Journal of Computational Physics **148,** 646–662 (1999)
Article ID jcph.1998.6134, available online at http://www.idealibrary.com on IDEAL®

Parallel Simulated Annealing by Mixing of States

King-Wai Chu,* Yuefan Deng,* and John Reinitz†

*Department of Applied Mathematics and Statistics, State University of New York at Stony Brook, Stony Brook, New York 11794; †Brookdale Center for Molecular Biology and Department of Physiology and Biophysics, Box 1126, Mt. Sinai School of Medicine, One Gustave L. Levy Place, New York, New York 10029
E-mail: {kingwai,deng}@ams.sunysb.edu, reinitz@kruppel.molbio.mssm.edu

Received June 14, 1998; revised October 6, 1998

We report the results of testing the performance of a new, efficient, and highly general-purpose parallel optimization method, based upon simulated annealing. This optimization algorithm was applied to analyze the network of interacting genes that control embryonic development and other fundamental biological processes. We found several sets of algorithmic parameters that lead to optimal parallel efficiency for up to 100 processors on distributed-memory MIMD architectures. Our strategy contains two major elements. First, we monitor and pool performance statistics obtained simultaneously on all processors. Second, we mix states at intervals to ensure a Boltzmann distribution of energies. The central scientific issue is the inverse problem, the determination of the parameters of a set of nonlinear ordinary differential equations by minimizing the total error between the model behavior and experimental observations.

Key Words: simulated annealing; parallel processing; inverse problems.

1. INTRODUCTION

Simulated annealing (SA) is an effective method for the optimization of complex cost functions for which heuristic methods do not exist. Its strength is that under appropriate conditions it will attain the global extremum of the cost function; its weakness is high computational demand. For this reason it is of great importance to parallelize SA. Previous work in this area has produced a number of methods that perform well on certain problems, but we are not aware of algorithms that perform well on general applications.

In this paper we introduce a new class of algorithms for parallel SA that does not depend on the structure of the optimization problem. We show that one particular implementation of this algorithm is scalable for up to 100 processors at nearly 100% parallel efficiency.

In Section 2, we discuss the Metropolis method, SA, and the Lam-enhanced SA, and then we review previous work and introduce our new approach to parallelizing SA. In Section 3,

0021-9991/99 $30.00

we describe our particular application, an inverse problem which arises in developmental biology, in which the parameters of a set of nonlinear ordinary differential equations (ODEs) are determined from time series data by a fitting process. In Section 4, we present our own approach to the problem and show the dependence of the performance of our approach on algorithmic parameters. We also introduce an accurate method of measuring the quality of results.

Accurate measurement of the quality of results is important for two reasons. First, the evaluation of the performance of a SA algorithm requires good statistics because of the stochastic nature of the algorithm itself. Second, wall clock time reduction in computing is almost always meaningless for measuring the parallel efficiency of the optimization algorithms. A speedy floating point operation done by multiple processors does not guarantee quicker solution of the problem because of possible alteration of convergence behavior by parallelization. We show how changes in the quality of the answer computed in parallel can be compensated for by constructing a function relating the quality of the answer to the number of iterations required to obtain it in serial.

2. ALGORITHMS

Our approach involves a sequence of algorithms based upon the Metropolis algorithm [17]. First SA [11] was introduced as an optimization method, generalizing the Metropolis algorithm to include variable temperature for escaping from local minima effectively. Second, Lam [12] further enhanced SA by adaptively controlling the rate of temperature decrease and the size of moves so as to maximize the optimization efficiency. Third, we have parallelized Lam's version of SA. As is well known, SA is difficult to parallelize due to its intrinsic serial nature and direct parallelization of the Metropolis algorithm will not be scalable. Our approach involves strategies of pooling statistics of both state and algorithm parameters, while stirring the system.

Now, we will describe the Metropolis algorithm, the SA, and our parallelization.

2.1. The Metropolis Method

The well-known Metropolis algorithm is a method of sampling a Boltzmann distribution. A system with energy E_{old} is provisionally perturbed into a new state with energy E_{new}. Such a perturbation is called a "move." If $E_{\text{new}} < E_{\text{old}}$ the new state is accepted, and if $E_{\text{new}} > E_{\text{old}}$ the new state is accepted with probability $\propto \exp(-(E_{\text{new}} - E_{\text{old}})/T)$, where T is the temperature. If the new state is not accepted, the system remains in the old state. This algorithm tends to transform any distribution into a Boltzmann distribution and maintains a preexisting Boltzmann distribution [17].

The original purpose of this algorithm was to perform ensemble calculations numerically. Although an ensemble typically has an infinite number of members and traditionally was used for analytical calculations, the advent of computing machinery has led to the popularity of making ensemble calculations by Monte Carlo methods such as the Metropolis algorithm. In these methods, the ensemble is sampled and placed in computer storage. The ensemble is stored as a very large array of structures, each containing the values of the microscopic and macroscopic variables for that particular ensemble member.

The utility of an ensemble stems from the fact that an appropriate average for some quantity over the ensemble will give the same result as taking a time average over observations with a physical measuring device—the "ergodic property." We note that the Metropolis

algorithm itself can be thought of as an application of ergodicity, in the sense that an average over Metropolis samples gives the same result as either an ensemble average or a physical measurement. Although the Metropolis algorithm has its own internal sense of time, in terms of a current state that may or may not be replaced by a proposed new state, that succession of states cannot be identified with the succession of states in an actual physical system. The ergodic nature of the Metropolis algorithm suggests the basic strategy of parallelizing SA discussed throughout this paper.

2.2. Simulated Annealing

SA was first introduced as an optimization method by Kirkpatrick *et al.* [11] using a metaphor from statistical physics. SA generalizes the Metropolis algorithm by allowing the temperature parameter to change. This temperature is usually lowered slowly during the annealing process. The precise rate of change of the temperature, or the annealing schedule, is the most essential part of SA as it determines the performance of the algorithm. A high cooling rate leads to poor results because of lack of representative states, while a low cooling rate fails to produce results in a practical amount of time.

A number of annealing schedules have been used, of which the three most important are called logarithmic, Cauchy, and exponential [3]. It has been proved that SA will converge to the global minimum of the cost function if temperature changes are governed by the logarithmic schedule, in which the temperature T_k at step k is given by $T_k = T_0/\ln k$ [6, 3]. These proofs use moves drawn from a Gaussian distribution. A faster schedule in which $T_k = T_0/k$ was shown to converge to the global minimum when moves are drawn from a Cauchy distribution [27], and this schedule is known as the Cauchy schedule. It is sometimes called "fast simulated annealing." Even faster is the exponential, or geometric, schedule where $T_k = T_0 e^{-ck}$. To our knowledge, no rigorous proof of the convergence of this schedule to the global optimum has been made, although good heuristic arguments for its convergence have been made for a system in which annealing state variables are bounded [9].

The severe computational requirements of SA mean that an exponential schedule is required for practical computation. Convergence behavior must be assessed numerically. In these computations there is often little prior knowledge of the properties of the cost function, a fact with two important consequences. First, it is usually necessary to choose a move generation function in a problem-dependent manner. This contrasts to proofs of convergence, where the move generation function is essentially chosen in a schedule-dependent manner. Second, the lack of prior knowledge of cost function properties can be compensated for by using information about the cost function obtained during the annealing run itself. Schedules that make use of such information are called "adaptive schedules." Such schedules adjust the rate of temperature decrease, based on the partial derivatives of the cost function at a single point [9] or on its statistical properties [1, 8]. Of particular importance is the adaptive schedule of Lam.

2.3. The Lam Schedule

The Lam schedule [12, 13] was derived by optimizing the rate at which temperature can be decreased subject to the constraint of maintaining quasi-equilibrium. It differs from other adaptive schedules in that it explicitly takes into account the effect of move generation strategies and provides an adaptive recipe for their control. The Lam schedule is not widely known, and its key features are reviewed below.

The Lam schedule is an adaptive exponential schedule given by

$$s_{k+1} = s_k + \lambda\left(\frac{1}{\sigma(s_k)}\right)\left(\frac{1}{s_k^2\sigma^2(s_k)}\right)\left(\frac{4\rho_0(s_k)(1-\rho_0(s_k))^2}{(2-\rho_0(s_k))^2}\right), \tag{1}$$

where $s_k = 1/T_k$, and T_k is the temperature at the kth evaluation of the cost function E. $\sigma(s_k)$ is the standard deviation of E at this step, and $\rho_0(s_k)$ is the *acceptance ratio*; that is, the ratio of accepted to attempted moves. The four factors play the following roles:

1. λ is a quality factor. Making λ smaller increases the quality of the answer but it also increases the computation time.
2. $(1/\sigma(s_k))$ measures the distance of the system from quasi-equilibrium.
3. $(1/s_k^2\sigma^2(s_k))$ is the inverse of the statistical specific heat which depends on the variance [11].
4. $(4\rho_0(s_k)(1-\rho_0(s_k))^2/(2-\rho_0(s_k))^2)$ is equal to $\rho_2/2$, where ρ_2 is the variance of the average energy change during a move. It is a measure of how effectively the state space is sampled and was found to be at a maximum value when $\rho_0 \approx 0.44$.

Lam supplies a set of statistical estimators for $\sigma(s_k)$ and the average energy $E(s_k)$. These estimators are computed from averages taken at two scales. The energy and variance are averaged every τ moves, where τ is an empirical algorithm parameter on the order of 100 to 1000. These averaged energies are in turn averaged in an annealing-time decaying running average, the time constants of which are set by the user. The running average is useful in reducing errors in σ due to autocorrelation in the SA process. The dependence of the estimated energy and variance on temperature is found by a fit to the probability density function modeled by a local gamma distribution. Thus, the estimated σ_s is used to lower the temperature at each step according to Eq. (1).

2.4. Parallelization

The original Metropolis algorithm [17] from which most SA algorithms are derived serializes a parallel physical process, i.e., computing ensembles. In the Metropolis algorithm, a new state always depends on one or more previous states plus one or more random variables. This serial nature of SA is an inherent distraction to parallelization. Much previous work in parallelizing SA did not consider this underlying physical metaphor but attempts to parallelize by dividing state variables or by multiple moves on the same variable among processors [7].

The most successful scalable method, known as asynchronous parallelization, can lead to acceptable speedup. For cost functions that are loosely coupled, i.e.,

$$E(x_1, \ldots, x_i, \ldots, x_n) \approx E(x_1) + \cdots + E(x_i) + \cdots + E(x_n), \tag{2}$$

each processor receives a copy of the state $(x_1, \ldots, x_i, \ldots, x_n)$. Each processor j then executes the Metropolis algorithm, altering a range of state variables $x_j, x_{j+1}, \ldots, x_{j+m}$. Node j uses unchanged values of x_i for $i < j$ and $i > j+m$ while these state variables are being altered by other processors. The state information of each processor will become increasingly erroneous as this takes place, but after some number of iterations the processors pool their new values of x_i and then dispatch them anew for further annealing. This method can give good parallel speedup, but only if (2) is satisfied. For cost functions without such properties, the algorithm can show catastrophic divergence [16].

Our approach is to take the notion of the Metropolis algorithm as a convenient way of generating ensemble samples on a serial computer and extend it by considering how to generate ensemble samples on a parallel computer, while being careful to maintain the Boltzmann distribution as the temperature is lowered. The essential ideas are very simple. A Boltzmann distribution can be sampled in parallel by a set of processors which all execute the Metropolis algorithm using independent Markov chains. Averages over macroscopic variables are performed by pooling statistics obtained from all processors. As the temperature is lowered, from time to time mixing occurs during which each processor chooses a preexisting state of energy E_s with probability $\propto \exp(-E_s/T)$. The detail of this procedure are given in Section 4.1.2.

Elements of the above approach, pooling of statistics combined with mixing of states, have been used separately by others. Frost has developed a method based on pooling energy statistics among many processors, but without communication of state information. This scheme implements an adaptive schedule which adjusts the cooling, based on estimates of specific heat. No explicit measurements of speedup are reported [5, 4].

The idea of mixing and selecting states has been used in some related work. The systolic algorithm of Aarts *et al.* makes a series of pairwise state selections according to the Boltzmann distribution [2] and achieved 75% parallel efficiency on eight processors. Roussel-Ragot and Dreyfus have reported a method designed to maintain the Boltzmann distribution in parallel in which all processors synchronously make a move from a common state and one of the accepted moves is randomly chosen as the new common state [24]. If no moves are accepted, the process repeats until a new state is accepted by one processor. If move generation is not adaptively controlled then, typically, acceptance rates are high at high temperatures and low at low temperatures, so this algorithm can produce good speedup at low temperatures. However, the same efficiency increase can be had in serial by adaptive control of the move size. Ingber has proposed a similar method, but no information is given on the speedup achieved [10]. Slezak was able to obtain 50% parallel efficiency on 40 processors using a mixing method in which the state of the lowest energy processor is copied to all processors [26]. None of these methods use all-to-all Boltzmann mixing, however, nor are they parallelizations of an adaptive serial algorithm.

Our algorithm for parallel SA does not depend on any special properties of the cost function. The algorithm was developed to solve a particular inverse problem arising in developmental biology, in which the parameters of a set of nonlinear ODEs are determined from time series data by a fitting process. This problem has proved refractory to many of the common annealing schedules because the state variables for the optimization have widely differing characteristic scales. Hence the problem requires the Lam schedule for solutions in serial. Furthermore, the cost function associated with this problem is completely inseparable, and asynchronous SA fails to converge. Therefore, we need to parallelize the Lam schedule to solve this problem.

3. THE INVERSE PROBLEM IN GENE CIRCUITS

A set of techniques, called "gene circuits" has been developed for analyzing gene networks [18, 21–23, 25, 20]. The gene circuit method takes gene expression data as input and computes a regulatory circuit as output. We represent a circuit by the elements T^{ab} of a matrix, each one of which characterizes the regulatory effect of one gene on another by a single real number for each possible pair a and b. Thus if T^{ab} is positive gene b activates

gene a; if it is negative gene b represses gene a, and if T^{ab} is zero gene b has no effect on gene a. This is the simplest model that allows for the possibility that each gene regulates every other gene.

3.1. Chemical Kinetic Equations

The equations describe a stage of insect development in which the embryo contains only cell nuclei which are not yet separated by cell membranes. Consider a linear array of cell nuclei indexed by i, such that nuclei $i+1$ and $i-1$ are the neighbors of nucleus i. Each cell nucleus contains a copy of a regulatory circuit composed of N genes, and which is determined by an $N \times N$ matrix $\mathbf{T}$. (Note that N, n, m, and λ refer to quantities in Eq. (3) in this section only.) The concentration of the ath gene product in nucleus i is a function of time, denoted by $v_i^a(t)$. Then

$$\frac{dv_i^a}{dt} = R_a g_a\left(\sum_{b=1}^{N} T^{ab} v_i^b + m^a v_i^{bcd} + h^a\right) + D^a(n)\left[\left(v_{i-1}^a - v_i^a\right) + \left(v_{i+1}^a - v_i^a\right)\right] - \lambda_a v_i^a, \tag{3}$$

where N is the number of zygotic genes included in the circuit. The first term on the right-hand side of the equation describes gene regulation and protein synthesis, the second describes exchange of gene products between neighboring cell nuclei, and the third represents the decay of gene products.

In (3), T^{ab} is a matrix of genetic regulatory coefficients. $m^a v_i^{bcd}$ is a bias term arising from the spatially varying but time-invariant distribution of a maternally expressed morphogenetic protein known as Bicoid (Bcd) [15, for a review], where v_i^{bcd} is the concentration of Bcd protein in nucleus i and m^a is the regulatory coefficient of Bcd acting on zygotic gene a. g_a is a "regulation-expression function," which we assume takes the form $g_a(u^a) = (1/2)[(u/\sqrt{u^2+1}) + 1]$ for all a, where $u^a = \sum_{b=1}^{N} T^{ab} v_i^b + m^a v_i^{bcd} + h^a$. R_a is the maximum rate of synthesis from gene a, and h^a summarizes the effect of general transcription factors on gene a. The diffusion parameter $D^a(n)$ depends on the number n of cell divisions that have taken place and varies inversely with the square of the distance between nuclei. We assume that the distance between adjacent nuclei is halved after a nuclear division. λ_a is the decay rate of the product of gene a. Nuclear divisions are incorporated by shutting down synthesis during each mitosis and doubling the number of nuclei.

3.2. The Cost Function

Our cost function is given by

$$E = \sum_{\substack{\text{all } a, i, t, \text{ and} \\ \text{genotypes for} \\ \text{which data exists}}} \left(v_i^a(t)_{\text{model}} - v_i^a(t)_{\text{data}}\right)^2 + (\text{penalty terms}) \tag{4}$$

which is a function of ODE parameters and the experimental data. This dependence is encoded in the first term which contains the sum of the squared deviation of the solutions to (3) for each time, gene product, genotype, and nucleus for which data exists. The penalty terms limit the search space by adding a very large or (in some cases) infinite term to the energy when parameters are outside the search space. Clearly, Eq. (2) is not satisfied by this equation.

The penalty terms Π in Eq. (2) are given by

$$E_{\text{penalty}} = \Pi_{R^a} + \Pi_{\lambda^a} + \Pi_{u^a},$$

where Π_{R^a} and Π_{λ^a} represent search space limits for R^a and λ^a, respectively. When these search space limits were exceeded, the move was rejected without integrating the equations, so this term may be thought of as being zero within the search space and infinity outside it. The third penalty term is given by

$$\Pi_{u^a} = \begin{cases} \exp\left(\Lambda\left(\sum_{ab}\left(T^{ab}v^b_{\max}\right)^2 + \left(m^a v^{bcd}_{\max}\right)^2 + (h^a)^2\right)\right) - 1 \\ \qquad \text{iff } \Lambda\left(\sum_{ab}\left(T^{ab}v^b_{\max}\right)^2 + \left(m^a v^{bcd}_{\max}\right)^2 + (h^a)^2\right) > 0 \\ 0, \qquad \text{otherwise.} \end{cases} \quad (5)$$

Here $v^b_{\max}$ and $v^{bcd}_{\max}$ are the largest values of v for gene a and v^{bcd} found in the dataset. Λ controls the size of the search space for terms that contribute to u^a, since a very large penalty will arise whenever the argument of the exponential is greater than $1/\Lambda$. Thus, the effect of the penalty on u^a is to limit the maximum saturation of u to $(1 - \Lambda)$.

3.3. Serial Optimization of the Cost Function

Lam's schedule (1) is not dependent on any particular problem. A problem-specific move generation scheme must also be supplied. In order to keep the overall acceptance rate close to 0.44, the move generation scheme must allow for changes in the average size of moves. Increases in the size of proposed moves tend to decrease the acceptance rate, and decreases in the move size increase the acceptance rate. Larger moves are less likely to be accepted at lower temperatures. The net result is that the size of moves decreases slowly during the annealing run as the system samples the state space more and more selectively during convergence to a global minimum. Here and in previous work move size was controlled by drawing moves from an exponential distribution with a mean of $\boldsymbol{\theta}^i$, so that a move consists of changing x_i to x_i^{new} by setting

$$x_i^{\text{new}} = x_i \pm \boldsymbol{\theta}^i \ln \xi, \quad (6)$$

where the sign is chosen randomly, and the random number $\xi \in (0, 1]$ is drawn from a uniform distribution. $\boldsymbol{\theta}^i$ is adaptively controlled for each variable by accumulating acceptance statistics and periodically changing $\boldsymbol{\theta}^i$ by setting

$$\ln \boldsymbol{\theta}^i_{\text{new}} = 3.0(\rho_0 - 0.44) + \ln \boldsymbol{\theta}^i_{\text{old}} \quad (7)$$

in such a way as to keep the acceptance rate at the desired level of 0.44.

The annealing process is initialized by first making a large number (10,000 in the work reported here) of moves at a high ($T = 1000$) initial temperature, discarding all statistics to erase any trace of the initial state, and then performing the same number of moves again to gather initial statistics.

The annealing procedure is terminated when the average energy after τ steps fails to change by more than a stopping criterion κ three times in succession. The need for a stopping criterion is characteristic of a problem with continuous variables. In a problem with discrete variables like the traveling salesman problem (TSP) [14], the annealer would stop when the energy became completely fixed—that is, when no further moves are accepted.

The importance of the Lam schedule to this optimization problem can be summarized by two observations. Without the adaptive control of move size, the algorithm *fails* to converge to the global minimum. Without the statistical estimators for the variance, convergence takes about three times as many iterations as when they are used.

The reliability of the SA procedure was assessed by assigning parameters randomly, solving the equations, and then using the solutions at a few time steps as synthetic "data" for the annealing procedure. Accuracy of the algorithm is then assessed by recovery of the original parameters. In this paper we consider a test problem with two genes and eight nuclei. The parameters associated with one gene are fixed, so seven are sought by the annealer. For this problem R_a, λ_a, and D^a are recovered to about 0.01% accuracy, T^{ab} and m^a to about 1%, and h^a to about 7%.

4. IMPLEMENTATION AND PERFORMANCE ANALYSIS

In this section we discuss the details of the implementation, how the results were analyzed, and the results themselves.

4.1. Implementation

The two design principles of pooling statistics and periodically mixing states are implemented as follows. This implementation was originally done with the Paragon message-passing NX library, and the results described here were obtained with that implementation running on a 128-processor Paragon XPS at SUNY–Stony Brook. This machine is a distributed-memory MIMD architecture with i860 processors. More recently the implementation has been ported to MPI for use on clustered workstations coupled by fast Ethernet. Source code is available on request.

The analysis of performance in our study does not depend on the specifics of the particular architecture as our focus of the analysis is on the quality of algorithms. The measurement of parallel efficiency requires results from serial machines. For our serial performance data, we used workstations with the Alpha 21164 processor, which is about 10 times faster than the i860 in the Paragon.

4.1.1. *Parallel Statistics*

The Lam schedule takes three kinds of averages, of which two are concerned with the current energy and variance and the third, with the control of move generation. Current energy and its variance is estimated in two stages, in which averages over τ steps are in turn averaged with past averages over τ steps with exponentially decaying weights. In the work described here, we divide the cost function evaluations required to compute the average over τ among all of the available processors. Following each evaluation of the cost function, each processor lowers its temperature by an amount equal to the change in temperature in the serial case multiplied by the number of processors, so that the temperature change over τ moves is the same in the parallel and serial cases.

We do not parallelize the longer term averages used to estimate the local gamma distribution and, through it, the energy and variance. This approach was chosen as the most direct route to parallelism; it has the advantage that Lam's estimators can be used unchanged with a synchronous and identical calculation of them taking place in each processor. It has the disadvantage of introducing an artificial scaling limitation into the parallel algorithm

because it is not possible to split the job onto more than τ processors. In all calculations reported in this paper, $\tau = 100$.

During initialization, the 10,000 moves designed to erase the initial state are performed separately on each processor. The subsequent set of initial moves designed to collect statistics are allocated among the processors and performed in parallel. We have not made any attempt to measure the minimum number of iterations required for the initial run of decorrelation moves, which are not part of the parallel algorithm. For this reason, measurements of the number of cost function evaluations N reported below do not include the initial 10,000 decorrelation moves, but do include the initial moves used to gather statistics.

Statistics on the acceptance rate of proposed moves are calculated in parallel by a straightforward parallelization of the serial method. In the serial case, acceptance statistics are calculated over 100 sweeps, where a sweep is defined as a cycle of making moves on each parameter in turn.

In the parallel case, each processor independently attempts a move on the same parameter. After one sweep with P processors, P attempted moves have been made on each parameter among all the processors. After 100 attempted moves on each parameter have been performed among all processors ($100/P$ sweeps), acceptance statistics are pooled and an identical calculation of the new average move size is performed in each processor. Move generation (see Eq. (6)) is problem specific, and hence, our move generation module shares almost no data with the code module responsible for calculating Lam statistics.

This fact, together with the need to make synchronous and identical calculations of both Lam estimators and acceptance statistics, introduces the disadvantage that the number of processors used for the calculation must be a divisor of τ.

4.1.2. *Mixing of States*

The above procedure constitutes a parallelization of the sampling of a Boltzmann distribution. Extension to a full parallel SA algorithm requires taking advantage of parallelism to maintain a Boltzmann distribution while the temperature is being lowered. The fundamental limitation on increasing the rate of cooling in SA is that the Markov chain cannot sample a sufficient number of states in the neighborhood of a given temperature to stay in quasi-equilibrium. The solution to this problem is to allow each processor to sample the states of other processors. After m repetitions of the loop over τ, such that the cost function has been evaluated $m\tau$ times altogether and $m\tau/P$ times in each processor, a set of synchronous messages are passed by which each processor broadcasts its current energy to all other processors. Each processor then randomly chooses a particular target processor, with the probability of choosing processor p given by

$$\frac{e^{-E_p/T}}{\sum_i e^{-E_i/T}}, \tag{8}$$

and these choices are synchronously shared among processors. At this point each processor is committed to adopting a new state from its particular target processor p, and the new states are distributed with point-to-point messages.

In the absence of an analytical model for this algorithm, the optimal choice of m is an empirical process. Communication is essentially eliminated when m is large. In the case of small m, favorable states will populate the system so rapidly that state variables become excessively correlated. This correlation will lead to erroneously small estimates of

the variance which in turn will cause the temperature to be lowered too rapidly, resulting in severe loss of quality of the results. In the sequel, we show that both of these effects indeed take place, but that a suitable choice of m can lead to very high annealing efficiencies.

4.2. Performance and Analysis

Careful study of the efficiency of parallel SA encounters difficulty from two sources. First, the stochastic nature of the algorithm requires careful statistical treatment of the results. Second, parallelization can introduce changes in the quality of the final results which affect the estimate of the speedup.

4.2.1. *Average Performance with the Associated Error*

Repeated annealing runs on the test problem typically gave results that had large variance, about an order of magnitude in both the final energy and the number of iterations (N). In performance studies, at least 100 runs were performed under each set of conditions and arithmetic averages were taken over N and the final energy E. When error ranges for these averages are shown, they were calculated by a bootstrapping ("jackknifing") procedure [19]. Given a set of J final states, J samples are drawn randomly from the results without regard to duplication and averaged. The error shown represents the extrema of the resulting set of averages.

4.2.2. *The Serial Performance Curve*

In order to compensate for changes in the quality of the result because of parallelization, it is desirable to know the expected number of serial iterations corresponding to a particular energy. Then the speedup can be calculated by dividing the number of expected serial iterations at the final energy obtained in parallel by the average number of parallel iterations required. We first must establish the existence of such a serial performance curve for a given cost function.

In the application of SA to a classical combinatorial optimization problem such as the TSP, it is easy [13] to construct a curve relating the average final energy to the average number of iterations required to attain that energy. Such a curve is obtained by varying the overall cooling rate, λ. In a system with a continuous state space, the resulting quality is determined not only by λ, but also by the stopping criterion, κ. This raises the possibility that a given final energy does not correspond to a unique number of iterations.

We characterized the dependence of final energy E_f and the number of serial iterations N_s on λ and κ by an extensive series of numerical runs. At large values of λ we noticed a bimodal distribution of final energies, in which a small proportion of runs finished with E_f significantly greater than unity, while most runs completed with E_f on the order of 10^{-3} or less. In order to understand this behavior better, we performed a series of quenching runs, in which the temperature was instantly lowered from the initial equilibration level to zero at the start of annealing. A histogram of the E_f resulting from such a set of runs is shown in the inset to Fig. 1. The resulting distribution is strongly bimodal, with two empty decades separating a peak around 10^2 from one at around 10^{-4}. These indicate that the stochastic dynamics of the annealing process is bistable, and we refer to the left-hand peak in the inset to Fig. 1 as the "annealed attractor" and the right-hand one as the "quenched attractor." Each contains approximately the same number of points, are clearly separable, and the quenched attractor contains enough points for statistical study.

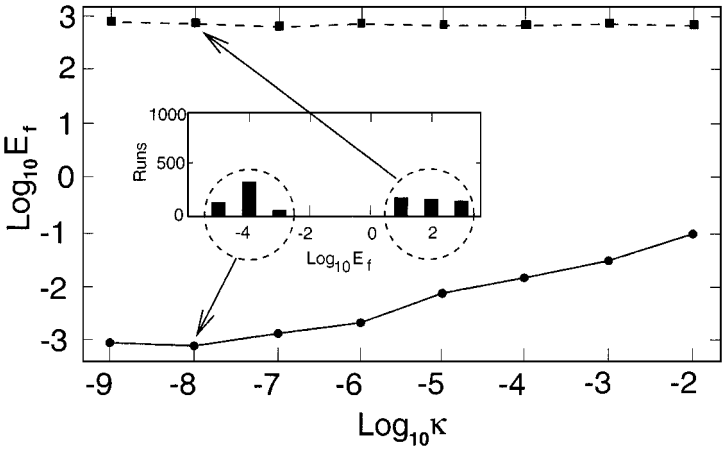

FIG. 1. Bimodal behavior. This figure shows the bimodal behavior of the solution E_f for given values of algorithm parameter κ with algorithm parameter $\lambda = \infty$. As shown by the inserted histogram, for $\kappa = 10^{-8}$ we see runs producing E_f with mode around 10^2 and around 10^{-4} separated by two empty decades (4 runs fell in the -2 bin). Note that the histogram uses logarithmic coordinates, while arithmetic averages were performed to obtain the points on the graph. Hence the modes of the histogram differ from the plotted means.

We characterized the behavior of these two attractors further by varying κ while holding $\lambda = \infty$. The result is shown in Fig. 1. It indicates that the annealing attractor shows decreasing E_f with decreasing κ, but that E_f for the quenching attractor remains essentially constant. As λ decreases, E_f's belonging to the quenching attractor become increasingly rare. When $\lambda < 3 \times 10^{-3}$, the frequency drops to less than 10^{-3} and becomes undetectable. We have never seen a representative of the quenching attractor with E_f less than unity, and in all cases for which averages are shown there was at least an order of magnitude separating members of the quenching attractor from members of the annealing attractor. For this reason, in the subsequent results we rejected all $E_f > 1$ as a member of the quenching attractor. These constituted less than 1% at $\lambda = 1 \times 10^{-2}$ and 2 out of about 7800 experimental runs performed in parallel.

We then characterized the dependence of the average E_f and N_s of the annealing attractor on λ and κ, as shown in Fig. 2. Inspection of this figure shows that the upper right-hand side of the energy/iterations plane is well coordinatized by curves of constant λ and κ. In this region, changing λ alters the number of iterations required but has little effect on the energy. Similarly, alteration of κ affects the final energy but has little effect on the required number of iterations. The lower left-hand side of the E_f–N_s plane is the region where SA is most efficient. Curves which approach the lower left-hand portion of the plane run into an envelope of maximum efficiency, which constitutes the desired characteristic relationship between energy and serial iterations for efficient annealing. In particular, note that a trajectory along this envelope can be traversed by varying either λ or κ.

Close inspection of Fig. 2 reveals another interesting feature. Decreasing κ, while holding $\lambda = \text{const}$ for most values of λ, produces an almost horizontal trajectory on the E_f–N_s plane which, when it encounters the envelope of maximum efficiency, traverses the envelope towards the upper left. Similarly, increasing λ while holding $\kappa = \text{const}$ for most values of κ produces an almost vertical trajectory which, when it encounters the envelope of maximum efficiency, traverses the envelope towards the lower right. The curves for $\lambda = 10^{-1}$

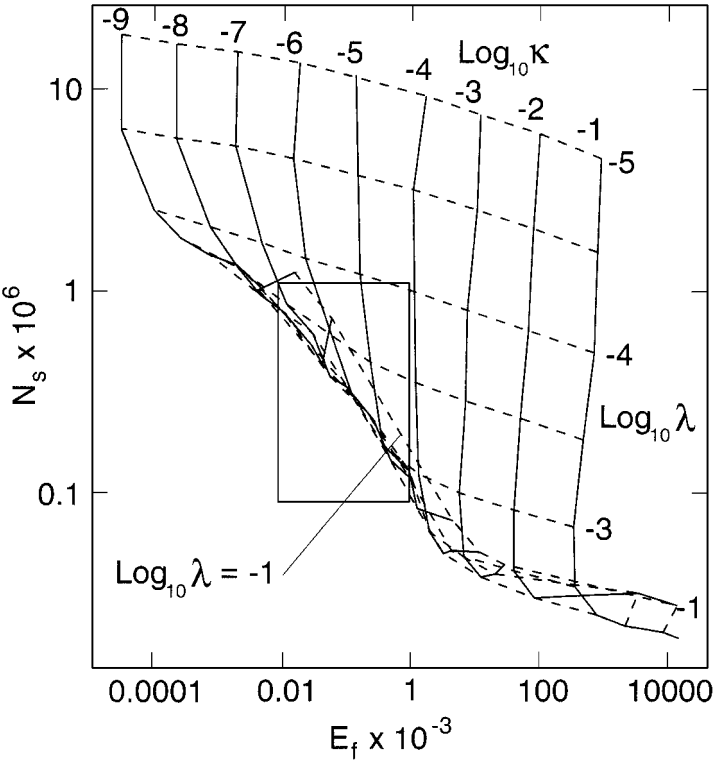

FIG. 2. SA characteristics. The dependence of Lam-enhanced SA's serial performance on the algorithmic parameters λ and κ. The solid and dashed lines in this figure connect sets of annealing runs performed with common values of κ and λ, respectively, as shown. Each point of intersection of a solid and dashed line represents the results of a set of 100 annealing runs with the indicated values of κ and λ. The point is plotted at a location that indicates the average number of iterations and average final energy of that annealing run as shown. The solid and dashed lines are linear interpolations between observed data points. The hollow box shows the region of the N_s–E_f plane plotted at high resolution in Fig. 3.

and $\kappa = 10^{-9}$ provide informative exceptions to this generalization. The $\kappa = 10^{-9}$ curve approaches the envelope and then leaves it, curling back toward the right. The $\lambda = 10^{-1}$ curve initially follows the envelope, but then departs from it and follows a trajectory parallel to, but above, the envelope. These two curves describe the edge of a λ–κ sheet which has been folded back on itself and projected on the E_f–N_s plane, with the "envelope" the projection of the fold.

The envelope of maximum annealing efficiency is in fact the desired serial performance curve. We note that it contains approximately three log-linear regions, each characterized by a particular power law dependency. That portion of the envelope corresponding to the range of energies obtained in parallel annealing runs lies approximately on a single linear segment (boxed), and we constructed a serial performance curve by sampling that region densely and fitting to a power law.

Figure 3 shows the result, together with the fitting function. Each data point is the average of 1000 independent annealing runs, with errors determined by bootstrapping as shown. The data is well fit by the equation

$$\log_{10} N_s = 8.396379 - 0.466807 \log_{10} E_f . \tag{9}$$

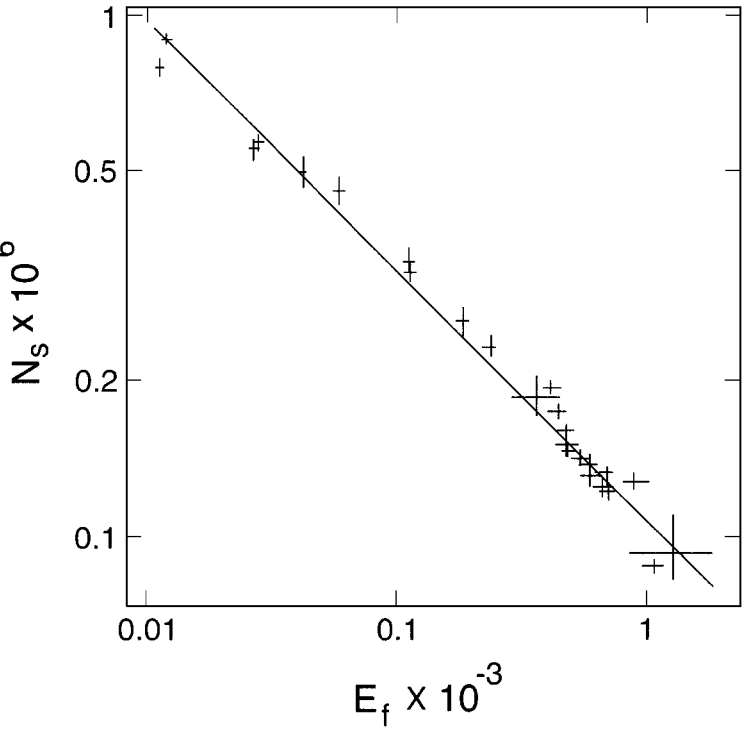

FIG. 3. The serial performance curve. The average number of iterations is shown on the vertical axis and the average final energy is shown on the horizontal axis. Each point is the average of 1000 annealing runs, with probable error determined by bootstrapping, as shown. The solid line represents the results of a power law fit to this data (see text for coefficients).

With this equation in hand, we know that if a parallel run requires N_P iterations to run on P processors and finishes with a final energy E_P, then the speedup S is given by

$$S = N_s(E_P)/N_P. \tag{10}$$

We now have all the tools required to analyze the results of parallel annealing runs.

4.2.3. *The Parallel Algorithm Performance*

We performed parallel annealing at a variety of mixing intervals m. In all parallel runs, $\kappa = 10^{-5}$ and $\lambda = 10^{-3}$, which causes N_s and E_f to lie on the appropriate part of the serial performance curve. Because the initial set of decorrelating moves is not performed in parallel, we do not count it when evaluating N_P. Both E_f and N_P varied with m, as shown in Table I which gives the dependence of E_f and N_P for 50 processors. N_P falls with increasing m until $m \approx 0.5P$, whereupon it slowly rises to over 3 times the minimum at $m = 500$. At small m, E_f is larger than in the serial case and at first rises to a value about 4 times larger than the serial value at $m \approx 0.4P$. As m increases further, E_f gradually decreases, and above $m \approx P$, it becomes less than the serial value. This behavior is generic to all P but becomes more pronounced as P increases. $P = 50$ is shown because studies were made over a larger range of m than $P = 100$.

TABLE I
Mixing Data for 50 Processors

m	$E_f \times 10^{-3}$	N
10	2.96078	3084
20	1.45944	2900
25	0.918273	2942
30	0.585394	2911
35	0.716595	3209
40	0.661397	3438
50	0.499726	3706
100	0.335584	4870
200	0.276520	6591
500	0.302882	10070
Serial	0.417433	182322

Note. See text for details.

This interplay of E_f and N_P results in sharply peaked S as a function of m, as shown in Fig. 4. The decline in E_f with increasing m means that, although N_P is at a minimum when $m \approx 0.5P$, the speedup S is largest at $m \approx 0.6P$. Using this empirical observation to tune the algorithm, we constructed the speedup curve shown in Fig. 5. This figure shows that the algorithm provides parallel speedup at 100% parallel efficiency at up to 50 processors, and approximately 80% efficiency at 100 processors.

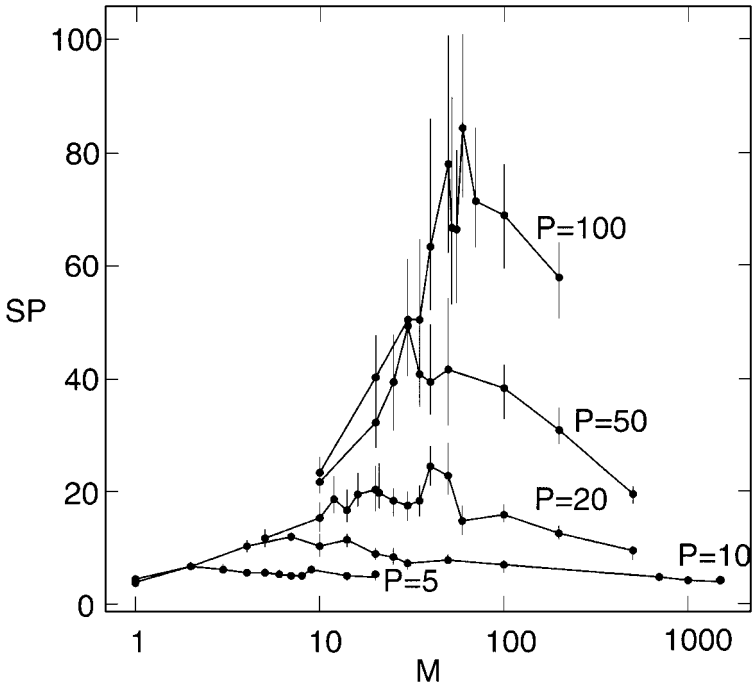

FIG. 4. Mixing behavior. The mixing interval m is shown logarithmically on the horizontal axis and the quality-corrected speedup SP with probable error on the vertical axis. Each curve shows SP as a function of m for the indicated number of processors P. Each point is the average of 100 runs.

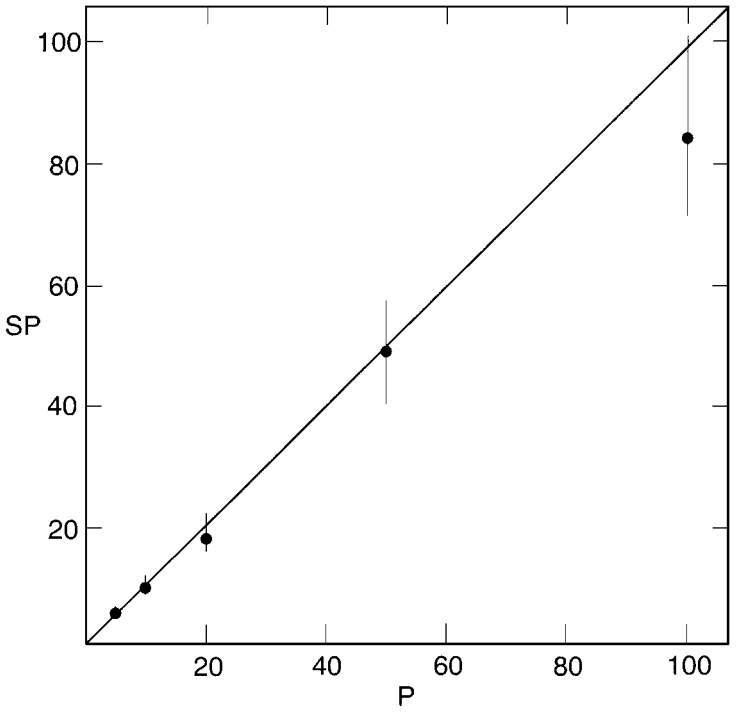

FIG. 5. Speedup. This curve shows the speedup SP with probable error as a function of the number of processors P, while keeping the mixing interval at the optimal value $m = 0.6P$.

5. DISCUSSION

We have demonstrated that our new SA parallelization approach can lead to nearly 100% parallel efficiency for optimization problems with a strictly nonseparable cost function. All prior parallelization methods, including the powerful asynchronous parallel SA, fail to converge to a solution to this problem. We have neglected the communication costs in our performance analysis of sample runs because, in realistic applications of the gene circuit problem the time of evaluating the cost function is much larger than that for the communication needed for pooling statistics, which makes communication time indeed negligible.

We have also characterized the annealing behavior of a particular problem and shown how to measure parallel efficiency accurately in optimization problems involving continuous search space.

In summary, optimization problems can be divided into two groups for parallelization according to the relative costs of computing the cost function versus communication for pooling statistics and mixing states. Group 1 is for problems with large ratio of computation to communication. These problems can be solved using the algorithm presented here.

Group 2 is for problems with a lesser or equal ratio of computation to communication such as the TSP, for which the communication must be done at coarser scale. Good performance on group 2 problems is largely a matter of tuning the code to perform well on a particular set of hardware in the context of a specific problem. Here the challenge does not include the fundamental method of parallelizing SA. Instead, it is the construction of a new kind of adaptive SA in which communication schedules, as well as temperature schedules, are adaptively determined.

ACKNOWLEDGMENTS

This work was supported by Grant DMS9727177 from the U.S. NSF, by Grant RO1-RR7801 from the U.S. NIH, and Grant N00014-97-1-0422 from the U.S. ONR. J.R. and Y.D. thank D. H. Sharp and R. Peierls for helpful discussions. We thank M. Mezei for helpful comments on the manuscript. This is publication No. 260 of the Brookdale Center for Molecular and Developmental Biology and SUNYSB-AMS-98-04 of the University at Stony Brook.

REFERENCES

1. E. Aarts and P. van Laarhoven, Statistical cooling algorithm: A general approach to combinatorial optimization problems, *Philips J. Res.* **40**, 193 (1985).
2. E. H. L. Aarts, F. M. J. de Bont, E. H. A. Habers, and P. J. M. van Laarhoven, A parallel statistical cooling algorithm, in *Proceedings of the Symposium on the Theoretical Aspects of Computer Science, 1986*, Vol. 210, p. 87.
3. R. Azencott, Sequential simulated annealing: Speed of convergence and acceleration techniques, in *Simulated Annealing: Parallelization Techniques* (Wiley, New York, 1992), p. 1.
4. R. Frost and P. Heineman, Simulated annealing: A heuristic for parallel stochastic optimization, Technical report, San Diego Supercomputer Center, 1997. [Submitted to Proceedings of PDPTA '97, URL http://www.sdsc.edu/frost/Publications/sahpso.ps]
5. R. Frost, N. Priebe, and P. Salamon, An ensemble-based simulated annealing library for multiprocessors, in *ISUG-93, St. Louis, MO*, edited by N. Branan (Intel Supercomputer Systems, Portland OR, 1993), p. 251.
6. S. Geman and D. Geman, Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images, *IEEE Trans. Pattern Anal. Mach. Intell.* **6**, 721 (1984).
7. D. R. Greening, Parallel simulated annealing techniques, *Physica D: Nonlinear Phenomena* **42**, 293 (1990).
8. M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, An efficient general cooling schedule for simulated annealing, in *Proceedings, IEEE International Conference on Computer Aided Design, 1986*, p. 381.
9. L. Ingber, Very fast simulated reannealing, *Math. Comput. Modelling* **12**, 967 (1989).
10. L. Ingber, Adaptive simulated annealing (asa): Lessons learned, *Control Cybern.* **25**, 33 (1996).
11. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by simulated annealing, *Science* **220**, 671 (1983).
12. J. Lam and J.-M. Delosme, *An Efficient Simulated Annealing Schedule: Derivation*, Technical Report 8816, Electrical Engineering Department, Yale, New Haven, CT, September 1988.
13. J. Lam and J.-M. Delosme, *An Efficient Simulated Annealing Schedule: Implementation and Evaluation*, Technical Report 8817, Electrical Engineering Department, Yale, New Haven, CT, September 1988.
14. E. L. Lawler, *The Traveling Salesman Problem* (Wiley, New York, 1986).
15. P. A. Lawrence, *The Making of a Fly* (Blackwell Scientific, Oxford, UK, 1992).
16. W. G. Macready, A. G. Siapas, and S. A. Kauffman, Criticality and parallelism in combinatorial optimization, *Science* **271**, 56 (1996).
17. N. Metropolis, A. Rosenbluth, M. N. Rosenbluth, A. Teller, and E. Teller, Equation of state calculations by fast computing machines, *J. Chem. Phys.* **21**, 1087 (1953).
18. E. Mjolsness, D. H. Sharp, and J. Reinitz, A connectionist model of development, *J. Theor. Biol.* **152**, 429 (1991).
19. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing* (Cambridge Univ. Press, Cambridge, 1988).
20. J. Reinitz, D. Kosman, C. E. Vanario-Alonso, and D. Sharp, Stripe forming architecture of the gap gene system, *Develop. Genetics* **23**, 11 (1998).
21. J. Reinitz, E. Mjolsness, and D. H. Sharp, Cooperative control of positional information in *Drosophila* by *bicoid* and maternal *hunchback*, *J. Exp. Zool.* **271**, 47 (1995).
22. J. Reinitz and D. H. Sharp, Mechanism of formation of eve stripes, *Mech. Develop.* **49**, 133 (1995).
23. J. Reinitz and D. H. Sharp, Gene circuits and their uses, in *Integrative Approaches to Molecular Biology*, edited by J. Collado, B. Magasanik, and T. Smith, Chap. 13, p. 253 (MIT Press, Cambridge, MA, 1996).

24. P. Roussel-Ragot and G. Dreyfus, A problem-independent parallel implementation of simulated annealing: Models and experiments, *IEEE Trans. Computer-Aided Design* **9**, 827 (1990).
25. D. H. Sharp and J. Reinitz, Prediction of mutant expression patterns using gene circuits, *BioSystems* **47**, 79 (1998).
26. T. Slezak, Automated integration of genomic physical mapping data via parallel simulated annealing, Lawrence Livermore National Laboratory, 1994. [URL ftp://humpty.llnl.gov/pub/supercomp94.text.ps.Z]
27. H. Szu and R. Hartley, Fast simulated annealing, *Phys. Lett. A* **122**, 157 (1987).