An Analysis of some Stochastic Optimization Methods

by

Neil J. Calkin, R. Shonkwiler and M. C. Spruill
Georgia Institute of Technology
Atlanta, GA 30332
shonkwiler@math.gatech.edu, www.math.gatech.edu/ shenk

Given a real-valued function C, the *objective*, defined on a set of points, the problem is to find the optimal function value, maximum or minimum, along with a point where it occurs. These are the *optimum* and *optimizer* respectively. Just to be definite, assume here the objective is such that we want to minimize it, as for example a cost function. The points are often points in the usual sense such as n-tuples in some Cartesian product space; but they can also be abstract as for example the tours in a Traveling Salesman problem. Collectively we refer to them as the *domain* or *solution space* and we denote their number by N. In this work we will be interested in estimating how long it takes to find an optimizer. Since our algorithms are stochastic, we take this to be the expected running time E of the solution method.

Generally, optimization methods are iterative and successively approximate the minimum cost although progress is not always monotonic. The various solution methods differ in the way iterations are conducted. Those methods for which each new approximation depends only on the present one in the iteration loop, can be analyzed as a Markov Chain. Most methods are of this type, including, for example, Genetic Algorithms.

Graph Theoretical Formulation

Search methods can also be modeled graph theoretically. We may take the points of (a finite) solution space to be the vertices of a graph. Every search process has associated with it, either explicitly or implicitly, a neighborhood system. The *neighborhood* of a point is the set of all other points to which the process can move in one iteration. The edges of the graph join a vertex to its neighbors.

Retention and Acceleration

As mentioned above, when the successive solutions depend only on the previous solution, the process is a (homogeneous) Markov Chain and is described by a *transition matrix* P,

$$P = (p_{ij}).$$

In this, p_{ij} is the probability of a transition from solution i to solution j on any given iteration. Let α_t, the *state vector*, denote the probability distribution over the solution space for the whereabouts of the algorithm on iteration t and let α_0 denote the same for the starting solution. Thus if the starting solution is chosen equally likely, α_0 will be the row vector all of whose components are $1/N$. The successive states of the algorithm are given by the matrix product

$$\alpha_t = \alpha_{t-1} P$$

and hence

$$\alpha_t = \alpha_0 P^t.$$

It is well-known that the expected hitting time E can be calculated as follows. Let $\hat{P}$ denote the matrix which results from P when the rows and columns corresponding to the optimizers are deleted and $\hat{\alpha}_t$ the vector that remains after deleting the same columns of α_t. Then the expected hitting time is given by

$$E = \hat{\alpha}_0 (I - \hat{P})^{-1} \mathtt{1} \tag{1}$$

where $\mathtt{1}$ is the column vector of 1's.

Equation (1) may be re-written as the Neumann series

$$E = \hat{\alpha}_0 (I + \hat{P} + \hat{P}^2 + \hat{P}^3 + \ldots)\mathtt{1}.$$

By the Perron-Frobenius theorem,

$$\hat{P}^k \approx \lambda^k \chi \omega \qquad \text{as} \quad k \to \infty$$

where χ is the right and ω the left eigenvectors for the principle eigenvalue λ of $\hat{P}$. The eigenvectors may be normalized so that $\omega \mathtt{1} = 1$ and $\omega \chi = 1$. With these substitutions the equation for E becomes

$$\begin{aligned} E &\approx \frac{1}{s}(1 + \lambda + \lambda^2 + \ldots) \\ &= \frac{1}{s}\frac{1}{1-\lambda} \end{aligned}$$

where $1/s = \hat{\alpha}_0 \chi$. We therefore arrive at the result that two scalar parameters govern the convergence of the process, *retention* λ and *acceleration* s. In most applications λ is just slightly less than 1 and s is just slightly more than 1. See reference [2].

Theorem 1. *The convergence rate of a homogeneous Markov Chain is geometric, i.e.*

$$Pr(\textit{an optimizer is not found by } k\textit{th iteration}) \leq \frac{1}{s}\lambda^k,$$

and the expected number of iterations needed to find an optimizer is approximately given by

$$E = \frac{1}{s}\frac{1}{1-\lambda}.$$

Note that this result continues to hold for any collection of points in place of the set of optimizers. Thus one might define a goal to be any point x for which $C(x) \leq c$ for some target cost c. The theorem holds for the these points x.

In some cases it is possible to estimate retention and acceleration. One way is from an empirical graph of the complementary hitting distribution $chd(\cdot)$ defined by

$$\begin{aligned} chd(t) &= \Pr(\text{hitting time} \geq t), \quad t = 0, 1, \ldots \\ &\approx \frac{1}{s}\lambda^{t-1}. \end{aligned}$$

Plotting $\log(chd)$ vs $t-1$ gives, asymptotically, a straight line whose slope is λ and whose intercept is $-\log s$. While this technique is adequate for retention, there is a better way to estimate acceleration, and that is by parallel execution speedup.

IIP parallel search

Now consider running m copies of a given algorithm in parallel, independently of each other; we call this independent identical processes parallel execution. By independence, their expected hitting time $E(m)$ is given by

$$\begin{aligned} E(m) &= \hat{\alpha}_0 (I - \hat{P}^m)^{-1} \mathtt{1} \\ &= \hat{\alpha}_0 (I + \hat{P}^m + (\hat{P}^m)^2 + (\hat{P}^m)^3 + \ldots)\mathtt{1}. \\ &\approx \frac{1}{s^m} \frac{1}{1-\lambda^m}. \end{aligned}$$

If we define *speedup* $SU(m)$ to be relative to the single-processor running time, we find

$$\begin{aligned} SU(m) = \frac{E(1)}{E(m)} &= s^{m-1} \frac{1-\lambda^m}{1-\lambda} \\ &\approx s^{m-1} m \end{aligned}$$

for λ near 1. These results show that IIP parallel is an effective technique when $s > 1$ accelerating convergence superlinearly. See reference [2].

Iterative Improvement/Tabu Search

An *iterative improvement algorithm* is one in which the successive approximations are monotically decreasing. It is a deterministic process, if run twice starting from the same initial point, the same sequence of steps will occur. Eventually the alogrithm reachs a local minimum relative to its neighborhood system and no additional improvement is possible. When this occurs the algorithm must stop.

By its nature, an iterative improvement algorithm partitions the solution space into basins. A *basin* being all those points leading to the same local minimum. Graph theoretically, an iterative improvement alogrithm is a forest of trees. Each tree corresponds to a basin, the root of the tree is the local minimum of the basin.

For problems having a differentiable objective function, the gradient is generally used to compute downhill steps required for improvement. In discrete problems, one has a "candidate neighborhood system," that is, each point has a neighborhood of candidates. Candidates are examined until one is found which improves the objective value and that one becomes the next iteration point. Various heuristics are used for assigning a candidate neighborhood. For example, when the domain is a cartesian space of some sort, neighbors can be the one-coordinate perturbations of the present point.

Tabu search is a modification of iterative improvement to deal with the problem of premature fixation in local minima. A short history of visited points is retained by the algorithm. When no improvement is possible among the candidate neighbors, then backtracking is allowed. Thus Tabu Search is not a Markov Chain process.

Iterative Improvement with Random Restart

Another method for dealing with premature fixation is *restart*. We envision a process combining a deterministic downhill operator g, acting on points of the solution space, and a uniform random selection operator U. The process starts with an invocation of U resulting in a randomly selected *starting* point. This is followed by repeated invocations of g until a local minimum is reached. Then the process is restarted with another invocation of U and so on.

As above, this process enforces a topology on the domain which is a forest of trees. The domain is partitioned into *basins* B_i, $i = 0, 1, \ldots$ as determined by the equivalence relation $x \equiv y$ if and only if $g^k(x) = g^j(y)$ for some k, j. The *settling point* or *local minimum* b of basin B is $\lim_{k\to\infty} g^k(x)$ where x is any point of B. By the *depth* of a tree we mean its maximum path length.

The transition matrix for such a process assumes the following form

$$P = \begin{bmatrix} B_0 & 0 & \ldots & 0 \\ Q & B_1 & \ldots & Q \\ \vdots & \vdots & \ddots & \vdots \\ Q & Q & \ldots & B_n \end{bmatrix}.$$

By overload of notation, we also use B_i to denote the matrix corresponding to basin B_i. Each sub-matrix B_i is of the form

$$B_i = \begin{bmatrix} p & p & p & \ldots & p \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{bmatrix},$$

the 1's are in the lower triangle but not necessarily on the sub-diagonal. The blocks designated by Q are generic for the form

$$Q = \begin{bmatrix} p & p & \ldots & p \\ 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \end{bmatrix}.$$

Let E denote the expected hitting time to the basin B_0 containing a minimizer, the *goal basin*, and let G denote the same thing for the set of global optima themselves, Let T_i be the expected time to reach the settling point of basin B_i. Let $|B_i|$ denote the number of points in basin B_i and θ_i the ratio $|B_i|/N$ where $N = \sum |B_i|$, i.e. θ_i is the probability of landing in basin B_i on a restart. Then by decomposition of events

$$G = (1 + T_0)\theta_0 + (1 + T_1 + G)\theta_1 + \ldots + (1 + T_n + G)\theta_n$$

or

$$G = \frac{1}{\theta_0}\left(1 + \sum_{i=0}^{n} T_i\theta_i\right).$$

And

$$E = \theta_0 + (1 + T_1 + E)\theta_1 + \ldots + (1 + T_n + E)\theta_n \tag{2}$$

or

$$E = \frac{1}{\theta_0}\left(1 + \sum_{i=1}^{n} T_i\theta_i\right). \tag{3}$$

As above E is also asymptotically given by

$$E = \frac{1}{s}\frac{1}{1-\lambda}.$$

Because of the special structure of P in this case, both retention and acceleration can be calculated directly.

Solving for λ and s, the Fundamental Polynomial

In the forest of trees model, it is clear that all states which are a given number of steps from a settling point are equivalent as far as the algorithm is concerned. For example, equation (3) may be rederived. Let $r_j(i)$ be the number of vertices j steps from the local minimizer of basin i and let $r_j = \sum_{i=1}^{n} r_j(i)$ denote the total number of vertices which are j steps from a local minimizer. In particular, $r_0 = n$ is the number of local minimizers.

Now T_i, the expected time to reach the local minimizer of basin i given that basin i has been chosen for restart, can be calculated directly

$$T_i = 1\frac{r_1(i)}{|B_i|} + 2\frac{r_2(i)}{|B_i|} + \ldots,$$

and so on up to the depth of basin i. Substituting this into (2) we get

$$\begin{aligned} E &= \theta_0 + \left(1 + E + 1\frac{r_1(1)}{|B_1|} + 2\frac{r_2(1)}{|B_1|} + \ldots\right)\frac{|B_1|}{N} + \\ &\qquad \ldots + \left(1 + E + 1\frac{r_1(n)}{|B_n|} + 2\frac{r_2(n)}{|B_n|} + \ldots\right)\frac{|B_n|}{N} \\ &= \theta_0 + (1+E)\frac{|B_1| + |B_2| + \ldots}{N} + 1\frac{r_1(1) + \ldots + r_1(n)}{N} + 2\frac{r_2(1) + \ldots + r_2(n)}{N} + \ldots. \end{aligned}$$

Now the sum $|B_1| + |B_2| + \ldots$ is just the number of non-goal basin vertices as is the sum $r_0 + r_1 + \ldots$. Therefore we may continue

$$\begin{aligned} E &= \theta_0 + (1+E)\frac{r_0 + r_1 + \ldots}{N} + 1\frac{r_1}{N} + 2\frac{r_2}{N} + \ldots \\ &= \theta_0 + (1+E)\frac{r_0}{N} + (2+E)\frac{r_1}{N} + (3+E)\frac{r_2}{N} + \ldots. \end{aligned}$$

This is exactly the direct calculation of E.

Therefore the given forest of trees model in which each vertex counts 1 is equivalent to a single, linear tree in which each vertex counts equal to the number of vertices in the original forest which are at that distance from a settling point.

Under the equivalency, the $\hat{P}$ matrix becomes

$$\hat{P} = \begin{bmatrix} p_0 & p_1 & p_2 & \cdots & p_{n-1} & p_n \\ 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 \end{bmatrix}. \tag{4}$$

In this, $p_i = r_i/N$ where, as above, N is the cardinality of the domain. It is easy to calculate the characteristic polynomial of this matrix directly, expand $\det(\hat{P} - \lambda I)$ by minors along the first row,

$$-\lambda^{n+1} + p_0\lambda^n + p_1\lambda^{n-1} + \ldots + p_{n-1}\lambda + p_n.$$

Upon setting $\eta = 1/\lambda$ we get a polynomial we will refer to as the *fundamental polynomial*

$$f(\eta) = p_0\eta + p_1\eta^2 + \ldots + p_{n-1}\eta^n + p_n\eta^{n+1} - 1. \tag{5}$$

Notice that the degree of the fundamental polynomial is equal to the depth of the deepest basin.

As above, letting θ_0 be the probability of landing in the goal basin, then

$$\theta_0 + p_0 + p_1 + \ldots + p_n = 1.$$

Note that $f(1) = -\theta_0$ and

$$f'(\eta) = p_0 + 2p_1\eta + 3p_2\eta^2 + \ldots + np_{n-1}\eta^{n-1} + (n+1)p_n\eta^n.$$

This is positive for all $\eta \geq 0$. Hence the unique greater than 1 root of f, denote it η, is the reciprocal of the Perron-Frobenius eigenvalue λ.

The right Perron-Frobenius eigenvector, χ, of $\hat{P}$ is easily calculated. From (4) we get the recursion equations

$$\begin{aligned} \chi_0 &= \lambda\chi_1 \\ \chi_1 &= \lambda\chi_2 \\ \vdots &= \vdots \\ \chi_{n-1} &= \lambda\chi_n \end{aligned}$$

From this we find

$$\chi_k = \eta^k\chi_0, \qquad k = 1, \ldots, n.$$

Similarly, from (4)we get the following recursion equations for the components of the left Perron-Frobenius eigenvector, ω,

$$\begin{aligned} \omega_0 p_0 + \omega_1 &= \lambda\omega_0 \\ \omega_0 p_1 + \omega_2 &= \lambda\omega_1 \\ \vdots + \vdots &= \vdots \\ \omega_0 p_{n-1} + \omega_n &= \lambda\omega_{n-1} \\ \omega_0 p_n &= \lambda\omega_n \end{aligned} \tag{6}$$

From these we get equations for the components in terms of ω_0;

$$\begin{aligned}
\omega_n &= \omega_0 \eta p_n \\
\omega_{n-1} &= \omega_0(\eta p_{n-1} + \eta^2 p_n) \\
\vdots &= \vdots \\
\omega_1 &= \omega_0(\eta p_1 + \eta^2 p_2 + \ldots + \eta^n p_n)
\end{aligned}$$

Actually, all we need here is the sum of the (omega-sys) system of equations; recall that by normalization, $\sum \omega_i = 1$,

$$\begin{aligned}
\omega_0 \sum p_i + \sum_i^n \omega_i &= 1/\eta \\
\omega_0(1-\theta_0) + 1 - \omega_0 &= 1/\eta
\end{aligned}$$

Solve for ω_0 to get

$$\omega_0 = \frac{\eta - 1}{\eta \theta_0}. \tag{7}$$

Further recall that under the normalization, $\sum \omega_i \chi_i = 1$; hence, using (root),

$$\begin{aligned}
\frac{1}{\omega_0 \chi_0} &= 1 + \eta^2 p_1 + 2\eta^3 p_2 + \ldots + n\eta^{n+1} p_n \\
\frac{1}{\omega_0 \chi_0} &= \eta p_0 + 2\eta^2 p_1 + 3\eta^3 p_2 + \ldots + (n+1)\eta^{n+1} p_n \\
\frac{1}{\omega_0 \chi_0 \eta} &= p_0 + 2\eta p_1 + 3\eta^2 p_2 + \ldots + (n+1)\eta^n p_n \\
\frac{1}{\omega_0 \chi_0 \eta} &= f'(\eta)
\end{aligned}$$

From this we get that

$$\chi_0 = \frac{1}{\omega_0 \eta f'(\eta)}.$$

Finally we calculate $s = 1/(\chi \cdot \hat{\alpha}_0)$ where $\hat{\alpha}_0$ is the non-goal partition vector of the starting distribution; thus

$$\hat{\alpha}_0 = (\, p_0 \quad p_1 \quad \ldots \quad p_n \,).$$

Therefore

$$\begin{aligned}
\frac{1}{s} &= p_0\chi_0 + p_1\chi_1 + \ldots + p_n\chi_n \\
&= p_0\chi_0 + p_1\eta\chi_0 + p_2\eta^2\chi_0 + \ldots + p_n\eta^n\chi_0 \\
&= \frac{\chi_0}{\eta} = \frac{1}{\eta^2 \omega_0 f'(\eta)}
\end{aligned}$$

So

$$s = \frac{\eta(\eta - 1)f'(\eta)}{\theta_0}.$$

Run time estimation of retention, acceleration and hitting time

Returning to the fundamental polynomial, we notice that its coefficients are the various probabilities for restarting a given distance from a local minimum. Thus the linear coefficient is the probability of restarting on a local minimum, the quadratic coefficient is the probability of restarting one iteration from a local minimum and so on.

As a result, it is possible to estimate the fundamental polynomial during a run by keeping track of the number of iterations spent in the downhill processes. Using the estimate of the fundamental polynomial, estimates of retention and acceleration and hence also expected hitting time can be affected. As a run proceeds, the coefficient estimates converge to their right values and so does the estimate of E.

Random Restart with Stochastic Basin Search

In an effort to adapt the analysis of the previous section to genetic algorithms, we now allow stochastic intra-basin processes. Within each basin there can operate a sub-Markov processes, which eventually reaches the settling point; then a uniform restart occurs.

The transition matrix for the process is as before

$$P = \begin{bmatrix} B_0 & 0 & \dots & 0 \\ Q & B_1 & \dots & Q \\ \vdots & \vdots & \ddots & \vdots \\ Q & Q & \dots & B_n \end{bmatrix}$$

except that each matrix B_i is now the probability transition matrix of an absorbing chain. The blocks designated by Q are exactly as before. For the calculation of expected hitting times we may use equation (3) since we again have a basin topological structure. This equation requires the basin hitting times T_i and, of course, these times depend on the details of the algorithm.

Suppose the sub-Markov process is simply U, equally likely selection among the $|B_i|$ members of B_i. Graph theoretically this is a clique on $|B_i|$ vertices. In this case it is easy to compute T_i;

$$T_i = |B_i| \qquad \text{for uniform selection within basin } B_i.$$

Below we show how to calculate the $|B_i|$.

More generally, it is known that if all vertices have the same out degree, then the basin hitting time is, at most, $O(|B_i|^2)$.

Shuffle GA, Restriction GA

We define two genetic algorithms to which the basin analysis applies.

Shuffle GA Consider a GA as follows: after the initial population is chosen, an iteration consists of pairing off all population members followed by a "cross-over" for each pair. In particular, there is no mutation and there is no selection. Instead, the progress of the best performer is tracked over iterations. When this indicator no longer improves, the population is declared to be "niched". At this point the entire population is thrown out and a completely new randomly selected population is chosen to begin the process again.

Restriction GA Upon selection of the initial population, a exact duplicate is created from which mutant alleles will be chosen. Here the population itself is modified in the usual manner for a genetic algorithm, via mutation, cross-over and roulette wheel selection. The only stipulation being that mutant alleles are restricted to come from the duplicate population which remains fixed up until a restart. In this way, none of the original alleles are lost as they might be through roulette wheel selection. As above, when the population has niched, it and the duplicate are discarded and the process is restarted.

Basin Analysis

Since the initially selected population freezes the set of alleles in both these genetic algorithms, population space is partitioned into basins as we describe next. Therefore these genetic algorithms are examples of restart with stochastic basin search.

The goal basin B_0 consists of those gene pools which contain the globally optimal set of alleles possibly scattered among different population members. For example, consider a Cartesian product solution space consisting of n-tuples, i.e. "chromosomes" of string length n, and assume the "alleles" for each coordinate are 0 or 1. Assume the population size is Z. There are 2^n distinct chromosomes running from all n bits 0 through all n bits 1. We can identify the chromosomes as the set of integers 0 through $2^n - 1$, and this set is the domain of the objective function. If population size were 1 this would also be the solution space. However, since solutions in a genetic algorithm are populations, the size of the GA solution space is

$$N = \binom{2^n + Z - 1}{Z}.$$

Using Stirling's approximation for $n! \approx \sqrt{2\pi n} n^n e^{-n}$, we obtain an asymptotic formula for N,

$$N \approx 2^{nZ}/Z!.$$

Finally assume the minimizer consists of exactly one chromosome. It is of no loss of generality to assume that it is the chromosome having all 0 bits. Any population containing this chromosome is considered to be a globally optimal population. The number of such populations can be counted in the same manner as N above, in addition to the 0 chromosome, such a population may have any other $Z - 1$ chromosomes; thus

$$\begin{aligned} \text{populations containing the all 0 bits chromosome} &= \binom{2^n - Z - 2}{Z - 1} \\ &\approx 2^{n(Z-1)}/(Z-1)! \end{aligned} \tag{8}$$

asymptotically. Therefore the fraction of populations containing the optimizer outright is

$$\text{fraction of pop. having optimizer} = \frac{Z}{2^n}. \tag{9}$$

However, the goal basin B_0 is much larger than this. Any population such that, for each bit position, some chromosome has a 0 in that position, belongs to the goal basin. The reason is, eventually, in shuffling the alleles among its chromosomes, one will be assembled having the string of all 0 bits, the global optima.

It is possible to calculate the size of this goal basin. One way is by the "inclusion/exclusion" principle. Let p_1 be the property for populations that no chromosome has a 0 in the first bit position. Similarly for p_2, p_3, ..., p_n. Then a gene pool which fails to have a 0 bit in some position is exemplified by the union $p_1 \cup p_2 \cup \ldots p_n$. The number of such populations is given by the sum

$$S_1 - S_2 + S_3 - \ldots \pm S_n$$

where, letting $c(P)$ denote the number of gene pools satisfying property P,

$$\begin{aligned} S_1 &= c(p_1) + c(p_2) + \ldots + c(p_n) \\ S_2 &= c(p_1 \cap p_2) + c(p_1 \cap p_3) + \ldots + c(p_{n-1} \cap p_n) \\ &\vdots = \vdots \\ S_n &= c(p_1 \cap p_2 \cap \ldots p_n), \end{aligned}$$

that is S_2 is the sum of the properties taken two at a time, and so on. This then is the count of populatins not satisfying the goal basin condition so the number we want is the difference from the total number of populations, N. By the same inclusion/exclusion principle, $S_1 = \binom{n}{1}\binom{2^{n-1}+Z-1}{Z}$, $S_2 = \binom{n}{2}\binom{2^{n-2}+Z-1}{Z}$ and so on. Hence

$$\text{size of goal basin} = F(n, Z, 0)$$

where

$$\begin{aligned} F(n,Z,k) = \binom{n}{k}\binom{2^{n-k}+Z-1}{Z} - \binom{n}{k+1}\binom{2^{n-k-1}+Z-1}{Z} \\ + \binom{n}{k+2}\binom{2^{n-k-2}+Z-1}{Z} - \ldots \pm 1. \end{aligned}$$

Again from Stirling's approximation we may derive an asymptotic expression for this,

$$|B_0| \approx (2^Z - 1)^n / Z!.$$

The size of the remaining basins can be similarly calculated keeping careful track of alleles allowed in the various bit positions. The size of even the second basin depends on the exact nature of the objective function and so must be done on a case by case basis. One thing does hold in general, B_0 is the largest basin, see below.

From the asymptotic formulas above,

$$\frac{|B_0|}{N} = \frac{(2^Z-1)^n}{2^{nZ}} = (1-\frac{1}{2^Z})^n \approx e^{-n/2^Z}. \tag{10}$$

Example

Consider a 10 bit problem and a population size of 4. The size of the domain is $2^{10} = 1024$. The size of the population space is $\binom{2^{10}+4-1}{4} = 46,081,900,800$. The number of populations containing $x = 0$ outright is $\binom{2^{10}+3-1}{3} = 179,481,600$, this is nearly 4/1000 of the population space. As above, with no loss of generality, assume the optimizer is $x = 0$. The size of the goal basin is

$$\begin{aligned} F(10,4,0) = & \\ & \binom{1027}{4} - 10\binom{515}{4} + \binom{10}{2}\binom{259}{4} - \binom{10}{3}\binom{131}{4} + \binom{10}{4}\binom{67}{4} - \binom{10}{5}\binom{35}{4} \\ & + \binom{10}{6}\binom{19}{4} - \binom{10}{7}\binom{11}{4} + \binom{10}{8}\binom{7}{4} - 10\binom{5}{4} + 1 \\ & = 24,097,745,486. \end{aligned}$$

Now we calculate the size of the next best basin, B_1 with settling point b_1. We will see that the size of B_1 depends on how many bits b_1 has in common with b_0, the more in common means a smaller B_1.

Assume first that b_1 differs from b_0 in only one bit, for example, $b_1 = 1$ (all zero bits except the last). We can count these in the following way, basically the same as the goal basin above but from those gene pools that have only a 1 bit in the least significant place; evidently, any population that had at least one 0 in each other place would need to have all 1's in the last place or else it would be in the global basin.

$$\begin{aligned} F(9,4,0) = & \\ & \binom{515}{4} - 9\binom{259}{4} + \binom{9}{2}\binom{131}{4} - \binom{9}{3}\binom{67}{4} + \binom{9}{4}\binom{35}{4} - \binom{9}{5}\binom{19}{4} \\ & + \binom{9}{6}\binom{11}{4} - \binom{9}{7}\binom{7}{4} + \binom{9}{8}\binom{5}{4} - 9\binom{5}{4} + 1 \\ & = 1,611,904,064. \end{aligned}$$

Next, for contrast, take $b_1 = 1023$, i.e. all 1 bits and therefore differing as much as possible from b_0. The basin B_1 will consist of all gene pools with at least one 1 in each position but not in the goal basin.

Count those with no 1 in the first position, but not in the goal basin,

$$\begin{aligned} & \binom{2^9+4-1}{4} - \text{no 1 in first position in goal basin} \\ & \binom{2^9+4-1}{4} - \left[\binom{2^9+4-1}{4} - 9\binom{2^8+4-1}{4} + \binom{9}{2}\binom{2^7+4-1}{4} - \ldots - 1\right] \\ & 1,285,082,176. \end{aligned}$$

This is exactly $F(9,4,1)$ or equivalently $\binom{2^9+4-1}{4} - F(9,4,0)$. Similarly count those with no 1 in the second position, then the third and so on. All these will be the same, so this round gives

$$\binom{10}{1} F(9,4,1) = 12,850,821,760.$$

Following the inclusion/exclusion technique, we must now subtract from this those populations that have been doubly counted, that is those with no 1 in both 1st and 2nd positions, and no 1 in both 1st and 3rd, and so on. For those with no 1 in both 1st and 2nd this is

$$\begin{aligned}
&\binom{2^8+4-1}{4} - \text{no 1 in 1st and 2nd positions in goal basin} \\
&\binom{2^8+4-1}{4} - \left[\binom{2^8+4-1}{4} - 8\binom{2^7+4-1}{4} + \binom{8}{2}\binom{2^6+4-1}{4} - \ldots\right] \\
&8\binom{2^7+4-1}{4} - \binom{8}{2}\binom{2^6+4-1}{4} + \ldots + 1 \\
&74,950,059
\end{aligned}$$

or $F(8,4,1)$; equivalently $\binom{2^8+4-1}{4} - F(8,4,0)$. Counting all combinations of pairs gives

$$\binom{10}{2} F(8,4,1) = 3,372,752,655$$

with a total now of 9,478,069,105.

Continuing in this way, we obtain the size of B_1,

$$\begin{aligned}
|B_1| &= \binom{10}{1} F(9,4,1) - \binom{10}{2} F(8,4,1) + \ldots \pm \binom{10}{9} F(1,4,1) \\
&= 9,953,804,132.
\end{aligned}$$

The following table gives the relative sizes of the largest 18 basins.

Table 1
Basin size relative to the population space
for the top 18 basins (population space = 1)

goal basin	0.5229	B_6	0.0150	B_{12}	0.0014
B_1	0.2056	B_7	0.0150	B_{13}	0.0014
B_2	0.0159	B_8	0.0148	B_{14}	0.0013
B_3	0.0159	B_9	0.0148	B_{15}	0.0013
B_4	0.0154	B_{10}	0.0147	B_{16}	0.0013
B_5	0.0154	B_{11}	0.0015	B_{17}	0.0012

The above calculation may be generalized to give the following.

Theorem 2. *The size of the second largest basin is no bigger than*

$$|B_0| - \binom{n}{0}\binom{2^n + Z - 1}{Z} - \sum_{r=1}^{n-1}\binom{n}{r}F(n-r, Z, 0).$$

Hitting time for sub-basin families

When the restart (or initial) population is selected, besides belonging to a specific basin, it also belongs to a sub-basin within that basin. By the *type* of a population we mean the number of target bits in each place of the population. Suppose the population is one that belongs to the goal basin, then it has at least one 0 in each bit position; the type lists exactly how many 0's occur in each place. For example, in a 5-bit problem with population size 4, the type "2,1,4,3,3" means among the 4 chromosomes there are 2 zero bits in the first place, 1 in the second, all bits in the third place are zero and 3 zero bits in both the fourth and fifth places.

Given a type designation, we wish to calculate the *assembly time* by which we mean the expected time needed for the population to assemble the local (or global) minimizer. Of course such a calculation depends on the details of the genetic operators. Our shuffle GA is one of the simplest for this purpose since each iteration merely shuffles the alleles in place among the chromosomes. Even in this case the calculation is difficult unless the population is assumed to be "labelled" meaning the chromosomes are distinguishable other than by genotype. It is a known (cf. the birthday paradox) that for $Z < 2^{n/2}$ the probability of duplicate chromosomes is very small so that the calculation for a labelled population is a good approximation to that of a unlabelled population.

Assume then a labelled population of size Z and of type: $(k_1, k_2, \ldots, k_n)$. We may assume without loss of generality that $k_1 \le k_2 \le \ldots \le k_n$. The total number of such types is

$$\prod_2^n \binom{Z}{k_i}$$

and the number of these having all target bits on the same chromosome is, by inclusion/exclusion,

$$\binom{k_1}{1}\prod_2^n\binom{Z-1}{k_i-1} - \binom{k_1}{2}\prod_2^n\binom{Z-2}{k_i-2} + \ldots \pm \binom{k_1}{k_1}\prod_2^n\binom{Z-k_1}{k_i-k_1}.$$

The probability that a shuffle will assemble to the local minimum is their ratio and the assembly time $T_{k_1,\ldots,k_n}$ is the reciprocal of that

$$T_{k_1,\ldots,k_n} = \frac{\prod_2^n \binom{Z}{k_i}}{\sum_{j=1}^{k_1}\binom{k_1}{j}\prod_{i=2}^n\binom{Z-j}{k_i-j}}.$$

For example, in a 10 bit problem with population size $Z = 6$, the type $t = (3, 3, \ldots, 3)$ can be expected to take

$$T_t = \frac{\binom{6}{3}^9}{3\binom{5}{2}^9 - 3\binom{4}{1}^9 + 1} = 170$$

iterations to assemble all 0's. On the other hand, the type $tt = (1, 1, \ldots, 1)$ will require

$$T_{tt} = 6^9 = 10,077,696$$

iterations on the average to assemble all 0's.

Pruning Level Effectiveness for 10 bit shuffle GA

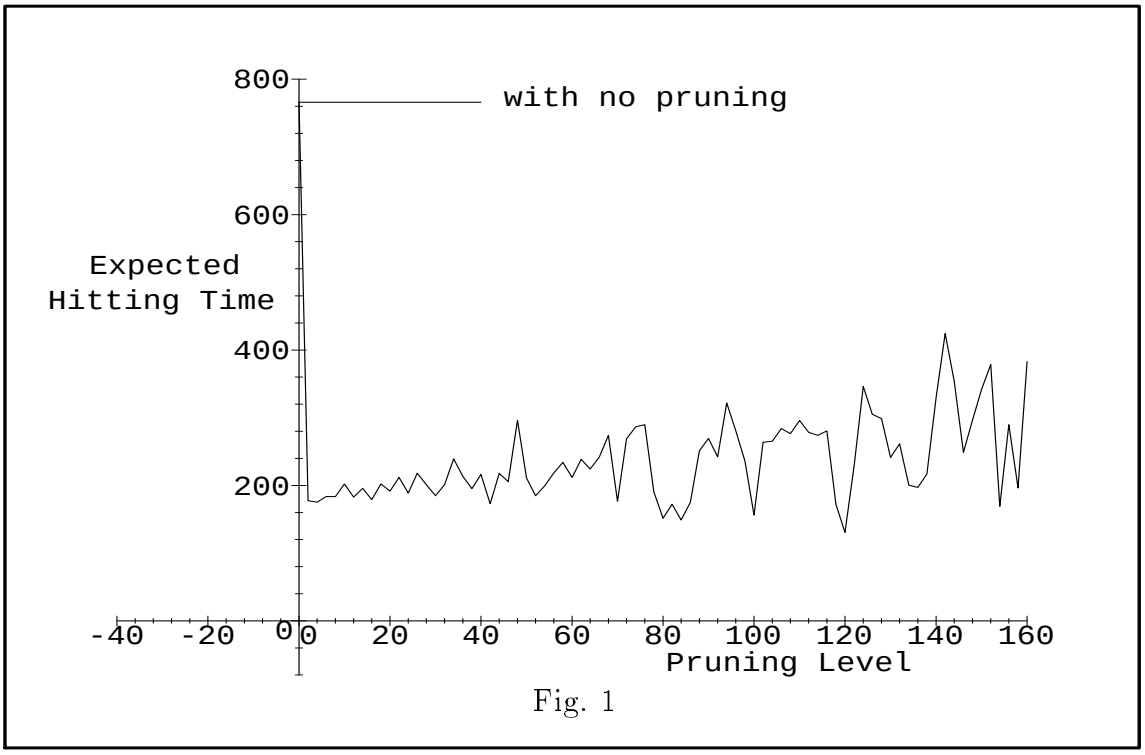

Fig. 1

Pruning

Summarizing and amplifying the above facts, we know that the goal basin is by far the largest basin. The goal basin along with a few others comprise most of the population space. If a restart occurs in a smaller basin, it has a large chance of doing so on that basin's local minimum outright or quickly finding the local minimum. Therefore, the preponderance of the contribution to the expected hitting time due to the small basins is their number rather the time spent in them.

On the other hand, the few large basins are very large and their contribution to the expected hitting time is the rather long search time within them. However, the time spent within one of these basins depends on the sub-basin type of the restart population. There is a vast difference in assembly times among the sub-basin types.

Pruning Level Effectiveness for 10 bit restriction GA

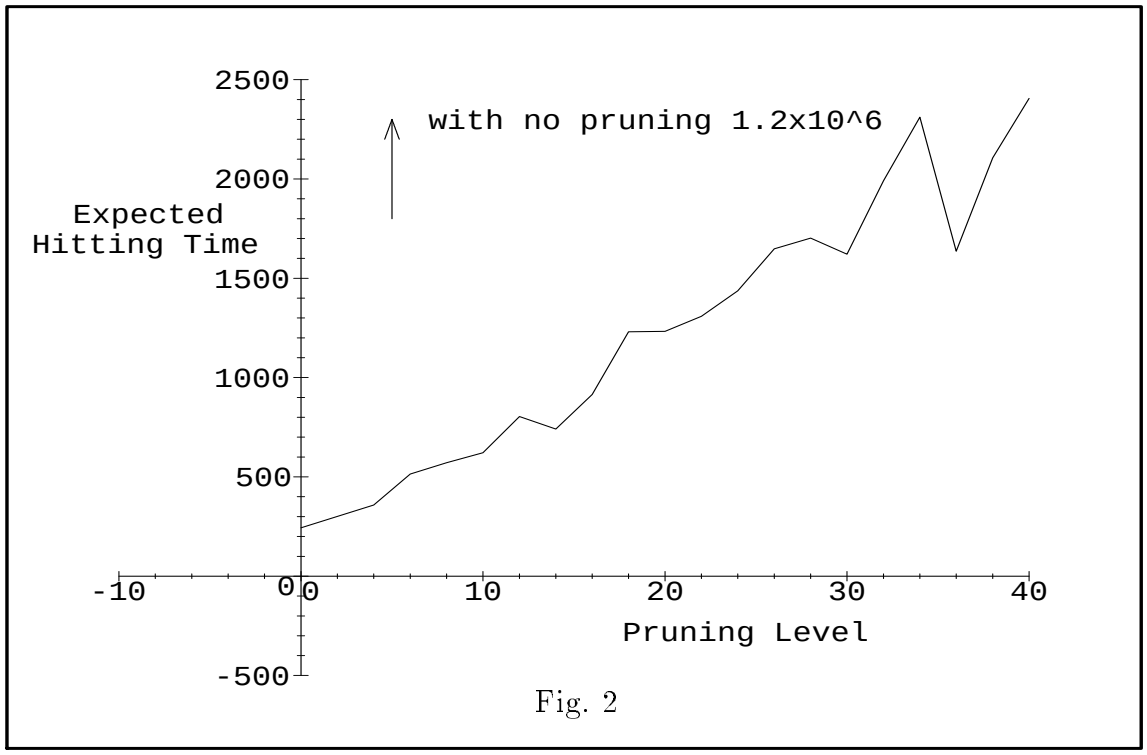

Fig. 2

Pruning Level Effectiveness for 10 bit standard GA [1]

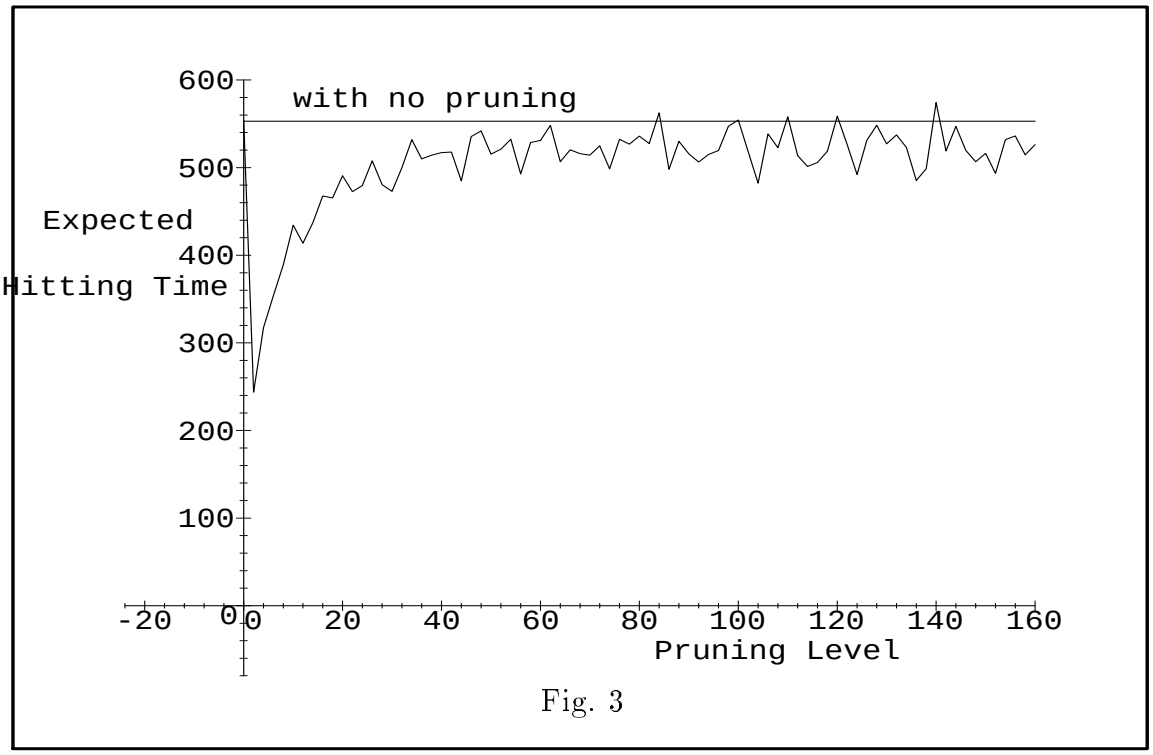

Fig. 3

These facts suggest a strategy: an algorithm should allocate only a fixed number of iteration within a basin, the value of which we refer to as the *pruning level*. If the local minimum is not found by this time, a restart should be initiated anyway.

A possible choice for the pruning level is the "$Z/2$" type, i.e. $t = (Z/2, Z/2, \ldots, Z/2)$. The rationale being that a restart has a $1/2$ probability of generating a population with more target bits per place than this. The matter is still open however.

We close with experimental results on some 20 bit and 10 bit problems.

Multi-modal Test Function
$f(x) = 0.1x + 1 - \cos(\frac{60x}{30+x})$

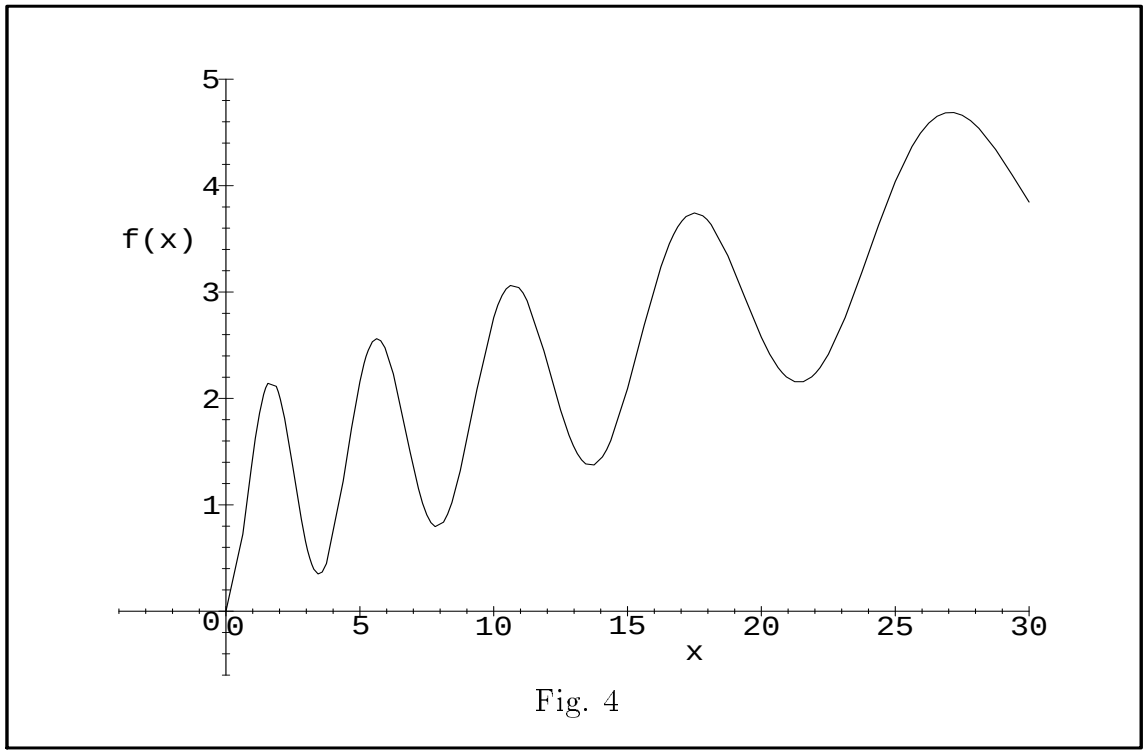

Fig. 4

References

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

[2] Shonkwiler, R., and Van Vleck, E., *Parallel speed-up of Monte Carlo methods for global optimization*, *J. of Complexity* **10**(1994) 64-95

Pruning Level Effectiveness for 20 bit standard GA [1]
(population size: 20, bit mutation rate: 0.001)

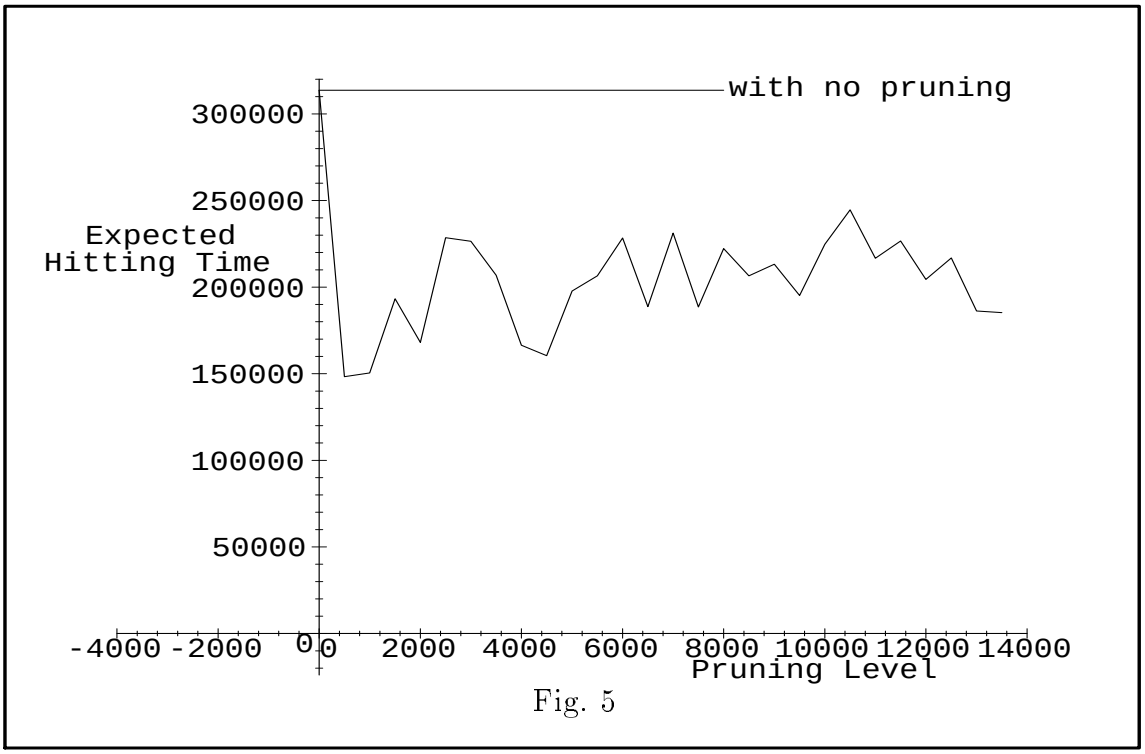

Fig. 5