover with reduced surrogates. In this example, the two parents differ by only four bits. If the crossover points were chosen between two differing bits (i.e., on either side of a contiguous set of '-' symbols), then the offspring would be identical to the parents. The use of reduced surrogates prevents this from occurring. If parent strings differ by two or more bits, the offspring produced are guaranteed to differ from the parents.

Finally, the offspring are mutated with some small probability before they are inserted into the new population. To maintain the best solution across all generations, *elitism* can be used. Elitism ensures that the best solution from the population at time t is copied into the population at time $t+1$. When elitism is used, the SGA algorithm will be abbreviated as ESGA. The parameter settings used for the (E)SGA were: tournament size of four, mutation rate is $\frac{1}{L}$ (where L is the length of the chromosome), probability of crossover 0.6 and population size is 200.

3.4.4 CHC

The CHC acronym stands for Cross-generational selection, Heterogeneous recombination, and Cataclysmic mutation. The CHC algorithm was developed by Eshelman (1991) and is shown in Figure 3.5. CHC employs a parent population of size μ ($\mu = 50$ for all experiments done here) but instead of selecting highly fit parents for recombination as is typical of most genetic algorithms, the parents

$x_i \in [-5.12, 5.11]$	
Rastrigin	$F(x_i \mid_{i=1,N}) = (N * 10) + \left[\sum_{i=1}^{N} (x_i^2 - 10\cos(2\pi x_i))\right]$
$x_i \in [-512, 511]$	
Schwefel	$F(x_i \mid_{i=1,N}) = \sum_{i=1}^{N} -x_i \sin(\sqrt{\mid x_i \mid})$
Griewangk	$F(x_i \mid_{i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N}(\cos(x_i/\sqrt{i}))$
Powell	$F(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + (\sqrt{5}(x_3 - x_4))^2 + ((x_2 - 2x_3)^2)^2 + (\sqrt{10}(x_1 - x_4)^2)^2$
Whitley	$F(x, y) = -x\sin(\sqrt{\lvert x - \frac{(y+47)}{2}\rvert}) - (y + 47)\sin(\sqrt{\lvert y + 47 + \frac{x}{2}\rvert})$
Rana	$F(x, y) = x\sin(\sqrt{\lvert y + 1 - x\rvert})\cos(\sqrt{\lvert x + y + 1\rvert}) + (y + 1)\cos(\sqrt{\lvert y + 1 - x\rvert})\sin(\sqrt{\lvert x + y + 1\rvert})$

Table 3.2: Test Suite used for studying the effects of representation on GA performance.

are randomly and uniformly paired and conditionally mated to produce λ offspring. The algorithm then chooses the μ best strings from the combined parent and offspring populations as the next generation of reproducing parents. Thus, the CHC algorithm maintains the best μ strings that have been encountered over the course of the search.

Parents are not allowed to mate unless they are sufficiently different as determined by an ever decreasing mating threshold. The crossover operator used by CHC is the HUX operator, where HUX stands for half uniform crossover. The HUX operator ensures that exactly half of the differing bits between parents are exchanged to produce offspring. This operator is extremely disruptive from a schema processing perspective.

CHC does not utilize a mutation operator in the traditional sense, and that fact, coupled with the small population in CHC and cross-generational selection, causes the population to converge quickly. When the population has converged, CHC is partially restarted by copying the best member of the current population into a new population and seeding the rest of the new population with massively mutated (35% of the bits) versions of the best member of the current population.

3.4.5 Test Functions

All three algorithms were run on two different test suites. The first test suite consists of parameter optimization problems that are commonly used as test functions (Whitley, Mathias, Rana, and Dzubera 1996). A second suite of test problems was generated using a test function generator. The test function generator creates functions with a prespecified number of local optima. The next two sections describe both sets of test problems used in the experiments.

3.4.5.1 Test Suite Problems

The test suite functions are shown in Table 3.2. For Rastrigin's function, Schwefel's function and Griewangk's function (Mühlenbein 1991), the dimension of the test problems was 10. For both the Rana and Whitley functions, the Weighted-Wrap expansion method (Whitley, Mathias, Rana, and Dzubera 1996) was used to expand these functions to more than two variables. The wrap method consecutively pairs off the parameters that are input into the function, wrapping around the last parameter. For a three parameter version of a wrapped function, the expansion would be: $EF(x_1, x_2, x_3) = F(x_1, x_2) + F(x_2, x_3) + F(x_3, x_1)$. Random weights were then multiplied to each term before the summation. The set of random weights used in these experiments is given in Appendix A. For both Rana and Whitley's functions, a 10 parameter version of the function was used. All five of these functions used a 10-bit Gray encoded string for each of the 10 parameters. The remaining function is known as the Powell singular function (More, Garbow, and Hillstrom 1981). For these experiments, each of the four parameters in Powell's function was encoded using a 20-bit Gray coded string. Finally, all test suite problems were minimized and the global minima are known *a priori*.

3.4.5.2 Discrete Function Generation

The second set of test problems was artificially generated using a **Discrete Function Generator** (DFG), which creates functions with a prespecified number of local optima in numeric space. Recall that any discrete function can be thought of as a permutation of evaluations. Analogously, any discrete function can also be represented by a permutation of the *rankings* of evaluations. Given a permutation of the rankings, a complete function can be generated by mapping a particular fitness distribution onto the permutation. Dissecting discrete functions by treating them as permutations of rankings allows us to make general observations about the number of local optima expected for classes of functions of a fixed size subject to a fixed size search neighborhood. The treatment of discrete functions as permutations facilitates the random generation of functions with a specified number of local optima.

Given a search problem of size N, assuming that all function evaluations are unique, every permutation of the N function evaluations will produce a valid function each with its own set of local optima. Clearly, there is little control over the problem structure in a random permutation. The purpose of the Discrete Function Generator (DFG) is to produce pseudo-random permutations that yield functions with a specified number of local optima.

The first issue to address is neighborhood size. Local optima cannot be defined without a neighborhood description. The neighborhood assumed in the DFG is a neighborhood of size two and neighbors are the points immediately to the left and right of a point (and the function wraps at the end points). The next two issues to address are how to choose which points are local optima and what shapes for their basins of attraction. To keep the set of parameters to a minimum, the DFG determines the local optima and basin shapes randomly using a uniform pseudo-random number generator.

The DFG algorithm requires two parameters: the number of local optima q and the number of points in the search space N. Given N and q, the DFG algorithm consists of five steps:

Step 1.
Generate a random permutation of values from 1 to N.

Step 2.
Randomly segment the permutation into q segments containing at least two points.

Step 3.
Choose a location within each segment to place the local optimum.

Step 4.
Sort (increasing) all values to the right of and including the local optimum.

Step 5.
Sort (decreasing) all values to the left of and including the local optimum *and* the right endpoint of the neighboring segment.

An optional step is to apply a random offset to the entire function so that an extremum does not always occur in the first position in the permutation.

The algorithm is guaranteed to produce a permutation with exactly q optima. The two sorting steps are the crux of the algorithm. Assume that the first three steps are carried out as described, producing the locations of each optimum and q segments $\mathtt{s} : s_1..s_q$ representing the basins of attraction for each optimum. Each segment is divided into two sections by each local optimum. Applying a sort on just one section of the segment will place the point with the highest rank within that section of the segment as its right endpoint and will place the point with the lowest rank at the local optima position. The second sort is applied to the remaining points in the section, the point currently occupying the local optimum position and the rightmost endpoint from the neighboring segment. This two pass sort will reshuffle the points such that local optimality is guaranteed for exactly q points in the space. The algorithm was verified by generating all possible permutations of size $N = 8$.

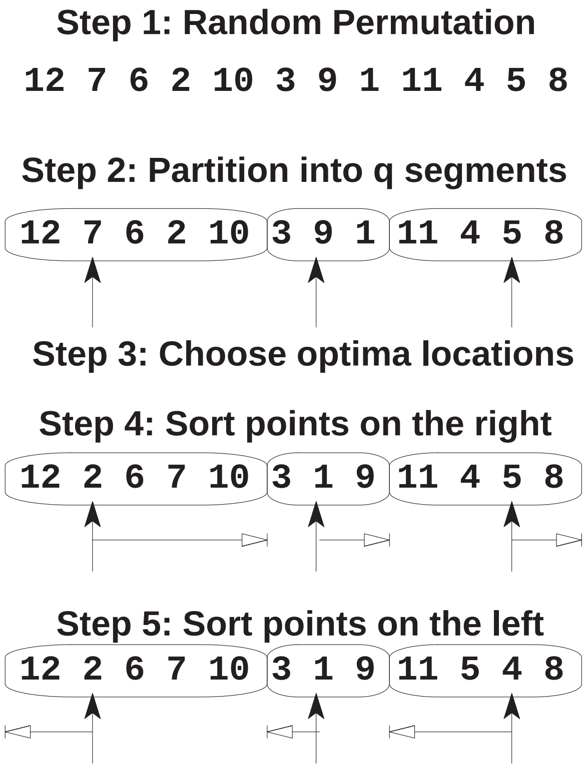


Figure 3.6: Example construction of a discrete function with three optima.

Figure 3.6 is an example of discrete function generation using $N = 12$ and $q = 3$. The algorithm begins by generating a random permutation. Next, the permutation is partitioned into three segments, and the local optima positions within each segment are chosen. A sort (increasing) is applied starting with the local optimum position and extending to the right within each segment. Finally, a sort (decreasing) is applied over all points left of the local optimum position as well as the rightmost endpoint of the neighboring segment. In this example, the local optima are ranked **2**, **1** and **4**. Once the permutation is constructed, a fitness distribution can be generated and mapped onto the permutation. If the problem is intended to be maximized, the fitnesses should be mapped onto the points in decreasing order. If the problem is intended to be minimized, then the fitness values should be mapped onto the points in increasing order.

A suite of one dimensional DFG test problems was generated for 5, 10, 50, 100 and 500 local optima. Additionally, a randomly generated test problem (i.e., a random permutation) was used to baseline the DFG experiments by indicating whether or not the problems with fewer optima were easier or harder to optimize than a completely random function. All algorithms were presented with a *separable* two dimensional version of each problem. Let $f(x)$ represent the discrete function, then the two dimensional version of the function $g(x, y)$ is defined as $g(x, y) = f(x) + f(y)$. Although separability in test problems is not a desirable quality (Whitley, Mathias, Rana, and Dzubera 1996), the separability ensures that the local optima that occur in the two dimensional problem are combinations of the local optima that occur in the one dimensional problem. Furthermore, as will be seen in the results, separability did not make these functions easy for any algorithm tested.

3.5 Results: Gray vs. Binary

The RBC, ESGA and CHC algorithms were run on the DFG test problems and the small set of traditional parameter optimization test problems. One hypothesis of these experiments is to examine whether or not a Binary Reflected Gray encoding makes search easier for the algorithms when compared to a Standard Binary encoding. A hypothesis of this experiment is that genetic algorithm performance depends on the number of local optima.

Several variables were tracked throughout execution. The variables μ_s and σ_s correspond to the mean and the standard deviation of the solutions found over 30 runs of each algorithm. Note that a low solution is best since all problems are being minimized. The variables μ_t and σ_t correspond to the mean and standard deviation of the number of evaluations (trials) taken to solve the problem optimally. The sample size for the μ_t and σ_t variables is the number of problems out of 30 that were solved; this number is listed in the # row. Finally, RBC and CHC each used restart mechanisms. Restarts will occur when the algorithms become trapped in local optima, so the mean number of restarts taken across all 30 runs is given by μ_r. This notation will be used for all results presented.

3.5.1 Algorithm Performance on Test Suite Functions

The first set of results are for the set of six traditional parameter optimization problems. Table 3.3 lists the results for all three algorithms run on the test suite problems. Each algorithm was run 30 times on each problem, and they all sampled a maximum of $100,000$ points. The upper half of the table lists the results for each algorithm run on the Binary encoded versions of the test problems; the lower half of the table lists the results for each algorithm run on the Gray encoded version of the test problems. Within each half some numbers are printed in a bold font, indicating that the value was significantly better than its Gray or Binary (as appropriate) counterpart. For the μ_s and the μ_t comparisons, differences were verified using a One-tailed Student's T-test with $p \leq 0.05$. For the # variable, a Chi-Square test with $p \leq 0.05$ is used to determine whether results are significantly different.

The purpose of this comparison was to determine whether encoding affects the performance of each algorithm. For this reason, pairwise comparisons of the encodings on each problem were made for each algorithm independent of the other algorithms, for a total of 18 comparisons. RBC found significantly lower solutions for Whitley and Powell's functions using Binary encoding, but for Rastrigin and Rana's functions there was no significant difference between the solutions found under Gray or Binary encodings. The solutions found by RBC under a Binary encoding for Schwefel's function were significantly lower than those found using a Gray encoding while Griewangk's function was solved significantly more often using a Gray encoding.

Test Suite Functions – Binary Encoding							
Alg.	Var.	Rast	Schw	Grie	Pow	Whit	Rana
	μ_s	8.1712	**-3852.28**	0.1254	**$1.5x10^{-6}$**	**-785.22**	-430.24
	σ_s	1.8497	99.40	0.0505	$2.3x10^{-6}$	54.23	12.02
RBC	μ_t	–	–	70082	–	–	–
	σ_t	–	–	–	–	–	–
	#	0	0	1	0	0	0
	μ_r	258.86	255.83	249.20	272.70	198.66	209.66
	μ_s	1.4528	-4125.91	0.1839	101388	-790.60	-448.08
	σ_s	1.1256	59.23	0.0562	380326	82.97	14.90
ESGA	μ_t	–	40190	–	33055	–	–
	σ_t	–	30482	–	15153	–	–
	#	0	3	0	3	0	0
	μ_s	0.0072	-4189.83	0.1117	$3.7x10^{-5}$	-927.62	-469.26
	σ_s	0.0288	0.0000	0.0428	$5.1x10^{-5}$	46.60	8.82
CHC	μ_t	–	15262	–	19438	55382	–
	σ_t	–	4978	–	6417	22850	–
	#	0	30	0	4	23	0
	μ_r	15.96	2.70	18.16	19.76	12.93	11.63

Test Suite Functions – Gray Encoding							
Alg.	Var.	Rast	Schw	Grie	Pow	Whit	Rana
	μ_s	8.9770	-3687.85	0.0175	0.0003	-722.29	-432.81
	σ_s	1.7128	137.09	0.0171	$3.9x10^{-5}$	29.47	13.84
RBC	μ_t	–	–	39927	–	–	–
	σ_t	–	–	25638	–	–	–
	#	0	0	**14**	0	0	0
	μ_r	184.23	199.63	122.70	8.6	117.53	96.06
	μ_s	0.0333	-4146.01	**0.1240**	0.0040	-816.07	-448.13
	σ_s	0.1825	72.64	0.0638	0.0206	75.90	18.59
ESGA	μ_t	35219	13742	56799	–	24407	–
	σ_t	13818	8059	–	–	25626	–
	#	**29**	**20**	1	0	**6**	0
	μ_s	0.0000	-4189.83	0.0065	**$1.1x10^{-6}$**	-939.88	**-479.93**
	σ_s	0.0000	0.0000	0.0132	$2.9x10^{-6}$	0.2444	13.26
CHC	μ_t	39834	**8834**	45178	72442	25865	–
	σ_t	19264	2626	22941	16509	12762	–
	#	**30**	30	**24**	3	**30**	0
	μ_r	7.10	1.46	10.23	18.16	5.56	12.73

Table 3.3: Results of running RBC, ESGA and CHC on the Test Suite functions with Binary and Gray encodings.

For the genetic algorithms, the results were biased in favor of a Gray encoding. For Rastrigin and Whitley's functions, both CHC and ESGA found the optimal solution significantly more often using a Gray encoding versus a Binary encoding. For Schwefel's function, ESGA found the solution significantly more often, and CHC found the solution significantly faster using a Gray encoding. ESGA and CHC also performed better on Griewangk's function using a Gray encoding; ESGA had significantly lower solutions while CHC solved the problem significantly more often. For Powell's function, there was no significant difference in solution quality for ESGA under a Gray and Binary encoding while the solutions found for CHC were significantly lower using a Gray encoding. Finally, for Rana's function, there was no significant difference in solution quality for ESGA under a Gray or Binary encoding and CHC found significantly lower solutions under a Gray encoding.

At first glance, these results indicate that the genetic algorithms perform better under a Gray encoding than under a Binary encoding. However, as has been noted previously (Whitley, Mathias, Rana, and Dzubera 1996), there are often biases in test functions. These functions by and large have a very regular structure. The use of trigonometric functions induces local optima on the surface of several of the test functions, but many of the functions still have a global trend that leads to the global optimum.

Figure 3.7 shows three plots of two dimensional versions of test problems. The topmost plot is of Rastrigin's function, the middle plot is Griewangk's function and the lower plot is Rana's function. Rastrigin's function was plotted using the range -5 to 5, which illustrates the general shape of a slice through the numeric search space. This is close to the full range of the function (along one dimension) used for the experiments. This plot shows that Rastrigin's function has a parabolic shape but the surface of the function has been corrugated by the cosine term. Griewangk's function also has parabolic shape with a corrugated surface; however, the range that was used in the plot is not large enough to fully capture the global shape of the function. The surfaces of Rastrigin's function and Griewangk's function are very similar, but the functions are defined over different ranges of values. The only function that was notoriously problematic for all three algorithms was Rana's function. The third plot illustrates that this function does not have the same local surface structure as the other functions. Of course, the range has been limited to a small region in order to visualize the details of the surface. While the surface of Rana's function does not appear to be periodic, as in Rastrigin or Griewangk's functions, the surface is still structured in numeric space. Thus, the preservation of the structure under a Gray encoding is beneficial to genetic algorithms on most of the numeric test problems.

3.5.2 Algorithm Performance on DFG Functions

The regular structure that occurs in the numeric test functions may contribute to the benefits associated with using a Gray encoding. To examine this question more closely, the DFG test functions were created without bias towards any particular global structure. Figure 3.8a is a plot of a DFG test function with 50 local optima. The fitnesses were drawn from a Gaussian distribution and were subsequently scaled between the range of zero to one. Figure 3.8b is a close up of the first $5,000$ points in the function.

Although the pattern in this function seems highly structured in numeric space, this function was generated randomly using the DFG algorithm. It is clear from the pictures that the local optima in this function tend to be highly fit and have similar shaped basins of attraction, but there is no global trend to the function (in numeric space). The reason that the optima are all similar, even though the function was generated randomly, is that there are certain points in the search space that tend to occur as local optima more than other points. The first section of this chapter illustrated that, over the set of all discrete functions, points with a high fitness ranking will tend to

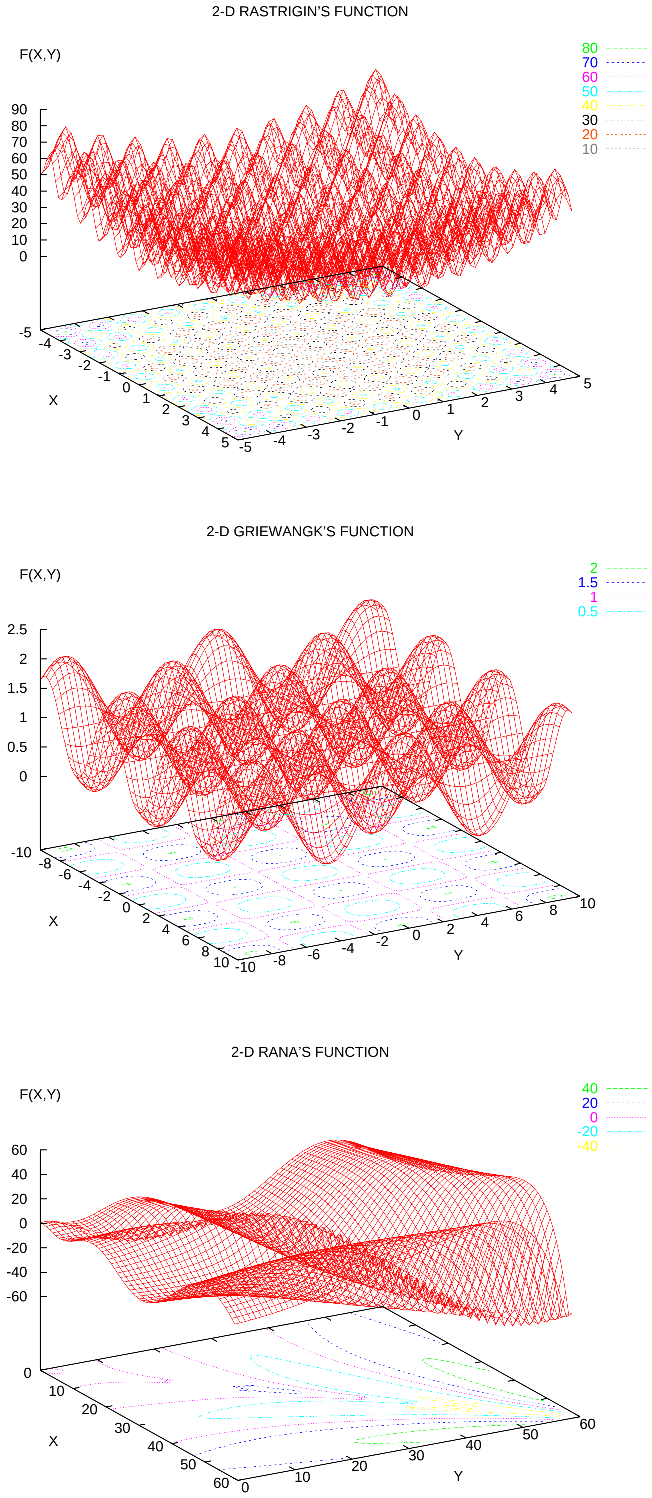
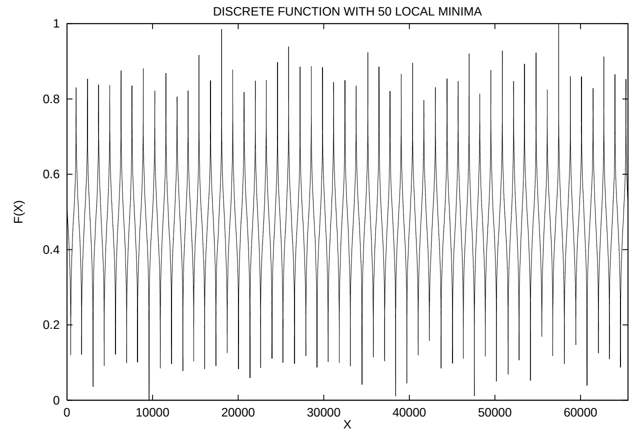
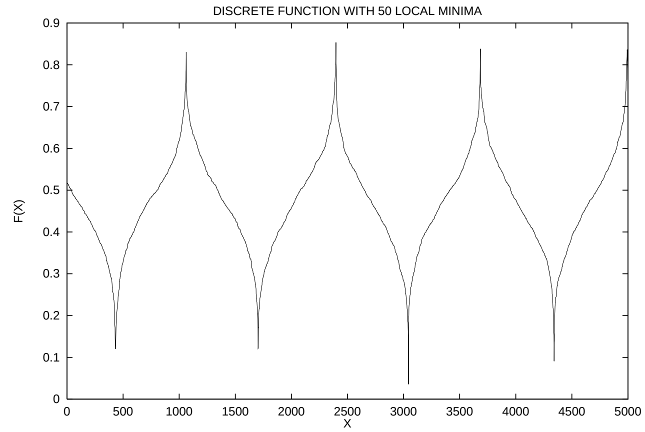


Figure 3.7: Plots for three two dimensional test functions.



(a)



(b)

Figure 3.8: A plot of the test function with 50 optima.

occur as local optima more often than points with a low fitness ranking. The fact that the function was generated from a random permutation of ranks is precisely the reason that the optima seem similar but also lack a global trend.

The experiments were conducted on a suite of six DFG functions with varying number of local optima. Five one dimensional DFG test functions were generated to have 5, 10, 50, 100 and 500 local optima in numeric space. An additional function was generated randomly to provide a baseline. The one dimensional functions were defined over 16-bits so that the size of the problem is $N = 65,536$. Note that, on average, functions with $N = 65,536$ and $k = 16$, have approximately $3,855$ local optima. Table 3.4 lists the number of local optima counted in both Standard Binary and Binary Reflected Gray encodings. The randomly generated function falls very close to the expected number of local optima for problems of that size. As expected, the number of local optima under a Binary encoding is much larger than the number of local optima in numeric space, while the number of local optima under a Gray encoding is very close to the number of local optima in the numeric space.

The Gray encoded functions did not remove many local optima because the functions lack global structure. Figure 3.9 illustrates why some minima can be coalesced and others cannot. Consider the first case in the top image; the function exhibits a global trend towards the global minimum

DFG	5	10	50	100	500	Random
Binary	35	54	233	375	944	3846
Gray	5	10	50	99	476	3847

Table 3.4: Number of local optima of 16-bit DFG functions under Binary and Gray encodings (expected number of optima is 3855 for all functions).

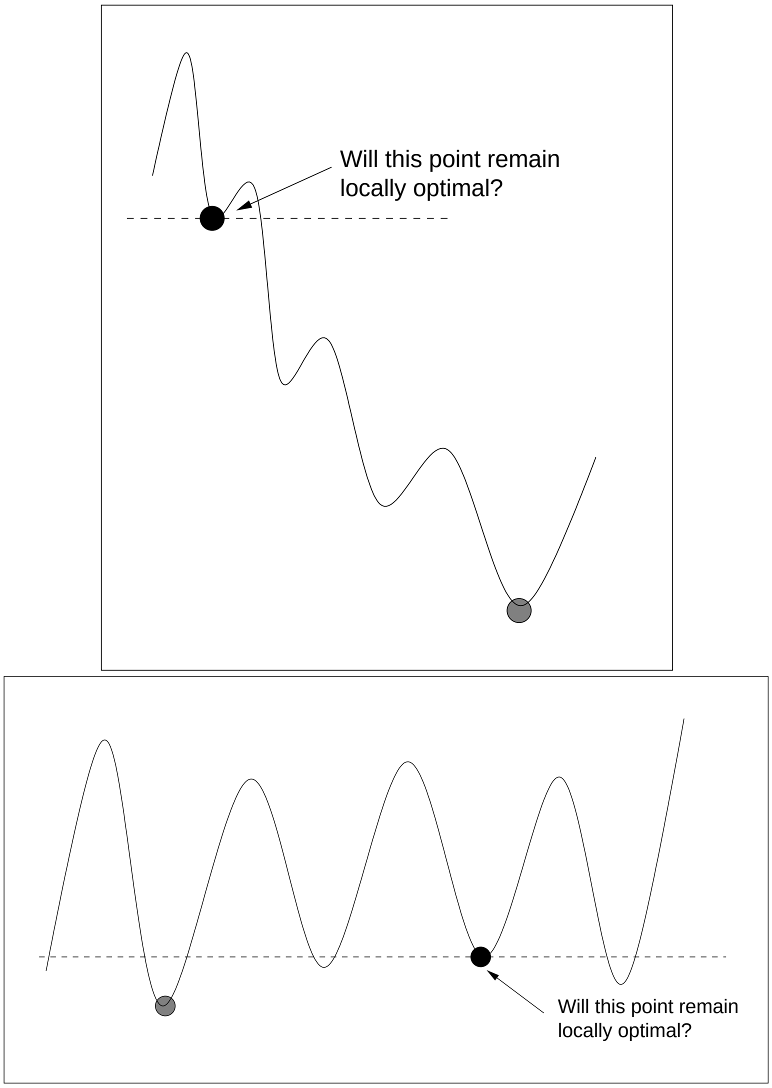


Figure 3.9: Gray encoding and coalescing of minima.

(the gray circle). Now consider the local minimum marked by the black circle. A dotted line was drawn through that point to illustrate how many points lie above and below the local minimum. Any encoding (particularly Gray) will connect that point to L other points. If the fitness of the local optimum is such that it is easy to locate points with higher fitness, regardless of the local structure of the search space, the local optimum is likely to be coalesced due to the encoding.

The second image in Figure 3.9 is more demonstrative of what happens in the DFG functions. DFG functions are generated randomly, and in most cases, they do not have global trends. Using similar notation, the local optimum represented by the black circle may disappear if an encoding is chosen that connects that point to a point of higher fitness (i.e., lower evaluation). In contrast to the previous example, the dotted line that partitions the function based on a fitness threshold has very few points that fall below the line. Consequently, it is unlikely that this optimum, or any of the other optima shown, will coalesce when a new encoding is chosen.

All three algorithms were run on the six DFG test problems. The first set of results presented in Table 3.5 summarize the results of running each algorithm for only $40,000$ evaluations. Recall that the DFG functions were presented to the algorithms as two dimensional optimization problems with 16-bits per parameter ($L = 32$). Again, entries in the table that have been highlighted to indicate significant differences between Gray and Binary encodings. The results for RBC are clearly affected by the number of local optima. The trend is predictable. The DFG problems with 5 and 10 local optima were solved significantly more often by RBC when the problems were encoded using a Gray encoding. RBC applied to the DFG problems with 50 and 100 optima located significantly lower solutions under a Gray encoding. On the DFG function with 500 optima, there was no significant difference between RBC's performance under a Gray or Binary encoding. Finally, RBC run on the random function resulted in significantly lower solutions under Binary than Gray.

While the results for RBC were predictable based on the number of local optima in each problem, the results for both genetic algorithms were a bit surprising. For *all* of the DFG functions, there was no significant difference between the solution quality of ESGA under Gray or Binary encodings. With the exception of the 100-optima DFG, there was no significant difference in solution quality for CHC under either encoding. CHC solved the 100-optima DFG significantly more often using a Gray encoding.

From a schema processing standpoint, the behavior of the genetic algorithms is predictable. There is no global structure in the DFG functions, thus low order schema fitnesses will provide little guidance to the genetic algorithm. From a local optima standpoint, the number of local optima is not a factor in the genetic algorithm performance. The number of local optima under the Binary encoding is much higher than in Gray encoding, yet the solutions found under either encoding are not significantly different. Again, this result can also be attributed to the lack of global structure in the search space. While the functions are smooth, all local optima have similar fitnesses and similar basins of attraction in numeric space. Consequently, when the problem is encoded in a Binary representation, the number of local optima changes, but the local optima are no more or less structured than in the Gray representation. Thus, the genetic algorithms are converging to highly fit local optima because most local optima, regardless of how many occur in the function, are highly fit.

Given that the algorithms were run for only $40,000$ trials, and in many cases were not successful in locating the global optimum, one might argue that the results are an effect of inadequate runtime. To avoid this issue, the experiments were run again but each algorithm was allowed $1,000,000$ evaluations. These results are presented in Table 3.6.

As with the shorter run, RBC behaved predictably. It was successful in solving the problems, and was more successful solving problems with fewer optima. Again, for the first four functions, RBC solved them significantly more often when a Gray encoding was used. For the 500-optima problem, there was no significant difference in the solutions found under a Gray or Binary encoding. Finally, for the Random function, the Binary encoding resulted in significantly better solutions for RBC.

The genetic algorithms behaved consistently as well. With the exception of the 5-optima and 50-optima DFG functions, there were no significant differences between ESGA run with the Gray encoding or Binary encoding. Although CHC was successful much more often given the longer run time, the results were still not significantly different under a Gray or Binary encoding, with the

DFG Functions – Binary Encoding							
Alg.	Var.	5	10	50	100	500	Random
	μ_s	0.0968	0.0999	0.1384	0.1006	0.1501	**0.1571**
	σ_s	0.0458	0.0284	0.0388	0.0319	0.0386	0.0600
RBC	μ_t	23065	–	–	–	–	–
	σ_t	–	–	–	–	–	–
	#	3	0	0	0	0	0
	μ_r	376.90	384.96	412.90	425.53	439.83	486.73
	μ_s	0.1284	0.1422	0.2058	0.1422	0.1822	0.1252
	σ_s	0.0527	0.0681	0.0592	0.0534	0.0484	0.0688
ESGA	μ_t	2853	–	–	–	–	19389
	σ_t	–	–	–	–	–	10078
	#	1	0	0	0	0	2
	μ_s	0.0725	0.0750	0.1018	0.0615	0.1093	0.0977
	σ_s	0.0561	0.0601	0.0366	0.0385	0.0446	0.0545
CHC	μ_t	16822	23089	–	12843	–	39284
	σ_t	10479	4939	–	13280	–	–
	#	6	9	0	4	0	1
	μ_r	12.83	11.13	10.03	9.26	10.53	12.83

DFG Functions – Gray Encoding							
Alg.	Var.	5	10	50	100	500	Random
	μ_s	0.0000	0.0029	**0.0516**	**0.0387**	0.1374	0.1984
	σ_s	0.0000	0.0067	0.0296	0.0161	0.0304	0.0475
RBC	μ_t	4537	13051	21400	21247	–	–
	σ_t	3730	10567	10376	–	–	–
	#	**30**	**25**	3	1	0	0
	μ_r	23.00	96.16	255.73	288.63	353.00	487.36
	μ_s	0.1125	0.1383	0.1960	0.1423	0.1877	0.1077
	σ_s	0.0559	0.0651	0.0588	0.0511	0.0638	0.0513
ESGA	μ_t	1905	–	–	–	–	–
	σ_t	1224	–	–	–	–	–
	#	2	0	0	0	0	0
	μ_s	0.0749	0.0958	0.1013	0.0378	0.1309	0.1164
	σ_s	0.0581	0.0579	0.0365	0.0393	0.0502	0.0604
CHC	μ_t	16330	17524	–	20307	–	–
	σ_t	13812	6231	–	12792	–	–
	#	6	5	0	**10**	0	0
	μ_r	16.06	12.63	10.50	8.10	10.50	13.30

Table 3.5: Results of a short run of RBC, ESGA and CHC on DFG functions with Binary and Gray encodings.

DFG Functions – Binary Encoding							
Alg.	Var.	5	10	50	100	500	Random
	μ_s	0.0134	0.0201	0.0550	0.0332	0.0631	**0.0723**
	σ_s	0.0209	0.0101	0.0251	0.0144	0.0234	0.0331
RBC	μ_t	430725	781062	347835	100474	–	232685
	σ_t	333690	157137	50232	–	–	–
	#	21	3	2	1	0	1
	μ_r	5943.9	9447.2	9871.4	10340	11037	11825
	μ_s	0.1172	0.1401	**0.1519**	0.0974	0.1270	0.0569
	σ_s	0.0638	0.0520	0.0567	0.0497	0.0446	0.0418
ESGA	μ_t	3159	–	–	7048	–	17878
	σ_t	–	–	–	–	–	–
	#	1	0	0	1	0	1
	μ_s	0.0000	0.0000	0.0346	0.0006	0.0249	0.0035
	σ_s	0.0000	0.0000	0.0212	0.0036	0.0123	0.0055
CHC	μ_t	134537	145349	627838	359171	553175	567926
	σ_t	99995	118840	243938	287887	464436	309674
	#	30	30	5	29	3	18
	μ_r	50.50	50.36	243.93	99.50	255.20	234.83

DFG Functions – Gray Encoding							
Alg.	Var.	5	10	50	100	500	Random
	μ_s	0.0000	0.0000	0.0018	0.0061	0.0533	0.0961
	σ_s	0.0000	0.0000	0.0071	0.0071	0.0216	0.0263
RBC	μ_t	4537	18337	208733	383049	–	–
	σ_t	3730	24005	290311	303716	–	–
	#	**30**	**30**	**28**	**17**	0	0
	μ_r	23.00	100.60	2202.5	4775.4	8875.5	12177
	μ_s	**0.0758**	0.1168	0.1868	0.0905	0.1162	0.0758
	σ_s	0.0511	0.0606	0.0598	0.0396	0.0436	0.0383
ESGA	μ_t	29350	3053	–	305443	–	–
	σ_t	3053	819	–	–	–	–
	#	5	3	0	1	0	0
	μ_s	0.0000	0.0000	0.0345	0.0000	0.0263	0.0031
	σ_s	0.0000	0.0000	0.0230	0.0000	0.0171	0.0038
CHC	μ_t	160616	130983	455089	**88423**	423690	510565
	σ_t	156260	87475	348734	78486	176145	243970
	#	30	30	6	30	5	16
	μ_r	78.43	46.80	240.56	22.33	240.06	243.26

Table 3.6: Results of running RBC, ESGA and CHC on the DFG functions with Binary and Gray encodings.

exception of the 100-optima problem in which CHC found the solution significantly faster under a Gray encoding. What is particularly interesting about the CHC results is that it was very successful at locating the global optimum in the Random function under both encoding schemes.

Due to the random placement of local optima and the similarities of local optima in the DFG functions, the schema relationships in the DFG functions were not amenable to genetic search. If the schema relationships in the problems were such that there was a clear path to the global optimum, then both ESGA and CHC would have been able to solve these problems. Instead, given the increased amount of time, ESGA was still unsuccessful at locating the global optimum while RBC and CHC were very successful once the limit on search effort was increased. CHC and RBC's success is related to the number of random restarts that could be performed in the extra time allowed. However, the fact that CHC's performance was not dramatically different under a Gray versus a Binary encoding clearly indicates that the number of local optima under the Gray and Binary encodings was not a factor in its behavior.

While the DFG results support Horn's conjecture that the number of local optima does not necessarily affect genetic algorithm performance, it is still important to consider how the encoding transforms the search space. The numeric parameter optimization problem results illustrate that problems with strong global trends towards the global optimum are better represented in a Gray encoding rather than a Binary encoding. When there are no global trends in the function, then the choice of encoding can be arbitrary, since it will neither reliably help nor hurt the genetic algorithm performance. For this reason, choosing a Gray encoding as a default encoding (in the absence of domain knowledge) is arguably best.

3.6 Summary

This chapter illustrates that increasing the size of the search neighborhood affects the number of local optima. Furthermore, the choice of representation (for parameter optimization problems) can also affect the number of local optima in a search problem. Increasing the size of the search neighborhood is a common side effect of applying encoding techniques to numeric parameter optimization problems. This larger neighborhood can potentially cause the removal of local optima, based on the relative fitness of the optima. If an optimum has low fitness, then increasing the neighborhood size may connect that point with other points of higher fitness. However, depending on the specific encoding method used, the number of local optima might also increase.

Genetic algorithms applied to parameter optimization problems typically use some form of a bit representation, which often requires the use of an encoding method to transform numeric inputs to bit strings. The encoding methods used most frequently are Binary Reflected Gray encoding and a Standard Binary encoding. While the number of local optima is expected to decrease as the search neighborhood increases, Gray encoding is the only encoding method that guarantees that it will not increase the number of local optima. The empirical results on the numeric test function illustrate that this feature of Gray encodings offers an advantage when search problems are highly structured and have useful global trends in the numeric space. The comparative results for Gray and Binary encodings on the DFG functions illustrate that the relative performance of the genetic algorithm cannot be predicted solely by the number of optima in the search problem. However, the bit-climbing local search algorithm tended to behave rather predictably. As the number of local optima increased, the performance of the bit-climbing search algorithm degraded. Although the experimental evidence suggests that, in the absence of global structure, Gray encoding offers no advantage to the genetic algorithm, it also offers no inherent disadvantage.

Although the number of local optima did not appear to be a factor in the DFG experiments, further analysis of the genetic operators is needed to fully explain and understand why the genetic algorithm was seemingly unaffected by the number of local optima. While representation and/or encoding methods govern the number of HD_1 local optima that occur in a search problem, the ability or inability of the genetic algorithm to escape local optima is related to the genetic operators being used. The next chapter considers how the choice of genetic operators can affect how the genetic algorithm samples points in the search space.

Chapter 4

Genetic Operators, Hamming Distance Neighborhoods and Local Optima

Striking a balance between exploration and exploitation is essential for any search algorithm. Selection, crossover, and mutation, the genetic operators, are the means by which genetic algorithms attempt to achieve that balance. Selection is vital to the execution of the genetic algorithm because it culls the population and narrows the focus of search to regions with potentially high fitness. Selection is the mechanism for resampling points in the population. Crossover and mutation are the mechanisms for exploration, but *both* crossover and mutation are not always necessary for exploration. In particular, the merits of crossover for genetic search have often been questioned (Fogel and Atmar 1990; Spears 1992; Eshelman and Schaffer 1993; Jones 1995a). However, in the context of local optima, crossover is the means by which genetic algorithms take controlled "large steps" in the search space; thus, it is the primary mechanism for escaping local optima.

Crossover enables the genetic algorithm to take "large steps", but it is not an effective operator as the sole means of exploration. Typically, mutation is a necessity for traditional genetic algorithms because it maintains diversity in a finite population and also ensures that every point in the search space has a nonzero probability of being visited. Unlike mutation, crossover does not introduce allelic diversity into the population (i.e., the number of zero and one bits in the population at any particular bit position remains constant) and does not always guarantee that every point in the search space can be visited. Furthermore, since crossover requires two parent strings, its exploratory power depends on the differences between those parents. As the population converges, the exploratory power of crossover diminishes. However, even if crossover is taken out of a population-based search paradigm, its exploratory power is still limited by its inherent bias in how points are sampled.

There are two forms of crossover biases: positional and distributional (Eshelman, Caruana, and Schaffer 1989). Positional bias refers to the frequency with which crossover exchanges bits occurring in particular locations on the string. For instance, given the parent strings `01110` and `10001`, the offspring `00000` cannot be produced under one point crossover because the values at both the first and the last bit can never be exchanged concurrently. Distributional bias refers to the number of bits that are swapped under a specific crossover operator. Distributional bias represents the possible *step-sizes* that can be made by crossover. Analytical results for the positional bias distributions for several crossover operators have been previously described by Booker (1992).

This chapter presents mathematical descriptions for the distribution of Hamming distances between parents and offspring for four commonly used crossover operators. These distributions model how "large steps" are made by crossover in the first generation of a genetic algorithm with a randomly initialized population. Furthermore, selection and mutation will also be discussed as they pertain to local optima and genetic algorithm convergence behavior. Finally, the remainder of the chapter will illustrate how genetic algorithms sample points in the search space. This work shows that local optima are sampled more heavily by genetic algorithms than other points in the search space and illustrates that highly fit local optima can potentially trap genetic algorithms.

4.1 Crossover Operators and Local Search Neighborhoods

Genetic algorithms can be viewed as a population-based form of local search (Ulder, Aarts, Bandelt, Laarhoven, and Pesch 1990; Reeves 1994). Local search is an iterative process whereby a point or population is continually modified and evaluated until a goal state, such as convergence to a local optimum, is reached. Since one generation of a genetic algorithm consists of applying selection, crossover and mutation to a population to produce a new population, genetic search fits into the paradigm of local search. Furthermore, all three genetic operators can, in turn, be considered as individual local search operators (Jones 1995a).

Local search operators explore the search space by sampling points from a predefined **neighborhood** (Papadimitriou and Steiglitz 1982). Given a discrete search problem defined over a set of points representing all possible inputs S, a neighborhood N is a mapping:

$$N : S \rightarrow \mathbb{P}(S)$$

where $\mathbb{P}(S)$ is the power set of S. So the neighborhood N is a function that takes a single point, $s \in S$, and generates a subset of points that are neighbors of s. For most local search operators, N is defined to be a small change to a single solution. For instance, given an L-bit string, $x \in \mathcal{B}^L$, $\mathcal{B} = \{0, 1\}$, define:

$$N_1(x) = \left\{ y : y \in \mathcal{B}^L \text{ such that } \mathsf{bc}(x \oplus y) = 1 \right\}$$

where $\mathsf{bc}(x)$ returns the number of 1-bits in the bit string x and $\oplus$ is the exclusive-or operator (i.e., addition modulo two). So, $N_1(x)$ is simply the single-bit flip or the 1-change neighborhood (Papadimitriou and Steiglitz 1982). To define a k-change neighborhood about a single bit string:

$$N_k(x) = \left\{ y : y \in \mathcal{B}^L \text{ such that } \mathsf{bc}(x \oplus y) = k \right\}$$

The local search neighborhood for the mutation operator is typically considered to be the 1-change neighborhood. The local search neighborhood for crossover; however, cannot be defined as a simple k-change neighborhood because crossover neighborhoods must be defined by pairs of input strings (Reeves 1994; Jones 1995a; Culberson 1992). However, a k-change neighborhood can be used to model the step-sizes taken by crossover.

In order to formally describe crossover neighborhoods, additional terminology needs to be introduced. Let $i, j \in \mathcal{B}^L$ and define $i \subseteq j$ to denote i is a **submask** of j to indicate that i has a 0-bit wherever j has a 0-bit and i has a 0-bit or 1-bit wherever j has a 1-bit.

Crossover can be performed by applying a mask m to a string pair x and y (Syswerda 1989). The crossover mask, m, contains a 1-bit where the parent strings x and y should exchange bit values. Consider the following example:

Let $x =$ 11101010, $y =$ 00101000,$m =$ 11110000

Then m indicates the x and y should swap their first four bit values to produce 00101010 and

```
Parents
1 0 1 1 0 1 1 0 1
 * * * * * * * * *
0 0 1 0 0 1 0 0 0

Reduced Surrogates

1 - - 1 - - 1 - 1
   *     *   *   *
0 - - 0 - - 0 - 0
```

Figure 4.1: Sample of two point crossover with reduced surrogates.

11101000, yet x and y would be exchanging bits that they have in common and the amount of *useful* information exchanged during crossover is two bits rather than four.

If the crossover mask m is restricted by the Hamming distance between the parent strings x and y so $m \subseteq x \oplus y$ where the string $\mathsf{d} = x \oplus y$ is called a **Hamming distance mask**, then the offspring can be produced using the exclusive-or operator. So, given $m \subseteq \mathsf{d}$, the possible offspring are $x \oplus m$ and $y \oplus m$. Then the set of all potential offspring for a string pair whose Hamming distance mask is d can be generated for all $m \subseteq \mathsf{d}$, and the resulting distance between parents and offspring is $\mathsf{bc}(m)$. For instance, in the previous example:

$$x = \mathsf{11101010},\ y = \mathsf{00101000},\ m = \mathsf{11110000}$$

However, if m is restricted by the distance between x and y, then $m = \mathsf{11000000}$. The same pair of offspring are produced with (x, y) are:

$$x \oplus m = \mathsf{00101010} \text{ and } y \oplus m = \mathsf{11101000}$$

and the offspring differ from the parents by two bits. Note that Hamming distance between parents and offspring is measured conservatively so that if z is an offspring of (x, y), then the parent-offspring distance is $\min(\mathsf{bc}(x \oplus z), \mathsf{bc}(y \oplus z))$.

Rather than explicitly defining and counting the crossover neighborhoods for specific pairs of strings, the task of counting all possible Hamming distances sampled by a particular crossover operator reduces to counting all possible crossover submasks of all possible Hamming distance masks subject to positional bias constraints. Over the space of all possible pairs of strings, each of the possible 2^L Hamming distance masks will be encountered exactly 2^L times. The set of all possible submasks with $\mathsf{bc}(m) = k$ for a specific Hamming distance mask, $\mathsf{d} \in \mathcal{B}^L$, can be computed using a k-change neighborhood definition over Hamming distance masks:

$$N_k(\mathsf{d}) = \{m : m \subseteq \mathsf{d} \ \text{ such that } \ \mathsf{bc}(m) = k\} \qquad \text{where} \qquad 0 \leq k \leq \mathsf{bc}(\mathsf{d})$$

This general form of the neighborhood of submasks can be redefined for each particular crossover operator in order to select only the submasks that meet the specific positional biases of that crossover operator.

4.1.1 Hamming Distance Neighbors for One point and Two point Crossover

One point crossover and **two point** crossover are two examples of crossover operators with positional biases. For an L-bit string, one point crossover chooses a crossover point, c, somewhere between the first and last bit and creates offspring by concatenating the first c bits from one parent with the remaining bits from the second parent. Two point crossover uses two crossover points to

HD-Mask	One-Pt			Red. Sur			
	Xover Point Locations						
	1	2	3	0	1	2	3
d_0=0000	0	0	0	0			
d_1=0001	0	0	0	0			
d_2=0010	0	0	0	0			
d_3=0011	0	0	1		1		
d_4=0100	0	0	0	0			
d_5=0101	0	1	1		1		
d_6=0110	0	1	0		1		
d_7=0111	0	1	1		1	1	
d_8=1000	0	0	0	0			
d_9=1001	1	1	1		1		
d_{10}=1010	1	1	0		1		
d_{11}=1011	1	1	1		1	1	
d_{12}=1100	1	0	0		1		
d_{13}=1101	1	1	1		1	1	
d_{14}=1110	1	1	0		1	1	
d_{15}=1111	1	2	1		1	2	1

Table 4.1: Enumeration of four-bit space to compute Hamming distance results of one point crossover with and without reduced surrogates.

create a segment that is swapped between the parents. In both cases, a contiguous block of bits must be exchanged between two parents.

Reduced surrogates are a modification of traditional crossover, which ensures there is an equal chance of producing any possible offspring (Booker 1987). Figure 4.1 illustrates the use of reduced surrogates with two point crossover. Normally, two point crossover can select from any of the nine possible crossover points. Reduced surrogates constrain the set of available crossover points by Hamming distance so that a crossover point occurs between a pair of differing bits. In this example, two point crossover must choose two out of four possible crossover points. Reduced surrogates also guarantees that when the parent strings are at least two bits different, the offspring will always differ from the parents under one or two point crossover.

Reduced surrogates restrict the neighborhood of crossover, which directly impacts the count of parent-offspring distances. In Table 4.1, the bit strings represent the Hamming distance masks for all possible parent pairs for a four-bit problem. The table represents the Hamming distance between the parents and offspring if we were to choose any of the possible crossover points for either one point crossover or one point crossover using reduced surrogates. Regardless of the mask d_i, traditional one point crossover always has three potential crossover points. The number of crossover points varies from zero to three for one point crossover with reduced surrogates. The Hamming distances (HD) are enumerated for each possible crossover point. The d_0 represents the case where both parents are identical. For one point crossover, we can mate two identical parents three ways to produce offspring that do not differ from the parent pair. For reduced surrogates, there are no crossover points so the offspring are simply copies of the parents (i.e., there is one way to generate the case of HD_0). Also note that for d_3, there are two ways in which traditional one point crossover can result in no change to the parent strings.

One goal of this chapter is to compute the distributions of Hamming distances between parents and all possible offspring to examine how different crossover operators make "large jumps". The number of available crossover points will affect the shape of the distribution by allowing multiple opportunities to create the same offspring. While this type of calculation can certainly be performed, the use of reduced surrogates simplifies the calculations. It is assumed throughout the remainder of the chapter that crossover is performed using reduced surrogates.

4.1.1.1 Hamming Distance Counts

The function, $N_k(\mathsf{d})$, computes all possible submasks of a particular Hamming distance mask. Every crossover operator that uses reduced surrogates will have a neighborhood defined that is a subset of $N_k(\mathsf{d})$. The subsets are determined by the specific constraints for the crossover operators. Formalizing the constraints for one point and two point crossover requires additional functions to be defined.

The function **pack** is a mapping:

$$\mathsf{pack} : \mathcal{B}^L \times \mathcal{B}^L \rightarrow \mathcal{B}^M \quad \text{where} \quad M \leq L$$

The pack function takes an L-bit string as its first argument and an L-bit mask containing exactly M $\mathsf{1}$-bits as its second argument. The result of pack is an M-bit string extracted from the L-bit input string by the bit mask. So, for example:

$$\mathsf{pack}(\mathsf{011000}, \mathsf{011010}) \Longrightarrow \mathsf{110}$$

Now define two additional functions, **span** and b_i:

$$\mathsf{span} : \mathcal{B}^L \rightarrow \mathbb{N}$$

where span counts the number of bits (inclusive) between the outermost 1 bits in the string. For instance $\mathsf{span}(0010010) = 4$. Also, define the function $b_i : \mathcal{B}^L \rightarrow \{0, 1\}$ to return the value at bit position i in the input string where $b_0(x)$ returns the rightmost bit in the string and $b_{L-1}(x)$ returns the leftmost bit in the string.

The two neighborhood submask descriptions for one point crossover, $N_k^o(M)$, and two point crossover, $N_k^t(M)$ can be expressed as:

$$N_k^o(\mathsf{d}) = \begin{cases} m : m \in \mathsf{d} \qquad \text{such that} \\ \\ \mathsf{bc}(m) = k \;\; \text{and} \\ \mathsf{span}(\mathsf{pack}(m, \mathsf{d})) = k \;\; \text{and} \\ b_0(\mathsf{pack}(m, \mathsf{d})) \neq b_{\mathsf{bc}(\mathsf{d})}(\mathsf{pack}(m, \mathsf{d})) \end{cases} \tag{4.1}$$

$$\text{where} \;\; 1 \leq k \leq \lfloor L/2 \rfloor \;\; \text{and} \;\; 2k \leq \mathsf{bc}(\mathsf{d}) \leq L$$

and

$$N_k^t(\mathsf{d}) = \begin{cases} m : m \in \mathsf{d} \qquad \text{such that} \\ \\ \mathsf{bc}(m) = k \;\; \text{and} \\ \mathsf{span}(\mathsf{pack}(m, \mathsf{d})) = k \end{cases} \tag{4.2}$$

$$\text{where} \;\; 1 \leq k \leq \lfloor L/2 \rfloor \;\; \text{and} \;\; 2k \leq \mathsf{bc}(\mathsf{d}) \leq L$$

The case for string pairs resulting in zero crossover points, namely when $\mathsf{bc}(\mathsf{d}) = 0$ and $\mathsf{bc}(\mathsf{d}) = 1$, are handled separately.

The number of neighbors with a specific Hamming distance, HD_k, can be computed by generating and counting all possible neighbors for each crossover operator. Starting with one point crossover, for $1 \leq k \leq \lfloor L/2 \rfloor$, the only strings that can produce HD_k offspring must differ by at least $2k$ bits. Parent strings, x, y, are considered to be k bits away from an offspring z if $\mathsf{bc}(x \oplus z) = k$ or $\mathsf{bc}(y \oplus z) = k$ and $\mathsf{bc}(x \oplus y) \geq 2k$.

Since we are considering one point crossover, there are only two ways to produce HD_k neighbors when the Hamming distance masks contain at least $2k$ 1-bits: to choose crossover submasks with exactly k 1-bits to the right or left of the crossover point. We can count both cases by summing over all possible strings of Hamming distance $2k$ to L for both cases. When there is a Hamming distance of $2k$, both offspring will differ by k bits on both the left and right so this case should be counted only once.

The formula for computing the total number of HD_k neighbors produced under a one point crossover operator with reduced surrogates is:
When $k = 0$,

$$HD_0 = \binom{L}{0} + \binom{L}{1} = 1 + L$$

when $1 \leq k \leq \lfloor L/2 \rfloor$,

$$HD_k = \left[2 \sum_{i=2k}^{L} \binom{L}{i} \right] - \binom{L}{2k}$$

For two point crossover, the number of HD_0 neighbors is exactly the same as for one point crossover because there are the same number of Hamming distance masks that result in no crossover points. Given a Hamming distance mask, d, with at least $2k$ 1-bits, there are $\mathsf{bc}(\mathsf{d})$ pairs of crossover points that will produce submasks with exactly k 1-bits. Thus the counting formula for the HD_k neighbors for two point crossover with reduced surrogates is:
When $k = 0$,

$$HD_0 = \binom{L}{0} + \binom{L}{1} = 1 + L$$

when $1 \leq k \leq \lfloor L/2 \rfloor$,

$$HD_k = \sum_{i=2k}^{L} i \binom{L}{i}$$

4.1.2 Hamming Distance Neighbors for Uniform Crossover

Two crossover operators that have no positional bias are **uniform** crossover and **HUX** (Eshelman 1991). Uniform crossover can be thought of as n point crossover because each bit that differs between parents can be flipped independently of one another. It is assumed that uniform crossover will also be applied with reduced surrogates. HUX is a variant of uniform crossover that, by definition, toggles exactly half of the differing bits between parents.

The two neighborhood submask descriptions for uniform crossover, $N_k^u(\mathsf{d})$, and HUX, $N_k^h(\mathsf{d})$ can be expressed as:

$$N_k^u(\mathsf{d}) = \{\ m : m \in \mathsf{d} \quad \text{such that} \quad \mathsf{bc}(m) = k\ \} \tag{4.3}$$
$$\text{where} \quad 0 \leq k \leq \lfloor L/2 \rfloor \quad \text{and} \quad 2k \leq \mathsf{bc}(\mathsf{d}) \leq L$$

and

$$N_k^h(\mathsf{d}) = \{\ m : m \in \mathsf{d} \quad \text{such that} \quad \mathsf{bc}(m) = k\ \} \tag{4.4}$$

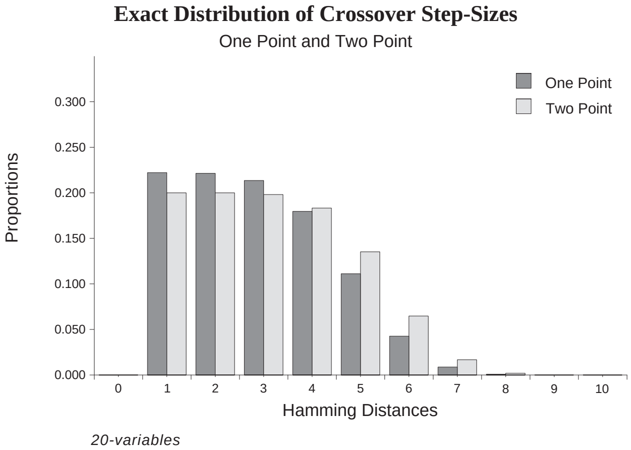


Figure 4.2: Exact sampling distributions for one and two point crossover with reduced surrogates.

$$\text{where} \quad 0 \leq k \leq \lfloor L/2 \rfloor \quad \text{and} \quad 2k \leq \mathsf{bc(d)} \leq 2k+1$$

For uniform crossover, the only way to produce an HD_k neighbor is to have a Hamming distance mask with at least $2k$ 1-bits. Given Hamming distance masks with i 1-bits where $2k \leq i \leq L$, there are $\binom{i}{k}$ ways to generate crossover submasks with k bits set. When the Hamming distance mask has exactly $2k$ bits set, then both parents will produce offspring with a distance of k, so this case should only be counted once.

The formula for counting the possible HD_k neighbors under uniform crossover is:

$$HD_k = \left[2 \sum_{i=2k}^{L} \binom{L}{i}\binom{i}{k}\right] - \binom{L}{2k}\binom{2k}{k}$$

when $0 \leq k \leq \lfloor L/2 \rfloor$.

HUX randomly flips exactly half of the differing bits. In order to produce offspring that are HD_k from a parent under HUX, the parents need to differ by either $2k$ or $2k+1$ (when $k < \lceil L/2 \rceil$) bits.

The formula for computing the number of offspring under HUX is:

when $0 \leq k < \lceil L/2 \rceil$.

$$HD_k = \binom{L}{2k}\binom{2k}{k} + \binom{L}{2k+1}\binom{2k+1}{k}$$

when $k = L/2$ (i.e., k is exactly half of the string length and L is even),

$$HD_k = \binom{L}{k}$$

4.1.3 The Distributions of Hamming Distances for Crossover

Normalizing the counting formulas for Hamming distances zero through $L/2$ forms the distribution of all possible Hamming distance neighbors for crossover. The distributions represent the exact distribution of parent-offspring distances for crossover if all strings are paired with equal probability and also represent the exact distributional biases that exist for this set of four crossover operators.

Figure 4.2 shows the distributions for parent-offspring distances for one and two point crossover using reduced surrogates applied to a 20-bit problem. The values on the horizontal axis represent

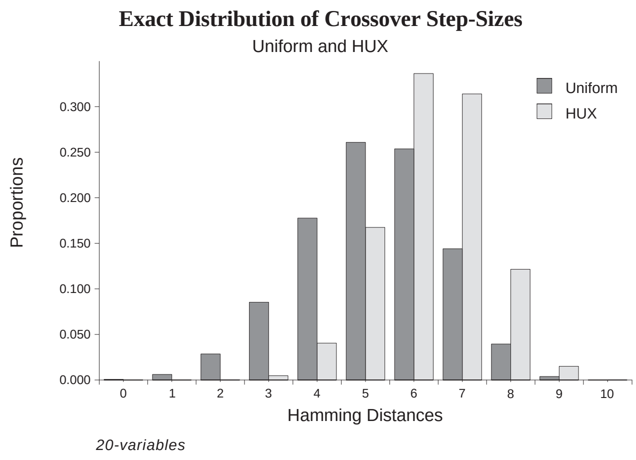


Figure 4.3: Exact sampling distributions for uniform crossover and HUX.

the set of possible Hamming distances. The values on the vertical axis represent the proportion of crossover events resulting in specific parent-offspring distances. Due to the conservative calculation of distance, the maximum Hamming distance between parents and offspring is $\frac{L}{2}$. Note that this distribution is heavily weighted to the left, which means that these two operators are biased to make moves that are of low Hamming distances. In practice, a genetic algorithm would not sample the distribution in this manner. The genetic algorithm will tend to bias the sampling even more heavily towards the lower Hamming distances because selection drives the population to become more uniform. This sampling distribution illustrates that there are practical limits to the exploratory power of crossover operators.

Figure 4.3 shows the distributions for parent-offspring distances for uniform crossover and HUX. In this case, this distribution represents the step-sizes that would be taken by uniform crossover and HUX if they were applied to a population with all strings in equal proportion and if all strings were randomly paired. Note that these distributions produce offspring with much larger Hamming distances (on average) than one or two point crossover.

Larger Hamming distances can affect the amount of exploration done during the course of search. Previously, Syswerda (1989) empirically found uniform crossover to perform better than one point crossover for several optimization problems. The exploratory advantage of uniform crossover over one point crossover is that it has no positional bias and tends to widely sample the neighborhoods defined by the differences between two parent strings. Based on the conclusions of Syswerda's empirical study, the added exploratory power of Uniform crossover is beneficial.

4.2 Example: ESGA applied to NK landscapes

The distributions computed in the previous section assume that all strings are paired with equal probability. While this may seem impractical due to the effects of selection, the exact distributions are a very close approximation to the distribution of parent-offspring distances sampled in the first generation for a Simple Genetic Algorithm (Goldberg 1989c) with a large population. To illustrate how the initial population of a genetic algorithm samples offspring, the distances between parent strings and resulting offspring for an Elitist Simple Genetic Algorithm (ESGA) were tracked during execution. The ESGA had a population size of 500, used elitism and tournament selection with a tournament size of two. The ESGA was run for 250 generations. The tournament size was relatively low and low mutation (p_m=0.0125) was used so the population would maintain some

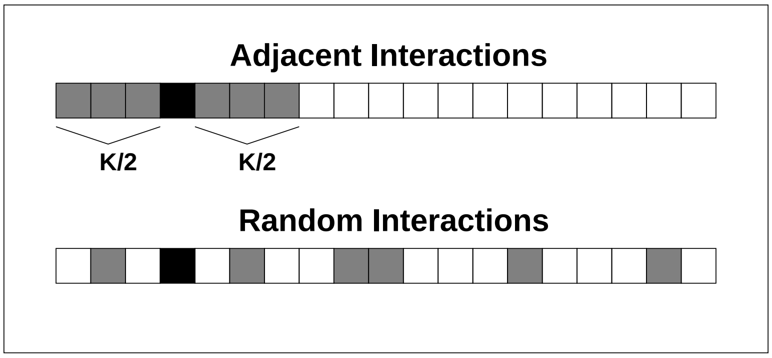


Figure 4.4: An illustration of two types of NK landscapes used in the experiment.

diversity throughout all 250 generations.

The ESGA was studied on several problem instances of two forms of NK landscapes. Kauffman's NK landscapes are a class of problems used in theoretical biology to study rugged fitness landscapes (Kauffman 1993). NK landscapes require two input parameters, N and K. N represents the number of bits and K controls the *epistatic interactions*, or bit interactions, in the problem. For each of the N variables, a set of K distinct variables are chosen to interact with that variable. So, there are N sets of $K+1$ variable combinations constrained so that each variable occurs in at least one set. In order to ensure that the variable combinations interact epistatically, a uniform random fitness between zero and one is assigned to all possible 2^{K+1} values for each set of variable combinations. These $N2^{K+1}$ values are stored in a lookup table. An example of the lookup table for an NK landscape with N=4, K=1 is:

Variables	Settings for v_0:v_1			
v_0: v_1	`0:0`	`0:1`	`1:0`	`1:1`
b_0: b_1	0.1	0.2	**0.5**	0.8
b_1: b_2	0.3	**0.7**	0.2	0.6
b_2: b_0	0.9	0.3	0.7	**0.4**
b_3: b_1	**0.4**	0.6	0.9	0.1

Each row in the table corresponds to a function that is enumerated over all possible values of the variable pairs in the leftmost column. To evaluate an input string, the fitnesses of specific bit combinations are retrieved from the lookup table and averaged together. For example, to evaluate $f(0101)$, we extract the bit combinations of the pairs of bits specified in the lookup table. Since $b_0 = 1$ and $b_1 = 0$, we access the entry in row $b_0 : b_1$ and column `1:0`. We repeat this process for the remaining bits. The four appropriate values are shown in bold in the lookup table. So, $f(0101) = \frac{1}{4}(0.5 + 0.7 + 0.4 + 0.4) = 0.5$.

Two different types of NK landscapes were used to generate test functions for the ESGA. NK landscapes can be generated such that the bit interactions are chosen randomly or fixed so that interactions occur at adjacent bits. Figure 4.4 illustrates the two types of bit interactions for an $N = 20$ and $K = 6$ NK landscape. A fixed NK landscape will choose $\frac{K}{2}$ adjacent neighbors on either side of the bit position of interest. When K is odd, the extra interaction will be randomly placed on the end of the left or right blocks. This method ensures that each bit in the string will fall into a $K+1$ contiguous block of interacting bits. The alternative approach is to choose, without duplicates, all bit interactions at random. Since the set of four crossover operators have different forms of positional biases, the method for choosing bit interactions can potentially alter the relative performance of crossover. The experiments were performed on two sets of 100 instances

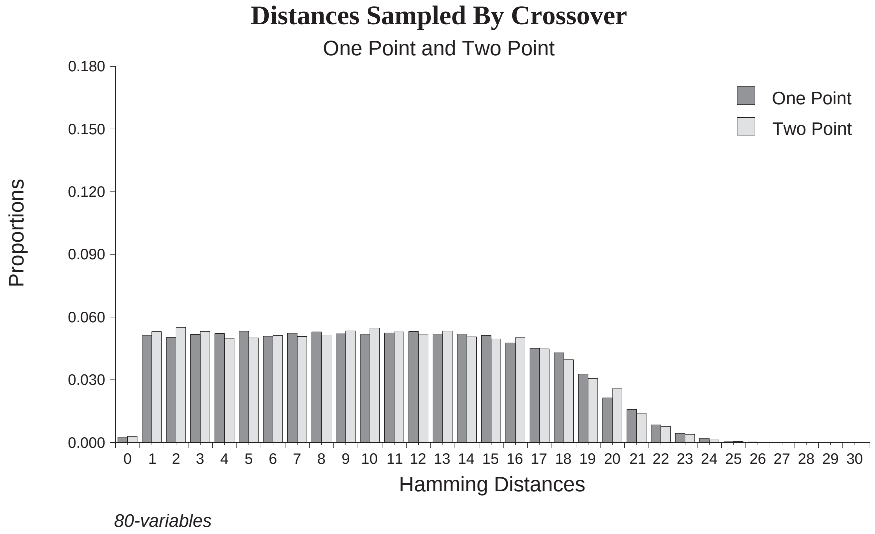


Figure 4.5: Distribution of step-sizes taken by one and two point crossover in the first generation of an ESGA.

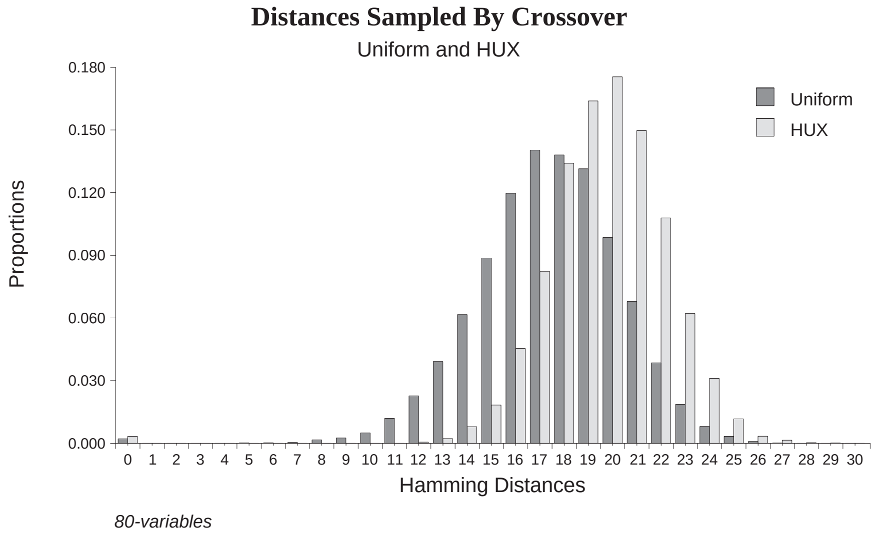


Figure 4.6: Distribution of step-sizes taken by uniform crossover and HUX in the first generation of an ESGA.

of NK landscapes with $N = 80$ and $K = 3$, each using the fixed or random method for choosing the bit interactions.

To empirically verify the distributional bias formulae, the parent-offspring distances were tracked in the first generation of the ESGA and were accumulated for each problem in the set. The distances are tracked solely based on crossover before mutation was applied. Since the population size is 500 and there are 100 problems in each set, the number of samples per problem set is approximately $50,000 \times p_c$ and $p_c = 0.6$ for all four crossover operators, where p_c is the probability that crossover occurs.

Figures 4.5 and 4.6 show the sampling distributions of the resulting parent-offspring distances. Again, the horizontal axis represents the set of Hamming distances, and the vertical axis represents the proportion of the occurrence of the particular Hamming distances. For both graphs, the range

Operator	Random		Adjacent	
	μ	σ	μ	σ
Mutation Only	0.7528	0.01585	0.7474	0.01519
OnePt+Mutation	0.7602	0.01454	0.7587	0.01503
TwoPt+Mutation	0.7607	0.01546	0.7585	0.01488
Uniform+Mutation	0.7622	0.01634	0.7548	0.01522
HUX+Mutation	0.7629	0.01553	0.7540	0.01524

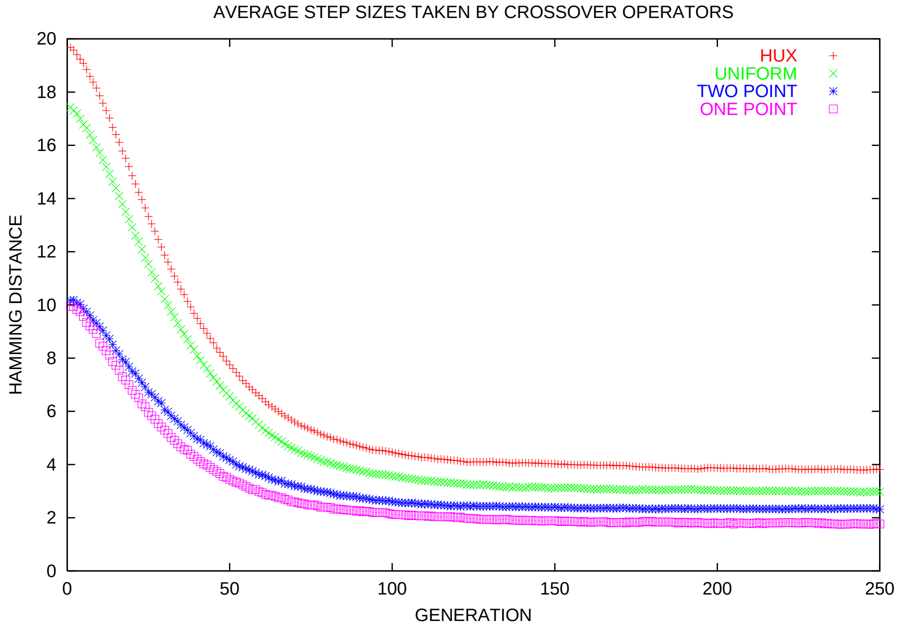
Table 4.2: Final results of ESGA using four crossover operators on two types of NK landscapes.

of Hamming distances was truncated to 30 because Hamming distances larger than 30 were not encountered. Although the problem size is 80 bits rather than 20 bits, the distributions generated for the 20 variable exact sampling distributions are similar to those found empirically for 80 variable problems. These distributions were generated from the set of NK landscapes with adjacent bit interactions; however, it should be noted that the distributions generated from the NK landscapes with randomly chosen bit interactions were indistinguishable from these results.

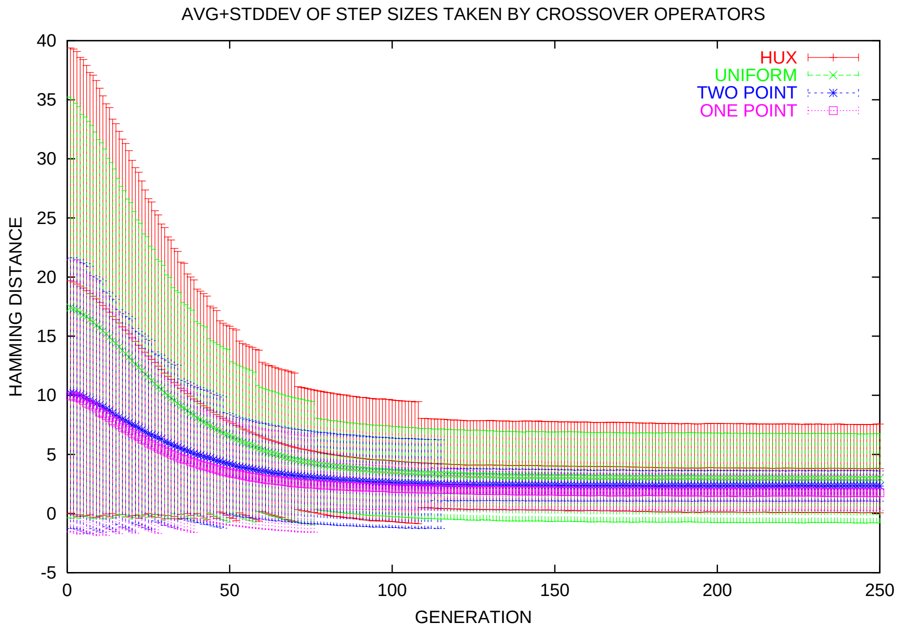
Figure 4.7a tracks the average parent-offspring Hamming distances (or step-sizes) over 250 generations. Figure 4.7b is the average parent-offspring Hamming distances plotted with errorbars indicating the standard deviation of the distances. Each point is the average of 100 problem instances. Again, the plots represent results from the set of NK landscapes with adjacent bit interactions, but the results are illustrative of the results for NK landscapes with random bit interactions. Syswerda (1992) has noted that the parent distances in an initial population are approximately 50 percent of the string length; so crossover should exchange approximately 25 percent of the total number of bits. The graphs illustrate that the actual number of *useful* swaps can be much lower than 25 percent of the string length and drops off quickly as search progresses. Although the standard deviations for all crossover operators are relatively large initially, it is clear from the graphs that HUX and uniform crossover are sampling from a wider range of Hamming distances than one and two point crossover. Over time, the average Hamming distance and the standard deviation of the Hamming distances sampled are quickly decreasing. The use of a large population, small tournament size and mutation were intended to maintain diversity throughout execution; so the results are not an unreasonable indication of how quickly the exploratory power of crossover diminishes over time.

This plot illustrates that the exploration and "global" sampling that is believed to take place during genetic search is considerably limited. The diversity within the genetic algorithm population is quickly diminishing; thus, the points sampled are relatively close to one another. This limitation can be related to the genetic algorithms sampling of local optima. The existence of multiple local optima can be a both a benefit and a detriment to a genetic algorithm. If the basins of attraction of highly fit optima are connected so that they might be used as stepping stones that lead to the global optimum, then even with small step sizes, the genetic algorithm may bypass these optima. If there are numerous randomly spaced highly fit optima, then selection will bias the search towards one basin due to genetic drift and crossover will be increasingly unable to move the population away from the optimum. As the population becomes more focused, the population effectively becomes *stuck* on a suboptimal local optimum.

The relative performance of the various crossover operators varied slightly between the two types of NK landscapes as is illustrated in Table 4.2. The fitness values for arbitrary NK landscapes fall in the range $[0, 1)$, and the ESGA maximized fitness. Table 4.2 shows the mean and standard deviation for the best fitnesses found after 250 generations on both sets of 100 NK landscapes. For the NK



a



b

Figure 4.7: Step sizes taken by crossover over 250 generations.

landscapes with randomly chosen bit interactions, all crossover operators performed similarly. There were no statistically significant differences between the crossover operators. For the NK landscapes with adjacent bit interactions, a one-way ANOVA statistical test confirmed that the crossover operators with positional bias performed significantly different from the crossover operators with no positional bias ($p \leq 0.01$). A one-tailed t-test confirmed there is no significant difference between one and two point crossover; however, both uniform crossover and HUX performed significantly worse than either one or two point crossover ($p \leq 0.05$).

According to a one-tailed t-test ($p \leq 0.05$) comparing the performance of the crossover operators used with mutation versus mutation alone, the crossover operators used in conjunction with mutation significantly outperform mutation alone. Since using crossover and mutation together always outperformed mutation alone regardless of the specific crossover operator or specific problem set, the distributional biases for each crossover operator were categorically beneficial for either set of NK landscapes. The relative performance of the crossover operators may change if the algorithm or algorithm parameters were tuned to the specific problem sets; however, an assessment of the utility of crossover for any other purpose than a means of making "large steps" was beyond the scope of this research.

4.3 Selection, Sampling and Solution Quality

The crossover operator is the primary mechanism for the genetic algorithm to take "large jumps". Initially, different crossover operators sample points that are a limited distance from one another depending on their inherent biases. As search progresses, the population becomes more uniform, and the exploratory effects of crossover wane. From a local optima perspective, as the effects of crossover diminish, the genetic algorithm is more likely to become trapped in a local optimum. However, not all local optima are equally likely to trap a genetic genetic algorithm. Using a fitness based selection operator, particularly with elitism, ensures that only the best points within the population at any given time could be convergence points. The use of population and fitness based selection methods, coupled with the exploratory effects of crossover and mutation make only the highly fit local optima candidate convergence points.

The use of selection biases genetic search to regions of the search space that have a potentially high fitness. To achieve this, selection allocates mating opportunities to highly fit members of the population more often than members with low fitness. Of course, *highly fit* is a relative measure and depends on the particular mixture of the population. However, given a static evaluation function, certain points in the search space may categorically be considered highly fit. For instance, collectively, we might consider the top one percent of all possible evaluations to be considered highly fit. If a highly fit point, relative to all other points in the search space, occurs in the genetic algorithm population, selection will move the genetic algorithm population towards that point.

A common measure of genetic algorithm performance is to track the most highly fit individual encountered during search. Practical genetic algorithms use some form of elitism to guarantee that the fitness of the best individual in any population is monotonically improving over several generations. At any given generation, the fitness of the best individual in the population sets the lower bound on the fitness for any potential convergence points.

The quality of the overall best solution in a population is driven by selection pressure. However, the population size also affects the quality of the best individual in the initial population. Since a genetic algorithm, particularly when elitism is used, will exploit the highly fit members of the population, it is instructive to estimate the fitness of the "best" member of an initial population. Furthermore, the genetic algorithm with elitism is guaranteed not to converge to any point in the

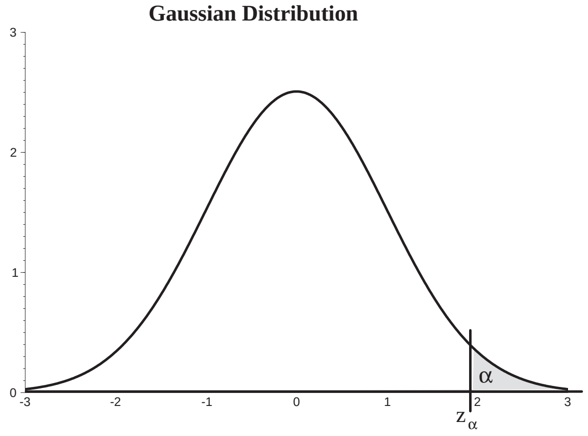


Figure 4.8: Calculating probabilities for $\mathsf{P}(X \geq c)$.

		Population Size				
		50	100	250	500	1000
$K = 2$	μ	1305.90	604.040	306.180	131.880	64.0400
	σ	1672.10	693.172	272.414	104.454	64.6520
$K = 9$	μ	1171.22	484.040	272.720	151.840	47.4200
	σ	1977.42	413.482	265.861	141.469	44.6246
Expected		1310.72	655.36	262.144	131.072	65.536

Table 4.3: Rank of the best fitness in the initial population of a genetic algorithm.

search space with a fitness lower than the best member of the population. The approximate fitness of the best individual in the initial population can be estimated using population size. Since we only need one "best" point in any population, we can estimate the value c for which:

$$\mathsf{P}(X \geq c) \approx \frac{1}{Popsize}$$

where $Popsize$ is the population size, X is a random variable and c is the fitness value at which the probability of encountering that point or points with higher fitness is $\frac{1}{Popsize}$. Let the Gaussian distribution shown in Figure 4.8 represent a fitness distribution (i.e., an enumeration of all possible function evaluations). Suppose we are concerned with locating the value for z_α such that the area to the right of z_α is α. In this case, we are interested in locating the point such that there is a probability of $\frac{1}{Popsize}$ that a point with equal or higher fitness will be chosen.

Since the search spaces used for this research are discrete, finite search spaces, we can rank all of the fitness values from 1 to 2^L. Assume that a low rank indicates a high fitness and assume that all fitnesses are unique. The fitness ranks are a one-to-one mapping onto the set of fitnesses. Regardless of the specific fitness distribution, we can compute the rank r_α that corresponds to the index of the fitness at position z_α. For any distribution, the r_α that is approximately $\frac{1}{Popsize}$ is at position $\lceil \frac{2^L}{Popsize} \rceil$. The rank index of $\lceil \frac{2^L}{Popsize} \rceil$ is an approximation of the rank of the best fitness in an initial random population, or any random sample, of size $Popsize$.

An experiment was performed to examine the accuracy of the estimated rank of the best individual in the initial population. Table 4.3 lists the results of sampling two sets of enumerable NK landscapes with $N = 16$ and $K = \{2, 9\}$. The population size varied from 50 to 1000 individuals. There were 50 randomly generated problem instances used to generate each result. Additionally,

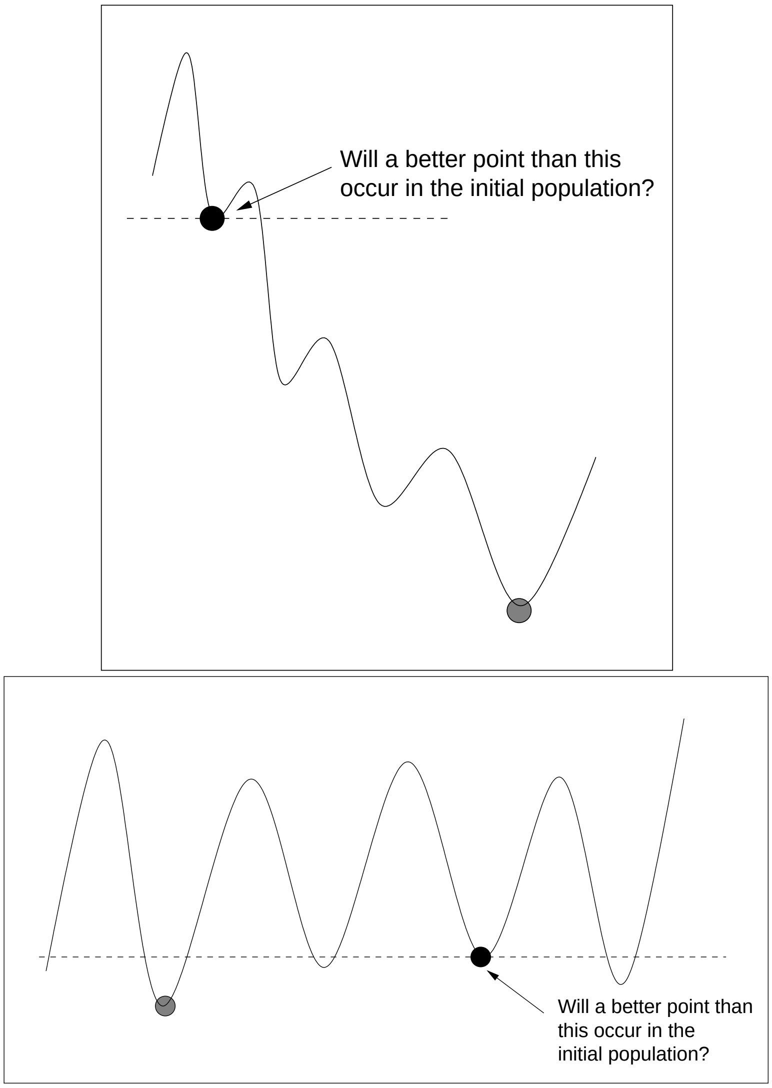


Figure 4.9: Two examples of local optima.

both problems were verified to contain unique fitness values. The ranks of the best individuals in the initial population were computed and averaged together over each problem instance. The numbers in the bottom row correspond to $\lceil \frac{2^L}{Popsize} \rceil$. The mean for each of the NK landscape examples vary from the expected rank of the best individual; however, the value produced by $\lceil \frac{2^L}{Popsize} \rceil$ is a coarse grained approximation to the empirical results. For each population size, the results for $K = 2$ and $K = 9$ were found to have no significant differences ($p \leq 0.05$ using a Students' t-test). Note that the results were generated using only a random sample of size $Popsize$; no search had taken place when the results were generated.

Figure 4.9 represents two functions being optimized by a genetic algorithm. In the top example, the point represented by the black circle is a local optimum; however, it is unlikely that a large population could not randomly sample points with a higher fitness. In other words, since the genetic algorithm is initialized with a sample of points rather than a single point, its initial population will contain points lying in multiple basins of attraction and the basins of attraction for local optima with relatively low fitnesses will quickly be eliminated by selection. The lower function shows a multimodal function containing several highly fit local optima with large basins of attraction. This type of function will pose a problem for the genetic algorithm because all optima are competitive. It is less likely that local optima will be eliminated based on the sampling of the population.

This figure is nearly identical to the figure in Chapter 3 that explained why some minima would or would not be coalesced during the encoding process. A similar sampling argument holds for increasing the size of the search neighborhood as well. A local optimum of low fitness under one encoding is not likely to remain locally optimal under a new encoding if the size of the search neighborhood is increased.

Simply counting the number of optima does not take into account the relative fitnesses of the optima or their basins of attraction. A function can be highly multimodal, but have a global trend that causes many of the local optima to have low fitnesses. A population will randomly sample from the search space; the genetic algorithm should effectively ignore local optima with low fitnesses. As the population samples points in the search space using crossover, the fitness requirements for local optima are restricted even further. Between selection and crossover, fitness standards are set to limit the quality of the optima that could be candidate convergence points for a genetic algorithm. For this reason, just measuring the number of local optima cannot indicate problem difficulty for a genetic algorithm.

4.4 Why GA Convergence Points Should be Defined by the Mutation Landscape

Jones (1995) argues that the search landscapes be defined with respect to each operator used during search rather than the default mutation landscape. As an alternative to Jones' argument, it will be argued that when multiple exploratory operators are used in tandem, local optima should be defined with respect to the lowest order neighborhood used. Local optima for crossover tend to be defined using a mixture of higher order (higher than Hamming distance one) neighborhoods. During the execution of a genetic algorithm, the exploratory capabilities of crossover depend on the mixture of the population at any given time. For traditional genetic algorithms, the exploratory power of crossover diminishes rapidly with population convergence. Without a restart mechanism, crossover ultimately has little effect on the latter generations of genetic algorithm execution. Without mutation the genetic algorithm might not converge to an HD_1 local optimum; however, if a random mutation is used in a genetic algorithm with a probability near $\frac{1}{L}$, then assuming the genetic algorithm is allowed to run to convergence, mutation and selection will drive the genetic algorithm population to a HD_1 local optimum.

Conjecture:

> A genetic algorithm using fitness based selection and random mutation with a rate of $\frac{1}{L}$ should converge to a Hamming distance one local optimum if the algorithm is run to convergence. Because mutation is a completely random operator, it guarantees that the global optimum will be sampled infinitely often given an infinite amount of time. However, if a genetic algorithm using mutation is given an adequate, but finite, amount of run time, it will converge to *some* HD_1 local optimum with a high probability.

As was seen in the NK landscapes example, crossover has little exploratory effect after a fairly small number of generations. At this point, the population has reached a point where the population is sampling one region or basin of attraction. Suppose that the population of the genetic algorithm has converged to a set of points in the basin of attraction for a HD_1 local optimum, but is not sampling the local optimum. Then any move towards the local optimum will result in a higher fitness and the fitness based selection strategy will propagate the improved string (in some form) into future generations.

The only way that the genetic algorithm can avoid moving towards the optimum is to continually choose non-improving moves that lead away from the optimum. To maximize the probability that every member of the population *avoids* an improving move, assume that each point chooses from $L-1$ bits to move away from the local optimum. This represents the worst case scenario where there is only one bit that will yield an improvement. The probability that any string does not move towards the local optimum is $1-\frac{1}{L}$. Then the probability that no string in the population moves towards the local optimum in one generation is $(1-\frac{1}{L})^{popsize}$. Over t generations, the probability that no string in the population will move towards the HD_1 local optimum is $(1-\frac{1}{L})^{t \cdot popsize}$. Taking the limit of this equation:

$$\lim_{t \to \infty} \left(1 - \frac{1}{L}\right)^{t \cdot popsize} = 0$$

where *popsize* and L are constant.

Typically, the number of improving moves emanating from a single point is higher than just a single bit. Additionally, fitness based selection will bias the sampling so that points of high fitness are allocated more mating opportunities than other points in the population. Since the points in the population fall within the basin of attraction for the local optimum, the points with higher fitness will tend to be closer, in terms of Hamming distance, to the local optimum. All of these factors will enable mutation to move towards a local optimum.

Both Culberson (1992) and Jones (1995) illustrate that crossover operators define a set of local optima that are different from the set of mutation (single-bit flip) local optima. If a genetic algorithm using both crossover and mutation converges to a local optimum defined by the crossover operator, then this local optimum must also be locally optimal with respect to mutation. In a traditional genetic algorithm, crossover is not applied to complementary pairs of strings; so the population would quickly become uniform. At that point, mutation would be the only practical means for exploration; thus, any convergence point of the genetic algorithm must be a mutation, HD_1 local optimum.

4.5 Genetic Algorithm Behavior and Local Optima

This chapter has illustrated how the three genetic operators affect the quality and type of local optima that can be potential convergence points for genetic algorithms. Three primary observations are:

- Crossover operators sample from large HD neighborhoods early in search but sample smaller HD neighborhoods as search progresses.
- The "best" member of a population is initially ranked near $\frac{1}{Popsize}$ and improves from that point.
- The convergence point of a genetic algorithm using a mutation rate of $\frac{1}{L}$ must be a HD_1 local optimum.

These three observations imply that the set of possible convergence points that can be solutions found by a genetic algorithm using mutation should be a highly fit subset of the HD_1 local optima, given a sufficient, but finite, amount of time. The subset of possible genetic algorithm convergence points should be highly fit HD_1 ranked higher than $\frac{1}{Popsize}$.

The specific genetic operators affect the pool of potential genetic algorithm convergence points. Since crossover samples higher order Hamming distance neighborhoods, they can avoid some HD_1 local optima. Precisely how this occurs depends on the biases of the crossover operators coupled with the rate of convergence of the genetic algorithm. The rate of convergence of the genetic algorithm depends on the relative sizes of the basins of attraction of the set of optima coupled with the amount of selection pressure used in the specific genetic algorithm. If the basins of attraction are large, then highly fit points will tend to occur near other highly fit points. Thus, selection will direct the population to one region containing highly fit points. How quickly selection drives the population to that region depends on the specific selection mechanism and the amount of selection pressure. If the basins of attraction are small, then highly fit points will tend not to be localized in one region, and the population will remain diverse until selection causes the population to *drift* towards one local optimum. Ultimately, local optima do affect genetic algorithm behavior, but the genetic algorithm may not be affected by *all* local optima.

In Chapter 3, the experimental results illustrated that the number of local optima does not always affect genetic algorithm behavior. It was argued in this chapter that only the highly fit local optima affect the behavior of genetic algorithms. The remainder of this chapter will illustrate this conjecture empirically.

4.5.1 Genetic Algorithm Sampling: A DFG Example

To illustrate the relevance of local optima to genetic search, several experiments were performed to determine whether or not the genetic algorithm treats local optima differently from other points in the search space. The theoretical arguments presented in this chapter suggest that the genetic algorithm will heavily sample and potentially become trapped by Hamming Distance one local optima; therefore, this experiment should substantiate or refute the theoretical arguments.

The Discrete Function Generator (DFG), introduced in Chapter 3, can be used to create functions with no biased global structure and a specific number of local optima. The first experiment uses a 16-bit DFG test function with 50 local optima. Three algorithms were tested to determine how points in the search space were sampled throughout execution. The baseline algorithm was the Random Bit Climber (RBC) (Davis 1991), which illustrates how a single-point local search algorithm using random restarts samples points in the search space. The next two algorithms studied are the Simple Genetic Algorithm using elitism (ESGA), and the simple genetic algorithm not using elitism (SGA). Algorithm descriptions can be found in Chapter 3.

For this set of experiments, all three algorithms were run on a Binary Reflected Gray coded version of the function that was verified to contain exactly 50 local optima. Figure 4.10 is a plot of the test function with 50 local optima. The fitnesses were drawn from a Gaussian distribution and were subsequently scaled between the range of zero to one. The algorithms were presented with a *separable* two dimensional version of the problem. Since the problem is separable (i.e., each parameter can be optimized independently), we can treat each problem parameter as separate problem instances and track the optima independent of one another.

All algorithms were intended to minimize the fitness, and the minimum fitness is 0.0 for the problems (i.e., the global optimum for each problem is known to have the fitness of 0.0). Each algorithm was run 25 times and was allowed $40,000$ evaluations for each run. A total of $1,000,000$ points in the search space were sampled for each experiment. Furthermore, two parameters meant that $2,000,000$ samples were drawn from the one dimensional discrete function. Since there are only $65,536$ unique fitness values in the one dimensional function, a sample size of $2,000,000$ is adequate for sampling the one dimensional function.

Across all 25 runs, the frequency that the algorithms visited each point in the search space

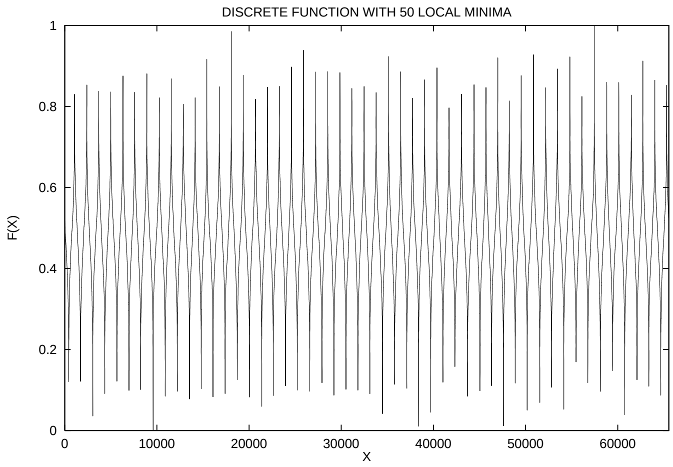
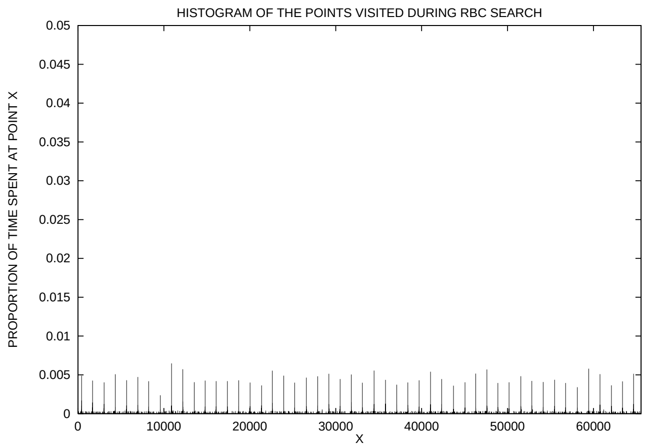
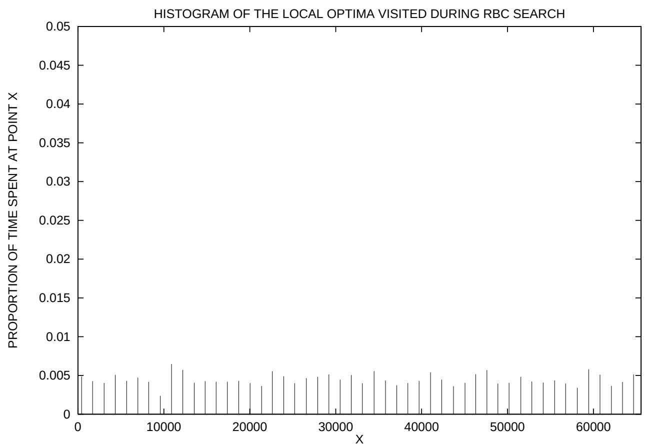


Figure 4.10: A plot of the test function with 50 optima.



(a)



(b)

Figure 4.11: Histogram for the points visited by RBC.

was counted. For RBC, every point that was evaluated was counted as "visited". For SGA and ESGA, every member of the population during each generation was considered "visited". For each algorithm, a histogram was computed to show how often any of the $65,536$ points were visited during search. Each of the bins in the histograms was normalized by dividing by $2,000,000$ so that the plots indicate the proportion of samples occurring at a specific point rather than a raw count. Note that the histograms are the cumulative sampling of points during the entire search; the distribution of points within the genetic algorithm population vary from generation to generation.

Figure 4.11a is the resulting histogram of points visited by RBC. The vertical axis ranges from 0.0 to 0.05 to match the axis of the SGA and ESGA results. RBC visited each of the points in the search space relatively frequently and does not tend to be biased to search one portion of the space more heavily than another. Over all 25 runs, RBC sampled 92 percent of the one dimensional function. Figure 4.11b is the same histogram that has been filtered so that only the local optima are included. This plot illustrates that each of the optima were sampled uniformly. Since the basins of attraction for the optima had very similar shapes and sizes, this histogram is what should be expected for RBC. RBC climbs to an optimum and randomly restarts. If the basins of attraction are all roughly the same size, then RBC should visit the optima with nearly the same frequency. Less than 23 percent of RBC's $2,000,000$ possible samples were local optima.

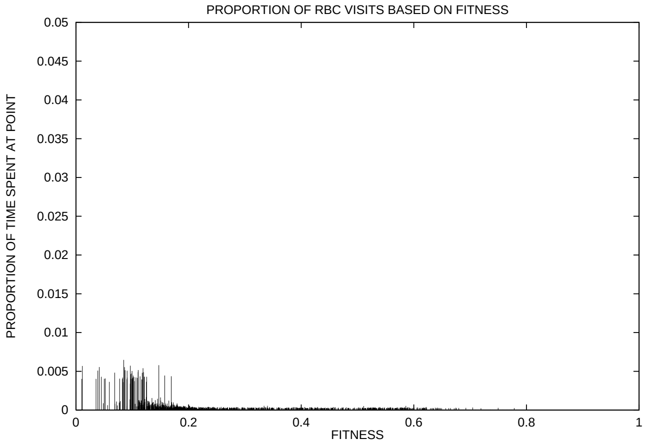
We can view the results from a different perspective to determine whether RBC samples points biased by the fitness of the points. Figure 4.12a is the same data from Figure 4.11a, except that it is plotted against the fitness of every point. Since this function is a minimization problem, the leftmost points on the x-axis represent points with high fitnesses. Figure 4.12b is the same plot except that only the local optima are plotted. The horizontal stripe of points occurring at $y = 0.025$ mark where the local optima occur in the function. It is clear from this plot that all of the local optima occurring in the DFG function have a relatively high fitness. However, with respect to one another, there are clearly some optima that are better than others. RBC is sampling each of the optima in a seemingly unbiased manner because it is only sensitive to the size of the basin of attraction.

Figure 4.13a is the histogram of the proportion of time SGA (no elitism) spent at each of the possible points in the discrete function. The genetic algorithm sampled 85 percent of the one dimensional function. It is clear that the SGA sampled the search space in a much more biased fashion than RBC. Figure 4.13b is the same histogram filtered so that only the local optima are shown. The SGA sampled the local optima much more frequently than RBC. The amount of search effort placed at local optima comprised over 40 percent of the allotted $2,000,000$ samples.

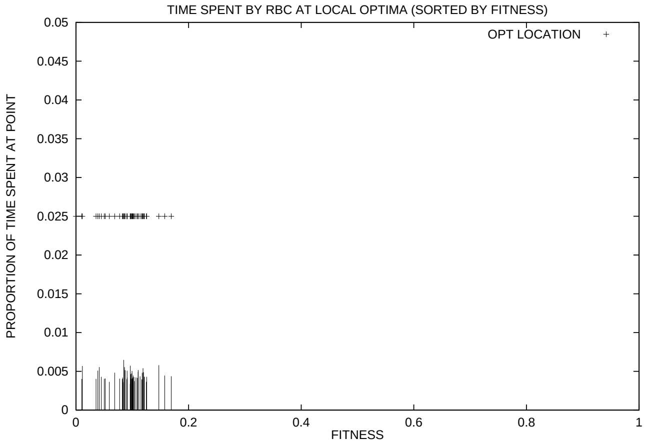
Figure 4.14a is a histogram of the proportion of time SGA spent at points ordered according to fitness. Recall that each point in the search space had a unique fitness drawn from a Gaussian distribution that was normalized to fall in the interval [0,1]. Like RBC, most sampling was performed at the points having extremely high fitness. Figure 4.14b is the same plot filtered so that only the local optima are shown. Unlike RBC, SGA did not sample the optima evenly and in some cases sampled the optima very infrequently. SGA had a tendency to sample the more highly fit optima more often than the optima with lower fitnesses. The fact that it samples some of the suboptimal local optima more heavily than others indicates that the SGA was *stuck* at these points.

Finally, Figure 4.15a presents the histogram for the proportion of time ESGA (SGA with elitism) spent at each of the possible points in the discrete function. ESGA sampled approximately 82 percent of the one dimensional problem. The histogram for ESGA is very similar to the histogram for SGA. Figure 4.15b is the histogram of the proportion of time ESGA spent at local optima. As with SGA, ESGA spent approximately 40 percent of its time sampling local optima.

Figure 4.16a is a histogram of the proportion of time ESGA spent at points having a particular fitness. Figure 4.16b is the same plot filtered so that only the local optima are shown. The results for



(a)

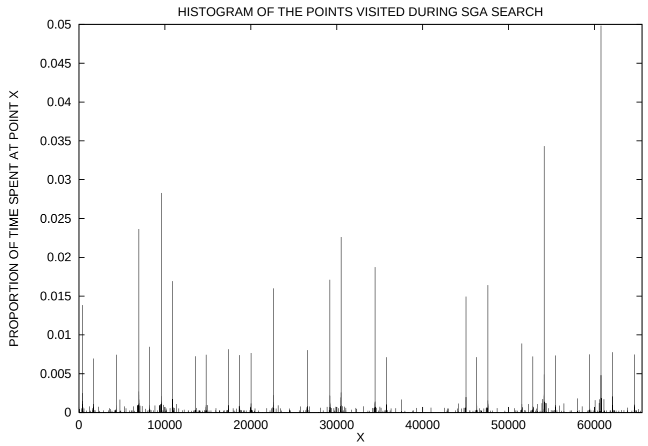


(b)

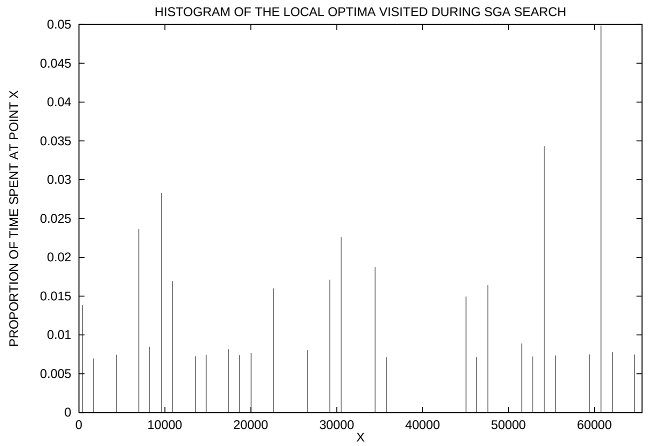
Figure 4.12: Histogram for the points visited by RBC (sorted by fitness).

ESGA are similar to the SGA results. In both cases, a few select optima were sampled more heavily than others. The purpose of considering ESGA in addition to SGA was to determine whether the use of elitism biased the manner in which points were sampled by the genetic algorithm. These results do not indicate a dramatic change between the two forms of the genetic algorithm. However, one key difference between the two versions of the genetic algorithm was how frequently the global minimum was sampled. SGA sampled the global minimum over 2.5 percent of the time while ESGA sampled the global minimum less than 1.0 percent of the time. Since the placement of the optima within the DFG function was random, the added exploration occurring in the SGA was beneficial in this case.

This experiment illustrates that the genetic algorithm heavily samples local optima. SGA and RBC both sampled all 50 optima while ESGA sampled 49 of the 50 optima. However, the genetic algorithm tends to sample some optima more heavily than others. The histograms illustrate that the genetic algorithm frequently sampled the highly fit local optima much more often that the optima with lower fitnesses. When the genetic algorithm heavily samples local, rather than global optima, such problems are deemed difficult.



(a)



(b)

Figure 4.13: Histogram for the points visited by SGA (no elitism).

4.5.2 Genetic Algorithm Sampling: Rastrigin's Function

A second experiment examines how the genetic algorithm samples points in a problem that is known to have a regular structure. Rastrigin's function is formulated as a summation of parabolic cosine waves:

$$F(x_i \mid_{i=1,N}) = (N * 10) + \left[\sum_{i=1}^{N} \left(x_i^2 - 10\cos(2\pi x_i) \right) \right]$$

A plot of a one dimensional version of Rastrigin's function is given in Figure 4.17. The problem is formulated as an unbounded, continuous optimization problem; however, it is typically discretized, bounded between -5.12 and 5.11 and encoded using a bit encoding method. For this experiment, each parameter will be encoded using a 10-bit Binary Reflected Gray encoding.

This experiment illustrates how the Random Bit Climber and the Simple Genetic algorithm sampled points using Rastrigin's function as the evaluation function. ESGA and SGA showed little variation in the DFG function experiment, the SGA (no elitism) was used for this experiment. Since Rastrigin's function is separable and uses only a 10-bit encoding for each input parameter, histograms can be generated to track how frequently RBC and SGA visit each point in the one-

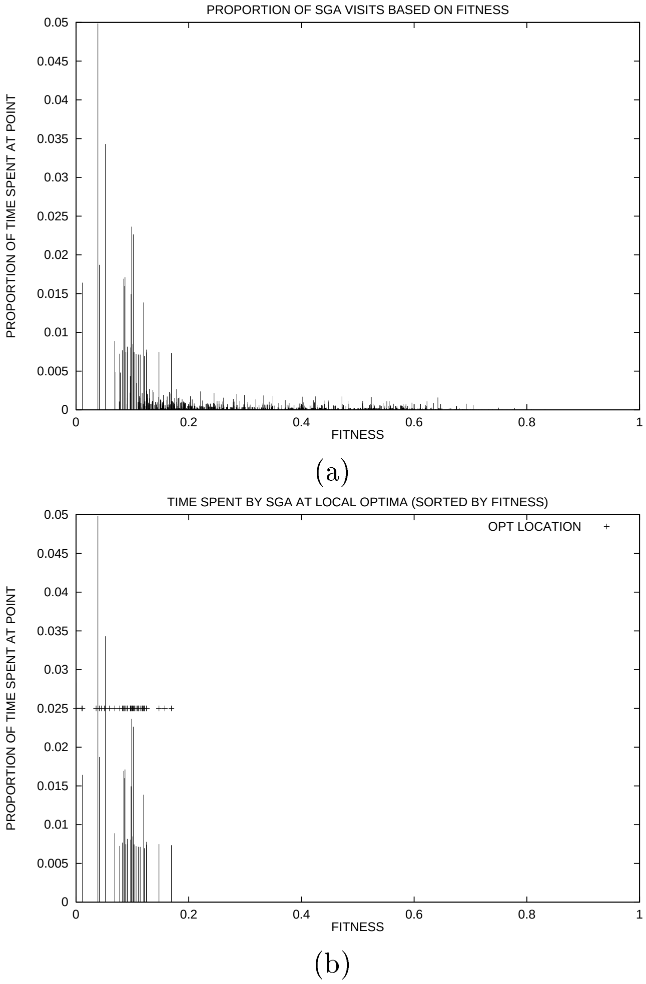
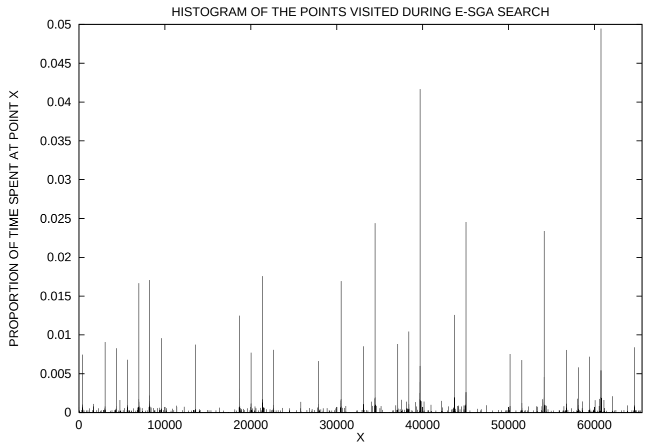


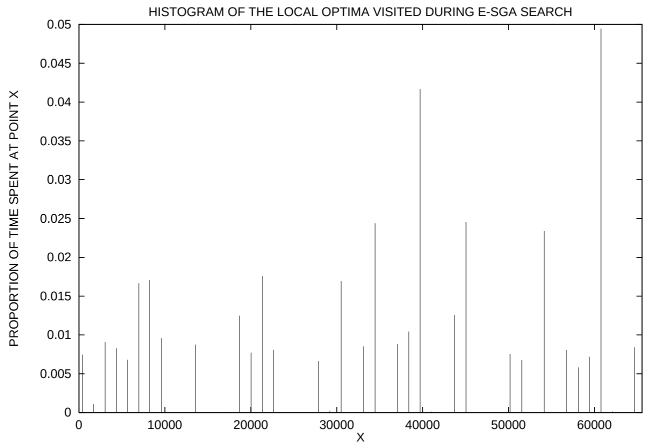
Figure 4.14: Histogram for the points visited by SGA (sorted by fitness).

dimensional search space. Each run of RBC was allowed 30,000 evaluations. SGA used a population size of 200 and was allowed to run for 150 generations; so 30,000 points were also sampled for each run. Both algorithms were run 25 times on a 10 parameter version of Rastrigin's function. Since each parameter can be optimized independently, there were 7,500,000 opportunities to sample each of the 1024 points in the one-dimensional problem. Although it seems that 7,500,000 sampling opportunities for each of the 1024 points is excessive, 30,000 evaluations of a 100 bit problem is actually an extremely small sample of the search space. The seemingly excessive sampling of each point is a direct result of the separable design of the problem.

Figure 4.18a is the histogram illustrating the time RBC spent at each point in the search space. Rastrigin's function should be an unbounded optimization problem, but has traditionally been bounded between $[-5.12, 5.11]$ for genetic algorithm comparative studies (Whitley, Mathias, Rana, and Dzubera 1996). Using this range of inputs, there are 11 local minima in the numeric function. For this experiment, the function is encoded using a Binary Reflected Gray encoding. The Gray encoding reduces the number of optima from 11 to five. The five spikes in Figure 4.18a correspond to the local optima under a Gray encoding. Note that the X range has been discretized and changed to $[0, 1023]$, which corresponds to the decoded bit string values before they have been mapped onto



(a)

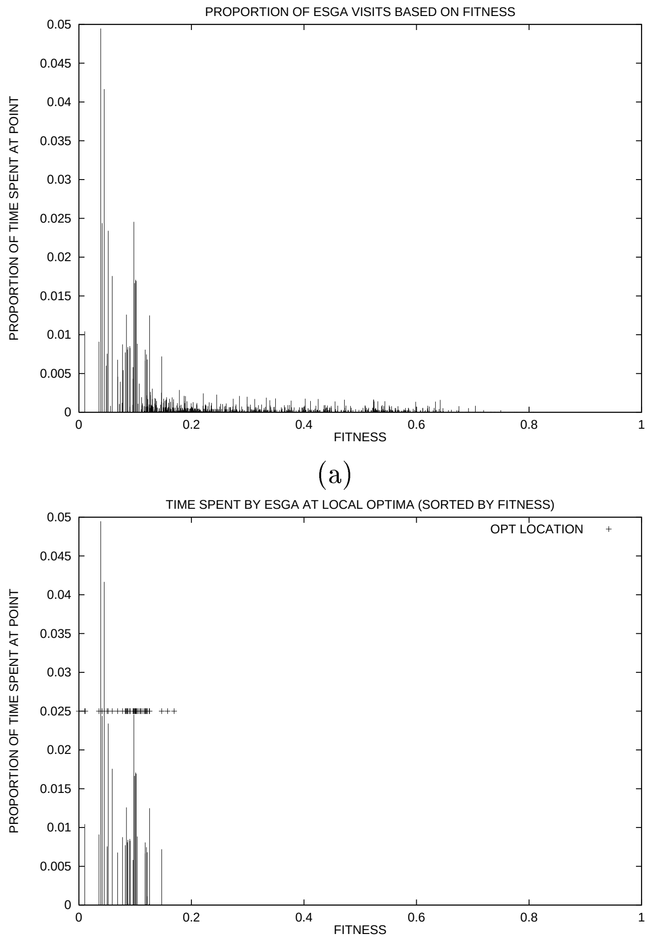


(b)

Figure 4.15: Histogram for the points visited by ESGA.

the range $[-5.12, 5.11]$. The two outermost optima have an evaluation of approximately 3.98, the next two optima at $X = 413$ and $X = 611$ have an evaluation of approximately 1.0 and the center optimum is the global optimum with an evaluation of 0.0. It is clear from the histogram that RBC did not sample these optima based on their fitness. Figure 4.18b is the corresponding histogram for SGA. SGA sampled the global optimum significantly more often than any other point in the search space. It also sampled the next best optima in nearly equal proportions due to the symmetry in the search space. The other two local optima were sampled very infrequently (about 0.5 percent of the time).

This experiment substantiates the DFG results presented in the previous section. Even on a very structured problem, RBC shows no obvious bias in terms of how it samples the set of local optima. This indicates that the basins of attraction for each of the optima in Rastrigin's function are roughly the same size. SGA predictably samples the highly fit points and local optima considerably more often than other points in the search space. Furthermore, SGA samples the highly fit local optima considerably more often than other local optima.



(a)

(b)

Figure 4.16: Histogram for the points visited by ESGA (sorted by fitness).

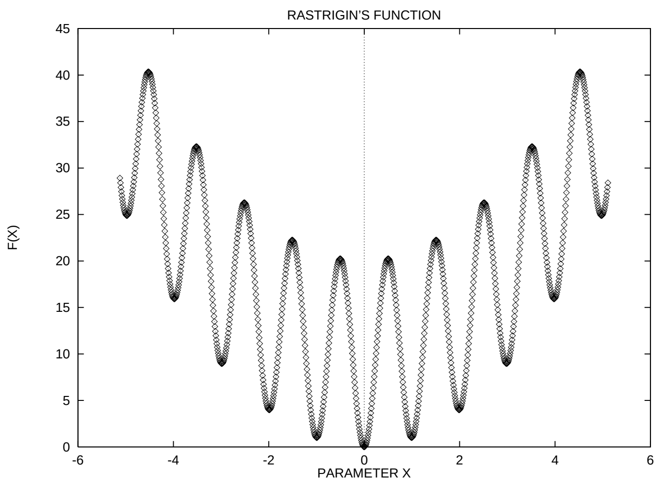
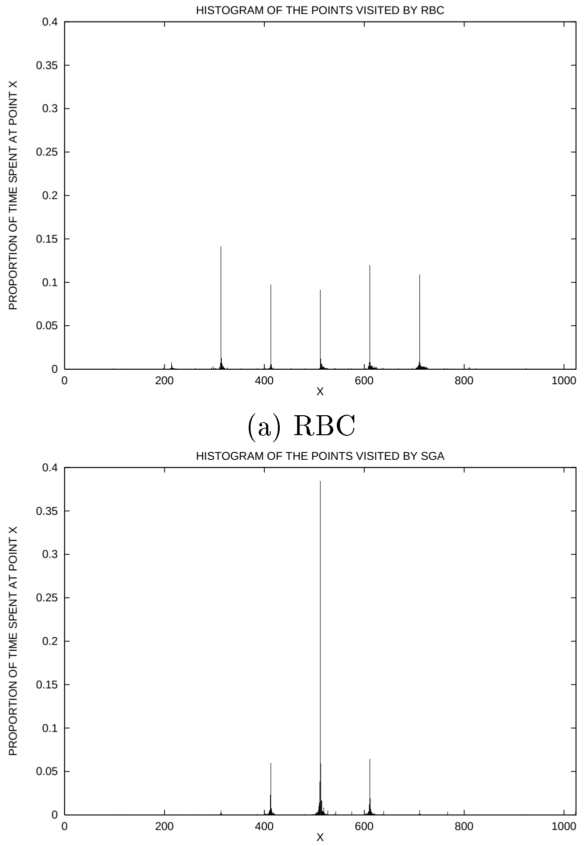


Figure 4.17: One dimensional plot of Rastrigin's function.



(a) RBC

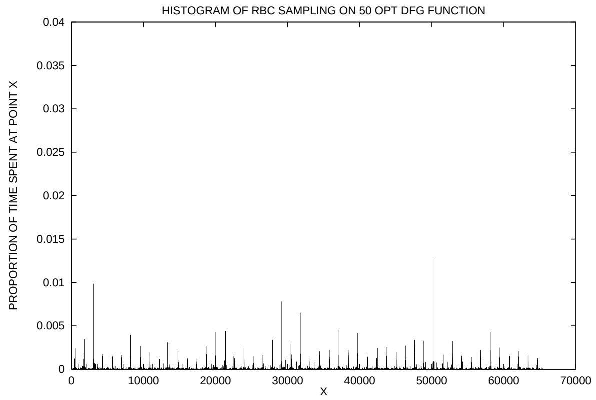
(b) SGA

Figure 4.18: Histogram for the points visited by RBC and SGA on Rastrigin's Function.

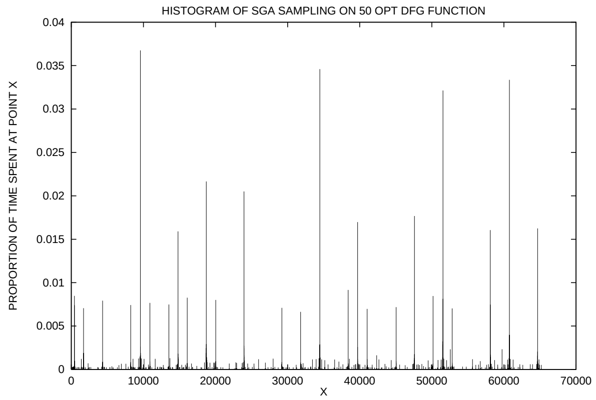
4.5.3 Genetic Algorithm Sampling: Gray vs. Binary

In Chapter 3, the behavior of the genetic algorithm on the DFG test suite illustrated that the number of local optima generated by the Gray and Binary encodings did not affect performance. In this chapter, analytical arguments and empirical results show that the genetic algorithm does not treat optima equally; the genetic algorithm samples optima based on fitness. Given this new insight into the behavior of genetic algorithms, this next experiment revisits the issue of how the encoding method can (indirectly) affect genetic algorithm performance.

Recall that, in the previous experiments, the 50 optimum DFG test function and Rastrigin's functions had been encoded using a Binary Reflected Gray encoding. This final experiment illustrates the results of running both RBC and SGA, without elitism, on Binary encoded versions of the DFG function and Rastrigin's function. The configuration of each algorithm is identical to the previous experiments. Each algorithm was run 25 times on each problem. For the DFG function, RBC and SGA were allowed $40,000$ evaluations. SGA had a population size of 200 and was run for 200 generations. The DFG function was presented as a two dimensional minimization problem, which allowed $2,000,000$ samples of the 16-bit one dimensional DFG function. On Rastrigin's function, both algorithms were allowed $30,000$ evaluations; SGA was run for 150 generations with a population size of 200. The only difference in the experiment configuration is the use of a Binary encoding rather than a Gray encoding.



(a) RBC



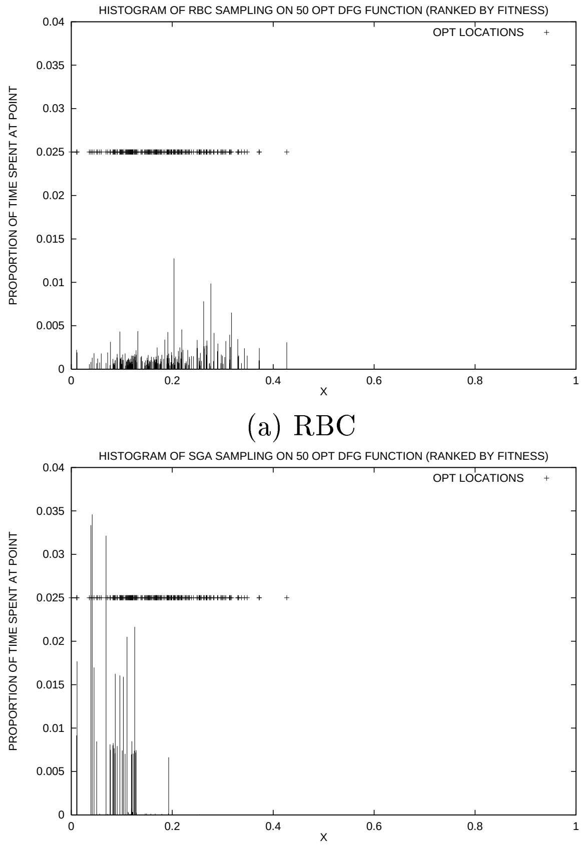
(b) SGA

Figure 4.19: Histogram for the points visited by RBC and SGA on a Binary encoded DFG function.

The original version of the 16-bit DFG function contained 50 optima. When this function was encoded using a Standard Binary encoding, the number of optima increased to 200. Figure 4.19 presents the histograms of points visited by RBC (a) and SGA (b). The histograms are consistent with the results for RBC and SGA on the Gray encoded version of the problem. RBC sampled *most* optima equally; however there were four optima that appear to have been visited more frequently by RBC. This effect can be explained by the change in connectivity of the search space when a Binary encoding is used. The basins of attraction for those four local optima are larger and stronger than the basins of attraction for the other local optima. Figure 4.19b presents the corresponding histogram for SGA (no elitism). Unlike RBC, SGA sampled fewer of the local optima but sampled those optima more frequently.

Figure 4.20 is the same data for RBC (a) and SGA (b) plotted with respect to fitness rather than the input value for X. Again, the horizontal collection of points at $y = 0.025$ marks where the local optima occur. Both graphs have been filtered so that only the local optima are plotted. From Figure 4.20a, it is clear that RBC frequently sampled all local optima regardless of fitness. The local optima that were sampled more heavily than others still had relatively low fitnesses, which indicates that their basins of attraction were somehow easier for RBC to encounter. Figure 4.20b illustrates the sampling of local optima by SGA. Again, SGA spent the majority of its time at the *highly fit* local optima and was undeterred by the inferior local optima.

The results for SGA on the DFG function explain the empirical results illustrating that the



(a) RBC

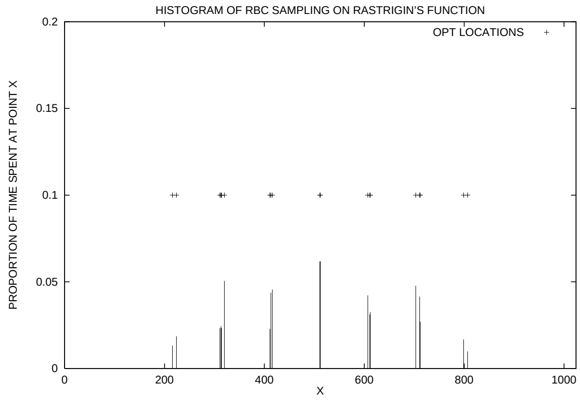
(b) SGA

Figure 4.20: Histogram for the local optima visited by RBC and SGA on a Binary encoded DFG function.

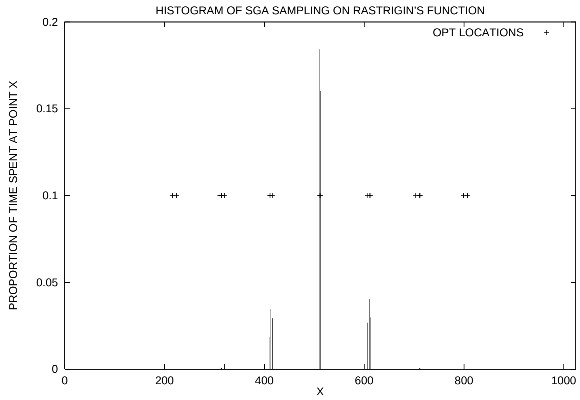
genetic algorithms behaved similarly under a Gray encoding and Binary encoding on the DFG test suite. Although the Binary encoding induced many local optima in each problem, there was still a small set of *highly fit* local optima under both encodings. Since SGA rarely located the global optimum in the set of DFG functions, the comparisons were made based on the solution quality of the best member in the population. However, if the local optima that the genetic algorithm was sampling were highly fit, then the performance of the genetic algorithm was seemingly the same under both representations. Since the genetic algorithm tends to overlook local optima that have a low relative fitness, the raw number of local optima was not a factor the performance of the genetic algorithm on the DFG functions.

In addition to the DFG function, Rastrigin's function was also tested to determine how the genetic algorithm and RBC would sample points under a Binary encoding. The Gray encoded version of Rastrigin's function has five local optima while the numeric version of the problem contains 11. The Binary encoded version of the problem increases that number to 19. The optima for Rastrigin's function, under the Binary encoding, occur in tight clusters around the true local optima under the numeric encoding.

A common effect of using a Binary encoding is the Hamming cliff phenomenon (Caruana and Schaffer 1988; Whitley, Mathias, Rana, and Dzubera 1996). A Hamming cliff occurs when two numbers that are adjacent in numeric space are complementary strings in a Binary encoding. For



(a) RBC



(b) SGA

Figure 4.21: Histogram for the points visited by RBC and SGA on a Binary encoded Rastrigin's Function.

instance, the four-bit strings `0111` and `1000` are seven and eight in numeric space, but have a Hamming distance of four in Binary space. While not all local optima induced under a Binary encoding are the result of full Hamming cliffs, they are a result of having two numbers that are adjacent in numeric space differ by two or more bits in Binary space. Since Rastrigin's function is so structured, the neighboring points of local optima tend to be highly fit points; thus, they become local optima when a Binary encoding is used.

Figure 4.21 presents the results of running RBC (a) and SGA (b) on Rastrigin's function using a Binary encoding. The horizontal collection of points occurring at $y = 0.1$ mark the locations of the local optima in the problem. Unlike the DFG function, the fitnesses in Rastrigin's function are not unique; so the histograms will not be reordered based on fitness. However, the fitnesses of the local optima are regularly structured so that the central optimum (at $X = 512$) is the global minimum and the fitnesses of the other local optima increase as we radiate out from that point. Figure 4.21a illustrates that RBC sampled the local optima fairly evenly throughout search. However, the local optima occurring in the outer spikes tended to be sampled less often. Since these optima have such low fitnesses, the basins of attraction for these points had to be relatively small (i.e., there are too many improving moves available to have these basins be large). Figure 4.21b illustrates that SGA behaves similar under the Binary encoding as it did under the Gray encoding. However, the global optima is sampled considerably less often in Binary than it is in the Gray encoding. This difference is due to the Hamming cliff that occurs between the optimum at 512 and its highly fit neighbor at 511. Although numerous local optima have been induced, the genetic algorithm still has a tendency to sample the *highly fit* local optima most heavily.

All of the experiments illustrate that local optima do affect the performance a genetic algorithm. The number of local optima seems to play a role in the behavior of RBC; however, the extent to which local optima affect local search depends on the sizes of the basins of attraction for each optimum (Tovey 1985). The fitness of each optimum is not a strong factor in the frequency that it is sampled by RBC. On the other hand, the genetic algorithm samples local optima in a biased fashion by visiting local optima with relatively high fitnesses. This result explains why the number of local optima is not a good measure of problem difficulty for genetic algorithms; the number of local optima is not an indication of how many highly fit local optima occur in the space.

4.6 Summary

It is not uncommon for algorithms to adapt search step-sizes so that a large step-size is used early in execution and decreases as search progresses. The most common means for a genetic algorithm to adapt its search step-size is through the use of crossover. When initialized with a random population, the genetic algorithm will sample points that are distant in Hamming space according to the neighborhood sampling distribution for the specific crossover operator. The degree to which crossover continues to explore the search space depends on the amount of diversity in the population. In addition to population diversity, the exploratory power of crossover is limited by its inherent positional and distributional bias.

This chapter presents exact calculations for the Hamming distances sampled by: one point crossover, two point crossover, uniform crossover and HUX. The calculations are representative of the sampling that takes place in the early generations for a genetic algorithm with a large population. The distributional bias for crossover is the Hamming distances between parents and offspring, so these calculations can be used to characterize the distributional bias for practical crossover operators.

The analytical results were verified empirically by examining the distances sampled by crossover during the execution of an ESGA on two variants of NK landscapes. The NK landscapes results illustrate that the analytical formulae for computing the distribution of distances between parents and offspring accurately models the distributions of distances occurring in the first generation of a simple genetic algorithm. In addition, results were presented to illustrate that the sampling that takes place by a genetic algorithm population restricts the set of possible "best" individuals to a number of potential solutions proportional to the population size.

The net results of this work are evidence that genetic algorithms are restricted to converge to Hamming distance one local optima that have a high relative fitness. The results in this chapter also have implications for local search algorithms that use random restarts. Just as a genetic algorithm can restrict the set of potential "best" solutions by sampling a large number of strings in the initial population, local search algorithms that use random restarts can also reap the same benefit. If a local search algorithm is initialized using a single point, that point may have a very low fitness and lead to an optimum with a very low fitness. However, if a local search algorithm is initialized to a point with a relatively high fitness, it has a better opportunity (in non-random functions) to locate a local optimum with a high fitness. Using random restarts enables local search to randomly have multiple starting points, which provides more opportunities to sample starting points with a relatively high fitness.

Furthermore, this chapter illustrates that the local optima in search problems *can* deter the progress of the genetic algorithm because the genetic algorithm tends to sample the local optima more heavily than other points in the search space. Compared to a simple single-point local search algorithm, the genetic algorithm samples the local optima nearly twice as often. While the single-point local search algorithm sampled the local optima in an unbiased manner, the local optima sampled by the genetic algorithm were only those that were highly fit. Although this may sometimes be beneficial, assuming that the genetic algorithm is able to move from good local optima to better local optima, this treatment of local optima can put the genetic algorithm at a disadvantage by allowing the genetic algorithm to become trapped at local optima. Towards the latter generations of a genetic algorithm execution, the genetic algorithm emphasizes exploitation more than exploration by sampling and resampling points in the population. Furthermore, over time, the population becomes dominated by points which are similar or identical. From an optimization perspective, this exploitation is only beneficial when the genetic algorithm population contains points that occur in the basin of attraction for the global optimum.

Chapter 5

Schema Processing, Plateaus and MAXSAT

Random Boolean Satisfiability (SAT) function generators have recently been proposed as test function generators for genetic algorithms(DeJong, Potter, and Spears 1997). However, empirical evidence suggests that genetic algorithms do not perform well on this class of problems (Frank 1994; Lee, Koh, and Nakakuki 1994) while simple local search algorithms perform exceptionally well (Selman, Levesque, and Mitchell 1992; Selman, Kautz, and Cohen 1993). This chapter explains the genetic algorithm's poor performance relative to local search by examining what problem features of the Satisfiability problem make it amenable to local search but not to genetic search.

This chapter will illustrate that a Walsh analysis of MAXSAT problems can be performed in polynomial time. The Walsh analysis facilitates the exact calculation of low-order schema fitnesses (Goldberg 1989a), which the genetic algorithm is hypothetically estimating throughout its execution (Holland 1975). In Chapter 4, the genetic algorithm was shown to behave like local search inasmuch as it heavily samples local optima; however, the local surface structure of MAXSAT problems have exponentially many local optima in the form of plateaus, where a plateau is a region of the search space where points and one or more of their neighbors have identical fitnesses. The schema fitnesses reflect the local structure of the search space by offering very little information to guide the genetic algorithm. Ultimately, it is shown that the behavior of the genetic algorithm is guided only partially by the low order schema information, and that the genetic algorithm performs poorly due to its inability to explore the flat, local regions of the MAXSAT landscapes.

5.1 Boolean Satisfiability

Boolean Satisfiability was the first problem proven to be NP-Complete. SAT problems consist of literals, defined as variables or negated variables, that are combined together using **and** ($\wedge$) and **or** ($\vee$). Typically, SAT problems are presented in conjunctive normal form, which groups literals together into disjunctive clauses. A SAT problem is considered solved when an instantiation of variables is found such that the formula is true or it can be proven that no such instantiation exists. The typical form of SAT is a decision problem; every evaluation either returns a TRUE (1) or a FALSE (0). The optimization counterpart to SAT, which is used as the evaluation function for a genetic algorithm, is the MAXSAT problem. The MAXSAT evaluation sums together the individual truth values of clauses rather than performing an AND.

For example, a SAT problem in CNF form is:

$$(\neg x_2 \vee x_1 \vee x_0) \wedge (x_3 \vee \neg x_2 \vee x_1) \wedge (x_3 \vee \neg x_1 \vee \neg x_0)$$

and can be rewritten as a MAXSAT problem:

$$(\neg x_2 \vee x_1 \vee x_0) + (x_3 \vee \neg x_2 \vee x_1) + (x_3 \vee \neg x_1 \vee \neg x_0)$$

A special form of MAXSAT, MAXKSAT, requires that all clauses contain exactly K literals. Although MAX2SAT is NP-complete (Papadimitriou 1994), MAX3SAT problems will be considered here.

5.2 Problem Difficulty and SAT

Problem difficulty with respect to SAT has recently been associated with a **phase transition** (Cheeseman, Kanefsky, and Taylor 1991; Mitchell, Selman, and Levesque 1992). In general, a phase transition is a change that can be controlled or described by an **order parameter**. The order parameter is a variable whose modification causes a change to the specific phase of system. For example, lowering the temperature of water causes a phase transition to occur (i.e., water moves from a liquid to a solid state) at zero degrees Celsius, so temperature is the order parameter. Random SAT problems can be generated by randomly creating a specified number of clauses, which are typically generated by randomly choosing K variables and negating them with a probability of 0.5. For random SAT problems, the order parameter is the number of clauses in the expressions. As the number of clauses increases, the chance that the expressions are satisfiable abruptly drops, which constitutes a phase transition. Problems that are the most difficult tend to fall at the point in the phase transition where randomly generated SAT problems have approximately 50% satisfiable problem instances. For random 3SAT problems, this point occurs when there are approximately 4.3 times the number of variables in the expressions; however, this ratio can vary with the particular values of K.

```
Davis Putnam(C,v):
        C is a set of clauses, v is a set of variables
    IF C contains an UNSAT clause, return FALSE
    IF all clauses in C have been valued, return TRUE
    UnitPropagate: IF C contains any unit clauses, assign
                   the remaining unvalued variable so the
                   unit clauses are made TRUE
   Recurse:  Pick an unvalued variable
             Assign the variable to FALSE and simplify C
             IF DavisPutnam(C,v) THEN
                 return TRUE
             ELSE
                 Backtrack
                 Assign the variable to TRUE and simplify C
                 return DavisPutnam(C,v)
```

Figure 5.1: Overview of the Davis-Putnam algorithm.

Problem difficulty is relative; different algorithms may perform differently on the same problem. Problem difficulty with respect to the phase transition is determined by exact algorithms solving the SAT decision problem. An exact algorithm is used for two reasons. First, the only way to accurately detect a phase transition is if an algorithm can prove that problem instances are unsatisfiable. Second, problem difficulty is measured by the amount of effort it requires to locate a solution or prove that the problem cannot be satisfied. Although stochastic local search methods are simpler, more efficient, and can quickly solve many satisfiable problem instance, these algorithms cannot currently prove that problems are unsatisfiable (Selman, Kautz, and McAllester 1997).

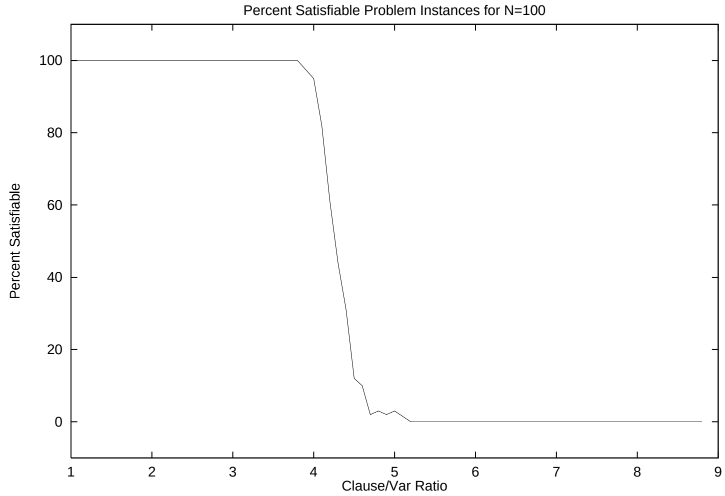
The traditional method for computing both the phase transition and the problem difficulty is to use the **Davis-Putnam** (DP) algorithm (Davis and Putnam 1960; Davis, Logemann, and Loveland 1962). An overview of the Davis-Putnam (DP) algorithm is given in Figure 5.1. The DP algorithm is an exact algorithm for solving the SAT decision problem. The algorithm is passed a set of clauses and a set of **unvalued** variables. Unvalued variables are Boolean variables that have not been instantiated with a TRUE or FALSE assignment. The first step of the algorithm is to check that there are no inconsistencies with the current set of valued variables, where an inconsistency occurs when there is an unsatisfied clause. Next, the DP algorithm expands **unit clauses**, where a unit clause is a clause containing only one unvalued variable. Suppose that we have a clause $(x_0 \vee \neg x_1 \vee x_2)$ and the variables $x_0 = FALSE$ and $x_1 = TRUE$, then the clause reduces to a unit clause because its truth assignment depends solely on x_2. Variables in a unit clause are always set so that the clause is made TRUE. Once all unit clauses have been expanded, a branch must be performed; so the procedure sets a variable to a value and recurses.

Figure 5.2a illustrates the phase transition for a large set of 100 variable 3SAT problem instances. The horizontal axis refers to the clause to variable ratio. Each point in the graph was generated using 100 problem instances and the points were generated in increments of 0.2. A finer grained increment, 0.1, was used to generate points between the clause/variable ratios 4.0 and 5.0. At the 4.3 clause to variable ratio, 49 percent of the problem instances are satisfiable. Figure 5.2b tracks the number of recursive calls required to solve the set of 100 variable SAT problem instances. As the clause to variable ratio increases, the amount of work required to decide the SAT problem instances also increases and peaks near the clause/variable ratio of 4.3.

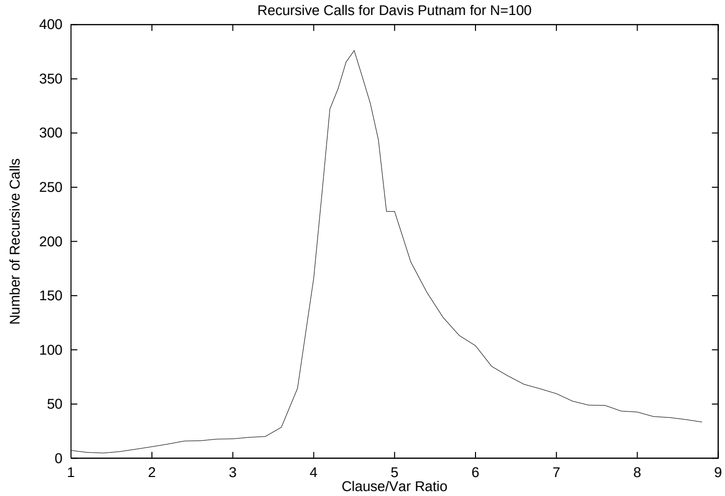
Although exact algorithms are guaranteed to solve the SAT decision problem, exact algorithms are computationally limited to solving small SAT problems. Currently, the problem size limit for exact algorithms is 700 variables (Selman, Kautz, and McAllester 1997). While local search algorithms currently cannot prove that a SAT problem is unsatisfiable, local search can solve satisfiable problem instances with thousands of variables.

The best known local search algorithm for solving satisfiable SAT/MAXSAT problems is **Walksat**. The Walksat algorithm is presented in Figure 5.3. Walksat is a bitclimbing local search algorithm that utilizes a form of random walk (Selman, Kautz, and Cohen 1993). Unlike DP, Walksat is manipulating complete solutions, where all variables have been valued. Starting from a random setting of all variables, Walksat partitions the set of clauses into those that are satisfied and unsatisfied. Walksat explores the search space by repeatedly making an unsatisfied clause satisfied. The heuristic that Walksat uses to make moves is to randomly choose an unsatisfied clause and flip the setting of one variable used in that clause. The variable choice is either greedy or it is random – the parameter setting corresponding to the probability of taking the random move is typically 0.5. The greedy move attempts to minimize the number of **breaks** that result from the bit flip; a break occurs when a clause that was previously satisfied but becomes unsatisfied when a bit flip is made.

Quite often, problem difficulty for genetic algorithms is treated separately from problem difficulty for local search algorithms because the two types of algorithms can perform dramatically different on different optimization problems. For the SAT problems, problem difficulty is typically measured by the number of recursive calls made by the DP algorithm. Gauging problem difficulty for Walksat, CHC and the ESGA is complicated by the fact that they are optimizing the MAXSAT



(a)



(b)

Figure 5.2: Phase transition and problem difficulty for 3SAT problems.

version of the problem instances; thus, they are allowed to run until they either solve the problem or reach some user-defined limit on evaluations. Furthermore, the genetic algorithms and Walksat measure evaluations differently.

For MAXSAT problems, Walksat evaluates a solution after a single bit flip while the genetic algorithms evaluate a solution after applying crossover and mutation, which usually results in multiple bit flips. This key difference between the algorithms creates a dilemma with respect to performing meaningful comparisons between the algorithms. The information used by Walksat to make a single bit flip should involve multiple evaluations, as evaluations have been defined for genetic algorithms. Since MAXSAT is not truly a blackbox optimization problem, even a steepest ascent move can be performed without needing a full evaluation for each flip.

Taking this difference into consideration, a small experiment was conducted to examine whether or not genetic algorithms could solve satisfiable MAX3SAT problems as well as Walksat. Walksat, ESGA and CHC were run on the set of *satisfiable* problem instances from a 100 variable set of

```
WALKSAT(maxtries, maxflips)

FOR i=1 to maxtries
    s = random binary string
    FOR j=1 to maxflips
        IF SAT(s) THEN return TRUE
        cl = a random UNSAT clause
        IF random() <= p THEN
           FLIP(cl, random()*NVARS(cl))
        ELSE
           FLIP(cl, MINBREAK(cl));
    END FOR
END FOR
```

Figure 5.3: The Walksat algorithm.

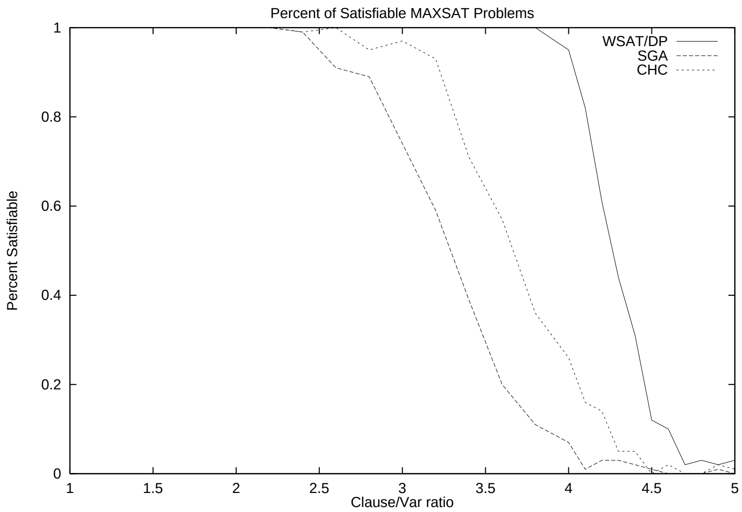


Figure 5.4: Phase transition for MAX3SAT problems with solutions found by ESGA and CHC.

MAX3SAT problems. Walksat was allowed 10,000 bit flips to locate the solutions and it was successful on all satisfiable problem instances. The limit on the number of bit flips for both genetic algorithms was 100,000 bit flips, which corresponded to approximately 20,000 evaluations.

The phase transition plot computed by the DP algorithm illustrates the ground truth on the percentage of satisfiable problem instances within the set of SAT test problems. Figure 5.4 plots the percentage of satisfiable problems solved by both ESGA and CHC at each of the clause/variable ratio points. Walksat, labeled WSAT in the plot, was also run on this same set of problems. Walksat solved all of the satisfiable problem instances 100 percent of the time, so it tracks the ground truth (i.e., the results of DP) perfectly. As the clause to variable ratio increases, the probability of success of both genetic algorithms degrades rapidly. Even though the genetic algorithms were both allowed more computational effort than Walksat, Walksat still outperformed both genetic algorithms. The quick degradation in behavior of the genetic algorithms near the phase transition is consistent with the hypothesis that problem difficulty for genetic algorithms is also associated with the phase transition phenomenon.

5.3 Epistasis, Problem Difficulty and MAXSAT

Measuring the epistasis, or bitwise nonlinearity, in MAXSAT problems can illustrate why genetic algorithms perform poorly. A method for studying the epistasis in a binary function is to use Walsh analysis (Goldberg 1989a; Goldberg 1989b; Reeves and Wright 1995). All binary functions can be represented as a weighted sum of **Walsh functions** denoted by ψ_j, where $0 \leq j \leq 2^L - 1$ with each Walsh function being $\psi_j : \mathcal{B}^L \rightarrow \{-1, 1\}$. The real-valued weights are called **Walsh coefficients**. Walsh functions are defined using a bitwise-AND of the function index and its argument. Operations on indices act on the binary representation. Let $\mathsf{bc}(j)$ be a count of the number of 1 bits in string j. The 2^L Walsh coefficients for function $f(i)$ can be computed by a Walsh transform:

$$w_j = \frac{1}{2^L} \sum_{i=0}^{2^L-1} f(i)\psi_j(i) \quad \text{where} \quad \psi_j(x) = (-1)^{\mathsf{bc}(j \wedge x)}$$

So, if $\mathsf{bc}(j \wedge x)$ is odd, then $\psi_j(x) = -1$ and if $\mathsf{bc}(j \wedge x)$ is even, then $\psi_j(x) = 1$.

The calculation of Walsh coefficients can also be thought of in terms of matrix multiplication. Let $\vec{f}$ be a column vector of 2^L elements where the i^{th} element is the value of the evaluation function $f(i)$. Similarly define a column vector $\vec{w}$ for the Walsh coefficients. If M is a $2^L \times 2^L$ matrix where $M_{i,j} = \psi_j(i)$ then:

$$\vec{w} = \frac{1}{2^L} \vec{f}^{\,T} M$$

An L-bit MAXSAT problem can be represented as a sum of C disjunctive clauses, f_i:

$$f(x) = \sum_{i=1}^{C} f_i(x)$$

where $f, f_1, f_2, ... f_C : \mathcal{B}^L \rightarrow \mathbb{R}$. Each clause evaluation, f_i, takes an L-bit string as input, but extracts and uses only K-bits in the calculation, where K is the length of the clause. This constraint means that each clause contributes to only 2^K Walsh coefficients (Rana, Heckendorn, and Whitley 1998).

Since the Walsh transform can be performed by a simple linear transformation, the Walsh transform of a MAXSAT problem can be treated as a sum of the Walsh transforms of the individual clauses.

$$W(f(x)) = \sum_{i=1}^{C} W(f_i(x))$$

Consider the example function consisting of a single clause $f(x) = \neg x_2 \vee x_1 \vee x_0$. This function can only be false when the assignment of bit values causes every literal to be false. Since this can happen only in one way, the vector representing the function values $f(x)$ is composed of all 1's except for a single 0.

The explicit calculation of the Walsh coefficients for the example function is obtained using the matrix form of the Walsh transform. The computation $\vec{f}^{\,T} M$ is shown in equation 5.1. Notice the single zero in the function vector for assignment $x_2 = 1, x_1 = 0, x_0 = 0$.

$$\vec{w} = \frac{1}{8}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 0.875 \\ -0.125 \\ -0.125 \\ -0.125 \\ 0.125 \\ 0.125 \\ 0.125 \\ 0.125 \end{bmatrix} \tag{5.1}$$

The fitness vector for any disjunctive clause must be 1 everywhere except for a single 0 where the clause is FALSE. Since ψ_0 is the average fitness, $w_0 = \frac{2^K-1}{2^K}$. The remaining Walsh functions evaluate to -1 exactly as often as 1. All but one value will cancel and this value depends on where the fitness is zero. Since the clause is disjunctive, the only time the fitness is zero is when all literals are simultaneously FALSE. Define the function $z = negvec(c, L)$ to take a clause description as input and return an L-bit string that makes the clause false. Variables included in the vector that are unused in the clause will be filled with 0's in z. The remaining Walsh coefficients can be written as:

$$w_j = -\frac{1}{2^K}\psi_j(negvec(c, L)) \qquad \forall j \neq 0$$

5.4 Observations about the Walsh Analysis of MAXSAT

The number of Walsh coefficients is exponential with respect to K; however, $K << L$ and is typically a bounded constant. The most commonly used value for K is three, which will be adopted for the remainder of the paper. There are only $7C$ calculations required to enumerate the set of nonzero Walsh coefficients for arbitrary MAX3SAT expressions. Normally the value of C is linear with respect to L; therefore, the set of nonzero Walsh coefficients is sparse and enumerable in polynomial time. Furthermore, the set of all nonzero Walsh coefficients are simply counts of variable uses over the set of clauses.

Consider a small MAXSAT function $f : \mathcal{B}^4 \rightarrow \mathbb{R}$ with $f(x) = f_1 + f_2 + f_3$ and

$$f_1 = (\neg x_2 \vee x_1 \vee x_0) \qquad f_2 = (\neg x_3 \vee x_2 \vee x_1) \qquad f_3 = (x_3 \vee \neg x_1 \vee \neg x_0).$$

The Walsh coefficient w_2 is computed for f as follows:

$$w_2 = -\frac{1}{8}\psi_2(negvec(f_1, 4)) - \frac{1}{8}\psi_2(negvec(f_2, 4)) - \frac{1}{8}\psi_2(negvec(f_3, 4))$$

The w_2 coefficient is a measure of the linear contribution of the Boolean variable x_1. All three clauses use the variable x_1 so they are all included in the calculation.

The Walsh function indices are masks that isolate variables or combinations of variables. The $negvec()$ function produces a mask corresponding to negated variables. When this vector and the Walsh function index are merged together by the bitwise-AND, the information extracted is the parity of the negation information for a particular subset of variables.

Thus, the Walsh coefficients for MAXSAT problems are nothing more than a constant $-\frac{1}{2^K}$ multiplied by a count over the uses of subsets of variables across the set of all clauses. The following method shows how to compute a simple count that can be used to calculate the nonzero Walsh coefficients for arbitrary MAXSAT problems.

Define $T(x,c)$ for variable x and clause c as follows:

$$T(x,c) = \begin{cases} 1 & \text{if } x \text{ is present} \\ -1 & \text{if } \neg x \text{ is present} \\ 0 & \text{otherwise} \end{cases}$$

Define $use(c)$ to return a subset of all variables used by a clause c. Let $S_{v,c}$ be a sign (± 1) determined for a clause c and a subset of variables v as follows:

$$S_{v,c} = \prod_{x \in v} T(x,c) \quad \text{where} \quad v \in \bigcup_{c \in f} (\mathbb{P}(use(c))$$

where $\mathbb{P}$ represents the power set.

Then the Walsh term associated with a particular variable combination v is

$$w_v = \frac{-1}{2^K} count_v \quad \text{where} \quad count_v = \sum_c S_{v,c}$$

For order-1 Walsh coefficients, these counts compute the difference between the number of times a variable is used positively and the number of times a variable is used negatively.

5.4.1 Walsh Coefficients and Schema Fitnesses

Walsh coefficients, when available, can be used to calculate the exact average fitnesses of low order schemata. Low order schema averages are sometimes used to try to understand the behavior of genetic algorithms. For order-1 schemata such as `**0**` and `**1**`, if the population of a genetic algorithm is distributed such that strings with a `0` in the third position are on average better than those strings that contain a `1` in the third position, then more samples should be allocated to the schema `**0**` in the next generation. In some sense genetic algorithms stochastically hill-climb in the space of schemata rather than in the space of binary strings. As with other hill-climbing search strategies, a genetic algorithm can perform poorly if there is an absence of information or if that information is misleading (deceptive).

It is usually difficult to statically analyze large functions to determine whether or not useful low order schema relationships exist. To compute the *exact* average fitness, an order-k schema in a function defined over $\mathcal{B}^L$ requires 2^{L-k} points in the search space to be enumerated. Since k is small for low order schemata, computing the average fitness of a low order schema quickly becomes intractable as L increases. In general, computing the exact average fitnesses of low order schemata requires exponential time for most large problems. However, we can use Walsh coefficients to efficiently compute low order schema averages. Functions α and β are defined on a schema, h (Goldberg 1989a):

$$\alpha(h)[i] = \begin{cases} \mathtt{0} & \text{if } h[i] = * \\ \mathtt{1} & \text{if } h[i] = \mathtt{0} \text{ or } \mathtt{1} \end{cases}$$

$$\beta(h)[i] = \begin{cases} \mathtt{0} & \text{if } h[i] = * \text{ or } \mathtt{0} \\ \mathtt{1} & \text{if } h[i] = \mathtt{1} \end{cases}$$

For example, consider $h = \mathtt{**01**}$. Because schema h is a subset of strings, its fitness is the average fitness of all strings that belong in that subset. Rather than enumerating the subset of strings, the fitness of h can be computed using Walsh coefficients:

$$f(h) = \frac{1}{|h|} \sum_{x \in h} f(x) = \sum_{j \subseteq \alpha(h)} w_j \psi_j(\beta(h))$$

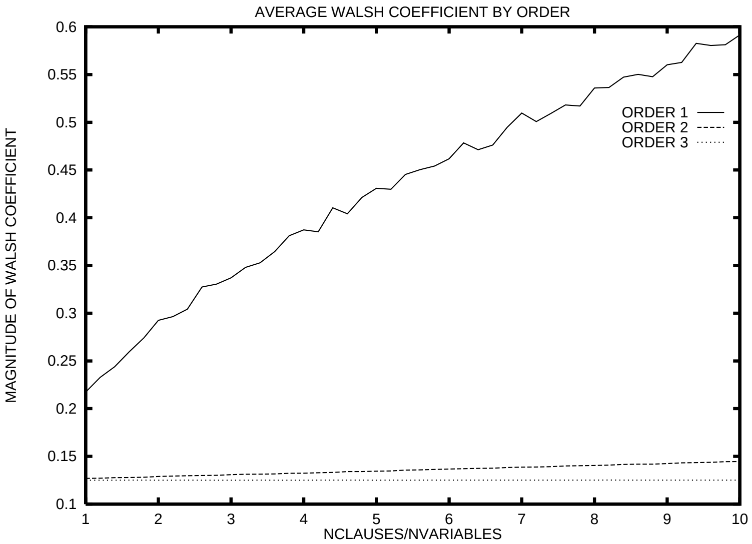


Figure 5.5: Average magnitudes of nonzero Walsh coefficients (excluding w_0).

For all $j \subseteq \alpha(h)$, $\mathsf{bc}(j) \leq \mathsf{bc}(\alpha(h))$. This implies that the highest order Walsh coefficient used in the formula has the same order as the schema.

The average fitness of schema $h = $ `**01**`, depends only on Walsh coefficients w_0, w_4, w_8 and w_{12}. Since $\alpha($`**01**`$) = $ `001100`, the subset generated using α is:

{`001100, 001000, 000100, 000000`}

corresponding to the indices $\{12, 8, 4, 0\}$. The string generated by $\beta($`**01**`$) = $ `000100`, so $\psi_4(000100) = -1$, $\psi_8(000100) = 1$ and $\psi_{12}(000100) = -1$. The sign of w_0 is always `1`. So, $f($`**01**`$) = w_0 - w_4 + w_8 - w_{12}$.

Order-1 schema are computed using Walsh coefficients w_0 and w_i, where w_i measures the contribution of a single bit (Goldberg 1989a); thus, the order-1 schema averages are only a constant offset of the order-1 Walsh coefficients. For instance, the average fitness of the L-bit schema $f($`**...**1`$) = w_0 - w_1$. The fitnesses of all order-1 schemata can be determined by w_0 plus or minus an order-1 Walsh coefficient regardless of L. Because the order-1 Walsh coefficients for MAXSAT problems are proportional to the difference between the number of times a variable is used positively and the number of times a variable is used negatively, all order-1 schema competitions are *decided* by the following *naive* heuristic:

> If a variable occurs positively more often than negatively, the variable should be set to TRUE, otherwise set the variable FALSE.

Genetic algorithm behavior is theoretically guided by low-order schemata; thus, it relies on the existence of low-order schemata that ultimately lead to the global optimum (i.e., not deceptive). If the order-1 schemata are not misleading, then the naive heuristic *decides* the MAXSAT problem. In practice, all non-trivial MAXSAT problems are *deceptive:* the schema information is inherently misleading. We might also ask if there is useful order-2 or order-3 schema information. Figure 5.5 shows the average magnitude of *nonzero* Walsh coefficients for a large set of randomly generated 100 variable MAX3SAT problems. The horizontal axis is the ratio of clauses to variables; steps were taken in increments of 0.2. Each point in the graph is the average of 30 problem instances. Each line tracks the average magnitude of all nonzero Walsh coefficients for order-1, order-2, and order-3 interactions.

The plot indicates that the order-1 terms are relatively large compared to the order-2 and order-3 Walsh coefficients. Since order-2 and order-3 schema averages include order-1 Walsh coefficients, it follows that these schema averages will also tend to be strongly correlated with the order-1 schema averages. Furthermore, since all of this information can be computed in polynomial time and it is fairly consistent, this information cannot lead to the global optimum if $P \neq NP$. Therefore, MAXSAT problems with a bounded, maximum clause length (K), must be deceptive to a genetic algorithm. Also notice that many of the Walsh coefficients are the same (either 0 or 0.125), and thus many schema averages will be identical. Not only is the MAXSAT landscape misleading, it is also relatively flat.

5.5 Local Surface Structure of MAXSAT

The local surface structure of random MAXSAT problems is characterized by large regions of similar fitnesses (Selman and Kautz 1993; Frank, Cheeseman, and Stutz 1997). The large flat regions (plateaus) within the MAXSAT landscape can be characterized in several ways (Frank, Cheeseman, and Stutz 1997). Plateaus can occur in the form of a step where there is a large area of identical fitness, but it is connected to another region with higher fitnesses. This landscape feature is referred to as a **bench**. A bench is simply a plateau with exits to one or more better points in the search space. A second type of plateau is one that is **locally optimal**[1] because no improving move is available anywhere on the plateau.

Some of the notable observations made regarding the plateau structure in MAXSAT problems are (Frank, Cheeseman, and Stutz 1997):

- The number of highly fit locally optimal plateaus grows with problem size.
- Highly fit locally optimal regions tend to be small.
- Globally optimal regions tend to be smaller than locally optimal regions.
- Benches tend to be 10-30 times larger than locally optimal regions.
- Benches with higher fitness tend to have fewer exits.

Plateaus with high fitness tend to be locally optimal plateaus rather than benches. This means that downhill moves must be taken to move from a locally optimal plateau to ultimately reach a more promising region of the search space. Benches tend to be larger and have a varying number of exits depending on the fitness of the bench. However, the local structure of a MAXSAT problem depends on how the MAXSAT problem was generated; different types of MAXSAT problems exhibit different forms of the plateau phenomenon.

The plateaus in MAXSAT problems can be partially explained by the fitness distribution of MAXSAT problems. The fitness distribution of MAXSAT problems includes many points of equal fitness. If a landscape is structured so that neighboring points have similar fitnesses, then this property translates to a flat area. Consider the plots shown in Figure 5.6. The topmost graph corresponds to a function with fitness values chosen from a uniform fitness distribution. In fact, this function is simply a permutation of fitness values. The second plot illustrates how the local surface of the function can change when fitnesses drawn from a Gaussian distribution are mapped

[1]Frank et al. (1997) treated MAXSAT as a minimization problem where the goal was to minimize the number of unsatisfied clauses in an expression. What he referred to as a **local minimum** will be referred to as **locally optimal**.

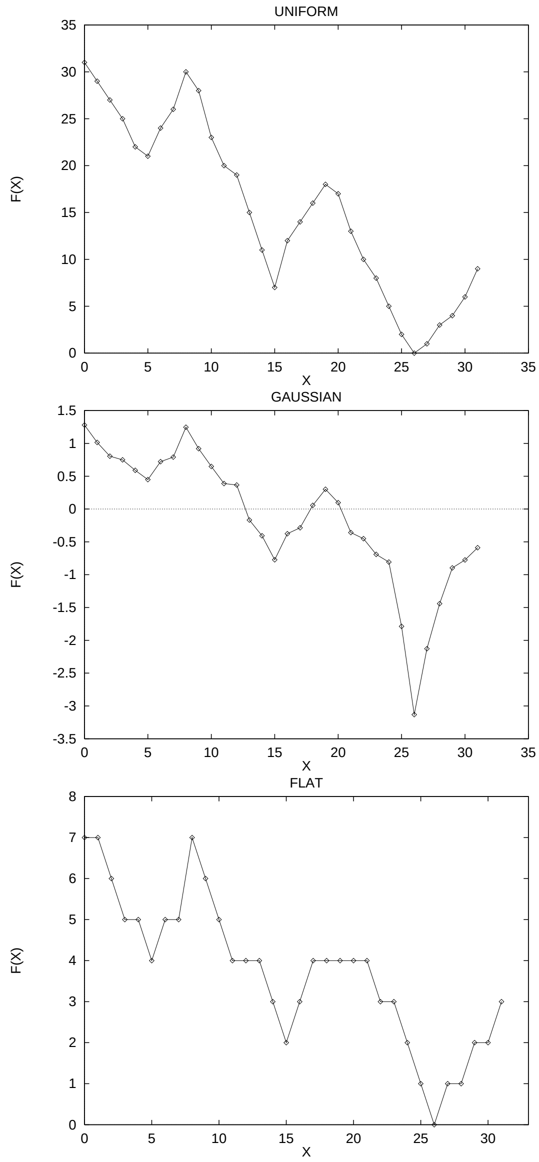


Figure 5.6: The topological effects of fitness distributions.

onto the permutation. Finally, the third plot illustrates what happens to the function when the fitness distribution contains numerous duplicate fitness values. The function becomes flat because of the limited range of fitnesses. Heckendorn et al. (1998) has shown that the plateaus can also be detected by the restricted set of Walsh coefficients computed for MAXSAT problems.

5.5.1 The Fitness Distributions for MAX3SAT Problems

The plateau observations made by Frank et. al. (1997) can be partially explained by examining fitness distributions of MAXSAT problems. Recall from Chapter 3 that expected number of local optima can be computed analytically for all discrete functions based on the size of the search neighborhood. When a set of discrete functions defined for a fitness distribution with a large number of points sharing the same fitness, the expected number of local optima increases. The reason for this increase is that plateaus are created when points with the same fitness occur as neighbors. The

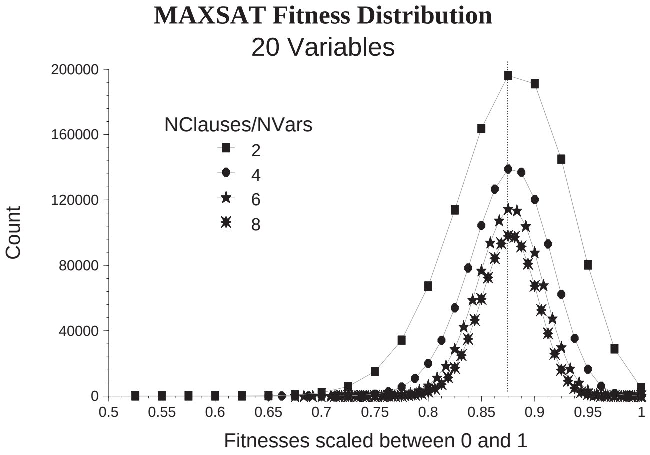


Figure 5.7: Fitness distributions for 20 variable problem instances.

fitness distributions for random MAXSAT problems are illustrative of the distributions one should expect for problems with many large plateau regions.

To examine how solutions are distributed for MAX3SAT problems, the specific fitness distributions of enumerable MAX3SAT problems are represented by histograms. Figure 5.7 is the composite result of sets of 50 randomly generated 20 variable MAX3SAT problems; the composite histogram was calculated by averaging the 50 individual histograms. The figure shows four composite histograms for clause to variable ratios of 2, 4, 6, and 8. The x-axis corresponds to the *proportion* of satisfied clauses, while the y-axis represents the frequency that any particular value occurred. The scaling of the outputs means that the optimum occurs at 1.0, and the mean fitness is 0.875 for all problem sets. The distributions appear to become shorter and narrower as the clause/variable ratio is increased. However, this is an effect of normalizing the range of values between zero and one; the number of bins in the histograms increase with the number of clauses while the number of points, 2^{20}, remains constant. Note that these histograms are the average fitness distributions of many problem instances; thus, the histogram for any single problem instance will vary.

The set of histograms illustrates that the fitness distributions are skewed to the left of the mean fitness of 0.875; however, the bulk of the MAXSAT evaluations lie near the mean. Consequently, plateau regions of moderate fitness have many opportunities to locate improving solutions (i.e., they are benches) while plateau regions of high fitness have fewer opportunities to locate improving solutions (i.e., they are locally optimal plateaus). As the problems scale up in terms of the number of clauses, the fitness distributions exhibit more variation, but the solutions are still clustered near the mean solution. The skew tends towards zero but still remains negative so the majority of fitness values are higher than $\frac{7}{8}$ optimal. However, the actual number of globally optimal solutions is decreasing as the clause to variable ratio increases; in fact, the number of global and local optima decreases at an exponential rate as illustrated in Figure 5.8. This figure shows the number of points falling into the 1.0 bins in the histograms (on a log scale). Since the fitness distributions of the MAXSAT problems are moving closer to the mean solution of 0.875 as the clause to variable ratio increases, the tails at both extremes are also moving closer to the mean. Since there is a ceiling effect at the global fitness of 1.0, the number of globally optimal solutions is decreasing rapidly as the number of clauses is increased.

The Walsh coefficients can be used to indirectly study the fitness distribution of MAXSAT problems, or any other problem whose Walsh coefficients are known. Appendix B derives calculations for computing the mean, variance, skew and kurtosis for MAXSAT fitness distributions using

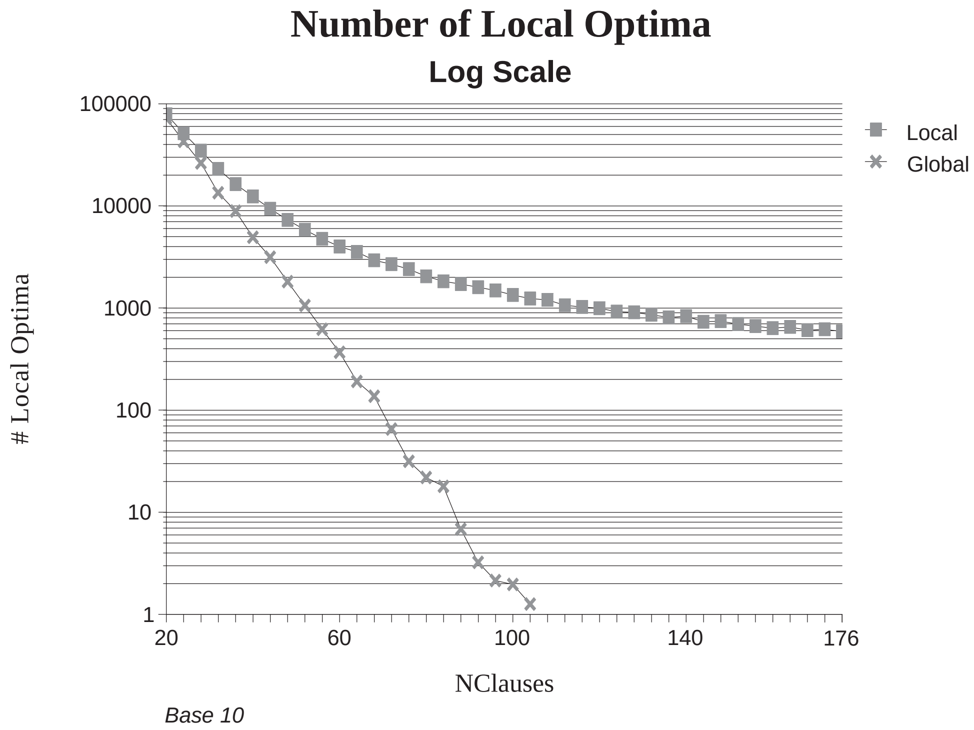


Figure 5.8: Number of local optima for 20 variable MAX3SAT problems.

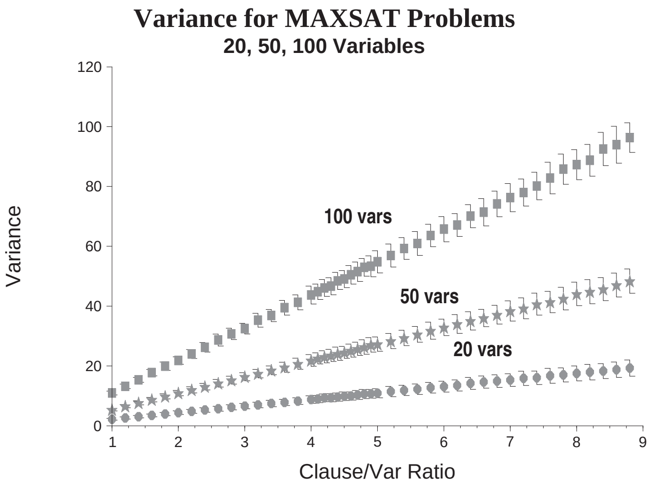


Figure 5.9: Variances of fitness distributions for MAX3SAT problems.

Walsh coefficients. The summary statistics can be used to partly explain the observations about plateau structure in non-enumerable MAXSAT problems.

Figures 5.9 and 5.10 respectively illustrate the average variance and skew of the fitness distributions for 20, 50 and 100 variable MAX3SAT problems. The plots for both the 20 and 50 variable problems were averaged over 500 problem instances. We used 100 problem instances at each point for the 100 variable problems. The region representing clause/variable ratios between four and five was more heavily sampled by using increments of 0.1, while the remaining points were computed at increments of 0.2. Since kurtosis calculations are $\mathcal{O}(n^4)$, we limited our kurtosis calculations to only the set of 500 20 variable problems. The kurtosis was approximately 3.0 regardless of the

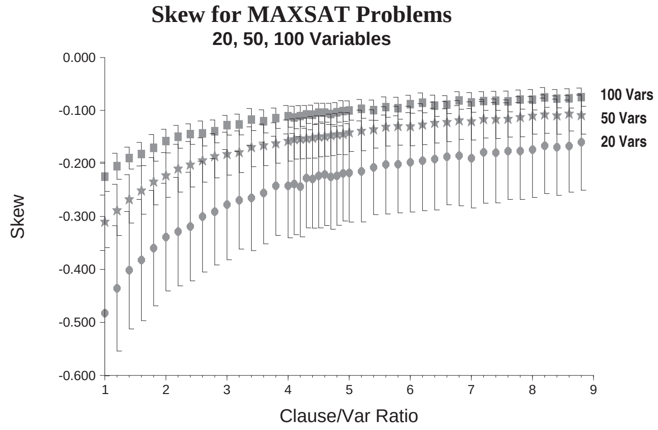


Figure 5.10: Skew of fitness distributions for MAX3SAT problems.

clause/variable ratio.

The summary statistics for problem instances of randomly generated MAX3SAT problems indicate that the majority of solutions are actually higher than average. For over one thousand MAX3SAT problem instances, the skew was negative. While there are cases of other MAX3SAT problem instances with positive skew, e.g., Crawford's parity function learning problems from the DIMACS benchmark problem set (Trick and Johnson 1993), the randomly generated MAX3SAT problems have negatively biased fitness distributions which suggests that it is easy to locate above average solutions.

Since the majority of fitness values are better than average, it also seems intuitive that these fitnesses will create large benches with a high number of exits to better regions; however, as the quality of solutions increases, there are fewer possible exits available because the number of possible improvements is decreasing. The drop in the number of globally optimal solutions can explain why plateaus of higher fitnesses tend to be small, locally optimal regions that do not contain exits to regions of higher fitness. Furthermore, since the variance of the fitness distribution increases both with the number of clauses and number of variables, the observation that the size of plateaus decreases both with number of clauses and number of variables is consistent with this trend.

5.6 Empirical Verification

Problem difficulty for genetic algorithms is often associated with misleading schemata (i.e., schemata that have higher than average observed fitnesses but do not contain the global optimum) (Whitley 1991; Goldberg 1989c). The analog to plateaus in the context of schemata are schema averages that are similar or even identical. Considering the limited set of Walsh coefficients, it would appear that many schemata should share identical fitnesses. In other words, just as the space appears flat to Walksat, the space will also appear flat to a genetic algorithm and should be inherently misleading (deceptive). In addition, the clustering of similar fitnesses, which causes the plateaus and benches at the local level, also influences schema fitnesses. Once the genetic algorithm is drawn towards those regions by lower order schema fitnesses, the traditional genetic algorithm is incapable of exploring the plateaus to locate solutions. This section examines the convergence behavior of both Walksat and ESGA to illustrate the relationship between the local surface structure and the schema information.

The Walsh analysis provides theoretical evidence to support the claim that genetic algorithms

Number of Solutions Found (max=30)											
C/V Ratio	4.0	4.1	4.2	4.3	4.4	4.5	4.5	4.7	4.8	4.9	5.0
ESGA	3	16	8	0	1	0	1	1	1	3	20
Walksat	30	25	23	3	27	6	11	17	28	29	30

# Global Optima in Each Problem											
# Solutions	1,256,358	578,928	12,536	80	1,962	256	5,195	8	128	1,466	8,448

Table 5.1: Number of solutions found by ESGA and Walksat on 11 100 variable MAX3SAT problems.

should not perform well in the MAX3SAT domain due to the lack of schema information. The experiment described in section 5.2 illustrated that problems occurring near the phase transition are difficult for genetic algorithms. However, the reason why MAXSAT problems are difficult for genetic algorithms may be due to misleading schema information or a flat local surface structure or both. If the genetic algorithm is being misled by schemata, then its convergence behavior should be strongly influenced by the low-order schema relationships, which can be computed exactly using Walsh analysis. If the genetic algorithm is failing to optimize MAXSAT problems because it is unable to explore the local (flat) surface of the landscape, then the convergence behavior of the genetic algorithm should deviate from the low-order schema relationships and simply become stuck on a plateau. To examine how a genetic algorithm is affected by schema relationships and/or local surface structure in the random MAXSAT domain, the convergence behavior of a Simple Genetic Algorithm (ESGA) and Walksat were compared across several random MAX3SAT problem instances.

The ESGA was run with tournament selection using a tournament size of four, elitism, population size of 500, mutation rate 0.1, and probability of crossover of 0.6. Walksat was run using a 0.5 probability of taking a random move. Since Walksat and ESGA are not exact algorithms, the test problems were chosen to be 100-variable satisfiable problem instances that were randomly generated. The detailed examination of the convergence behavior of the genetic algorithm required that a small number of problems be used, so one problem was used for the clause to variable ratios of 4.0 through 5.0 at increments of 0.1 for a total of 11 test problems. These particular clause to variable ratios were chosen because they span the phase transition for MAX3SAT.

Table 5.1 lists the results of running ESGA and Walksat 30 times on each of the test problems. Walksat measures effort by the number of single bit flips that are taken during search while ESGA measures effort by the number of full evaluations, which usually involve multiple bit flips. For this experiment, the ESGA and Walksat were allowed to run for differing amounts of time to allow each algorithm to sufficiently explore the search space. ESGA had a population size of 500 and was allowed to run for 200 generations which means the genetic algorithm was allowed to sample $100,000$ points for each run. As will be seen in the next section, this was ample time for ESGA to converge. Walksat was allowed to run for $2,000$ bit flips per run. Although it would seem that Walksat was allowed only a fraction of the maximum effort alloted to the ESGA, the results illustrate that ESGA was still unsuccessful at locating a global optimum across the set of test problems.

Table 5.1 also lists a count of the number of global optima in each of the problems. The previous section illustrated that the number of global and local optima decays exponentially as the clause to variable ratio increases. As the clause to variable ratio approaches the phase transition, the number of local optima also becomes countable for an exact algorithm such as DP. The last row of the table was generated by running a variant of the Davis-Putnam algorithm that was designed to fully explore the search space and count the number of satisfiable solutions. Since the DP algorithm

Solution Quality												
C/V Ratio		4.0	4.1	4.2	4.3	4.4	4.5	4.5	4.7	4.8	4.9	5.0
ESGA	μ	1.00	0.60	1.26	2.36	1.60	2.90	2.06	1.76	2.56	3.10	1.03
	σ	0.45	0.72	1.04	0.85	0.85	0.71	1.11	0.89	1.07	1.37	1.65
Walksat	μ	0.00	0.16	0.26	0.93	0.16	0.96	0.63	0.43	0.13	0.06	0.00
	σ	0.00	0.37	0.52	0.36	0.53	0.61	0.49	0.50	0.50	0.36	0.00
# Global Optima in Each Problem												
# Solutions		1,256,358	578,928	12,536	80	1,962	256	5,195	8	128	1,466	8,448

Table 5.2: Solution quality found by ESGA and Walksat on 11 100 variable MAX3SAT problems.

can solve hard unsatisfiable 100 variable problem instances, it is also capable of counting optima when the number of local optima is reasonably small (such as a few million).

The trend in terms of the number of times Walksat was able to solve each problem is strongly correlated with the number of global optima in each problem. To a lesser extent, the ability of the genetic algorithm to solve each problem is also related to the number of global optima in each problem. Since the MAXSAT search space is known to be made up of many flat plateaus and benches (Frank, Cheeseman, and Stutz 1997), the success of Walksat is related both to its ability to explore the flat areas and the density of global optima. Furthermore, the overall solution quality of the final "best" solutions found by each algorithm also varied with the number of global optima. Table 5.2 presents the mean and standard deviation of the solutions found by ESGA and Walksat on the set of test problems. Both algorithms were designed to minimize the number of unsatisfied clauses. The table illustrates that there was very little variation in terms of the quality of the solutions found by Walksat while there was more variation in the quality of the solutions found by ESGA. Note that the solution quality for both algorithms is affected by the number of global optima in each problem: when there are many global optima, both algorithms perform better than when there are few global optima.

5.6.1 Distances between Convergence Points

The ESGA performs notably worse than Walksat on the small set of test problems. The purpose of this section is to determine whether the inferior performance of the genetic algorithm can be attributed to the misleading schema information in the MAX3SAT problems or if the genetic algorithm is simply unable to maneuver through the plateaus in the landscape. The set of convergence points for ESGA and Walksat was generated from the 30 different runs on each test problem. For ESGA, a convergence point was considered to be the *best* member of the population, with ties broken randomly, after generation 200. For Walksat, the convergence point was the point visited after Walksat performed its $2,000^{th}$ bit flip. When Walksat converged to a global optimum, the convergence point was that optimum since search technically terminates when a global optimum is reached.

The distances between convergence points coupled with fitness information can indicate whether or not there are large regions of the space that ESGA and Walksat find particularly attractive. The Hamming distances between all pairs of convergence points for each test problem were collected and a histogram was created to indicate how those distances were distributed. There were 30 convergence points and thus 435 pairings of convergence points. The Hamming distances between convergence points were computed for the test problems with clause to variable ratios of 4.0, 4.3 and 5.0. Large Hamming distances between convergence points and low variance in solution quality indicates that the algorithm encountered a large plateau or several large plateaus with similar fitnesses. Small Hamming distances between convergence points and low variance in solution quality indicates that the algorithm encountered a small plateau.

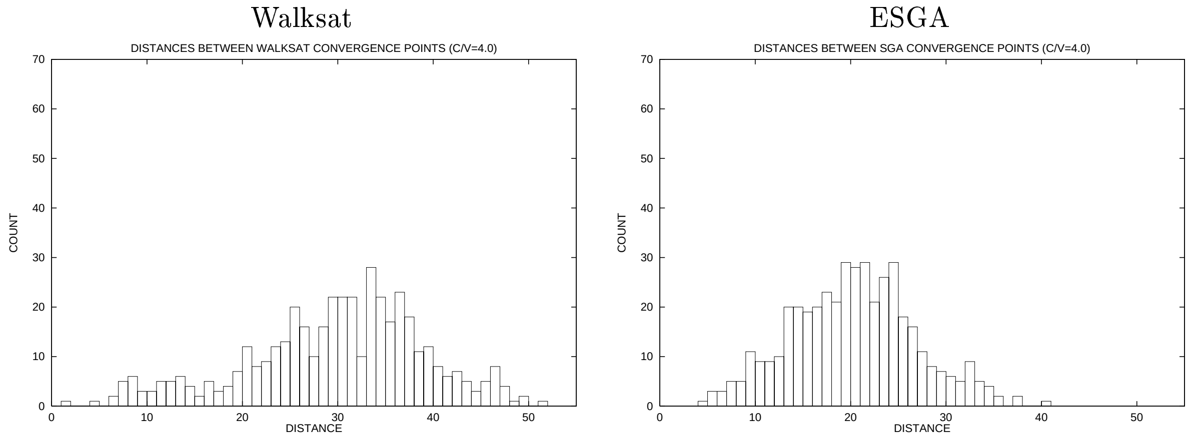


Figure 5.11: Distance between convergence points on the 400 clause problem.

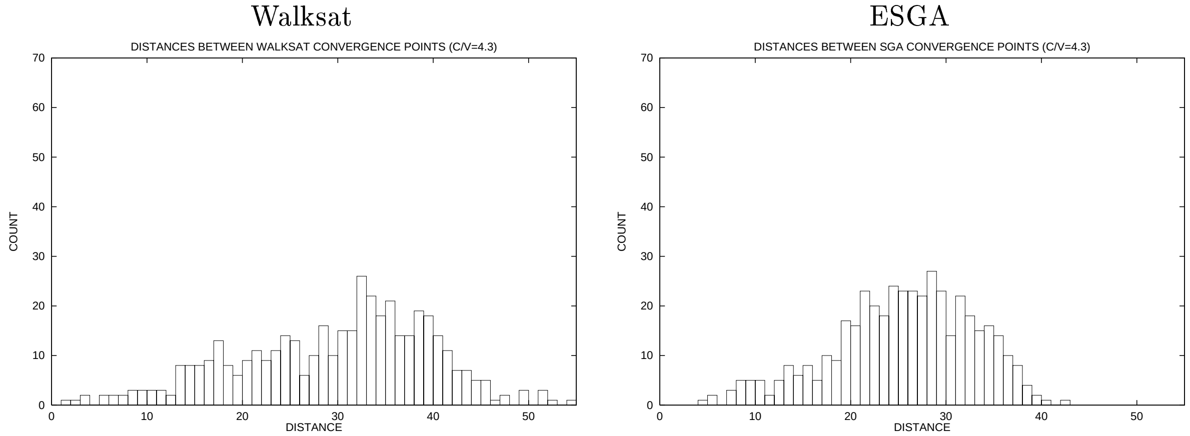


Figure 5.12: Distance between convergence points on the 430 clause problem.

Figure 5.11 shows the distribution of distances between the set of convergence points for Walksat and ESGA on an underconstrained problem with the clause to variable ratio of 4.0. Since Walksat solved this problem 30 times, all convergence points for Walksat are a satisfying solution. Because there were over 1.5 million global solutions, it is not surprising that the Hamming distance between solutions, is very large. In some sense, this problem is easy for local search because there are solutions scattered throughout the search space. Despite the large number of global optima, the ESGA was only able to solve this problem optimally three times, while it was able to locate near optimal solutions 27 times; ESGA found solutions with one unsatisfied clause in 24 runs and two unsatisfied clauses in three runs.

The histograms in Figure 5.12 present the Hamming distances between the convergence points for a hard problem with the clause to variable ratio of 4.3. The shape of the Hamming distance histograms is not dramatically different from those in Figure 5.11. The distribution of Hamming distances for Walksat tend to be flatter than the ESGA, indicating that Walksat explored the search space more thoroughly than ESGA. However, given the amount of exploration done by Walksat and the fact that there were only 80 global optima in the space, it also appears that there are many highly fit regions that are distant and do not contain global optima. In both histogram pairs, the shape of the histogram for ESGA is more peaked than that of Walksat. This additional peakedness

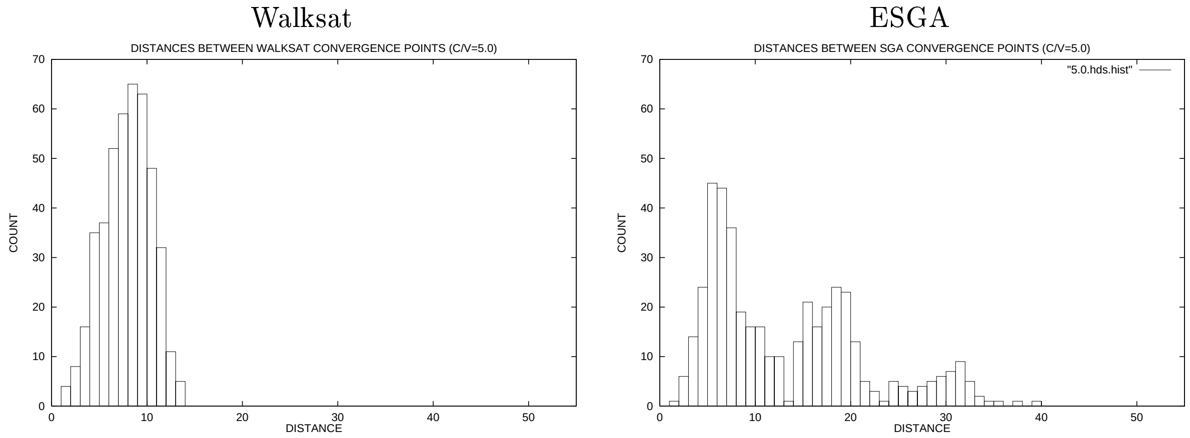


Figure 5.13: Distance between convergence points on the 500 clause problem.

of the histogram supports the hypothesis that the low-order schema relationships are problematic to the genetic algorithm; however, given the distance variation between convergence points, the local plateau structure was also problematic.

The last figure, Figure 5.13, presents the distributions of Hamming distances between convergence points on an overconstrained problem with a clause to variable ratio of 5.0. The histogram for ESGA is multimodal. The histogram for Walksat shows that the convergence points were very close to one another. Walksat solved all problem instances, so there is at least one subset of the solutions that are clustered together. The ESGA histogram appears to have three peaks. The leftmost, largest peak in the histogram corresponds to the distances between the global solutions found by the ESGA. The global solutions found by the genetic algorithm never differed from one another by more than 10-bits. The center peak corresponds to the distances between the convergence points and the moderately fit convergence points with evaluations of one, two, or three clauses unsatisfied, while the last peak corresponds to the distances between the least fit convergence points with evaluations of four and six clauses unsatisfied. This multimodal set of distances illustrates that the genetic algorithm was attracted to several distinct regions in the search space and that the genetic algorithm became trapped on these plateaus.

In general, the Hamming distances between convergence points indicates that ESGA and Walksat were converging to points that were different from one another but they were still closer together than a randomly selected set of points. The fitnesses of the convergence points showed little variation. For the underconstrained problem and hard problem instance, both algorithms converged to highly fit points that were scattered throughout the search space. This behavior is consistent with the argument that the MAXSAT landscape is very flat and there exist large plateaus and benches throughout the search space. The overconstrained problem had small, distinct regions that attracted each of the algorithms. In general, the genetic algorithm did not reliably converge to a single region of the search space, so the plateau phenomenon appears to be problematic for the genetic algorithm. On the other hand, the genetic algorithm and Walksat both tended to converge to points that were similar (less than 30 percent of the bits were different from other convergence points). This behavior indicates that low-order schemata could provide a very rough indication of where the global optimum is located; however, the algorithms would be required to search a significant portion of the space locally in order to find the global optimum.

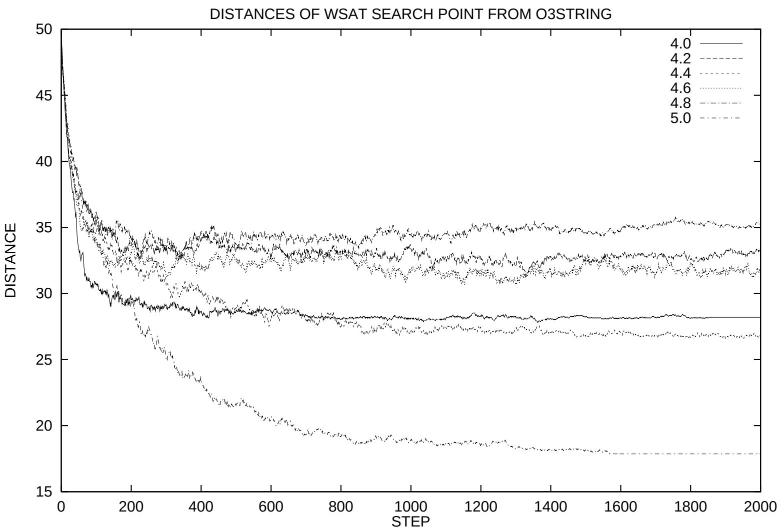
5.6.2 Where the Low Order Schemata Lead ...

This section examines whether or not the convergence points for the algorithms were consistent with the low-order schema information in the MAX3SAT problems. Define the **O3string** to indicate the string that is most consistent with the low order schema information. This string is constructed by enumerating all possible order-3 partitions and ranking the schemata according to their fitness. For each partition, the schema with the highest fitness contributed "votes" for its fixed bit positions. The votes were tallied for each bit position across all partitions, and the O3string was created based on the votes. For instance, if the order-3 schemata with a `1` bit in the first bit position were ranked higher than the schemata with a `0` bit in the first bit position, then the O3string would contain a `1` bit in the first bit position. The order-1 information was not used to generate the **O3string** because a large number of the bit positions had Walsh coefficients of zero and would result in a partially specified string. The remaining positions that were specified by order-1 schema information were identical to the corresponding values in the **O3string**. In all cases, the **O3string** was not a global optimum, which means the low-order schema information is deceptive for this set of MAX3SAT problems.

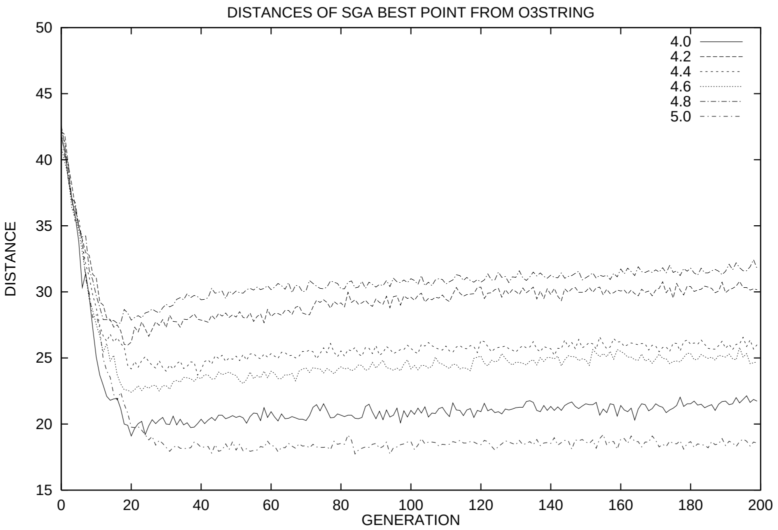
Figure 5.14 is a plot of the average distance (over 30 runs) of the convergence points and the O3strings. For simplicity of viewing, results for only six of the 11 problems were plotted; however, the remaining problems exhibited similar convergence curves. The x-axis on the plot for Walksat is numbered from zero to $2,000$, which corresponds to each step taken by Walksat. The x-axis on the ESGA plot is numbered zero to 200 and represents each generation of execution. While the two plots look similar, the scale varies on the x-axis.

This plot illustrates that *both* ESGA and Walksat moved to a region that was partially consistent with the O3string. These graphs illustrate that search initially moved towards the O3string which would normally indicate that the low order schema was guiding the search. However, since Walksat uses a single point to search, one can hardly say that it is processing schemata. What this plot says is that the regions of the search space containing highly fit points are related to the low order schema fitnesses, which are directly related to the naive heuristic for solving SAT problems. However, the true convergence points were still rather distant, approximately 15 to 35 bits away from the O3string. At a global level, the schema information was fairly consistent with the convergence points for both Walksat and ESGA, but this information is deceptive because it does not lead to the global optimum. The deception is not the only source of problem difficulty for the genetic algorithm since the ESGA remains relatively far from where the low order schemata lead. The problem difficulty for the ESGA on the MAX3SAT problems comes from the need to explore the region at a local level.

In Chapter 4, crossover distances were examined to illustrate that crossover had limited exploratory power. Figure 5.15 shows the size of the steps taken by crossover through the ESGA search on the MAX3SAT problems. By the 50^{th} generation, crossover is exploring a very low-order neighborhood (approximately two to four bits). At this point, ESGA is primarily behaving as a local search algorithm, but with no specific mechanisms to exploit the domain knowledge. On the other hand, Walksat is guiding its exploration using the heuristic that it will always flip a single bit and cause an unsatisfied clause to be satisfied. This heuristic allows Walksat to explore the plateau regions more thoroughly and efficiently than the ESGA. To summarize, the genetic algorithm is led to a region that is partially consistent with the low order schemata; however, its ability to explore that region is impaired because it has no explicit mechanisms for exploring plateaus.



Walksat



ESGA

Figure 5.14: Distances between ESGA and Walksat convergence points and O3string.

5.7 Summary

Genetic algorithms have previously been shown to perform poorly in the MAX3SAT domain (Frank 1994; Lee, Koh, and Nakakuki 1994). This chapter analytically shows why genetic algorithms should perform poorly using this evaluation function. The MAX3SAT evaluation function can be analyzed using Walsh Analysis. A Walsh analysis can provide useful information about schema relationships. The MAX3SAT problem is a special class of problems in which Walsh analysis can be performed even for non-enumerable problem instances. The examples of Walsh analysis for large problems illustrates that there is little useful information at the schema level for MAX3SAT problems. In fact, the order-1 schema fitnesses offer no more information than a naive heuristic:

> If a variable occurs positively more often than negatively, the variable should be set to TRUE, otherwise set the variable FALSE.

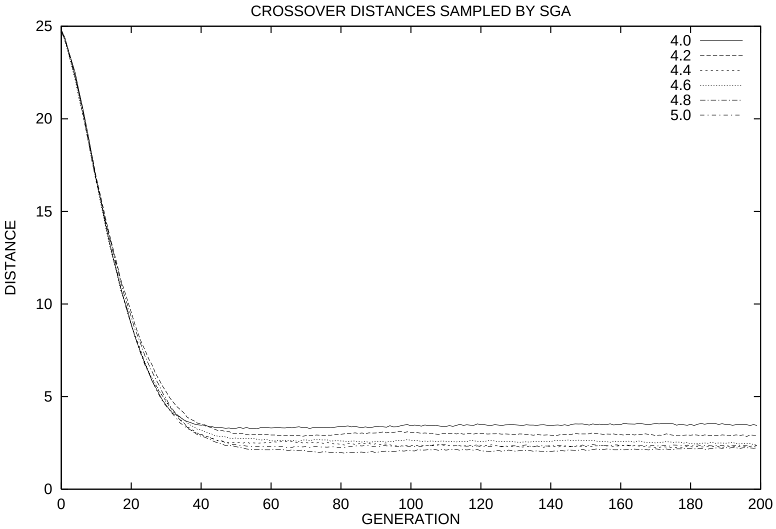


Figure 5.15: Crossover distances sampled by ESGA.

Since the naive heuristic cannot reliably lead to the global optimum unless P=NP, this information must be misleading.

This chapter illustrates that there is a relationship between local surface structure and schema fitnesses. Just as the local surface structure of MAX3SAT problems were flat, so were the schema fitnesses. Furthermore, both the local search algorithm and the genetic algorithm were drawn to a region of the search space that was partially consistent with the low-order schema fitnesses. However, much of the problem difficulty associated with MAX3SAT is that the highly fit plateaus are difficult to explore. Both the genetic algorithm and local search algorithm were drawn to the same regions of the search spaces during execution; however, the inability of an ESGA to efficiently explore plateaus put the genetic algorithm at a serious disadvantage.

Chapter 6

Problem Difficulty Measures and Local Optima

In general, problem difficulty metrics attempt to quantify different aspects of fitness landscapes that are believed to make search difficult for genetic algorithms and local search. This chapter will consider problem difficulty as measured by correlation length (Manderick, de Weger, and Spiessens 1991), fitness distance correlation (FDC) (Jones and Forrest 1995) and Static-ϕ (Whitley, Mathias, and Pyeatt 1995) to examine the relationship between the problem difficulty measures and landscape features.

Correlation length has been empirically shown to predict problem difficulty for genetic algorithms on NK landscapes and the Traveling Salesman problems (Manderick, de Weger, and Spiessens 1991). Fitness distance correlation has been used to predict problem difficulty for genetic algorithms across a variety of common test problems (Jones and Forrest 1995). Before being used to predict problem difficulty for genetic algorithms, correlation length was used to measure the amount of ruggedness in an NK landscape (Weinberger 1990). The ruggedness that occurs under NK landscapes can be directly measured by the number of local optima. Fitness distance correlation was designed to be a simple measure of problem difficulty for genetic algorithms and local search. FDC measures the smoothness or continuity of the landscape centered around a global optimum. By definition, FDC explicitly accounts for global optima information. Both correlation length and fitness distance correlation are directly related to local optima. An alternative to the purely local perspective is to study schema fitness relationships. Since genetic algorithms theoretically process schemata rather than individual points in the search space, problems that induce schema fitness relationships that lead to the global optimum should be easy for a genetic algorithm to optimize. One might view genetic algorithms as hill-climbers that try to locate schemata (rather than individual points) with high fitnesses. If these highly fit schemata all contain the global optimum, then the genetic algorithm will be guided to the global optimum. If the highly fit schemata lead to a suboptimal region or point, then the genetic algorithm will be misled. A metric that attempts to measure the amount of support at the schema level for a particular target string, ω, is **static-ϕ** (Whitley, Mathias, and Pyeatt 1995).

This chapter will examine several metrics on several test functions created using the function generators described in the previous chapters. In addition, simpler metrics will be compared to determine whether or not the more sophisticated information used in the static-ϕ calculation is needed to predict the genetic algorithm convergence point. Finally, this chapter analytically describes the relationship between schema fitnesses and the basins of attraction for HD_1 local optima.

6.1 A Case Study: Correlation Length and Fitness Distance Correlation

Problem difficulty for genetic algorithms can be viewed from two perspectives: schema relationships or landscape analysis. Landscape analysis methods attempt to determine whether problems are difficult for genetic algorithms based on the local surface structure of the search space. This local view of the search space contrasts with the notion of genetic algorithms as schema processors. This section will empirically analyze two landscape analysis measures: correlation length and fitness distance correlation.

6.1.1 Correlation Length

The first measure that will be examined is correlation length, which is computed using the autocorrelation of a random walk (Weinberger 1990; Manderick, de Weger, and Spiessens 1991). Given a search operator with a neighborhood description N, a **random walk**, $\mathsf{p} : p_1..p_m$, is a walk from a starting point, p_1, to an end point, p_m, based on repeated applications of the search operator: all possible moves have an equal chance of occurring and fitness information is not used to make moves.

The autocorrelation for a series of points is precisely what the name implies: the correlation of the series with itself. The random walk, p, is an ordered series of steps from p_1 through p_m. The autocorrelation of p is taken as the **correlation** of p with p shifted (temporally) by some number h. The correlation coefficient of two random variables x and y is defined as (Cohen 1995):

$$\rho_{x,y} = \frac{\sigma_{xy}^2}{\sigma_x \sigma_y}$$

where σ_{xy} is the covariance of x and y and σ_x and σ_y are the standard deviations for x and y respectively. Covariance is computed by:

$$\sigma_{xy} = \mathsf{E}[(x_i - \mu_x)(y_i - \mu_y)]$$

The autocorrelation for the random variable x can be computed as the autocovariance of x, $R_x(h)$, divided by the variance of x (Manderick, de Weger, and Spiessens 1991):

$$\rho_x(h) = \frac{R_x(h)}{\sigma_x^2}$$

To estimate the autocovariance for x, the sample autocovariance is:

$$\hat{R}_x(h) = \frac{1}{m-h-1} \sum_{i=1}^{m-h} (x_i - \bar{x})(x_{i+|h|} - \bar{x})$$

where $\bar{x}$ is the average of all m values in x.

Finally, the **correlation length** τ is defined as the first value $1 \leq l \leq m$ where distance $\rho(l) \leq 0.5$.

The correlation length measure is used to gauge the ruggedness of a fitness landscape. A large correlation length indicates few local optima while a short correlation length can indicate many local optima. Since correlation length can measure the density of local optima or ruggedness, researchers have taken an intuitive leap by relating problem difficulty for search algorithms to correlation length (Weinberger 1990; Manderick, de Weger, and Spiessens 1991).

Var	Values for K							
	2	3	4	5	6	7	8	9
τ	2.48	1.98	1.34	1.00	1.00	0.82	0.08	0.00
μ_{soln}	0.743	0.756	0.770	0.781	0.776	0.783	0.782	0.783
μ_{rank}	0.20	0.26	0.56	0.74	2.7	2.54	2.32	3.54
μ_{opts}	21.64	52.06	113.1	196.12	305.24	469.96	641.3	874.92

Table 6.1: Results of correlation length over 50 NK landscape problem instances with $N = 16$ and K varying from two to nine.

6.1.2 Correlation Length on NK landscapes

The relationship between correlation length and problem difficulty for genetic algorithms was empirically illustrated by Manderick (1991). In his experiments, he applied a genetic algorithm to several NK landscape problems. Chapter 4 thoroughly describes NK landscapes and presents a small example of how an NK landscape is constructed. The most important feature of this class of functions is that as K is increased, the landscapes become more rugged and have a higher density of local optima.

This experiment verifies the results found by Manderick et. al. (1991) which relate correlation length to problem difficulty for genetic algorithms run on NK landscapes. The results given by Manderick indicate that the genetic algorithm performed worse as the value for K increased. However, the metric for determining problem difficulty was based on the number of improving moves found by a genetic algorithm rather than solution quality. NK landscapes are intended to be maximized, but the solution quality of the global maximum can vary in different problem instances. This experiment uses alternative performance measures to relate correlation length to problem difficulty.

For this experiment, a Simple Genetic Algorithm (ESGA) was designed maximize several 16-bit NK landscapes, varying K from two to nine. The ESGA used a population size of 200, elitism, tournament size of four, probability of crossover of 0.6, probability of mutation of 0.0625 and was run for 200 generations. The correlation length results were generated using a random walk of size $2,048$ and all of the results are the average of 50 problem instances.

The results of the experiment are given in Table 6.1. The table lists the average solution (μ_{soln}) found across each set of 50 NK landscape problem sets; however, these results do not accurately indicate algorithm performance since the true global maximum can vary. For this experiment, the measure of genetic algorithm performance is the **fitness rank** (rank) of the best solution found by the genetic algorithm. The table presents the average fitness rank (μ_{rank}) of the best solution found by the genetic algorithm over all 50 problem instances. The last row in the table is the mean number of maxima (μ_{opts}) in the test problems.

Although NK landscapes *solutions* are maximized, the fitness ranking is performed such that the *best* point in the search space is always ranked zero and the worst point in the search space is ranked $2^L - 1$. Therefore, the genetic algorithm performance will be measured by how well it minimizes the rank of the best solution found during search. The results indicate that as the value for K increases, the ranking of the *best* solution found for the ESGA tends to increase, which indicates that the convergence point of the ESGA is becoming increasingly less fit. The increased ranking indicates that problems do become more difficult for the genetic algorithm as K is increased. This result corroborates the results found by Manderick et al. that led to the conclusion that correlation length appears to loosely predict the problem difficulty for a genetic algorithm.

The table also illustrates that the number of optima (on average) increases with K. The relationship between number of optima and autocorrelation has been discussed by Weinberger (1990). Given that NK landscapes were designed to be tunably rugged landscapes based in part on the K parameter, the relationship between autocorrelation and the parameter K is not surprising. However, correlation length as a measure of problem difficulty for genetic algorithms provides no more information than counting the number of local optima occurring in a search problem. Although, it is a cheaper way to estimate the number of local optima than enumeration, the estimation is based on the assumption that the problem is **isotropic**(Stadler 1999).

6.1.3 Correlation Length on MAX3SAT Problems

The correlation length, as measured by the autocorrelation of a random walk, can be an informative landscape measure provided that the landscape is **isotropic** or nearly isotropic. A function is isotropic when it is statistically similar in all directions from any point on its surface. One interpretation of isotropy is that the statistics of a landscape are invariant to rotation. In the context of fitness landscapes, landscapes are isotropic when the statistics of fitnesses encountered during a random walk are the same regardless of the starting point of the random walk (Weinberger 1990; Manderick, de Weger, and Spiessens 1991).

A rigorous mathematical description of isotropy as it pertains to fitness landscapes was presented by Stadler and Happel (1999a). A brief overview of the properties of pseudo-isotropy will be given here. There are three conditions that must be met for a fitness landscape to be considered pseudo-isotropic. To describe these conditions, additional terminology needs to be introduced. Let $F : \{f : \mathcal{B}^L \rightarrow \mathbb{R}\}$ represent a fitness landscape defined over the set of L-bit strings. Let $\mathcal{E}[f(x)]$ and $\mathcal{V}[f(x)]$ represent the expected value and variance of the evaluation of a single point x over all $f \in F$. The covariance matrix for F is defined as:

$$\mathbf{C}_{xy} = \mathcal{E}[f(x)f(y)] - \mathcal{E}[f(x)]\mathcal{E}[f(y)] \qquad \text{where} \qquad x, y \in \mathcal{B}^L$$

Then the three conditions for pseudo-isotropy are (Stadler and Happel 1999):

(i) There is a constant a_0 such that $\mathcal{E}[f(x)] = a_0$ for all $x \in \mathcal{B}^L$.

(ii) There is a constant s such that $\mathcal{V}[f(x)] = s^2$ for all $x \in \mathcal{B}^L$.

(iii) There is a constant w such that $\frac{1}{2^L} \sum_{y \in \mathcal{B}^L} \mathbf{C}_{xy} = w$

Ultimately, proving that a landscape is or is not isotropic depends on the statistics of the underlying fitness distributions of the family of landscapes. Commonly encountered landscapes that are isotropic or partially isotropic are: NK landscapes (Weinberger 1990), Traveling Salesman Problems, Graph Bipartitioning problems and several forms of the Spin Glass problem (Stadler 1999).

The MAX3SAT problem has currently not been shown to be isotropic. It has been argued that landscape features such as ridges and plateaus are not likely to occur in isotropic landscapes (Stadler and Happel 1999). Since both plateaus and ridges are common features in MAX3SAT problems (Frank, Cheeseman, and Stutz 1997), it may be the case that MAX3SAT problems are not isotropic.

The correlation length measure can and will be applied to MAX3SAT problems; however, the utility of this measure depends on whether or not the MAX3SAT domain is isotropic. Figure 6.1 is the correlation length of random walks taken on the set of 100 variable MAX3SAT problem instances. These are the same problem instances used in Chapter 5 to illustrate the phase transition phenomenon. There were 100 problem instances for each point in the graph and the problems were

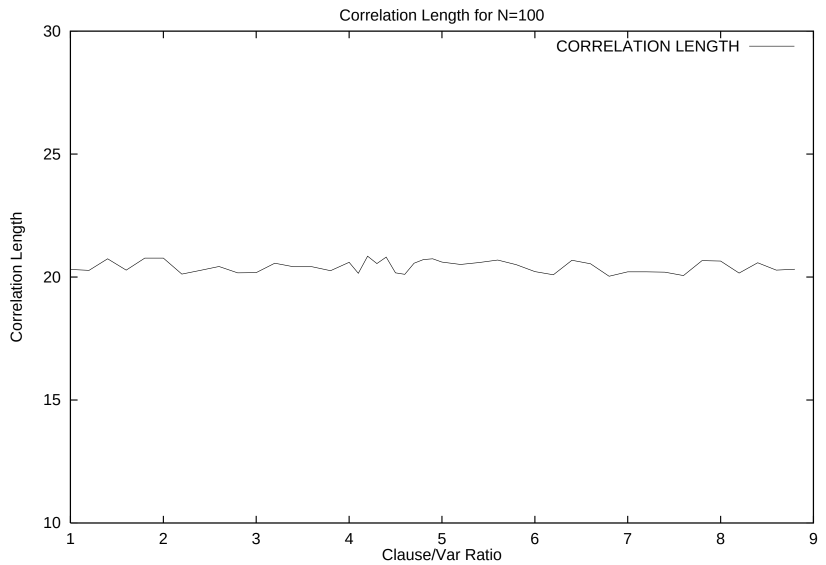


Figure 6.1: Correlation Length computed for MAX3SAT problems.

all generated randomly. The random walks took $10,000$ steps on each problem instance, and the correlation length of the random walks were averaged over each problem set. There was very little variation in the correlation lengths; the standard deviations of the correlation lengths over each set were between 2.0 and 3.0 (the median standard deviation over 45 sets of problems was 2.47). This graph clearly illustrates that the correlation length does not change with the clause-to-variable ratio. If MAX3SAT is not isotropic (anisotropic), then these results cannot provide any insight into the number of optima occurring in the MAX3SAT landscape. If MAX3SAT is isotropic, then these results also indicate that correlation length does not accurately measure the number of local optima in the MAX3SAT landscape. In either case, the use of the correlation length measure does not provide an accurate estimate of problem difficulty for genetic algorithms applied to MAX3SAT problems.

6.1.4 Fitness Distance Correlation

Fitness distance correlation (FDC) (Jones and Forrest 1995) is another metric that attempts to characterize problem difficulty based on the local structure of the search space. FDC is computed by measuring the correlation between the fitnesses of a set of points and their distance from the global optimum. If there are multiple global optima, distance is measured to the closest global optimum. Given a set of pairs (f_i, d_i) where f_i is the fitness of a point and d_i is its distance from a global optimum, the correlation coefficient r is computed as:

$$r = \frac{\sigma_{fd}}{\sigma_f \sigma_d}$$

where

$$\sigma_{fd} = \sum_{i=1}^{i=n} (f_i - \bar{f})(d_i - \bar{d})$$

and n is the number of samples. The set of (f_i, d_i) pairs is collected by randomly sampling the search space.

Var	Values for K							
	2	3	4	5	6	7	8	9
μ_{fdc}	-0.0477	0.0314	-0.0045	-0.0126	-0.0170	0.0050	-0.0049	0.0038
σ_{fdc}	0.1923	0.1463	0.1199	0.0828	0.0696	0.0603	0.0346	0.0299
Easy	14	5	6	2	0	0	0	0
Uncorr	28	33	37	47	50	50	50	50
Hard	8	12	7	1	0	0	0	0
μ_{soln}	0.743	0.756	0.770	0.781	0.776	0.783	0.782	0.783
μ_{rank}	0.20	0.26	0.56	0.74	2.7	2.54	2.32	3.54
μ_{opts}	21.64	52.06	113.1	196.12	305.24	469.96	641.3	874.92

Table 6.2: Results of fitness distance correlation over 50 NK landscape problem instances with $N = 16$ and K varying from two to nine.

The range of values for r is -1 to 1. A -1 indicates that the fitnesses and distances are negatively correlated so that as the distance increases, the fitness decreases. A 1 indicates a positive correlation so that fitness and distance increase simultaneously (and linearly). In general, the correlation score is interpreted as follows (Jones and Forrest 1995):

	Minimization	**Maximization**
Easy	$0.15 \leq r \leq 1.0$	$-1.0 \leq r \leq -0.15$
Uncorrelated	$-0.15 < r < 0.15$	$-0.15 < r < 0.15$
Misleading	$-1.0 \leq r \leq -0.15$	$0.15 \leq r \leq 1.0$

The correlation score needs to be interpreted differently for minimization problems and maximization problems. A high magnitude, positive correlation is desirable for a minimization while a low magnitude, negative correlation is desirable for a maximization problem.

6.1.5 Fitness Distance Correlation on NK landscapes

The fitness distance correlation measure was run on the same set of 16-bit NK landscape problems that were discussed in the previous section. Table 6.2 lists the results of running the fitness distance correlation measure, using a sample size of $4,000$ points, on the set of NK landscape problems. For convenience, the same ESGA results used for the correlation length experiment are repeated in this table. The first row in the table lists the mean correlation value over each set of 50 problem instances. The second row is the standard deviation of the correlation value. The mean of the correlation values do not show any trend in terms of problem difficulty across the different sets of NK landscapes. However, the standard deviations steadily decline as K increases.

Using the set of inequalities given in the previous section, we can categorize the specific problem instances into easy, uncorrelated and difficult based on their individual FDC score which are given in the table. Although the majority of problems are classified as uncorrelated, the number of problems classified as difficult or easy decreases with K.

If FDC was accurately characterizing the NK landscapes problem set, the number of problems classified as easy would steadily decrease with K. To a certain extent, this decrease can be seen in the results; however, on the lower values of K, ESGA performs remarkably well despite the fact that FDC classified most of these problems as uncorrelated. Overall, FDC is producing a conservative classification of the problems by classifying the problems with low K values as uncorrelated when they are easy for the genetic algorithm.

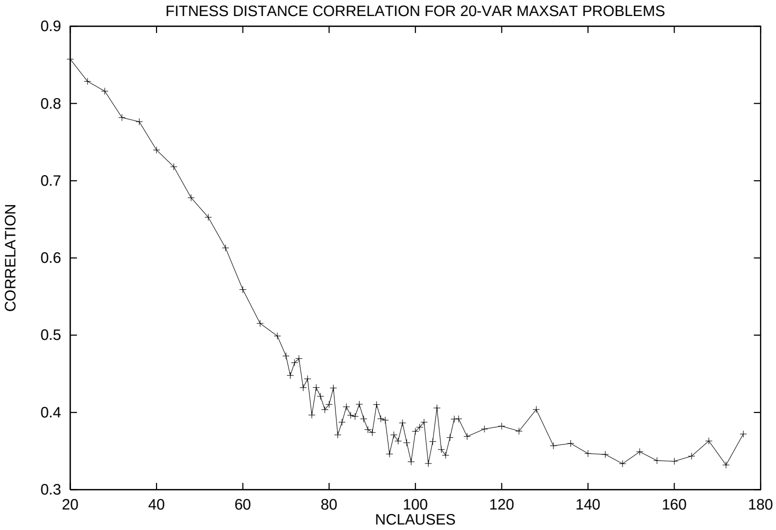


Figure 6.2: Fitness distance correlation for MAX3SAT problems.

6.1.6 Fitness Distance Correlation on MAX3SAT

Fitness distance correlation was also computed for a set of 20-variable MAX3SAT problems. The 20-variable problems were used because FDC requires all global optima to be known. The set of random MAX3SAT problems were generated with the number of clauses ranging from 20 to 180. This number of clauses spans the phase transition. Problem sets of 50 were generated with the number of clauses varying by two; however, problems between 70 clauses and 110 clauses were generated at increments of one clause to ensure that the problems at the phase transition would be sampled adequately.

Figure 6.2 is a plot of the results of running FDC on the 20-variable MAX3SAT problem set. For this set of experiments, $4,000$ points were sampled to compute the FDC scores. Unlike the autocorrelation results, the FDC metric steadily declines until it approaches the phase transition region (approximately 86 clauses). However, the MAX3SAT problem was posed as a minimization problem, and the minimum value for r is consistently above 0.15, which should indicate that the problems are straightforward. As was seen earlier, the genetic algorithm did not reliably locate the global optimum for problems even when the problems were satisfiable.

For the MAX3SAT problems, the fitness distance correlation metric is more accurate than the correlation length because it explicitly accounts for the location and existence of global optima. The trend in the FDC scores mirror the downward trend of the genetic algorithm performance in the MAX3SAT domain. However, for MAX3SAT, the trend in the FDC results are strongly related to the number of global optima in the problem instances.

Figure 6.3 is a plot of the number of local optima that occur in the set of 20-variable MAX3SAT problems used in the FDC experiments. The plot is presented on a log scale. The lower line represents the number of satisfiable solutions seen in the satisfiable problem instances while the upper line represents the number of local optima, where a local optimum is a point at which no single bit flip will locate an improvement. This set of optima is not exactly the same set of local optima used during the FDC calculations. Since FDC is considering the MAX3SAT problem, there are globally optimal solutions even in unsatisfiable problem instances. The number of global optima in the MAX3SAT problem correspond to the center line in the graph. The number of the global optima also decreases exponentially with the number of clauses because the fitness distribution

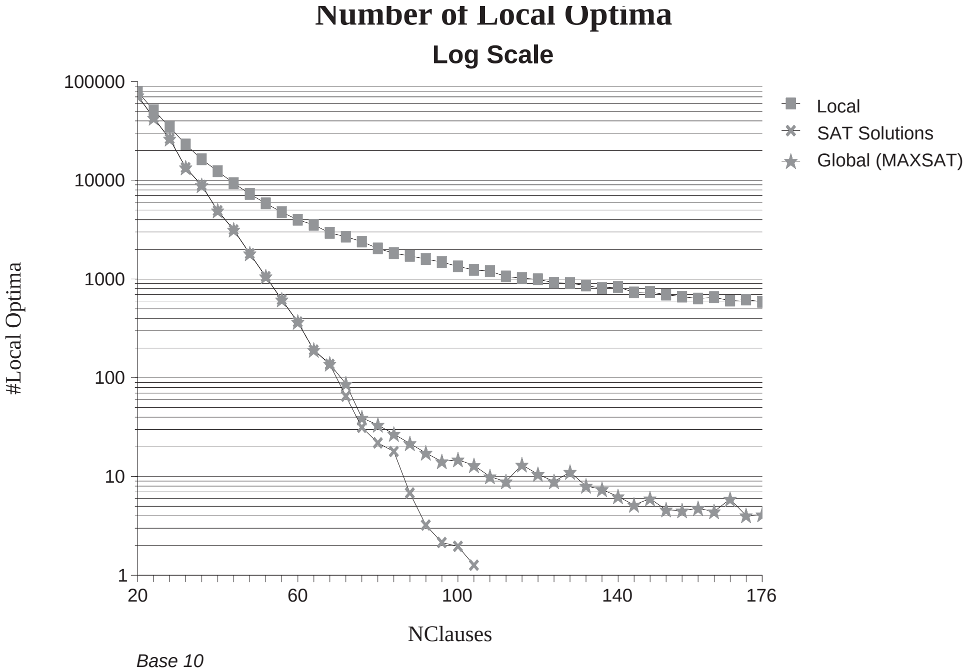


Figure 6.3: Number of local optima for MAX3SAT problems.

becomes more tightly distributed about the mean as the number of clauses increases. Since FDC was measuring distance with respect to the minimum of each problem instance, the negative trend seen in the FDC graph is also related to the number of global optima in each problem set.

6.1.7 Discussion: Correlation Length and Fitness Distance Correlation

At first glance, correlation length and fitness distance correlation appear to give two conflicting accounts of problem difficulty for genetic algorithms on the NK landscapes. Correlation length shows that as K increases, the problems become more difficult while fitness distance correlation illustrates that most of the NK landscape problems are uncorrelated regardless of K. The MAX3SAT problem domain illustrates that both measures deem all of the problems "easy", which is incorrect given the relationship between the phase transition and problem difficulty. Correlation length shows no significant variation across the set of all MAX3SAT problem instances; however, the utility of the correlation length results depends on whether or not the MAX3SAT problems are inherently isotropic landscapes. The performance of FDC as a problem difficulty measure for MAX3SAT is slightly more predictive: FDC decreases rapidly as the clause-to-variable ratio increases.

Altenberg (1995) has discussed the shortcomings of autocorrelation as a measure of problem difficulty for genetic algorithms. Autocorrelation of a random walk does not take into account the ability of a genetic algorithm population to locate improvements *throughout the course of search*. Autocorrelation of random walks fails to capture the dynamics of an evolving genetic algorithm population. Furthermore, the correlation length measure can only be applied to isotropic landscapes, which precludes its use on many test problems.

Fitness distance correlation overcomes many of the shortcomings of autocorrelation analysis. FDC specifically computes distances from randomly chosen points in the search space with global optima and can be applied to any type of *enumerable* search problem or any problem in which global optima are known *a priori*. Additionally, since FDC uses an unbiased random sample of the search space rather than just a random walk, FDC is not as affected by individual local optima as autocorrelation. For these reasons, FDC will be compared against other problem difficulty measures, and correlation length will be abandoned.

6.2 An Alternative to Correlation Measures: Static-ϕ

A schema based measure of problem difficulty is **static-ϕ**. The **static-ϕ** metric, also denoted ϕ_s, metric measures the degree of **consistency** between a point, ω, and the schemata across all **partitions** of the search space. Before describing how to compute the ϕ_s metric, several definitions need to be introduced.

A **partition** is a subset of competing schemata where the subset is defined by a string of length L composed of the `b` and `*` symbols. A `b` indicates that the competing schemata must have a `0` or `1` in that bit position, while a `*` means that the schemata must have a `*` in that position. Furthermore, the **order** of a partition is the number of `b` symbols that occur in the string. Analogously, the **order** of a schema is the number of `0` or `1` bits that occur in the string. So for instance, the four-bit order-1 partition `b***` is made up of the order-1 schemata `0***` and `1***`. The `b***` partition actually partitions the entire search space into two subsets: the strings that begin with a `0` and those that begin with a `1`. So, if there are k `b` symbols occurring in a partition definition (i.e., it is an order-k partition), then there are 2^k schemata in the partition.

The basis for the ϕ_s calculation is the notion of schema fitnesses and schema consistency. Given a schema, H, and a fitness function $f(x) : \mathcal{B}^L \rightarrow \mathbb{R}$, define:

$$\mu_H = \frac{1}{|H|} \sum_{x \in H} f(x)$$

where $|H|$ is the number of strings in the schema. So the fitness associated with any schema H is the average fitness of all of the constituent strings of H. Computing schema fitness averages in this manner is a *static analysis* of the schema relationships since the schema is enumerated. An alternative to a static analysis of schema fitnesses is a *dynamic analysis*. Dynamic analysis uses an *observed* schema fitness calculation that is based on the mixture of a sample (i.e., a genetic algorithm population) of points, referred to as $\hat{\mu}_H$ (Holland 1975). The variable $\hat{\mu}_H$ refers to the observed average fitness of schema H in the population. The key difference between μ_H and $\hat{\mu}_H$ is that μ_H represents a population (in the statistical sense) average and $\hat{\mu}_H$ is a sample average for a random variable. Holland (1992) hypothesized that, over time, $\hat{\mu}_H$ becomes a more accurate estimate of μ_H. However, it has been illustrated that the dynamic schema fitnesses are likely to differ from the static schema fitnesses as the population converges (Whitley, Mathias, and Pyeatt 1995). **Consistency** across a partition occurs when all schemata are sorted according to fitness and as the fitness decreases, then the distance from the *most* fit schema increases. For example, consider the order-2 partition `**bb`, a consistent ranking of the schemata is:

$$\mu_{\texttt{**00}} > \mu_{\texttt{**01}} > \mu_{\texttt{**10}} > \mu_{\texttt{**11}}$$

Note that there are other consistent rankings of the partitions (e.g., having $\mu_{\texttt{**10}} > \mu_{\texttt{**01}}$ is also consistent). If we were now to ask whether or not this ranked set of schemata was consistent with the bit string `0000`, then this particular ranking supports this point. If we were to ask whether or not this ranked set of schemata was consistent with the bit string `1111`, then this particular ranking does not support this point. In general, given the rankings of schemata within a partition, we can measure the amount of consistency between the ranking and any target string $\omega \in \mathcal{B}^L$. The $\phi_s(\omega) : \mathcal{B}^L \rightarrow \mathbb{R}$ metric computes the consistency of all schemata, with respect to the target string ω.

6.2.1 Formally computing ϕ_s

The underlying mathematics of ϕ_s are derived using a **match count function**:

$$\mathsf{M}(H, \omega) : \{\mathtt{0},\mathtt{1},\mathtt{*}\}^L \times \mathcal{B}^L \rightarrow \mathbb{R}$$

which defines the number of bits that schema H has in common with the bit string ω. For instance $\mathsf{M}(\mathtt{**001*0*}, \mathtt{00101011}) = 2$ because the 01 in the fourth and fifth bit positions (respectively) match in the schema and bit string.

Using the match count function, we define a subfunction for ϕ_s called ϕ_p:

$$\phi_p(\pi, \omega) = \sum_{i=1}^{2^k} \sum_{j=1}^{2^k} [\mu_{H_i} > \mu_{H_j}][\mathsf{M}(H_i, \omega) > \mathsf{M}(H_j, \omega)](\mathsf{M}(H_i, \omega) - \mathsf{M}(H_j, \omega)) \tag{6.1}$$

where π defines an order-k partition, $H_i, H_j \in \pi$ and the square brackets denote a predicate function that returns 0 when the inequality is FALSE and 1 when the inequality is TRUE.

F_1			F_2		
H_i	$\mu(H_i)$	$M(H_i, \omega)$	H_i	$\mu(H_i)$	$M(H_i, \omega)$
111**	19.0	3	111**	20.0	3
110**	17.0	2	011**	18.0	2
101**	13.0	2	010**	18.0	1
011**	11.0	2	001**	15.0	1
001**	7.0	1	110**	8.0	2
010**	5.0	1	101**	4.0	2
100**	3.0	1	100**	2.0	1
000**	2.0	0	000**	1.0	0
$\phi_p(\pi, \omega) = 30$			$\phi_p(\pi, \omega) = 25$		

Table 6.3: Example computations of ϕ_p for two fitness functions ($\omega = \mathtt{11111}$,$\pi = \mathtt{bbb**}$).

Table 6.3 shows an example computation of $\phi(\pi, \omega)$ where $\pi = \mathtt{bbb**}$ and $\omega = \mathtt{11111}$ for two different functions F_1 and F_2. The partition π is enumerated for all schemata $H_i \in \pi$. The schemata are listed in the table sorted according to their fitness. The $M(H_i, \omega)$ scores are given for each schema. F_1 is ranked perfectly so that the schemata are ordered consistently with respect to 11111. Furthermore, F_1 yields a maximum possible ϕ_p value.

F_2 provides a good example of the computation of ϕ_p. Using the fitnesses for F_2 given for each of the schemata in Table 6.3, Figure 6.4 illustrates how the $\phi_p(\pi, \omega)$ calculation is performed. Each schema, H_i, is compared to every other schema H_j according to formula 6.1. When the fitness of H_i is greater than the fitness of H_j *and* when $M(H_i, \omega)$ is greater than $M(H_j, \omega)$, that particular pair of schemata will have a nonzero entry in the matrix. It is clear from the figure that only a fraction of the pairings will result in a nonzero entry. The $\phi_p(\pi, \omega)$ calculation is the sum of all entries in the matrix. For F_2, the ϕ_p score is 25.

This version of the ϕ_p function differs from that originally used by Whitley et al. (1995) in that the original version used an explicit fitness ranking of the schemata so that $\frac{2^k(2^k-1)}{2}$ comparisons were needed to perform the calculation. However, this method poses a problem when multiple schemata have equal fitnesses. Schema with equal fitnesses could be compared based on the match count and credit could be assigned (or not assigned) based on their ranking. In Figure 6.4, the shaded matrix entry could have been assigned a one instead of zero using the original formulation j

H (i)	000**	001**	010**	011**	100**	101**	110**	111**
000**	0	0	0	0	0	0	0	0
001**	1	0	0	0	0	0	0	0
010**	1	0	0	0	0	0	0	0
011**	2	1	0	0	1	0	0	0
100**	1	0	0	0	0	0	0	0
101**	2	0	0	0	1	0	0	0
110**	2	0	0	0	1	0	0	0
111**	3	2	2	1	2	1	1	0

$\phi_p = 25$

Figure 6.4: Computing ϕ_p for a small function F_2.

of the ϕ_p function. Thus, the tie-breaking strategy used by the ranking function could affect the final result. By performing all 2^{2k} raw fitness comparisons, all ambiguity is removed. Only schema with differing fitnesses are used in the calculations. The effects of this change are negligible to the final ϕ_s calculation for most test functions.

Two additional supporting functions need to be defined for the ϕ_s calculation. Since each partition contains 2^k schemata, the actual maximum values for ϕ_p vary with k. A normalization factor, $\phi_{max}(\pi)$ is defined as:

$$\phi_{max}(\pi) = \sum_{q=1}^{k} \binom{k}{q} \sum_{i=0}^{q-1} \binom{k}{q} (q-i)$$

where π is an order-k partition. Finally, we define $\hat{\phi}_p(\pi, \omega)$ as a normalized version of ϕ_p:

$$\hat{\phi}_p(\pi, \omega) = \frac{\phi_p(\pi, \omega)}{\phi_{max}(\pi)}$$

so that $\hat{\phi}_p \in [0, 1]$.

Let P represent the set of partitions $\{\mathtt{b,*}\}^L$ excluding the string of all $*$ symbols. The **static phi** calculation, ϕ_s, is defined over all $2^L - 1$ partitions in P:

$$\phi_s(\omega) = \sum_{\pi \in \mathsf{P}} \hat{\phi}_p(\pi, \omega)$$

So for a search problem defined over bit strings of length L, $\phi_s(\omega) \in [0, 2^L - 1]$. A function that exhibits perfect consistency about a string ω has $\phi_s(\omega) = 2^L - 1$.

The ϕ_s metric can be computed for any string in the search space. The **ϕ-spectrum** can be computed by using all strings in the search space as ω (Heckendorn, Whitley, and Rana 1996). The purpose of the ϕ-spectrum is to study which points in the search space are most consistent with the schema fitnesses. The points in the search space with the strongest degree of consistency are the most likely candidates for genetic algorithm convergence points. However, in Chapter 4 it was argued that the convergence points of a genetic algorithm (using mutation) should be a subset of HD_1 local optima. If the genetic algorithm is run until convergence, then the possible values for ω need only be HD_1 local optima.

6.3 A Second Case Study: Fitness Ranking, Fitness Distance Correlation, Static-ϕ and Basin-δ

The calculations for static-ϕ are cumbersome to compute. For each problem, all 3^L schema fitnesses must be computed and then grouped and compared according to each partition. A natural question to ask is whether or not this complex information is actually resulting in a more accurate prediction of genetic algorithm convergence behavior than some other, simpler metrics. To examine this question, several measures were compared to determine how accurately they predict the convergence point of a genetic algorithm.

A variant of the fitness distance correlation (FDC) metric is used in the comparison. The traditional calculation of FDC computes the correlation of the landscape with respect to a global optimum, but this calculation only provides information about how smooth the space is relative to a global optimum (i.e., how much support is given to the global optimum). Rather than computing FDC about a global optimum, FDC is instead computed about all local optima. The resulting correlations are ranked so that the optimum with the most negative score, which is the most attractive point assuming maximization, has a rank of zero. This application of FDC eliminates the need to classify problems as easy, misleading, or difficult. Instead, FDC is just measuring the relative attractiveness of each local optimum.

In addition to ϕ_s and FDC, a third metric was used. The Basin-δ (B_δ) metric measures the fitness difference between neighboring points that occur near a local optimum. The formula for B_δ is:

$$\delta(\omega) = \sum_{\mathsf{bc}(\omega \oplus x) \leq \frac{L}{3}} \sum_{\mathsf{bc}(\omega \oplus y) \leq \frac{L}{3}+1} [\mathsf{bc}(y \oplus \omega) - \mathsf{bc}(x \oplus \omega) = 1](f(x) - f(y))$$

where x, y are bit strings. Emanating outward from the string ω, the fitness difference is calculated by subtracting the fitness of an HD_k neighbor, x, of ω by an HD_{k+1} neighbor of ω, y, such that x and y are HD_1 neighbors (i.e., x is one bit away from y). The raw difference is summed, and the calculations do not extend out further than $\frac{L}{3}$ bits away from ω. The $\frac{L}{3}$ limit is used is due to the limited reach of crossover. Since crossover is the main source of large jumps and those large jumps are limited to relatively low distances, the fraction of the basin used in the calculation is limited to a third of the string length. B_δ attempts to measure the steepness of the basin of attraction around a local optima. Large B_δ scores indicate a large, steep basin of attraction around an optimum. Small scores indicate a small basin of attraction and/or a local optimum with low fitness.

A fourth, and simplest, measure used in the comparison is, the **fitness ranking** (Fit) for each local optimum. The fitness ranking corresponds to the ranking of the optimum with respect to the global optimum, which has a rank of zero. For example, the next best local optimum is ranked one, the third best is ranked two and so on. In this case, the goal of the genetic algorithm is to minimize the fitness ranking of its convergence point. Note that the fitness ranking does not provide a problem difficulty score of any kind.

These four metrics are tested to determine how accurately they each can predict the convergence point of a genetic algorithm. Given all local optima within a search problem, the static-ϕ

measure requires an exceptionally large amount of computational effort to compute, the FDC and B_δ measures require a moderate amount of computational effort, and the fitness ranking requires a minimal amount of computational effort. The purpose of this experiment is to evaluate whether or not predicting the convergence point of a genetic algorithm really requires complex landscape measures.

6.3.1 The Infinite Population Model Genetic Algorithm

The experiment in this section uses an **infinite population model genetic algorithm** to simulate the convergence behavior of a genetic algorithm. The infinite population model genetic algorithm can be thought of as a simulation of a genetic algorithm; this model produces the exact behavior of a genetic algorithm run with an infinite population and models the expected behavior of finite population genetic algorithms. (The original source code for the infinite population model genetic algorithm used in this work was developed by Michael Vose, at the University of Tennessee, Knoxville.) Vose (1993) uses $\mathcal{G}$ to represent this model. The function $\mathcal{G}(\vec{x}_t) : \mathbb{R}^{2^L} \rightarrow \mathbb{R}^{2^L}$ takes in a vector x_t and produces a new vector $\vec{x}_{t+1}$. The $\mathcal{G}$ function is treated as a simulation:

$$\vec{x}_{t+1} = \mathcal{G}(\vec{x}_t)$$

The $\vec{x}$ vector is a vector of proportions for all strings in the search space. The sum of all entries in any $\vec{x}$ vector must be 1.0. The function $\mathcal{G}$ modifies the $\vec{x}$ vector to adjust the proportions according to models of the three genetic operators: selection, crossover and mutation. Theoretically, special versions of $\mathcal{G}$ can be constructed to model specific versions of the genetic operators such as tournament selection or rank based selection. For these experiments, the $\mathcal{G}$ function uses fitness proportionate selection, uniform crossover and no mutation. Crossover must fully explore the search space; thus, uniform crossover is used because it has no positional bias and samples from a wide variety of Hamming distance neighborhoods. Since all strings are represented initially in the population, all strings can be reached during search, and there is no lack of diversity and any point in the search space can, theoretically, be a convergence point. However, providing that the genetic algorithm can reach a local or global optimum, it is hypothesized in this thesis that it is only these points which can be viable convergence points. For a finite population genetic algorithm, mutation is required to ensure that every point can be reached with a nonzero probability during search, while in an infinite population model genetic algorithm with all strings initialized with a nonzero proportional representation, all points in the search space are guaranteed to be reachable. For these reasons, mutation is unnecessary in the infinite population model genetic algorithm.

The convergence behavior of $\mathcal{G}$ is exact and deterministic for any specific initial $\vec{x}$ vector. However, different initial $\vec{x}$ vectors can lead to different fixed points, where a fixed point is a local optimum or an equilibrium state for the population. The same initial $\vec{x}$ vector, with each entry initialized to $\frac{1}{1024}$, was used for all experiments to avoid any initialization bias that might cause the genetic algorithm to converge to a specific point in the search space.

6.3.2 The Test Suite

This experiment evaluates the four metrics for predicting the convergence behavior of the infinite population model genetic algorithm. The test suite for the experiment is comprised of $1,000$ 10-bit test problem instances. The test problems used were required to be small and enumerable so that all of the metrics could be computed and the infinite population model could be used. The 10 sets of problems used in the evaluation are:

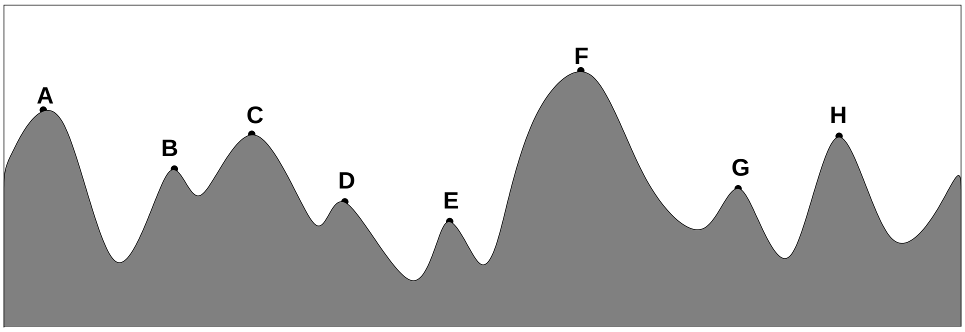


Figure 6.5: Example one dimensional test function.

RND	100 Random problem instances with Gaussian fitness distribution
D10	100 Discrete Function Generator instances (10 optima)
D50	100 Discrete Function Generator instances (50 optima)
D100	100 Discrete Function Generator instances (100 optima)
NK2	100 NK landscape problem instances (N=10,K=2)
NK3	100 NK landscape problem instances (N=10,K=3)
NK4	100 NK landscape problem instances (N=10,K=4)
NK5	100 NK landscape problem instances (N=10,K=5)
NK6	100 NK landscape problem instances (N=10,K=6)
SAT	100 Random MAX3SAT problem instances (55% satisfiable, 4.7 C/V ratio)

By definition, the fitness ranking (Fit) measure provides a relative fitness ranking of the local optima within each of the problems. The remaining measures provide problem difficulty measures by returning a high or low score as appropriate. Each of the measures should indicate the degree of "attractiveness" of each of the optima, relative to one another. Each metric is computed with respect to each of the local optima, and the scores for each optimum are sorted and ranked so that the optimum with the best score, with respect to the metric, has a rank of zero. Thus, each metric will designate a particular rank to the convergence point of the genetic algorithm, assuming that the convergence point of the genetic algorithm is a local optimum as theorized in Chapter 3. Then the metric that produces the overall lowest ranking with respect to the genetic algorithm convergence point is the most accurate.

To provide a more concrete example of the ranking method, consider the one dimensional function shown in Figure 6.5. The eight maxima in the function are labeled A through H. Suppose we have a metric $m(\omega)$ that can be applied to different points in the search space to determine their relative attractiveness to a genetic algorithm. Given a set, S, of the maxima, compute the score for each $s \in S$ using m. Now suppose that m is a metric in which a high value is a better score. Then the relative rankings of the maxima might be:

$$m(A) > m(F) > m(C) > m(B) > m(D) > m(H) > m(E) > m(G)$$

The measure m is not a pure fitness ranking because some of the less fit maxima have higher scores than other higher fit maxima. The actual magnitudes of the score for each point are not as important as the relative rankings; what is important is that A was given the highest score of all local maxima. Thus, a ranking, r of the optima subject to m would be $r(A) = 0$, $r(F) = 1$, $r(C) = 2$, $r(B) = 3$, $r(D) = 4$, $r(H) = 5$, $r(E) = 6$, and $r(G) = 7$. Depending on the convergence behavior of the genetic algorithm, the metric m may or may not have been correct in predicting that A had more support than F. Determining the accuracy of m will be measured by the rank of the true genetic algorithm convergence point. To test the accuracy of the problem difficulty

Metric (Rankings)		Test Functions									
		RND	D10	D50	D100	NK2	NK3	NK4	NK5	NK6	SAT
Fit	median	10	2	6	5	0	0	1	1	1	0
	μ	12.44	2.93	7.28	8.81	0.53	0.7	1.57	1.85	2.63	0.39
	σ	11.51	2.58	6.49	9.14	0.85	1.11	1.96	2.34	3.46	1.16
FDC	median	36	4	17	28	0	1	1	1	2	4
	μ	40.13	4.32	18.3	28	0.97	1.47	2.34	3.32	5.88	3.52
	σ	28.79	2.85	12.19	17.51	1.73	1.99	2.87	4.78	6.78	4.40
B_δ	median	14	1	3	6	0	0	1	1	1	1
	μ	19.69	1.40	4.58	9.88	0.31	0.66	1.34	1.89	3.55	2.66
	σ	17.70	1.78	5.73	11.00	0.66	1.14	1.90	2.65	5.13	3.36
ϕ_s	median	4	1	3	3	0	0	1	1	2	1
	μ	7.73	1.71	5.1	6.23	0.82	1	1.51	1.96	3.36	3.26
	σ	10.98	1.67	6.44	7.42	1.38	1.45	2.03	2.85	4.23	4.89
#LO	μ	92.51	9.65	39.01	59.27	6.93	11.45	18.59	27.17	37.63	38.87
	σ	5.73	0.62	2.72	3.82	3.01	3.45	3.94	4.20	4.76	8.10

Table 6.4: Ranked metric predictions for 10 sets of test functions.

measures, the infinite population model genetic algorithm was run on all problems to determine the actual convergence point.

6.3.3 Results

In this large set of trials, the infinite population model *always* converged to a HD_1 local optimum. Fixed points of the infinite population model genetic algorithm are near equilibrium states (the change to the proportional representation vector is less than 1.0×10^{-12} from generation to generation) and do not always imply that a single point in the search space accounts for the entire population; several points may co-exist in the population even when the population has reached a fixed point. When the population converged such that there was no point with representation of 1.0, the point with the highest representation was considered the convergence point. However, in the $1,000$ problem instances, there were 76 cases where the fixed point did *not* coincide with total convergence to a single point. Of this set, 71 of these instances came from the set of 100 MAX3SAT problem instances.

The rankings with respect to each metric were computed for all of the local optima in each of the test problems. Table 6.4 lists the results of ranking the metric scores for the convergence points of the infinite population model genetic algorithm over 10 problem sets with 100 problem instances per set. The ϕ_s, B_δ and FDC measures can have a wide range of possible values for different sets of problems, making it impossible to categorically say that a particular value for those metrics is good or bad. Therefore, the raw scores were not reported, and all comparisons are made based on the ranked optima. The table reports the median rank (from 100 instances), mean (μ) and standard deviation (σ) of the ranks for all four metrics. Since the largest possible rank depends on the number of optima within each problem, the mean and standard deviation for the number of optima (#LO) in each problem set is reported in the last row of the table. In all cases, a low rank indicates that the metric produced an accurate prediction of the genetic algorithm convergence point.

In general, all four metrics perform well at predicting which points are likely candidates for convergence by the genetic algorithm. The baseline measure of ranking the fitnesses of the optima indicates that the genetic algorithm tended to converge to highly fit local optima. In terms of predicting the convergence point of the genetic algorithm, a ranking of local optima fitnesses is always biased: the global optimum is always ranked as the *top* candidate for convergence. However, the baseline measure of fitness rank (Fit) consistently outperformed FDC. The poor performance of FDC on the Discrete Function Generator (DFG) functions is probably due to the lack of global structure of the problem. Since the basins of attraction are similar for each optimum and the optima are placed uniformly throughout the search space, there is little correlation between the fitness and distance from a single point in the space. When global problem structure exists, as it does for many test problems in the literature, then FDC provides a useful measure. However, when there is little global structure as in a randomly generated MAX3SAT problem or DFG problems, then FDC is less informative.

The B_δ metric attempts to locally measure the steepness of the basin of attraction around each optimum rather than taking a large, global sample as in ϕ_s and FDC. In general, the B_δ value is higher for highly fit local optima; however, unlike a pure fitness ranking, this metric does not always support the global optimum as the top candidate for the genetic algorithm convergence point. In terms of the overall performance between B_δ, ϕ_s and Fit, there were few differences. On the DFG functions, ϕ_s predicted optima with significantly higher ranks than Fit on the 10, 50 and 100 optima problems. On $D100$ and the RND function, ϕ_s predicted significantly higher ranked points than B_δ. The B_δ metric predicted optima with significantly higher ranks than Fit on the 10 and 50 DFG problems. In addition, B_δ predicted optima more accurately than Fit or ϕ_s on the $NK2$ problem. The only occasion when the baseline Fit predicted the optima more accurately than ϕ_s or B_δ was on the SAT problem set. When B_δ and ϕ_s are compared to one another, they tended to perform similarly. The accuracy of the B_δ and ϕ_s were comparable across sets of test problems; however, their accuracy on individual test problem sets was uncorrelated (i.e., there was no similarity in terms of *which* specific problem sets were predicted better by one metric or the other).

6.3.4 Examining the Relationship Between Convergence Points

The results in the previous section illustrate that the convergence points of the genetic algorithm tend to be highly fit local optima, but are not always the global optimum. Two possible factors that might contribute to making a suboptimal local optimum more attractive than other optima are:

1) the relative fitness of the convergence point.

2) the distance from the convergence point to a global optimum.

To empirically examine whether or not these two factors affect the "attractiveness" of the suboptimal genetic algorithm convergence points, the set of all convergence points was examined to determine what their *actual* fitness ranking was and how far away they were from the (or a) global optimum.

Figure 6.6 is a histogram of the true fitness ranks of the suboptimal convergence points. In this case, the true fitness rank is determined by ranking all points in the search space, not just the fitness rankings for the local optima in a problem. There were 634 problem instances out of $1,000$ in which the infinite population model genetic algorithm did *not* converge to the global optimum. The horizontal axis represents the fitness ranking of the points starting from one, and the vertical axis represents the number of convergence points out of 634 with that ranking. The true maximum (worst) possible rank is 1023 for 10-bit problems. However, the maximum value found in this set was 61. While Table 6.4 illustrates that the majority of the convergence points tended to be high ranking with respect to other local optima, this histogram shows that these points also tended to be high ranking points in the entire search space.

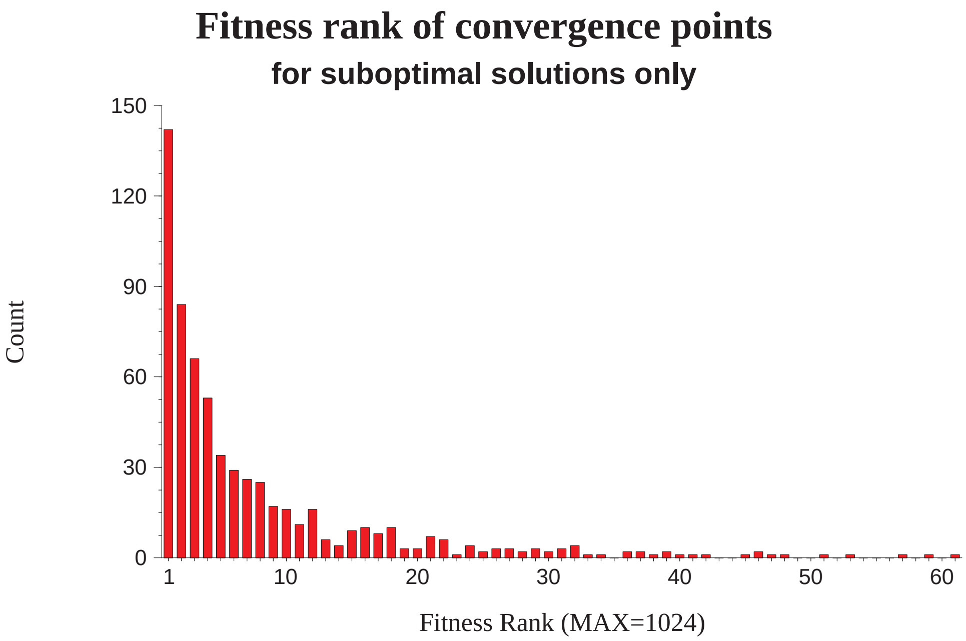


Figure 6.6: Fitness ranking for the convergence points.

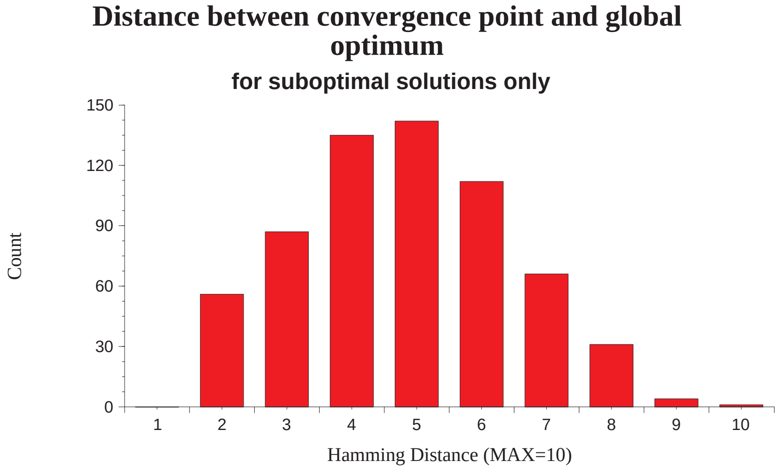


Figure 6.7: Distance between the convergence point and the global optimum.

Figure 6.7 is a histogram of the Hamming distance between the suboptimal convergence point for each problem and the global optimum in each problem instance (or a randomly chosen global optimum when multiple global optima existed). The maximum possible distance between the any points in the 10-bit problems is 10. The horizontal axis represents the Hamming distance and the vertical axis represents the count of problem instances, out of 634, in which the convergence point differed from the optimum by a particular Hamming distance. The histogram for this set of points is indistinguishable from a histogram of the distances between randomly chosen points (shown in Figure 6.8).

Intuitively, it would seem that suboptimal convergence points would need to be relatively distant from global optima because, for instance, it should be easy to move off a local optimum that is only two bits away from a global optimum. The histogram in Figure 6.7 illustrates that the points

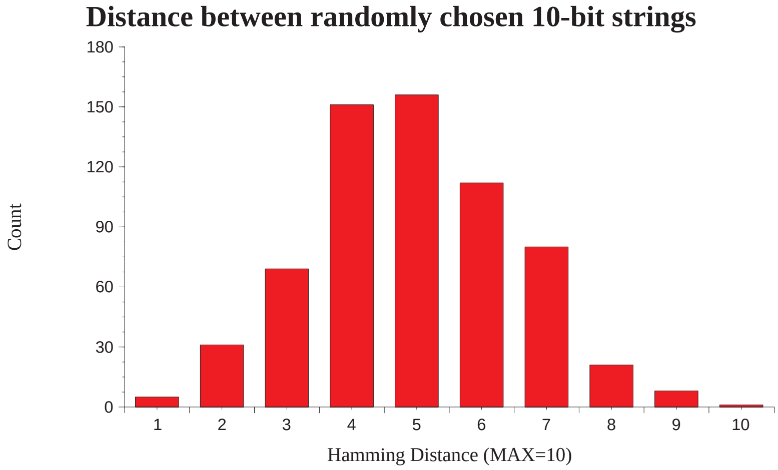


Figure 6.8: Distance between a random set of points.

	Test Functions									
	RND	D10	D50	D100	NK2	NK3	NK4	NK5	NK6	SAT
μ	625.69	605.28	610.01	614.27	762.79	732.33	711.89	699.01	667.51	750.53
σ	49.55	49.17	50.96	47.61	80.42	72.48	71.38	64.04	61.33	82.03

Table 6.5: Raw ϕ_s values over 10 sets of test functions.

do not need to be any more distant from the global optimum than any other string in the search space. In fact, the local optimum could fall as close as two bits from a global optimum and still be a convergence point for the infinite population model genetic algorithm. Although one notable difference between the two histograms is that the distances between suboptimal convergence points and global optima cannot be zero or one, which is a floor effect. This floor effect will cause the two distributions to differ slightly, but the distributions are not significantly different ($p \leq 0.05$).

6.3.5 Predicting Convergence Points with Static-ϕ

The B_δ metric and the optima fitness rank do not provide a raw measure of problem difficulty for the genetic algorithm. They are estimates of the likelihood that a point is a convergence point for a genetic algorithm. However, the ϕ_s metric has the potential to characterize problem difficulty by estimating the degree of consistency in a problem. In this case, the magnitude of the ϕ_s metric is meaningful. If a problem exhibits a high degree of consistency about a point ω, then that point should be attractive to the genetic algorithm. Conversely, if a problem exhibits a low degree of consistency about a point ω, then that point will not be attractive to the genetic algorithm. In many ways, the magnitude of the ϕ_s measure indicates the amount of schema support for the string ω and also measures the strength of that point as a genetic algorithm attractor. Problems with particularly high ϕ_s values at the convergence points should be easy to predict and solve. This section examines the magnitudes of ϕ_s to determine whether some problem classes had ϕ_s scores that were higher or lower than other problem classes.

Table 6.5 presents the raw ϕ_s values computed for the convergence points over the set of test problems. The maximum ϕ_s score possible for a 10-bit problem is 1023. Overall, the ϕ_s scores tended to be high with respect to the convergence points, indicating that the problems exhibit a high degree of consistency at the schema level. Across the set of problems, the only visible trend occurs with the set of NK landscapes. As the degree of epistatic interaction, K, is increased, the ϕ_s scores decrease, which indicates that the convergence points are not as strongly supported by the schema information for high values of K. The DFG functions tended to have the lowest ϕ_s scores across the set of test functions. This is consistent with the observation that these functions contain little or no global structure.

Metric	Solved		Not Solved	
	μ	σ	μ	σ
ϕ_s	714.79	86.21	656.66	79.07

Table 6.6: Cumulative ϕ_s measurement partitioned by whether or not the global optimum was found.

A valid question to ask is whether or not there was a difference between the ϕ_s results for the problems in which the genetic algorithm converged to the optimal solution and for those problems in which the genetic algorithm converged to a suboptimal solution. To examine this relationship, the ϕ_s values were partitioned according to whether or not the problem was solved. Table 6.6 presents the mean and standard deviation for the ϕ_s metric. Over the set of all problem instances, 366 were solved optimally and 634 were not. Across these two sets, there was significant difference between the values of ϕ_s, with $p < 0.01$. The mean ϕ_s for the set of problems that were solved optimally was significantly higher than the mean ϕ_s computed over the set of problems solved suboptimally. The problems with the suboptimal convergence points had lower ϕ_s measures. In addition, the ϕ_s metric also returned low scores for the global optima in this set of problems: the mean ϕ_s with respect to a global optimum in this problem set was 555.91 and the standard deviation was 88.39. This difference in ϕ_s magnitudes indicates that the problems that were solved optimally had an overall stronger consistency and that the global optima was the most supported string in the search space. Problems with high consistency for the global optimum should be categorically easier for the genetic algorithm to optimize.

The results of these experiments illustrate that problem difficulty for a genetic algorithm can be accurately measured using the ϕ_s metric. In addition, the simpler calculation based on local information B_δ also performed well in terms of predicting genetic algorithm convergence points. In terms of predictive capabilities, B_δ and ϕ_s rivaled one another; however, B_δ, is based on local information and ϕ_s is based on schema information.

6.4 Relating Schema Consistency to Local Optima

A natural question that arises is: Is there a relationship between what we see at the macroscopic (schema) level and what we see at the microscopic (bit flip neighborhoods) levels for different search landscapes? This section illustrates why and how the structure of the basins of attraction for local optima affect schema consistency. The use of schema fitnesses for analyzing search spaces is consistent with the hypothesis that the global structure of the search space is more relevant to genetic algorithms than the local structure. Yet the analytical results for the crossover operators in Chapter 4 illustrate that the largest jumps made by crossover have relatively low Hamming distances; the distances are much lower than half of the string length and decrease as search progresses. The moves that are made due to crossover and mutation are at a local level, so local information, in addition to global information is important.

The **basin of attraction** for a local optimum is the subset of points in the search space that *support* that local optimum. Starting search at any of the points in the basin of attraction of a local optimum will ultimately lead to that optimum. In the case of stochastic algorithms, the convergence behavior cannot be predicted exactly, but the concept of a basin of attraction is still useful. Since the basin of attraction is a subset of points then that subset can be represented by special subsets of schemata.

Theorem 3

The basin of attraction of width d for a local optimum can be represented by subsets of schemata of order $L - d$ or higher.

Proof:

Assume that some local optimum ω has a basin of attraction that extends such that the farthest point(s) in the basin are d bits different from ω. Also assume that all points within a radius of d bits from the optimum are included in the basin. Without loss of generality, the entire search space can be remapped about the ω optimum using exclusive-or ($\oplus$) such that ω is remapped to $\mathbf{0}$ (where $\mathbf{0}$ is the string of all `0` bits). The $\oplus$ remapping preserves the distances between neighboring points, so the entire basin of attraction is still centered about $\mathbf{0}$. Let

$$\beta_d : x \in \beta_d \rightarrow \mathsf{bc}(x) \leq d$$

(i.e., the subset composed of all points with d or fewer `1` bits). This subset is the set of:

$$HD_0(\mathbf{0}), HD_1(\mathbf{0}), ..., HD_d(\mathbf{0}) \text{ neighbors of } \mathbf{0}$$

The subset of points in the basin of attraction for $\mathbf{0}$ is β_d.

The β_d subset can be described as the union of sets of schemata of order-$L - 1$ or lower. Any schemata that contains a string with more than d `1`-bits cannot be included in the union. Create a counting function bc_* that takes a schema H as input and returns a count of the number of positions in H that contain a `1` or `*`. For example, bc_*(`***10*`) $= 5$. Then the β_d subset can be defined equivalently by:

$$\beta_d \equiv \bigcup_{\forall H_i s.t. \mathsf{bc}_*(H_i) \leq d} H_i \qquad \text{where } 1 \leq i \leq 3^L$$

The union of all schemata such that there are fewer than d `1` and `*` symbols combined will yield the β_d subset. Of course, each string in β_d can be represented in multiple schemata.

□

If the basin of attraction of $\mathbf{0}$ did not extend out equally in all directions, then the basin could still be described in terms of unions of schemata. However, the order of the schemata included in the union depends on the specific mixture of points in the basin. Knowing the points that are in the basin of attraction for $\mathbf{0}$ enables us to determine which subsets of schemata represent that basin. Conversely, knowing which subsets of schemata represent the basin allows us to explicitly calculate the strings in the basin. The same observation holds for genetic algorithm populations.

6.4.1 Consistency and the Size of the Basin of Attraction

Since every basin of attraction can be represented by *some* subset of schemata, we can start to ask questions about the fitness relationships between those subsets of schemata. It is true that *any* subset of points can be represented by subsets of schemata, but the subsets of points in β_d are related to each other by both distance and fitness. Altenberg (1995) discusses the importance of understanding when a genetic algorithm (or any search algorithm) can locate improving moves. When a point lies within a basin of attraction for an optimum, then, by definition, there must be at least one improving move that a search algorithm could take to move *towards* the optimum. This feature of basins of attraction relates directly to the amount of consistency between competing sets of schemata.

Consider the previous case of a local optimum at $\mathbf{0}$ with a basin of attraction that extends out exactly d bits in all directions. Now impose a fitness constraint on the basin of attraction:

$$\min\{f(x), \forall x \in HD_k(\mathbf{0})\} > \max\{f(y), \forall y \in HD_{k+1}(\mathbf{0})\} \qquad \text{where} \qquad 0 \le k \le d-1.$$

So the fitnesses of the strings that are HD_k neighbors of $\mathbf{0}$ are all greater than any fitness of strings that are HD_{k+1} neighbors of $\mathbf{0}$.

Now reconsider what it means for a set of schemata to be consistent:

$$\mu_{\texttt{**00}} > \{\mu_{\texttt{**01}}, \mu_{\texttt{**10}}\} > \mu_{\texttt{**11}}$$

for a four-bit problem and order-2 partition. A basin of attraction is composed of a subset of schemata up to order-$(L-d)$. Given the relationship between the fitnesses of the points in the basin of attraction, we can carry out a partial ordering of the schemata that define the basin of attraction for a local optimum.

<u>Theorem</u> 4

Given a local optimum, ω, with a basin of attraction extending out exactly d bits and

$$\min\{f(x), \forall x \in HD_k(\omega)\} > \max\{f(y), \forall y \in HD_{k+1}(\omega)\} \quad \text{where} \quad 0 \le k \le d-1$$

all competing schemata that define the basin must be consistently ranked.

Proof:

First assume that the search space has been remapped about ω using the $\oplus$ transformation so that the local optimum is at $\mathbf{0}$. Further assume that the fitnesses of all strings in the basin of attraction of $\mathbf{0}$, β_d, are ordered such that:

$$\min f(x), \forall x \in HD_k(\mathbf{0}) > \max f(y), \forall y \in HD_{k+1}(\mathbf{0})$$

Within the basin β_d, there exists a subset of competing schemata $\mathbb{H}$ that belong to the same partition of the search space. Now, partition the set of schemata based on the number of `1` bits specified and label the subsets $H_0, H_1, .., H_q$ where q is the maximum number of `1` bits in any of the competing schemata. The value for q depends on the width of the basin d and the order of the schemata being tested.

The goal now is to establish that any two schemata chosen from $\mathbb{H}$ will have a consistent ranking. Choose two competing schemata $H_i \in \mathbb{H}$ and $H_j \in \mathbb{H}$ such that $i \neq j$ and $0 < i, j < q$. We now show that if $i < j$, $\mu_{H_i} > \mu_{H_j}$.

Now assume $i < j$. Then there are i `1` bits in schema H_i and j `1` bits in schema H_j. Now recall that schema fitness is computed by taking the average over the fitnesses of the constituent members of the schema. To show $\mu_{H_i} > \mu_{H_j}$, we can simply show that the sum of the fitnesses must have the same relationship since the number of strings in each schema is the same.

A key observation about the composition of the strings in a schema is that all points within any order-k schema are *at most* HD_k neighbors. In fact, within any order-k schema, the Hamming distance relationships are such that there are $\binom{k}{1}$ HD_1 neighbors, $\binom{k}{2}$ HD_2 neighbors, ... , $\binom{k}{k}$ HD_k neighbors for any point in the schema.

Since $\mathbf{0}$ is the central point in the basin of attraction and string fitnesses decrease as they move away from $\mathbf{0}$, we can compare the distances between all strings in H_i and H_j to $\mathbf{0}$. The points in

H_i are HD_i, HD_{i+1}, ... ,HD_{i+k} neighbors of **0**. Likewise, the points in H_j are HD_j, HD_{j+1}, ... ,HD_{j+k} neighbors of **0**. Furthermore, with respect to the **0** string, the number of HD_{i+p} neighbors in H_i is the same as the number of HD_{j+p} neighbors in H_j (where $0 \leq p \leq k$).

Finally, by definition, the fitnesses of the HD_{i+p} neighbors of **0** in H_i must all be higher than the fitnesses of any of the HD_{j+p} neighbors of **0** in H_j. Therefore, the sum of all fitnesses in H_i must be greater than the sum of the fitnesses in H_j, so $\mu_{H_i} > \mu_{H_j}$.

□

Consider the example of a consistent schema ranking:

$$\mu_{\texttt{**00}} > \{\mu_{\texttt{**01}}, \mu_{\texttt{**10}}\} > \mu_{\texttt{**11}}$$

Given a four-bit problem, suppose the local maximum **0** has a basin that extends out to three bits, and we need to order the schemata in the partition `**bb` according to fitness. Then the schemata `**00`, `**01` and `**10` all contain strings that are a part of the basin of attraction for **0**. Note that the `**11` schema contains `1111`, which is not in the basin of attraction for **0**. Further assume that the fitnesses of elements in the basin of attraction **0** have increasing fitnesses as in Theorem 2.

Then according to Theorem 6.4.1,

$$\mu_{\texttt{**00}} > \{\mu_{\texttt{**01}}, \mu_{\texttt{**10}}\}$$

To illustrate why this relation holds, consider the enumeration of all three schemata:

$$\begin{array}{rl} \texttt{**00} = & \{\texttt{0000} \quad \texttt{0100} \quad \texttt{1000} \quad \texttt{1100}\} \\ \texttt{**01} = & \{\texttt{0001} \quad \texttt{0101} \quad \texttt{1001} \quad \texttt{1101}\} \\ \texttt{**10} = & \{\texttt{0010} \quad \texttt{0110} \quad \texttt{1010} \quad \texttt{1110}\} \end{array}$$

The set is sorted so that the strings with the fewest number of `1` bits appear first and the strings with the highest number of `1` bits appear last. By definition, the following fitness relationships hold:

$$\begin{array}{l} f(\texttt{0000}) > \max\{f(\texttt{0001}), f(\texttt{0010})\} \\ f(\texttt{0100}) > \max\{f(\texttt{0101}), f(\texttt{0110})\} \\ f(\texttt{1000}) > \max\{f(\texttt{1001}), f(\texttt{1010})\} \\ f(\texttt{1100}) > \max\{f(\texttt{1101}), f(\texttt{1110})\} \end{array}$$

so $\mu_{\texttt{**00}} > \{\mu_{\texttt{**01}}, \mu_{\texttt{**10}}\}$. The specific relationship between $\mu_{\texttt{**01}}$ and $\mu_{\texttt{**10}}$ depends on the specific fitnesses of the strings; however, regardless of their relationship, the partial ranking is consistent.

In the context of the ϕ_s, the fact that `1111` occurs in another basin of attraction is problematic. It may be the case that its inclusion in the `**11` schema does not compensate for the low fitnesses of `0011`, `0111` and `1011`. In that case, the overall ranking could be completely consistent in the `**bb` partition. However, if the string `1111` has a very high fitness, as is common with deceptive functions, then the ranking of the schemata will be inconsistent. In general, a single basin of attraction contributes to a consistent partial ranking while other basins may contribute to *other* consistent partial rankings. This conflict in terms of support can result in a low consistency across all partitions. A high overall consistency implies that the basins are fully consistent with one another.

6.4.2 Illustrating the Relationship Between Static-ϕ and Local Optima

The analytical argument presented in the previous section relates static-ϕ to the quality and size of the basins of attraction for local optima. The relationship between basins of attraction for local optima and static-ϕ can also be examined empirically. This small set of examples illustrates how one can estimate the size and shape of a basin of attraction for a local optimum. A basin of attraction for a local optimum is a region that *supports* the optimum by having a fitness gradient that leads towards the optimum. In the context of bit represented search spaces, as we radiate outwards from a local optimum, in terms of Hamming distance, the fitnesses of points should become lower as distance increases. Unfortunately, visualizing the neighborhoods in Hamming space are impossible due to the high-dimensionality. A technique for taking **basin estimates** is introduced that gathers fitness information in the basin of attraction for an optimum from several walks from a local optimum to its complement.

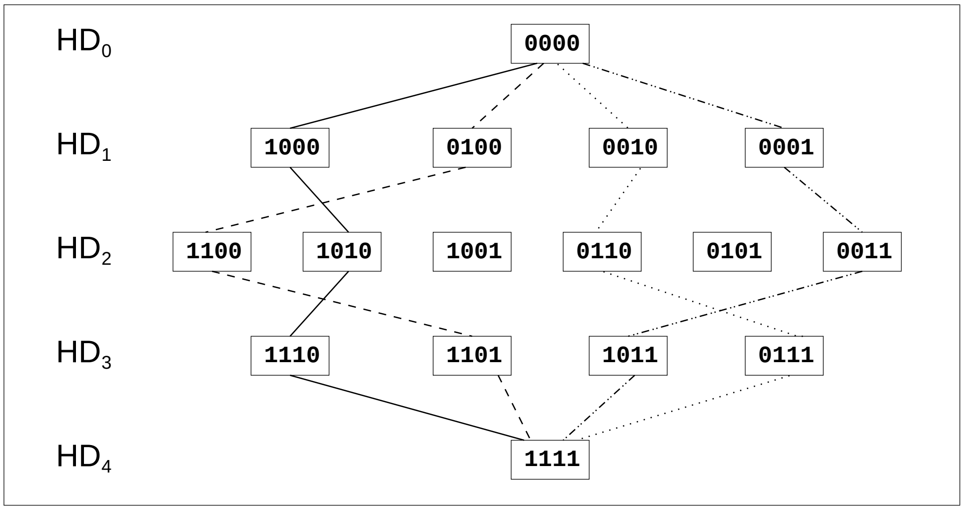


Figure 6.9: Sample random paths in a four bit search space.

Suppose the local optimum or point of interest is the bit string 0000. Figure 6.9 is the set of all four-bit strings ordered by Hamming distance from the string 0000. The walks that are used for computing a **basin estimate** are restricted so that every move is a single-bit flip that increases the number of 1 bits in the string. The figure illustrates four different walks from 0000 to 1111. The fitness at each point in the walk can be averaged so that we can estimate the steepness of the gradient moving outward from 0000 at single bit flip moves. By considering all single bit steps between two complementary strings, we can also gauge the width and influence of the basin of attraction about the string 0000. As we increase the Hamming distance, the points will either have an increasing or decreasing fitness. If we are maximizing, then having a decreasing fitness for as long as possible is advantageous.

In the general case, this type of walk can be performed between local optima and their bit-wise complements. The basin estimate is just the average fitness at each Hamming distance level (i.e., each step in the walk) for multiple walks. According to the analytical arguments given in the previous section, if an optimum resides in a large, steep basin of attraction, then the ϕ_s score computed with respect to that optimum should be relatively high. To illustrate the relationship between schema consistency and basin estimates, three 10-bit example problems were analyzed.

Figure 6.10 shows the basin estimates for several local optima in a single Gray encoded DFG test function containing 10 local optima. For clarity, only four of the basin estimates were included

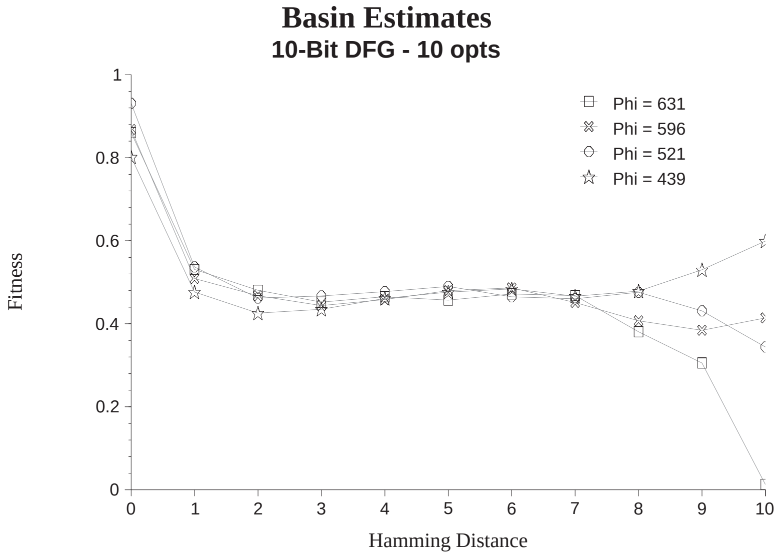


Figure 6.10: Basin Shape Estimates for a DFG function.

in the plot. The horizontal axis is the Hamming distance between steps taken in the walk, and the vertical axis is fitness (1.0 is the global maximum). The basin estimates were computed using 100 walks that were averaged together. Since we are maximizing, an optimum with a basin that constantly moves downward is ideal. Although each of the basin estimates appear to have similar shapes through Hamming distance seven, the basins become noticeably different at higher Hamming distances. Given the practical limits on the step sizes of crossover, these optima will appear similar to a genetic algorithm. The ϕ_s scores are listed for each of the four optima. Notice that all of the ϕ_s scores are relatively low for each of the optima, since the maximum ϕ_s is 1023. Since the basins of attraction are so similar to one another, there is more potential for inconsistencies in terms of schema fitnesses. However, consider the fitness differences between the basin estimates at the high Hamming distances. The optimum with $\phi_s = 631$ is the optimum with the strongest basin estimate; a strong basin estimate is one with a long downward trend in terms of fitness. Although, the ϕ_s scores are relatively low, the ϕ_s scores are ordered according to the strength of the basin estimate.

Figure 6.11 is a plot of the basin estimates for several local optima for a 10-bit MAX3SAT problem instance. There are 47 clauses in the problem, and the problem is formulated to maximize the number of satisfied clauses. Thus, the global optimum is at 47. The specific problem instance had approximately 200 local optima; therefore, six optima were selected to illustrate the variation in ϕ_s scores. The optima with the highest and lowest ϕ_s scores were included in the set, and the remaining points were chosen at random. The overall shape of the basin estimates for the MAX3SAT optima differ from the DFG function optima. Until Hamming distance five, most of the basin estimates have a downward (nearly linear) fitness trend. This downward trend indicates the width of the basin of attraction for each point. Beyond Hamming distance five, the curves diverge, and some curves continue the downward trend while others curve upwards. Again, the ϕ_s scores are highest for the points that have the fitness trends that are downward and steep, while the ϕ_s scores are lowest for the points with upwards fitness trends.

Finally, Figure 6.12 is a plot of the basin estimates for all of the local optima in a 10-bit NK landscape problem (with $K = 2$). The optima all have bowl shaped basins of attraction. The basins tend to extend out to Hamming distance five before curving upwards. Again, the points that curve upwards the earliest and steepest tend to have the lower ϕ_s scores, while the basins that

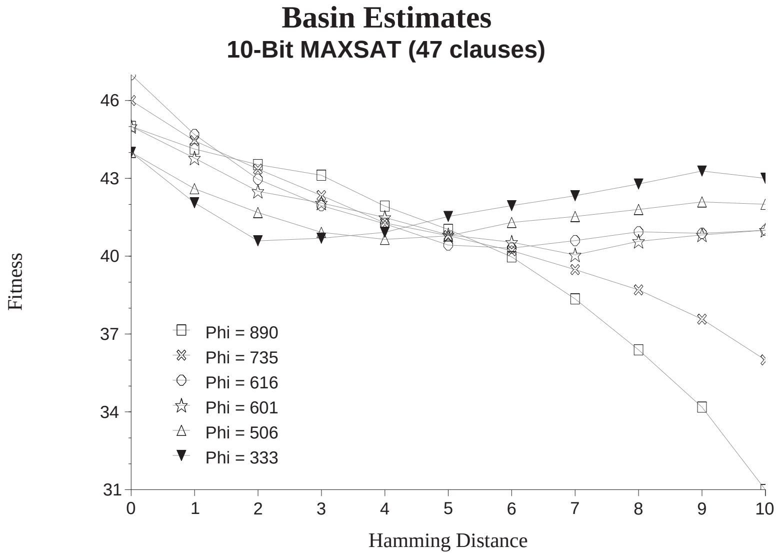


Figure 6.11: Basin Shape Estimates for a MAX3SAT function.

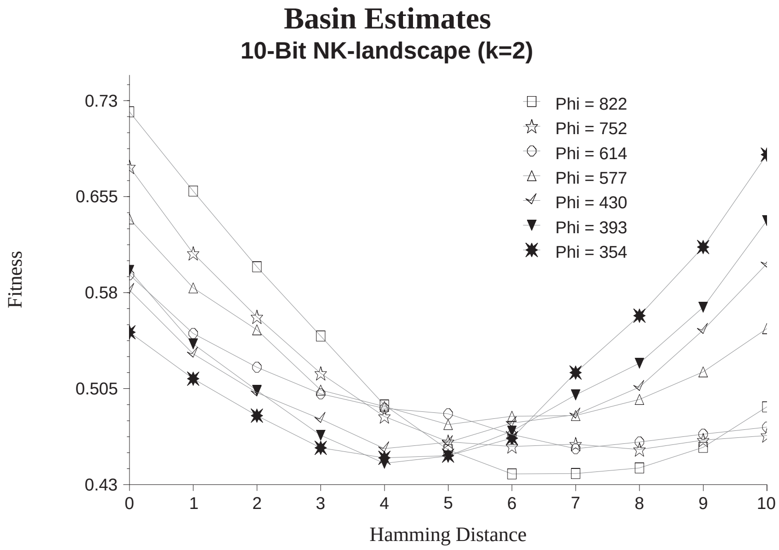


Figure 6.12: Basin Shape Estimates for a NK landscape function.

curve downward for the longest amount of time have the higher ϕ_s scores.

The analytical results in the previous section argue that the amount of consistency that exists for a particular local optimum is related to the width of the basin of attraction. The basin estimates presented in this section illustrate graphically that the ϕ_s scores for the optima can be ranked by the influence of the basin of attraction. These three results also illustrate that the basins of attraction for the optima in the different problem classes are shaped differently. The differences in the shapes of the basins of attraction can affect the overall magnitudes of the ϕ_s scores; however, their relative rankings are based on the relative fitness and width of their basins of attraction.

These results also partially explain the performance of the B_δ measure. This measure was intended to measure the steepness of basins of attraction. Although the measure was only computed over a restricted neighborhood (it extends out 30 percent of the string length), the predictive quality of the B_δ rivaled the ϕ_s measure. The similarities between the performance of B_δ and ϕ_s can be explained by the relationship between the shape and steepness of basins of attraction and schema consistency.

Unfortunately, the B_δ measure needed to be introduced because the other measures based on local landscape information did not accurately predict genetic algorithm behavior. The fitness distance correlation measure is designed to characterize the smoothness of the basins of attraction for local/global optima. The method of sampling points for basin estimates is similar to, but differs from that of fitness distance correlation. In Figure 6.9, it is clear that there are an unequal number of strings at each level in the graph. Consequently, a random sample of strings would tend to be biased by the number of strings at each Hamming distance level. In fact, the distances sampled by a random set of strings should approximate a Binomial distribution. The sampling biases may not be entirely problematic; the basin estimates can also be performed using purely random samples and the basin shapes do not change dramatically. The most problematic aspect of FDC is that the use of correlation can not accurately characterize the smoothness of the basin of attraction. The graphical results for the basin estimates clearly illustrate that the basins of attraction are curves with inflection points. Since the basins are not linearly increasing or decreasing, the correlation coefficient is a poor characterization of the smoothness of the basins for the optima.

6.5 Summary

Problem difficulty measures are designed to predict which problems will be difficult for genetic algorithms. These measures are constructed based on some underlying notion of how genetic algorithms work. Problem difficulty measures may be based upon local neighborhood or a schema based approach.

Correlation length and fitness distance correlation are measures that base their prediction on local neighborhood information. Correlation length is measured using an autocorrelation of a random walk (Manderick, de Weger, and Spiessens 1991), while fitness distance correlation is computed using a random sample of points in the search space (Jones 1995b). Both measures assume a single bit flip neighborhood. One major problem with using the autocorrelation of a random walk is that it offers a very narrow view of the search space, and that is not always indicative of what will be seen by a genetic algorithm population. Fitness distance correlation is more related to a genetic algorithm because it is computed from a large random sample of points: similar to a genetic algorithm population. Fitness distance correlation also explicitly takes into account the number and positions of global optima. Since autocorrelation rarely samples highly fit points, the measure is invariant to the frequency and position of global optima. Therefore, fitness distance correlation is a much better measure of problem difficulty for genetic algorithm and local search methods than correlation length; however, the correlation coefficient does not accurately model the shape of the basins of attraction of global (or local) optima.

A schema based problem difficulty measure is static-ϕ. Static-ϕ measures difficulty as the degree of consistency between the rankings of all possible schemata and a target string. The magnitude of the static-ϕ calculation indicates the strength of support given to the particular target string. Problems deemed difficult for genetic algorithms are those in which there is more support for local optima than there is for global optima; thus, the genetic algorithm tends to converge to a local rather than a global optimum. By considering the amount of support for a global optimum versus a local optimum, static-ϕ can be used to predict where a genetic algorithm will converge. A non-schema based measure, B_δ, provides similar predictive capabilities, but it is based entirely on measuring the steepness of portions of basins of attraction of local optima.

The predictive capabilities of static-ϕ and B_δ stem from the fact that the width and steepness of basins of attraction for local optima influence the consistency of schema rankings about that local optimum. Formal arguments were presented to illustrate this relationship. The relationship between schema consistency and the size and quality of the basins of attraction of local optima illustrate why and how local optima can affect the performance of genetic search.

Chapter 7

Conclusions

This thesis applies theoretical results for genetic algorithm behavior to the practical problem of optimization. In particular, genetic algorithm performance is shown to be adversely affected by high quality local optima with large basins of attraction because such points cause the schema relationships to be misleading. The existence of multiple local optima is known to be problematic for local search algorithms; however, the importance of local optima to genetic search has been overlooked. The schema processing ability of genetic algorithms does not entirely negate the effects of local optima on search, which implies that genetic algorithms and local search algorithms are related. Ultimately, this research shows that what makes search hard for local search, highly fit local optima with large basins of attraction, also make search difficult for genetic algorithms.

7.1 Summary

In summary, this research has examined the relationship between local optima, schema processing, and genetic search from four perspectives.

1) How encoding and representation change the number of local optima.

2) How the genetic operators alter a genetic algorithm's ability to escape local optima.

3) How schema relationships and local problem structure jointly influence genetic algorithm behavior.

4) How schema based problem difficulty measures predict which highly fit local optima are problematic for a genetic algorithm.

7.1.1 Representation and Local Optima

Genetic algorithm design decisions, such as the choice of encoding method or choice of genetic operators, can affect the performance of the genetic algorithm. Researchers have empirically shown that choosing Binary Reflected Gray encoding often improved the performance of genetic algorithms on parameter optimization problems (Caruana and Schaffer 1988; Whitley, Mathias, Rana, and Dzubera 1996). Such observations can be explained by the fact that Gray encodings are guaranteed to preserve or remove the local optima that occur in a numeric space without creating new local optima. On the other hand, Binary encodings offer no such guarantee (Whitley and Rana 1997). Most empirical studies have pitted Gray encodings versus Binary encodings using highly structured numeric optimization problems. In that setting, the fact that Gray encoding preserves the underlying structure of the problem is advantageous to the genetic algorithm. While it is not true that a Gray encoding will always be beneficial, when a problem is well-structured, preserving the structure using a Gray encoding can improve search performance. When a problem lacks global structure, then choosing a Gray or Binary encoding may be equally beneficial.

Problems that lack global structure in the numeric space tend to have a large number of highly fit local optima residing in basins of attraction with similar shapes and sizes. Without global structure, a Gray encoding is not likely to coalesce optima and a Binary encoding is likely to create new highly fit local optima. These highly fit local optima effectively trap the genetic algorithm during search; thus, the genetic algorithm performance may appear similar under both encodings. One reason that genetic algorithms can be trapped by local optima is that the genetic operators may not allow sufficient exploration to move off or avoid local optima.

7.1.2 Genetic Operators and Local Optima

Crossover operators are the means by which genetic algorithms make "large jumps" through the search space. Chapter 4 presented the distributions of step-sizes for several crossover operators, assuming that every string has an equal chance of mating with any other string. This is a reasonable estimation of the sampling that takes place early in the execution of a genetic algorithm. In addition, it is illustrated empirically that the genetic algorithm spends very little time taking "large" steps due to the effects of selection. Once crossover is no longer taking "large" steps, the convergence behavior of the genetic algorithm is governed by selection and mutation. Since genetic algorithms are population based and crossover has performed some amount of "global" exploration, the set of local optima that can trap a genetic algorithm, using a low mutation rate, is limited to highly fit Hamming distance one local optima. While the raw number of local optima does not necessarily affect genetic search, genetic search is affected by the highly fit local optima with large basins of attraction.

7.1.3 MAXSAT, Schema Fitnesses and Local Optima

Despite the fact that highly fit local optima are problematic for genetic algorithms, problem difficulty for genetic algorithms has typically been associated with low order schema relationships. When the fitnesses of low order schemata are such that the global optimum is contained in the schemata with the highest fitnesses, genetic algorithms tend to perform well. Conversely, when low order schema fitnesses are such that the global optimum is not contained in the schemata with the highest fitnesses, genetic algorithms tend to perform poorly. However, analyzing problems to compute low order schema relationships is intractable for large problems. This thesis presents a case study of the MAXSAT problem domain, illustrating that a Walsh analysis can be performed in polynomial time directly from the problem description.

The Walsh analysis allows low order schema relationships to be computed exactly. An empirical analysis was performed which illustrated that a genetic algorithm converged to points that were only partially consistent with the low order schemata information. Furthermore, the experiment also illustrated that the traditional single-point local search algorithm convergence points were just as consistent with the low order schema information as the genetic algorithm convergence points. The reason why the MAX3SAT problem domain is difficult for genetic algorithms is due in part to the misleading schema information, but also to the local landscape of MAX3SAT problems. MAX3SAT problems are plagued with plateaus. Successful local search strategies are built specifically to explore these regions, while genetic algorithms have no such built in exploratory mechanisms. The genetic algorithm tends to converge to regions of the search space that are consistent with the low-order schema information, but the genetic algorithm, without special operators, is unable to maneuver through the plateau regions in the MAX3SAT landscape. This case study illustrates that the misleading schema relationships and the local surface structure of MAX3SAT landscapes are both problematic for genetic search.

7.1.4 Problem Difficulty, Schema Fitnesses and Local Optima

In the general case, predicting when problems are difficult for genetic algorithms can be performed using several problem difficulty measures. Problem difficulty with respect to genetic search has been approached from two directions: local landscape analysis and schema analysis. Several problem difficulty measures were studied to determine which measures most accurately predict the performance of a genetic algorithm. Three commonly used problem difficulty measures were analyzed in this dissertation: correlation length, fitness distance correlation, and static-ϕ.

The first two measures, correlation length and fitness distance correlation, measure the local surface features of the fitness landscape. Using the correlation length measure based on a random walk on the landscape has been shown to be problematic for predicting the behavior of genetic algorithms (Altenberg 1995). In addition, correlation length requires that the landscape be isotropic. The isotropy assumption limits the types of search problems on which the correlation length measure can be meaningfully applied. When correlation length can be used, it offers a reasonable prediction of the number of local optima that occur in a specific problem instance. A more appropriate problem difficulty measure for genetic algorithms is fitness distance correlation. Fitness distance correlation focuses on the relationship between the fitness and distance of points from the global optimum. This measure incorporates more fitness information than correlation length; however, it is also not an accurate measure of difficulty for genetic algorithms.

Static-ϕ measures the amount of consistency between schemata, with respect to a target string. This measure incorporates both the local and global fitness relationships because schema of all orders are used in its calculation. Compared to the other methods for characterizing problem difficulty for genetic algorithms, static-ϕ appears to be a good predictor of genetic algorithm behavior. The schema consistency that is measured by static-ϕ relates directly to local optima and their basins of attraction. Local optima occur in basins of attraction where points lying within the basin have similar fitnesses. Typically, the fitnesses of the points within a basin of attraction decreases as their distance from the local optimum increases. This fitness-distance relationship between points in a basin of attraction and a local optimum affects the amount of consistency between schemata. When a basin of attraction about the local optimum is large and there are few competitive local optima with large basins, the schema consistency will be high for that optimum. As the number of highly fit local optima with large basins of attraction increases, the schemata will become less and less consistent.

Genetic algorithms are affected both by local optima and by schema relationships: highly fit local optima with large basins of attraction tend to influence the amount of consistency between schemata. Since genetic algorithms perform the bulk of their exploration in the early stages of search using crossover, low order schema relationships are important. If there is no global structure or if the schema relationships are misleading, the genetic algorithm may commit to searching a region of the search space that does not contain the global optimum. As the generations progress, the population becomes increasingly uniform and the beneficial effects of crossover are diminished. Once crossover is no longer exploring the search space, the genetic algorithm has committed to searching a particular region of the space, which can be characterized as a basin of attraction, and will likely become trapped in a highly fit local optimum.

7.2 Implications of this Research

The results in this thesis illustrate how genetic algorithms are similar to and different from local search algorithms. This work illustrates that what makes search difficult for genetic algorithms is similar to what makes search difficult for local search algorithms: highly fit local optima with large basins of attraction. However, this thesis has also illustrated that local search tends to sample points in the search space differently from a genetic algorithm. Specifically, genetic algorithms sample local optima with low fitnesses infrequently whereas local search samples all local optima with frequency proportional to the size of the basin of attraction for each optimum.

Understanding the relationship between genetic search and local optima is vital for designing and improving genetic algorithms. Unless a genetic algorithm is able to detect and avoid becoming trapped by local optima, it may be a less effective optimization technique than single-point local search algorithms using random restarts. If a genetic algorithm is destined to be drawn into basins of attraction of inferior local optima, they should be constructed to recognize that situation and continue exploring the search space. Hybrid methods (genetic algorithms utilizing local search) and random restarts are useful techniques for handling local optima. Local search can find local optima more quickly than a population that is sampling and resampling many of the same search points. A hybrid algorithm, which uses genetic algorithms to globally explore the search space and local search to locally explore the search space, should be more effective than a traditional genetic algorithm on well-structured optimization problems.

In addition, approaches such as shifting (Rana and Whitley 1997) can be beneficial because they are mechanisms that can be used to restart search once the population is trapped by a local optimum. Shifting is a means for changing representation when a search algorithm converges to a local optimum. Although shifting can be used with any bit encoding method, Gray encodings offer the most benefits because they always preserve or coalesce the local optima that occur in numeric space. Using shifting to switch Gray encodings upon convergence to a local optimum alters the connectivity of the search space and may allow search to progress.

Since genetic algorithms are affected by local optima, a key to improving their performance as optimizers is to design them to detect and handle local optima more efficiently. This thesis provides both theoretical and empirical evidence to justify genetic algorithm modifications that allow the genetic algorithm to search more efficiently at the local levels.

Appendix A

Test Function Weights

A method for scaling two-dimensional test problems to higher dimensions is the Weighted-Wrap method(Whitley, Mathias, Rana, and Dzubera 1996). Given a test function in the form $F(x, y)$, a new n-dimensional version of the function, $G(x_1, x_2, .., x_n)$, can be expanded by using $F(x, y)$ as a weighted subfunction. The expansion in the n-dimensional case is:

$$G(x_1, x_2, .., x_n) = w_1 F(x_1, x_2) + w_2 F(x_2, x_3) + .. + w_{n-1} F(x_{n-1}, x_n) + w_n F(x_n, x_1)$$

For a *wrapped* function of n parameters, there are n uses of each parameter in each of the two-dimensions of the 2-D subfunction. Without weighting, this additive expansion can result in a smoothing effect on the landscape of the n-dimensional problem. To alleviate the smoothing problem, randomly generated weights are used to perturb the landscape. The random weights used for the Weighted-Wrap expansion of Whitley and Rana's functions in Chapter 3 are:

10-D Weights		
w_0	(0, 1)	0.3148
w_1	(1, 2)	0.5187
w_2	(2, 3)	1.0105
w_3	(3, 4)	0.7156
w_4	(4, 5)	0.7189
w_5	(5, 6)	0.6712
w_6	(6, 7)	0.3151
w_7	(7, 8)	0.8299
w_8	(8, 9)	0.6410
w_9	(9, 0)	0.5798

Appendix B

Using Walsh Analysis to Compute Summary Statistics

Walsh analysis can be used to compute summary statistics for **fitness distributions** of discrete optimization problems (Heckendorn, Rana, and Whitley 1999). Note that the fitness distribution is the distribution formed by evaluating all possible inputs to a problem. So, for a problem defined over 2^L possible inputs, the distribution would be comprised of all 2^L evaluations of the inputs. Clearly, computing summary statistics for arbitrary fitness distributions would require exponential time. Goldberg and Rudnick (Goldberg and Rudnick 1991) have used Walsh coefficients to calculate fitness variance for fitness distributions of schemata; however, the calculations were intended for enumerable functions.

Since Walsh analysis can be performed for MAXSAT in polynomial time with respect to the length, K, of the clauses, so can the calculations for summary statistics. In this section, we show how higher order statistics such as skew and kurtosis can also be computed from the Walsh coefficients by using a general formula for computing the r^{th} moment for any MAXSAT fitness distribution and for any other problem where all nonzero Walsh coefficients are known.

Since $\psi_0(x) = 1$ for all inputs, the w_0 coefficient is the mean of all fitnesses, so this is already available for MAXSAT problems. Given the mean, the formula used to compute the r^{th} moment for a discrete random variable X is:

$$\mu_r = E[(X - \mu)^r] = \sum_{x \in X} (x - \mu)^r p(x)$$

For our purposes, the function $p(x) = \frac{1}{2^L}$ since we are enumerating a function over binary strings and each point occurs $\frac{1}{2^L}$ times. The function then becomes:

$$\mu_r = \sum_{x \in X} \frac{(x - \mu)^r}{2^L}$$

So, given any MAXSAT function $f(x)$, the r^{th} moment over the distribution of fitness for all 2^L possible input strings, assuming we can encode the input value as a binary string, is:

$$\mu_r = \frac{1}{2^L} \sum_{x=0}^{2^L-1} (f(x) - \mu)^r$$

Recall that $f(x) = \sum_{i=0}^{2^L-1} w_i \psi_i(x)$, so

$$\mu_r = \frac{1}{2^L} \sum_{x=0}^{2^L-1} \left(\sum_{i=0}^{2^L-1} w_i \psi_i(x) - \mu \right)^r$$

Since $\mu = w_0$, and $\psi_0(x) = 1\ \forall x$:

$$\mu_r = \frac{1}{2^L} \sum_{x=0}^{2^L-1} \left(\sum_{i=1}^{2^L-1} w_i \psi_i(x) \right)^r$$

Now create a set of r, indices a_j for $1 \leq j \leq r$, then we can expand the formula to be:

$$\mu_r = \frac{1}{2^L} \sum_{x=0}^{2^L-1} \left(\sum_{a_1=1}^{2^L-1} w_{a_1} \psi_{a_1}(x) \right) \left(\sum_{a_2=1}^{2^L-1} w_{a_2} \psi_{a_2}(x) \right) ... \left(\sum_{a_r=1}^{2^L-1} w_{a_r} \psi_{a_r}(x) \right)$$

Since the Walsh coefficients do not depend on x, the formula can be rewritten as:

$$\mu_r = \frac{1}{2^L} \sum_{a_1=1}^{2^L-1} \sum_{a_2=1}^{2^L-1} ... \sum_{a_r=1}^{2^L-1} w_{a_1} w_{a_2} ... w_{a_r} \sum_{x=0}^{2^L-1} \psi_{a_1}(x) \psi_{a_2}(x) ... \psi_{a_r}(x)$$

The latter summation can be reduced to:

$$\mu_r = \frac{1}{2^L} \sum_{a_1=1}^{2^L-1} \sum_{a_2=1}^{2^L-1} ... \sum_{a_r=1}^{2^L-1} w_{a_1} w_{a_2} ... w_{a_r} \sum_{x=0}^{2^L-1} \psi_{a_1 \oplus a_2 \oplus .. \oplus a_r}(x)$$

The sum of the ψ_i functions over all possible inputs x is always 0 with the exception of when $i = 0$ in which case the sum is 2^L. So, only when $a_1 \oplus a_2 \oplus ... \oplus a_r = 0$ is the inner sum nonzero. Therefore,

$$\begin{aligned} \mu_r &= \frac{1}{2^L} \sum_{a_1 \oplus a_2 \oplus ... \oplus a_r = 0} w_{a_1} w_{a_2} ... w_{a_r} 2^L, \qquad a_i \neq 0 \\ &= \sum_{a_1 \oplus a_2 \oplus ... \oplus a_r = 0} w_{a_1} w_{a_2} ... w_{a_r}, \qquad a_i \neq 0 \end{aligned} \tag{B.1}$$

To summarize this formula, given the set of nonzero Walsh coefficients, we can compute the r^{th} moment for the fitness distribution using products of the Walsh coefficients such that the exclusive-or of the indices is zero.

This formula allows us to compute the variance, skew and kurtosis for any fitness distribution provided we are given the Walsh coefficients.

$$variance = \mu_2 = \sigma^2 \qquad skew = \frac{\mu_3}{\sigma^3} \qquad kurtosis = \frac{\mu_4}{\sigma^4}$$

For example, since $a_1 \oplus a_2 = 0$ if and only if $a_1 = a_2$ then the variance for any function can be computed

$$\sum_{i=1}^{2^L-1} w_i w_i$$

In the special case of MAXSAT there is only a polynomial number of nonzero Walsh coefficients and we can identify and compute them in polynomial time. Therefore all the summary statistics can be computed from formula B.1 in polynomial time.

REFERENCES

Altenberg, L. (1995). The schema theorem and price's theorem. In D. Whitley and M. Vose (Eds.), *Foundations of Genetic Algorithms - 3*, pp. 23–50. Morgan Kaufmann.

Booker, L. (1987). Improving search in genetic algorithms. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*, Chapter 5, pp. 61–73. Morgan Kaufmann.

Booker, L. B. (1992). Recombination distributions for genetic algorithms. In D. Whitley (Ed.), *Foundations of Genetic Algorithms - 2*, pp. 29–44. Morgan Kaufmann.

Caruana, R. and J. Schaffer (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proc. of the 5th Int'l. Conf. on Machine Learning*, pp. 152–161. Morgan Kaufmann.

Cheeseman, P., B. Kanefsky, and W. M. Taylor (1991). Where the *really* hard problems are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*.

Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. The MIT Press.

Culberson, J. (1992). Genetic invariance: A new paradigm for genetic algorithm design. Technical Report TR-92-02, University of Alberta, Edmonton, Alberta, Canada.

Davis, L. (1991). Bit-climbing, representational bias, and test suite design. In L. Booker and R. Belew (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 18–23. Morgan Kaufmann.

Davis, M., G. Logemann, and D. Loveland (1962). A machine program for theorem-proving. *Communications of the ACM*, 394–397.

Davis, M. and H. Putnam (1960). A computing procedure for quantification theory. *Journal of the ACM 7*, 201–215.

DeJong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph. D. thesis, University of Michigan, Department of Computer and Communication Sciences, Ann Arbor, Michigan.

DeJong, K. A. (1992). Genetic algorithms are not function optimizers. In L. D. Whitley (Ed.), *Foundations of Genetic Algorithms - 2*, pp. 5–18. Morgan Kaufmann.

DeJong, K. A., M. A. Potter, and W. M. Spears (1997). Using problem generators to expore the effects of epistasis. In T. Bäck (Ed.), *icga7*, pp. 338–339.

Eshelman, L. (1991). The chc adaptive search algorithm. how to have safe search when engaging in nontraditional genetic recombination. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, pp. 265–283. Morgan Kaufmann.

Eshelman, L. and J. D. Schaffer (1991). Preventing premature convergence in genetic algorithms by preventing incest. In L. Booker and R. Belew (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.*

Eshelman, L. J., R. A. Caruana, and J. D. Schaffer (1989). Biases in the crossover landscape. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 10–19.

Eshelman, L. J. and J. D. Schaffer (1993). Crossover's niche. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 9–14.

Fogel, D. B. and J. W. Atmar (1990). Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. *Biolgical Cybernetics 63*(2), 111–114.

Forrest, S. and M. Mitchell (1992). Relative building-block fitness and the building-block hypothesis. In *Foundations of Genetic Algorithms - 2*, pp. 109–126.

Frank, J. (1994). A study of genetic algorithms to find approximate solutions to hard 3cnf problems. In *Golden West International Conference on Artificial Intelligence.* Kluwer Academic Publishers.

Frank, J., P. Cheeseman, and J. Stutz (1997). When gravity fails: Local search topology. *Journal of Artificial Intelligence Research 7*, 249–281.

Ginsberg, M. (1993). *Essentials of Artificial Intelligence.* Morgan Kaufmann.

Glover, F. (1994). Tabu search fundamentals and uses. Technical report, University of Colorado at Boulder, CU Boulder.

Goldberg, D. (1989a). Genetic algorithms and walsh functions: Part i, a gentle introduction. *Complex Systems 3*, 129–152.

Goldberg, D. (1989b). Genetic algorithms and walsh functions: Part ii, deception and its analysis. *Complex Systems 3*, 153–171.

Goldberg, D. (1989c). *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison-Wesley Publishing, Co.

Goldberg, D. E. and M. W. Rudnick (1991). Genetic algorithms and the variance of fitness. *Complex Systems 5*(3), 265–278.

Grefenstette, J. J. (1992). Deception considered harmful. In D. Whitley (Ed.), *Foundations of Genetic Algorithms - 2*, pp. 75–92. Morgan Kaufmann.

Heckendorn, R. B., S. Rana, and L. D. Whitley (1998). Test function generators as embedded landscapes. In *Foundations of Genetic Algorithms - 5*, Leiden, The Netherlands.

Heckendorn, R. B., S. Rana, and L. D. Whitley (1999). Polynomial time summary statistics for a generalization of maxsat. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida.

Heckendorn, R. B., L. D. Whitley, and S. Rana (1996). Nonlinearity, walsh coefficients, hyperplane ranking and the simple genetic algorithm. In *Foundations of Genetic Algorithms - 4*, San Diego, California.

Holland, J. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems* (second ed.). Cambridge, MA: MIT Press.

Hordijk, W. (1996). A measure of landscapes. *Evolutionary Computation 4*(4), 335–360.

Horn, J. and D. E. Goldberg (1995). Genetic algorithm difficulty and the modality of the fitness landscape. In D. Whitley and M. Vose (Eds.), *Foundations of Genetic Algorithms - 3*, Volume 3, Palo Alto, CA, pp. 243–270. Morgan Kaufmann.

Johnson, D. S., C. H. Papadimitriou, and M. Yannakikis (1988). How easy is local search? *Journal of Computer and System Sciences 37*, 79–100.

Jones, T. (1995a). *Evolutionary Algorithms, Fitness Landscapes and Search.* Ph. D. thesis, University of New Mexico, Albuquerque, New Mexico.

Jones, T. (1995b). One operator, one landscape. Technical Report SFI TR 95-02-025, Santa Fe Institute.

Jones, T. and S. Forrest (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 184–192.

Kauffman, S. A. (1989). Adaptation on rugged fitness landscapes. In D. Stein (Ed.), *Lectures in the Science of Complexity*, pp. 527–618. Addison-Wesley.

Kauffman, S. A. (1993). *Origins of Order.* Oxford Press.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983). Optimization by simulated annealing. *Science 220*(4598), 671–680.

Lee, T. C. S., A. L. G. Koh, and Y. Nakakuki (1994). Genetic algorithm approaches to sat problems. In *Singapore International Conference on Intelligent Systems*, pp. B171–B176.

Liepins, G. E. and M. D. Vose (1991). Deceptiveness and genetic algorithm dynamics. In G. J. E. Rawlins (Ed.), *Foundations of Genetic Algorithms*, pp. 36–50.

Lin, S. and B. Kernighan (1973). An efficient heuristic procedure for the traveling salesman problem. *Operations Research 21*, 498–516.

Manderick, B., M. de Weger, and P. Spiessens (1991). The genetic algorithm and the structure of the fitness landscape. In *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann.

Mathias, K. E. and L. D. Whitley (1994). Transforming the search space with gray coding. *IEEE Transactions on Evolutionary Computation 1*, 513–518.

Mitchell, D., B. Selman, and H. Levesque (1992). Hard and easy distribution of sat problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA.

More, J. J., B. S. Garbow, and K. E. Hillstrom (1981). Testing unconstrained optimization software. *ACM Transactions on Mathematical Software 7*(1), 17–41.

Mühlenbein, H. (1991). Evolution in time and space: The parallel genetic algorithm. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms*, pp. 316–337. Morgan Kaufmann.

Papadimitriou, C. H. (1994). *Computational Complexity.* Addison-Wesley Publishing, Co.

Papadimitriou, C. H. and K. Steiglitz (1982). *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall.

Radcliffe, N. J. (1991). Equivalence class analysis of genetic algorithms. *Complex Systems 5*, 183–205.

Radcliffe, N. J. (1992). Non-linear genetic representations. In R. Männer and B. Manderick (Eds.), *Parallel Problem Solving from Nature, 2*, pp. 259–268.

Rana, S., R. Heckendorn, and D. Whitley (1998). A tractable walsh analysis of sat and its implications for genetic algorithms. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence.*

Rana, S. and D. Whitley (1997). Bit representations with a twist. In T. Bäck (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms.* Morgan Kaufmann.

Rana, S. and L. D. Whitley (1998). Search, binary representations and counting optima. In *Evolutionary Computing: AISB Workshop.*

Reeves, C. and C. Wright (1995). Epistasis in genetic algorithms: An experimental design perspective. In L. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 217–224. Morgan Kaufmann.

Reeves, C. R. (1994). Genetic algorithms and neighourhood search. In T.C.Fogarty (Ed.), *Evolutionary Computing: AISB Workshop*, Leeds, UK. Springer-Verlag.

Schaffer, J. D. (1987). Some effects of selection procedures on hyperplane sampling by genetic algorithms. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*, pp. 89–130. Morgan Kaufmann.

Selman, B., H. Kautz, and B. Cohen (1993). Local search strategies for satisfiability testing. In Trick and Johnson (Eds.), *Second DIMACS Challenge on Cliques, Coloring and Satisfiability.*

Selman, B., H. Kautz, and D. McAllester (1997). Ten challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, Nagoya, Japan.

Selman, B. and H. A. Kautz (1993). An empirical study of greedy local search for satisfiability testing. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, D.C.

Selman, B., H. Levesque, and D. Mitchell (1992). A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, pp. 440–446.

Spears, W. M. (1992). Crossover or mutation? In L. D. Whitley (Ed.), *Foundations of Genetic Algorithms - 2*, pp. 221–238. Morgan Kaufmann.

Spears, W. M. (1998). *The Role of Mutation and Recombination in Evolutionary Algorithms.* Ph. D. thesis, George Mason University, Fairfax, Virginia.

Stadler, P. F. (1999). Spectral landscape theory. In J. Crutchfield and P. Schuster (Eds.), *Submitted to: Evolutionary Dynamics - Exploring the Interplay of Selection, Neutrality, Accident and Function.* Oxford University Press.

Stadler, P. F. and R. Happel (1999). Random field models for fitness landscapes. *Journal of Mathematical Biology (to appear).*

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann.

Syswerda, G. (1992). Simulated crossover in genetic algorithms. In L. D. Whitley (Ed.), *Foundations of Genetic Algorithms - 2*, pp. 239–255.

Tovey, C. A. (1985). Hill climbing and multiple local optima. *SIAM Journal on Algebraic and Discrete Methods 6*(3), 384–393.

Trick and Johnson (Eds.) (1993). *The Second DIMACS International Algorithm Implementation Challenge on Clique, Graph Coloring and Satisfiability*, http://dimacs.rutgers.edu/pub/challenge.

Ulder, N., E. Aarts, H. Bandelt, P. V. Laarhoven, and E. Pesch (1990). Genetic local search algorithms for the traveling salesman problem. In *Parallel Problem Solving from Nature*, Dortmund, Germany, pp. 109–116.

Vose, M. D. (1993). Modeling simple genetic algorithms. In L. D. Whitley (Ed.), *Foundations of Genetic Algorithms - 2*, pp. 63–73. Morgan Kaufmann.

Vose, M. D. (1995). Modeling simple genetic algorithms. *Evolutionary Computation 3*(4), 453–472.

Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics 63*, 325–226.

Whitley, D. (1999). A free lunch proof for gray versus binary encodings. In *Proceedings of the Genetic and Evolutionary Computation Conference.*

Whitley, D., K. Mathias, and L. Pyeatt (1995). Hyperplane ranking in simple genetic algorithms. In L. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms.* Morgan Kaufmann.

Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1995). Building better test functions. In L. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms.* Morgan Kaufmann.

Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1996). Evaluating evolutionary algorithms. *Artificial Intelligence Journal 85.*

Whitley, D. and S. Rana (1997). Representation, search and genetic algorithms. aaai.

Whitley, L. D. (1991). Fundamental principles of deception in genetic search. In G.J.E.Rawlins (Ed.), *Foundations of Genetic Algorithms*, pp. 221–241. Morgan Kaufmann.

Wolpert, D. H. and W. G. Macready (1995). No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute.