[6] Atomic Simulations of Protein Folding, Using the Replica Exchange Algorithm

By HUGH NYMEYER, S. GNANAKARAN, and ANGEL E. GARCÍA

Introduction

Molecular dynamics simulations of biomolecules are limited by inadequate sampling and possible inaccuracies of the semiempirical force fields. These two limitations cannot be overcome independently. The best way to optimize force fields is to validate against experiments such that simulation results are consistent with experimental data in a broad range of system's; however, validation cannot be done in the absence of adequate sampling. The energy landscape of biomolecular systems is rough. It contains many energetic barriers that are much larger than thermal energies, and these large barriers can trap biomolecular systems in local minima.[1,2] Overcoming this sampling problem is a crucial step needed to advance the field of biomolecular simulations. New techniques have been developed to enhance sampling and speed the determination of kinetic and equilibrium properties in these types of rough landscapes.[3] These techniques fall into several classes, some designed to study kinetic properties and some designed to study equilibrium properties. Many of these techniques are well suited for use on the parallel clusters common today and with distributed computing schemes.[4–6] In this chapter we provide a tutorial on the use of the replica exchange molecular dynamics (REMD) method in the context of biomolecular simulation. To place this method in context, we describe briefly some of the different methods that are being applied to sample protein conformational space.

One of the oldest methods to enhance the calculation of static properties is umbrella sampling.[7] In the umbrella sampling method a biasing potential is added to the system to enhance sampling in regions of high energy or away from equilibrium, and the sampled configurations are corrected to

[1] J. N. Onuchic, Z. Luthey-Schulten, and P. G. Wolynes, *Annu. Rev. Phys. Chem.* **48,** 545 (1997).
[2] J. N. Onuchic, H. Nymeyer, and A. E. García, *Adv. Protein Chem.* **53,** 87 (2000).
[3] S. Gnanakaran, H. Nymeyer, J. J. Portman, K. Y. Sanbonmatsu, and A. E. García, *Curr. Opin. Struct. Biol.* **13,** 168 (2003).
[4] M. R. Shirts and V. S. Pande, *Phys. Rev. Lett.* **86,** 4983 (2001).
[5] D. K. Klimov, D. Newfield, and D. Thirumalai, *Proc. Natl. Acad. Sci. USA* **99,** 8019 (2002).
[6] M. R. Shirts and V. S. Pande, *Science* **290,** 1903 (2000).
[7] G. Torrie and J. Valleau, *J. Comp. Phys.* **23,** 187 (1977).

METHODS IN ENZYMOLOGY, VOL. 383
0076-6879/04 $35.00

account for this bias. Usually several separate simulations are carried out, each with a different biasing potential. This method can be combined with the weighted histogram analysis method[8,9] to describe the equilibrium free energy of a system in terms of order parameters describing the system's structural properties, such as radius of gyration and number of native contacts. In the context of protein folding, this method has been most extensively utilized by Brooks and co-workers.[10–13] The umbrella sampling method parallelizes trivially because multiple simulations can be performed independently, and each calculation can be biased to selectively sample a different region of the free energy map.

A less obvious method of umbrella sampling is to use a biasing potential that is solely a function of the potential energy, the multicanonical potential function. Determining this bias self-consistently so that all potential energies are equally sampled allows the system to make a random walk in potential energy space and surmount large enthalpic barriers. This method, the multicanonical sampling method, was invented by Berg and Neuhaus[14] to enhance sampling across discontinuous phase transitions. It was first applied to peptides by Hansmann and Okamoto.[15] Although widely used, the determination of the biasing function is difficult, especially for systems with explicit solvent. The self-consistent determination of the biasing function limits the application of this method, although iterative procedures have been proposed to speed this process.[16]

Highly parallel methods have also been used to enhance the determination of protein kinetics. The simplest parallel sampling method is to run many uncoupled copies of the same system under different initial conditions.[17] The massive parallelism inherent in this method has been useful in projects such as Folding@Home,[6] which uses the excess compute cycles of weakly coupled private computers. This simple parallel simulation method is most successful in systems with implicit solvent, in which the slowest relaxation rates are increased.[18,19]

[8] A. M. Ferrenberg and R. H. Swendsen, *Phys. Rev. Lett.* **63,** 1195 (1989).
[9] S. Kumar, D. Bouzida, R. H. Swendsen, P. A. Kollman, and J. H. Rosenberg, *J. Comp. Chem.* **13,** 1011 (1992).
[10] E. M. Boczko and C. L. Brooks III, *Science* **269,** 393 (1995).
[11] C. L. Brooks III, *Acc. Chem. Res.* **35,** 447 (2002).
[12] J. E. Shea and C. L. Brooks III, *Annu. Rev. Phys. Chem.* **52,** 499 (2001).
[13] J. E. Shea, J. N. Onuchic, and C. L. Brooks III, *Proc. Natl. Acad. Sci. USA* **99,** 16064 (2002).
[14] B. Berg and T. Neuhaus, *Phys. Lett. B* **14,** 249 (1991).
[15] U. Hansmann and Y. Okamoto, *J. Comp. Chem.* **14,** 1333 (1993).
[16] T. Terada, Y. Matsuo, and A. Kidera, *J. Chem. Phys.* **118,** 4306 (2003).
[17] I. C. Yeh and G. Hummer, *J. Am. Chem. Soc.* **124,** 6563 (2002).
[18] P. Ferrara, J. Apostolakis, and A. Caflisch, *Proteins* **46,** 24 (2002).
[19] M. Shen and K. Freed, *Biophys. J.* **82,** 1791 (2002).

A more sophisticated method to enhance the time-domain sampling problem is the parallel replica method.[20] In this method independent simulations are started from the same conformational basin. When one simulation exits the basin, all the other simulations are restarted with random velocities from the new basin position. In the ideal case that barrier crossing is fast and waiting times are exponential, this method yields a linear speed-up in the rates of conformational transitions.[4] A rigorous application of this method to biomolecular systems has not been carried out because of the difficulty of defining transition states and assessing when they have been crossed.

The list of methods we have briefly described is not exhaustive. We have merely highlighted some of the methods that have been more widely applied to biomolecular systems.

In this chapter we focus on the description of a particular technique that has been effective for enhancing sampling in biomolecular simulations. This method, commonly referred to as the simulated tempering or replica exchange (RE) method, was invented independently on several occasions.[21–24] In this method, several copies or replicas of a system are simulated in parallel, only occasionally exchanging temperatures through a Monte Carlo (MC) move that maintains detailed balance. This algorithm is ideal for a large cluster of poorly communicating processors because temperature exchanges can be relatively infrequent and require little data transfer. A parallel version of this algorithm was first proposed in 1996.[24] The first application of this algorithm to a biological system was a study of Met-enkephalin.[25] It was adapted for use with molecular dynamics and named the replica exchange molecular dynamics (REMD) method.[26] This REMD method has numerous advantages. It is easy to implement and requires no expensive fitting procedures; it produces information over a range of temperature; and it works well on systems with explicit solvent as well as implicit solvent. A comparison of this algorithm with constant temperature molecular dynamics applied to peptides at room temperature showed that this algorithm decreased the sampling time by factors of 20 or more.[27] REMD has been used to study the equilibrium of protein folding and binding in models using explicit[27–30] and implicit[31–34] solvent. REMD

[20] A. Voter, *Phys. Rev. B* **57,** 13985 (1998).
[21] R. Swendsen and J. Wang, *Phys. Rev. Lett*. **57,** 2607 (1986).
[22] C. Geyer and E. Thompson, *J. Am. Stat. Assoc*. **90,** 909 (1995).
[23] E. Marinari and G. Parisi, *Europhys. Lett*. **19,** 451 (1992).
[24] K. Hukushima and K. Nemoto, *J. Phys. Soc. Jpn*. **65,** 1604 (1996).
[25] U. Hansmann, *Chem. Phys. Lett*. **281,** 140 (1997).
[26] Y. Sugita and Y. Okamoto, *Chem. Phys. Lett*. **314,** 141 (1999).
[27] K. Y. Sanbonmatsu and A. E. García, *Proteins* **46,** 225 (2002).

has been extended to include constant pressure,[35] extended to sample in the grand canonical ensemble,[31] and combined with other enhanced sampling methods.[36–39]

In the RE and REMD methods, all copies of the peptide system are identical except for temperature. Temperature exchanges are attempted at specified intervals of time. These exchanges allow individual replicas to bypass enthalpic barriers by moving to high temperature. Although the RE method enhances sampling, there currently exists no rigorous method for extracting information about folding kinetics from its application. In most instances we must resort to the energy landscape theory to compute kinetics.[40] These calculations require a measure of the configurational diffusion coefficient, which depends on the order parameters used to describe the energy surface. Lattice[41] and off-lattice[42] folding simulations on minimalist models have shown that the energy landscape theory describes the kinetics for folding and unfolding to about an order of magnitude, which is remarkable considering the many orders of magnitude that such rates can span.

We briefly outline the REMD method and discuss details necessary for applying REMD effectively to biological systems.

Replica Exchange Molecular Dynamics

We motivate the replica exchange method as a special type of Monte Carlo move for parallel umbrella sampling.

Umbrella sampling usually begins with the simulation of N copies or replicas of the system. Each replica has a unique potential chosen to enhance the sampling of some region of phase space. These simulations can be subsequently combined to "fill in" regions of phase space that would

[28] S. Gnanakaran and A. E. García, *Biophys. J.* **84,** 1548 (2003).
[29] A. E. García and K. Sanbonmatsu, *Proteins Struct. Funct. Genet.* **42,** 345 (2001).
[30] R. Zhou, B. J. Berne, and R. Germain, *Proc. Natl. Acad. Sci. USA* **98,** 14931 (2001).
[31] M. Fenwick and F. Escobedo, *Biopolymers* **68,** 160 (2003).
[32] R. Zhou and B. Berne, *Proc. Natl. Acad. Sci. USA* **99,** 12777 (2002).
[33] A. Mitsutake and Y. Okamoto, *Chem. Phys. Lett.* **332,** 131 (2000).
[34] A. Mitsutake and Y. Okamoto, *J. Chem. Phys.* **112,** 10638 (2000).
[35] T. Okabe, Y. Okamoto, M. Kawata, and M. Mikami, *Chem. Phys. Lett.* **335,** 435 (2001).
[36] Y. Sugita, A. Kitao, and Y. Okamoto, *J. Chem. Phys.* **113,** 6042 (2000).
[37] Y. Sugita and Y. Okamoto, *Chem. Phys. Lett.* **329,** 261 (2000).
[38] A. Mitsutake, Y. Sugita, and Y. Okamoto, *J. Phys. Chem.* **118,** 6676 (2003).
[39] A. Mitsutake, Y. Sugita, and Y. Okamoto, *J. Phys. Chem.* **118,** 6664 (2003).
[40] J. Bryngelson and P. G. Wolynes, *J. Phys. Chem.* **93,** 6902 (1989).
[41] N. D. Socci, J. N. Onuchic, and P. G. Wolynes, *J. Chem. Phys.* **104,** 5860 (1996).
[42] N. Hillson, J. N. Onuchic, and A. E. García, *Proc. Natl. Acad. Sci. USA* **96,** 14848 (1999).

otherwise be poorly sampled. In general, replica $i \in [1, N]$ has a unique potential $U_i(\mathbf{x}_i)$, which is a function of its coordinates $\mathbf{x}_i$. In many instances, $U_i(\mathbf{x}_i)$ is the sum of a generic molecular dynamics potential, which is the same for all replicas, and a restraint potential, which is unique to each replica.

More general potentials can be used for replicas other than this. For example, it is often useful to simulate one system at many different temperatures and to combine the results. In these simulations each replica shares the same potential $U(\mathbf{x}_i)$, but has its own temperature T_i. Because the potential energy and temperature always enter in the ratio $U(\mathbf{x}_i)/RT_i$ in MC, this is equivalent (in coordinate space) to simulating the N replicas at the same unit temperature but at different scaled potentials $E_i(\mathbf{x}_i) = U(\mathbf{x}_i)/RT_i$. In this sense, this is a type of umbrella sampling in temperature space.

The umbrella sampling method is ideally suited for parallel computers, because it involves the simulation of multiple noninteracting (and hence noncommunicating) replicas. This is conceptually equivalent to simulating one large system with many noninteracting subsystems. This supersystem is described by coordinates $\mathbf{x} = \mathbf{x}_1 \otimes \cdots \otimes \mathbf{x}_N$ and potential $U(\mathbf{x}) = U_1(\mathbf{x}_1) + \ldots + U_N(\mathbf{x}_N)$.

In umbrella sampling the N component replicas of the supersystem are completely isolated. This restriction is not necessary for preserving the equilibrium properties of the system. The RE method derives from the observation that adding extra types of MC moves that exchange coordinates between multiple replicas can dramatically increase the relaxation rates of the system. The new type of MC move that is added in the RE method is to randomly choose two replicas i_1 and i_2 in the supersystem and exchange all their coordinates. The change in energy from this MC move is

$$\Delta U = U_{i_2}(x_{i_1}) + U_{i_1}(x_{i_2}) - U_{i_2}(x_{i_2}) - U_{i_1}(x_{i_1}) \tag{1}$$

To preserve detailed balance this exchange is normally made with probability

$$\min\{1, \exp(-\Delta U/RT)\} \tag{2}$$

at a temperature T. R is the gas constant in units of kcal/mol/K if U has units of kcal/mol and T has units of degrees Kelvin.

We have described how multiple simulations at different temperatures can be understood as umbrella simulations. In this special case, replica i has an effective potential $E_i(x_i) = U(x_i)/RT_i$. Substituting this into Eq. (1) gives us the expression for the change in effective energy for a temperature exchange move between replicas i_1 and i_2:

$$\Delta E = U(x_{i_1})/RT_{i_2} + U(x_{i_2})/RT_{i_1} - U(x_{i_2})/RT_{i_2} - U(x_{i_1})/RT_{i_1} \quad (3)$$

$$= -(U_{i_1} - U_{i_2}) \times (1/RT_{i_1} - 1/RT_{i_2}) \quad (4)$$

$$= -\Delta U \times \Delta\beta \quad (5)$$

where ΔU is the difference in the real energy of replicas i_1 and i_2 and $\Delta\beta$ is the difference in the inverse temperatures. Detailed balance can be preserved if exchange moves are accepted with probability

$$\min\{1, \exp(\Delta U \times \Delta\beta)\} \quad (6)$$

For pedagogical reasons we have presented the RE as occurring via the exchange of all the coordinates of system i_1 with all the coordinates of system i_2 but with fixed temperatures T_{i_1} and T_{i_2}. In practice it is much simpler to fix the coordinates of systems i_1 and i_2 and exchange the temperatures instead. This requires the exchange of only one number, the temperature, instead of $3N$ numbers, the particle coordinates. It is in this sense then that we speak of replicas exchanging temperatures. Likewise, a general RE method may correctly be thought of as the exchange of potential functions rather than coordinates.

The RE method has been adapted for use with molecular dynamics.[26] Molecular dynamics is generally easier to implement in classic simulations than in MC and samples basins more efficiently. The essence of the REMD method is to use molecular dynamics to generate a suitable canonical ensemble in each replica rather than MC.

REMD normally occurs in coordinate and momentum space instead of just coordinate space. The total energy for a replica with particle masses $\mathbf{m}$, coordinates $\mathbf{x}$, and velocities $\mathbf{v}$ is now $H = \frac{1}{2}\mathbf{m} \cdot \mathbf{v}^2 + U(\mathbf{x})$. Similar to the RE method, exchanging temperatures T is equivalent to an exchange of coordinates and momenta. For this exchange, the acceptance probability would be

$$\min\{1, \exp(\Delta H \times \Delta\beta)\} \quad (7)$$

where the exchange depends on the difference in total energy ΔH between replicas rather than on just the potential energy U. If done in this way no velocity scaling is done.

This method for REMD is inefficient and should not be used. The inefficiency occurs because the total energy changes more rapidly with temperature than just the potential energy, so the acceptance ratio for RE moves decreases unless more replicas are used in the same temperature range. A better method for REMD was proposed by Sugita and Okamoto[26] and is in common use. The idea is to do a combined move: first exchange temperatures and then scale the momenta by $(T_{\text{new}}/T_{\text{old}})^{1/2}$. This scale transformation has a unit Jacobian, because one replica has its momenta scaled up by some amount and another replica has its momenta scaled down by the same amount. Consequently, this combination move is unbiased, and detailed balance may be preserved by accepting this attempted exchange between replicas i_1 and i_2 with probability

$$\min\{1, \exp(-\Delta E)\} \tag{8}$$

where ΔE is the change in effective energy:

$$\begin{aligned}\Delta E &= E_{\text{after}} - E_{\text{before}} \\ &= \left\{\frac{1}{2}\mathbf{m}\cdot\left[\sqrt{T_2/T_1}\boldsymbol{v}_1\right]^2/RT_2 + U(\mathbf{x}_1)/RT_2\right. \\ &\quad \left.+\frac{1}{2}\mathbf{m}\cdot\left[\sqrt{T_1/T_2}\boldsymbol{v}_2\right]^2/RT_1 + U(\mathbf{x}_2)/RT_1\right\} \\ &\quad -\left\{\frac{1}{2}\mathbf{m}\cdot\boldsymbol{v}_2^2/RT_2 + U(\mathbf{x}_2)/RT_2 + \frac{1}{2}\mathbf{m}\cdot\boldsymbol{v}_1^2/RT_1 + U(\mathbf{x}_1)/RT_1\right\} \\ &= \{U(\mathbf{x}_1)/RT_2 + U(\mathbf{x}_2)/RT_1\} - \{U(\mathbf{x}_2)/RT_2 + U(\mathbf{x}_1)/RT_1\} \\ &= -\Delta U \times \Delta\beta\end{aligned} \tag{9}$$

Notice that scaling the velocities exactly cancels the change in the effective energy due to the momentum part of phase space; consequently, the probability of accepting this combined temperature exchange and velocity scaling move depends as before only on the difference in potential energy between the two replicas, not their total energy.

Because (1) the temperatures of the replicas are fixed, (2) each replica is identical in all respects except the temperature (for REs in temperature), and (3) there is always one replica at each set temperature, we know that each replica must spend on average an equal amount of time at each temperature. Thus, the RE method is a way to flatten the potential of mean force (PMF) along the direction of temperature. Although temperature is most often exchanged in biological simulations, it may be useful to flatten the PMF along other coordinates (single coordinates or multiple coordinates simultaneously). In particular, the RE method may be used to approximate a multicanonical method by using umbrella restraints that restrain each replica to small overlapping windows in the potential energy. Replicas can then exchange restraints, move easily up and down in potential energy, and will spend on average an equal amount of time in each restraining window of energy.

Because each replica has a temperature that is varying in time, a direct calculation of transition or relaxation rates from the simulations cannot be made. Thermodynamic averages may be computed directly or with methods such as the weighted histogram averaging method.[8,9] To do this, the replicas must be sorted according to temperature after the simulations are complete.

The variation in temperature of any particular replica resembles that seen in the simulated annealing method[43]; however, instead of following a predetermined heating and cooling schedule, this process is self-regulated in RE methods. Replicas that are at high temperature randomly search conformational space until they find a low energy minimum. If the potential energy of this minimum is less than the potential energy of another low-temperature replica, then these two replicas will exchange temperature. Because of this, large temperature changes are slaved to conformational changes in the system. The typical temperature fluctuations and their relation to conformational changes is demonstrated from an actual REMD simulation of a solvated peptide in Fig. 1.

It is the ability of high-temperature replicas to easily cross barriers and search conformational space that makes REMD so effective. In essence, any replica can go around barriers by going up in temperature where the barriers are smaller when measured in units of RT. The flattening of barriers on the free energy surface can be illustrated by plotting the PMF along a structural coordinate and temperature simultaneously for a fully solvated peptide, GB1.[29] This type of PMF is shown in Fig. 2.

Rate enhancement can reach 10-fold or more, depending on the size of the barriers and the choice for the lowest and highest temperatures. Figure 3 shows how helical content as a function of time evolves in the same solvated helix-forming system. The increase in helical content as a function of simulation time fits well to single exponential curves with times in the 1- to 2.5-ns range. We used the last 4.0 ns of the simulation to calculate ensemble averages.

We emphasize that the temperature T that occurs in Eq. (9) is a parameter and hence constant. The temperature is not the so-called instantaneous temperature, which measures the instantaneous kinetic energy of the system and has fluctuations of magnitude $T(2/N_f)^{1/2}$, where N_f is the number of momentum degrees of freedom in the system.

The REMD method is predicated on the fact that the velocity distribution is a Maxwell–Boltzmann distribution. There are some thermostatting

[43] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Science* **220,** 671 (1983).

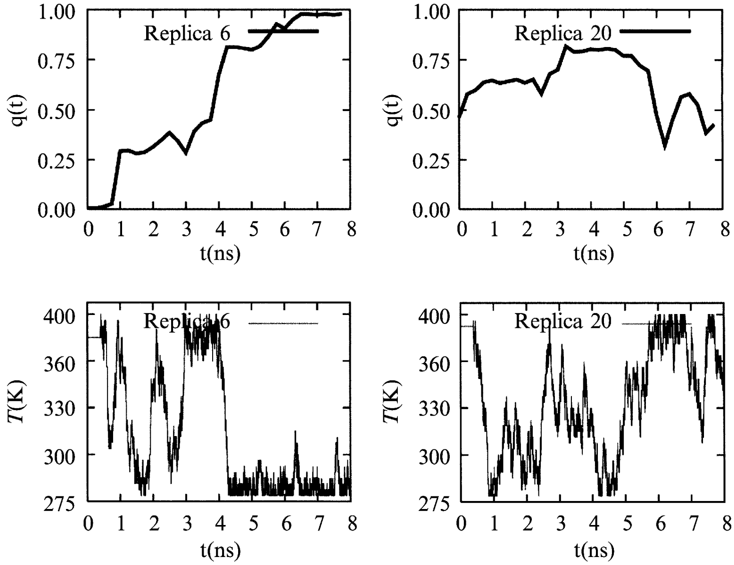

FIG. 1. The variation in helical content, q(t), and temperature for two replicas from a 48-replica REMD simulation. The system is a fully solvated 21-residue peptide (Ala_{21}) that forms a single α helix at low temperature. Notice that temperature appears to undergo a random walk for each replica. Usually, large temperature changes in the replicas are connected to structural changes, where the helical content increases (replica 6) or decreases (replica 20). In replica 6, when helical content increases, the average energy decreases, and this causes the temperature of the replica to drop.

methods that may produce a nearly canonical ensemble in coordinate space, but not in momentum space. These methods should be used with caution. If the momentum space distribution at temperature T_{low} does not map onto the momentum space distribution at temperature T_{high} under scaling all velocities by an amount $(T_{\text{high}}/T_{\text{low}})^{1/2}$, then scaling the velocities after a temperature exchange can destroy the canonical distribution in coordinate space. Thermostatting methods such as the so-called weak coupling method of Berendsen *et al.*[44] produce a momentum distribution that is correlated with the coordinate space distribution and that does not change with temperature via this scale transformation. Thus this thermostatting method when used with REMD may produce artifacts. This is

[44] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak, *J. Chem. Phys.* **81,** 3684 (1984).

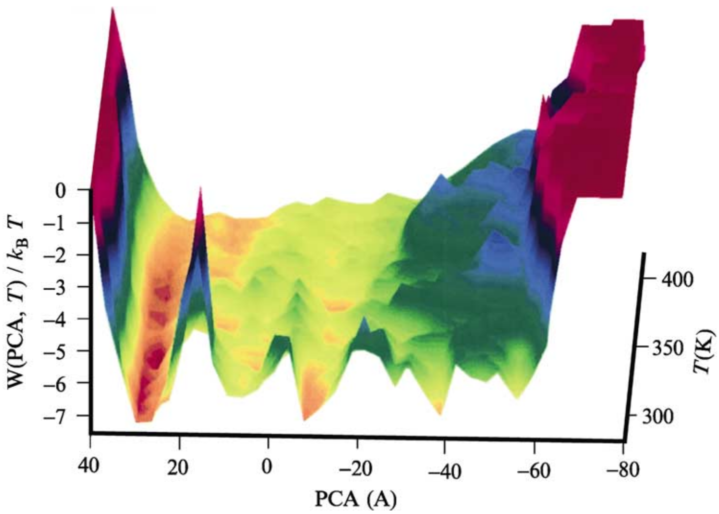

FIG. 2. Illustration of the free energy surface sampled by the REMD method as a function of a structural reaction coordinate (PCA) and temperature (T).[29] At a constant low temperature the energy landscape is rugged, with high-energy barriers separating local minima. However, replicas can move in temperature space and thereby avoid kinetic traps by moving around energetic barriers.

especially worrisome in systems with implicit solvent, which do not have a large thermal bath.

In our opinion there has not been presented in the literature a proper demonstration or "proof" that combining constant temperature molecular dynamics (MD) with MC temperature exchanges will produce a canonical ensemble over long periods of time. In the absence of such a proof, we feel that it is best to let the MD system equilibrate for some time after making replica exchange moves.

To use REMD one must specify the number of replicas, the temperature of these replicas (i.e., the temperature schedule), and the frequency for attempting temperature exchanges. The effective use of REMD depends critically on the choice of these parameters. In the following section we discuss how to choose appropriate values for these parameters.

Practical Issues

In both RE and REMD methods, there are two types of moves. There are local moves, which move each replica forward by one time step, and RE moves, which exchange temperatures or potentials between replicas.

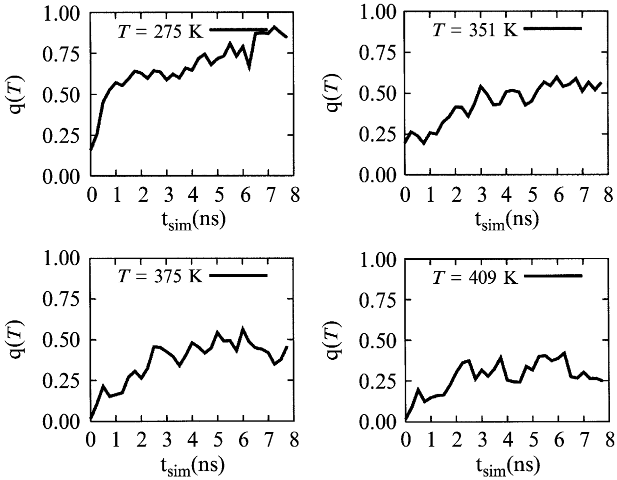

FIG. 3. Helical content (q) as a function of the replica exchange simulation time (in ns) at T = 275, 351, 375, and 409 K. Averages are calculated over a 0.25-ns time window (1000 configurations). Helical content at each temperature approaches equilibrium exponentially with equilibration times between 1 and 2.5 ns.

Although one may randomly alternate between local updates and replica exchange moves, this is not an efficient way to implement the REMD algorithm. Usually, local moves for each replica are made on a different processor. Thus, a temperature exchange between replicas requires the two processors to be synchronized with respect to where in the integration cycle they are. To keep the processors synchronized, it is better to make a fixed number of local updates for each replica followed by one or more attempted RE moves. For example, one might choose to advance each replica forward by 250 integration steps and follow this by one or more attempted temperature exchanges. Making a large fixed number of local moves between RE attempts keeps the synchronization cost to a minimum. If, however, all the replicas are running on a single processor, there is not such a large advantage with this update scheme.

The number of local moves between attempted replica moves should be fairly large. First, because RE moves require synchronization among two or more replicas, these moves are more "expensive" to make than local update moves. Making frequent RE attempts will drastically slow most simulations. Second, RE moves should be made infrequently because there are not normally large barriers to equilibration along the direction of the exchange (usually temperature), so only a few attempted moves are needed to attain a temperature equilibrium. Third, large temperature changes are slaved to conformational changes in the replicas; consequently, temperature changes are not normally rate limited by the exchange attempt frequency but by an intrinsic relaxation process in the system. Fourth, in the REMD method it has not been clearly demonstrated that molecular dynamics with all types of thermostatting will correctly generate a canonical ensemble when combined with RE moves. It is possible that for some thermostats a period of local equilibration must set in before exchanges are attempted. In our estimation, for typical molecular dynamics simulations with 1- to 2-fs time steps, making about 250 integration steps between attempted RE moves is appropriate.

An important issue for RE is the number of replicas and the choice for their potentials. In the special case of temperature this amounts to choosing the number of temperatures (number of replicas), the temperature spacing between replicas, and the minimum and maximum temperatures. These are obviously not all independent choices. The minimum temperature is usually set to be near the temperature of interest biologically, with the knowledge that sampling difficulties and simulation time will increase quickly as the minimum temperature is decreased. The maximum temperature must be set high enough so that replicas at this temperature quickly cross their largest energetic barriers and lose memory of their initial condition. For most small biological simulations this is more than 500 K or about two times larger in temperature than room temperature. The number of replicas and their temperature spacing must then be adjusted to effectively span this temperature range.

Let us discuss generally how the number of replicas needed increases with system size. In large systems the average energy is an extensive quantity, scaling linearly with system size. Also, the energy fluctuations of distant parts of large systems are uncorrelated, so the mean squared fluctuation in energy is also an extensive quantity. Consequently, doubling the system size or number of particles N ($\sim$ Volume) doubles the average energy spacing between adjacent temperatures, but only increases the energy fluctuations at a given temperature by $N^{1/2}$. Because the size of the energy fluctuations must be similar to the energy spacing between replicas for efficient exchanges to occur, the total number of replicas necessary to span a temperature range will in general increase as $N^{1/2}$, the square root of the size of the system, or Volume$^{1/2}$. This dependence ultimately limits the application of the RE method with temperature exchanges.

Simulations of systems with explicit solvent are more costly than simulations with implicit solvent because the explicitly solvated systems have a much greater number of particles. A rough estimate for the temperature spacing for replicas at room temperature that are explicitly solvated can be obtained by assuming that they are mostly water. If the temperature spacing between replicas is ΔT, then the energy spacing between replicas is approximately

$$\Delta E = C_v m \Delta T \tag{10}$$

where C_v is the coordinate space contribution to the specific heat at constant volume (and the mean temperature of the replicas) and m is the mass in grams. The root mean squared fluctuations are approximately

$$\sqrt{\Delta E^2} = \sqrt{C_v m k_B T^2} \tag{11}$$

where $k_B \approx 1.38 \times 10^{-23}$ J/K is the Boltzmann constant. We use k_B in this formula instead of R, the gas constant, because we are working in energy units of joules instead of kcal/mol. This ratio of these quantities should be of order 1, so the temperature spacing should be about

$$\Delta T = \sqrt{k_B T^2 / m C_v} \tag{12}$$

Using $C_v \approx 2.8$ J/g/K $= 4.2$ J/g/K $- 1.4$ J/g/K (approximate C_v from Ref. 45 minus an estimated momentum space contribution of $6/2 k_B$ per water molecule because vibrational modes are not significantly excited at room temperature) and assuming a cubic box 30 Å on a side with a water density of 1000 g/liter gives a mass of 27×10^{-21} g and a temperature spacing at 295 K of $\Delta T \approx 4.0$ K. This is in agreement with actual spacings which have worked successfully in solvated systems using REMD (see Table I[26–28, 46–48] and Fig. 4).

Equation (12) indicates several other useful things. First, it confirms our previous statement that the temperature spacing in a homogeneous system should decrease as $1/m^{1/2}$, where m is the mass, so the number of replicas should increase like the square root of the number of particles. Second, if a system has an approximately constant heat capacity over the span of replica temperatures, then the ideal temperature distribution (i.e., the distribution for which exchanges between neighboring replicas is approximately constant) is an exponential distribution. For example, replica $i \in [1, N]$

[45] D. R. Lide (ed.), "CRC Handbook of Chemistry and Physics," 81st Ed. CRC Press, Boca Raton, FL, 2000.
[46] A. E. García and K. Y. Sanbonmatsu, *Proc. Natl. Acad. Sci. USA* **99,** 2782 (2002).
[47] H. Nymeyer and A. E. García, *Proc. Natl. Acad. Sci. USA* **100,** 13934 (2003).
[48] A. E. García and J. N. Onuchic, *Proc. Natl. Acad. Sci. USA* **100,** 13898 (2003).

TABLE I
PARAMETERS USED IN SOME REMD SIMULATIONS CARRIED OUT ON PROTEINS AND PEPTIDES OF VARIOUS SIZES[a]

System	No. of protein atoms	No. of water molecules	ΔT at 300 K	No. of replicas	T range (K)	Ref.
Met-enkephalin, TIP3P	84	587	9	16	275–419	27
Met-enkephalin, implicit SA	84	0	56	8	200–700	26
SH3 divergent turn, TIP3P	132	903	9	24	276–469	28
A21, TIP3P	222	2640	4	48	278–500	47
A21, GB/SA	222	0	9	16	275–419	48
Fs peptide, TIP3P	282	2660	5	46	275–551	47
Fs peptide, GB/SA	282	0	9	16	200–624	48
Protein A, TIP3P	734	5107	2.2	82	277–548	49

Definitions: The explicit TIP3P water model and the implicit surface area (SA) and generalized born/surface area (GB/SA) models are described in reference 48.

[a] The systems are Met-enkephalin in explicit and implicit (surface area model) solvent, the SH3 divergent turn peptide, A21, and Fs peptides in explicit and GB/SA implicit solvent, and a 46-amino acid fragment of protein A. ΔT is the approximate temperature spacing of neighboring replicas at 300 K.

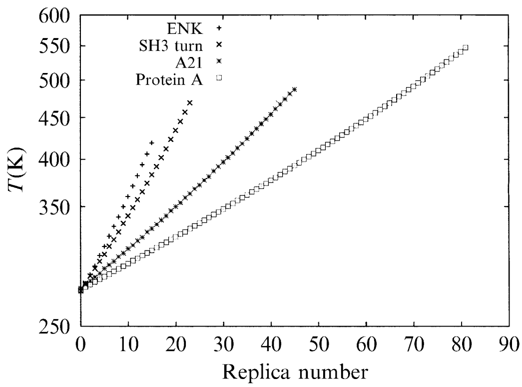

FIG. 4. A semilog plot of the temperature schedules used for various explicitly solvated systems with REMD: Met-enkephalin (ENK), the 7-residue SH3 diverging turn, a 21-residue polyalanine sequence, and protein A. In these systems, individual temperature spacings were adjusted to produce equal temperature exchange rates between neighboring replicas. This optimization produces a nearly exponential distribution of temperatures, which is optimal for systems with constant heat capacities as predicted from Eq. (12). The exponent of these distributions varies as $1/N^{1/2}$.

should have a temperature $T_i = T_1(\Delta T)^{(i-1)}$, where $\Delta T = T_2 - T_1$ is the temperature spacing of the lowest two temperatures. Third, in regions of temperature where the specific heat is larger, the spacing in temperature must be reduced. In particular, this indicates that at a first-order phase transition with a divergent heat capacity the spacing must become infinitely close. This in general indicates that the RE and REMD methods (with exchanges in temperature) do not work well across first-order transitions. However, most applications are to small systems in which finite size broadening smears out sharp specific heat peaks sufficiently that the RE and REMD methods are effective.

In the absence of detailed knowledge about the underlying specific heat, it is best to make some initial guesses about the form of the specific heat and subsequently adjust the temperature spacing of the replicas to equalize replica exchange (RE) move acceptance ratios. For explicitly solvated systems, an exponential temperature distribution is sufficient to start with; however, implicit solvent simulations tend to have a much stronger variation of specific heat with temperature. For implicit solvent simulations the higher temperatures must be spaced much closer. In small systems a uniform temperature spacing may be sufficient.

In Table I we give examples of some biological simulations that used the RE and REMD methods with both implicit and explicit solvent. Notice that for implicit solvent calculations the number of replicas is small, and the temperature steps between replicas are large. For explicit solvent models, the majority of atoms in the system are in the solvent (water). The largest REMD simulation performed to date is for a 46-amino acid fragment of protein A. This system has more than 16,000 atoms. The temperature steps at low temperature are small (2.2 K), and the number of replicas is large (82). Figure 4 shows a semilog plot of the replica temperatures as a function of replica number for the explicit solvent systems listed in Table I. The replica temperatures shown here were adjusted to give an almost constant 20% RE acceptance rate. The replica temperatures approximates an exponential distribution, but for larger systems it increase slightly faster than exponential. Exponential fits to these curves give exponents that are proportional to $1/N^{1/2}$, where N is the number of atoms in the system.

Various applications of the REMD method have been cited in text and in Table I. Among the most important results obtained from explicit solvent simulations have been the use of REMD to study the thermodynamics of helix–coil transitions in peptides that are experimentally characterized. These calculations used two different AMBER force fields (Amber 94 and Amber 96), and found that both force fields fail to reproduce the experimental observations. Use of a perturbation approach showed that a modified force field in which the bias in the backbone dihedral angles was eliminated reproduced the experimental results reasonably well.[49] These calculations showed that the use of exhaustive sampling of well-characterized systems, combined with the use of multiple force fields, can identify deficiencies in the existing force fields. In addition, it demonstrated that existing force fields are not wildly inaccurate, but that small modifications can bring them into agreement with experimental observations.

The REMD simulations have some limitations. The most obvious is the limitation in system size for models using explicit solvent. Systems with a large number of atoms will require a large number of replicas, and longer simulation times. One advantage of the trivial parallelization features of the REMD is that the method can be implemented in a dual parallelization scheme, in which each replica uses multiple processors, but such implementations may require many hundreds of processors for a large system. Other limitations encountered in common implementations of the REMD are the use of constant volume conditions. Constant volume simulations will exaggerate the hydrophobic effect at high temperatures and can lead to artificially high temperature transitions and exaggerate the amount of chain collapse.[50,51] This could be overcome by performing constant pressure calculations. However, simple water models commonly used in biomolecular simulations do not reproduce the water phase diagram far away from 300 K. Another option is to do the calculations at densities dictated by the water–vapor coexistence curve.

Use of the REMD method has been limited so far to a few systems. We expect that this method, and variations on it, will be widely used in the near future. The accessibility of low-cost computer clusters in particular will spur the use of this method.

Appendix

We describe how one may practically implement REMD, using existing molecular dynamics codes.

Most molecular dynamics codes are essentially large loops over the number of integration steps. Each integration step advances the time forward one step. After a certain number of integration steps quantities such as energy or pressure are output. To implement REMD in an existing MD code, the update steps and output steps should be followed at certain intervals by an attempted RE.

[49] S. Gnanakaran and A. E. García, *J. Phys. Chem. B* **107,** 12555 (2003).
[50] G. Hummer, S. Garde, A. E. García, A. Phorille, and L. R. Pratt, *Proc. Natl. Acad. Sci. USA* **93,** 8951 (1996).
[51] G. Hummer, L. W. Pratt, and A. E. García, *J. Phys. Chem. A* **102,** 7885 (1998).

Most MD codes today utilize parallelism through message-passing directives. It can be cumbersome to insert RE moves into such an MD code. First of all, this requires the use of different inputs. Unfortunately, data input/output (I/O) is often a nonstandard feature of message-passing standards. Second, codes often have hard-coded communication that is difficult to locate and change.

To avoid these problems, many people have come up with the idea of using a master program that goes into a loop, within which it creates multiple input files for a system that are identical except for initial conditions and temperature, spawns multiple MD programs to advance the system with these different inputs, and then extracts from the output of these programs the data needed for an RE. This method may run slowly because stopping and restarting an MD program often carries with it considerable overhead such as computing initial cell lists, generating pair exclusion lists, tabulating data for interpolating a complicated function, and doing the initial I/O.

A better method is to use a client/server arrangement. A single server receives data from multiple MD clients, makes any RE moves, and sends the new temperatures to the MD clients. Any modification to the existing MD programs is minimal, usually a few lines of code within the primary integration loop and after the I/O statements to send the potential energy and temperature to the server, receive from the server the new temperature, and then rescale the velocities and reset any temperature-dependent parameters. To avoid difficulties in adapting existing message-passing code, one can do the communication with sockets. Socket codes are robust, working on a range of UNIX and non-UNIX operating systems.

We present the source code in the C language for a rudimentary server and for a subroutine to make an existing MD code into a client able to communicate with that server. This code is provided without any warranties. The user runs it at his or her own risk.

The code consists of several subroutines with the dependencies shown as in Fig. 5. *arbiter* is the server. *arbitrate* is a subroutine called from a MC or MD program. An example MC program for a particle in a bistable potential that utilizes this code is also included. In practice, one copy of *arbiter* is started first, followed by several copies of the MC or MD program, which are informed of the Internet address and port number of the server code.

util.h

```
#ifndef __util_h_
#define __util_h_
```

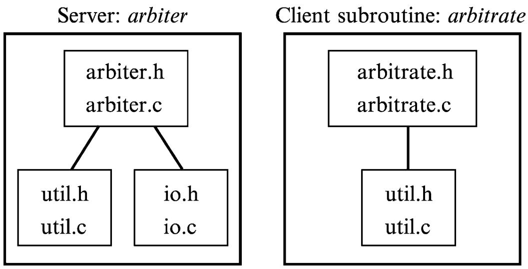

FIG. 5. Diagram of code dependencies for a server and client subroutine that implement the RE or REMD method. For example, *arbiter* can be compiled via a command: cc arbiter.c util.c io.c -o arbiter -lm. The client subroutine should be compiled to object code via a command: cc -c arbitrate.c util.c.

```
double ntohd(double x);/* convert double from network to host
   order */
double htond(double x);/* convert double from host to network
   order */
int big_endian(void);/* is this machine big_endian? */

#endif
```

util.c

```
#include "util.h"

double
ntohd(double x)
{
  static char p;
  static int i;
  static int len;

  if (big_endian() )        /* do nothing, if big_endian */
    return x;

  len = sizeof(double);    /* otherwise reverse the byte order */
  for (i = 0 ; i < len/2 ; i++) {
    p = ( (char *)&x) [i];
    ( (char *)&x) [i] = ( (char *)&x)[len - i - 1];
    ( (char *)&x) [len - i - 1] = p;
  }
  return x;
}
```

```
double
htond(double x)
{
  static char p;
  static int i;
  static int len;

  if (big_endian() )/* do nothing, if big_endian */
    return x;

  len = sizeof(double);/* otherwise reverse the byte order */
  for (i = 0;i < len/2;i++) {
    p = ( (char *)&x) [i];
    ( (char *)&x) [i] = ( (char *)&x) [len - i - 1];
    ( (char *)&x) [len - i - 1] = p;
  }
  return x;
}

int
big_endian(void)
{
  static unsigned short s = 1;
  return ( (unsigned char *)&s)[1];
}
```

io.h

```
#ifndef __io_h_
#define __io_h_
/*
  open_connections opens an array of incoming socket
  connections at the specified port number in timeout number
  of seconds
*/
int
open_connections(int *sockets,
                 int connects_wanted,
                 int port_number,
                 int timeout);
int
close_connections(int *sockets,
                  int connections_open);
#endif
```

io.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <time.h>
#include "io.h"
#include "arbiter.h"

int
open_connections(int *sockets,
                 int connects_wanted,
                 int port_number,
                 int timeout)
{
  int sin_size;
  int sock_master;
  time_t current_time;
  int connects_made=0;
  struct sockaddr_in my_address, *their_address;

  if ( (their_address =
       calloc(connects_wanted, sizeof(struct sockaddr_in) ) )
       == 0 ) {
    perror("open_connections");
    exit(1);
  }
  if ((sock_master = socket(AF_INET,SOCK_STREAM,0)) == -1) {
    perror ("socket");
    exit(1);
  }

  fcntl(sock_master, F_SETFL, O_NONBLOCK);

  my_address.sin_family = AF_INET;
  my_address.sin_port = htons(port_number);
  my_address.sin_addr.s_addr = INADDR_ANY;
```

```
    bzero(&(my_address.sin_zero),8);

    if ( bind( sock_master,
             (struct sockaddr *)&my_address,
             sizeof(struct sockaddr) ) == -1 ) {
      perror ("bind");
      exit(1);
    }

    if (listen(sock_master,BACKLOG) == -1) {
      perror ("listen");
      exit(1);
    }

    current_time = time(0);
    sin_size = sizeof(struct sockaddr_in);
    do {

      if ((sockets[connects_made] = \
          accept(
                 sock_master,
                 (struct sockaddr *) &(their_address
                   [connects_made]),
                 &sin_size
                 )
         ) != -1)
      ++connects_made;
    } while ( (time(0) < current_time + timeout) &&
             (connects_made < connects_wanted) );
    free(their_address);
    return connects_made;
}

int
close_connections(int *sockets, int connections_open)
{
    int connects_close = 0;
    int i;

    for (i = 0;i < connections_open;i++)
      connects_closed += !close(sockets[i]);
    return connects_closed=connections_open;
}
```

arbiter.h

```
#ifndef --arbiter_h
#define --arbiter_h

#define BACKLOG 10   /* buffer for waiting connections */
#define TIMEOUT 400 /* exit if conect's aren't made in this #
   seconds */
#define sq(x) ((x)*(x))

void
arbitrate(int *sockets,
          int number_of_connects);

void
collect(int *sockets,
        double *Epot,
        double *T,
        int number_of_connects);

void
rearrange(double *Epot,
          double *T,
          int number_of_connects);

void
swap(double *x, double *y);

void
distribute(int *sockets,
           double *T,
           int number_of_connects);
```

arbiter.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <math.h>
#include <time.h>
#include "io.h"
#include "util.h"
#include "arbiter.h"
```

```
/*
  This is the server code which communicates through the
  specified port number (essentially arbitrary except
  low numbers are reserved) and expects a certain number
  of clients to connect.
*/

int
main(int argc, char *argv[])
{
  int *sockets;
  int port_number;
  int numconnects;
  int connects_made;
  long seed;

  if (argc != 4) {
    fprintf(stderr,"!usage: %s port_number #_of_connections
    seed\n",argv[0]);
    exit(0);
  };

  sscanf(argv[1],"%d",&port_number);
  sscanf(argv[2],"%d",&numconnects);
  sscanf(argv[3],"%d",&seed);

  srand48(seed);

  sockets = calloc(numconnects, sizeof(int));
  if (!sockets) {
    perror("calloc");
    exit(1);
  }

  if (numconnects != (connects_made =
    open_connections(sockets, numconnects,port_number,
      TIMEOUT))) {
    perror("open_connections");
    fprintf(stderr,
        "! Only %d connections made instead of %d\n",
        connects_made, numconnects);
    exit(1);
  }

  arbitrate(sockets,numconnects);
```

```
    return 0;
  }

  void
  arbitrate(int *sockets,
            int number_of_connects)
  {
  double *Epot, *T;

  if ((Epot = calloc(number_of_connects,sizeof(double))) == 0) {
    perror("calloc");
    exit(1);
  }

  if ((T = calloc(number_of_connects,sizeof(double))) == 0) {
    perror("calloc");
    exit(1);
  }

  collect(sockets.Epot,T,number_of_connects);
  while (T[0]! = 0.0) {
    rearrange(Epot, T, number_of_connects);
    distribute(sockets, T, number_of_connects);
    collect(sockets, Epot, T, number_of_connects);
  }

  close_connections(sockets,number_of_connects);
  free(Epot);
  free(T);
}

void
collect(int *sockets,
        double *Epot,
        double *T,
        int number_of_connects)
{
  static int i;
  static int recv_size;

  for (i = 0;i < number_of_connects;i++) {
    recv_size = recv(sockets[i],Epot + i,sizeof(double),0);
    if (recv_size! = sizeof(double)) {
        perror("recv");
        exit(1);
```

```
        }
        Epot[i] = ntohd(Epot[i]);
    }

    for (i = 0;i < number_of_connects;i++) {
        recv_size = recv(sockets[i],T + i,sizeof(double),0);
        if (recv_size! = sizeof(double)) {
            perror("recv");
            exit(1);
        }
        T[i] = ntohd(T[i]);
    }
}

void
rearrange(double *Epot,
          double *T,
          int number_of_connects)
{
    static int i;
    static int first,second;
    static double dE;

    for (i = 0;i < number_of_connects*number_of_connects;i++) {
        first = drand48()*number_of_connects;
        second = drand48()*number_of_connects;
        dE = Epot[first] / T[second] + \
            Epot[second] / T[first] - \
            Epot[first] / T[first] - \
            Epot[second] / T[second];
        if (dE< = 0.0 || drand48() < exp(-dE*200.0/0.397441))
          swap(T + first,T + second);
    }
}

void
swap(double *x, double *y)
{
    static double z;
    z = *x;
    *x = *y;
    *y = z;
}
```

```
void
distribute(int *sockets,
           double *T,
           int number_of_connects)
{
    static int i;
    for (i = 0;i < number_of_connects;i++) {
    T[i] = htond(T[i]);
    if (send(sockets[i],T + i,sizeof(double),0)! = sizeof
      (double)) {
       perror("send");
       exit(1);
    }
  }
}
```

arbitrate.h

```
#ifndef __arbitrate_h_
#define __arbitrate_h_

int
client_open(char* addr_string,
            int port_number);

/*
Subroutine arbitrate makes an attempted exchange from
the current replica with energy *E and temperature
*T through the server at the internet address
*inet1.*inet2.*inet3.*inet4 and port number given by
*port_number
*/

void
arbitrate(double *E,
          double *T,
          int *inet1,
          int *inet2,
          int *inet3,
          int *inet4,
          int *port_number);

#endif
```

arbitrate.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>      /* inet_addr function */
#include <netinet/in.h>
#include <unistd.h>         /* close function */
#include "util.h"
#include "arbitrate.h"

int
client_open(char* addr_string,
            int port_number)
{
  static int sock;
  static struct sockaddr_in dest_addr;
  static int error_flag;
  /*-------------------------*
   open an IPv4 socket of type SOCK_STREAM
  *--------------------------*/
  sock = socket(AF_INET,SOCK_STREAM,0);
  if (sock == -1) {
    perror("socket");
    exit(1);
  }
  /*------------------------ --*
        set the destination address
  *------------------------ --*/
  dest_addr.sin_family      = AF_INET;
  dest_addr.sin_port        = htons(port_number);
  dest_addr.sin_addr.s_addr = inet_addr(addr_string);
  bzero(&(dest_addr.sin_zero), 8);
  /*- ----------------------*
            connect to server
  *-------------------------*/
  error_flag = connect(sock,
                   (struct sockaddr*)&dest_addr,
                   sizeof(struct sockaddr));
  if (error_flag == -1) {
     perror("connect");
```

```
        exit(1);
    }

    /*-------------------*
       return the open socket
    *-------------------*/
    return sock;
}

void
arbitrate(double *E,
          double *T,
          int *inet1,
          int *inet2,
          int *inet3,
          int *inet4,
          int *port_number)
{
    static int is_initialized=0;
    static int sock;
    static double energy, temperature;
    static char addr_string[16]; /* max inet address + 1 */
    static int len;

    /*
       This subroutine makes an attempted exchange from
       the current replica with energy *E and temperature
       *T through the server at the internet address
       *inet1.*inet2.*inet3.*inet4 and port number
       given by *port_number

       Pointers are used to facilitate use in Fortran code.
       If this subroutine is placed in Fortran code, it may
       need a trailing underscore in the name.
    */
    /*-----------------------------*
       First time called? Then open a socket
    *-----------------------------*/
    if (!is_initialized) {
      snprintf(addr_string,16,"%d.%d.%d.%d",
           *inet1,*inet2,*inet3,*inet4);
      sock = client_open(addr_string,*port_number);
       is_initialized=1;
```

```
    }
    /*-------------- ----------- --*
         communicate with the server: send
    *----------------------------*/
    energy = htond((double)*E);
    temperature = htond((double)*T);

    len = send(sock, &energy, sizeof(double),0);
    if (len! = sizeof(double)) {
      perror("send");
      exit(1);
    }
    len = send(sock, & temperature, sizeof(double),0);
    if (len!= sizeof(double)) {
      perror("send");
      exit(1);
    }

    /*------------------------------*
       if energy=temperature=0 close socket
    *---------------------------- -*/
    if (*E == 0 && *T == 0) {close(sock); return;};

    /*------------------------------*
              receive new temperature
    *---------------------------- --*/
    len = recv(sock, &temperature, sizeof(double),0);
    if (len! = sizeof(double) ) {
      perror("recv");
      exit(1);
    }

    *T = ntohd((double)temperature);
  }
```

MC.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "arbitrate.h"

#define NSTEP_EQUIL 100
```

```
#define NSTEP_PROD 1000
#define NSTEP_SWAP 10
#define NSTEP_OUTPUT 100
double energy(double x);

/*
    A toy Monte Carlo code used to illustrate the replica
    exchange program. "inetA.inetB.inetC.inetD" must be
    the internet address of the machine on which the server is
    running. port must match the port number set by the server.
    Compile via: cc MC.c arbitrate.c util.c -o MC -lm
*/

int main(int argc, char* argv[])
{
  int i;
  doubel T,x,dx,E,dE;
  long seed;
  int inetA = 127, inetB = 0, inetC = 0, inetD = 1;
  int port = 123457;

  if (argc! = 4) {
    fprintf(stderr, "!usage: %s temperature seed
       start-pos\n",argv[0]);
    exit(1);
}

  sscanf(argv[1],"%lf",&T);
  sscanf(argv[2],"%ld",&seed); srand48(seed);
  sscanf(argv[3],"%lf",&x); E = energy(x);

  for (i = 0;i < NSTEP_EQUIL;i++) {
    do {dx = drand48()-0.5;} while (dx == -0.5);
    dE = energy(x + dx)-E;
    if (dE< = 0.0 ||drand48()< = exp(-dE/T)) {x += dx; E += dE;};
  }

  for (i = 0;i<NSTEP_PROD;i++) {
    do {dx = drand48()-0.5;} while (dx == -0.5);
    dE = energy(x + dx)-E;
    if (dE <= 0.0 || drand48()<= exp(-dE/T)) {x += dx; E += dE;};
    if (i%NSTEP_SWAP == 0)
       arbitrate(&E,&T,&inetA,& inetB,&inetC,&inetD,
     &port);
    if (i%NSTEP_OUTPUT == 0)
```

```
      printf("%10.41f %10.41f\n",T,x);
    }
    E=T=0.0;
    arbitrate(&E,&T,&inetA,&inetB,& inetC,&inetD,&port);

  return 0;
    }

  double energy(double x)
  {
  #define X_MIN 2.0

    return (x*x-X_MIN*X_MIN) * (x*x-X_MIN*X_MIN);
  }
```

[7] DNA Microarray Time Series Analysis: Automated Statistical Assessment of Circadian Rhythms in Gene Expression Patterning

By Martin Straume

Introduction

DNA microarray technology has emerged as a novel and powerful means for investigating the network architecture underlying transcriptional regulation of biological systems.[1–3] The unique advancement afforded by microarrays is the capacity to conveniently experimentally interrogate genome-wide system responsiveness in a temporally parallel manner.[4–7] This technology is not only expanding our understanding in basic scientific studies of cell and developmental biology,[8] but is also being developed to apply in clinical settings for purposes of diagnosis, prognosis, and

[1] M. Chee, R. Yang, E. Hubbell, A. Berno, X. C. Huang, D. Stern, J. Winkler, D. J. Lockhart, M. S. Morris, and S. P. Fodor, *Science* **274,** 610 (1996).

[2] A. Schulze and J. Downward, *Nat. Cell Biol.* **3,** E190 (2001).

[3] J. Ziauddin and D. M. Sabatini, *Nature* **411,** 107 (2001).

[4] M. N. Arbeitman, E. E. Furlong, F. Imam, E. Johnson, B. H. Null, B. S. Baker, M. A. Krasnow, M. P. Scott, R. W. Davis, and K. P. White, *Science* **297,** 2270 (2002).

[5] R. J. Cho, M. Fromont-Racine, L. Wodicka, B. Feierbach, R. Stearns, P. Legrain, D. J. Lockhart, and R. W. Davis, *Proc. Natl. Acad. Sci. USA* **95,** 3752 (1998).

[6] B. Futcher, *Curr. Opin. Cell Biol.* **12,** 710 (2000).

[7] I. Simon, J. Barnett, N. Hannett, C. T. Harbison, N. J. Rinaldi, T. L. Volkert, J. J. Wyrick, J. Zeitlinger, D. K. Gifford, T. S. Jaakkola, and R. A. Young, *Cell* **106,** 697 (2001).

0076-6879/04 $35.00