

Statistical mechanics of program systems

Juan P Neirotti¹ and Nestor Caticha²

¹ The Neural Computing Research Group, Aston University, Birmingham B4 7ET, UK

² Departamento de Física Geral, Instituto de Física, Universidade de São Paulo, Rua do Matão, Travessa R 187, São Paulo, SP 05508-900, Brazil

Received 9 January 2006, in final form 27 June 2006

Published 2 August 2006

Online at stacks.iop.org/JPhysA/39/10355

Abstract

We discuss the collective behaviour of a set of operators and variables that constitute a program and the emergence of meaningful computational properties in the language of statistical mechanics. This is done by appropriately modifying available Monte Carlo methods to deal with hierarchical structures. The study suggests, in analogy with simulated annealing, a method to automatically design programs. Reasonable solutions can be found, at low temperatures, when the method is applied to simple toy problems such as finding an algorithm that determines the roots of a function or one that makes a nonlinear regression. Peaks in the specific heat are interpreted as signalling phase transitions which separate regions where different algorithmic strategies are used to solve the problem.

PACS numbers: 02.50.Ng, 05.10.-a, 07.05.Mh

The relation between statistical mechanics and computer science has been a central and growing topic in the last few decades. Combinatorial optimization is naturally associated with the physics of disordered systems and collective properties as described by phase transitions are intimately related to average complexity. In this paper we study the behaviour of a set of a special type of agents, operators and variables, which by way of their interactions make a program. The emergent collective behaviour of the set of agents is the output of the program. We do this with the methods and language of statistical mechanics.

The calculation of mean values is one of the most important goals of statistical mechanics. The mean value of a function of the state of the system $U(\mathbf{x})$ is a multidimensional integral $\langle U \rangle_\gamma \equiv \int d\mathbf{x} P_\gamma(\mathbf{x}) U(\mathbf{x})$, where P_γ is the probability density, γ represents the set of parameters describing the ensemble and \mathbf{x} is the state of the system. Analytical expressions for these integrals are usually impossible to derive; therefore, it is a common practice to rely on numerical techniques to calculate them. Monte Carlo (MC) simulations are frequently used for calculating multidimensional integrals numerically. The integral $\langle U \rangle_\gamma$ can be approached by the arithmetic mean value $\bar{U} \equiv \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N U(\mathbf{x}_n)$, where the sampling $\{\mathbf{x}_n\}$ has

been generated according to $P_\gamma(\mathbf{x})$. This technique has been widely used with much success in a number of systems. In all these applications the state of the system \mathbf{x} has been represented by n -tuples, with fixed type and number of entries.

However, there are systems with states that cannot be well represented by n -tuples. In the last few years, systems whose states are better represented by *hierarchical structures* have been studied in the framework of genetic programming (GP) [1]. Genetic algorithms (GA) comprise a set of techniques used to optimize a fitness function typically defined over configurations represented by ordered n -tuples, by means of natural selection [2]. GP is an extension of the GA techniques acting over a population of programs, which are hierarchical structures themselves [3–5]. It is important to note that both GA and GP aim at finding a (nearly) optimal solution for a problem, whereas MC is used to approximate numerically multidimensional integrals.

Here we present a MC technique, inspired by GP, that is suitable to calculate mean values in a program system (i.e. a system with states represented by programs). The paper is organized as follows: firstly, we will describe briefly what we mean by a program and how the MC simulations work. Afterwards we will present two toy systems used to illustrate how the MC technique works and what information we obtain from the analysis of the mean values. At the end we will present a general discussion.

In this work the word *program* is synonymous to *parsing tree*. Let \mathbb{V} be a set of variables and \mathbb{F} a set of functions. The set of all parsing trees π formed by terminals from \mathbb{V} and functions from \mathbb{F} is denoted by $\mathcal{S}_\pi(\mathbb{V}, \mathbb{F})$ (\mathcal{S}_π for short). The elements of \mathcal{S}_π are either a terminal or a function linked to an appropriate number of parsing trees. Throughout this paper we will identify parsing trees with LISP programs, for notation sake.

We have chosen to work in an ensemble obeying the Boltzmann distribution. Generalization of the method to other ensembles is straightforward. In this ensemble the probability density of the state \mathbf{x} is given by $P_\beta(\mathbf{x}) \propto \exp\{-\beta E(\mathbf{x})\}$, where $E(\mathbf{x})$ is the energy of the configuration. The parameter β (the inverse of the temperature) sets the scale of the fluctuation of the energy. Before starting with the technical details of the MC simulation we would like to clarify the notation to be used. There is a difference between the program π , which is a hierarchical structure and the value that the program returns when the input data input is fixed $[\pi](\text{input})$, which is an output value (number, logical value, vector, a program, etc).

The mean of the functional U over \mathcal{S}_π takes the form of the sum: $\langle U \rangle_\beta = \sum_{\kappa \in \mathcal{S}_\pi} P_\beta[\kappa] U[\kappa] \simeq \bar{U}_M = \frac{1}{M} \sum_{m=1}^M U[\kappa_m]$, where $\{\kappa_m\}$ is the sequence of programs generated by sampling from P_β and both $U[\kappa]$, $P_\beta[\kappa] \in \mathbb{R}$. A sufficient condition for ensuring ergodicity asymptotically is given by a detailed balance condition [6]: $P_\beta[\kappa_0] K_\beta[\kappa_n|\kappa_0] = P_\beta[\kappa_n] K_\beta[\kappa_0|\kappa_n]$, where $K_\beta[\kappa_n|\kappa_0]$ is the transition probability from state κ_0 to state κ_n . There is great freedom in choosing transition probabilities without changing equilibrium properties. In many MC approaches the conditional probability is not known and is replaced by the expression: $K_\beta[\kappa_0|\kappa_n] = T[\kappa_0|\kappa_n] \text{acc}_\beta[\kappa_0|\kappa_n]$, where $T[\kappa_0|\kappa_n]$ is called the trial probability and $\text{acc}_\beta[\kappa_0|\kappa_n]$ is an acceptance probability constructed to ensure $K[\kappa_0|\kappa_n]$ satisfies the detailed balance condition. The trial probability can be any normalized density function chosen for convenience. A common choice for the acceptance probability is given by:

$$\text{acc}_\beta[\kappa_n|\kappa_0] = \min \left\{ 1, \frac{P_\beta[\kappa_n] T[\kappa_0|\kappa_n]}{P_\beta[\kappa_0] T[\kappa_n|\kappa_0]} \right\} = \min\{1, \exp[-\beta(E[\kappa_n] - E[\kappa_0])]\}. \quad (1)$$

The Metropolis method [7], obtained from equation (1) by choosing $T[\kappa_n|\kappa_0]$ to be a uniform distribution of points of width Δ centred about κ_0 , is arguably the most widely used MC method and the basis for the approach discussed in the current work. The Metropolis

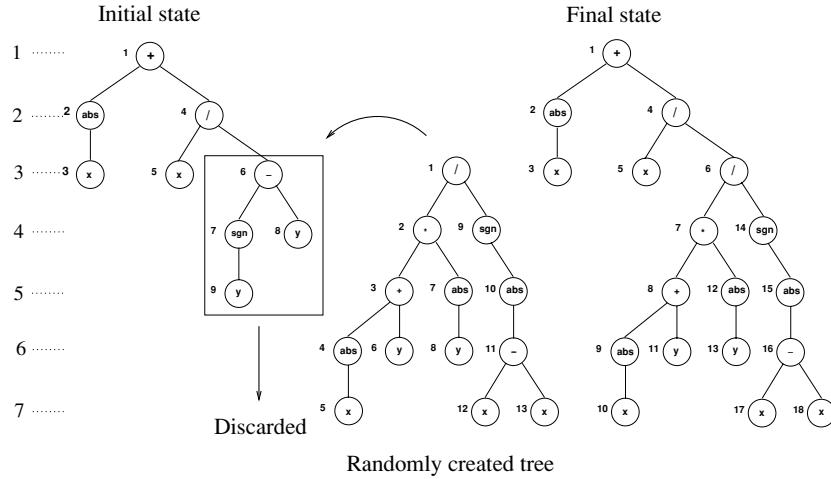


Figure 1. Example of a Metropolis step, in the parsing tree representation. It is proposed a change at level 3 (atom 6) in the initial state $(+ (\text{abs } x) (/ x (- \text{sgn } y) y))$. The change to be inserted is the program $(/ (* (+ (\text{abs } x) y) (\text{abs } y)) (\text{sgn} (\text{abs} (- x x))))$, which has been randomly created. The state proposed is the program $(+ (\text{abs } x) (/ x (/ (* (+ (\text{abs } x) y) (\text{abs } y)) (\text{sgn} (\text{abs} (- x x))))))$.

method rigorously guarantees a random walk visits the space of configurations proportional to the probability P_β asymptotically in a limit of an infinite number of steps [8].

To apply the Metropolis prescription, we need to define a measure of similarity for parsing trees (observe figure 1). The leaf at level 1 (the root of the tree) in the tree labelled *initial state* presents the + function. This function needs two arguments, so we find, at level 2, the roots of two trees. It is intuitively arguable that any change at the lower levels (leaves 3, 5 and 6 to 9 for instance) has a lower impact than a change in the upper levels (leaves 1, 2 and 4). Consider two parsing trees π' and π'' , obtained from the tree π by cutting a branch at levels d' and d'' respectively and replacing the cut branches by pasting randomly created (but syntactically correct) trees. We will assume that the tree π' is *more similar* to π than π'' if $d' > d''$. Therefore, to fix d for the cutting and pasting process is equivalent to fixing the parameter Δ for the usual Metropolis step. Although there is no analytical justification for this procedure, its application leads to reasonable results.

It is easy to realize that there are many programs that differ in detail but perform exactly the same task. Therefore, we can expect a very rough energy landscape, typical of disordered systems, neural networks or atomic clusters. This fact justifies the use of techniques such as parallel tempering MC (PTMC) [9], initially developed to study disordered systems and also used to study clusters of particles [10].

In PTMC we consider at least two Metropolis processes, one at β , the other at β' , running in parallel. At a slow rate the two processes try to interchange configurations according to detailed balance condition: $P_\beta[\kappa]P_{\beta'}[\kappa']W[\beta, \kappa \rightleftharpoons \beta', \kappa'] = P_{\beta'}[\kappa']P_\beta[\kappa]W[\beta, \kappa' \rightleftharpoons \beta, \kappa]$, where $W[\beta, \kappa \rightleftharpoons \beta', \kappa']$ is the probability that the process at β in state κ interchanges its configuration with the process at β' which is in the state κ' . A common choice is:

$$W[\beta, \kappa \rightleftharpoons \beta', \kappa'] = \min \left\{ 1, \frac{P_{\beta'}[\kappa']P_\beta[\kappa]}{P_\beta[\kappa]P_{\beta'}[\kappa']} \right\} = \min\{1, \exp[(\beta' - \beta)(E[\kappa'] - E[\kappa])]\}. \quad (2)$$

We now show the implementation of this technique in two simple examples. In the first one the states are programs that represent functions that fit a set of points $\mathbb{P} \subset \mathbb{R}^2$. We choose

the energy of the system to be represented by the sum of two competing terms. One term is proportional to the quadratic error, and the other is proportional to a roughness function. To express the latter we define the grid $\mathbb{M} = \{x_m = x_{\min} + m(x_{\max} - x_{\min})/M, m = 0, \dots, M\}$. The number of elements in the grid is set to 40. The energy of program κ is defined as:

$$h[\kappa] = \frac{1}{|\mathbb{P}|} \sum_{(x_p, y_p) \in \mathbb{P}} \{y_p - \lceil \kappa \rceil(x_p)\}^2 + \frac{1}{|\mathbb{M}|} \sum_{x_m \in \mathbb{M}} |\lceil \kappa \rceil(x_{m+1}) - \lceil \kappa \rceil(x_m)|, \quad (3)$$

where $\lceil \kappa \rceil(x)$ is the value of the function represented by the program κ in the point x . Values to be calculated are the energy $E(\beta) = \langle h \rangle_\beta$ and the specific heat $C(\beta) = \beta^2 \langle [h - E(\beta)]^2 \rangle_\beta$. The sets of variables and functions used to construct the programs are: $\mathbb{V} = \{x\}$ and $\mathbb{F} = \{\text{Pexp Plog sine cos} + - * \% \}$, where the functions are $\lceil (\text{Pexp } x) \rceil \stackrel{\text{return}}{\equiv} \sqrt{|x|}$, $\lceil (\text{Pexp } x) \rceil \stackrel{\text{return}}{\equiv} \exp(\min\{13, x\})$, $\lceil (\text{Plog } x) \rceil \stackrel{\text{return}}{\equiv} \ln(\max\{10^{-17}, x\})$, sine, cosine, addition, subtraction, product and protected division $\lceil (\% xy) \rceil \stackrel{\text{return}}{\equiv} \Theta(|y| - 10^{-4})x/y + \Theta(10^{-4} - |y|)10^4 \text{sgn}(x)$, where Θ is the Heavyside function.

In order to calculate E and C as functions of β , we perform a PTMC in the following way. We first set the number of Metropolis processes K that will run in parallel and the corresponding temperatures $T_1 > T_2 > \dots > T_K$. As discussed elsewhere for parallel tempering [11], the gaps between adjacent temperatures must be chosen in such a way that exchanges between adjacent Metropolis processes are accepted frequently enough ($\mathcal{O}(0.1)$ acceptance rate). The selection of the temperatures is done by trial and error in a preliminary simulation with no data accumulation. A randomly created program is assigned to be the initial state for each Metropolis process. For M_w MC passes (a MC pass is an attempt to modify the state of the system at least once) and without data accumulation, the k th process undergoes an annealing starting from T_1 and ending at T_k , reducing the temperature of the system by $\Delta T_k = (T_1 - T_k)/M_w$ at each MC pass. During the simulation, the parallel tempering is also performed. An exchange between adjacent processes is attempted every ten MC passes. At the end of this initial stage we expect to find each Metropolis process nearly in a steady state. Afterwards we carry on with the data accumulation for M_s MC steps. For the current problem we set $K = 30$, $T_1 = 10^2$, $T_K = 10^{-4}$, $M_w = 10^4$ and $M_s = 510^5$.

The level d_k at which the Metropolis step is proposed varies with temperature T_k . We have chosen to set d_k by an adaptive strategy. For each system we set initially $d_k = 5$. If, for a set of ten MC steps, more (less) than five of the programs proposed are accepted, d_k is decreased (increased) by one. The maximum number of levels a program may have has been set to 15.

In figure 2(a) the points of the set \mathbb{P} are shown (represented by \bullet), together with some curves obtained in the simulation. The behaviour of the curves is quite different at different temperatures. At high temperatures the curves found are very smooth and fit the clusters of points, not the points. When the temperature is lowered, the curves start to fit the internal structure of the clusters by the emergence of high frequency oscillations inside each cluster.

The mean values $E(\beta)$ and $C(\beta)$ reflect this behaviour (figure 2(b)). At high temperatures (low β) all the programs are equivalent, so the mean energy is the highest possible. There is a region in temperature around $\beta = 4000$ at which the behaviour of the curves starts to change from the cluster-fitting regime to the point-fitting regime, as it is shown by the hump in the $C(\beta)$ curve. Finally, at a very low temperature, the mean value of the energy approaches the lowest possible value for the energy.

In the second problem we investigate, the states of the system are update expressions for root finding algorithms. An iterative algorithm for finding numerically the root of a function can be expressed as: $x_{n+1} = F(f, f', \dots, x_n, a, b)$ where x_{n+1} is the new estimate for the root

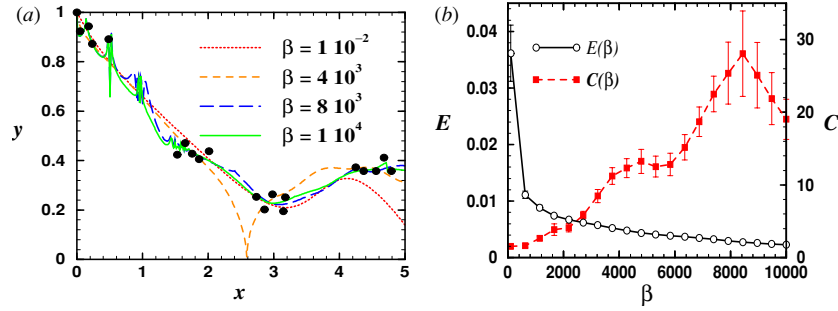


Figure 2. (a) Data points and best curves found at $\beta = 0.01$, $\beta = 4000$, $\beta = 8000$ and $\beta = 10000$. (b) Mean energy $E(\beta)$ and its fluctuation $C(\beta)$, for the constraint regression problem.

and $F(f, f', \dots, x_n, a, b)$ is an update expression that could depend on the function studied f and its derivatives, the previous estimate for the root x_n , and initial conditions like an interval where the root is located $x_r \in [a, b]$ or an initial seed (considered as the 0th estimate to x_r).

To define the energy of the states we need a set of test functions \mathbb{G} that have a root in the (positive) interval $[a, b]$. Every $f \in \mathbb{G}$ can be represented by a program γ_f , thus $[\gamma_f](x) \stackrel{\text{return}}{\equiv} f(x)$. Given \mathbb{G} and $[a, b]$, we have at least two alternatives, either we define the energy proportional to minus the exponent of the convergence rate, or we simply make the energy proportional to the number of iterations needed to meet the convergence criterion. Although the first alternative defines the energy as a function of a general property of the update expression, it makes the energy very sensitive to the given set \mathbb{G} . This happens because the calculation of the rate of convergence has to be carried out numerically. If there is a function in \mathbb{G} for which a particular update expression, given $[a, b]$, does not provide a convergent sequence, the exponent obtained numerically is zero. Thus, we will define the energy according to the second alternative.

A state π of the system represents an update expression for an iterative algorithm. Therefore, if $f \in \mathbb{G}$, $[\pi](f, x_n, a, b)$ provides a real number which is the new estimate for the root, x_{n+1} . The sets of variables and functions used to construct the programs are $\mathbb{V} = \{x\}$, and $\mathbb{F} = \{\text{G Gr Gl Dv Psqr} + - * \% \}$. If the function $f \in \mathbb{G}$ is the one currently considered, G is a dummy operator that returns the program γ_f , i.e. $[(\text{G } x)] \stackrel{\text{return}}{\equiv} \gamma_f$. The actions of Gr are to replace the variable x by b and to return a number, e.g. $[(\text{Gr}(\text{G } x))] \stackrel{\text{return}}{\equiv} [\gamma_f](b) = f(b)$. Gl is the same as Gr but replacing x by a . Dv is the derivative operator, it returns a program, e.g. $[(\text{Dv}(\text{G } x))] \stackrel{\text{return}}{\equiv} (*0.000001(-(\text{G}(+0.000001 x))(\text{G } x)))$.

To calculate the energy of the state π we proceed with the following iterative algorithm for all functions $f \in \mathbb{G}$:

1. Set the counter n to zero and $x_0 = a$ (alternatively $x_0 = b$)
2. Calculate the estimate $x_{n+1} = [\pi](f, x_n, a, b)$ and set n to $n + 1$
3. If the termination criterion has been met, then return n , else, update the lower and upper bounds (a and b respectively) and go to 2.

The energy h of the program π is $h[\pi] \propto \sum_{f \in \mathbb{G}} n_f$, where n_f is the number returned by the iterative algorithm. The termination criterion is met when either a maximum number of iterations n_{\max} is reached (if $n = n_{\max}$ then return n) or the value of the function in the actual estimate is smaller than a given tolerance ε (if $|f(x_n)| < \varepsilon$ then return n). We considered $n_{\max} = 100$ and $\varepsilon = 10^{-4}$.

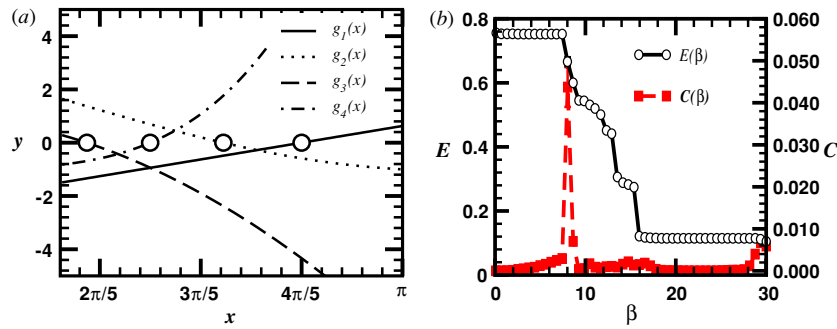


Figure 3. (a) The functions that conform the set \mathbb{G} . The interval considered is $[1, \pi]$ and the functions are: (a) $g_1(x) = x - 4\pi/5$, (b) $g_2(x) = \cos(x) - \ln(x/\pi)$, (c) $g_3(x) = \tan(x/3.24) - x^2 + 1$ and (d) $g_4(x) = (2x/\pi)^4 - 1$. The o indicate the position of the roots. (b) $E(\beta)$ and $C(\beta)$ for the algorithm for the root finding problem.

The four functions of the set \mathbb{G} are presented in figure 3(a).

In the PTMC simulation for this system the selection of the temperatures, the annealing technique used during the warm-up period with no data accumulation, and the adaptive strategy to set the level at which the Metropolis step is proposed are as described in the previous problem. The number of temperatures considered was $K = 50$, the highest temperature was set to $T_1 = 100$, and the lowest to $T_K = 0.02$. The number of warm-up MC steps was set to $M_w = 10^4$ and the total number of MC steps with data accumulation to $M_s = 10^5$. The maximum level that a program can reach was set to 15.

The results for $E(\beta)$ and $C(\beta)$ are presented in figure 3(b). At high temperatures, as expected, all the programs are equivalent and the mean value of the energy goes towards its maximum value. For not so high temperatures, after the peak in the $C(\beta)$ curve a relative of the bisection method starts to appear frequently in the simulation. Instead of approaching the root by the arithmetic mean between the upper (x_u) and lower (x_l) bounds to the root (traditional bisection), this method uses the geometric mean value, i.e. $x_n = \sqrt{x_u x_l}$. For even lower temperatures, the methods found perform better. For very low temperatures we found a set of methods with very good performances. The best method found³ can find the roots of the functions in \mathbb{G} in 7, 3, 5 and 4 iterations respectively. The secant method applied to the functions in \mathbb{G} finds the roots in typically 1, 5, 10 and 32 iterations respectively.

We have calculated the mean energy and specific heat of two systems with states represented by programs, by PTMC simulations. In both cases the landscape of the energy is rough enough to make the Metropolis algorithm not practical. In both cases we found a phase-transition-like behaviour with anomalies in the specific heat curve. The temperature regions separated by these anomalies can be characterized by the properties of the states.

Even though a particular ensemble has been chosen to illustrate this technique, this does not represent a limitation of the method. Simulations in other ensembles are possible.

In both cases the maximum level for a program has been set to 15. Although this appears to be a limitation, the possible number of different programs that can be created is larger than 2^{15} . If we also consider that the higher the level at which a modification is done, the smaller the difference between the old and the new programs (as discussed before), then fixing a cut-off for the maximum level is not an important limitation.

³ (Gr (% (+ (Gr x) (Gl x)) (Dv (Dv (+ (* x x) (* (- (Gr x) (Gl x)) (Psqr (Psqr (+ (* (Gl (+ x x)) (Gl (+ x x)))))))))))).

In general, an adaptively reacting Metropolis step size may lead to (somewhat subtle) violations of detailed balance, which lead to systematic sampling errors and hence incorrect average values. In the cases investigated, the adaptive strategy for the parameter d_k leads to constant values after the warm-up period. Therefore, we can argue that no systematic sampling errors have been introduced by this means.

Despite the fact that the performance of the MC technique, at this point, is not better than that of the GP for optimizing programs, thermodynamic information can be naturally obtained by MC simulations. We have by no means exploited the full set of possible research avenues and scores of technical issues remain. Whether this technique will in the future be competitive or complementary to other available techniques for automatic program design remains unanswered and deserves to be more deeply studied. Nevertheless, even if proven not competitive for program design, the thermodynamic analysis of the system of programs might provide valuable insights into the computational complexity, not of a single algorithm but of the variations on its theme.

Acknowledgment

JPN would like to acknowledge support from EVERGROW, IP No. 1935 in FET, EU FP6.

References

- [1] Koza J R, Keane M A, Streeter M J, Mydlowec W, Yu J and Lanza G 2003 *Genetic Programming IV: Routine Human-Competitive Machine Intelligence* (Dordrecht: Kluwer)
- [2] Holland J H 1975 *Adaptations in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence* (Ann Arbor, MI: University of Michigan Press)
- [3] Oussaid Ene M, Chopard B, Pictet O V and Tomassini M 1997 *Parallel Comput.* **23** 1183
- [4] Koza J R, Mydlowec W, Lanza G, Yu J and Keane M A 2000 Reverse engineering and automatic synthesis of metabolic pathways from observed data using genetic programming *Technical Report SMI-2000-0851*
- [5] Neirotti J P and Caticha N 2003 *Phys. Rev. E* **67** 041912
- [6] Kalos M A and Withlock P A 1986 *Monte Carlo Methods* (New York: Wiley)
- [7] Metropolis N, Rosenbluth A W, Rosenbluth M N and Teller A H 1953 *J. Chem. Phys.* **21** 1087
- [8] Wood W W and Parker F R 1957 *J. Chem. Phys.* **27** 720
- [9] Marinari E and Parisi G 1992 *Europhys. Lett.* **19** 451
- [10] Neirotti J P, Calvo F, Freeman D L and Doll J D 2000 *J. Chem. Phys.* **112** 10340
- [11] Falcioni M and Deem M W 1999 *J. Chem. Phys.* **110** 1754