Parallel Genetic Algorithms

R. Shonkwiler
School of Mathematics
Georgia Institute of Technology
Atlanta, GA. 30332
e-mail: shenk@math.gatech.edu

Abstract

A universal method for parallelizing a Genetic Algorithm is given. Referred to as *IIP* parallel, independent and identical processing, theoretical analysis shows that the technique achieves a speedup using m processors given by ms^{m-1} where the acceleration factor s is a parameter depending on the details of the GA. Typically $s > 1$. The results are illustrated on two problems small enough that the exact calculation of s can be made and compared to the asymptotic estimates. The results are also illustrated on the non–trivial Inverse Fractal Problem. It is noted that the same results have been attained elsewhere on a wide variety of well known significant problems.

1 INTRODUCTION

Our method for parallelizing a Genetic Algorithm is simple – run the identical algorithm on each processor each independently of the other. (Of course the random number generator is seeded differently in each process.) The only communication between processes is that one of them gathers the ending result of them all and reports to the user. We call this identical, independent processing *IIP* parallel. Can this simple technique be effective? In fact in numerous applications including the 1-D fractal inverse problem (Shonkwiler, Mendivil, Deliu 1991), optimizing network throughput (Akyildiz, Shonkwiler 1990), the join problem (Omiecinskii, Shonkwiler 1990), the n-queens problem (Ghannadian, Shonkwiler, Alford 1993), the k-clique problem (Miller, Shonkwiler 1992), and the catalog search problem (Carlson, Ingrim, Shonkwiler 1993), it has achieved superlinear parallel speedup. These empirical results are now supported theoretically (Shonkwiler, Van Vleck 1993). It is the aim of this paper to explain the basis for *IIP* speedup as applied to Genetic Algorithms and to illustrate it with two small problems for which exact calculations can be done. In summary, the *IIP* speedup, SU, expected from the parallelization of a GA using m processors, is given by

$$SU \approx ms^{m-1}$$

where the *acceleration* parameter s depends on the details of the algorithm but is usually greater than 1.

Assume the GA is seeking the maximum of some fixed real-valued function, or objective, f defined on a set D called the *domain*. Suppose the GA consists of a sequence of "populations" X_k, $k = 0, 1, 2, \ldots$, that is finite subsets of D,

$$X_k = \{x_1^k, x_2^k, \ldots, x_n^k\} \subset D,$$

and a stochastic transition scheme $\mathcal{O}$

$$X_{k+1} = \mathcal{O}(X_k),$$

carrying the k^{th} population into the $k+1^{st}$. Of course the transition scheme can be complex and will depend upon the objective f. We will suppose however that the transition process is such that X_{k+1} depends only on X_k and perhaps on k but not on $X_{k-1}, X_{k-2}, \ldots, X_0$. Then we may take as the *states* of a Markov Chain the set of all possible populations A_1, A_2,...,A_N. We assume that the collection $\mathcal{A}$ of these populations is finite (but very large in number generally). This assumption is always satisfied for a Genetic Algorithm. The transition scheme defines a transition probability matrix P as the matrix of probabilities p_{ij} where

$$p_{ij} = \Pr(X_{k+1} = A_j \,|\, X_k = A_i), \qquad i, j = 1, 2, \ldots, N.$$

Here X_k is the random variable representing the specific population selected on the k^{th} iteration of the

algorithm. In most GA's the probabilities p_{ij} do not depend on the iteration count k and we will make this assumption.

2 EXPECTED HITTING TIME

The optimization problem is solved at time k if an optimizing domain point x_* belongs to the population X_k. Let $\mathcal{G}$ denote the set of populations containing x_*, then we are interested in the first time, $k = \theta$, that $X_k \in \mathcal{G}$. However since the process is stochastic, this could be different from run to run. The *expected hitting time* is the average θ over all possible runs. Now by definition

$$E(\theta) = \sum_{t=1}^{\infty} t\Pr(\theta = t),$$

but it is easy to see that this sum may be re-written as

$$E(\theta) = \sum_{t=1}^{\infty} \Pr(\theta \geq t). \tag{2.1}$$

We refer to the sequence $\Pr(\theta \geq t), t = 1, 2, \ldots$ as the *complementary hitting time distribution.*

Let $\hat{P}$ be the matrix resulting from P by the deletion of every row and column containing a population $A \in \mathcal{G}$. It can be shown, cf. (Shonkwiler, Van Vleck 1993), that the terms of the complementary hitting time distribution are given by the dot products

$$\Pr(\theta \geq k) = \hat{\alpha} \cdot \hat{P}^{k-1}\mathbf{1} \ , \quad k = 1, 2, \ldots, \tag{2.2}$$

where $\mathbf{1}$ is the vector of appropriate dimension all of whose terms are 1 and $\hat{\alpha}$ is the goal states deleted starting distribution on the populations. That is the i^{th} component α_i of $\hat{\alpha}$ is the probability that the genetic algorithm begins with the i^{th} population A_i (deleting goal states).

3 THE ACCELERATION FACTOR s

By the Perron–Frobenius Theorem (Seneta 1981)$\hat{P}$ has a positive eigenvalue λ equal to its spectral radius, i.e. λ is the largest in magnitude eigenvalue. Under mild additional conditions on $\hat{P}$ (irreducibility and aperiodicity) λ will be the only eigenvalue of this magnitude and

$$0 < \lambda < 1.$$

In terms of the Genetic Algorithm, λ represents the probability of not finding a goal population in one iteration of the algorithm and so $1 - \lambda$ is the probability that the goal will be found in one iteration. These probabilities apply however only after special effects due to starting up the Genetic Algorithm have died out. The acceleration accounts for the start-up effect.

Let χ and ω be positive right and left eigenvectors of $\hat{P}$ corresponding to λ, (guaranteed to exist by the Perron-Frobenius Theorem), that is

$$\hat{P}\chi = \lambda\chi \qquad \text{and} \qquad \omega^T\hat{P}\Phi = \lambda\omega^T$$

where ω^T is the transpose (row vector) of ω. By normalization we may assume ω is a probability vector, $\sum\omega_i = 1$ and that $\omega^T\chi = 1$. Then ω and χ are uniquely determined.

Definition. Define the *acceleration* s to be the reciprocal dot product

$$s^{-1} = \hat{\alpha} \cdot \chi \tag{3.1}$$

where $\hat{\alpha}$ and χ are as above. Then the following holds (Shonkwiler, Van Vleck 1993)

Theorem.

$$s^{-1} = \lim_{k\to\infty} \frac{1}{\lambda^k}\hat{\alpha} \cdot \hat{P}^k\mathbf{1} \ . \tag{3.2}$$

Substituting the indicated limiting value into (2.2) gives an approximate expression for the expected hitting time

$$\begin{aligned} E(\theta) &= \sum_{k=1}^{\infty} \Pr(\theta \geq k) \\ &= \sum_{k=1}^{\infty} \hat{\alpha} \cdot \hat{P}^{k-1}\mathbf{1} \\ &\approx \sum_{k=1}^{\infty} s^{-1}\lambda^{k-1}. \end{aligned}$$

Summing the geometric series gives

$$E(\theta) \approx \frac{1}{s}\frac{1}{1-\lambda}. \tag{3.3}$$

4 PARALLEL GENETIC ALGORITHMS

As stated in the introduction, suppose m GA's are run independently in parallel. Let $\theta_1, \theta_2, \ldots, \theta_m$ be the respective hitting time random variables for each process. Note that this parallel GA solves the problem when the first independent process does, i.e. the expected hitting time, Θ, for the parallel GA is given by

$$\Theta = \min\{\theta_1, \theta_2, \ldots, \theta_m\}.$$

It is evident that the event $\Theta \geq t$ is equivalent to the event

$$\theta_1 \geq t \quad \text{and} \quad \theta_2 \geq t \quad \text{and} \cdots \text{and} \quad \theta_m \geq t.$$

Therefore by independence we have the following.

Proposition. For m identical, independent processes

$$\Pr(\Theta \geq t) = (\Pr(\theta \geq t))^m, \quad t = 1, 2, \ldots \tag{4.1}$$

and so

$$\begin{aligned} E_m &= E(\Theta) = \sum_{t=1}^{\infty} (\Pr(\theta \geq t))^m \approx \left(s^{-1}\lambda^{t-1}\right)^m \\ &\approx \frac{1}{s^m}\frac{1}{1-\lambda^m} \end{aligned} \tag{4.2}$$

where E_m is the expected hitting time for m parallel processes.

Definition. By *speed-up* for m processors we mean the ratio

$$\text{Speed-Up} = \frac{E(\theta)}{E(\Theta)} \tag{4.3}$$

of the expected running time for one process to that of the m parallel processes.

Theorem. Under the conditions stated above, the speed-up for m processes is given by

$$\begin{aligned} \text{Speed-up} &= s^{m-1}\tfrac{1-\lambda^m}{1-\lambda} + O(1-\lambda^m) \\ &\longrightarrow ms^{m-1} \quad \text{as} \quad \lambda \to 1. \end{aligned}$$

Here $O(1-\lambda^m)$ means the order of the error is some constant times $1-\lambda^m$. The proof is given in (Shonkwiler, Van Vleck 1993).

Note that for λ near 1, the ratio $(1-\lambda^m)/(1-\lambda)$ is near m and so the speed-up is approximately ms^{m-1}. Hence speedup is superlinear for $s > 1$, see fig. 1.

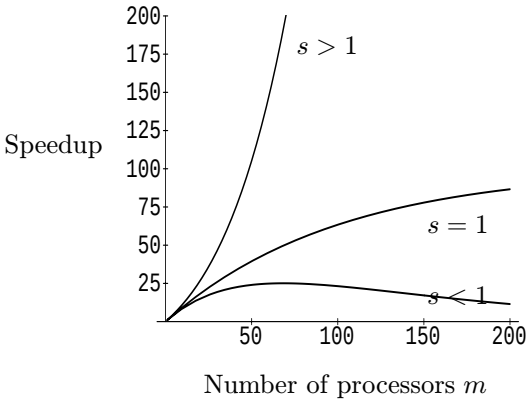

Figure 1: SPEEDUP FOR $\lambda = 0.99$

5 RESULTS

We include results for two problems small in size so that exact theoretical calculations can be made. The *password problem* is characterized by its completely neutral, totally flat objective function. It serves as a reminder why it is that general results about Monte Carlo methods are hard to come by. The *Sandia Mountain problem* is characterized by its relatively large basin for the suboptimal minima compared with the small basin for the true global minimum (hence is "deceptive" see (Goldberg 1989)).

We also include results for the non–small and difficult Inverse Fractal Problem. Despite intensive efforts to solve this problem analytically, it remains open in that regard, (Vrscay 1991). However a Genetic Algorithm can be highly effective in computing approximate solutions, see (Shonkwiler, Mendivil, Deliu 1991).

5.1 PASSWORD PROBLEM

Assume that a J character password chosen from an alphabet of M symbols is to be found. Trying a proposed solution results in either failure or success, there are no hints. The domain D consists of all strings of J legal symbols, $\text{card}(D) = M^J$, and for $x \in D$ the objective function will be taken as

$$f(x) = \begin{cases} 1, & \text{if } x \text{ is correct} \\ 0, & \text{if } x \text{ is incorrect.} \end{cases}$$

In reality, except for the extreme nature of its objective function, the password problem is typical of very many problems encountered in practice. Indeed, any problem $u = f(x_1, \ldots, x_n)$ defined on a rectangle

$$a_i \leq x_i \leq b_i, \qquad i = 1, \ldots, n$$

in n-dimensional Euclidean space $\mathbf{R}^n$ has this form computationally. For if the binary floating point representations of each component consisting of md mantissa digits, ed exponent digits, and one sign digit

$$x_i = s^i b_1^i b_2^i \ldots b_{md}^i e_1^i \ldots e_{ed}^i$$

are concatenated, there results a password problem with $J = n(md + ed + 1)$ characters from the alphabet $\{0, 1\}$ of size $M = 2$. In this way such a problem with a general objective function becomes a *password problem with hints*.

Returning to the problem at hand, we attempt its calculation by a GA having a single unary stochastic operator, i.e. a mutation. Our unary operator will be

to modify the present (failed attempt) x by selecting a character position at random from $1, 2, \ldots, J$ and replacing the letter of x at that position by a randomly chosen letter from the alphabet, *one character uniform replacement*. There results a transition probability matrix whose row for x, unless x is the solution, has a zero in every column corresponding to a $y \in D$ differing from x in two or more positions. The probability for those $y \in D$ differing in exactly one position from x is $1/(JM)$, and the probability that x itself is reselected is $1/M$.

Note that P is symmetric. Further $\hat{P}$ (equal to P with the goal row and column removed) is also symmetric and has unequal row sums. The latter follows since some states of $\hat{P}$ lead to the goal while others do not.

With the choices $J = 4$, and $M = 5$ the matrix P is 625×625 and $\hat{P}$ is 624×624 and it is possible to calculate all the relevant optimization characteristics exactly. Assuming a uniformly selected starting state, in Table 1 we show the principle eigenvalue λ, the s-factor s, the exact expected hitting time E, and the exact expected hitting times E_2, E_4, E_8 for 2, 4, and 8 multiprocesses implementations. The expectation E is obtained from the limit (2.1) or can be calculated directly (see (Isaacson, Madsen 1976)). The expectations E_2, E_4, and E_8 are calculated from (4.2). Further in Table 1 we give the averaged empirical expectations $\hat{E}_2$, $\hat{E}_4$, $\hat{E}_8$ of 100 runs (simulations of the Markov Chain) and the corresponding empirical speed-up's. By "time" in the runs we mean the number of iterations taken by the solver process. We (artifically) exclude the possibility of starting in the goal state.

Table 1

Password Problem without hints		
$\lambda = .998830$	$s = 1.000255$	-
$E = 854.7$	$\hat{E} = 870.2$	-
$E_2 = 427.5$	$\hat{E}_2 = 424.2$	$SU_2 = 1.99$
$E_4 = 213.9$	$\hat{E}_4 = 235.7$	$SU_4 = 3.99$
$E_8 = 107.1$	$\hat{E}_8 = 125.9$	$SU_8 = 7.98$

5.2 GENETIC ALGORITHM SOLVER FOR THE SANDIA MOUNTAIN PROBLEM

Let the domain D be the set of integers $D = \{0, 1, ..., N\}$, card$(D) = N + 1$, and let the objective function be

$$f(x) = \begin{cases} \frac{N-x}{N-1}, & x = 1, 2, \ldots, N \\ -1, & x = 0, \end{cases}$$

i.e. a long gradual uphill slope from $x = N$ to $x = 1$, but then a steep drop at $x = 0$, see fig. 2. The global

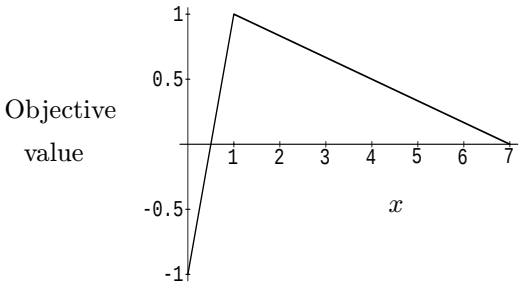

Figure 2: SANDIA MOUNTAIN PROBLEM

minimum of -1 occurs at $x = 0$ and this minimum has a basin of two domain points. There is also a local minimum of 0 occurring at $x = N$. This basin is of size N. To keep the example within reasonable size let $N = 7$ and represent the $N + 1 = 8$ domain values in binary

$$0 \leftrightarrow (000)_2, \quad 1 \leftrightarrow (001)_2, \ldots, 7 \leftrightarrow (111)_2.$$

We employ a standard Genetic Algorithm (cf. (Goldberg 1989)) with a population size of 2, reproductive success taken in proportion to fitness ϕ which will be defined as

$$\phi(x) = 2 - f(x), \quad x \in D,$$

crossover based on bit strings and a bit mutation rate of $p_m = 0.1$. The number of distinct populations is $\frac{8 \cdot 9}{2} = 36$.

An iteration of the algorithm will consist of: (1) a reproduction of the present population, each "individual" in proportion to its fitness; let P_R be the 36×36 transition probability matrix for this process. Next (2) a crossover or mating process based on bit strings. Note that the mate selection matrix is just the identity because the population size is 2. For this 3 bit example the two crossover sites, between bits 1 and 2 or between bits 2 and 3, are chosen equally likely. Let P_c denote the 36×36 matrix for this process. Then (3) a mutation in which one of the two population members is chosen equally likely and each bit of the chosen member is reversed ($0 \to 1$ and $1 \to 0$) with probability p_m independently; P_m denotes the resulting 36×36 transition matrix. Finally (4) the required function evaluations are performed to obtain the next generation's fitness and to update the "best" random variable B_t.

These processes may be elaborated as follows. During the reproduction process the population $< i, j >$ will

become one of the populations $< i, i >$ or $< i, j >$, or $< j, j >$. If $\phi_i \equiv \phi(i)$ is the fitness of $i \in D$, then the probability of obtaining $< i, i >$ is $\left(\frac{\phi_i}{\phi_i+\phi_j}\right)^2$, of $< i, j >$ is $2\left(\frac{\phi_i\phi_j}{\phi_i+\phi_j}\right)$ and of $< j, j >$ is $\left(\frac{\phi_j}{\phi_i+\phi_j}\right)^2$. During crossover the population $< i, j >$ with corresponding bit strings $i = b_1b_2b_3$ and $j = B_1B_2B_3$ will become

$$b_1B_2B_3 \quad \text{and} \quad B_1b_2b_3 \quad \text{with probability } \frac{1}{2}$$

or

$$b_1b_2B_3 \quad \text{and} \quad B_1B_2b_3 \quad \text{with probability } \frac{1}{2}.$$

Finally, under mutation, the population $< i = (b_1b_2b_3)_2,\ j = (B_1B_2B_3)_2 >$ will become, with prime denoting bit complementation,

$$b_1b_2b_3 \quad \text{and} \quad B_1B_2B_3 \quad \text{with probability } (1-p_m)^3$$

or

$$b_1b_2b_3' \quad \text{and} \quad B_1B_2B_3 \quad \text{with probability} \frac{1}{2}(1-p_m)^2p_m$$

and so on until

$$b_1b_2b_3 \quad \text{and} \quad B_1'B_2'B_3' \quad \text{with probability } \frac{1}{2}p_m^3.$$

The 36×36 overall transition probability matrix P is the product of its three component parts reproduction, crossover and mutation by the independence of these processes, and works out to be

$$P = P_RP_cP_m = \begin{bmatrix} .729 & .081 & \dots & .000 \\ .425 & .324 & \dots & .000 \\ \vdots & \vdots & & \vdots \\ .000 & .000 & \dots & .729 \end{bmatrix}.$$

Note that each of the 8 populations $< 0, 0 >, \dots, < 0, 7 >$ containing 0 solve the problem. Therefore the deleted transition matrix $\hat{P}$ is 28×28 and omits the first 8 rows and columns of P.

As in the first example we may calculate the optimization characteristics from the transition probability matrix. These data are shown in Table 2. Note that the theoretically predicted speedup of over 11 for 8 processors was in fact bourne out experimentally.

5.2.1 Remark

This example affords a simple explanation as to why superlinear speed-up is possible. There is a certain (relatively large) probability p that the process will be in the sub-optimal state $x = 7$. (Here $p = .185$ and is the last component of the normalized left eigenvector ω for $\hat{P}$.) By contrast the probability q that the process will be in state $x = 1$, the threshold of the basin for the solution, is small ($q = .029$, the first component of ω). However for m independent processes, the probability they *all* will be in state $x = 7$ is p^m – small for large m, while the probability that *at least one* of the m processes is in state $x = 1$ increases with m, $1-(1-q)^m$. But it only takes one process to find the solution.

Table 2

Sandia Mountain Problem		
$\lambda = .975624$	$s = 1.1488966$	-
$E = 36.23$	$\hat{E} = 37.76$	-
$E_2 = 16.58$	$\hat{E}_2 = 16.61$	$SU_2 = 2.19$
$E_4 = 7.30$	$\hat{E}_4 = 7.10$	$SU_4 = 4.96$
$E_8 = 3.22$	$\hat{E}_8 = 3.20$	$SU_8 = 11.25$

5.3 THE INVERSE FRACTAL PROBLEM

A description appears in detail in (Shonkwiler, Mendivil, Deliu 1991), here we give an abbreviated description and the results.

5.3.1 Fractal Sets Generated by Iterated Function Systems

Let $\mathcal{W} = \{w_1, w_2, \dots, w_n\}$ be a finite set of affine maps of the unit interval $I = [0, 1]$ into itself, that is maps of the form

$$w(x) = sx + a, \qquad 0 \le x \le 1.$$

Here the parameter s is the scale factor and the parameter a is the translation. Alternatively, putting $l = a$ and $r = s + a$, then

$$w(x) = l(1-x) + rx. \tag{5.3.1}$$

In this form the unit interval is seen to map into the interval between l and r and so $0 \le l, r \le 1$. We impose the condition that the image set $w(I)$ be a strict subset of I, i.e. that w be a contraction map and $|s| < 1$. In this case such a map w has a fixed point in I.

Associated with every such collection $\mathcal{W}$ is the *attractor* A, that is the unique subset $A \subset I$ characterized by the selfcovering property that

$$A = \bigcup_{i=1}^{n} w_i(A). \tag{5.3.2}$$

For the proof of the existence and uniquness of an attractor which satisfies (5.3.2) see (Barnsley 1988). It is however easy to obtain points in A. Obviously the fixed point x_i^* of each map w_i in $\mathcal{W}$ belongs to A by (5.3.2). Further this equation provides that if $x \in A$ then also $w_i(x) \in A$ for each $i = 1, 2, \ldots, n$. It follows that for every composition $f = w_{i_1} w_{i_2} \cdots w_{i_k}$, where $i_j \in \{1, \ldots, n\}$, for all $j = 1, \ldots, k$, if $x \in A$ then also $f(x) \in A$. It is in this way that $\mathcal{W}$ is an iterated function system (IFS).

Moreover the observation above provides a method to visualize an attractor on a computer screen. Starting from the fixed point x^*, say of map w_1, choose $i_1 \in \{1, 2, \ldots, n\}$ at random and plot $x_1 = w_{i_1}(x^*)$. (To make the image easier to see, plot a short vertical line at x_1.) Now repeat this step with x_1 in place of x^* and $x_2 = w_{i_2}(x_1)$ in place of x_1, then repeat with x_2 in place of x_1, e.t.c. until say 10,000 points have been plotted. This construction is known as the *Random Iteration Algorithm* for constructing the attractor.

A difficulty is that the mapping $\mathcal{W} \to A_{\mathcal{W}}$, which assigns to an IFS W its attractor $A_{\mathcal{W}}$, is not one-to-one. As an example the Cantor set is the attractor of the IFS

$$w_1(x) = \frac{1}{3}x, \qquad w_2(x) = \frac{1}{3}x + \frac{2}{3},$$

as well as the attractor of the distinct IFS

$$w_1'(x) = -\frac{1}{3}x + \frac{1}{3}, \qquad w_2'(x) = \frac{1}{3}x + \frac{2}{3}.$$

Typically the attractor of an IFS is a fractal set.

5.3.2 The Inverse Problem

The *inverse attractor problem* consists in finding an IFS $\mathcal{W}$ whose attractor A is given. This is also known as *encoding* an attractor.

To solve the inverse problem we need a notion of closeness or distance between two subsets A and B of I. The *Hausdorff* metric, $h(A, B)$, which is defined as

$$h(A, B) = \sup_{x \in A} \inf_{y \in B} |x - y| + \sup_{y \in B} \inf_{x \in A} |y - x|$$

works nicely.

The level surfaces which result using this distance function is very pathological even for small numbers of maps. The surfaces rise and fall abruptly with small changes in parameter values and have numerous local minima, see (Mantica, Sloan 1989).

5.3.3 Results

A Genetic Algorithm, described in (Shonkwiler, Mendivil, Deliu 1991), was run both on parallel platforms according to the *IIP* scheme and on a single processor computer.

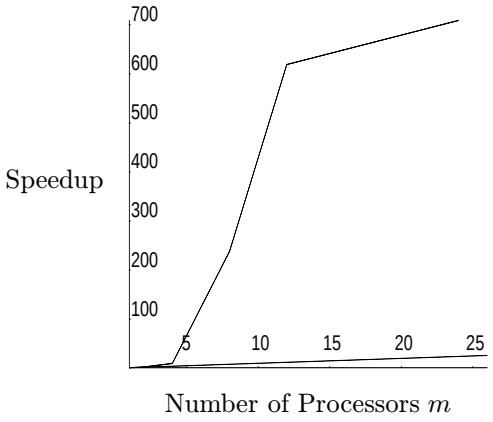

Figure 3: PARALLEL SPEEDUP FOR THE INVERSE FRACTAL PROBLEM

The runs were made on randomly generated IFS attractors discretized to a resolution of 512. Only 2 and 3 map attractors were tested (4 and 6 parameter problems). Despite the small number of maps, the resulting attractors are quite complex and difficult to solve. The run times are on a Sun Sparc workstation and take about 1 hour for 10,000 iterations.

For the parallel algorithm the natural statistic to show is speed-up,

$$\text{speed-up} = \frac{\text{time for a single process}}{\text{time for the parallel process}}$$

However for a stochastic algorithm there is an inherent difficulty. For any time T however large (expressed in iterations until solution say), there is a non-zero probability the algorithm will take T time to finish. So it is that in 24 runs on the problem above, 10 required an unknown number of iterations exceeding 100,000 in order to solve it. A solution was defined to be the achievement of a Hausdorff distance of less than 500.

Nevertheless by using the 100,000 value for these runs, we obtain an estimate of the expected run time which is therefore understated, thus our calculated speed-ups are under estimates of the actual ones.

The observed speed-up, shown in fig. 3, is on the order of 600 to 700 for 10 to 20 processes. The runs were made on a LAN of Sun Sparc stations. The straight line in this figure just above the horizontal axis is linear speedup. As shown in (Shonkwiler, Van Vleck 1993), such large speedups are possible for a "deceptive" problem where the time to reach the goal can

be infinite. This is precisely the case where *IIP* parallel excels as the independent processors avoid being dragged to the same sub-optimum point.

6 CONCLUSIONS

We have shown that superlinear speed-up is possible with these types of algorithms. A given Monte Carlo method is characterized by its deleted transition probability matrix $\hat{P}$ and in particular its hitting time expectation depends on the complementary hitting time distribution. Two parameters, the principle eigenvalue λ of $\hat{P}$, and the s-factor, completely describe the tail of the hitting time distribution; asymptotically

$$\Pr(\theta \geq k) \approx s^{-1}\lambda^{k-1}.$$

The complementary hitting time distribution can be used to rigorously compare two Genetic Algorithms.

Finally one intriguing consequence of a superlinear parallel algorithm is the possibility of a new single process algorithm. In this case it is the running of multiple processes on a single processor machine. This can be simply achieved. One merely sets up the data structures for several "colonies". Then one loops through these colonies processing each in turn according to a single process GA. We refer to this as *in code parallel*.

References

Akyildiz, I., Shonkwiler, R., "Simulated Annealing for Throughput Optimization in Communication Networks with Window Flow Control," *IEEE-ICC Conference Proceedings*, Vol. 3 (1990), 728-738.

Barnsley, M.F., *Fractals Everywhere*, Academic Press, NY, (1988).

Carlson, S., Ingrim, M., and Shonkwiler, R., "A Comparative Evalution of Search Methods Applied to Catalog Selection," *Proceedings of the International Conference on Design of the SME*, (1993).

Ghannadian, F., Shonkwiler, R., and Alford, C., "Parallel Simulated Annealing for the n-Queen's Problem," *Proceedings of the 7th International Parallel Processing Symposium of the IEEE*, Newport Beach (1993).

Goldberg, D., "Genetic Algorithms in Search, Optimization and Machine Learning", *Addison-Wesley*, Reading, Mass. (1989).

Holland, J., "Adaptation in Natural and Artificial Systems", *Univ. of Michigan Press*, Ann Arbor, MI (1975).

Isaacson, D., and Madsen, R., "Markov Chains Theory and Applications", *Krieger Pub. Co.*, Malabar, FL (1976).

Mantica, G., and Sloan, A., "Chaotic optimization and the construction of fractals: solution of an inverse problem", *Complex Systems*, **3**, (1989) 37-62.

Miller, K., and Shonkwiler, R., "Genetic Algorithm/Neural Network Synergy For Nonlinearly Constrained Optimization Problems," *Proceedings of the 1992 International Joint Conf. on Neural Networks*, Baltimore (1992)

Omiecinskii, E., Shonkwiler, R., "Parallel Join Processing Using Nonclusterded Indexes for a Shared Memory Multiprocessor," *Proceedings of the IEEE Symposium on Parallel and Distributed Processing*, (1990).

Seneta, E., "Non-negative Matrices and Markov Chains", *Springer-Verlag*, New York (1981).

Shonkwiler, R., and Van Vleck, E., "Parallel speed-up of Monte Carlo methods for global optimization", to appear in *Journal of Complexity*, (1993).

Shonkwiler, R., Mendivil, F. and Deliu, A., "Genetic Algorithms for the 1-D Fractal Inverse Problem", *Proceedings of the Fourth International Conference on Genetic Algorithms*, edited by Belew, R. and Booker, L., Morgan Kaufmann, San Mateo, CA, (1991) 495-501.

Varga, *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ (1963).

Vrscay, E. R., "Moment and collage methods for the inverse problem of fractal construction with Iterated Function Systems", in *Fractals in the Fundamental and Applied Sciences*, Peitgen, H.-O., Henriques, J. M., and Penedo, L.F., Editors, Elsevier, (1991).