Mathematical and Computer Modelling, 16(11) 1992, 87-100.

GENETIC ALGORITHMS AND VERY FAST SIMULATED REANNEALING: A COMPARISON

LESTER INGBER
Lester Ingber Research, P.O. Box 857, McLean, VA 22101, U.S.A.
ingber@alumni.caltech.edu

BRUCE ROSEN
Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716, U.S.A.
brosen@cis.udel.edu

Abstract—We compare Genetic Algorithms (GA) with a functional search method, Very Fast Simulated Reannealing (VFSR), that not only is efficient in its search strategy, but also is statistically guaranteed to find the function optima. GA previously has been demonstrated to be competitive with other standard Boltzmann-type simulated annealing techniques. Presenting a suite of six standard test functions to GA and VFSR codes from previous studies, without any additional fine tuning, strongly suggests that VFSR can be expected to be orders of magnitude more efficient than GA.

1. INTRODUCTION

This paper compares Genetic Algorithms (GA) with a functional search method, Very Fast Simulated Reannealing (VFSR), that not only is efficient in its search strategy, but also is statistically guaranteed to find the function optima. We first review Genetic Algorithms as evolutionary optimization methods, and illustrate their operation. We next introduce the VFSR algorithm, and show how it can use a faster annealing schedule than either Boltzmann Annealing, or Fast Annealing. We then compare GA with VFSR on a robust suite of 6 test problems involving relatively easy to difficult function optimization, and comment on their relative performances. Our preliminary results show that VFSR is at least an order of magnitude superior to Genetic Algorithms in convergence speed and is more likely to find the global optima during a time limited search. We conclude with a discussion on the relative merits of parallelizing the algorithms.

2. GENETIC ALGORITHMS

Genetic Algorithms are population based search strategies that maintain the locations and values of a set of points in a function space. The standard of comparison for which new points are generated or old points are discarded is a function of the existing population. In these GA, combinations of individual points in the population are used to derive new points.

2.1. *Genetic Reproduction*

Genetic algorithms are optimization algorithms based on Darwinian models of natural selection and evolution. The basic idea behind GA is that an initial population of candidate states of size n are chosen at random and each state is evaluated according to the optimization function. Iteratively, a small set of elite individuals of the population are chosen by a selection procedure according to their fitness value f, derived from an individual's relative functional performance, a measure of how well they satisfy the optimization function. The selection procedure probabilistically selects an individual i to remain in the population and reproduce with probability $p_i = f_i / \sum_{j=1}^{n} f_j$. Those states not selected are culled from the population. Because the average fitness of the population is defined as $\bar{f} = \sum_{j=1}^{n} f_j / n$, if there are $m_i(t)$ copies of an individual i at time t, the new population will have $m_i(t+1) = m_i(t) f_i / \bar{f}$ copies of f_i. The effect of this reproduction scheme is that above average performing individuals reproduce, replacing poorly performing individuals. The test, select, and reproduce operators are applied each iteration (generation), and replace the standard generate and test paradigm.

2.2. *Genetic Recombination*

To interpolate between states with high fitness values, the crossover operators genetically recombines individuals states. New individuals are created by encoding their values as bit strings of a given length, l, typically by using gray or binary coding. Two parent states from the new population are chosen randomly to create two new child states. A crossover position between 1 and l is chosen at random, and the two parents exchange portions of their binary representations. The first child contains bits 1 through i from its first parent and bits $i+1$ to l from its second parent, while the second child contains bits 1 through i from the second parent and bits $i+1$ to l from the first parent. (These bit strings are analogous to chromosomes in biological systems, with the crossover function creating new states by sexual reproduction.) This process continues until enough new children are produced to replace the parent population, although some parents may kept in the new population.

For diversity, mutations that alter individual states occasionally occur (with small probability) each generation. The mutation operator randomly flips a bit in an individual's bit string representation. The resulting effect allows the population to sample states away from its mean, preventing the population from converging and stagnating at any minima, but the mutation operator does not cause the population to sample the state space ergodically. The reproduction, crossover and mutation cycle continues until an acceptable solution is found, until the average population fitness converges to a stable fixed points, or until a fixed number of generations have evolved.

Consider the optimization task of finding an integer x that minimizes the function $f(x) = x^2(x-1)^2 + 0.1x^2$ subject to the constraint $0 < x < 16$. The function has a global minimum at $x = 0$, and a slightly higher local minimum at $x = 1$. For the purposes of this example, the population states represented are generated as bit strings of length 4. The strings are initialized with randomly generated 0 or 1 values, and the fitness function to maximize is $-f(x)$. Using a simple (nonnegative) binary decoding algorithm, these strings represent integer values from 0 to 15. Each individual is ranked according to the fitness of its decoded value. The top performing individuals reproduce replacing those with poor fitness values. The new population are genetically recombined, although mutations occasionally occur. Consider two individuals bit strings, A = 0111, and B = 1001, that are chosen to breed. A crossover point between 0 and 3 is randomly chosen. If the crossover point is 1, the two newly created children C and D have values C = 0001, and D=1111. While strings A and B represent positive integer values of 7 and 9 respectively, their children bit strings (C and D) represent integer values of 1 and 15 respectively. Individuals, such as C, having relatively high fitness values (as determined by f) will increasingly dominate the population, while less significant individuals, such as D, die off. Each successive generation, this eugenic method increases the mean population fitness, until the majority of the population settles in the local minimum at x=1 (precision is lost by nature of the encoding). The mutation operator, occasionally flipping random bits in the population, causes individuals to sample points that span the space with precision defined by the encoding method. New states are typically suboptimal relative to the population mean, but can infrequently increase the mean fitness of the population. Should the rightmost bit of C be mutated, the global minimum will be found, and successive populations will converge to $f(0000) = 0$.

The original concepts of Genetic Algorithms were developed by Holland [1], and have been shown to provide near-optimal heuristics for information gathering in complex search spaces. They frequently outperform other more direct methods such as gradient descent on difficult problems, i.e., those involving highly nonlinear, high dimensional, discrete, multimodal or noisy functions. (However, gradient descent methods are more efficient for finding solutions when searching convex function spaces with tight constraints e.g., continuous, low dimensional, unimodal spaces.) Many empirical simulations have demonstrated the efficiency and robustness of GA on different optimization tasks [2-4]; however, GA are not guaranteed to find the global functional optima because: (1) the precision limits in the encoding process can substantially reduce the solution accuracy, and (2) the search process does not ergodically cover and search the state space. The first deficiency is the limited precision that is dependent on the length of the population bit strings. A bit string of length four, when decoded, allows only 15 distinct values; gray coding is even more limited. Methods such as dynamic parameter encoding have addressed the former issue and have achieved good results on a variety of function optimization problems [5]. However, in the GA domain, the second deficiency has been largely unresolved.

3. VERY FAST SIMULATED REANNEALING

An algorithm of Very Fast Simulated Reannealing (VFSR) has been developed to fit empirical data to a theoretical cost-function over a D-dimensional parameter-space [6]. This methodology has been applied to several systems, ranging from combat analysis [7,8], to finance [9,10], to neuroscience [11,12]. This Section gives a self-contained description of VFSR. The general outline of presentation of simulated-annealing heuristics here closely follows that of Szu and Hartley [13].

3.1. *Boltzmann Annealing (BA)*

Boltzmann annealing was essentially introduced as a Monte Carlo importance-sampling technique for doing large-dimensional path integrals arising in statistical physics problems [14]. This method was generalized to apply more generally to fitting non-convex cost-functions arising in a variety of problems, e.g., finding the optimal wiring for a densely wired computer chip [15].

The method of simulated annealing consists of three functional relationships.

1. $g(x)$: Probability density of state-space of D parameters $x = \{x^i; i = 1, D\}$.
2. $h(x)$: Probability density for acceptance of new cost-function given the just previous value.
3. $T(k)$: schedule of “annealing” the “temperature” T in annealing-time steps k, i.e., of changing the volatility or fluctuations of the two previous probability densities.

Based on functional form derived for many physical systems belonging to the class of Gaussian-Markovian systems, the algorithm chooses for g,

$$g(x) = (2\pi T)^{-D/2} \exp[-\Delta x^2/(2T)] \ , \tag{1}$$

where $\Delta x = x - x_0$ is the deviation of x from x_0 (usually taken to be the just-previously chosen point to test), and where T is clearly a measure of the fluctuations of the Boltzmann distribution g in the D-dimensional x-space.

The acceptance probability is based on the chances of obtaining a new state E_{k+1} relative to a previous state E_k,

$$h(x) = \frac{\exp(-E_{k+1}/T)}{\exp(-E_{k+1}/T) + \exp(-E_k/T)} = \frac{1}{1 + \exp(\Delta E/T)} \ , \tag{2}$$

where ΔE represents the "energy" difference between the present and previous values of the cost-function appropriate to the physical problem, i.e., $\Delta E = E_{k+1} - E_k$. This essentially is the Boltzmann distribution contributing to the statistical mechanical partition function of the system [16].

Given $g(x)$, it has been proven [17] that it suffices to obtain a global minimum of $E(x)$ if T is selected to be not faster than

$$T(k) = \frac{T_0}{\ln k} \ . \tag{3}$$

For the purposes of this paper, a heuristic demonstration follows, to show that Equation (3) will suffice to give a global minimum of $E(x)$ [13].

In order to statistically assure, i.e., requiring many trials, that any point in x-space can be sampled infinitely often in annealing-time (IOT), it suffices to prove that the products of probabilities of not generating a state x IOT for all annealing-times successive to k_0 yield zero,

$$\prod_{k_0}^{\infty} (1 - g_k) = 0 \ . \tag{4}$$

This is equivalent to

$$\sum_{k_0}^{\infty} g_k = \infty \ . \tag{5}$$

The problem then reduces to finding $T(k)$ to satisfy Equation (5).

For BA, if $T(k)$ is selected to be Equation (3), then Equation (1) gives

$$\sum_{k_0}^{\infty} g_k \ge \sum_{k_0}^{\infty} \exp(-\ln k) = \sum_{k_0}^{\infty} 1/k = \infty \ . \tag{6}$$

Although there are sound physical principles underlying the choices of Equations (1) and (2) [14], it was noted that this method of finding the global minimum in x-space was not limited to physics examples requiring *bona fide* "temperatures" and "energies." Rather, this methodology can be readily extended to any problem for which a reasonable probability density $h(x)$ can be formulated [15].

[The above Equations (1) and (6) correct Equations (1) and (6), resp., in the first VFSR paper [6]. Nothing else is affected in the rest of that paper.]

3.2. *Fast Annealing (FA)*

It was also noted that this methodology can be readily extended to use any reasonable generating function $g(x)$, without relying on the principles underlying the ergodic nature of statistical physics. Specifically, it was noted that the Cauchy distribution has some definite advantages over the Boltzmann form [13]. The Cauchy distribution,

$$g(x) = \frac{T}{(\Delta x^2 + T^2)^{(D+1)/2}} \ , \tag{7}$$

has a "fatter" tail than the Gaussian form of the Boltzmann distribution, permitting easier access to test local minima in the search for the desired global minimum.

It is instructive to note the similar corresponding heuristic demonstration, that the Cauchy $g(x)$ statistically finds a global minimum. If Equation (3) is replaced by

$$T(k) = \frac{T_0}{k} \ , \tag{8}$$

then

$$\sum_{k_0}^{\infty} g_k \approx \frac{T_0}{\Delta x^{D+1}} \sum_{k_0}^{\infty} \frac{1}{k} = \infty \ . \tag{9}$$

Note that the "normalization" of g has introduced the annealing-time index k.

The method of FA is thus statistically seen to have an annealing schedule exponentially faster than the method of BA. This method has been tested in a variety of problems [13].

3.3. *Very Fast Annealing*

In a variety of physical problems we have a D-dimensional parameter-space. Different parameters have different finite ranges, fixed by physical considerations, and different annealing-time-dependent sensitivities, measured by the curvature of the cost-function at local minima. BA and FA have g distributions which sample infinite ranges, and there is no provision for considering differences in each parameter-dimension, e.g., different sensitivities might require different annealing schedules. Also, there is no quick algorithm for calculating a D-dimensional Cauchy random generator.

For example, one might choose a D-product of one-dimensional Cauchy distributions, because the one-dimensional Cauchy has a few quick algorithms. This would also permit different T_0's to account different sensitivities.

$$g_{ik} = \frac{T_{i0}}{\Delta x^{i2} + T^2} \ . \tag{10}$$

But then we would require an annealing schedule going as

$$T_i(k) = T_0 / k^{1/D} \ , \tag{11}$$

which, although faster than BA, is still quite slow. E.g., consider $D = 6$ and assume a final temperature $T_f = 10^{-4} T_0$ is desired.

The above problems provide motivation for the development of a new algorithm. Consider a parameter α_k^i in dimension i generated at annealing-time k with the range

$$\alpha_k^i \in [A_i, B_i] \ , \tag{12}$$

calculated with the random variable y^i,

$$\alpha_{k+1}^i = \alpha_k^i + y^i (B_i - A_i) \ ,$$

$$y^i \in [-1, 1] \ . \tag{13}$$

Define the generating function

$$g_T(y) = \prod_{i=1}^{D} \frac{1}{2(|y^i| + T_i) \ln(1 + 1/T_i)} \equiv \prod_{i=1}^{D} g_T^i(y^i) \ . \tag{14}$$

Its cumulative probability distribution is

$$G_T(y) = \int_{-1}^{y^1} \cdots \int_{-1}^{y^D} dy'^1 \cdots dy'^D \, g_T(y') \equiv \prod_{i=1}^{D} G_T^i(y^i) \ ,$$

$$G^i_T(y^i) = \frac{1}{2} + \frac{\operatorname{sgn}(y^i)}{2} \frac{\ln(1 + |y^i|/T_i)}{\ln(1 + 1/T_i)} \ . \tag{15}$$

y^i is generated from a u^i from the uniform distribution

$$u^i \in U[0, 1] \ ,$$

$$y^i = \operatorname{sgn}(u^i - \frac{1}{2}) T_i [(1 + 1/T_i)^{|2u^i - 1|} - 1] \ . \tag{16}$$

It is straightforward to calculate that for an annealing schedule for T_i

$$T_i(k) = T_{0i} \exp(-c_i k^{1/D}) \ , \tag{17}$$

a global minima statistically can be obtained. I.e.,

$$\sum_{k_0}^{\infty} g_k \approx \sum_{k_0}^{\infty} [\prod_{i=1}^{D} \frac{1}{2|y^i| c_i}] \frac{1}{k} = \infty \ . \tag{18}$$

It seems sensible to choose control over c_i, such that

$$T_{fi} = T_{0i} \exp(-m_i) \text{ when } k_f = \exp n_i \ ,$$

$$c_i = m_i \exp(-n_i/D) \ , \tag{19}$$

where m_i and n_i can be considered “free” parameters to help tune VFSR for specific problems.

It has proven fruitful to use the same type of annealing schedule for the acceptance function h as used for the generating function g, i.e., Equations (17) and (19), but with the number of acceptance points, instead of the number of generated points, used to determine the k for the acceptance temperature.

New parameters α^i_{k+1} are generated from old parameters α^i_k from

$$\alpha^i_{k+1} = \alpha^i_k + y^i(B_i - A_i) \ , \tag{20}$$

constrained by

$$\alpha^i_{k+1} \in [A_i, B_i] \ . \tag{21}$$

I.e., y^i’s are generated until a set of D are obtained satisfying these constraints.

3.4. *Reannealing*

Whenever doing a multi-dimensional search in the course of a real-world nonlinear physical problem, inevitably one must deal with different changing sensitivities of the α^i in the search. At any given annealing-time, it seems sensible to attempt to “stretch out” the range over which the relatively insensitive parameters are being searched, relative to the ranges of the more sensitive parameters.

It has proven fruitful to accomplish this by periodically rescaling the annealing-time k, essentially reannealing, every hundred or so acceptance-events, in terms of the sensitivities s_i calculated at the most current minimum value of the cost function $\underline{L}$,

$$s_i = \partial \underline{L} / \partial \alpha^i \ . \tag{22}$$

In terms of the largest $s_i = s_{\max}$, it has proven fruitful to reanneal by using a rescaling for each k_i of each parameter dimension,

$$k_i \rightarrow k'_i \ ,$$

$$T'_{ik'} = T_{ik}(s_{\max}/s_i) \ ,$$

$$k'_i = (\ln(T_{i0}/T_{ik'})/c_i)^D \ . \tag{23}$$

T_{i0} is set to unity to begin the search, which is ample to span each parameter dimension. [N.b. The VFSR code used here was used previously to investigate the inverse weighting of $s_{\max}/s_i$ in the above equation. Since no code was changed for this project, the old ratio was used here. Using the equation above as given here increases the efficiency of VFSR.]

The acceptance temperature is similarly rescaled. In addition, since the initial acceptance temperature is set equal to a trial value of $\underline{L}$, this is typically very large relative to the global minimum. Therefore, when this rescaling is performed, the initial acceptance temperature is reset to the most current minimum of $\underline{L}$, and the annealing-time associated with this temperature is set to give a new temperature equal to the lowest value of the cost-function encountered to annealing-date.

Also generated are the "standard deviations" of the theoretical forms, calculated as $[\partial^2 \underline{L}/(\partial\alpha^i)^2]^{-1/2}$, for each parameter α_i. This gives an estimate of the "noise" that accompanies fits to stochastic data or functions. During the runs, also calculated are the off-diagonal elements of the "covariance matrix" for all parameters. This inverse curvature of the theoretical cost function can provide a quantitative assessment of the relative sensitivity of parameters to statistical errors in fits to stochastic systems.

A few other twists can be added, and such searches undoubtedly will never be strictly by rote. Physical systems are so different, some experience with each one is required to develop a truly efficient algorithm.

The above two sub-Sections define this method of Very Fast Simulated Reannealing (VFSR). The annealing schedule for the temperatures T_i decrease exponentially in annealing-time k, i.e., $T_i = T_{i0} \exp(-c_i k^{1/D})$. Of course, the fatter the tail of the generating function, the smaller the ratio of acceptance to generated points in the fit. However, in practice, it is found that for a given generating function, this ratio is approximately constant as the fit finds a global minimum. Therefore, for a large parameter space, the efficiency of the fit is determined by the annealing schedule of the generating function.

4. SIMULATION COMPARISONS

4.1. *Test Functions*

We evaluated the two algorithms, VFSR and GA, on a set of six robust optimizing test functions. Functions 1-5, shown below, are De Jong's five function test bed and are typically used for GA benchmarking [3]. The performances of GA and their variations on these functions are well documented [5]. For consistency with [5], our GA runs were also simulated on the University of California at San Diego GENESIS 1.2 Genetic Algorithms simulator [18]. We also choose function f_0, the objective function of Corana *et al* [19], which contains a very large number of local minima, and is very difficult to optimize.

The test suite was designed to test algorithms on functions that are:

Continuous or Discontinuous
Convex or Concave
Unimodal or Multimodal
Linear or Nonlinear
Low Dimensional or High Dimensional
Deterministic or Stochastic.

The test suite is composed of a set of six test functions $\{f_n;\, n = 0, 5\}$ that are optimized by their minimization:

$$f_0(x_1, \ldots, x_4) = \sum_{i=1}^{N} \begin{cases} (t_i \,\mathrm{sgn}\,(z_i) + z_i)^2 c d_i & \text{if } |x_i - z_i| < |t_i| \\ d_i x_i^2 & \text{otherwise ,} \end{cases}$$

$$z_i = \left\lfloor \left| \frac{x_i}{s_i} \right| + 0.49999 \right\rfloor \mathrm{sgn}\,(x_i) s_i \ ,$$

$$s_i = 0.2,\ t_i = 0.05,\ i = 1, 4 \ ,$$

$$d_i = \{1.0,\ 1000.0,\ 10.0,\ 100.0\} \ ,$$

$$c = 0.15 \; ,$$

$$-1000.0 \le x_i \le 1000.0 \; , \; i = 1, 4 \; , \tag{24}$$

where s_i, t_i, d_i, and c are coefficients defined such that f_0 defines a paraboloid with axis parallel to the coordinates, and a set of holes that increase in depth near the origin.

$$f_1(x_1, x_2, x_3) = \sum_{j=1}^{3} x_j^2 \; ,$$

$$-5.12 \le x_i \le 5.12 \; , \; i = 1, 3 \; . \tag{25}$$

$$f_2(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \; ,$$

$$-2.048 \le x_i \le 2.048 \; , \; i = 1, 2 \; . \tag{26}$$

$$f_3(x_1, \cdots, x_5) = 30.0 + \sum_{j=1}^{5} \lfloor x_j \rfloor \; ,$$

$$-5.12 \le x_i \le 5.12 \; , \; i = 1, 5 \; . \tag{27}$$

$$f_4(x_1, \ldots, x_{30}) = \sum_{j=1}^{30} x_j{}^4 + \eta \; ,$$

$$-1.28 \le x_i \le 1.28 \; , \; i = 1, 30 \; . \tag{28}$$

where η is (a) a random number with distribution (0,1), or (b) a random number bounded between [0,1).

$$f_5(x_1, x_2) = \frac{1}{500 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ji})^6}} \; ,$$

$$a_{j_1} = \{ -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16, 0, 16, 32, -32, -16,$$

$$0, 16, 32, -32, -16, 0, 16, 32\} \; ,$$

$$a_{j_2} = \{ -32, -32, -32, -32, -32, -16, -16, -16, -16, -16, 0, 0, 0, 0, 0, 16, 16,$$

$$16, 16, 16, 32, 32, 32, 32, 32\} \; ,$$

$$-65.536 \le x_i \le 65.536 \; , \; i = 1, 2 \; . \tag{29}$$

Function f_0 is somewhat like function f_2. It is very difficult to minimize correctly when $n = 4$, with over 10^{20} local minima to be trapped in. Optimization methods based on gradient descent are quite likely to be caught in one of the many local minima. Function f_1 tests simple sum of squares with 1 minimum at $x_i = 0$. The function f_2 is the classical function of Rosenbrock and Chebyquad in two dimensions that is unimodal, yet difficult to minimize [19]. The next function, f_3, is the plateau function, generated as the sum of integer threshold values. The five dimensional space has one minimum and is discontinuous. Function f_4 is a noisy quartic function of 30 variables originally defined as $f_4(x_1, \ldots, x_{30}) = \sum_{j=1}^{30} x_j{}^4 + \eta$, where η is produced by Gaussian noise with distribution n(0,1) [3]. While the intent of this function is to determine an optimizer's performance in the presence of noise, this function is perhaps flawed, as no definite global minimum exists. Once an algorithm minimizes $\sum_{j=1}^{30} x_j{}^4$, it could continue executing, waiting for smaller values of η to be produced. By performing more function evaluations, an inefficient optimization technique might find better solutions than an efficient optimization (that terminates with fewer function evaluations). In response to this problem, we choose η to be a random variable with uniform distribution, and bounded by [0,1). This ensured that there could be only one functional minima located at 0. Lastly, f_5 spans a 2-dimensional space, with a global minima ≈ 0.998004. It is similar to f_0 but has only 25 local minima. In our opinion, f_0 is a much better test of an optimizing algorithm than De Jong's f_5.

In each run, each algorithm performed a minimum of 1000 function calls unless it terminated prematurely by finding the function's global minima. Global minima values of 0 are shown as 10^{-12}. For those runs not converging to the optimum, limits were placed on the number of VFSR function calls, and the number GA generations. For example, for VFSR, a cutoff of $f < 10^{-12}$ was one criteria for those functions with zero as a minimum (although zero was reached to within machine double-precision accuracy).

For all the Genetic Algorithm simulation runs, the fitness functions to maximize were $-f_i$. Real values were encoded as binary strings of various lengths. f_0 used 32 bits. For the others, we used the defaults given in the GA simulator: f_1 used 10 bits, f_2 used 12 bits, f_3 used 10 bits, f_4 used 8 bits, and f_5 used 17 bits. (Although this entailed some prior tuning of GA for these specific systems, no such tuning was done for the VFSR runs reported here.) Bit string lengths depend on the function being optimized; when states are encoded as short bit strings, they converge more quickly than those encoded as long string. However, the decoded precision of the shorter string is less than longer strings. Factors such as the string length, population sizes, mutation rates, number of generations, and percent of population culled were heuristically determined, and were dependent on the function being optimized. These relatively minor procedures are considered to be part of the algorithm defining GA.

Both the GA and VFSR algorithms were taken from existing libraries previously prepared for other problems. While this approach does not guarantee the best fitting procedures for GA or VFSR on these specific problems, our results do suggest the relative merits of these algorithms that might be expected on new problems. Population sizes for the GA runs were 30000, 100, 50, 2000, 30000, and 200 individuals for functions f_0 through f_5 respectively. No determination was made to find optimal population sizes. (Again, although this entailed some prior tuning of GA for these specific systems, no such tuning was done for the VFSR runs reported here.) Additionally, we have not plotted each generation versus its best function evaluation, because VFSR has no corresponding attribute and because the GA runs were simulated sequentially. However, a generation-performance plot would be a more accurate estimate of the power of the genetic algorithms, especially if the GA simulations were run on parallel hardware such as the Connection Machine [20]. Then all function evaluations (performed in one generation) could be evaluated in parallel. It should be noted that parallel processing of GA can lead to spurious correlations and premature convergence [21].

It should be noted in this context that VFSR also lends itself well to parallelization. Blocks of random numbers can be generated in parallel, and then sequentially checked to determine a generating point satisfying all boundary conditions. Also, advantage can be taken of the low ratio of acceptance to generated points typical in VFSR, to generate blocks of cost functions, and then sequentially check these to determine the next best current minimum.

4.2. *Results*

FIGURES 1-6

Figures 1 through 6 illustrate the performances of the two algorithms on the corresponding functions f_0 through f_5. Solid and short dashed lines each represent one VFSR run each, and dashed and long dashed lines represent one GA run each. The runs are log-log plotted to show relative convergence rates of each algorithm. The abscissa indicates the number of function calls, while the ordinate shows the best function evaluation found so far. The VFSR and GA optimizations of functions f_1 and f_2 were fairly competitive, which may be attributed to the fact that these functions are continuous and relative concave within the boundary constraints.

Although stochastic, the VFSR performance variances on functions f_0, f_1, f_3, and f_4 were relatively small. VFSR's performance on f_0 is most illustrative. The GA variances were substantial on f_1 and f_2. A VFSR run also was performed for f_0 for the case of 10 parameters, and it converged to zero within 10^7 generations of that cost function; we considered it to be too much a strain on resources to use this number of variables with GA. Both algorithms found the minimum for function f_5 in about the same time, although VFSR had the quickest approach. Overall, the VFSR runs converged faster than the GA runs, and with smaller variances.

5. DISCUSSION

No rigorous proofs have been given for the convergence of VFSR, only a heuristic rationale for its success. It is expected that the obvious utility of this algorithm will motivate such proofs. However, actual fits to data are a finite process, and often even only heuristic guides to algorithms that obviously fit many classes of data are important. The annealing schedule for the temperatures T_i decrease exponentially in annealing-time k, i.e., $T_i = T_{i0} \exp(-c_i k^{1/D})$. This algorithm is exponentially faster than the fast Cauchy annealing, where $T_i = T_0/k$, and even faster than Boltzmann annealing, where $T_i = T_0/\ln k$.

With regard to the Genetic Algorithm simulations, Genetic Algorithms have the problem of parameter estimation. Currently only heuristic estimates of parameters such as population size and string length were used in our simulations, and these parameter values were taken from [18].

When comparing the two algorithms, one should note that the GA and VFSR random number generator values differ almost immediately during the optimization. VFSR makes additional calls to the generator to incorporate noise into its decision process, making it difficult to rigidly compare performances.

Genetic algorithms are not designed to ergodically sample and cover the space in a maximally efficient way. There is some doubt as to whether they are ergodic at all, but the prime benefit of Genetic Algorithms occurs during each generation when all individuals can be evaluated in parallel, making Genetic algorithms excellent candidates for running on fine grained parallel processing hardware such as the connection machine. However, parallelization for some systems can be somewhat thwarted. For some applications, such as closed loop control, performance evaluation of the system can be delayed or occur infrequently. Additionally, the evaluations may depend not only on the current state, but also on previous states.

In contrast, VFSR is largely sequential in moving from one optimized value to the next. States must be sampled sequentially, for acceptability and to permit identification of current local minima about which new test parameters are chosen. However, we also have outlined how VFSR can be parallelized, at the stage of preparing random numbers for generating points, and at the stage for preparing cost functions for acceptance criteria. Additionally, when fitting dynamic systems, e.g., as performed for three physical systems to date [7,9,11], parallelization is attained by independently calculating each time epoch's contribution to the cost function.

An interesting variation of GA developed by Ackley [22], Stochastic Iterated Genetic Hillclimbing (SIGH), combines simulated annealing, hillclimbing, and genetic algorithms, creating a coarse-to-fine search strategy that is used as a function optimizer, which improves on the performances of Genetic Algorithms. While the SIGH algorithms and variations allow genetic algorithmic search to find states that are suitable for Hillclimbing, they still are not ergodic, and do not provide for importance sampling over the state space.

Problems with large dimensional problems and the "curse of dimensionality" (as illustrated by function f_0) yet remain to be solved. However, with the advent of fine grain parallel computers to perform concurrent function evaluation, the optimization performances of VFSR and GA would undoubtedly improve. For Genetic Algorithms, larger population sizes could sample larger areas of the state space, allowing the algorithm to more quickly converges to a local minima.

Our future investigations will examine and compare neural network function optimizers with VFSR, to more accurately characterize the performance benefits and deficits of the the VFSR algorithm. However, from our current simulations, VFSR is a robust and powerfully efficient tool for function optimization.

ACKNOWLEDGMENT

Larry Eshelman and Dave Schaffer, Philips Laboratories, gave a careful reading to our initial draft and pointed out several misprints. Mark Johnson, MicroUnity Systems Engineering, Inc., first brought the function f_0 to our attention by his posting of the code of this function to a public electronic bulletin board. Graphs were produced using XVGR (graphics for exploratory data analysis), a public domain software package running under UNIX and X11, developed by Paul Turner at the Oregon Graduate Institute.

REFERENCES

1. J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, (1975).
2. A.D. Bethke, Genetic Algorithms as Function Optimizers, Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, (1981).
3. K.A. De Jong, An Analysis of the Behavior of a Class of Genetic Adaptive system, Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, (1981).
4. D.E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley, Reading, Mass, (1989).
5. N.N. Schraudolph and R.K. Belew, Dynamic parameter encoding for genetic algorithms, Technical Report LAUR 90-2795, Los Alamos National Laboratory, Los Alamos, NM, (1990).
6. L. Ingber, Very fast simulated re-annealing, *Mathl. Comput. Modelling* **12** (8), 967-973 (1989).
7. L. Ingber, H. Fujio, and M.F. Wehner, Mathematical comparison of combat computer models to exercise data, *Mathl. Comput. Modelling* **15** (1), 65-90 (1991).
8. L. Ingber and D.D. Sworder, Statistical mechanics of combat with human factors, *Mathl. Comput. Modelling* **15** (11), 99-127 (1991).
9. L. Ingber, Statistical mechanical aids to calculating term structure models, *Phys. Rev. A* **42** (12), 7057-7064 (1990).
10. L. Ingber, M.F. Wehner, G.M. Jabbour, and T.M. Barnhill, Application of statistical mechanics methodology to term-structure bond-pricing models, *Mathl. Comput. Modelling* **15** (11), 77-98 (1991).
11. L. Ingber, Statistical mechanics of neocortical interactions: A scaling paradigm applied to electroencephalography, *Phys. Rev. A* **44** (6), 4017-4060 (1991).
12. L. Ingber, Generic mesoscopic neural networks based on statistical mechanics of neocortical interactions, *Phys. Rev. A* **45** (4), R2183-R2186 (1992).
13. H. Szu and R. Hartley, Fast simulated annealing, *Phys. Lett. A* **122** (3-4), 157-162 (1987).
14. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, Equation of state calculations by fast computing machines, *J. Chem. Phys.* **21** (6), 1087-1092 (1953).
15. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, Optimization by simulated annealing, *Science* **220** (4598), 671-680 (1983).
16. K. Binder and D. Stauffer, A simple introduction to Monte Carlo simulations and some specialized topics, in *Applications of the Monte Carlo Method in statistical physics*, (Edited by K. Binder), pp. 1-36, Springer-Verlag, Berlin, (1985).
17. S. Geman and D. Geman, Stochastic relaxation, Gibbs distribution and the Bayesian restoration in images, *IEEE Trans. Patt. Anal. Mac. Int.* **6** (6), 721-741 (1984).
18. N.N. Schraudolph and J.J. Grefenstette, A Users Guide to GAUCSD 1.2, Report, University of California at San Diego, La Jolla, CA, (1991).
19. A. Corana, M. Marchesi, C. Martini, and S. Ridella, Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm, *ACM Trans. Mathl. Software* **13** (3), 272-280 (1987).
20. W.D. Hillis, *The Connection Machine*, The MIT Press, Cambridge, MA, (1985).
21. J.D. Schaffer, L.J. Eshelman, and D. Offutt, Spurious correlations and premature convergence in genetic algorithms, in *Foundations of Genetic Algorithms*, (Edited by G. Rawlins), pp. 102-112, Morgan Kaufmann, San Mateo, CA, (1991).
22. D.H. Ackley, Stochastic Iterated Genetic Hillclimbing, Ph.D. Thesis, Department of Computer Sciences, Carnegie Mellon University, Pittsburgh, PA, (1987).

FIGURE CAPTIONS

Figure 1. Comparison between GA and VFSR is given for function f_0. Solid and short dashed lines each represent one VFSR run each, and dashed and long dashed lines represent one GA run each. The runs are log-log plotted to show relative convergence rates of each algorithm. The abscissa indicates the number of function calls, while the ordinate shows the best function evaluation found so far.

Figure 2. Comparison between GA and VFSR is given for function f_1. See Figure 1 for legend.

Figure 3. Comparison between GA and VFSR is given for function f_2. See Figure 1 for legend.

Figure 4. Comparison between GA and VFSR is given for function f_3. See Figure 1 for legend.

Figure 5. Comparison between GA and VFSR is given for function f_4. See Figure 1 for legend.

Figure 6. Comparison between GA and VFSR is given for function f_5. See Figure 1 for legend.

Figure 1

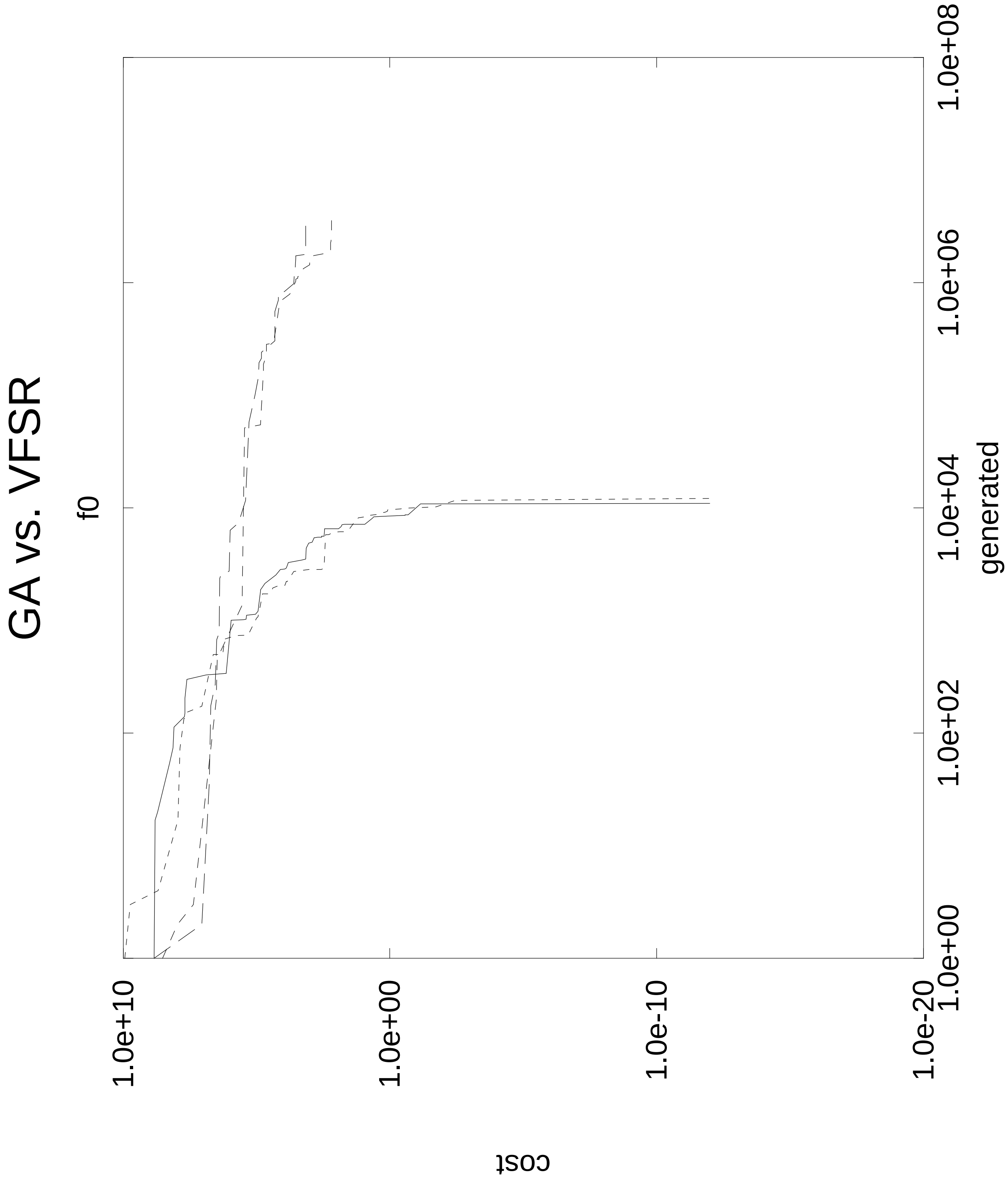

GA vs. VFSR
f0
generated
cost
1.0e+00
1.0e+02
1.0e+04
1.0e+06
1.0e+08
1.0e+10
1.0e+00
1.0e-10
1.0e-20

Figure 2

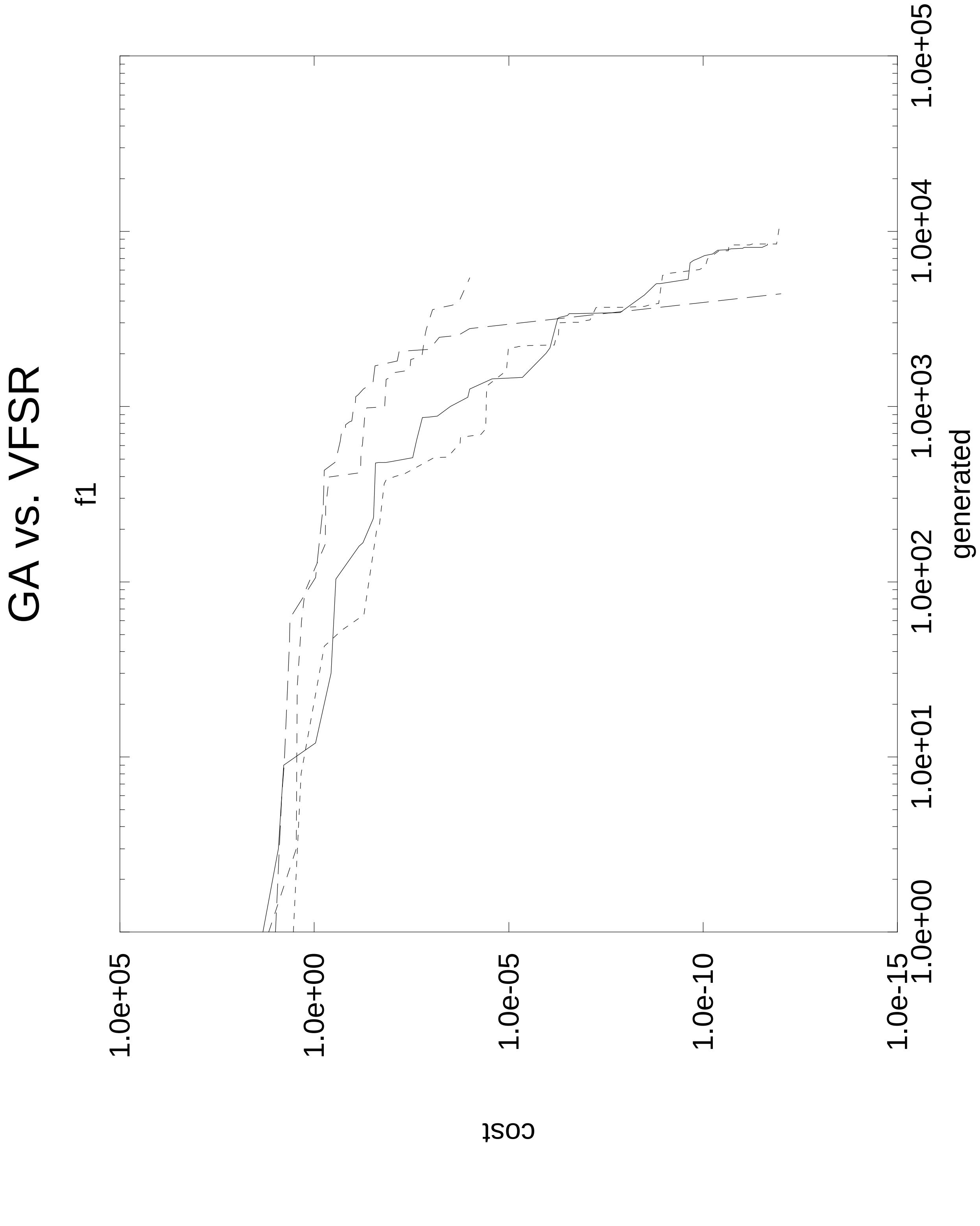

GA vs. VFSR
f1
generated
cost
1.0e+00
1.0e+01
1.0e+02
1.0e+03
1.0e+04
1.0e+05
1.0e+05
1.0e+00
1.0e-05
1.0e-10
1.0e-15

Figure 3

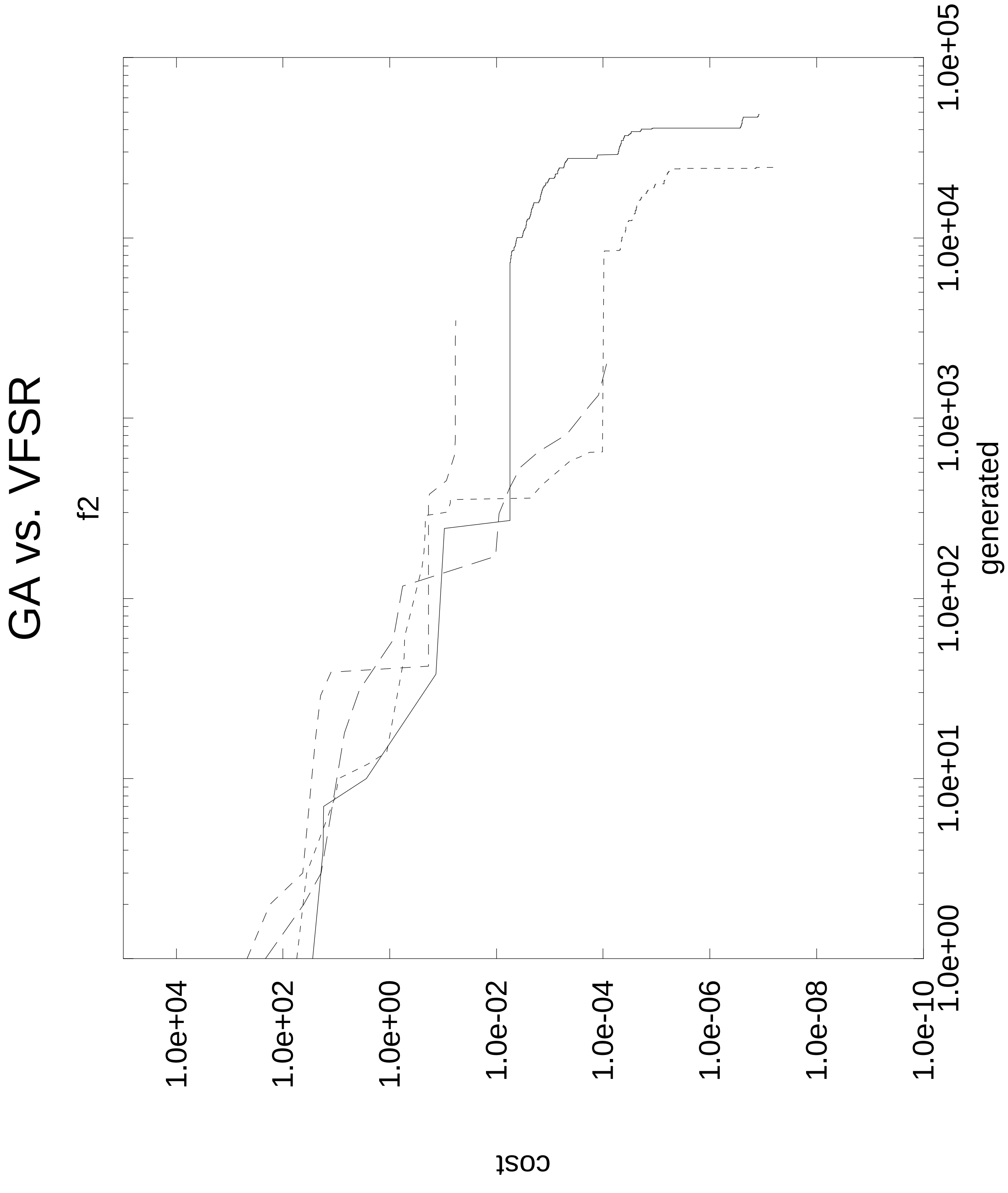

GA vs. VFSR
f2
generated
cost
1.0e+00
1.0e+01
1.0e+02
1.0e+03
1.0e+04
1.0e+05
1.0e+04
1.0e+02
1.0e+00
1.0e-02
1.0e-04
1.0e-06
1.0e-08
1.0e-10

Figure 4

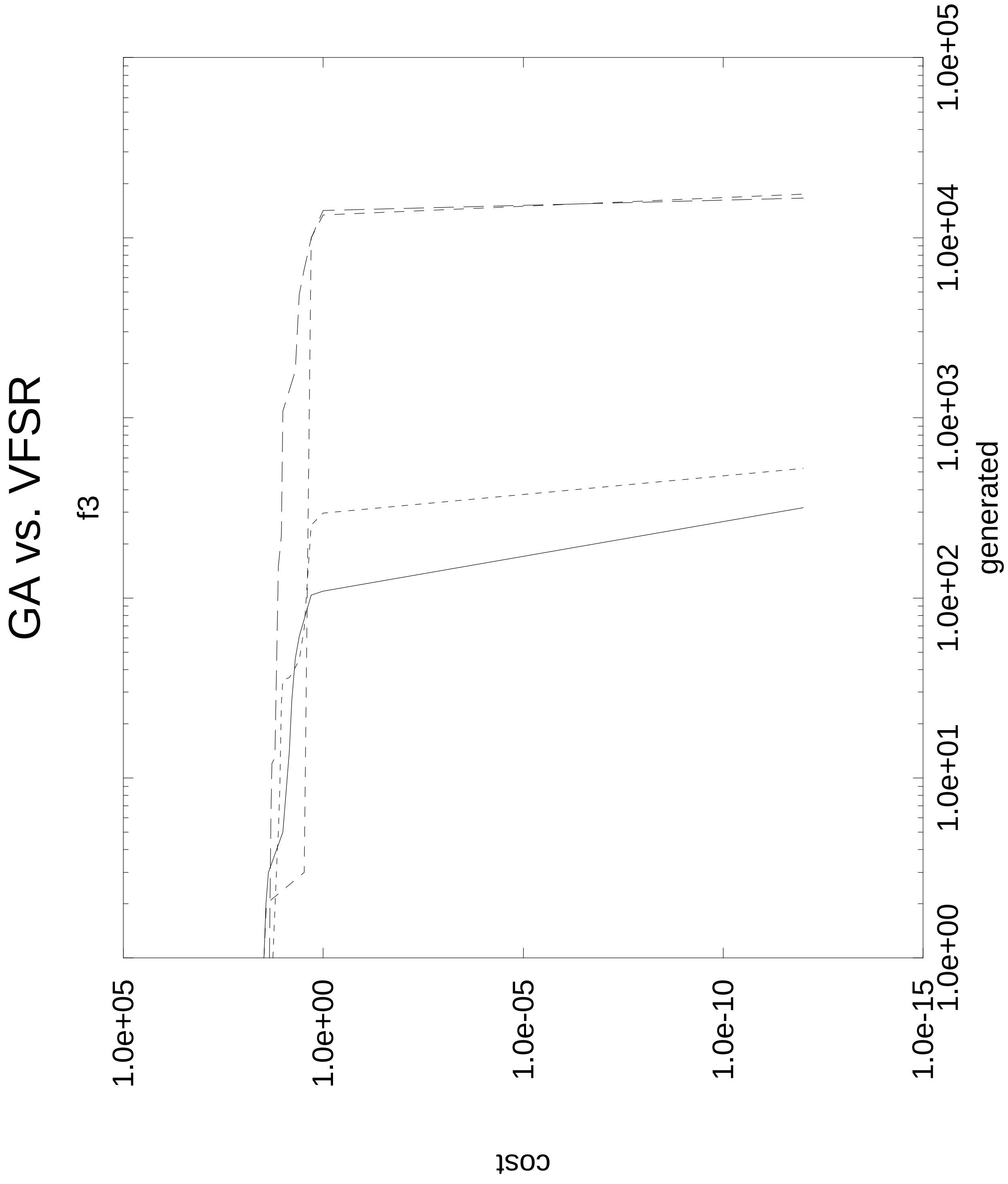

GA vs. VFSR
f3
generated
cost
1.0e+00
1.0e+01
1.0e+02
1.0e+03
1.0e+04
1.0e+05
1.0e+05
1.0e+00
1.0e-05
1.0e-10
1.0e-15

Figure 5

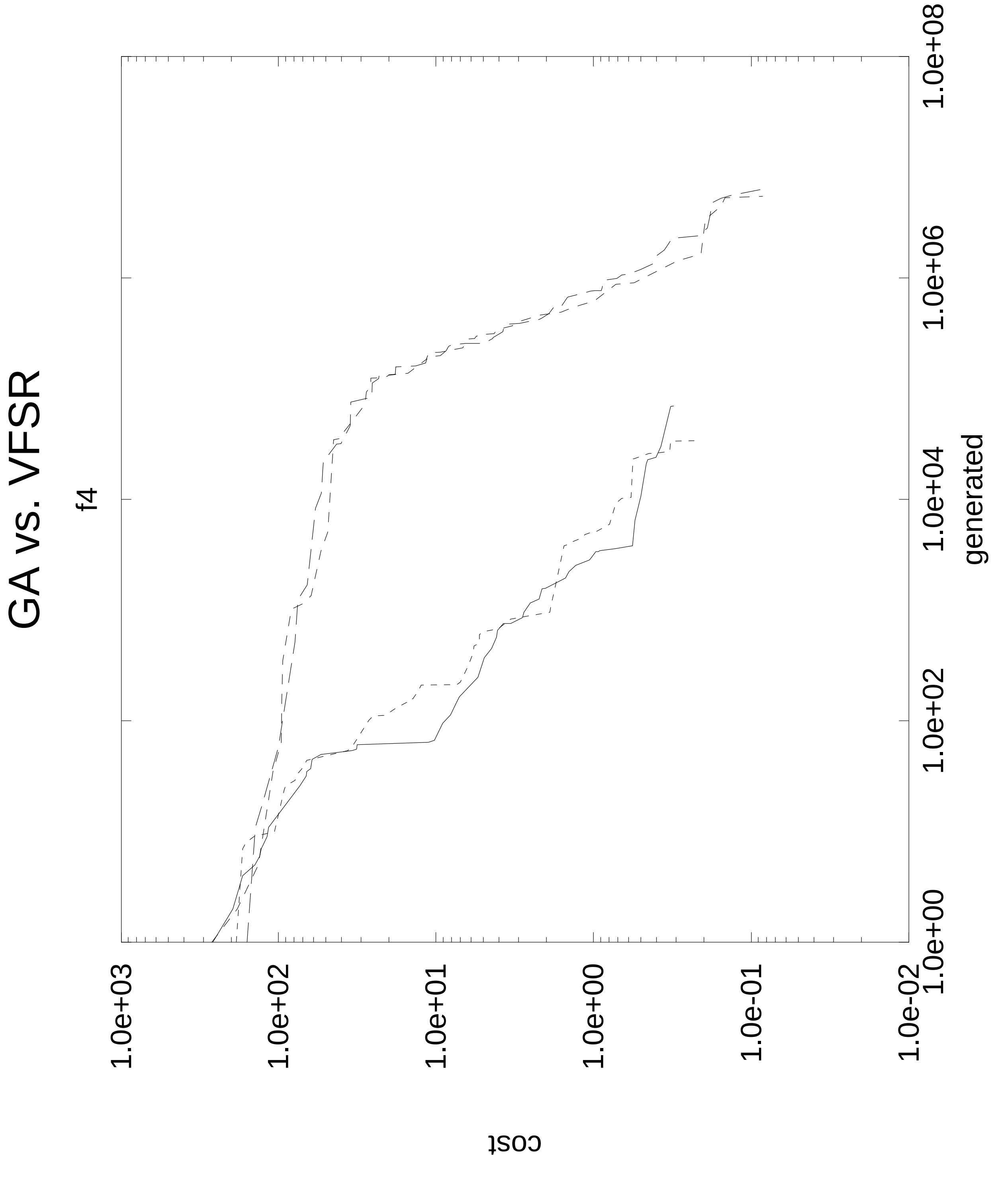
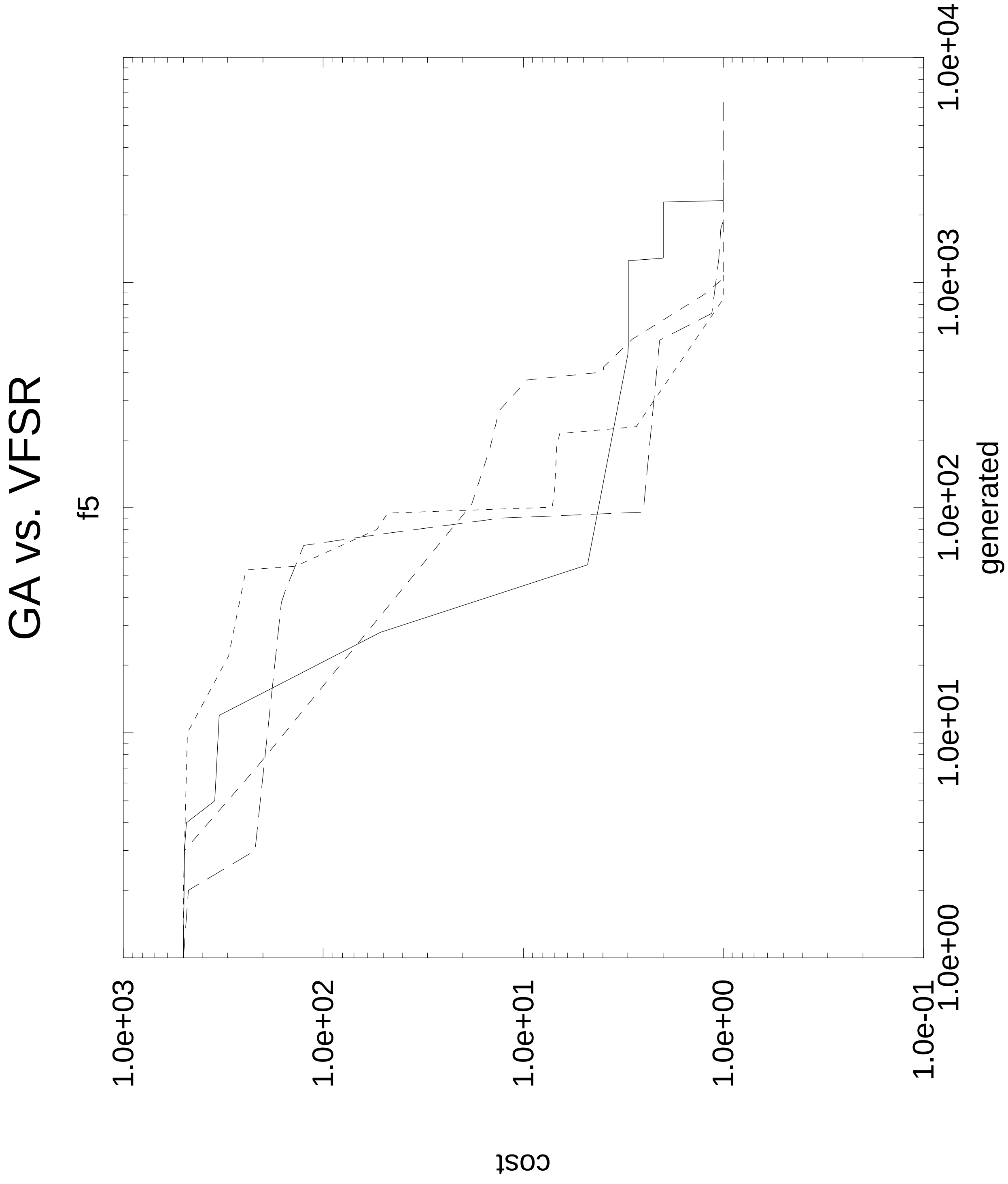

GA vs. VFSR
f4
generated
cost
1.0e+00
1.0e+02
1.0e+04
1.0e+06
1.0e+08
1.0e+03
1.0e+02
1.0e+01
1.0e+00
1.0e-01
1.0e-02

Figure 6


GA vs. VFSR
f5
cost
generated
1.0e+03
1.0e+02
1.0e+01
1.0e+00
1.0e-01
1.0e+00
1.0e+01
1.0e+02
1.0e+03
1.0e+04