Parallel Speed-up of Monte Carlo Methods for Global Optimization

by

R. Shonkwiler and Erik Van Vleck

Introduction

In this article we will be concerned with the optimization of a scalar-valued function $u = f(x)$, $x \in D$, defined on some set D. Often D is a subset of n-dimensional Euclidean space $\mathbf{R}^n$. Further we impose the condition that D be a finite set although we do not restrict the size of its cardinality. Thus D might be the set of all n-tuples of computer floating point numbers. In this way our assumption on D is not computationally restrictive. We make no assumption about the smoothness or even the continuity of f since we use only the values u of f. In the event that f has at least first order derivatives, much studied gradient methods efficiently find the optimal local value of f for the basin corresponding to any given starting value x_0. However should f have a large number of local optima, finding a global optimum is then a matter of chance depending on the x_0 selected. And so it is that when f has no smoothness as well as when f has a very large number of local optima, one can benefit from an understanding of how to optimize by means of function values alone.

Two widely known Monte Carlo methods are Simulated Annealing, [8], [9], and Simulated Evolution (or Genetic Algorithms), [3],[5]. The former is meant to model thermodynamic systems and their ability to achieve near minimal energy configurations through Boltzmann kinetics as environmental temperature is lowered. The latter is the effort to find optimal states in an abstract setting in a manner similar to the way biological systems optimize survival using abstract analogs of the mechanisms of mating, mutation and enhanced reproduction of the fittest. Because of the generality and importance of these two methods and because our results apply directly to them, we will briefly describe and compare them below.

An optimization process proceeds in steps indexed by "time" $t = 1, 2, \ldots$. On the t^{th} step one or more domain points x are selected to constitute the current *state* X_t of the process

$$X_t = \{x_1^t, \ldots, x_{n(t)}^t\} \subset D,$$

and any required function evaluations, $u_i^t = f(x_i^t)$, are performed. The sequence of random variables $X_1, X_2, \ldots$ is a stochastic process on the class of all finite subsets of D. Let D_{op}

denote the subset of the domain consisting of the optimal points. If the problem is one of global minimization, then

$$D_{op} = \{x_* \in D : f(x_*) \leq f(x), x \in D\}.$$

Some important criteria identified in the literature for a stochastic optimization method are that: (1) eventually the method should find the global optimum, and (2) moreover the method should identify it. Mathematically this occurs if the probability that $X_t \cap D_{op}$ is not empty as $t \to \infty$ tends to certainty,

$$\lim_{t\to\infty} P_r(X_t \cap D_{op} \neq \{\}) = 1. \tag{0.1}$$

in which case the probability is high that the process has found and is in an optimal state after long run times.

Another important issue for a Monte Carlo method, and one not treated in the literature that we know of, is the expected running time of the algorithm until an optimal state is found. Specifically, by the *hitting time* in D_{op} we mean the random variable θ equal to the first time t so that

$$X_t \cap D_{op} \neq \{\}.$$

In terms of hitting time, a process will eventually find an optimal state if

$$\Pr(\theta < \infty) = 1.$$

The expected hitting time for an algorithm is given by

$$E(\theta) = \sum_{t=1}^{\infty} t\Pr(\theta = t) \tag{0.2}$$

where $\Pr(\theta = t), t = 1, 2, \ldots,$ is the probability density function for θ. Since

$$\Pr(\theta = t) = \Pr(\theta \geq t) - \Pr(\theta \geq t + 1),$$

direct substitution above yields the following alternative equations

$$E(\theta) = \sum_{t=1}^{\infty} \Pr(\theta \geq t) = \sum_{t=1}^{\infty} (1 - F(t)) \tag{0.3}$$

where $F(t) = \Pr(\theta < t)$ is the cumulative distribution function for θ. This all important *complementary hitting time* distribution

$$\{\Pr(\theta \geq t)\}_{t=1}^{\infty} \tag{0.4}$$

characterizes the optimization process and occupies a central place in this work. Evidently

$$0 \leq \Pr(\theta \geq t) \leq 1, \qquad t = 1, 2, \ldots$$

and is a monotone decreasing sequence

$$\Pr(\theta \geq t+1) \leq \Pr(\theta \geq t), \qquad t = 1, 2, \ldots.$$

We will show that for a large class of Monte Carlo methods (namely memoryless processes having stationary transition probabilities, this includes Genetic Algorithms) parallel implementations of the algorithm reduce the expected hitting time for a problem in proportion to the number of processors (approximately). That is to say this class of methods experiences approximately linear speed-up when multi-processed. Even more is true, super linear speed-up is possible. This occurs independently of the number of processors used and does not require inter-process communication. We will see that Monte Carlo algorithms are typically quite simple and do not use large amounts of memory. Therefore it is feasible to implement a Monte Carlo method on parallel computers with a large number of processors, such as the 64K processor Connection Machine, and experience many thousand fold speed-up over the single process algorithm. This is even the case if the individual processes must proceed in lock step or inter-process communication is restricted. Each processor is executing the same code, namely the serial algorithm (of course each process must be seeded differently).

In some cases a Monte Carlo algorithm will have an infinite expected hitting time. This occurs when the sum of the complementary hitting time probabilities diverges. But in this case it is possible for the same algorithm when multi-processed to have *finite* expected hitting time.

Finally, we will show that in the worst case (or best case depending on one's point of view) when complete information about the past iterations is used, one can still expect a speed-up of approximately $m/2$ for m-fold parallelization.

Comparing expected hitting times between a given algorithm and its parallel implementation is but one example of the usefulness of the complementary hitting time distribution. More importantly this tool can rigorously compare any two different Monte Carlo methods for a given problem or class of problems and decide the superior.

We will assume the stochastic process constituting the optimization method to be memoryless, that is for any $t = 2, 3, \ldots$, X_t will depend only on X_{t-1} and not on $X_{t-2}, \ldots, X_1$. Such a process is a Markov chain. This is not a severe assumption in terms of practice. Understandably, the size of most problems prohibits saving a long history of trials. On the other hand, using a short history of trials or none at all to generate the next iteration can fit into the framework of Markov chains by considering Cartesian

product chains if necessary. We will not pursue this further. We note however that both Simulated Annealing and Genetic Algorithms are Markov chain processes.

Since the probabilities governing the selection of the next iterate of a Markov chain depend only on the present iterate, the process defines and is defined by an $N \times N$ matrix of transition probabilities P^t whose ij^{th} entry is the probability

$$p_{ij}^t = \Pr(X_{t+1} = s_j \mid X_t = s_i)$$

where s_k is the k^{th} possible state of the process $k = 1, \ldots, N$. These transition probabilities can vary with time t. If they do the chain is termed *non-stationary*. If they are fixed in time, it's a *stationary* Markov chain.

Throughout this work we shall always assume the Markov chains to be irreducible and aperiodic. The former means that given any two states s_1 and s_2 there is a finite sequence of states $s_{i_0}, s_{i_1}, \ldots, s_{i_k}$ such that $s_{i_0} = s_1$, $s_{i_k} = s_2$ and

$$p_{i_j i_{j+1}}^t > 0, \quad j = 0, \ldots, k-1, \quad t = 1, 2, \ldots.$$

The latter means that for any two states s_1 and s_2, if I is the set of all k (as above) such that there is some sequence of k transitions transforming s_1 to s_2, then the greatest common divisor of I is 1.

A finite irreducible aperiodic Markov chain possesses, for each fixed time t, a unique invariant distribution $\pi^t = (\pi_1^t, \ldots, \pi_N^t)$,

$$\pi^t = \pi^t P^t, \qquad t = 1, 2, \ldots .$$

Simulated Annealing as a Non-Stationary Markov Chain

In simulated annealing the states $X_0, X_1, \ldots$ are singleton sets consisting of a single domain value. The starting state X_0 can be randomly selected or can be the result of a problem based heuristic. Subsequent iterations $X_1, X_2, \ldots$ result from the application of a "generation" process followed by an "acceptance" process. The generation process is itself a Markov process and has an associated generation matrix G^t. Given that the present state at time t is the domain value x_i, then the proposed state at time $t+1$ is x_j with probability g_{ij}^t. Another way of viewing the generation process is in terms of a unary stochastic operator $\mu_t(\cdot) : D \to D$.

Once a state x_j is proposed from state x_i, the acceptance process selects this state to be the next iterate with probability a_{ij}^t,

$$a_{ij}^t = \min\{1, e^{-\Delta u/T}\}, \tag{0.5}$$

where $\Delta u = f(x_j) - f(x_i)$ and temperature T is a variable of the annealing process. According to annealing theory, [4],[2], temperature should depend on time t as

$$T = \frac{C}{\ell n(t+1)} \tag{0.6}$$

where C is equal to (if available) the depth of the deepest local non-global minimum.

Therefore simulated annealing is a non-stationary Markov chain with transition probability matrix

$$P^t = G^t \Delta A^t \tag{0.7}$$

where the matrix operation Δ means

$$P_{ij}^t = \begin{cases} g_{ij}^t a_{ij}^t, & \text{if } i \neq j \\ 1 - \sum_{k \neq i} g_{ik}^t a_{ik}^t & \text{if } i = j \end{cases}$$

In the event that temperature is given according to (0.6) then property (0.1) holds, [1],[2],[4].

Genetic Algorithms as Stationary Markov Chains

Each state X_i of a genetic algorithm is a finite set of domain values called a *population.* As above the starting population X_0 can be selected randomly or in some deterministic way. Subsequent iterates $X_1, X_2, \ldots$ are referred to as *generations* and arise through the application of a binary (or mult-ary) stochastic operator $\nu_t(\cdot,\cdot) : D \times D \to D$ referred to as a *mating* or *cross-over,* or through the application of a stochastic unary operator $\mu_t(\cdot) : D \to D$ as above known as a *mutation* in this context. In the literature there is considerable variety in the details of the application of these operators as well as in the exact nature of the operators themselves. Whatever the details, the generation cycle is completed by the application of a removal process resulting in a new population X_{i+1} whose size equals that of the population X_i at the start of the generation cycle.

The population X_{i+1} at the end of a cycle depends only on the population X_i at the beginning of the cycle and so such a Genetic Algorithm is a Markov chain. Most GA's appearing in the literature employ fixed probabilities over the course of the run and so the chain is stationary. Calculating the corresponding transition probability matrix depends greatly on the details of the implementation and is usually impractical.

We will always assume that every Monte Carlo implementation monitors the "best to the present time" random variable B_t, $t = 1, 2, \ldots$. By this we mean the first domain value occurring among the first t states $X_0, X_1, \ldots, X_t$ for which

$$f(B_t) \leq f(x), \quad x \in \bigcup_{i=0}^{t} X_i.$$

Evidently, by irreducibility, for stationary Markov chains one has condition (0.1) for B_t,

$$\lim_{t\to\infty} \Pr(B_t \in D_{op}) = 1.$$

Genetic Algorithms with Cooling

It is easy to imagine from the foregoing descriptions that the two methods may be combined. By allowing the probabilistic mutation and mating processes of a Genetic Algorithm to vary (possibly) with run time t, it becomes a non-stationary process and encompasses both Simulated Annealing and classical Genetic Algorithms. (Holland [5,p122] allows the possibility of varying probabilities in his algorithm, but makes no further mention of it.) The former is obtained by selecting a null mating operator, the latter by fixing the probabilistic process in time. Equally well the generalized process might be called Simulated Annealing with (stochastic) binary operators.

This paper is organized as follows. In Section 1 we derive formulas for the expected hitting time and the complementary hitting time distribution for stationary and non-stationary finite Markov chains. Also we apply these results to some specific problems. In Section 2 we derive the hitting time formulas for multi-processed implementations of these Monte Carlo methods. Our main result is that in the stationary Markov chain case m-fold multi-processing can result in m-fold speed-up or better in finding exact solutions. In Section 3 we illustrate the results by working through the annealing and evolution methods on sample problems. We show by example that in the non-stationary case, m-fold multi-processing can convert an infinite expectation process into a *finite* one (thereby achieving infinite speed-up).

§1.Hitting Time Calculations

1.1. *Stationary Markov Chains*

Assume that a Monte Carlo method is the simulation of a stationary Markov chain with transition probability matrix $P = (p_{ij})$ where for any $t = 1, 2, \ldots$

$$p_{ij} = \Pr(X_{t+1} = \mathcal{P}_j \mid X_t = \mathcal{P}_i),$$

and $\mathcal{P}_k = \{x_{k_1}, \ldots, x_{k_{n(k)}}\} \subset D$, $k = i, j$, are populations consisting of one or more domain points. Suppose that it is desired to calculate the expected hitting time to any one of a given set of g states which we may assume without loss of generality are the states $\mathcal{P}_1, \ldots, \mathcal{P}_g$. Let E_k denote the expected hitting time if the process starts in state $\mathcal{P}_k, k = g+1, \ldots, N$. Then conditioning on the possible transitions from $\mathcal{P}_k$, we have

$$E_k = (p_{k1} + p_{k2} + \cdots p_{kg}) + p_{k,g+1}(E_{g+1} + 1) + \cdots p_{k,N}(E_N + 1),$$

or equivalently

$$-p_{k,g+1}E_{g+1} - \cdots + (1-p_{kk})E_k - \cdots - p_{kN}E_N = p_{k1} + \cdots + p_{kg} + p_{k,g+1} + \cdots + p_{k,N} = 1.$$

This equation holds for $k = g+1, \ldots, N$, and therefore the list of conditional expectations $E_{g+1}, \ldots, E_N$ uniquely solves the linear system whose matrix form is

$$(I - \hat{P})\mathbf{E} = \mathbf{1} \tag{1.1}$$

where $\hat{P}$ is what remains of the transition matrix P with the latter's first g rows and columns deleted, $\mathbf{E}$ is the $N-g$ dimensional vector of expectations and $\mathbf{1}$ is the vector all of whose components are 1. This cannot be a degenerate system because by irreducibility and the Perron-Frobenius Theorem, the spectral radius of $\hat{P}$ is strictly less than 1. Hence 1 cannot be an eigenvalue for $\hat{P}$.

Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_N)$ be the starting distribution, that is the distribution of X_0,

$$\alpha_i = \Pr(X_0 = \mathcal{P}_i), \quad i = 1, \ldots, N,$$

and let $\hat{\alpha} = (\alpha_{g+1}, \ldots, \alpha_N)$ be the sub-vector of probabilities of starting in the non-goal states. Since the hitting time is 0 if $X_0 \in \{\mathcal{P}_1, \ldots, \mathcal{P}_g\}$ the terms corresponding to $\alpha_1, \ldots, \alpha_g$ make no contribution to the expectation and so the expected hitting time is given as

$$E(\theta) = \hat{\alpha} \cdot \mathbf{E}. \tag{1.2}$$

We see that $E(\theta)$ is finite.

Next we show how to calculate the complementary hitting time distribution, $\Pr(\theta \geq k)$, $k = 1, 2, \ldots$. As above we may assume without loss of generality that the target states are indexed $1, \ldots, g$. Let $\mathbf{c}^{(k)} = (c_{g+1}^{(k)}, \ldots, c_N^{(k)})^T$, $k = 1, 2, \ldots$, be the conditional probability vector of hitting times

$$c_i^{(k)} = \Pr(\theta \geq k \mid \text{ process starts in state } i), \quad i = g+1, \ldots, N.$$

Of course, $\mathbf{c}^{(1)} = \mathbf{1} = (1 \ldots 1)^T$ is the $N-g$ vector of 1's. As above assume the process starts according to the distribution vector $\alpha = (\alpha_1, \ldots, \alpha_N)$ and let $\hat{\alpha}$ be the $N-g$ starting sub-vector for the non-goal states. By conditioning on the starting state we get for $k = 1, 2, \ldots$,

$$\Pr(\theta \geq k) = c_{g+1}^{(k)}\alpha_{g+1} + \cdots + c_N^{(k)}\alpha_N = \hat{\alpha} \cdot \mathbf{c}^{(k)}.$$

But also by conditioning on the transition possibilities, we have

$$c_i^{(k)} = c_{g+1}^{(k-1)}p_{i,g+1} + \cdots + c_N^{(k-1)}p_{i,N} = \hat{p}_{i\cdot} \cdot \mathbf{c^{(k-1)}}, \quad k = 2, 3, \ldots, \quad i = g+1, \ldots, N,$$

where $\hat{p}_{i\cdot}$ is the $N-g$ vector consisting of the i^{th} row of the transition probability matrix P with the first g elements deleted. The matrix form of this equation captures all $N-g$ components at once,

$$\mathbf{c}^{(k)} = \hat{P}\mathbf{c}^{(k-1)}, \quad k = 2, 3, \ldots,$$

where $\hat{P}$ is the matrix whose rows are $\hat{p}_{i\cdot}$, and therefore $\hat{P}$ is, as above, the transition probability matrix P with the latter's first g rows and first g columns deleted. By trivial induction

$$\Pr(\theta \geq k) = \hat{\alpha} \cdot \mathbf{c}^{(k)} = \hat{\alpha} \cdot \hat{P}\mathbf{c}^{(k-1)} = \cdots = \hat{\alpha} \cdot \hat{P}^{k-1}\mathbf{c}^{(1)},$$

that is,

$$\Pr(\theta \geq k) = \hat{\alpha} \cdot \hat{P}^{k-1}\mathbf{1} \ , \quad k = 1, 2, \ldots . \tag{1.3}$$

We next give a useful numerical method for obtaining the probabilities $\Pr(\theta = k)$ and hence also the probabilities $\Pr(\theta \geq k)$. It is by calculating the step by step absolute distributions

$$\alpha_i^t = \Pr(X_t = \mathcal{P}_i) \tag{1.4}$$

where

$$\alpha^t = \alpha^{t-1}P',$$

and the transition probability matrix P' is the given matrix P modified to make the target states absorbing. This can be achieved numerically by noting on each step t the values α_i^t for $i \in \{$ target set$\}$. Their sum will be $\Pr(\theta = t)$. But before conducting the next step, these values are set to zero.

Next observe that $\hat{P}$ is a non-negative matrix whose row sums, $r_i, i = 1, \ldots, N-g$, are bounded by 1; in fact the i^{th} row of $\hat{P}$ is the $(g+i)^{th}$ row of P with its first g elements deleted and so

$$r_i = \sum_{j=1}^{N-g} \hat{p}_{ij} = \sum_{j=g+1}^{N} p_{g+i,j} = 1 - (p_{g+i,1} + \cdots p_{g+i,g}) \leq 1.$$

Furthermore, since we have assumed the process to be irreducible it must necessarily be the case that at least one such row sum is strictly less than 1. Consequently by the Perron-Frobenius theory the spectral radius $\rho(\hat{P}) < 1$ ([12,p.30]) and $\hat{P}$ has a non-negative eigenvalue λ equal to the spectral radius, $\lambda = \rho(\hat{P})$.

Let r_* denote the smallest row sum of $\hat{P}$ and r^* the largest,

$$r_*(r^*) = \min(\max)\{r_i\}_{i=1}^{N-g} \tag{1.5}.$$

Then by standard Perron-Frobenius theory ([12,p.31])

$$r_* < \lambda < r^* \tag{1.6}$$

unless all the row sums are equal, $r_* = r^*$. In the latter case λ is equal to the common row sum. It follows that

$$\lim_{r_* \to 1} \lambda = 1. \tag{1.7}$$

Let χ be a right eigenvector of $\hat{P}$ and ω^T a left eigenvector corresponding to λ.

$$\hat{P}\chi = \lambda\chi, \qquad \omega^T\hat{P} = \lambda\omega^T.$$

We may assume ω^T is a probability vector, $\sum \omega_i = 1$, and that $\omega^T\chi = 1$; then ω^T and χ are uniquely determined. The next result may be found in [11,p.9].

Theorem. With notation as above, put

$$s^{-1} = \hat{\alpha} \cdot \chi (= \hat{\alpha} \cdot \chi\omega^T\mathbf{1}\ ). \tag{1.8}$$

If $\lambda > |\lambda_2|$, where the latter is the magnitude of the next largest eigenvalue after λ, then there exists an integer $1 \le d < N - g$, and a fixed polynomial $h(k)$ of degree $d-1$ such that

$$\left| \frac{1}{\lambda^k}(\hat{\alpha} \cdot \hat{P}^k\mathbf{1}\ ) - s^{-1} \right| < h(k) \left| \frac{\lambda_2}{\lambda} \right|^k, \quad k = 1, 2, \ldots . \tag{1.9}$$

Of course only in unusual cases is it possible to contemplate calculating the factor s. When its exact calculation is feasible, it is better numerically to use (1.9) directly for it,

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

which is akin to the *power method* for eigenvectors and eigenvalues.

1.2 *Estimating the factor s*

The parameter s is estimated as follows. Since ω has been normalized to be a probability vector, $\omega^T\mathbf{1}\ = 1$, therefore $s^{-1} = \hat{\alpha} \cdot \chi$. Define the angle ϕ between ω and χ by $\cos\phi = \omega \cdot \chi / \|\omega\|\,\|\chi\|$ and similarly let δ be the angle between $\hat{\alpha}$ and χ. Then combining $1 = \omega \cdot \chi$ and $s = 1/\hat{\alpha} \cdot \chi$ gives

$$s = \frac{\|\omega\| \cos\phi}{\|\hat{\alpha}\| \cos\delta}. \tag{1.10}$$

This shows that s is the ratio between the projections of the probability vectors ω and $\hat{\alpha}$ onto χ.

Theorem 1.2.1. If the row sums of $\hat{P}$ are equal, $r_* = r^*$, then $s \ge 1$

Proof. Under the hypothesis χ has equal components, say $\chi = \xi(1, 1, \ldots, 1)$. Then $\omega \cdot \chi = \xi$ and $\hat{\alpha} \cdot \chi = \xi \sum_{g+1}^{N} \alpha_i \le \xi$. So $s \ge 1$. ∎

Theorem 1.2.2. If $\hat{\alpha}$ has equal components and $\phi < \delta$, then $s > 1$.

Proof. Under the hypothesis $\hat{\alpha}$ is orthogonal to the hyperplane containing the probability vectors and so has minimum Euclidean norm; hence $\|\hat{\alpha}\| \leq \|\omega\|$. ∎

Theorem 1.2.3. If $\hat{P}$ is symmetric and $\hat{\alpha}$ has equal components then $s \geq 1$. If in addition $\hat{P}$ has unequal row sums then $s > 1$.
Proof. Since $\hat{P}$ is symmetric, $\omega = \chi$ so that $\phi = 0$. Since $\hat{\alpha}$ has equal components, $\|\hat{\alpha}\| \leq \|\omega\|$ and hence $s \geq 1$. If also $\hat{P}$ has unequal row sums, then $\delta \neq 0$, and so $s > 1$. ∎

Starting a Monte Carlo method from a randomly (uniformly) selected state gives rise to an $\hat{\alpha}$ vector with equal components.

We show by example that s can be less than 1. In fact for

$$\hat{P} = \begin{pmatrix} .3 & .1 \\ .8 & .2 \end{pmatrix}$$

$\lambda = 0.537$, $\chi^T = (.761, 1.806)$, and $\omega^T = (.771, .229)$. Taking $\hat{\alpha}^T = (.296, .703)$ gives $s = 0.671$. In this example we see that a process in the states corresponding to $\hat{P}$ will "deliberately" leave these states, i.e. the second state leads to the first with high probability while the first leads to the solution with high probability. We associate such a deliberate path through the $\hat{P}$ states with s-factor values less than 1.

1.3. *Equally Likely Trials*

As a special case consider the optimization process in which on each iteration the next Markov state is selected uniformly at random. Then the transition probability matrix P has identical elements at every position.

$$P = \begin{bmatrix} p & p & \cdots & p \\ \vdots & \vdots & & \vdots \\ p & p & \cdots & p \end{bmatrix}$$

where $p = 1/N$ and N is the number of Markov states. If states 1 through g are the target states then $\hat{P}$ is the $N - g \times N - g$ matrix all of whose elements are p. In this case the $N - g$ vector $\mathbf{1}$ is a positive eigenvector of $\hat{P}$ and the eigenvalue λ is the common row sum

$$\lambda = (N - g)p = 1 - \frac{g}{N}.$$

Now suppose the process starts equally likely in a non-goal state, then $\hat{\alpha} = \left(\frac{1}{N-g} \cdots \frac{1}{N-g}\right)^T$ and

$$\Pr(\theta \geq k) = \hat{\alpha} \cdot \hat{P}^{k-1}\mathbf{1} \ = \hat{\alpha} \cdot \lambda^{k-1}\mathbf{1} \ = \lambda^{k-1}. \tag{1.11}$$

So the theorem of section 1.2 is confirmed with $h(k) = 0$ and $s = 1$. The expected hitting time for equally likely trials is

$$E(\theta) = \sum_{k=1}^{\infty} \lambda^{k-1} = \frac{1}{1-\lambda}. \tag{1.12}$$

1.4. *Non-Stationary Markov Chains*

As above we assume the Monte Carlo method corresponds to a Markov chain X_t, $t = 1, 2, \ldots$, but now we allow the transition probabilities $p_{ij}(t)$ to depend on the iteration index t. In this case the expected hitting time is given by an infinite series. As above assume the target states are indexed $1, \ldots, g$, and let E_i^t denote the expected incremental hitting time to one of $\mathcal{P}_1, \ldots \mathcal{P}_g$, starting from state $\mathcal{P}_i$ at time t, that is E_i^t is the expected increment in the hitting time beyond t. As before, conditioning on the possible transitions from state i, $i = g+1, \ldots, N$, at time t,

$$\begin{aligned} E_i^t &= p_{i1}(t) + \ldots + p_{ig}(t) + p_{ig+1}(t)(E_{g+1}^{t+1} + 1) + \cdots + p_{iN}(t)(E_N^{t+1} + 1) \\ &= p_{i1}(t) + \cdots + p_{iN}(t) + p_{ig+1}(t)E_{g+1}^{t+1} + \cdots + p_{iN}(t)E_N^{t+1} \\ &= \mathbf{1} \ + \hat{p}_{i\cdot}(t) \cdot \mathbf{E}^{t+1} \end{aligned}$$

where $\hat{p}_{i\cdot}(t)$ is the deleted i^{th} row vector of the time t transition matrix P^t, $i = g+1, \ldots, N$, and $\mathbf{E}^{t+1}$ is the expectation vector. In matrix notation

$$\mathbf{E}^t = \mathbf{1} \ + \hat{P}^t \mathbf{E}^{t+1}, \qquad t = 1, 2, \ldots,$$

where $\mathbf{1}$ is the $N - g$ vector of 1's and $\hat{P}^t$ is the matrix gotten from P^t by deleting the latter's first g rows and columns. The solution $\mathbf{E} = \mathbf{E}^1$ is given by induction on the equation above,

$$\mathbf{E} = 1 + \sum_{t=1}^{\infty} \prod_{j=1}^{t} \hat{P}^j \mathbf{1} \tag{1.13}$$

where $\prod_{j=1}^{t} \hat{P}^j = \hat{P}^1 \hat{P}^2 \cdots \hat{P}^t$. Note that this is a series of non-negative terms and so is either finite or the expected hitting times are infinite.

Just as in the stationary case, the hitting time distributions $\Pr(\theta \geq k)$ may be obtained numerically by modifying the transition matrix so that the target states are absorbing.

1.5. *Sampling Without Replacement*

We include for reference the hitting time calculation for a method which is not equivalent to a Markov chain in that each new trial uses the complete history of previous trials. In particular each previously tried point is remembered and not tried again. We include

this example because we will see that even with such complete information parallel Monte Carlo methods can nonetheless be substantially sped-up (cf. Section 2.5).

Suppose the Monte Carlo method is arranged so that the probability of finding an optimal state on the t^{th} iteration is hyperbolically increasing, $\frac{p}{1-(t-1)p}$ for $p = 1/N$, $N = \text{card}(D)$, and $1 \leq t \leq N$. This would arise if visited states are systematically eliminated from further consideration. Then it is easy to see that

$$\Pr(\theta = t) = p, \qquad 1 \leq t \leq N. \tag{1.14}$$

In this case the expected hitting time is finite

$$\Pr(\theta \geq t) = \begin{cases} 0, & t > N, \\ kp, & 1 \leq t = N + 1 - k \leq N \\ Np, & t \leq 1, \end{cases} \tag{1.15}$$

and so from (0.2)

$$E(\theta) = \frac{N(N+1)}{2}p = \frac{N+1}{2}. \tag{1.16}$$

§2. **Parallel Monte Carlo Methods**

We next examine the behavior of Monte Carlo optimization algorithms when multiple copies are run in parallel. We assume that m identical processes are run simultaneously and independently of each other except that any one process can flag them all to stop. The variable t now counts "wall clock time," i.e., the same individual time for each process, not their cumulative time.

This multiprocess can be viewed as an m-tuple $\mathcal{X}_t = (X_t^1, \ldots, X_t^m)$ of the m-individual processes defined on their m-fold product probability space. This holds for both the stationary and non-stationary chains. By the *multi-process hitting time* Θ we mean the random variable equal to the first time t that any one of the m-individual processes is in D_{op},

$$X_t^k \cap D_{op} \neq \{\}, \quad \text{for some } 1 \leq k \leq m$$

and

$$X_\tau^i \cap D_{op} = \{\}, \quad 1 \leq i \leq m, \quad \tau < t.$$

Or equivalently

$$\Theta = \min_{1 \leq i \leq m} \{\theta_i\},$$

where θ_i is the hitting time of the i^{th} process X_t^i, $i = 1, \ldots, m$. We use the notation E or E_1 for $E(\theta)$ and E_m for $E(\Theta)$ when the number of processes is m.

2.1. *Multiprocess Expectation and Hitting Time Distribution Calculation*

We describe here how multi-process expectations can be calculated using the formulas developed in Section 1. Since the multi-process is a Markov process $\mathcal{X}_t = (X_t^1, \ldots, X_t^m)$ which is the Cartesian product of m identical, independent Markov processes X_t, we may take as a state of the multi-process, the Cartesian product of single-process states. The cardinality of the product space is N^m and the multi-process transition matrix P is correspondingly large. Its elements are

$$\begin{aligned} p_{j_1,\ldots,j_m}^{i_1,\ldots,i_m} &= \Pr(\mathcal{X}_{t+1} = (x_{j_1}, \ldots, x_{j_m}) \mid \mathcal{X}_t = (x_{i_1}, \ldots, x_{i_m})) \\ &= \Pr(X_{t+1}^1 = x_{j_1} \mid X_t^1 = x_{i_1}) \cdots \Pr(X_{t+1}^m = x_{j_m} \mid X_t^m = x_{i_m}) \\ &= p_{i_1 j_1} \cdots p_{i_m j_m}. \end{aligned}$$

Now the multi-process expectations and hitting time distributions can be calculated as before using instead this Cartesian product transition matrix. Unfortunately the resulting matrix products quickly become unwieldy. In fact except in special cases P already is. However we show next that there is a much easier way to calculate the complementary hitting time distribution owing to the fact that the processes are identical and independent.

2.2. *The Multi-process Complementary Hitting Time Distribution*

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

$$\Pr(\Theta \geq t) = (\Pr(\theta \geq t))^m, \quad t = 1, 2, \ldots \tag{2.1}$$

and so

$$E_m = E(\Theta) = \sum_{t=1}^{\infty} (\Pr(\theta \geq t))^m. \tag{2.2}$$

We next calculate the expectation of Θ for the hitting time probability distributions of the previous section. From these calculations we will see that the parallel speed-up prospects of Monte Carlo algorithms are excellent. By speed-up we mean

$$\text{Speed-Up } = \frac{E(\theta)}{E(\Theta)}. \tag{2.3}$$

Main Theorem. Let $\mathcal{X} = (X_t^1, \ldots, X_t^m)$ be an m-fold multi-process of m identical independent stationary Markov chains X_t^i, $t = 1, 2, \ldots$, $i = 1, \ldots, m$. If λ is the principle

eigenvalue of the reduced transition probability matrix $\hat{P}$ and $\hat{P}$ is irreducible (see Section 1.1), then

$$\begin{aligned} \text{Speed-up } &= s^{m-1}\frac{1-\lambda^m}{1-\lambda} + \mathrm{O}(1-\lambda^m) \\ &\longrightarrow ms^{m-1} \quad \text{as } \lambda \to 1. \end{aligned} \tag{2.4}$$

Proof. By the results of Section 1 we known that

$$(s^{-1} - h(k-1)\delta^{k-1})\lambda^{k-1} \leq \Pr(\theta \geq k) \leq (s^{-1} + h(k-1)\delta^{k-1})\lambda^{k-1}$$

for $h(\cdot)$ a fixed polynomial and $\delta = \frac{|\lambda_2|}{\lambda} < 1$. Therefore

$$(s^{-1} - h(k-1)\delta^{k-1})^m(\lambda^m)^{k-1} \leq \Pr(\Theta \geq k) \leq (s^{-1} + h(k-1)\delta^{k-1})^m(\lambda^m)^{k-1}.$$

By the Root Test the series $\sum h(k-1)\delta^{k-1}\lambda^{k-1}$ converges for all $0 \leq \lambda \leq 1$, say to $S(\lambda)$. Further as $\lambda \to 1$, $S(\lambda) \to S(1)$ and $S(1)$ is finite. Therefore

$$\frac{s^{-1}}{1-\lambda} - S(\lambda) \leq E(\theta) \leq \frac{s^{-1}}{1-\lambda} + S(\lambda), \qquad 0 \leq \lambda < 1. \tag{2.5}$$

Also the series

$$\begin{aligned} \sum_{k=1}^{\infty}(s^{-1} \pm h(k-1)\delta^{k-1})^m(\lambda^m)^{k-1} &= \sum_{k=1}^{\infty}\sum_{j=0}^{m}\binom{m}{j}s^{j-m}(\pm h(k-1))^j(\delta^j)^{k-1}(\lambda^m)^{k-1} \\ &= \sum_{j=0}^{m}\binom{m}{j}s^{j-m}\sum_{k=1}^{\infty}(\pm h(k-1))^j(\delta^j)^{k-1}(\lambda^m)^{k-1} \end{aligned}$$

converges because for each $j = 0, \ldots, m$ the series

$$S_{\pm}^{(j)}(\lambda) = \sum_{k=1}^{\infty}(\pm h(k-1))^j(\delta^j)^{k-1}(\lambda^m)^{k-1}$$

does. For $j \neq 0$, $S_{\pm}^{(j)}(\lambda)$ is finite for all $0 \leq \lambda \leq 1$, but for $j = 0$

$$s^{-m}S^{(0)}(\lambda) = s^{-m}\sum_{k=1}^{\infty}(\lambda^m)^{k-1} = \frac{s^{-m}}{1-\lambda^m} \to \infty \text{ as } \lambda \to 1.$$

Letting $b_{\lambda}^{\pm}$ denote the sum

$$b_{\lambda}^{\pm} = \sum_{j=1}^{m} s^{j-m}\binom{m}{j}S_{\pm}^{(j)}(\lambda),$$

then by combining inequalities above we get

$$\frac{\frac{s^{-1}}{1-\lambda} - S(\lambda)}{\frac{s^{-m}}{1-\lambda^m} + b_\lambda^+} \leq \frac{E(\theta)}{E(\Theta)} \leq \frac{\frac{s^{-1}}{1-\lambda} + S(\lambda)}{\frac{s^{-m}}{1-\lambda^m} - b_\lambda^-}. \tag{2.6}$$

The third member of this chain of inequalities can be rewritten as

$$\frac{\frac{s^{-1}}{1-\lambda}(1-\lambda^m)}{s^{-m} - (1-\lambda^m)b_\lambda^-} + \frac{S(\lambda)(1-\lambda^m)}{s^{-m} - (1-\lambda^m)b_\lambda^-}.$$

Evidently the second term tends to 0 with $1-\lambda^m$. The difference between the first term and the quantity $\frac{s^m}{s}\left(\frac{1-\lambda^m}{1-\lambda}\right)$ divided by $1-\lambda^m$ is

$$\begin{aligned}&\frac{s^{-1}\left(1-\lambda^m\right)}{1-\lambda}\frac{s^{-m} - \left(s^{-m} - \left(1-\lambda^m\right)b_\lambda^-\right)}{\left(1-\lambda^m\right)s^{-m}\left(s^{-m} - \left(1-\lambda^m\right)b_\lambda^-\right)}\\ &= \frac{s^m}{s}\left(\frac{1-\lambda^m}{1-\lambda}\right)\frac{b_\lambda^-}{s^{-m} - \left(1-\lambda^m\right)b_\lambda^-} \longrightarrow \text{constant}\end{aligned}$$

as λ and hence also λ^m tend to 1. Evidently the first member of (2.6) behaves in like manner and so the estimate (2.4) is proved. ∎

Definition. On the basis of this result, we define the speed-up of a Monte Carlo method to be *linear* when $s = 1$, *superlinear* when $s > 1$, and *sublinear* if $s < 1$.
Figure 2.1 depicts the function $s^{m-1}(\frac{1-\lambda^m}{1-\lambda})$ as a function of m for fixed λ, $\lambda = .99$, and 3 different values of s, $s = 1.02 > 1$, $s = 1$, and $s = .98 < 1$.

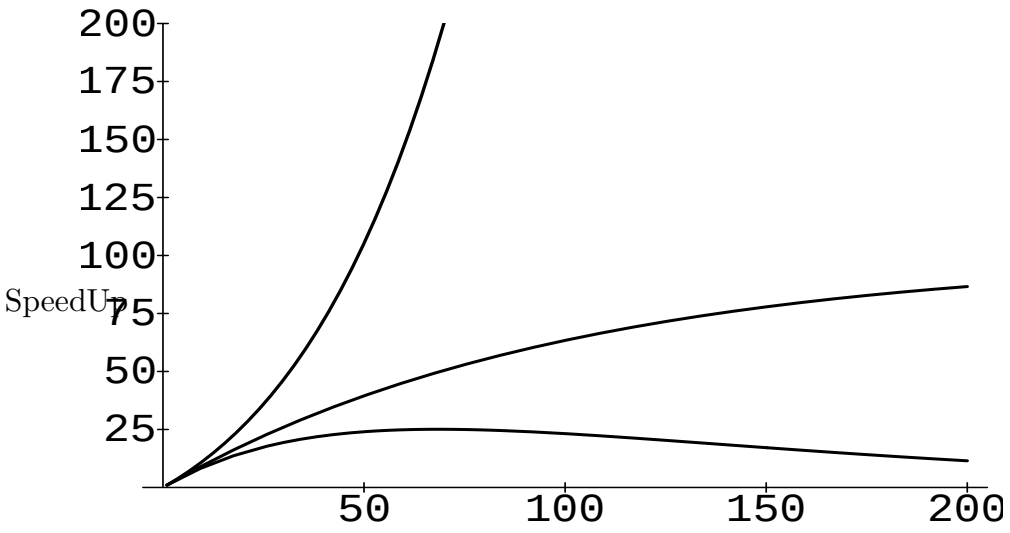

figure 2.1

As a function of m

$$\frac{1-\lambda^m}{1-\lambda} \longrightarrow \frac{1}{1-\lambda}, \qquad m \to \infty$$

so that its contribution to speed-up chokes off for large m. By contrast the term due to s is exponentially increasing with m when $s > 1$ so that the actual speed-up can be *superlinear*.
Remark 1. From eqn.(1.7) λ is near 1 when the smallest row sum r_* of $\hat{P}$ is near 1.
Remark 2. Of course E_m can never be less than 1 so that actual speed-up can never exceed E_1; the term $O(1-\lambda^m)$ does not go away with increasing m.

2.3. *Equally Likely Trials Revisited*

From before, eqn.(1.11), the hitting time distribution for the single process is

$$\Pr(\theta \geq t) = \lambda^{t-1},$$

so that $s = 1$ and $h(\cdot) = 0$. Therefore the exact speed-up for equally likely trials is

$$\frac{1-\lambda^m}{1-\lambda} \longrightarrow \begin{cases} m, & \lambda \to 1, \\ \frac{1}{1-\lambda}, & m \to \infty. \end{cases}$$

2.4. *Linear Speed-up as a Worst Case*

The importance of the result of section (2.3) lies in the fact that since the (stationary) Monte Carlo method which samples the domain space by selecting iterates uniformly at random over the entire space has $s = 1$, any other method found to be "worse" than this, $s < 1$, can simply be abondoned and replaced by equally likely trials. Of course linear speed-up in that case will be relative to the equally likely trials performance of the single process (not the performance of the original method as a single process). But it is even possible to insure that $s > 1$. From Theorem 1.2.3 we only have to make $\hat{P}$ symmetric with at least one state having a better chance of finding the goal (in one step) than the others and start the process with equal likelihood among all the states.

We will see in the next section (by example) that methods which take advantage of special infomation as a single process will not experience full linear speed-up as a multi-process, it could experience only half linear speed-up.

2.5. *Sampling Without Replacement*

Here the single process hitting time distribution is $\Pr(\theta = t) = p$, $1 \leq t \leq N$, according to equation (1.14) and the complementary hitting time distribution is given by equation (1.15). Therefore the multi-process hitting time distribution is

$$\Pr(\Theta \geq t) = \begin{cases} 0, & t \geq N, \\ (kp)^m, & 1 \leq t = N + 1 - k \leq N, \\ (Np)^m, & t \leq 1. \end{cases}$$

where $N = 1/p$. The expected multi-process hitting time is

$$E(\Theta) = (Np)^m + (N-1)^m p^m + \cdots + 2^m p^m + p^m = p^m \sum_{k=1}^{N} k^m.$$

But the sum in the latter member is known to be a polynomial $r(N)$ in N of degree $m+1$ and leading coefficient equal to $1/(m+1)$. Therefore the speed-up is

$$\frac{\frac{1}{2}(N+1)}{\frac{1}{m+1}p^m(N^{m+1} + q(N))} = \frac{m+1}{2} \frac{1 + \frac{1}{N}}{1 + \frac{1}{N^{m+1}} q(N)}$$

where $q(N)$ is a polynomial of degree m. Therefore as $N \to \infty$,

$$\text{Speed-up} \to \frac{m+1}{2}.$$

The speed-up is only about half linear here because by eliminating previously unsuccessful tries, the search algorithm is utilizing more and more information as it proceeds. Yet the

independent multiple processes gain no advantage from their mutual information during a run since they do not communicate and hence do not share it.

§3. **Computational Results on Some Specific Problems**

In this section we apply the foregoing considerations to two illustrative problems. 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. Such a circumstance is deceptive for both the annealing and evolutionary methods.

3.1. *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

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

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*.

There exist preprogrammed Genetic Algorithm solvers for this type of problem such as GAP and GAPR, cf. [10]. The user simply adds his particular alphabet, string length and objective calculation and brings up GAP with the file name containing these additions.

Returning to the problem at hand, we attempt its calculation by an annealer. 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*. Evidently the nature of the objective function obviates the need for a temperature here (all Δu's for states of $\hat{P}$ are 0). 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. Therefore Theorem 1.2.3 applies for a uniformly selected starting state and we can get superlinear speed-up for this problem.

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. In fig. 3.1 we show the complementary hitting time distribution $\{\Pr(\theta \geq k\}$ and its powers. These calculations are performed as follows. Using the power method for eigenvalues in conjunction with equations (1.3) and (1.9) gives λ, s, and $\{\Pr(\theta \geq k\}$. From (1.1) E is obtained. The expectations E_2, E_4, and E_8 are calculated from (2.2). Further in Table 1 we give the averaged empirical expectations $\hat{E}_2$, $\hat{E}_4$, $\hat{E}_8$ of 100 annealing runs (simulations of the Markov Chain) and the corresponding empirical speed-up's. By "time" in the annealing runs we mean the number of iterations taken by the solver process. Of course we (artifically) exclude the possibility of starting in the goal state (cf. section 1.1).

Table 1

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

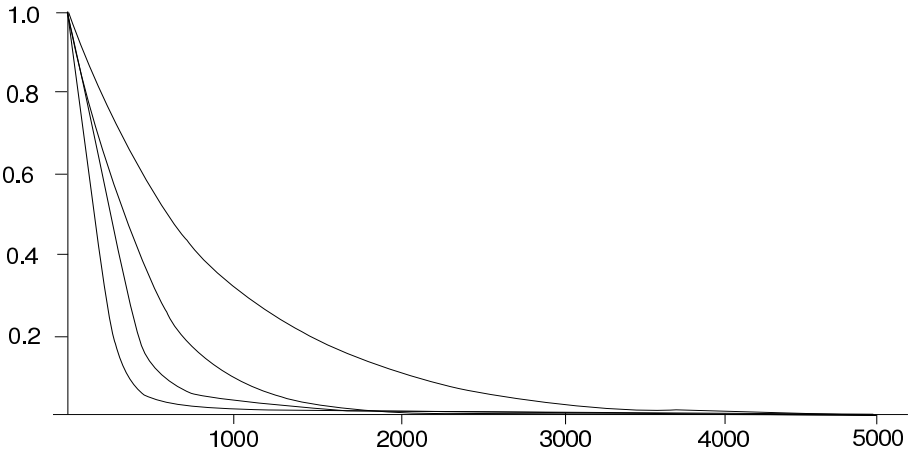

figure 3.1

3.2 *Genetic Algorithm Solver for the Sandia Mountain Problem*

Let the domain D be the set of integers $D = \{0, 1, ..., N\}$, $\text{card}(D) = N + 1$, 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. 3.2.

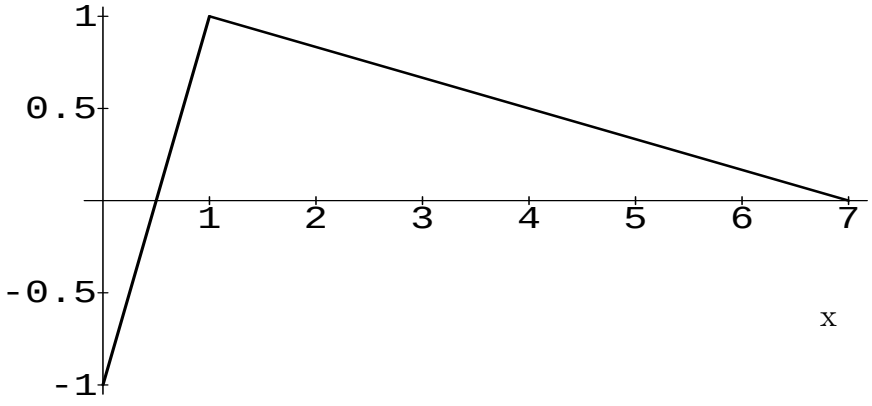

figure 3.2

The global 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. [3]) 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 $p_m = 0.001$. 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 $i = b_1 b_2 b_3$ and $j = B_1 B_2 B_3$ will become

$$b_1 B_2 B_3 \quad \text{and} \quad B_1 b_2 b_3 \quad \text{with probability } \frac{1}{2}$$

or

$$b_1 b_2 B_3 \quad \text{and} \quad B_1 B_2 b_3 \quad \text{with probability } \frac{1}{2}.$$

Finally, under mutation, the population $< i = (b_1 b_2 b_3)_2,\ j = (B_1 B_2 B_3)_2 >$ will become, with prime denoting bit complementation,

$$b_1 b_2 b_3 \quad \text{and} \quad B_1 B_2 B_3 \quad \text{with probability } (1 - p_m)^3$$

or

$$b_1 b_2 b_3' \quad \text{and} \quad B_1 B_2 B_3 \quad \text{with probability } \frac{1}{2}(1 - p_m)^2 p_m$$

and so on until

$$b_1 b_2 b_3 \quad \text{and} \quad B_1' B_2' B_3' \quad \text{with probability } \frac{1}{2} p_m^3.$$

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

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

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 and fig. 3.3.

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

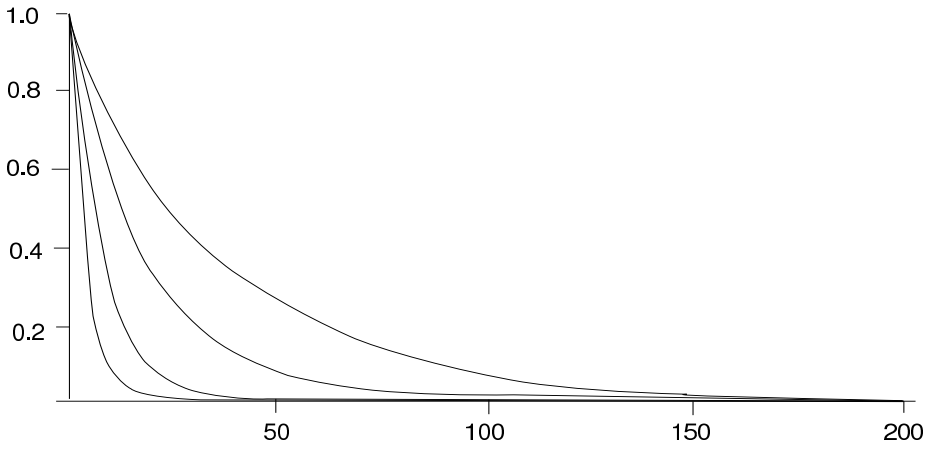

figure 3.3

3.3. *Simulated Annealing Solver for the Sandia Moutain Problem* $N = 2$ Our interest is in showing that the expected hitting time can be infinite when attempted by an annealer. For this purpose we consider the Sandia Mountain problem with $N = 2$ and use a standard simulated annealing algorithm, [9]. We will take the generation matrix to be the 3×3 symmetric matrix

$$G = \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \end{bmatrix}.$$

Letting $\Delta = \frac{1}{N-1} = 1$, the acceptance matrix is

$$A = \begin{bmatrix} - & e^{-2\Delta/T} & - \\ 1 & - & 1 \\ - & e^{-\Delta/T} & - \end{bmatrix}.$$

The transition probability matrix is given by

$$p_{ij} = \begin{cases} g_{ij}a_{ij} & \text{if } i \neq j \\ 1 - \sum_{k \neq i} p_{ik} & \text{if } i = j \end{cases}$$

thus

$$P = \begin{bmatrix} 1 - \frac{1}{2}e^{-2\Delta/T} & \frac{1}{2}e^{-2\Delta/T} & 0 \\ 1/2 & 0 & 1/2 \\ 0 & \frac{1}{2}e^{-\Delta/T} & 1 - \frac{1}{2}e^{-\Delta/T} \end{bmatrix}.$$

From annealing theory (cf. Introduction) the temperature T should vary with iteration count t according to the equation

$$T = \frac{C}{\ell n(t+1)}$$

where C is the depth of the deepest local non-global minimum. Here $C = 1$. Eliminating T gives the transition probabilities directly in terms of t, thus

$$p_{21}(t) = \frac{1}{2}e^{-\Delta \ell n(t+1)} = \frac{1/2}{(t+1)^{\Delta}} = \frac{1/2}{t+1}, \qquad t = 1, 2, \ldots,$$

and

$$p_{22}(t) = 1 - p_{21}(t) = 1 - \frac{1/2}{t+1}, \qquad t = 1, 2, \ldots.$$

To analyze this process we use equation (1.13) to calculate the expected hitting time. In general the iterates

$$\prod_{j=1}^{t} \hat{P}^j$$

quickly become intractable. Indeed the various terms of this product contain all the possible ways leading to state $x = 0$ in t iterations starting from a given state. Here however we estimate these probabilities. Hitting the goal at time k includes the possibility of remaining for $t = 1, 2, \ldots, k-2$ in state $x = 2$, then moving in two consecutive iterations to states $x = 1$ and $x = 0$. Therefore, the probability of hitting at time k is at least as large as

$$\begin{aligned} h_k &= \big(1 - \frac{1/2}{2}\big)\big(1 - \frac{1/2}{3}\big) \cdots \big(1 - \frac{1/2}{k-1}\big)\frac{1}{2}\frac{1}{k}\frac{1}{2} \\ &> \big(1 - \frac{1}{2}\big)\big(1 - \frac{1}{3}\big) \cdots \big(1 - \frac{1}{k-1}\big)\frac{1}{k}\frac{1}{4} \\ &= \frac{1}{4k(k-1)}, \quad k = 2, 3, \ldots. \end{aligned}$$

It follows that the expected hitting time from state 2 is at least as large as

$$\sum_{k=2}^{\infty} k h_k = \frac{1}{4}\sum_{k=2}^{\infty} \frac{1}{k-1} = \infty.$$

3.4. *Example*

Finally, we show by example that an annealer with infinite expectation can be converted to finite expectation when run in parallel. Consider the Sandia Mountain problem with $N = 1$ and the transition probability from state $x = 1$ to the goal $x = 0$ given by

$$p_{10}(t) = \frac{1}{t+1}, \quad p_{11}(t) = 1 - p_{10}(t), \qquad t = 1, 2, \ldots.$$

Then the event that the hitting time will be k occurs if and only if the process remains in state 1 for the first $k-1$ trials and moves to state 0 on the k^{th}; this has probability

$$(1 - \frac{1}{2})(1 - \frac{1}{3}) \cdots (1 - \frac{1}{k})\frac{1}{k+1} = \frac{1}{k(k+1)}, \qquad k = 2, 3, \ldots.$$

and for $k = 1$, $\Pr(\theta = 1) = 1/2$. Therefore

$$\Pr(\theta \geq t) = \sum_{k=t}^{\infty} \frac{1}{k(k+1)} = \frac{1}{t}, \qquad t = 1, 2, \ldots .$$

Then the single process expectation is infinite while the m multi-process expectation is

$$E_m = \sum_{t=1}^{\infty} \frac{1}{t^m} < \infty, \qquad m = 2, 3, \ldots .$$

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 Monte Carlo methods.

Hardware and software considerations for implementing multi-processed Monte Carlo algorithms are its strong point. In software hardly more than a fork call need be added. As to hardware, virtually any parallel architecture will do – shared memory, distributed memory, massively parallel, e.t.c..

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. Conceivably it is possible this technique will yield a faster converging uni-processor algorithm, but only in the most unusual circumstances. Unfortunately running the large number of processes required will in most cases entail so much overhead so as to nullify the gains stemming from the multi-processing.

References

[1] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions, and Bayesian restoration of images", *IEEE Trans*, PAMI (1984).

[2] B. Gidas, "Nonstationary Markov chains and convergence of the annealing algorithm", *J. Stat. Phy.*, **39**(1985), 73-131.

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

[4] B. Hajek, "Cooling schedules for optimal annealing", *Math of Operations Research*, **13** (Feb. 1988)

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

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

[7] J. Kemeny, L. Snell, "Finite Markov Chains", *van Nostrand Co.*, Princeton, NJ (1960).

[8] S. Kirkpatrick, C. Gelatt, M. Vecchi, "Optimization by simulated annealing", *Science* **220** (1983), 671-680.

[9] P. van Laarhoven, E. Aarts, "Simulated Annealing: Theory and Applications", *D. Reidel*, Boston (1987).

[10] M. Nailor, "GAP and GAPR Genetic Algorithm Prototyper", *School of Math. preprint* Ga. Inst. of Tech., Atlanta Ga. 30332 (1989)

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

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

AMS Subject Classification: Primary Secondary
Parallel Speed-Up of Monte Carlo Methods for Global Optimization
by
R. Shonkwiler and Erik Van Vleck
Math: 112989-056