

Determination of normalizing constants for simulated tempering

Faming Liang*

Department of Statistics, Texas A&M University, College Station, TX 77843-3143, USA

Received 18 October 2004

Available online 16 May 2005

Abstract

In this paper, we propose to estimate the normalizing constants for simulated tempering by a modified histogram algorithm, the so-called contour Monte Carlo algorithm, and compare the efficiency of simulated tempering and parallel tempering. Our analysis reveals that simulated tempering tends to mix faster than parallel tempering at low temperature levels for simulating from complex systems. The reason why simulated tempering is better than parallel tempering is discussed at length.

© 2005 Elsevier B.V. All rights reserved.

PACS: 02.70.Tt; 02.50.Ng

1. Introduction

Suppose that we are interested in sampling from the following Boltzmann distribution:

$$\pi(\mathbf{x}) = \frac{1}{Z} \exp\{-H(\mathbf{x})/\tau\}, \quad \mathbf{x} \in \mathcal{X}, \quad (1)$$

where $Z = \int_{\mathcal{X}} \exp\{-H(\mathbf{x})/\tau\} d\mathbf{x}$ is the normalizing constant (or the partition function), τ the temperature, \mathcal{X} the phase space, and $H(\mathbf{x})$ the energy function.

*Tel.: +1 979 845 8885; fax: +1 979 845 3144.

E-mail address: fliang@stat.tamu.edu.

In this article, we assume that the distribution $\pi(\mathbf{x})$ has a rough energy landscape, where there are many local energy minima separated by high-energy barriers. Many systems have such a kind of Boltzmann distribution, such as spin glasses [1], cluster melting [2], and protein [3]. As known by many researchers, in simulation from such a kind of distribution, canonical Monte Carlo algorithms, such as the Metropolis algorithm [4] and the Gibbs sampler [5], tend to get trapped in one local minimum indefinitely, rendering the simulation ineffective. To alleviate this difficulty, generalized ensemble algorithms have been proposed in the literature, including simulated tempering [6,7], parallel tempering [8,9], and the histogram-based algorithms [10–16].

1.1. Simulated tempering

Simulated tempering is a single-chain MCMC algorithm. It works by simulating from the joint distribution

$$\pi(\mathbf{x}, t) = \pi(\mathbf{x}|t)h(t) = \frac{1}{Z_t} \exp\{-H(\mathbf{x})/t\} \frac{1}{m}, \quad t \in \{t_1, \dots, t_m\}, \quad (2)$$

where t is the temperature and $Z_t = \int_{\mathbf{x}} \exp\{-H(\mathbf{x})/t\} d\mathbf{x}$ is the normalizing constant. The temperatures form a ladder with $t_1 \geq t_2 \geq \dots \geq t_m \equiv \tau$. The distributions with high temperatures are often called hot distributions, and those with low temperatures are called cold distributions. In this paper, we follow Ref. [6] to specify a uniform distribution for t , $h(t_i) = 1/m$ for $i = 1, \dots, m$. In theory, this is not necessary. We can specify a particular distribution for t according to the problem under consideration to maximize the efficiency of the algorithm.

Although the idea, accelerating the ergodicity of simulations by simulating from distributions with flatter energy landscapes, behind the algorithm is very attractive, simulated tempering suffers from a severe difficulty in practice. The normalizing constants, Z_1, \dots, Z_m , are unknown and need to be estimated prior to simulations, where Z_i denotes the normalizing constant of the distribution $\pi(\mathbf{x}, t_i)$.

To estimate these normalizing constants, several methods have been proposed in the literature [17], including stochastic approximation, reverse logistic regression, and trial and error. The trial and error method is the promising one for hard problems. It starts with a few hot distributions, running and tuning the normalizing constants of these distributions such that each distribution is visited with an approximately equal frequency; and then a few more distributions are added to the distribution sequence, and running and tuning the normalizing constants of all distributions in the current distribution sequence such that each distribution is visited with an approximately equal frequency. This process is repeated till all distributions have been added to the sequence. This method is non-automatic and time consuming. The resulting estimates are often not accurate. The complexity of the normalizing constant determination procedure has limited the use of simulated tempering. We note that some other methods [18] have been prescribed to estimate the ratios of normalizing constants of different distributions by using samples from

each of the distributions. However, these methods are not applicable here, as there is no sample available for cold distributions before a run of simulated tempering.

1.2. Parallel tempering

Similar to simulated tempering, parallel tempering also works by simulating along a sequence of distributions. The difference is that parallel tempering is a multiple-chain MCMC algorithm, and it simulates from the joint distribution

$$\pi(\mathbf{x}_1, t_1; \dots; \mathbf{x}_m, t_m) = \prod_{i=1}^m \pi(\mathbf{x}_i, t_i) = \frac{1}{\prod_{i=1}^m Z_i} \exp\left\{-\sum_{i=1}^m H(\mathbf{x}_i)/t_i\right\}. \quad (3)$$

Thus, it avoids the problem of normalizing constant estimation which has bothered simulated tempering. Parallel tempering has been applied successfully to a variety of problems, such as spin glasses [9,19] and polymer simulations [20,21].

1.3. Histogram-based algorithms

Currently, this is an active research area in computational physics. These algorithms are usually designed for a discrete system, and aim at estimating some physical quantities, say, the density of states, of the system instead of generating equally weighted MCMC samples. A re-sampling procedure has to be applied in order to get equally weighted samples.

In this paper, we propose to estimate the normalizing constants involved in simulated tempering by a modified histogram algorithm, the so-called contour Monte Carlo (CMC) [16], and compare the efficiency of simulated tempering and parallel tempering. Our analysis reveals that simulated tempering tends to mix faster than parallel tempering at low temperature levels for simulating from complex systems.

The remaining part of this paper is organized as follows. In Section 1.1, we give a brief review for the CMC algorithm. In Section 2, we apply CMC to estimate the normalizing constants to simulate tempering and compare the efficiency of simulated tempering and parallel tempering. In Section 3, we conclude the paper with a brief discussion.

2. Contour Monte Carlo

The CMC algorithm can be briefly described as follows. Suppose that our target distribution is as given in (1). Furthermore, suppose that the phase space has been partitioned according to a chosen parameterization into m non-disjoint subregions. For example, the partition is made according to the energy, and the m non-disjoint subregions are as follows: $E_1 = \{\mathbf{x} : H(\mathbf{x}) \leq u_1\}$, $E_2 = \{\mathbf{x} : u_1 < H(\mathbf{x}) \leq u_2\}$, \dots , $E_{m-1} = \{\mathbf{x} : u_{m-2} < H(\mathbf{x}) \leq u_{m-1}\}$, and $E_m = \{\mathbf{x} : H(\mathbf{x}) > u_{m-1}\}$, where u_1, \dots, u_{m-1} are $m - 1$ specified real numbers. Let $\psi(\mathbf{x})$ be a positive function defined on the phase space with $\int_{\mathcal{X}} \psi(\mathbf{x}) d\mathbf{x} < \infty$, and $g_i = \int_{E_i} \psi(\mathbf{x}) d\mathbf{x}$. If $\psi(\mathbf{x}) \equiv 1$ and \mathcal{X} is finite, then g_i is the number of configurations contained in the subregion E_i . The algorithm consists

of several stages. Let $\hat{g}_i^{(s,k)}$ denote the working estimate of g_i at the k th iteration of the s th stage of the simulation, and let

$$\hat{\pi}^{(s,k)}(\mathbf{x}) = \frac{1}{Z_{s,k}} \sum_{i=1}^m \frac{\psi(\mathbf{x})}{\hat{g}_i^{(s,k)}} I(\mathbf{x} \in E_i) \tag{4}$$

denote the working density to be simulated from, where $Z_{s,k}$ is the partition function of the working density and $I(\cdot)$ is the indicator function. Let $\mathbf{x}^{(s,k)}$ denote a sample drawn from $\hat{\pi}^{(s,k)}(\mathbf{x})$. Starting with $s = k = 0$ and $g_1^{(0,0)} = \dots = g_m^{(0,0)} = 1$, the algorithm iterates between the following two steps.

(a) (*Sampling*). Draw sample $\mathbf{x}^{(s,k)}$ from

$$\hat{\pi}^{(s,k)}(\mathbf{x}) = \frac{1}{Z_{s,k}} \sum_{i=1}^m \frac{\psi(\mathbf{x})}{\hat{g}_i^{(s,k)}} I(\mathbf{x} \in E_i). \tag{5}$$

(b) (*Weight updating*). Set

$$\hat{g}_i^{(s,k+1)} = \hat{g}_i^{(s,k)}(1 + \delta_s) I(\mathbf{x}^{(s,k)} \in E_i) \tag{6}$$

for $i = 1, \dots, m$, where δ_s is called the weight modification factor.

The algorithm iterates till a flat histogram has been produced in the space of subregions. Then we will reset the histogram, reduce δ_s to a smaller value, set $s \leftarrow s + 1$, and proceed to the next stage simulation. The estimates of g_i 's will be passed on as initial values to the next stage; i.e., setting $\hat{g}_i^{(s+1,0)} = \hat{g}_i^{(s,K_s)}$ for $i = 1, \dots, m$, where K_s denotes the total number of iterations performed at stage s . The weight modification factor δ_1 is usually set to a large number, e.g., 1 or 2, which allows the sampler to reach all subregions very quickly even for a large system. In the following stages, it will decrease in a function like $\delta_{s+1} = \gamma \delta_s$ with $\gamma < 1$, or $\delta_{s+1} = \sqrt{1 + \delta_s} - 1$. The algorithm will run till $\delta_s < \delta_{end}$, where δ_{end} is usually a very small number, say, a number less than 10^{-8} . The sample $\mathbf{x}^{(s,k)}$ can be drawn from $\hat{\pi}^{(s,t)}(\mathbf{x})$ with a number of Metropolis–Hastings (MH) [4] steps starting with $\mathbf{x}^{(s,k-1)}$. Let κ denote the number of MH steps performed for drawing $\mathbf{x}^{(s,k)}$. The choice of κ should not affect the convergence of the algorithm. However, a good choice of κ may make the algorithm perform more stably and save computation time.

When κ is large, we have the following theorem [16] for the convergence of the algorithm.

Theorem 2.1. *As $\delta_s \rightarrow 0$ and $K_s \rightarrow \infty$, for any positive function $\psi(\mathbf{x})$ with $\int_{\mathcal{X}} \psi(\mathbf{x}) \, d\mathbf{x} < \infty$, we have*

$$\hat{g}^{(s,k)}(E_i) \longrightarrow g(E_i) = c \int_{E_i} \psi(\mathbf{x}) \, d\mathbf{x} \quad \text{in probability} \tag{7}$$

for $i = 1, \dots, m$, where c is a constant which can be determined by an additional constraint on $g(E_i)$'s, for example, one of $g(E_i)$'s or $\sum_{i=1}^m g(E_i)$ is equal to a known number.

Theorem 2.1 implies that as $\delta_s \rightarrow 0$ and $K_s \rightarrow \infty$, $\hat{g}^{(s,k)}(E_i)/\hat{g}^{(s,k)}(E_j)$ forms a consistent estimator of $g(E_i)/g(E_j)$ for all $i \neq j$.

CMC can be viewed as a generalization of the Wang–Landau algorithm [15]. If $\psi(\mathbf{x}) \equiv 1$, \mathcal{X} is finite, and the sample $\mathbf{x}^{(s,k)}$ is drawn with a single MH step, then CMC is reduced to the Wang–Landau algorithm. In this case, \hat{g}_i 's form an estimate of the density of states of the system. Since the Wang–Landau algorithm can be viewed as a special case of CMC, the above theorems should approximately hold for the Wang–Landau algorithm.

3. Estimating normalizing constants for simulated tempering

Suppose that we are interested in simulating from the distribution (1), and a sequence of distributions have been constructed as in (2). Partition the sample space of the distribution (2) according to the temperature values into the following m subregions, $E_1 = \{(\mathbf{x}, t_1) : \pi(\mathbf{x}, t_1) > 0\}, \dots, E_m = \{(\mathbf{x}, t_m) : \pi(\mathbf{x}, t_m) > 0\}$. Let $\psi(\cdot) = \exp\{-H(\mathbf{x})/t\}$ and $\kappa = 1$. A run of CMC for the distribution (2) can be summarized as follows.

CMC-simulated tempering algorithm. Let $\hat{g}^{(s,k)}(E_i)$ denote the estimate of Z_i obtained at the k th iteration of the s th stage of the simulation, and let (\mathbf{x}, t_i) denote the current state of the iteration process. Each iteration consists of the following steps:

- (a) Set $j = i - 1, i$, or $i + 1$ according to probabilities $q_{i,j}$, where $q_{i,i} = \frac{1}{3}$ for $1 \leq i \leq m$, $q_{1,2} = q_{m,m-1} = \frac{2}{3}$, and $q_{i,i+1} = q_{i,i-1} = \frac{1}{3}$ for $1 < i < m$.
- (b) If $j = i$, update \mathbf{x} for $\pi(\mathbf{x}|t_i)$ using the Metropolis algorithm or the Gibbs sampler, and set $\hat{g}^{(s,k+1)}(E_i) = \hat{g}^{(s,k)}(E_i)(1 + \delta_s)$.
- (c) If $j \neq i$, accept the temperature transition with probability

$$\min \left\{ \frac{\hat{g}^{(s,k)}(E_i)}{\hat{g}^{(s,k)}(E_j)} \exp \left[-H(\mathbf{x}) \left(\frac{1}{t_j} - \frac{1}{t_i} \right) \right] \frac{q_{j,i}}{q_{i,j}}, 1 \right\}. \tag{8}$$

If it is accepted, set $i = j$, and $\hat{g}^{(s,k+1)}(E_j) = \hat{g}^{(s,k)}(E_j)(1 + \delta_s)$. If it is rejected, set $\hat{g}^{(s,k+1)}(E_i) = \hat{g}^{(s,k)}(E_i)(1 + \delta_s)$.

It follows from Theorem 2.1 that as $\delta_s \rightarrow 0$ and $K_s \rightarrow \infty$,

$$\frac{\hat{g}^{(s,k)}(E_i)}{\hat{g}^{(s,k)}(E_j)} \rightarrow \frac{Z_i}{Z_j} \quad \text{in probability}$$

for all $i \neq j$.

In practice, the normalizing constants are not required to be estimated very accurately. So a rough estimate of Z_i 's can be obtained very quickly with the above algorithm, and then the simulation can jump to the stage of $\delta_s = 0$ for a conventional run of simulated tempering.

Now we use a simplified witch’s hat distribution [17] to illustrate the above algorithm. The simplified witch’s hat distribution is defined on a unit hypercube in d dimensions $[0, 1]^d$. The unnormalized density is equal to $1 + \beta$ on the small hypercube $[0, \alpha]^d$ and 1 elsewhere in $[0, 1]^d$. We follow Ref. [17] to set $d = 30$, $\alpha = \frac{1}{3}$, and $\beta \approx 10^{14}$ so that the probability of the peak is exactly $\frac{1}{3}$. The small hypercube is called the “peak” and the rest the “brim”. The Gibbs sampler or the Metropolis algorithm has a very hard time for this distribution. The difficulty can be understood intuitively as follows. The peak is like an atom, while the distribution in the brim is uniform. If the simulation starts in the brim, it will approximately make a random walk in the brim and, on average, it will take 2.06×10^{14} ($\approx \alpha^{-d}$) steps to reach the peak. Once it reaches the peak, it will get stuck in the peak and, on average, it will take 10^{14} ($\approx 1 + \beta$) steps to leave the peak.

Some form of heating is necessary for simulating from this distribution efficiently. We follow Ref. [17] to heat the distribution by increasing the value of α gradually, meanwhile adjusting the value of β so that the probability of the peak is α exactly. Totally, 22 α values (or temperatures) were used as shown in Table 2. These values form a geometric sequence with a constant decreasing rate. The CMC-simulated tempering algorithm was run for three stages with $\delta_1 = 10^{-5}$ and $\delta_s = \sqrt{1 + \delta_{s-1}} - 1$. As δ_i is small, $\delta_s \approx \frac{1}{2}\delta_{s-1}$. Each stage consists of about 10^7 iterations. It costs a total of 78 s CPU time on a 2.8 GHz workstation. Fig. 1(a) shows the estimates and the true values of the normalizing constants of the 22 distributions. It indicates that the estimates are rather accurate. Simulated tempering was then run for 5×10^7 iterations with $\delta_s = 0.0$. Fig. 1(b) shows the histogram of the temperatures sampled in the run. The plot indicates that the initial idea of Ref. [6] has been successfully implemented: simulated tempering has made approximately a random walk along the temperature ladder with the estimates of the normalizing constants produced by CMC. Fig. 1 shows that there is a precise correspondence

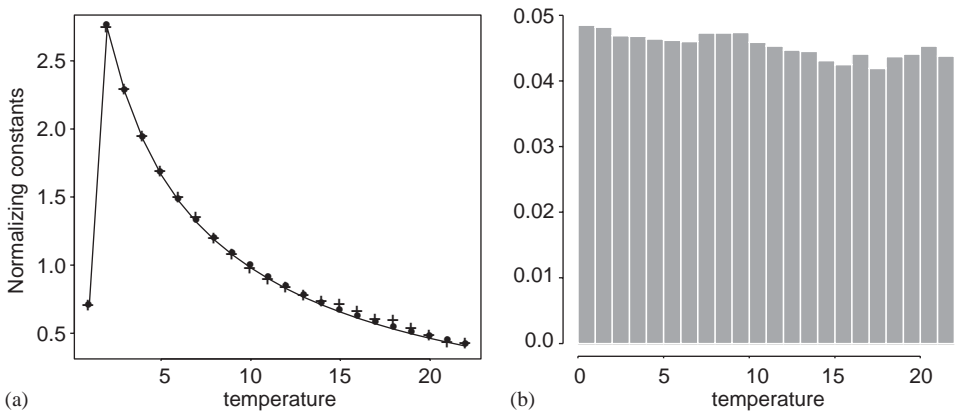


Fig. 1. Computational results for the simplified witch’s hat example. Panel (a) shows the estimated (+) and true (●) values of the normalizing constants. Panel (b) shows the histogram of temperatures sampled in a run of simulated tempering with the normalizing constants estimated by CMC.

between the estimates and the sampling frequencies. For example, Z_{15} , Z_{16} , and Z_{18} are overestimated as shown in Fig. 1(a), and the corresponding sampling frequencies are lower than the others as shown in Fig. 1(b). We note that the above method is general for estimating ratios of normalizing constants. It will work well even for the distributions of different dimensions. In this case, a proposal distribution can be designed as in Ref. [22] to accommodate the dimension difference between neighboring levels.

The following experiments compare the efficiency of simulated tempering and parallel tempering. These two algorithms are implemented as follows:

Simulated tempering.

- (a) (Local updating). Try to update \mathbf{x} once by the Metropolis algorithm.
- (b) (Temperature transition). Set $j = i - 1$, or $i + 1$ according to probabilities $q_{i,j}$, where $q_{1,2} = q_{m,m-1} = 1$, and $q_{i,i+1} = q_{i,i-1} = \frac{1}{2}$ for $1 < i < m$. Accept the transition with probability

$$\min \left\{ 1, \frac{\hat{g}(E_i) \pi(\mathbf{x}, t_j) q_{j,i}}{\hat{g}(E_j) \pi(\mathbf{x}, t_i) q_{i,j}} \right\}. \tag{9}$$

Parallel tempering. Let $\mathbf{x}_1, \dots, \mathbf{x}_m$ denote the samples at the respective temperature levels.

- (a) (Local updating). Try to update each \mathbf{x}_i once by the Metropolis algorithm.
- (b) (Exchange operation). Try to exchange each \mathbf{x}_i with its neighbors: set $j = i - 1$, or $i + 1$ according to probabilities $q_{i,j}$, where $q_{1,2} = q_{m,m-1} = 1$, and $q_{i,i+1} = q_{i,i-1} = \frac{1}{2}$ for $1 < i < m$; and accept the exchange operation with probability

$$\min \left\{ 1, \frac{\pi(\mathbf{x}_i, t_j) \pi(\mathbf{x}_j, t_i)}{\pi(\mathbf{x}_i, t_i) \pi(\mathbf{x}_j, t_j)} \right\}. \tag{10}$$

The above implementations are fair to each of the two algorithms: a local updating step is followed by a temperature transition step. The local updating steps of the two algorithms are also the same: 25% components of the current state of the Markov chain are proposed to be replaced by random numbers generated from Unif(0,1), and the proposed replacement is then accepted or rejected according to the Metropolis rule.

To compare the efficiency of these two algorithms, we consider four different temperature schemes as shown in Fig. 2. Each of the schemes is controlled by a single parameter ρ . In all the four schemes, we set $\alpha = 1$ and $\alpha_{22} = \frac{1}{3}$. In the linear scheme, the temperatures are equally spaced between α_1 and α_{22} , i.e., $\alpha_i - \alpha_{i+1} = \rho$ for $i = 1, \dots, 21$. In the geometric scheme, the temperature decreases geometrically in a constant rate, i.e., $\alpha_{i+1} = \rho \alpha_i$ for $i = 1, \dots, 21$. In the harmonic scheme, the temperatures form an equally spaced harmonic sequence, i.e., $1/(\alpha_{i+1}) - 1/\alpha_i = \rho$ for $i = 1, \dots, 21$. In the damped exponential scheme, we set $\alpha_{i+1} = \alpha_i \rho^{-0.85^i}$ for

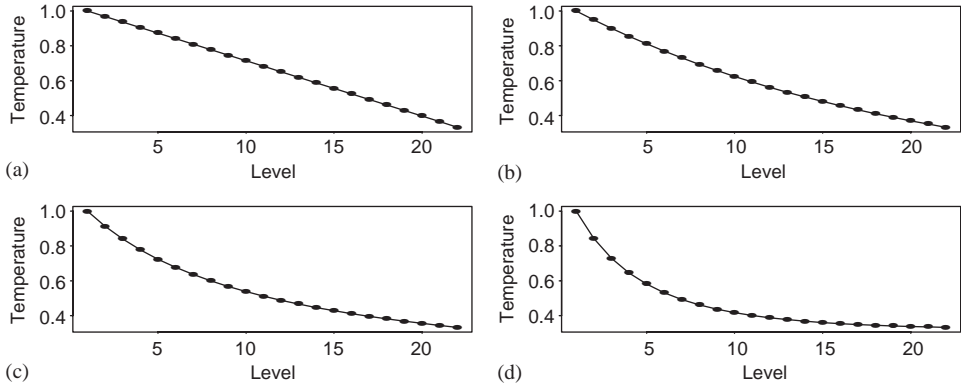


Fig. 2. Temperature decreasing schemes. The points in the graph denote different temperature values. (a) Linear scheme. (b) Geometric scheme. (c) Harmonic scheme. (d) Damped exponential scheme.

$i = 1, \dots, 21$. In the latter three temperature schemes, the spacing between the low temperatures is smaller than that between the high temperatures. Note that we often use this kind of temperature schemes in practice.

For each of the four temperature schemes, each of the two algorithms was run for 10 times independently. Totally we had 80 runs. Each run of simulated tempering consists of 2.5×10^8 iterations, and each run of parallel tempering consists of 1.135×10^7 iterations. They cost about the same CPU time. Samples were collected every 100 iterations at the temperature levels 8, 15 and 22. In each run of parallel tempering, 1.135×10^5 samples were collected at each of the three temperature levels. This is about the same for simulated tempering. Table 1 compares the integrated autocorrelation time (IAT) of the two algorithms. IAT is often used to compare the efficiency of Monte Carlo algorithms. The shorter the IAT, the larger the number of independent samples produced per CPU time unit, and the higher the efficiency of the algorithm. IAT can be estimated by λ as follows:

$$\lambda = 1 + 2 \sum_{k=1}^L c(k), \tag{11}$$

where $c(k) = \text{corr}(X_i, X_{i+k})$ is the autocorrelation of X_i and X_{i+k} , and L can be determined by the self-consistent windowing approach [23] as follows:

- (a) Determine the value of $L_1 = \min\{k : c(k) < 0\}$ and set $L = 2L_1$.
- (b) Calculate the current estimate of λ in Eq. (11).
- (c) If $L > 5\lambda$, stop; otherwise, set $L = 5\lambda$ and return to step (b).

In step (a), another reasonable choice for L is $L = L_1$, but it often leads to the value of L being too small. For example, if $c(k)$ decays slowly, then $c(L)$ could be small, but $\sum_{k>L} c(k)$ could be large. Or, chance could give a negative value of one $c(k)$ even though $c(j)$ is not very close to 0 for $j > k$. In this paper, we set $L = 2L_1$ in all computations.

Table 1
Integrated autocorrelation time (IAT) of simulated tempering and parallel tempering for different temperature schemes

Temperature scheme	Simulated tempering			Parallel tempering		
	8	15	22	8	15	22
Linear	1.020 (0.005)	2.692 (0.029)	73.401 (4.201)	1.018 (0.006)	3.255 (0.079)	105.425 (6.070)
Geometric	1.172 (0.007)	6.197 (0.092)	44.080 (2.778)	1.257 (0.016)	7.305 (0.293)	58.982 (2.826)
Harmonic	2.111 (0.023)	11.585 (0.359)	34.840 (1.013)	2.599 (0.048)	13.119 (0.818)	43.819 (2.244)
Damped exponential	13.861 (0.382)	31.603 (1.115)	38.001 (2.056)	16.311 (1.477)	37.384 (2.753)	47.020 (2.281)

The IAT and the standard deviation (the number given below in parenthesis) were calculated at the 8, 15, and 22 levels.

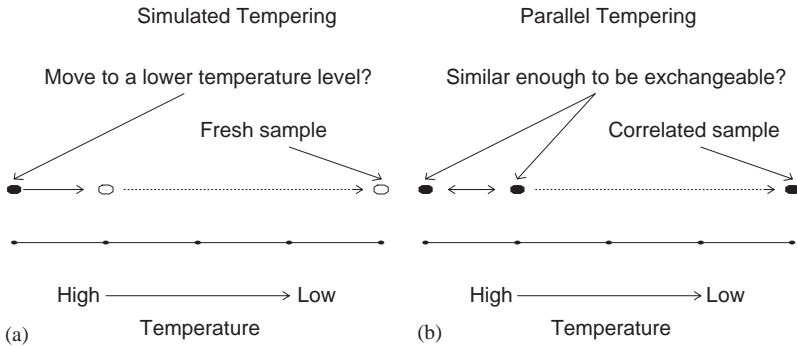


Fig. 3. Illustrative graph for the temperature jumping moves of simulated tempering (plot (a)) and parallel tempering (plot (b)).

Table 1 shows that simulated tempering produces shorter IATs than parallel tempering for all the four temperature schemes, and this advantage increases as the temperature decreases. This implies that simulated tempering is more efficient than parallel tempering for simulating from complex distributions. Here, we call a distribution complex if it has a hard time to mix in a simulation with canonical Monte Carlo algorithms. The reason why simulated tempering is more efficient than parallel tempering is illustrated by Fig. 3 and further explained as follows. First, we note that the exchange operation (operated by Eq. (10)) of parallel tempering works as a sample similarity checking mechanism. The higher the similarity of two samples, the higher the acceptance rate of the exchange operation, regardless of the temperature scheme used. In parallel tempering, when a sample moves from a high temperature level to a low one, the similarity of this sample with other samples will

be checked many times. Hence, only those samples which share a fair similarity with the low-temperature samples can survive the multiple checking and move eventually to the low temperature. (Of course, in the process that the sample moves from a high temperature level to a low one, it itself may be updated.) Hence, in parallel tempering, the Markov chain at a low temperature level will mix slowly, in other words, the sample autocorrelation will decay slowly at a low temperature level. In contrast, simulated tempering is a single-chain MCMC algorithm. It avoids the similarity checking when a sample moves from one temperature to another. At a low temperature level, all samples which come from high temperature levels are essentially independent of each other, and the sample autocorrelation decays much faster than that of parallel tempering. Note that the consecutive samples of simulated tempering should be more strongly correlated than that of parallel tempering, as the samples in simulated tempering can only be self-updated, while the samples in parallel tempering can be replaced by the samples at neighboring levels.

Fig. 4 compares the autocorrelation time of simulated tempering and parallel tempering at the 8, 15, and 22 levels. The temperature scheme used is the harmonic one, which corresponds to the best performance of these two algorithms. As analyzed above, the autocorrelation of simulated tempering samples decays much faster than that of parallel tempering samples, but the short-term autocorrelation of simulated tempering samples tends to be stronger than that of parallel tempering. Fig. 4 also shows that the relative advantage of simulated tempering increases as the temperature decreases. As the temperature decreases, it is more and more difficult for

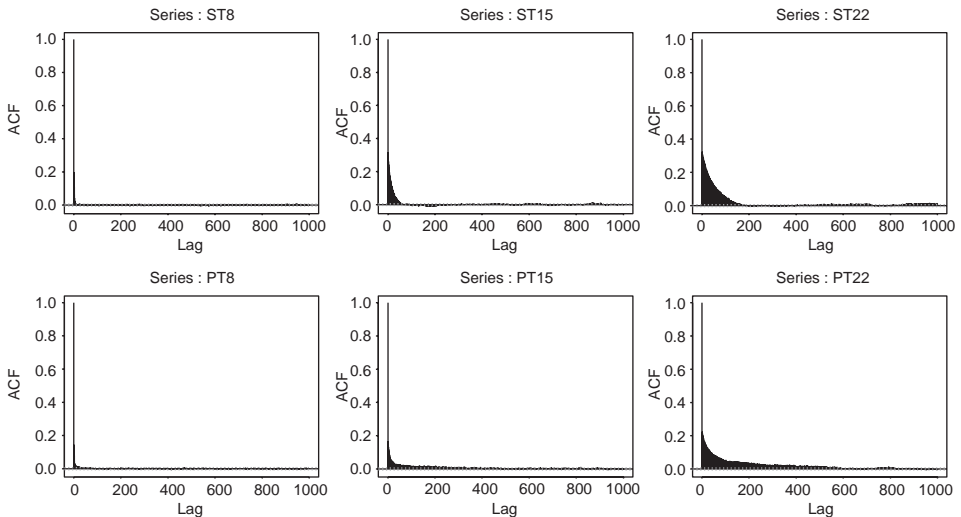


Fig. 4. Autocorrelation time of simulated tempering (upper panel) and parallel tempering (lower panel). The three graphs (from left to right) in each panel correspond to the 8, 15, and 22 levels, respectively.

Table 2
Comparison of simulated tempering and parallel tempering for the simplified witch’s hat example

<i>i</i>	α_i	Simulated tempering		Parallel tempering		Saving
		$\tilde{\alpha}_i$	SD ($\times 10^{-3}$)	$\tilde{\alpha}_i$	SD ($\times 10^{-3}$)	
1	1.000	1.000	0.000	1.000	0.000	1.000
2	0.949	0.949	0.041	0.949	0.068	2.778
3	0.901	0.902	0.115	0.901	0.147	1.627
4	0.855	0.857	0.212	0.855	0.289	1.858
5	0.811	0.813	0.297	0.811	0.480	2.613
6	0.770	0.772	0.435	0.769	0.674	2.397
7	0.731	0.732	0.608	0.729	0.979	2.589
8	0.693	0.693	0.710	0.692	1.124	2.508
9	0.658	0.656	0.980	0.656	1.196	1.490
10	0.624	0.623	1.155	0.622	1.582	1.877
11	0.593	0.591	1.438	0.590	1.971	1.880
12	0.562	0.559	1.788	0.559	2.190	1.501
13	0.534	0.530	1.897	0.531	2.767	2.129
14	0.507	0.503	2.139	0.505	2.918	1.861
15	0.481	0.479	2.260	0.475	3.069	1.844
16	0.456	0.454	2.866	0.449	3.341	1.358
17	0.433	0.431	3.651	0.429	4.074	1.245
18	0.411	0.410	4.456	0.403	5.071	1.295
19	0.390	0.393	5.067	0.381	5.923	1.366
20	0.370	0.370	5.619	0.367	6.160	1.202
21	0.351	0.355	5.881	0.346	7.125	1.468
22	0.333	0.328	6.457	0.331	7.670	1.411

The $\tilde{\alpha}_i$ is the estimate of α_i obtained by averaging over 50 runs, and SD is its standard error. The “Saving” is calculated in the formula $[\text{SD}(\text{PT})/\text{SD}(\text{ST})]^2$, which measures the saving in terms of CPU time.

the local updating moves to generate independent samples and, thus, independent samples have to be transmitted from high temperature levels.

In the second experiment, each algorithm was run with the geometric scheme 50 times independently. Each run of simulated tempering consists of 2.5×10^7 iterations. Each run of parallel tempering consists of 1.135×10^6 iterations. They cost about the same CPU time. Table 2 compares the estimates of α_i ’s produced by the two algorithms. It shows that simulated tempering is significantly better than parallel tempering, and the saving is about 40% in terms of CPU time.

The above experiments show that simulated tempering tends to mix faster than parallel tempering at low temperature levels for simulating from complex systems. We note that tuning is delicate in each of the two algorithms. For example, to maximize the efficiency of parallel tempering, we can put different strength on different chains; to maximize the efficiency of simulated tempering, we can specify some particular distribution for the temperatures in Eq. (2) to control the visiting frequency to each of the temperature levels. However, it is intrinsic that in simulating from a complex distribution, the samples of parallel tempering tend to have longer autocorrelation time than that of simulated tempering at low temperature levels.

This makes simulated tempering more attractive than parallel tempering potentially, provided that accurate estimates for the normalizing constants of simulated tempering are available.

4. Discussion

In this paper, we proposed to estimate the normalizing constants for simulated tempering using the CMC algorithm, and compared the efficiency of simulated tempering and parallel tempering. Our analysis reveals that simulated tempering tends to mix faster than parallel tempering at low temperature levels for simulating from complex systems.

In addition to estimating the normalizing constants for simulated tempering, CMC can be applied to many other problems, for example, function optimization and model selection. In function optimization, the phase space can be partitioned according to the energy function. In this case, CMC will lead to a random walk in the space of energy. Thus, it is capable of overcoming any energy barriers on the energy landscape and locating the global energy minimum eventually.

References

- [1] C. Maranas, C. Floudas, *J. Chem. Phys.* 97 (1992) 7667.
- [2] J.P.K. Doye, D.J. Wales, M.A. Miller, *I. Chem. Phys.* 109 (1998) 8143.
- [3] M. Karplus, G.A. Petsko, *Nature (London)* 347 (1990) 631.
- [4] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, *J. Chem. Phys.* 21 (1953) 1087;
W.K. Hastings, *Biometrika* 57 (1970) 97.
- [5] R. Glauber, *J. Math. Phys.* 4 (1963) 294;
M. Creutz, *Phys. Rev. D* 21 (1980) 2308;
S. Geman, D. Geman, *IEEE Trans. Pattern Anal. Machine Intell.* 6 (1984) 721.
- [6] E. Marinari, G. Parisi, *Europhys. Lett.* 19 (1992) 451.
- [7] A.P. Lyubartsev, A.A. Martinovski, S.V. Shevkunov, P.N. Vorontsov-Velyaminov, *J. Chem. Phys.* 96 (1992) 1776.
- [8] C.J. Geyer, in: *Computing Science and Statistics: Proceedings of the 23rd Symposium on the Interface*, Interface, Fairfax Station, VA, 1991, p. 156.
- [9] K. Hukushima, K. Nemoto, *J. Phys. Soc. Jpn.* 65 (1996) 1604.
- [10] B.A. Berg, T. Neuhaus, *Phys. Lett. B* 267 (1991) 291;
B.A. Berg, T. Neuhaus, *Phys. Rev. Lett.* 68 (1992) 9;
B.A. Berg, T. Celik, *Phys. Rev. Lett.* 69 (1992) 2292.
- [11] B.A. Berg, *Int. J. Mod. Phys. C* 3 (1992) 1083.
- [12] J. Lee, *Phys. Rev. Lett.* 71 (1993) 211.
- [13] B. Hesselbo, R.B. Stinchcombe, *Phys. Rev. Lett.* 74 (1995) 2151.
- [14] J.S. Wang, *Eur. Phys. J. B* 8 (1999) 287;
J.S. Wang, *Physica A* 281 (2000) 174.
- [15] F. Wang, D.P. Landau, *Phys. Rev. Lett.* 86 (2001) 2050;
F. Wang, D.P. Landau, *Phys. Rev. E* 64 (2001) 56101.
- [16] F. Liang, *J. Amer. Stat. Assoc.* (2005).
- [17] C.J. Geyer, E.A. Thompson, *J. Amer. Stat. Assoc.* 90 (1995) 909.

- [18] M.H. Chen, Q.S. Shao, J.G. Ibrahim, *Monte Carlo Methods in Bayesian Computation*, Springer, Berlin, 2000.
- [19] A. de Candia, A. Coniglio, *Phys. Rev. E* 65 (2002) 016132.
- [20] J.P. Neirotti, F. Calvo, D.L. Freeman, J.D. Doll, *J. Chem. Phys.* 112 (2000) 10340.
- [21] Q. Yan, J.J. de Pablo, *J. Chem. Phys.* 113 (2000) 1276.
- [22] F. Liang, *Phys. Rev. E* 67 (2003) 056101.
- [23] N. Madras, *Lectures on Monte Carlo Methods*, American Mathematical Society, Providence, RI, 2000.