Journal of Heuristics, 1: 77–104 (1995)
© 1995 Kluwer Academic Publishers

Test Driving Three 1995 Genetic Algorithms: New Test Functions and Geometric Matching

D. WHITLEY
Department of Computer Science, Colorado State University, Fort Collins, Colorado 80523 USA, phone: (303) 491-5373, email: whitley@cs.colostate.edu

R. BEVERIDGE
Department of Computer Science, Colorado State University, Fort Collins, Colorado 80523 USA, phone: (303) 491-5373, email: whitley@cs.colostate.edu

C. GRAVES
Department of Computer Science, Colorado State University, Fort Collins, Colorado 80523 USA, phone: (303) 491-5373, email: whitley@cs.colostate.edu

K. MATHIAS
Department of Computer Science, Colorado State University, Fort Collins, Colorado 80523 USA, phone: (303) 491-5373, email: whitley@cs.colostate.edu

Abstract

Genetic algorithms have attracted a good deal of interest in the heuristic search community. Yet there are several different types of genetic algorithms with varying performance and search characteristics. In this article we look at three genetic algorithms: an elitist simple genetic algorithm, the CHC algorithm and Genitor. One problem in comparing algorithms is that most test problems in the genetic algorithm literature can be solved using simple local search methods. In this article, the three algorithms are compared using new test problems that are not readily solved using simple local search methods. We then compare a local search method to genetic algorithms for geometric matching and examine a hybrid algorithm that combines local and genetic search. The geometric matching problem matches a model (e.g., a line drawing) to a subset of lines contained in a field of line fragments. Local search is currently the best known method for solving general geometric matching problems.

Key Words: genetic, algorithms, test suites, search

1. Introduction

The use of genetic algorithms as optimization tools is now familiar to a broad range of research communities, including Operations Research and Artificial Intelligence. These algorithms show great promise on synthetic and abstracted problems and have been put to good practical use. However, they also come in a variety of forms with a myriad of possible tunings. It is quite difficult to know in advance what type of genetic algorithm to consider for a new application and how to tune it. Moreover, too often results are presented showing that an algorithm solves a problem, but not whether the problem could be more effectively solved using simpler heuristic search methods.

One particularly simple form of heuristic search which predates genetic algorithms is local search (Kernighan and Lin, 1972; Papadimitriou and Steiglitz, 1982). Local search

is, arguably, simpler than genetic search, and therefore when the performance of local search is comparable it is the method of choice. Conversely, genetic algorithms exhibit properties that suggest their intrinisic superiority for some problems and it is important to identify which problems fall into this category.

Because of the additional overhead of a population based search, genetic algorithms are most useful whem it can be shown that these algorithms have advantages over other, simpler heuristic methods. This also suggests that comparisons between different forms of genetic search should also include a baseline comparison between local and genetic search. Moreover, if the results are to be of practical significance, then comarisons should be made on problems largely resistant to local search.

One goal of this article is to provide a basis for better understanding the relative performance of different genetic algorithms and therefore offer guidance to individuals trying to select between different algorithms. A secondary goal is to better understand the types of problems for which genetic search is an appropriate choice of search methods. A new and better set of synthetic test functions has been developed embodying complex nonlinear dependencies similar to those observed in difficult practical problems (Mathias et al. 1994). While the test functions commonly used to evaluate genetic algorithms are more easily solved using local search rather than genetic search, the advantages of genetic search are evident on these more difficult problems. The algorithms compared in this article include a relatively vanilla simple genetic algorithm, a steady state genetic algorithm (Genitor) and the CHC algorithm.

In addition to tests on the new test suite, genetic and local search are compared on a geometric matching problem. Geometric matching exemplifies a problem on which local search has already been shown to work relatively well. The task is to find the best example of a given line model in a set of line segments; for example, one might wish to automatically match the outline of a truck to a subset of line segments extracted from a photograph in which a truck appears. The field of line segments may also include clutter and the matched object may be fragmented or distorted. This is a difficult problem with important practical applications (Beveridge and Riceman, 1995; Beveridge, 1993; Collins and Beveridge, 1993; Fennema et al., 1990). It is therefore interesting to see whether genetic algorithms can improve performance beyond the baseline established using local search.

Several important themes emerge from our studies. The first is the importance of distinguishing problems with strong nonlinear dependencies between state variables from those lacking such dependencies. Most of the common test functions used in the past to evaluate genetic algorithms lack such dependencies. However, both the geometric matching problem presented here and the statics problem in geophysics (Mathias et al., 1994), for example, exhibit strong nonlinear dependencies. Thus, results on relatively simple test functions say little about how heuristic search algorithms will perform on more complex practical applications. The size of application problems also are often much larger than the size of common test problems; thus, scalability of search algorithms is also a critical concern.

The second major theme of this article is the need to understand and take account of domain specific challenges and opportunities and how these interact with assumptions underlying different heuristic algorithms. For example, the evaluation function for the geometric matching problem is defined such that bit string encodings of optimal and near optimal solutions tend to be sparse in terms of the number of 1 bits. Each of the genetic algorithms required some modification in order to accommodate this aspect of the problem.

One final and very important theme emerges from the comparison of genetic and local search on geometric matching: "Will your algorithm recognize when it can stop?" On the new synthetic test functions, the evaluation of the optimal solution is known in advance. This makes terminating upon finding the correct answer straightforward. However, for the geomtric matching problem as well as many other practical problems, the evaluation of the optimal solution is not known in advance. Thus both local search and genetic search face a problem: When do you terminate search? This problem surfaces in a variety of forms for a great many practical problems. For instance, for geometric matching using constraint satisfaction rather than optimization techniques, it has been shown that average case complexity is polynomial if a terminating condition is known; however, given that the terminating condition is generally unknown, average case complexity is exponential (Grimson, 1990).

For local and genetic search, the problem manifests itself in deciding how many times to restart the search algorithm. For genetic search, it is also necessary to define a stopping criterion indicating the population has converged. This is an often neglected and quite important issue. Given the default stopping criterion found in the genetic software package we used, a genetic algorithm hybridized with local search yields performance which is roughly comparable to a local search alone. However, the hybrid genetic algorithm typically "wastes" 50% or more of its time driving the population to convergence after it has already found the best solution. Thus, given a more intelligent stopping criterion, the hybrid genetic algorithm has significant potential for improvement compared to local search.

The comparisons run in this article on the new test functions show that CHC clearly outperforms the other genetic algorithms. On the geometric matching problem local search outperforms the various genetic algorithms—in part because local search can exploit a form of incremental approximate evaluation which cannot be directly exploited by the genetic algorithms. A hybrid algorithm integrates local search as an operator within the genetic algorithm, thus allowing a genetic algorithm to also exploit partial evaluation. Our results suggest that future refinements of the hybrid will produce an algorithm clearly superior to one reliant solely upon local search or genetic search alone.

2. The Algorithm Descriptions

The term genetic algorithm in a sense has come to have two meanings. First, it refers to the "genetic plan" first defined by Holland (1975), which DeJong (1975) called the "genetic algorithm" in this 1975 Ph.D. dissertation. There exist good descriptions of this algorithm (for example, Goldberg, 1989) as well as introductory papers on the genetic algorithm and analysis tools for modeling its behavior (for example, Whitley, 1994). Holland's genetic plan is a population based form of search that represents potential problem solutions as binary strings. In one generation a current population is transformed via selection and reproduction to create a new population. During the selection phase, an "intermediate population" is created that is made up entirely of duplicates of the strings in the current population.

The proportional representation of strings in the population increases or decreases based on fitness. This is accomplished by fitness proportionate reproduction. To determine the fitness factor used to increase or decrease the string's representation in the intermediate population a string's evaluation is divided by the average evaluation of the strings in the

current population. Strings which are better than average increase their representation in the population, while strings that are worse than average probabilistically decrease their representation in the population. After duplicating strings according to fitness, strings undergo "mating." Strings are recombined and mutated to generate new strings representing new potential solutions to the search problem. The creation of the next population via seletion and mating is referred to as one generation. Recombination between two strings can then be applied as follows.

```
1111   111111   1111111111 ⇒ 11110000001111111111
    \ /      \ /
    / \      / \
0000   000000   0000000000 ⇒ 00001111110000000000
```

In this way two parents can generate two offspring by exchanging string fragments. Mutation typically involves changing each bit with a very small probability so that on average only a few bits are changed in a string during the mutation process. In Holland's genetic algorithm the resulting offspring replace the parents. The new population is reevaluated and the process of fitness proportionate reproduction followed by recombination and mutation is repeated.

The second interpretation of "genetic algorithm" is any population-based form of search that uses some form of selection and recombination to generate a new population. The term *evolutionary algorithm* has also very recently come into use (Bäck and Schwefel 1993), but also includes *evolutional programming* (Fogel, Owens, and Walsh, 1966; Fogel, 1994) and *evolutionary strategies* (Bäck, Hoffmeister, and Schwefel, 1991). "Steady state" genetic algorithms differ from Holland's genetic plan in that only 1 offspring is generated at a time and the resulting offspring replaces the worst member of the population. Most algorithms also do not use strict fitness proportionate reproduction. Over time, the average evaluation of strings in the population becomes more uniform and there is less diversity in the population. Consequently, fitness proportionate reproduction results in less and less selective pressure toward the best strings in the population because the fitness of these strings (as normalized by the population average) becomes increasing closer to 1. Rescaling the fitness factor is one solution to this problem (Goldberg, 1989), but several researchers have proposed the use of rank-based selection schemes instead so that selection remains uniform over time (Goldberg, 1990; Whitley, 1989).

We next describe the various algorithms used in the current study in more detail.

2.1. ESGAT

Goldberg's simple genetic algorithm (SGA) (Goldberg, 1989) is close to the original genetic plan described by Holland. There are only two differences between our SGA implementation and Holland's original algorithm. First, the algorithm as implemented here is elitist: the best individual automatically survives from one generation to the next. This variation was tested in DeJong's 1975 Ph.D. dissertation (DeJong, 1975) and is a fairly standard feature of many implementations. Second, instead of using fitness proportionate reproduction we

use tournament selection. We refer to our modified SGA as ESGAT: Elitist SGA with Tournament selection.

Tournament selection (Goldberg, 1990) produces a probabilistic form of rank based selection. Explicit ranking schemes have the added overhead of maintaining a sorted population in order to rank the strings in the population. Tournament selection has the advantage that is does not require the population to be sorted. Tournament selection also allows for convenient parallel execution during the construction of the intermediate population. A basic form of tournament selection works as follows. Two strings are randomly selected from the current population and their fitness values are compared. The string with the best fitness is placed in the intermediate population. This process is then repeated $\mathcal{N}$ times, where $\mathcal{N}$ is the population size. To see that this produces a linear ranking with a bias of 2.0, consider the following. In expectation, every string in the population is sampled twice. The best string will win both tournaments and have a representation of 2 in the intermediate population. The median string will win 1 tournament and lose 1, and thus have an expected representation of 1. The worst string in the population will lose both tournaments. Thus, in expectation, a rank-based selection scheme with a linear bias of 2.0 is induced over the entire population. A lower selection bias can be achieved by probability selecting the best of the two strings in the tournament for insertion into the intermediate population.

2.2. CHC

The CHC adaptive search algorithm (Eshelman, 1991) is a monotonic, generational genetic search algorithm. However, this algorithm differs from the simple genetic algorithm in several ways. The CHC algorithm does not bias the selection of strings for reproduction in favor of those strings with a higher fitness. Instead the algorithm employs a *cross-generational selection* mechanism. Strings are paired for recombination from the parent population randomly and uniformly. Offspring are held in a temporary population. Then a survival competition is held where the *best* $\mathcal{N}$ strings (where $\mathcal{N}$ is the population size) from the parent and offspring populations are selected to form the next generation. Thus, the population is guaranteed to monotonically improve over time. This "selection" scheme differs from that used in the simple genetic algorithm where selection is applied to the parent population before the reproduction phase of the search. Also, no mutation is applied during the recombination phase of the CHC algorithm.

The CHC algorithm does not just randomly pair parents; it also employs *heterogeneous recombination* as a method of "incest prevention" (Eshelman, 1991). This is accomplished by only mating those string pairs which differ from each other by some number of bits (i.e., a mating threshold). The initial threshold is set at $L/4$, where L is the length of the string. When no offspring are inserted into the new population the threshold is reduced by 1. The crossover operator used in CHC, called HUX, is very disruptive and is designed to scatter the sampling of new points in the search space. HUX (Half, Uniform X-over) performs uniform crossover over half of the bits that differ between the two parent strings, where the positions are chosen randomly (Eshelman, 1991).

When no offspring can be inserted into the population of a succeeding generation and the mating threshold has reached a value of 0, CHC infuses new diversity into the population

so that genetic search may be continued. This is done through a form of restart known as *cataclysmic mutation* (Eshelman, 1991). Cataclysmic mutation uses the best individual in the population as a template to re-initialize the population. The new population includes one copy of the template string. The remainder of the strings in the new population are formed by repeatedly mutating some percentage of bits (typically 35%) in the template string and copying the result into the new population. This creates a population that preserves the progress made so far and is biased toward a good solution but with new diversity to continue the search.

2.3. Genitor

Genitor (Whitley and Kayth, 1988) is what Syswerda (1989) and Davis (1991b) describe as a "steady state" genetic algorithm: only 1 offspring is generated at a time and the resulting offspring replaces the worst member of the population. Genitor is therefore similar to CHC in that the best strings that have been found so far are maintained in the population; thus the population is monotonically improving.[1] The other distinguishing feature of Genitor is rank-based selection. The population is maintained in sorted order. The median individual is selected for recombination with a probability of $1/\mathcal{N}$. Using a linear selective bias, B, between 1 and 2, the top ranked individual has a probability $B(1/\mathcal{N})$ of being selected for reproduction, while the worst individual in the population is selected with probability $(2 - B)(1/\mathcal{N})$. The selective pressure for other strings is interpolated linearly according to rank. (Genitor has been implemented using Tournament Selection, but explicit ranking was used in the experiments reported here.) In all of the experiments on test functions, a population of 1000 strings with a selection bias of 1.25 was used for Genitor. The mutation rate was 1/L, where L is the length of the string.

2.4. Local Search: Steepest Ascent, RBC, and RMHC

For the set of synthetic test problems, we include local search to show that exiting problems are solved by simple methods—thus making genetic algorithms an inappropriate search method for this class of problems. Such basic search methods fail to solve the new test problems that are used for comparison in this article. For general problems of geometric matching, local search methods currently provide the best method for generating matches.

All of the test problems examined in this article are parameter optimization problems represented by a binary problem encoding. The geometric matching problem is a combinatorial problem that has a natural binary representation. Using a bit representation, a local neighborhood with respect to any string is defined to be the set of strings that are Hamming distance 1 away. Each of the L neighbors can be reached by changing any of the L bits that make up the string representing the current state. Steepest ascent for this representation involves checking each of the L neighbors and then picking the best of the L neighbors as the next state. The process is then repeated until a local optimum is reached (i.e., all neighbors are no better than the current state.)

Random Bit Climbing is a form of next ascent defined by Dave Davis (1991a). Next ascent is like steepest ascent, except instead of checking all L neighbors for the best improving move, the first improving move which is found is accepted. Every time a new neighborhood is searched for an improving move, Random Bit Climbing randomly orders the sequence in which neighbors are tested.

We also tested a local search method that changes multiple bits. Random Mutation Hill Climbing (RMHC) is a form of stochastic hill climber. In our implementation, each bit is flipped with probability 2/L, so that in expectation 2 bits are flipped at each step—but larger moves in the search space can occur with lower probability. Only improving moves are accepted. RMHC was tested on all of the synthetic test problems, but yielded poorer results than using multiple starts of Random Bit Climbing (RBC). Thus, only results for RBC are reported. There are, of course, more complex forms of local search. Scatter search, as defined by Glover (1994) applies local search to combinations of multiple solutions—thus giving it the ability to vary multiple parameters at a time. Like genetic algorithms, it is also capable of making much larger jumps in the search space than simple neighborhood searches in Hamming space. Our goal in the article, however, is to not to thoroughly compare local search to genetic search, but rather to compare the three forms of genetic algorithm on a set of problems that are known to be resistant to simple hill climbing in Hamming space.

For the geometric matching problem local search methods that change multiple bits were also tested in previous work (Beveridge, Weiss, and Riseman, 1989). However, as with RBC on the new synthetic test functions, the results were poorer than could be obtained with single bit changes and multiple restarts.

3. Some Test Suite Problems

Many of the functions that have been used to test genetic algorithms are small (less than 50 bits) or the test functions can be decomposed such that the individual pieces that make up the test function can be solved independently. For example, the DeJong test functions (labeled Fl through F5) are common test functions, as are the Rastrigin (F6), Schwefel (F7) and Griewangk (F8) functions (see Table 1) (Mühlenbein and Schlierkamp-Voosen, 1993). In addition, the functions labeled F9 and F10 are known as the sine envelope sine wave and the stretched V sine wave functions (Schaffer et al., 1989).

There are serious concerns with many of the existing test functions; methods for constructing new functions have recently been introduced (Whitley et al., 1995a, 1995b). Some of these new test problems are used for comparative purposes in the current study. All of the algorithms were allowed to run to 500,000 evaluations for comparison purposes. All test functions have a global solution with evaluation 0.

3.1. Limitations of Existing Test Problems

While all of the test problems in Table 1 are nonlinear, for most of these problems the interactions between variables are linear. Such problems are separable in the sense that the optimal value for each parameter can be determined independent of all the other variables.

Table 1. Common test functions.

$F1: f(x_i\|_{i=1,3}) = \sum_{i=1}^{3} x_i^2$	$x_i \in [-5.12, 5.11]$
$F2: f(x_i\|_{i=1,2}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$x_i \in [-2.048, 2.047]$
$F3: f(x_i\|_{i=1,5}) = \sum_{i=1}^{5} \lfloor x_i \rfloor$	$x_i \in [-5.12, 5.11]$
$F4: f(x_i\|_{i=1,30}) = \left[\sum_{i=1}^{30} ix_i^4\right] + Gauss(0, 1)$	$x_i \in [-1.28, 1.27]$
$F5: f(x_i\|_{i=1,2}) = \left[0.002 + \sum_{j=1}^{25} \frac{1}{j + \Sigma_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$	$x_i \in [-65.536, 65.535]$
$F6: f(x_i\|_{i=1,N}) = (N * 10) + \left[\sum_{i=1}^{N} (x_i^2 - 10cos(2\pi x_i))\right]$	$x_i \in [-5.12, 5.11]$
$F7: f(x_i\|_{i=1,N}) = \sum_{i=1}^{N} -x_i sin(\sqrt{\|x_i\|})$	$x_i \in [-512, 511]$
$F8: f(x_i\|_{i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} (cos(x_i/\sqrt{i}))$	$x_i \in [-512, 511]$
$F9: f(x_i\|_{i=1,2}) = 0.5 + \frac{sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{[1.0 + 0.001(x_1^2 + x_2^2)]^2}$	$x_i \in [-100, 100]$
$F10: f(x_i\|_{i=1,2}) = (x_1^2 + x_2^2)^{0.25}[sin^2(50(x_1^2 + x_2^2)^{0.1}) + 1.0]$	$x_i \in (-100, 100]$

Of the DeJong functions, Fl, F3 and F5 are separable functions that are always solved by decomposition. (F4 is also separable, although the addition of noise might prevent an algorithm from locating an optimum.) F6 and F7 are also separable. Only F2, F8, F9 and F10 are nonseparable, nonlinear problems. Yet, F2, F9 and F10 are relatively small problems that are not scalable.

Davis (1991a) has shown that many of the DeJong functions are quickly solved by Random Bit Climbing. Additionally, Mühlenbein and Schlierkamp-Voosen (1993) have used functions F6, F7 and F8 to argue that the "Breeder Genetic Algorithm" scales such that $O(n\ ln(n))$ function evaluations are needed to locate the global optimum, where n is the number of parameters used by these functions. However, Whitley et al. (1995a) have shown that problems F6 and F7 can be exactly solved in $O(n)$ time using simple hill climbing methods.

Only F8 (Griewangk's function) would appear to be scalable, nonlinear and nonseparable. However, the summation term of F8 induces a parabolic shape while the cosine function in the product term creates "waves" (which create local optima) over the parabolic surface. Thus, as the dimensionality of the search space of F8 is increased the contribution of the product term involving the cosine becomes smaller and the local optima become smaller. The function becomes simpler and smoother in numeric space, and thus easier to solve, as the dimensionality of the search space is increased (Whitley et al., 1995b).

In Table 2 we represent results for 10, 20, 50 and 100 dimensional versions of Griewangk's function using a Gray coded version of the function. *Nb Var* is the number of total variables, *Mean Soln* is the best solution found averaged over 30 runs and *sigma* is the standard deviation. *Succ* is the number of times the global optimum is found, and *Mean Trials* is the number of evaluations for those cases where the global optimum is found. *Restarts* refers to the average number of times local search is restarted. Clearly, as the dimensionality of the function increases Random Bit Climbing requires fewer restarts to find an initial random starting point in the basin of attraction of the global solution. The various genetic algorithms perform reasonably well on this problem, but are not competitive with RBC at higher dimensions.

Table 2. Result for a Gray code version of simple F8. *Nb Var* is the number of variables, *sigma* is the standard deviation. *Succ* is the number of times the global optimum is found, and *Mean Trials* is the number of evaluations required to find the global optimum. *Restarts* refers to the average number of times local search is restarted. All results are based on 30 experiments, with 500,000 evaluations per experiment.

Nb Var	Mean Soln.	σ	Succ	Mean Trials	Restarts
ESGAT Pop 200 Pc 0.90:					
10	0.0515	0.0381	6	354422	
20	0.0622	0.0400	5	405068	
50	0.0990	0.0564	0		
100	0.262	0.0459	0		
CHC:					
10	0.00		30	51015	
20	0.00		30	50509	
50	0.00104	0.00569	29	182943	
100	0.0145	0.0231	20	242633	
Genitor:					
10	0.00596	0.0121	25	92239	
20	0.0240	0.0289	17	104975	
50	0.0170	0.0284	21	219919	
100	0.0195	0.0321	21	428321	
RBC:					
10	0.00212	0.00808	28	184105	313.3
20	0.00		30	82215	56.9
50	0.00		30	33789	7.4
100	0.00		30	34119	3.0

4. Constructing New Test Problems

One way to introduce nonlinear interactions and still retain scalability is to use a nonlinear function of two variables. $F(x, y)$, as a starting primitive function. The function can then be scaled to three variables, for example, by constructing a new function E–F(x, y, z) where:

$$\text{E–F}(x, y, z) = F(x, y) + F(y, z) + F(z, x).$$

We will refer to E–F as an *expanded* function. The expanded function E–F is no longer "separable" and induces nonlinear interactions across multiple variables. Furthermore, this strategy can be extended to provide different degrees of nonlinearity. Consider the following matrix:

	q	x	y	z
q		qx	qy	qz
x	xq		xy	xz
y	yq	yz		yz
z	zq	zx	xy	

where the variables q, x, y, z are labels along the left and top edges and appear as variable pairs in the matrix. The *full matrix* scaling strategy uses $O(n^2)$ primitive subfunctions. The nonlinear interactions are such that every variable interacts with every other variable. Note that the cost of executing the evaluation funciton also scales; the cost cost of the full matrix evaluation grows as a function of $O(n^2)$. Functions built in this way are very similar to the evaluation function used in problems such as the "statics problem" in seismic data process, where the evaluation function is summed over correlations computed over pairs of seismic traces (Mathias, 1994).

F10 provides a convenient example of a function that has nonlinear interactions between two variables. Each parameter of F10 is encoded with 22 bits. In 2 dimensions, this function has many local optima, yet overall the function is regularly structured—which should make the function well suited to search via genetic algorithms.

After scaling up the expanded E–F10 function to 10, 20, and 50 variables we compared the performance of the three evolutionary algorithms using a Gray code representation (Whitley et al., 1995b; Mathias and Whitley, 1994). Results are shown in Table 3. The RBC results show that this problem is not readily solved by simple hill climbing, especially as the problem is scaled up. The evolutionary algorithms frequently locate the optimal solution at 10 dimensions. At 50 dimensions, CHC scales up with reasonable effectiveness while bothe ESGAT and Genitor have problems.

4.1. Composite Functions

Many of the test functions in Table 1 have been introduced into the literature because of interesting properties, but they lack nonlinear interactions between variables. Simple function

Table 3. Results for a Gray coded E–F10.

Nb Var	Mean Soln.	σ	Succ	Mean Trials
ESGAT Pop 200 Pc 0.90:				
10	0.572	2.254	25	282299
20	1.617	1.340	2	465875
50	770.576	584.476	0	
CHC:				
10	0.00		30	51946
20	0.00		30	139242
50	7.463	8.351	3	488966
Genitor:				
10	0.00	0.00	30	136950
20	3.349	3.276	4	339727
50	294.519	151.039	0	
RBC:				
10	27.935	17.084	0	
20	363.111	148.950	0	
50	4158.513	836.189	0	

composition can be used to convert these functions into a nonlinear function of 2 variables. The inner primitive function of the composition takes in two variables and outputs a single value which falls into the domain of the outer primitive function. This technique, when combined with the matrix expansion methods discussed previously, provides a method for constructing complex nonlinear fitness landscapes.

In the simplest case, a separable function such as Rastrigin (F6) might be expanded as follows. A separable function of the form

$$F(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} S(x_i)$$

becomes

$$\text{E–F}(x_1, x_2, \ldots, x_n) = S(T(x_n, x_1)) + \sum_{i=1}^{n-1} S(T(x_i, x_{i+1}))$$

where S is the subfunction used in the original function and the transformation function T maps two variables onto the domain of S. Results for an expanded version of F6 denoted E–F6 (AVG) uses a simple average (i.e., T(x,y) = (x + y)/2) for the transformation function are given in Table 4 for 10, 20 and 50 dimensions. Here again, RBC fails to solve the function, while ESGA and Genitor are competitive with CHC at lower dimensions, but CHC produces solutions an order of magnitude better when the function is scaled to higher dimensions.

Table 4. Results for E–F6 (AVG) using a Gray coded representation.

Nb Var	Mean Soln.	σ	Succ	Mean Trials
ESGAT Pop 200 Pc 0.9:				
10	14.392	21.448	20	90102
20	160.515	145.433	12	306583
50	2429.129	1229.784	0	
CHC:				
10	10.665	$7.988	11	33536
20	11.998	45.660	28	118935
50	144.502	361.058	18	398969
Genitor:				
10	26.662	19.175	10	56483
20	107.982	89.674	12	109041
50	1148.460	399.944	0	
RBC:				
10	43.094	11.853	0	
20	367.405	62.480	0	
50	4279.672	318.973	0	

One can also construct composite primitive functions using the Griewangk function (F8); by using F8 as a one-dimensional primitive function the scaling problem associated with the cosine term is eliminated. E–F8 (AVG) again uses simple averaging at the transformation function. This has the effect of making the problem harder at higher dimensions and also makes the function more resistant to hill-climbing by RBC. We also constructed a composite function E–F8 (F2) where DeJong's F2 was used as the inner primitive transformation function. Even CHC now has a more difficult time finding good solutions for E–F8 (AVG) and E–F8 (F2) at higher dimensions. (See Tables 5 and 6). However, CHC still outperformed ESGAT and Genitor.

4.2. *Discussion*

The test functions used here provide a much better test of genetic search algorithms than the existing test suites. These functions include more complex nonlinear interactions than existing test problems and also allow the study of how performance is affected by scaling the dimension of the test suite. The results show CHC to be a very effective form of heuristic search compared both to the elitist Simple Genetic Algorithm using tournament selection and Genitor. All of the new test problems are resistant to simple hill climbing by next ascent in Hamming space. This stands in sharp contrast to the existing test suites, which we have found can be solved more quickly by simple local search methods than by genetic algorithms (Whitley et al., 1995a).

The results suggest that CHC is generally more effective than Genitor and ESGAT. Aside from its performance, CHC also has the advantage that is required no parameter tuning. CHC uses a standard population size of 50. This is much smaller than the population size used by Genitor (1000 for all of these experiments) and by ESGAT (200). Increasing the

Table 5. Results for a E–F8 (AVG).

Nb Var	Mean Soln.	σ	Succ	Mean Trials
ESGAT Pop 200 Pc 0.90:				
10	3.131	0.942	0	
20	8.880	4.296	0	
50	212.737	36.803	0	
CHC:				
10	1.283	1.107	10	222939
20	8.157	4.114	0	
50	83.737	10.922	0	
Genitor:				
10	1.292	0.936	5	151369
20	12.161	2.232	0	
50	145.362	23.746	0	
RBC:				
10	4.738	0.9002	0	
20	42.779	8.443	0	
50	640.625	119.999	0	

Table 6. Results for E–F8 (F2).

Nb Var	Mean Soln.	σ	Succ	Mean Trials
ESGAT Pop 200 Pc 0.90:				
10	4.077	2.742	0	
20	47.998	32.615	0	
50	527.100	176.988	0	
CHC:				
10	1.344	0.921	0	
20	5.630	2.862	0	
50	75.0995	49.644	0	
Genitor:				
10	4.365	2.741	0	
20	21.452	19.459	0	
50	398.120	220.284	0	
RBC:				
10	0.139	0.422	0	
20	7.243	11.289	0	
50	301.561	72.745	0	

population size for CHC (or changing CHC in any other way for that matter) results in poorer performance on all the test functions considered in this article. Only on one large seismic application with approximately 500 variables—the "statics problem"—have we found that CHC works better with a population size of 200 (Mathias et al., 1994).

We should reemphasize that the current experiments were designed to compare the three genetic algorithms and that Random Bit Climbing was used to demonstrate that the problems

are resistant to simple hill-climbing. More complex forms of local search would no doubt perform better on these problems. It should also be pointed out that these test functions allow a form of fast incremental evaluation. Note that there are $O(N^2)$ subfunctions in an expanded function, but that any one parameter appears in only $O(N)$ of these function subfunctions. Thus, any form of local search that tests changes that affects only a single parameter can use a fast form of incremental evaluation, and thus would have a significant advantage compared to genetic search. Incremental evaluation also plays a critical role in the Geometric Matching problem discussed in the next section. For geometric matching, we are able to utilize a form of steepest ascent that uses a rapid heuristic approximate evaluation method to filter neighbors from further consideration. This reduces the cost of evaluating all L neighbors during one step of steepest ascent.

5. Near Optimal Geometric Matching

5.1 Background

For several years Beveridge (Beveridge, Weiss, and Riseman, 1991; Beveridge, 1993; Beveridge and Riseman, 1995) has studied the use of local search as a way of finding near optimal solutions to geometric matching problems. Here, geometric matching is the problem of solving for the optimal correspondence mapping and geometric transformation which relate a model to a set of data. For example, the model may be a line drawing and the data might be line segments extracted from a photograph. Problems of this type arise in a number of areas and are specifically relevant to problems associated with computer vision. Variations of local search are currently being used to solve geometric matching problems in the context of semi-autonomous photo-interpretation (Collins and Beveridge, 1993) and robot navigation (Fennema et al., 1990).

In this section, we compare local search techniques to CHC and Genitor and present initial results on a set of geometric matching problems. These matching test problems are useful because factors such as problem size, object model structure, data corruption and image clutter are all controlled.

The matching problems are extremely challenging for heuristic combinatorial optimization algorithms. The evaluation function exhibits a global connectivity missing from the simple test functions typically used to evaluate general heuristic search algorithms. To be specific, with elements of the discrete search space encoded as bit strings, a change to *any* bit changes the relative worth of *all* other bits. The reason for this connectivity is that changing any bit changes the correspondence mapping between object model and data. This, in turn, almost certainly changes the global best-fit transformation aligning model and data. Finally, this change in alignment changes the relative value or quality of all other pairs of corresponding features.

There are 32 distinct matching problems in the test data set. These problems involve relatively simple "stick figure" models. These geometric models are defined as sets of 2D straight line segments. In matching, these models may be rotated and translated to lie anywhere in an image. In addition, the size of the models is allowed to vary. The four models are shown in Figure 1.

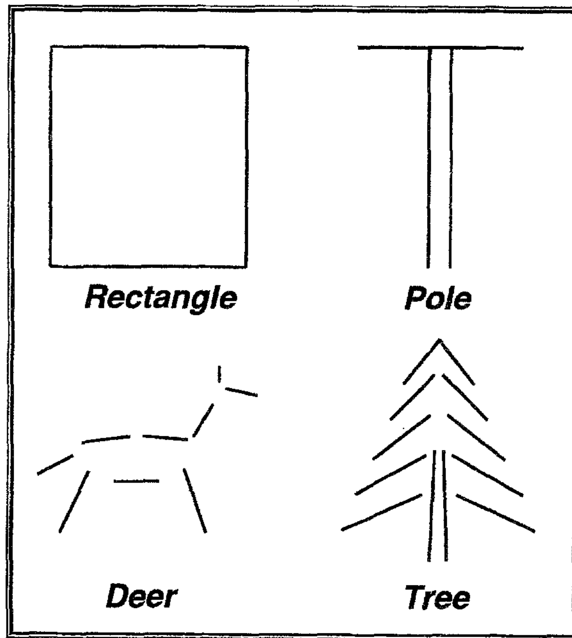

Figure 1. Four stick figure models used in geometric matching tests.

The Pole is an interesting test case because it is so simple: it is comprised of only three segments. Many heuristic matching techniques use distinctive local structures to direct search (Bolles and Cain, 1982; Lowe, 1985). The lack of such features in the Pole hinder these approaches. The Tree is interesting because it exhibits some partial symmetry: the model may shift up or down in the branch structure. Incorrect, yet relatively good evaluations can be produced in this way. Symmetries complicate matching for many well established techniques such as tree search (Grimson, 1990).

Figure 2 shows a sampling of the 32 problems. Figure 2a shows a relatively simple problem in which simulated image data has been generated by fragmenting and skewing

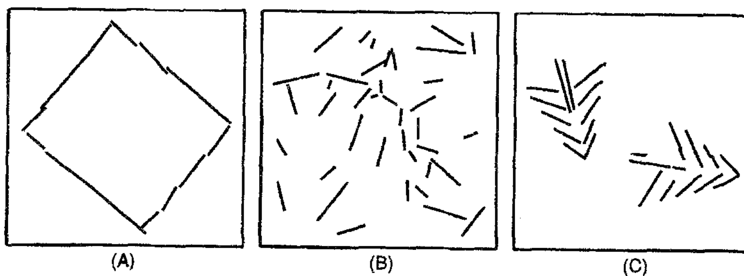

Figure 2. Sample of synthetic test data for matching: a) Rectangle model randomly placed, scaled and corrupted, b) Corrupted Deer model with random clutter, c) Multiple corrupted instances of the Tree model.

fragments of the original model line segments. This corruption has been carried out by a Monte Carlo simulator which steps through a multi-stage degradation process. The simulator also rotates, translates and scales the model so that the placement and size of the model is unknown to the matching algorithm. Figure 2b shows a problem in which randomly distributed clutter segments are added. Figure 2c shows an example in which multiple instances of the same model are present. While visual inspection would suggest that Figure 2b involving clutter represents a more difficult problem than Figure 2c, in reality, objects with multiple corrupted instances are often the most challenging for heuristic search methods. Problems are classified according to model type (Pole, Tree, Rectangle, Deer) as well as by string length and whether there are multiple objects (M) or clutter (C).

An example of a best match is shown in Figure 3. This match is for the Tree model. The line segments making up the model are labeled with letters and are shown on the left. The data line segments include three instances of the Tree and are shown to the right. The model is overlaid on top of the data in the best match position. The correspondence matrix indicates which pairs of model and data segments are part of this best match. Each square in the table may be thought of as representing one bit in the bit string encoding of the match. The filled in squares correspond to 1s in the bit string.

The evaluation function used to rank alternative matches is complex. Briefly, the function is defined over a space of possible mappings from the model to image segments. Letting $\mathcal{M}$ be the set of model segments, D the set of data segments, and S the cross product of

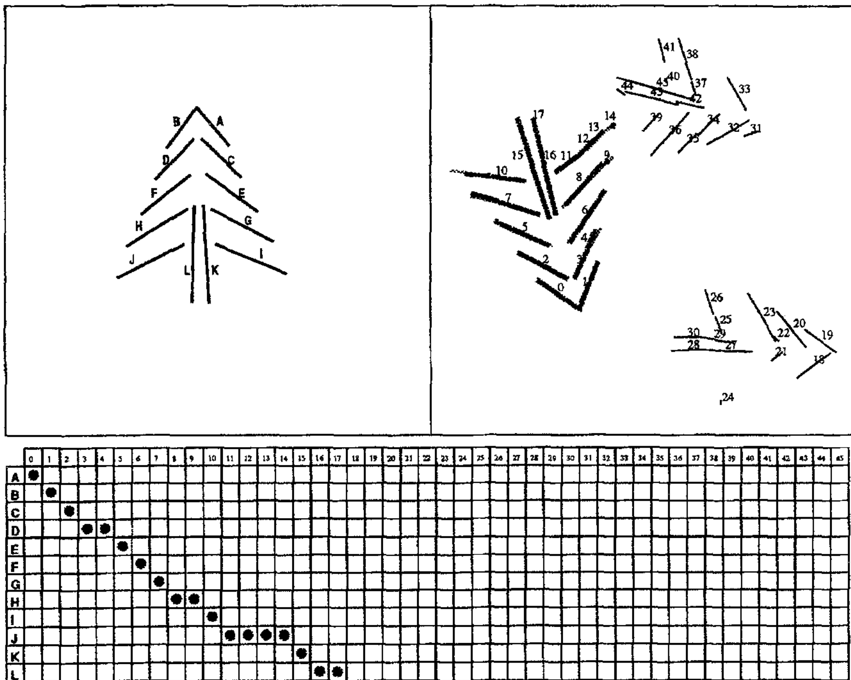

Figure 3. Example of a best match for one of the 32 matching problems.

$\mathcal{M}$ and D, the correspondence space C is the power set of S. This means the search space C consists of all possible combinations of pairings between model and data segments.

The match error, E_{match}, is defined over the correspondence space. In many practical applications, prior knowledge can be used to limit the sets of pairings considered in the search space, but here the most general case with $S = \mathcal{M} \times D$ and $C = 2^S$ is considered. While the complete search space contains all 2^L possible combinations (where L is string length), through biasing the selection of matches from which search is initiated, attention is focused upon the regions of space containing sparse solutions. Search is typically initialized with start states where the number of bits turned on in the binary encodings is related to the number of model segments, $\mathcal{M}$.

Intuitively, the match error measures two things: 1) the degree to which the model *fits* the corresponding data segments and 2) the degree to which the portions of the model are *omitted* from the match. Details of how fitting is performed and omission measured can be found in an early paper on this work (Beveridge, Weiss, and Riseman, 1991) and a much more detailed account appears in Beveridge's (1993) dissertation. While this article will be limited to comparing search techniques on 2D problems of the type described above, the local search matching work has been extended to handle matching of 3D models to image data (Beveridge and Riseman, 1995).

5.2. Empirical Results Comparing Genetic and Local Search

5.2.1. Local Search. The binary encoding for matching assigns a unique bit to every pairing of model and image line segments in the set $S = \mathcal{M} \times D$. Therefore, if $|S| = L$, then the search space C maps to the space of all possible bit strings of length L. A 1 bit in the ith position of the string indicates the ith pair $s_i \in S$ is part of the match. The local search neighborhood consists of all strings within Hamming distance 1 of the current match.

Local search is initiated from randomly and independently selected initial matches. However, a bias is introduced which favors initial matches with approximately the same number of matching pairs as are expected in the optimal match. For example, if on average a model line matches 2 data lines in the best match, the goal is to generate initial starting matches with 2 data lines per model line. For each subsequence of the string associated with a different model line m, if there are k data segments to which m might match, each pair in this sequence is assigned a 1 with independent probability q/k. The parameter q determines, on average, how many data segments are matched to a given model segment. In all experiments reported here $q = 2$.

Using the bit string encoding, toggling every bit to generate the Hamming distance 1 neighborhood adds or removes individual pairs of model-data segments. If a pair s is in the current match, then the result of removing s is tested. If a pair s is not in the current match, then the result of adding s is tested. Using this neighborhood, local search naturally tends to interleave the addition and removal of pairs until it converges upon a locally optimal match. This neighborhood definition does not permit one data segment to be swapped for another in a single move.

Our problem presents an example of a case in which local search, because it is evaluating states which are minor perturbations of a current state, can take advantage of this knowledge

to speed testing. In principle, for every match tested, the model must be completely fit to the corresponding data and the associated omission over the entire model computed. However, the exact change in fit error can be computed incrementally relative to the current match. This allows for very efficient evaluation of the new fit error (Beveridge, 1993). In contrast, omission must be recomputed from scratch and in practice this requires an order of magnitude more computation than does computing the new fit error.

Given the ability to incrementally compute the change in fit error, a useful heuristic is to first check the change in fit error, and then only compute the omission change if the change in fit suggests improvement is possible. The relative savings are illustrated in Table 8. Incremental evaluation can be used with the local search algorithm because local search only changes a single bit at a time. The recombination operator used by genetic algorithms produces much larger jumps in Hamming space, thus making it impossible to use the simple incremental evaluation exploited by local search. Incremental evaluation also reduces the cost difference between steepest ascent and Random Bit Climbing. Preliminary experiments suggested that steepest ascent was more effective, and since there is little cost difference between steepest ascent using incremental evaluation versus RBC, steepest ascent was used.

5.2.2. ***Genetic Search.*** We applied both the Genitor algorithm and the CHC algorithm to the model matching problem. To perform these experiments, the Genitor Software Package (a version of which also includes CHC) has been merged with the local search geometric matching software. Thus, the functions and parameter settings used to evaluate members of the genetic algorithms population are exactly the same as those used to evaluate matches by the local search algorithm.

The standard constant mutation rate was replaced by a mutation rate which varies as a function of the degree of convergence within the population. A previously defined adaptive mutation operator (Starkweather, Whitley, and Mathias, 1990) was tested but resulted in poor results: for this operator the probability of mutation, P_M is given by *MaxRate/PerDiff*, where *MaxRate* is the maximum mutation rate (set to 0.2) and *PerDiff* is the percentage of bits that are different between the two parents currently being recombined. The adaptive mutation algorithm created difficulties, especially for the larger matching problems. The source of the problem is that bit encodings of optimal solutions become increasingly sparse as problem size increases. To see why this is so, recall that string length L is the product of the number of line segments in the model m and the number of line segments in the image d. Typically, $d \geq m$, and so $L \geq m^2$. Thus, while string length grows as a function of m^2, the number of bits expected to be 1 in most solution strings remains $O(m)$.

To give an example, for the matching problem involving the Tree model and 30 random clutter lines, $m = 12$, $d = 43$, and thus $L = 516$. Assuming the genetic algorithm is converging upon the proper match with 12 bits set to 1, a mutation rate of 1% will turn on an additional 5 bits. This is enough to prevent convergence. Once identified, this problem has a relatively simple fix, and that is to make the mutation rate vary in response to the expected degree to which the desired solutions have sparse encodings. A linearly changing rate is computed where the highest rate is determined by the number of model lines m, and the overall bit-string length n. The highest number of changes desired is set at 25% of the number of model lines. The high end is then set so that on average, this maximum

number of bits are changed in the overall bit-string. The low rate is set such that on average, one mutation would take place in the entire bit-string. The mutation rate is varied linearly between these two extremes based on the Hamming distance between parents. The largest number of differences expected is set at four times the number of model lines. The smallest number of differences expected is of course one. If the variance between parents exceeds the expected max, the mutation rate is clamped at the low end.[2]

It is important that a good global sample of the space be represented in the population; at the same time, the desire to generate solutions in times that are competitive with local search also dictated the use of relatively small populations. The population size is set at one half of the bit-string length, with a minimum of 50, and a maximum of 300.

The same algorithm used to pick initial matches for local search was used to seed the initial populations for both CHC and Genitor. Thus, these initial strings are relatively sparse in terms of 1 bits. Recall this strategy is used because most solutions lie near the origin of the hypercube corresponding to the the search space. Consequently, most strings in the population are relatively near each other in Hamming space. Even if two matches share no 1 bits in common, they still only differ by approximately 2 times the number of lines in the model.

Several mechanisms used by CHC assume that the binary strings being recombined are initially drawn uniformly from the search space. The incest prevention mechanism of CHC requires that parents differ by some minimal Hamming distance. Strings in the initial seeded population often fail to satisfy this requirement; this failure becomes worse as CHC struggles to converge. HUX also uses Hamming distance to vary the number of bits swapped during uniform crossover. It starts at exactly half of the differing bits in the two parents, but as the Hamming distance between parents decreases, the number of differing bits swapped during crossover is adaptively reduced. Since Hamming distance is low in the seeded population, HUX's adaptive mechanisms are prematurely triggered. This causes the search to stall, which in turn triggers the restart mechanism used by CHC. High mutation rates (e.g. 35%) are used as part of the Cataclysmic mutation restart. This caused the same problems that Genitor experienced with higher adaptive mutation rates: the resulting strings were not sufficiently sparse. We attempted to correct these problems in the CHC algorithm by resetting Hamming distance threshold mechanisms and mutation rates; instead of using the length of the string to determine relative Hamming distance, we instead used m, the number of model lines. This improved the performance of CHC, but Genitor still outperformed CHC. CHC has many interrelated mechanisms that help it to more fully explore the entire search space. The fact that the geometric matching problem has solutions that are largely restricted to strings near the origin of the hypercube appears to conflict with the fundamental design of the CHC algorithm. Genitor is less specialized, and less sensitive to the sparseness of 1 bits in most strings which are generated during search. Thus, while CHC dominated the other algorithms on our test suite, it failed to generate competitive solutions on the geometric matching problems.

Table 7 illustrates the relative performance of the enhanced CHC+ algorithm, Genitor and Genitor+ using the new adaptive mutation operator. The results illustrate the relatively poor performance of CHC+ compared to Genitor. One key measure of performance is the probability of converging upon the correct solution.[3] The maximum likelihood estimate of this probability, P_s, is simply the percent of correct solutions found when running the

Table 7. Comparison of CHC+, Genitor, and Genitor+ on 32 test problems. CHC+ is the CHC algorithm with thresholds reset with respect to number of model lines. Genitor uses a constant mutation rate, while Genitor+ uses the new adaptive mutation operator that is also sensitive to number of model lines. *Params* describes problem parameters: the number indicates the length of the string encoding, while M indicates multiple objects and C indicates clutter. P_s denotes percent of time that a globally optimal solution is found. *Evals* refers to the number of evaluations.

		CHC+		Genitor		Genitor+	
Model	Params	P_s	Evals	P_s	Evals	P_s	Evals
Pole	12C	100	879.9	88	1051.0	98	88.5
Pole	24M	88	1528.7	72	1521.0	86	1411.6
Rectangle	28C	69	2024.6	44	1461.0	72	2282.5
Pole	42C	54	2162.3	42	3201.0	48	1302.0
Pole	42M	44	2120.9	26	1871.0	38	1973.3
Rectangle	52M	41	3099.5	48	2211.0	54	2175.1
Rectangle	68C	52	2902.2	56	2161.0	54	2263.9
Pole	72C	42	2447.5	32	1411.0	14	1069.4
Deer	81C	68	4065.7	50	2831.0	80	5418.2
Pole	81M	16	3656.7	24	2471.0	24	1829.0
Pole	96M	11	3579.2	30	2151.0	10	1521.6
Deer	99M	49	4849.7	62	2901.0	68	6858.8
Pole	102C	20	2774.5	26	1511.0	22	1256.6
Rectangle	108C	15	3341.9	48	2781.0	34	4349.0
Rectangle	108M	1	4665.7	16	3631.0	16	6231.1
Rectangle	124M	5	4140.0	14	3701.0	6	3512.7
Rectangle	148C	16	3970.3	28	7481.0	34	5459.8
Tree	156C	28	7387.7	28	6171.0	30	21998.1
Rectangle	168M	3	4807.7	16	5221.0	12	4814.4
Deer	171C	32	6143.1	24	5541.0	54	19596.0
Deer	180M	21	6269.2	38	6241.0	44	13659.9
Tree	216M	12	9321.8	50	9791.0	32	41698.0
Deer	261C	7	7517.4	35	12911.0	40	30488.6
Deer	261M	8	7706.3	26	14891.0	40	27215.9
Tree	276C	14	9848.8	32	19471.0	46	47862.2
Deer	342M	2	8802.8	12	38341.0	28	34512.4
Deer	351C	4	7638.3	42	39791.0	40	27949.6
Tree	396C	9	11501.4	40	62690.9	68	77713.3
Tree	432M	3	13442.1	18	96520.3	28	90889.3
Tree	516C	3	12215.8	8	100000.0	44	96694.7
Tree	552M	1	14136.0	0	100000.0	4	99683.2
Tree	780M	0	13253.4	0	10000.0	0	100000.0

genetic algorithm multiple times. These probabilities are shown in Table 7 for each of the three algorithms. Table 7 also shows the the number of evaluations required to reach the stopping criterion which terminates the genetic search in the column labeled *Evals*.

5.2.3. ***Comparing Relative Performance.*** Neither the local search algorithms nor the genetic algorithms converge upon the globally optimal match in all cases. Therefore, in comparing algorithms for model matching, two things must be taken into consideration. First is the likelihood that the algorithm once initiated will converge on the optimal solution. The second is how much work must be performed in order to reach convergence. The first is easily

measured for sample problems as illustrated above in Table 7. The second is more problematic. Typically, when evaluating different local search algorithms, the measured CPU time on a given system using a given implementation has been used as the measure of work.

Given an estimate of the average run-time required per trial of local search r, and the probability P_s that local search converges upon the optimal match, the run-time to find the correct solution with probability Q_s can be estimated as follows. Assuming search is initiated from randomly and independently chosen start states, then in running t trials, the probability Q_f of failing to find the optimal match in one or more trial drops exponentially as a function of t.

$$Q_f = (P_f)^t, \qquad \text{where } P_f = 1 - P_s. \tag{1}$$

Given this relationship, it is possible to compute the number of trials t_s required to find the optimal match with probability $Q_s = 1 - Q_f$:

$$t_s = \lceil \log_{P_f} Q_f \rceil \,. \tag{2}$$

The expected time r_s required to solve a problem with confidence Q_s is the product of r and t_s.

Despite its apparent objectivity, there are a number of problems with using execution time to compare the efficiency of local search and genetic search on the geometric matching problem. One important factor is that the local search is relatively simple and approximately 4 years have been spent developing the local search code so that it executes quickly. The Genitor software, on the other hand, has been built as a development environment; it was build to be easy to understand and flexible to use and not necessarily for extremely efficient execution. The geometric matching work using Genitor software has been done in approximately 4 months and there has been no attempt to change the software to alter its execution time.

One attractive alternative to total run-time is to compare number of evaluations. However, this too is unsatisfactory because different evaluations require a different amounts of time—even under the full evaluation function. The partial evaluation used by local search complicates matters further. Therefore, in this section we treat the execution time required by local search as a baseline for comparison. If a genetic algorithm can produce comparable results or improve on local search, then this approach warrants further attention as a means of solving geometric matching problems.

In the comparisons which follow, the values of P_s and r are estimated based upon many runs of each algorithm on individual test problems. Local search was run 1000 times, Genitor+, which has a higher execution cost, was run 50 times. Based upon these estimates, the expected run-time r_s for each problem is reported and compared for different algorithms. The confidence threshold, Q_s, will be set to 0.95 in this study.

5.3. Local Search and Genitor

In order to better understand the impact of incremental evaluation, Table 8 presents results for local search using the Full evaluation function (*LS Full*), local search using the approximate

Table 8. Comparison of steepest ascent local search using partial incremental evaluation (LS-Inc), steepest ascent local search using full evaluation (LS-Full), and Genitor+ algorithm on 32 test problems.

		Trials Required: t_{95}			Estimated Run Time		
Model	Params	LS Inc.	LS Full	Genitor+	LS Inc.	LS FUll	Genitor+
Pole	12C	6	5	1	0.3 *	0.2 *	0.3
Pole	24M	8	7	2	0.8 ○	0.7 *	1.4
Rectangle	28C	6	6	3	0.6 *	0.6 *	3.6
Pole	42C	26	20	5	2.6 ○	2.0 *	4.0
Pole	42M	12	10	7	1.2 ○	1.0 *	9.1
Rectangle	52M	5	5	4	1.0 *	1.5 ○	7.2
Rectangle	68C	11	9	4	1.1 *	2.7 ○	8.4
Pole	72C	35	39	20	3.5 *	7.8 ○	20.0
Deer	81C	13	11	2	3.9 *	8.8 ○	12.2
Pole	81M	37	29	11	3.7 *	8.7 ○	20.9
Pole	96M	45	24	29	9.0 *	9.6 ○	52.2
Deer	99M	19	15	3	7.6 *	18.0 ○	25.5
Pole	102C	96	72	13	9.6 *	21.6	20.8 ○
Rectangle	108C	47	17	8	9.4 *	10.2 ○	41.6
Rectangle	108M	12	13	18	3.6 *	10.4 ○	138.6
Rectangle	124M	14	15	49	5.6 *	13.5 ○	240.1
Rectangle	148C	42	26	8	12.6 *	23.4 ○	67.2
Tree	156C	42	32	9	33.6 *	118.4 ○	361.5
Rectangle	168M	20	24	24	10.0 *	72.0 ○	206.4
Deer	171C	29	34	4	17.4 *	85.0 ○	137.6
Deer	180M	31	33	6	21.7 *	99.0 ○	152.4
Tree	216M	51	52	8	66.3 *	291.2 ○	331.5
Deer	261C	43	67	6	43.0 *	341.7 ○	414.0
Deer	261M	59	46	6	59.0 *	239.2 ○	372.0
Tree	276C	51	52	5	76.5 *	473.2 ○	617.5
Deer	342M	106	67	7	159.0 *	576.2 ○	984.9
Deer	351C	64	66	6	96.0 *	607.2	517.2 ○
Tree	396C	52	70	3	114.4 *	1,113.0	829.2 ○
Tree	432M	135	99	7	310.5 *	1,871.1 ○	2,405.2
Tree	516C	99	96	5	297.0 *	2,438.4	2,156.5 ○
Tree	552M	149	106	75	461.9 *	3,307.2 ○	35,317.5
Tree	780M	299	175	—	1,554.8 *	10,202.5 ○	

incremental evaluation (*LS Inc*) and Genitor+ using the full evaluation function. The results indicate that a single "trial" of genetic search using Genitor+ almost always has a much higher probability of locating a global optimum than local search. Of course, the cost per trial for genetic search (i.e., running Genitor+ until the stopping criterion is met) is very costly compared to local search. The *Estimated Run Time* associated with a 95% probability of finding the global optimum shows that Genitor+ is generally poorer than local search using the full evaluation function, but its performance is sometimes competitive on some of the larger problems. (For the two largest problems, it is not clear that a sample of 50 runs is sufficient to adequately estimate the probability of convergence for Genitor+.) However, this table also makes it clear that incremental evaluation gives local search a large advantage when compared using Estimated Run Time. In Table 8 a * appears next to the

lowest run-time t_s and a ○ by the next best for each problem. For all but 3 of the smaller problems, local search using incremental evaluation is best, and the difference is dramatic for larger problems.

5.4. *Hybrid Genetic and Local Search*

One way in which the genetic algorithm can also exploit fast incremental evaluation is to include local search as an operator to improve the strings in the population. Such algorithms are often called "Hybrid Genetic Algorithms" (Davis, 1991b).

The genetic algorithm provides a population of strings for local search to act on, and local search in turn inserts a much better string into the population. Again, the best solution found is returned when the population has converged to within a user-specified tolerance. The parameters used for Genitor+ are unchanged.[4]

Another question which has received attention in the last few years in the genetic algorithm community is the rate with which local search is to be done. Some hybrid algorithms (Mühlenbein, 1991) apply local search to every string. More recently, some researchers have argued that local search should only be applied to a small percentage of the population in order to best exploit local search. We tried both strategies. In one approach local search is applied to every string generated during mating. The alternative approach is to use local search sparingly. To limit the use of local search, Genitor+ was allowed to produce $\mathcal{N}/10$ new offspring ($\mathcal{N}$ = population size) using only recombination and mutation. At this point local search is applied by using the linear selection bias to select strings for local search. The number of strings selected for improvement via local search is equal to 1% of the population size; a minimum of 1 string was always processed. The cycle is then repeated, running Genitor+ without local search until $\mathcal{N}/10$ new offspring are generated; then local search is again applied to 1% of the population.

This limited use of local search was far more effective than applying local search to every string. When local search was applied to every string, the population very quickly lost diversity and the best and worst strings in the population quickly became similar, thus rendering the genetic operators ineffective and triggering the automatic stopping criterion.

The comparison in Table 9 suggests there is little difference between the hybrid genetic algorithm compared to local search using incremental evaluation. Aside from concerns about the execution speed of the implementations of the different algorithms, there is another factor which very much influences the results shown in Table 9: What criterion should be used to terminate genetic search when evaluation of the global optimum is unknown? This issue typically does not arise for artificially constructed test functions and this issue has not received a great deal of attention in the genetic algorithm literature.

5.4.1. Stopping Criterion. Considerable computation is expended by the genetic algorithm between the time when it first finds the best solution upon which it is going to converge, and when the automatic stopping criterion is satisfied. Given the type of comparison used in Table 9, the choice of stopping criterion is critical; yet we used the default stopping criterion which had been implemented in the Genitor Software Package. The criterion was designed more to determine when to give up on search rather than to terminate search

Table 9. Comparison of steepest ascent local search using partial evaluation (LS Inc) and the Hybrid genetic algorithm. % Diff refers to the amount of time required by the Hybrid to find a solution with 95% reliability compared to local search.

		Success Prob.: P_s		Trials: t_{95}		Estimated Run Time		
Model	Params	LS Inc.	Hybrid	LS Inc.	Hybrid	LS Inc.	Hybrid	% Diff.
Pole	12C	0.42	0.94	6	2	0.2	0.2	83%
Pole	24M	0.32	0.98	8	1	0.8	0.3	38%
Rectangle	28C	0.44	1.00	6	1	0.6	0.5	83%
Pole	42C	0.11	0.52	26	5	2.6	2.0	77%
Pole	42M	0.23	0.80	12	2	1.2	1.2	100%
Rectangle	52M	0.47	0.98	5	1	1.0	1.4	140%
Rectangle	68C	0.24	1.00	11	1	1.1	1.1	100%
Pole	72C	0.08	0.42	35	6	3.5	4.8	137%
Deer	81C	0.21	0.66	13	3	3.9	7.2	185%
Pole	81M	0.08	0.62	37	4	3.7	6.4	173%
Pole	96M	0.07	0.40	45	6	9.0	9.6	107%
Deer	99M	0.15	0.84	19	2	7.6	6.0	79%
Pole	102C	0.03	0.40	96	6	9.6	8.4	88%
Rectangle	108C	0.06	0.64	47	3	9.4	6.0	64%
Rectangle	108M	0.23	0.94	12	2	3.6	8.0	222%
Rectangle	124M	0.19	0.92	14	2	5.6	7.6	136%
Rectangle	148C	0.07	0.60	42	4	12.6	14.0	111%
Tree	156C	0.07	0.58	42	4	33.6	36.8	110%
Rectangle	168M	0.14	0.76	20	3	10.0	18.3	183%
Deer	171C	0.10	0.60	29	4	17.4	29.2	168%
Deer	180M	0.10	0.90	31	2	21.7	16.8	77%
Tree	216M	0.06	0.70	51	3	66.3	47.7	72%
Deer	261C	0.07	0.52	43	5	43.0	80.5	187%
Deer	261M	0.05	0.82	59	2	59.0	35.4	60%
Tree	276C	0.06	0.64	51	3	76.5	78.3	102%
Deer	342M	0.03	0.64	106	3	159.0	105.6	66%
Deer	351C	0.05	0.54	64	4	96.0	145.2	151%
Tree	396C	0.06	0.80	52	2	114.4	125.2	109%
Tree	432M	0.02	0.64	135	3	310.5	235.5	76%
Tree	516C	0.03	0.74	99	3	297.0	390.6	132%
Tree	552M	0.02	0.74	149	3	461.9	482.4	104%
Tree	780M	0.01	0.46	299	5	1,554.8	1,735.0	112%

as quickly and as efficiently as possible. When the difference in fitness between the best and worst strings drops below $\mathfrak{N}/50$, then the best error thus far is recorded. If after $\mathfrak{N}$ additional iterations no better solution is found then the algorithm terminates.

For the steepest ascent algorithm, each "trial" terminates when a local optimum is reached; there is no unnecessary thrashing of the algorithm after the locally optimal solution is found. With the genetic algorithms a "trial" terminates only after some stopping criterion is reached. Our criterion compares the best and worst solutions in the populations; when their evaluations become similar, search is terminated.

The notion that the hybrid genetic algorithm requires some form of heuristic stopping criterion while the steepest ascent algorithm does not is somewhat misleading. While steepest

ascent automatically terminates when a local optimum is reached, it must be run many times in order to have any reasonable chance of finding the global solution. What then is the stopping criterion for deciding when enough trials of steepest ascent have been run? Here we defined that criterion in a post-hoc and problem specific fashion using equation 2. Given unseen problems, both algorithms face a similar problem. If user intervention is utilized to determine if a solution is adequate, then the hybrid genetic algorithm is a better solution, since it finds the correct solution with a much higher probability on each run, while the steepest ascent algorithm relies on running many trials in order to find a correct solution.

One question we wished to explore in more detail was whether this criterion for the genetic algorithm was overly conservative: How many evaluations were required for the Hybrid algorithms to find a solution (when the solution was found) compared to the total number of evaluations executed before satisfying the stopping criterion? The data presented in Table 10 indicates that the Hybrid Genetic algorithm is finding the solution to the problems long before the stopping criterion is satisfied. The data in Table 10 is broken down into the number of evaluations used by the local search algorithm (which can exploit incremental evaluation) as well as the full-evaluation calls used by the genetic algorithm after mating. In every single instance in each individual category, the number of evaluations required to find the best solution is less than half the number of evaluations required to satisfy the stopping criterion. In some cases the number of evaluations involved differ by an order of magnitude.

Cutting the number of evaluations in half would be easy if the stopping criteria were selected in a post hoc fashion. More conservatively, if we assume a generally effective stopping criterion is selected that reduces the number of evaluations by even as little as 20%, then the hybrid genetic algorithm represents a solution to the geometric matching problem that improves on the local search algorithm using incremental evaluation. This ignores additional time savings which could be obtained by recoding the Genitor software to maximize execution speed for this application instead of flexibility.

6. Conclusion

This article has compared three genetic algorithms on a set of test problems demonstrated to be resistant to simple hill climbing. Two of these algorithms are also applied to the geometric matching application. The performance of the different genetic algorithm is varied, with CHC performing particularly well on the set of test problems. These test problems were especially formulated to exhibit complex nonlinear interactions between variables. These interactions reflect properties of important applications such as the geometric matching and seismic data interpretation. Unlike the simpler and more common test functions, they demonstrate the advantages of genetic search relative to simpler local search techniques.

On the geometric matchine problems, solutions are generally found near the origin of the hypercube corresponding to the search space. This bias appears to cause problems for some of the genetic algorithms, especially CHC. In addition, the geometric matching problem allowed for a form of fast incremental evaluation which could be exploited by local search, but which could not be directly exploited by genetic algorithms. A hybrid genetic algorithm that includes local search, however, is able to exploit incremental evaluation and

Table 10. Comparison of number of evaluations required to find solutions in contrast to time required to reach the stopping criterion. This is further broken down into the incremental evaluations (Inc-Evals) used by the local search component of the Hybrid algorithm and full evaluations (Full-Evals) of strings generated by recombination and mutation.

			Best Match Found		Convergence		
Model	Params	P_s	Inc-Evals	Full-Evals	Inc-Evals	Full-Evals	Secs
Pole	12C	94	161.4	22.1	727.6	129.2	0.2
Pole	24M	96	452.0	16.3	2540.8	138.0	0.5
Rectangle	28C	100	438.8	12.2	3693.9	164.7	0.7
Pole	42C	54	1311.2	47.6	3264.2	141.8	0.0
Pole	42M	90	1284.0	30.2	4748.0	148.8	0.9
Rectangle	52M	100	1462.4	13.1	11597.6	178.9	1.9
Rectangle	68C	100	1920.9	22.1	10663.4	184.7	1.8
Pole	72C	42	2324.5	49.8	5949.4	150.8	1.2
Deer	81C	84	6175.3	40.6	23095.7	252.9	3.6
Pole	81M	72	4288.2	45.8	12057.3	162.4	2.4
Pole	96M	56	3999.0	38.6	12707.8	163.3	2.5
Deer	99M	92	9740.3	60.3	30829.1	280.7	4.7
Pole	102C	48	4161.0	58.3	10026.1	163.7	2.1
Rectangle	108C	80	4888.9	37.9	18074.7	197.9	3.4
Rectangle	108M	96	6251.5	28.7	28152.4	205.8	5.0
Rectangle	124M	94	6286.8	33.8	29511.1	233.6	5.7
Rectangle	148C	62	7684.1	74.2	19792.1	227.0	4.3
Tree	156C	68	37024.8	154.9	83272.4	505.6	13.4
Rectangle	168M	90	8978.0	48.0	41066.9	291.4	8.4
Deer	171C	78	24980.9	151.2	62196.5	520.5	10.6
Deer	180M	88	21017.0	116.8	73928.3	602.3	12.0
Tree	216M	74	52856.6	271.0	129904.2	897.4	21.1
Deer	261C	62	50067.9	285.2	113533.1	796.8	20.7
Deer	261M	84	37051.1	206.2	122304.8	925.7	22.7
Tree	276C	72	89913.4	370.7	202536.3	1082.0	33.8
Deer	342M	66	65857.8	274.4	184221.0	966.9	35.3
Deer	351C	64	85626.9	371.5	179823.7	941.0	35.2
Tree	396C	70	148512.5	504.8	332537.4	1392.8	62.9
Tree	432M	76	175521.8	693.4	387632.3	1864.1	82.9
Tree	516C	56	245209.8	539.8	571884.2	1590.3	113.7
Tree	552M	66	263373.7	613.2	643095.3	1937.2	130.7
Tree	780M	42	429647.0	701.4	1032966.4	2191.5	245.9

find a global solution with a high degree of reliability. As implemented, the hybrid genetic algorithm and local search using incremental evaluation yield similar results; however, the hybrid genetic algorithm is exploring only half as many possible solutions before finding the best. Any significant improvement in the performance of the hybrid algorithm would result in it being superior to steepest ascent local search for geometric matching. The data suggests that such improvements could be easily achieved by adjusting the stopping criterion used to terminate the hybrid algorithm.

Acknowledgments

This work was supported in part by NSF grants IRI-9312748 and IRI-9503366, by the Advanced Research Projects Agency (ARPA) under grant DAAH04-93-G-422 monitored by the U.S. Army Research Office, and by the Colorado Advanced Software Insitute (CASI). Our thanks to Soraya Rana and Drew Schwickerath for comments on an earlier draft of this article.

Notes

1. Technically, the $\mathfrak{N} - 1$ best strings are maintained in the population of size $\mathfrak{N}$, since the worst string in the population is replaced after every reproduction.
2. Selective Pressure was also reset to 2.0 for Genitor with the new adaptive mutation function.
3. In these geometric matching problems it is usually possible to determine the global optimum (or a good approximation thereof) through visual inspection. In general, the evaluation associated with a near optimal solution is not known until after a solution has been found.
4. Except the selection bias was reset to 1.25.

References

Bäck, T., Hoffmeister, F., and Schwefel, H.P. (1991). A survey of evolution strategies. In L. Booker and R. Belew (Eds.), *Proceedings of the Fourth International Conference on Gas* (pp. 2–9). San Mateo, CA: Morgan Kaufmann.

Bäck, T., and Schwefel, H.P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1, 1–23.

Beveridge, J. Ross. (1993). Local search algorithms for geometric object recognition: Optimal correspondence and pose. Ph.D. thesis, University of Massachusetts at Amherst.

Beveridge, J. Ross, and Riseman, E.M. (1995). Optimal geometric model matching under full 3D perspective. *CVGIP: Image Understanding*. (Short version in IEEE Second CAD-Based Vision Workshop).

Beveridge, J. Ross, Weiss, Rich, and Riseman, Edward M. (1989). Optimization of two-dimensional model matching. In *Proceedings: Image Understanding Workshop* (pp. 815–830). Los Altos, CA: DARPA, Morgan Kaufmann.

Beveridge, J. Ross, Weiss, Rich, and Riseman, Edward M. (1989). Optimization of two-dimensional model matching. In Hatem Nasr (Ed.), *Selected Papers on Automatic Object Recognition* (originally appeared in DARPA Image Understanding Workshop). SPIE Milestone Series. Bellingham, WA: SPIE.

Bolles, R.C., and Cain, R.A. (1982). Recognizing and locating partially visible objects: The local-feature-focus method. *International Journal of Robotics Research*, 1(3), 57–82.

Collins, R.T., and Beveridge, J. Ross. (1993). Matching perspective views of coplanar structures using projective unwarping and similarity matching. In *Proceedings: 1993 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 240–245). New York: IEEE.

Davis, Lawrence. (1991a). Bit-climbing, representational bias, and test suite design. In L. Booker and R. Belew (Eds.), *Proceedings of the Fourth International Conference on GAs* (pp. 18–23). San Mateo, CA: Morgan Kaufmann.

Davis, Lawrence. (1991b). *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold.

DeJong, Ken. (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.

Eshelman, Larry. (1991). The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G. Rawlins (Ed.), *FOGA-1* (pp. 265–283). San Mateo, CA: Morgan Kaufmann.

Fennema, Claude, Hanson, Allen, Riseman, Edward, Beveridge, J.R., and Kumar, R. (1990). Model-directed mobile robot navigation. *IEEE Trans. on System, Man and Cybernetics*, 20(6), 1352–1369.

Fogel, D.B. (1994). Evolutionary programming: An introduction and some current directions. *Statistics and Computing*, 4, 113–130.

Fogel, L.J., Owens, A.J., and Walsh, M.J. (1966). *Artificial Intelligence Through Simulated Evolution*. New York: Wiley.

Glover, F. (1994). Genetic algorithms and scatter search: Unsuspected potentials. *Statistics and Computing*, 4, 131–140.

Goldberg, David. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley.

Goldberg, David. (1990). A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. Technical Report No. 90003, Department of Engineering Mechanics, University of Alabama.

Grimson, W. Eric L. (1990). *Object Recognition by Computer: The Role of Geometric Constraints*. Cambridge, MA: MIT Press.

Holland, John. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.

Kernighan, B.W., and Lin, S. (1972). An efficeint heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49, 291–307.

Lowe, David G. (1985). *Perceptual Organization and Visual Recognition*. Boston: Kluwer.

Mühlenbein, H. (1991). Evolution in time and space: The parallel genetic algorithm. In G. Rawlins (Ed.), *FOGA-1* (pp. 316–337). San Mateo: Morgan Kaufmann.

Mühlenbein, H., and Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm. *Journal of Evolutionary Computation*, 1(1), 25–49.

Mathias, Keith E., and Whitley, L. Darrell. (1994). Transforming the search space with Gray coding. In J.D. Schaffer (Ed.), *IEEE International Conference on Evolutionary Computation* (pp. 513–518). IEEE Service Center.

Mathias, Keith E., Whitley, L. Darrell, Stork, Christof, and Kusuma, Tony. (1994). Staged hybrid genetic search for seismic data imaging. In J.D. Schaffer (Ed.), *IEEE International Conference on Evolutionary Computation* (pp. 356–361). IEEE Service Center.

Papadimitriou, Christos H., and Steiglitz, Kenneth. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Englewood Cliffs, NJ: Prentice-Hall.

Shaffer, J. David, Caruana, Richard A., Eshelmen, Larry J., and Das, Rajarshi. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In J.D. Schaffer (Ed.), *Proceedings of the Third International Conference on GAs* (pp. 51–60). San Mateo, CA: Morgan Kaufmann.

Starkweather, Timothy, Whitley, L. Darrell, and Mathias, Keith E. (1990). Optimization using distributed genetic algorithms. In H.P. Schwefel and R. Männer (Eds.), *Parallel Problem Solving from Nature* (pp. 176–185). Berlin: Springer/Verlag.

Syswerda, Gilbert. (1989). Uniform crossover in genetic algorithms. In J.D. Schaffer (Ed.), *Proceedings of the Third International Conference on GAs*. San Mateo, CA: Morgan Kaufmann.

Whitley, L. Darrell. (1989). The GENITOR algorithm and slective pressure: Why rank based allocation of reproductive trials is best. In J.D. Schaffer (Ed.), *Proceedings of the Third International Conference on Gas* (pp. 116–121). Morgan Kaufmann.

Whitley, L. Darrell. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4, 65–85.

Whitley, Darrell, and Kauth, Joan. (1988). GENITOR: A different genetic algorithm. In *Proceedings of the 1988 Rocky Mountain Conference on Artificial Intelligence*. City: Publisher.

Whitley, Darrell, Mathia, Keith, Rana, Soraya, and Dzubera, John. (1995). Building better test functions. In L. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Gas*. City: Morgan Kaufmann.

Whitley, Darrell, Mathias, Keith, Rana, Soraya, and Dzubera, John. (1995). Evaluating evolutionary algorithms. Manuscript.