Searching in the Presence of Noise

Soraya Rana, L. Darrell Whitley, Ronald Cogswell

Computer Science Department, Colorado State University, Fort Collins, CO 80523
{rana, whitley, cogswell}@cs.colostate.edu

Abstract. In this paper, we examine the effects of noise on both local search and genetic search. Understanding the potential effects of noise on a search space may explain why some search techniques fail and why others succeed in the presence of noise. We discuss two effects that are the result of adding noise to a search space: the annealing of peaks in the search space and the introduction of false local optima.

1 Introduction

The two major components to any search problem are the algorithm we choose to use to traverse the search space and the function used to provide feedback to the algorithm (i.e. the objective function). When the objective function provides noisy feedback, the resulting behaviors and long term performance of search algorithms can be surprising.

By design, genetic algorithms should be effective search techniques in noisy environments. The genetic algorithm population preserves both highly fit points in the space and less fit points in a space which allows for a certain amount of variance for the fitness values of its members. Selection allocates trials to subpartitions of the search space based on the relative fitness values of sample strings the population: providing that the amount of noise is not overwhelming, this operation is still effective in the presence of noise. Thus, one advantage of using an evolutionary algorithm in a noisy environment is that it is not quick to discard valuable information.

Local search techniques, while often producing very good results in noiseless environments, are brittle in comparison to genetic algorithms in noisy environments. Local neighborhood search methods are often unable to accurately identify the best improving move or they may not be able to confidently identify a local optimum without strong a priori information about the amount of noise present relative to the expected change in the objective function for any given move. On the other hand, noise can sometimes have a soft annealing effect on the search space: even if one is currently in a local optimum, noisy evaluation can still potentially indicate that an improving move is possible. This can potentially make it possible for local search to actually perform well in the presence of noise. Levitan refers to this effect as “melting” [8].

It is difficult to predict how an arbitrary algorithm will behave in the presence of noise. This paper empirically examines the performance of various search algorithms in both noisy and noiseless conditions. We also include some old and new test functions, since the type of test function that is used also impacts the performance. We tested both the CHC as well as an elitist simple genetic algorithm along with Adaptive Simulated Annealing [7] and two local search techniques: random bit climbing and Line Search.

2 Background

Holland's *schema theorem* is based on the idea that selection acts not just on individual strings, but also on hyperplanes representing subpartitions of the search space. The *schema theorem*, in fact, is just a lower bound on the change in the sampling rate for a single hyperplane from generation t to generation $t+1$. Let P(H,t) be the representation of hyperplane H in the population at generation t; let P(H,t+s) be the representation of hyperplane H in the population at a point intermediate between generation t and $t+1$ *after selection* but *before recombination*. Under fitness proportionate selection

$$P(H,t+s) = P(H,t)\frac{f(H,t)}{\bar{f}_t}$$

where f(H,t) is the average fitness of strings in the population that sample H at time t and $\bar{f}_t$ is the average population fitness. Now assume that random Gaussian noise is included as part of our fitness function. That is, for all strings i that are members of the current population,

$$f(H,t) = \sum_{i \epsilon H}(e(i) + G_i)$$

where $e(i)$ is the true evaluation of string i and G_i is noise added to string i. The noise is randomly drawn form a Gaussian distribution with $\mu = 0$. If we have a reasonably large sample for hyperplane H, then

$$f(H,t) = \sum_{i \epsilon H} e(i) + \sum_{i \epsilon H} G_i = \sum_{i \epsilon H} e(i).$$

In other words, noise has no impact on the expected average fitness of a hyperplane subpartition if the number of samples from that partition is large. This suggests that selection on large hyperplane subpartitions is not seriously affected by noise if the population is also relatively large. This argument has been used to suggest that genetic algorithms are robust in the presence of noise, but there have been few empirical studies confirming this argument.

It should also be noted that this argument depends only on selection and that similar arguments might be applied to other evolutionary algorithms using other forms of selection. In the current study we look at CHC, which combines features

$F101(x,y) = -xsin(\sqrt{\|x - \frac{(y+47)}{2}\|}) - (y+47)sin(\sqrt{\|y+47+\frac{x}{2}\|})$	$x, y \in [-512, 511]$
$F102(x,y) = xsin(\sqrt{\|y+1-x\|})cos(\sqrt{\|x+y+1\|}) + (y+1)cos(\sqrt{\|y+1-x\|})sin(\sqrt{\|x+y+1\|})$	$x, y \in [-512, 511]$

Table 1: Suite of test functions used in the experiments.

of genetic algorithms and $(\mu + \lambda)$ evolution strategies. CHC also used small population sizes (i.e., 50) and truncation selection. This doesn't appear to fit the robust "hyperplane sampling" model very well, but nevertheless our results indicate that CHC is quite effective in the presence of noise.

Previous studies on the effects of noise on genetic algorithms performance have dealt mainly in analyzing specific components of specific genetic algorithms that may improve convergence properties, namely selection methods [10, 6]. Hammel and Back [6] also discuss the relationship between population size and sample size in terms of the potential benefit of one over the other. Greffenstette [4] provides both theoretical and empirical evidence that indicates that better performance can be achieved by choosing a large population over doing more work per evaluation to reduce the impact of noise on each string evaluation when using a traditional genetic algorithm. Mathias and Whitley [9] have done similar initial studies, but only looked at the random bit climbing, RBC, and also used a restricted set of test functions.

We test a suite of five test functions with and without the addition of noise. These include Rastrigin's function (F6), Schwefel's function (F7) and Griewangk's function (F8). We include these relatively simple functions in order to observe how noise impacts functions with very limited nonlinearity and whose optima can be found rather quickly using simple methods.

The remaining two functions were developed by Whitley et al. [11] (See Table 1). For all functions, every parameter is represented using a ten bit substring and Gray coded. All functions were scaled to 20 parameters. $F6$, $F7$ and $F8$ are scaled using a summation from 1 to 20 of each parameter evaluated using the basic subfunction that defines each function. For the functions in Table 1 Whitley et al. [11] describe methods for expanding functions of two variables into functions of N variables in a nonlinear manner. The method used here is the *wrap* method. A general formula illustrating the wrap method using a function of two variables F is as follows:

$$EF(x_1, x_2, x_3, .., x_n) = F(x_1, x_n) + \sum_{i=1}^{n-1} F(x_i, x_{i+1})$$

where EF is a function obtained by the *expansion* of F.

We examine the performance of five search algorithms: two evolutionary algorithms, two local search techniques and adaptive simulated annealing. We do not tune any of the algorithms to deal with the noise. The CHC adaptive search algorithm [3] (CHC) is run using a population of 50 with the HUX recombination operator while the elitist simple genetic algorithm uses tournament selection [5] (ESGAT) and a population size 200. For ESGAT, recombination is accomplished using a 2-point reduced-surrogate crossover operator [1] applied with probability 0.9 and mutation is applied to each individual bit with a probability of 1/L, where L is the length of the string. (We stress that testing the effect of increasing population size is one goal of our future work.)

The two local search techniques used here are Davis's Random Bit Climber and Line Search [11] (LINE). Davis' random bit climber (RBC) starts by changing 1 bit at a time beginning at a random position. Each improving move is accepted. The sequence in which the bits are tested is also randomly determined. After testing every bit in the string, a new random sequence for checking the bits is chosen and RBC again tests every bit for an improvement. If a local optima has been reached (i.e., there are no improving moves) RBC is restarted from a new random point in the space by generating a new random string.

Line Search works with any discretization of the space. Starting with a randomly chosen solution of n parameters, Line Search enumerates each parameter to find the best possible setting for that parameter. The order in which the parameters are enumerated is also randomly determined. If there are no nonlinear interactions between variables, then Line Search is an exact method guaranteed to find the global optimum *if the objective function is also exact*. Assuming each of the n parameters has k possible values, Line Search will have reached the global optimum after nk evaluations if there are no nonlinear interactions. $F6$ and $F7$ are exactly solved by Line Search since there are no nonlinear interactions between variables. On functions with nonlinear interactions, Line Search acts like a local search method searching a neighborhood of nk points.

The last algorithm that we use in our experiments is adaptive simulated annealing (ASA). The implementation used is ASA version 10.16 which was downloaded from the CalTech archive of Lester Ingber [7].ASA is a simulated annealing algorithm first developed as Very Fast Simulated Reannealing (VFSR) in 1987. It differs from traditional simulated annealing algorithms by allowing for specialized annealing schedules for different parameters to address the issue of parameter sensitivity. For our application, tuning of the algorithm was minimal and limited to increasing performance on large parameter spaces. Reannealing is not utilized due to the high number of evaluation function calls required. Termination criteria was modified to prevent repeating evaluation values from ending the search.

3 Performance With and Without Noise

The first set of experiments were run using all four algorithms and all five test functions without the addition of any noise. The results are given in Table 2.

FUNC	ALG	Mean Solution	σ	Mean Trials	σ Trials	Mean Rstrts	σ Rstrts	Nbr Slv
EF101	CHC	-939.880	0.000	93388	50333			30
	ESGAT	-809.875	73.102					0
	RBC	-668.569	23.400			243.69	6.39	0
	LINE	-752.234	30.299					0
	ASA	-740.462	43.479					0
EF102	CHC	-476.904	7.451					0
	ESGAT	-415.845	12.906					0
	RBC	-407.564	14.087			192.76	5.45	0
	LINE	-481.829	5.301					0
	ASA	-445.798	16.328					0
F6	CHC	0.000	0.000	158839	72048			30
	ESGAT	12.910	3.285					0
	RBC	23.198	1.879			422.89	2.53	0
	LINE	0.000	0.000	20500	0			30
	ASA	4.646	1.154					0
F7	CHC	-8379.655	0.000	30029	6397			30
	ESGAT	-8294.249	57.703					0
	RBC	-6994.345	188.166			468.20	2.36	0
	LINE	-8379.655	0.000	20500	0			30
	ASA	-8379.171	0.145					0
F8	CHC	0.000	0.000	50509	39451			30
	ESGAT	1.185	0.056					0
	RBC	0.000	0.000	82215	98985	56.90	69.45	30
	LINE	0.037	0.024	178789	113710			7
	ASA	0.869	0.113					0

Table 2: All four algorithms run on the suite of five test functions without the addition of noise.

ASA produces relatively good results; it is not the best performer but is also not the worst. As predicted, Line Search solves F6 and F7 on its first iteration. RBC produces surprisingly good results on $F8$ while Line Search performed significantly worse than RBC in terms of the number of times the global optimum is found. ESGAT performs better than RBC on four of the five functions tested and never outperforms Line Search. In fact, CHC usually outperforms all algorithms with the exception of Line Search. These experiments illustrate that Line Search and RBC can be competitive with the genetic algorithms without the presence of noise.

The next set of experiments involve the same functions with the addition of Gaussian noise (See Table 3). The Gaussian noise is drawn from a distribution with $\mu = 0$ and $\sigma = 6$. Assuming the global optimum is g and its evaluation is $e(g)$, search was terminated if a noisy evaluation is obtained that is less than or equal to $e(g) - 3$. The first two columns of Table 3 are the noisy mean and

FUNC	ALG	Mean Solution	σ	Mean True Solution	σ	Mean Rstrts	σ Rstrts	Nbr Slv
EF101 +noise	CHC	-943.063	0.155	-939.294	0.220			30
	ESGAT	-840.927	82.007	-839.473	81.852			0
	RBC	-665.952	23.046	-663.263	22.935	370.33	6.65	0
	LINE	-737.868	50.032	-735.675	49.933			0
	ASA	-759.765	57.286	-756.886	57.346			0
EF102 +noise	CHC	-479.425	7.483	-478.582	7.466			0
	ESGAT	-411.256	14.053	-410.248	14.078			0
	RBC	-396.821	8.611	-394.252	8.727	399.46	7.57	0
	LINE	-482.405	6.861	-481.088	6.621			0
	ASA	-446.467	17.003	-444.356	17.088			0
F6 +noise	CHC	-2.270	1.779	0.660	0.186			23
	ESGAT	15.433	3.010	17.219	2.932			0
	RBC	27.161	3.550	29.772	3.660	434.36	5.98	0
	LINE	2.081	0.5265	4.048	0.754			0
	ASA	8.052	1.138	10.323	1.428			0
F7 +noise	CHC	-8382.810	0.157	-8379.099	0.220			30
	ESGAT	-8280.761	68.429	-8279.982	68.416			0
	RBC	-6977.993	168.510	-6975.396	168.516	393.16	5.06	0
	LINE	-8378.417	0.659	-8376.452	0.786			0
	ASA	-8379.029	0.826	-8376.451	0.991			0
F8 +noise	CHC	-0.380	1.296	2.231	0.544			0
	ESGAT	1.031	0.724	3.901	0.922			0
	RBC	1.561	0.649	4.523	0.711	427.79	6.52	0
	LINE	0.076	0.413	2.889	0.488			0
	ASA	-0.983	0.456	2.501	0.416			0

Table 3: All four algorithms evaluated on the suite of five test functions with noise added.

standard deviations for each algorithm based on 30 runs. The third and fourth columns are the true mean and standard deviations for the same experiments.

Typically, the performance of any algorithm on any function degrades when noise is added. However, the degree to which the performance changes varies not only from algorithm to algorithm but also from function to function. The performance of several of the algorithms is not significantly affected by the addition of noise on $EF101$, $EF102$ and $F7$ (verified using a two-tailed Student's T-test with $p = 0.05$). ESGAT's performance was particularly stable across $EF101$, $EF102$ and $F7$ and shows no significant change in performance when noise is added. RBC shows no significant change in performance on $EF101$ and $F7$ and Line Search and ASA show no significant change in performance on $EF101$ and $EF102$. The only function for which CHC does not show a significant change in performance is $F7$. When there are significant changes in performance (particularly for Line Search and CHC), it is usually because the functions were solved 30 out of 30 times when noise is not present. However, in those cases, the mean solutions found for the noisy function are often very close to mean solutions found for the corresponding non-noisy function.

The solutions for the noisy versions of $F6$ and $F8$ are categorically worse than the non-noisy results for all algorithms. In all cases, the mean of the best solutions found for each algorithms becomes significantly worse when noise is added.

3.1 Melting Optima Versus False Optima

There are at least two significant impacts of adding noise to an objective function: the soft annealing effect and the creation of false optima. Levitan [8], refers to the smoothing and soft annealing effect as "melting" of peaks. Noise can allow a local search algorithm to escape a local optimum by indicating an improving move is present when none in fact exists. Our results suggest that this effect can cause some algorithms to perform better on some functions with added noise– especially during the *initial* phases of search. However, the utility of the melting effect depends on the underlying function.

One primary difference between our experiments and those performed by Levitan are in the algorithms we tested. Levitan uses a single algorithm that is effectively the same as RBC without restarts. We attempted to use RBC without restarts but the resulting solutions were not competitive with those found using RBC with restarts simply because there is an advantage to sampling several points in the space as opposed to examining only one. An argument for removing the restarts is that their removal allows closer study of effects of noise on a given search space. However, it is also important to consider whether or not the algorithm will ultimately converge to a reasonably good solution even in the presence of noise. For this reason, we chose to report our results for RBC with restarts.

Like Levitan, we found evidence that supports the annealing effect. For $F7$, the number of restarts for RBC dropped significantly when noise was added to the function. This is evidence that the search space became "smoother" when noise was added; however, this was the only function in which this effect dominated and in the end RBC did not have an significant improvement in the quality of solutions it found. On functions $EF101$, $EF102$, $F6$ and $F8$, the number of restarts significantly increased when noise was added. This implies that the second effect, introduction of false optima, dominated in those functions.

We conjecture that if the objective function is relatively simple before the noise is added, the noise has little or no positive impact on search and only serves to make search more difficult. This effect is particularly evident in the performance on $F8$. Whitley et al. [11] show empirically and explain analytically that at higher dimensions $F8$ has smaller and more shallow local optima. Thus, scaling up $F8$ has a smoothing effect. At 20 variables the addition of noise to $F8$ caused a dramatic increase in the number of restarts for RBC. Given that the performance of RBC on $F8$ without noise was exceptionally good, its performance degraded dramatically when the noise was added.

4 Resampling Data During Search

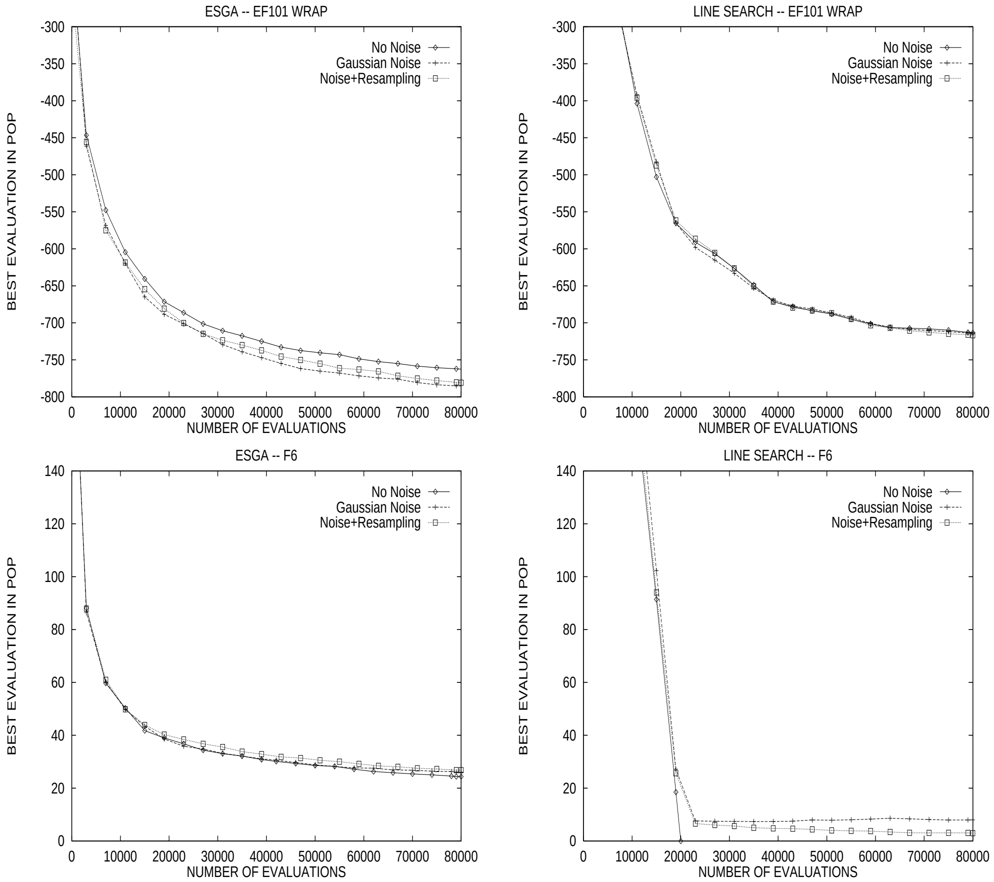

Figure 1: ESGAT (left) and Line Search (right) on EF101 (top), EF6 (bottom) without noise, with Gaussian noise added and with Gaussian noise with scheduled resampling.

Resampling data points is a common approach used to reduce the effects of noise. Other researchers [6, 4] point out that increasing the population size of a genetic algorithm can have similar effects as resampling data points. However, since two of our algorithms (ASA was not used in this set of experiments) are not population based, increasing the number of points sampled during execution by increasing population size was not used here. Instead, we wished to explore the benefits of sampling the evaluation function multiple times for a simple string in order to obtain more accurate evaluations as the execution progressed. A schedule is used so that we initially start out with 1 evaluation per string, then exponentially increase the number of evaluations per string every 20 thousand points sampled (20 thousand evaluations roughly corresponds to one iteration of Line Search.) This was done for both genetic algorithms and the local search techniques. The schedule is as follows:

```
  0-20K points: 1 evaluation      40K-60K points: 4 evaluations
20K-40K points: 2 evaluations     60K-80K points: 8 evaluations
```

Of the five functions and four algorithms tested using scheduled sampling, the only case where the addition of noise produced any improvement of performance is for ESGAT on EF101 as shown in the upper left graph of Figure 1. For the scheduled resampling to reduce noise, as the accuracy of the evaluation of each point is improved, the performance tends towards the evaluation of the true function performance which in this case is a degradation in performance. In the lower left graph of Figure 1, ESGAT's behavior on $F6$ is fairly stable with and without noise–which is typical of ESGAT's behavior on all of the other test functions. Line Search exhibits changes in performance for F6 in Figure 1 that are more consistent with intuition. Line Search has degraded performance in the presence of noise and as the resampling is increased, the performance improves to become competitive with the performance without noise. However, sometimes the noise had little effect, as is shown in Figure 1 for EF101.

5 Conclusions

Our empirical study of several algorithms on several test functions reveals interesting characteristics of the algorithms and the test functions. First, adding noise can potentially have a soft annealing effect in some cases. At the same time, adding noise can have the negative effect of adding false optima to the search space. The degree to which either of these effects occurs is directly related to the underlying objective function.

When run on functions without noise, Line Search and CHC produce very good results on all functions. As expected, both RBC and Line Search prove to be more subject to noise than the other algorithms. In fact, both genetic algorithms had very stable performance with or without noise. In general, the scheduled resampling did not improve the performance of the genetic algorithms. It is in the local search process where resampling of noisy data points has an opportunity to improve performance.

This work has been a preliminary study of the effects of noise on a variety of search techniques. Two future directions for this work is in the area of closer analysis of the behavior of local search methods in noisy environments and in additional experimentation with the population sizes of evolutionary algorithms. Levitan [8] introduced several mechanisms for closely analyzing the traversal of his *adaptive walk* (or our RBC with no restarts). Similar mechanisms can be integrated into our algorithms to more accurately determine whether or not the algorithms are benefiting or suffering from the addition of noise.

References

[1] Lashon Booker. Improving Search in Genetic Algorithms. In Lawrence Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 5, pages 61–73. Morgan Kaufmann, 1987.

[2] Lawrence Davis. Bit-Climbing, Representational Bias, and Test Suite Design. In L. Booker and R. Belew, editors, *Proc. of the 4th Int'l. Conf. on GAs*, pages 18–23. Morgan Kauffman, 1991.

[3] Larry Eshelman. The CHC Adaptive Search Algorithm. How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In G. Rawlins, editor, *FOGA -1*, pages 265–283. Morgan Kaufmann, 1991.

[4] J. Michael Fitzpatrick and John Grefenstette. Genetic Algorithm in Noisy Environments. *Machine Learning*, 3:101–120, 1988.

[5] David Goldberg. A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-oriented Simulated Annealing. Technical Report Nb. 90003, Department of Engineering Mechanics, University of Alabama, 1990.

[6] Ulrich Hammel and Thomas Bäck. Evolution Strategies on Noisy Functions How to Improve Convergence Properties. In Y. Davidor, H.P. Schwefel, and R. Manner, editors, *Parallel Problem Solving from Nature, 3*, pages 159–168. Springer/Verlag, 1994.

[7] L. Ingber. Adaptive Simulated Annealing: Lessons Learned. *Control and Cybernetics*, 1996.

[8] Bennett Levitan and Stuart Kauffman. Adaptive walks with noisy fitness measurements. *Molecular Diversity*, 1:53–68, 1995.

[9] Keith E. Mathias and L. Darrell Whitley. Noisy Function Evaluation and the Delta Coding Algorithm. In *Proceedings of the Conference on Neural and Stochastic Methods in Image and Signal Processing III*, 1994.

[10] Brad Miller and David Goldberg. Genetic Algorithms, Selection Schemes, and the Varying Effects of Noise. Technical Report IlliGAL Report No. 95009, Department of General Engineering, University of Illinois at Urbana-Champaign, 1995.

[11] Darrell Whitley, Keith Mathias, Soraya Rana, and John Dzubera. Evaluating Evolutionary Algorithms. *Artificial Intelligence Journal*, 85, August 1996.