THE UNIVERSITY OF CHICAGO

A MASSIVELY SCALABLE PARALLEL SIMULATED ANNEALING ALGORITHM

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSCIAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
ZHIHAO LOU

CHICAGO, ILLINOIS
DECEMBER 2016

TABLE OF CONTENTS

LIST OF FIGURES v

ACKNOWLEDGMENTS viii

ABSTRACT ix

1 INTRODUCTION 1

2 RELATED WORK 5
- 2.1 Serial simulated annealing algorithms 7
  - 2.1.1 Theoretical analysis 7
  - 2.1.2 Serial annealing in practice 10
  - 2.1.3 Lam-Delosme algorithm 11
- 2.2 Existing parallel simulated annealing algorithms 12
  - 2.2.1 Single Markov chain approach 13
  - 2.2.2 Population Based Hybrid Approaches 13
  - 2.2.3 Chu's parallelization 14
- 2.3 Search for a scalable parallel simulated annealing algorithm 16
  - 2.3.1 Cooling schedule 16
  - 2.3.2 Move generation 18
  - 2.3.3 Resampling of states 19
  - 2.3.4 Initial moves 19
  - 2.3.5 Stopping criterion 20
  - 2.3.6 Measurement of speedup 20

3 PILOT ALGORITHM WITH RASTRIGIN FUNCTION 22
- 3.1 Introduction 22
- 3.2 Methods 25
  - 3.2.1 The Rastrigin Function 25
  - 3.2.2 Implementation Details 25
  - 3.2.3 Performance Measurement 29
- 3.3 Resampling Interval 32
  - 3.3.1 Searching for an Optimal Interval 32
  - 3.3.2 Adaptive Resampling Interval 33
- 3.4 Discussion 36

4 APPLICATIONS IN REAL PROBLEMS 39
- 4.1 Problem Description 39
  - 4.1.1 The *Drosophila* pattern formation model 39
  - 4.1.2 The transcription model 40
- 4.2 Performance measurement 42
  - 4.2.1 Serial performance curve 42
  - 4.2.2 Success threshold 45

4.3 Challenges in move generation and control . . . 48
4.4 Performance of adaptive resampling intervals . . . 52
4.5 Conclusion . . . 55

5 DENSITY OF STATES ESTIMATION . . . 57
5.1 Motivation . . . 57
5.2 Methods of estimation . . . 58
5.2.1 Wang-Landau Sampling . . . 59
5.2.2 Density of States estimation in optimization problems . . . 61
5.3 Analyses of Test Problems . . . 62
5.3.1 Rastrigin function . . . 62
5.3.2 Application in Gene Regulation Problems . . . 69

6 DISCUSSION . . . 75
6.1 Difficulties in move control . . . 75
6.2 Scalability and the physical metaphor . . . 80
6.3 Future work . . . 81

REFERENCES . . . 83

LIST OF FIGURES

1.1 Performance measures of processors over the decades. Plotted by Christopher Batten[7]. . . . 2
1.2 Number of cores of the 1st and 500th entries on the top500 list from June 1993 to June 2016. Data from top500.org[65]. . . . 3

2.1 Pseudocode for the simulated annealing algorithm . . . 6
2.2 Estimated variance and autocovariance from the Rastrigin function and the pattern formation model problems. . . . 17

3.1 Serial performance boundary by varying K and λ. Annealing runs are conducted from a common temperature range from $T_0 = 10^5$ to $T_f = 10^{-10}$. The solid and dashed lines connect annealing runs with the same K and λ, respectively. The results show that the closer to the performance boundary, the more sensitive the results are from change of (K, λ) pair. . . . 30
3.2 Serial performance curve in (a) number of iterations, and (b) time. Legends show the values of $-\log_{10}(\kappa)$. Those results with energy > 0.01 are considered failed and excluded from the fitting of the curves. . . . 31
3.3 Effect of different resampling intervals on (a) success rate, (b) speedup in iterations, and (c) speedup in time. Legends are the same for all three panels. . . . 33
3.4 (a) Adoption rate for $R = 3$ on $P = 96$. (b) Resampling intervals in an adaptive algorithm on $P = 96$. (c) Adoption rate for adaptive algorithm. Data in (a) and (c) are smoothed by exponentially weighted moving average. Y-axis of (b) is linear between 0 and 10 and logarithmic from 10 above. . . . 34
3.5 Comparison of adaptive resampling and the best resampling interval in (a) speedup in iteration, (b) speedup in time, and (c) success rate. The best intervals are 18 for 24 processors, 8 for 48 processors, 4 for 96 processors and 1 for 192 processors. 35
3.6 Comparison of time spent in computation and communication for adaptive resampling and the best resampling interval. The best intervals are 18 for 24 processors, 8 for 48 processors, 4 for 96 processors and 1 for 192 processors. . . . 36

4.1 Detailed explanation of the transcription model. Adapted from Figures 2 and 3 of A.-R. Kim *et al.* (2013) [35]. . . . 41
4.2 Serial performance curve for the pattern formation model problem . . . 43
4.3 Serial performance curve for the transcription model problem . . . 44
4.4 Distribution of the relative errors of the pattern formation model problem over 100 runs each for serial, 32 cores, and 64 cores. The serial runs have $\lambda_{\text{serial}} = 3 \times 10^{-5}$, while the parallel runs have $\lambda_{32} = 9.6 \times 10^{-5}$ and $\lambda_{64} = 1.92 \times 10^{-4}$. The vertical red line is the proposed threshold at 1.5. . . . 46
4.5 Distribution of the relative errors of the transcription model problem over 100 runs each for serial, 32 cores, and 64 cores. The serial runs have $\lambda_{\text{serial}} = 3 \times 10^{-6}$, while the parallel runs have $\lambda_{32} = 9.6 \times 10^{-5}$ and $\lambda_{64} = 1.92 \times 10^{-4}$. The vertical red line is the proposed threshold at 0.5. . . . 47
4.6 Demonstration of the effects of the move control interval I_{mc} has on the average move size over the annealing process . . . 49

4.7 The effects of different move control parameter K on the pattern formation model 50
4.8 The effects of different move control parameter K on the transcription model . . 51
4.9 Comparison of fixed and adaptive resampling interval for the pattern formation model 53
4.10 Comparison of fixed and adaptive resampling interval for the transcription model 54
4.11 Scalability of the parallel simulated annealing in the pattern formation model. . 55
4.12 Scalability of the parallel simulated annealing in the pattern formation model . 56

5.1 (a) The 1-dimensional Rastrigin function with energy on the horizontal axis and x on the vertical axis. (b) Estimated entropy for 1-dimensional Rastrigin function. The entropy estimate is conducted on 10 cores for a total of 10 million iterations. Note the peaks in (b) correspond to the local minima and maxima in (a). 64
5.2 Estimated heat capacity for the 1-dimensional Rastrigin function compared to its theoretical value. Estimates are calculated from density of state estimates of 100 thousand and 1 million iterations in serial and 1 million and 10 million iterations in parallel using 10 cores. During the numerical integration, dashed lines are calculated using the rectangular method while the solid lines of the same color are using the trapezoidal method on the same data. Note the curve for 1 million serial (green) and parallel (red) coincide almost completely, and the curve for 10 million (magenta) overlaps a large portion of the theoretical curve (black). . . . 65
5.3 (a) Estimated log-density for 5000-dimension Rastrigin function. The estimate was based on 12 cores over 12 billion iterations in total. The raw data are compared to the density modeled as a normal distribution from both direct calculation of mean and variance and also by fitting a least square fit. (b) Calculated heat capacity of the 5000-dimension Rastrigin function over a range of temperature points. The figure shows the value from direct calculation using (5.5) and also by modeling the density as normal or Gamma distributed with same mean and variance. 68
5.4 (a) Estimated entropy for the pattern formation model problem over 700 million iterations on 10 cores. (b) Zoom-in plot of the entropy at the low energy end. (c) Calculated mean energy over a range of temperature points. (d) Calculated heat capacity over the same temperature range. 70
5.5 (a) Estimated entropy for the transcriptional model problem over 400 million iterations on 4 cores. (b) Zoom in plot of the red box in (a). (c) Calculated mean energy over a range of temperature points. (d) Calculated heat capacity over the same temperature range. 71
5.6 Calculated heat capacity curves of the pattern formation model problem from 10 independent density of states estimations 73

6.1 Demonstration of a simple optimization problem with cost function (6.1). (a) Cost function. (b) Density of states. (c) Heat capacity. 76

6.2 Relation between average move size and acceptance ratio for cost function (6.1) at different inverse temperature values. Solid lines are acceptance ratios calculated assuming the current state is at the local minimum where $x = 3$. Dotted lines of the same color represent the acceptance ratio of the same temperature starting from the global minimum $x = 0$. 77

ACKNOWLEDGMENTS

I would like to express my special appreciation to my advisor Professor John Reinitz. I would like to thank you for all the guidance and encouragement you gave me throughout the entire project. I also appreciate the numerous pieces of useless information you told me over the years. These will be remembered long after all the useful ones fade away. I would like to express my special thanks to my committee member Professor Yuefan Deng. Thank you for introducing me to the world of parallel computing. Thank you for serving on my committee and making the trips to Chicago. I would like to thank my co-advisor Professor Rick Stevens for his help over the years, especially in providing me access to the computing resources without which this project would not be possible. I would also like to thank Professor John Lafferty for taking the time to serve on my committee.

I would also like to extend my gratitude to Professor Heinrich Jaeger of the James Franck Institute, Professor Juan de Pablo of the Institute for Molecular Engineering, and Professor Jonathan Whitmer of the Department of Chemical and Biomolecular Engineering at the University of Notre Dame. Thank you all for many useful discussions that eventually become Chapter 5 of this dissertation.

I would like to have a special thanks to my family. Words cannot express how grateful I am to my parents and parents-in-law for their help and support over these years. At the end I would like to thank my beloved wife Fang Han for her support throughout and thank our lovely daughters Hannah and Sarah for being my great motivation.

ABSTRACT

Simulated annealing algorithm is a powerful general purpose stochastic optimization algorithm that relies only on the value of the cost function being optimized. However, existing parallelizations of the simulated annealing algorithm all provide limited speedup. This dissertation presents a parallel simulated annealing algorithm using an adaptive resampling interval. The algorithm gives a speedup of 170 using 192 processor cores when applied to a 5000-dimension Rastrigin function. It also achieves a speedup of more than 40 using 128 cores on two systems biology problems. In addition, this dissertation proposes an algorithm to study the structure of the search space based on the density of states. The results provide information on setting the parameters of the annealing algorithm for the problem being optimized.

CHAPTER 1

INTRODUCTION

In the past decades, the concept of parallel computing has turned from a novelty to a necessity. While Moore's law still applies after four decades, both clock speed and single thread performance have become stalled in recent years (Figure 1.1). The growth of the number of transistors packed in a chip turns into more computing cores instead. On the other end of the spectrum, top supercomputers are also packing more processors, connected with high speed, low latency networks (Figure 1.2). To make use of such increases in the available number of cores on applications with ever larger data sets, parallel algorithms, especially those which scale to thousands or even millions of cores, are in high demand.

Domain decomposition, or as it is sometimes called, data decomposition, is the most dominant technique used in developing parallel algorithms. It works very well where it is applicable. Top search engines routinely handle hundreds of millions of requests per day. In the realm of scientific computing, the top supercomputer is able to deliver 33.8 PFLOPS using 3.12 million cores with over 60% parallel efficiency[65] using Linpack, a parallel linear algebra package. A recent paper also demonstrates a 100 million atom molecular dynamics simulation on 298,992 cores with 64% parallel efficiency using NAMD[62].

However, not all of the problems have a natural manner of decomposition. Optimization problems, especially non-linear, high dimension problems requiring stochastic optimization algorithms generally take millions of iterations to reach a satisfactory result. With each individual step much smaller than the whole algorithm, decomposing each iteration is not enough. Decomposing the long iterative process, on the other hand, changes the behavior of the algorithm itself, and when done naively may reduce the quality of the results.

Without lose of generality, an optimization problem can be described as follows. Given an objective function $f : \mathcal{S} \to \mathbb{R}$, where $\mathcal{S}$ is the feasible region or the search space, we want

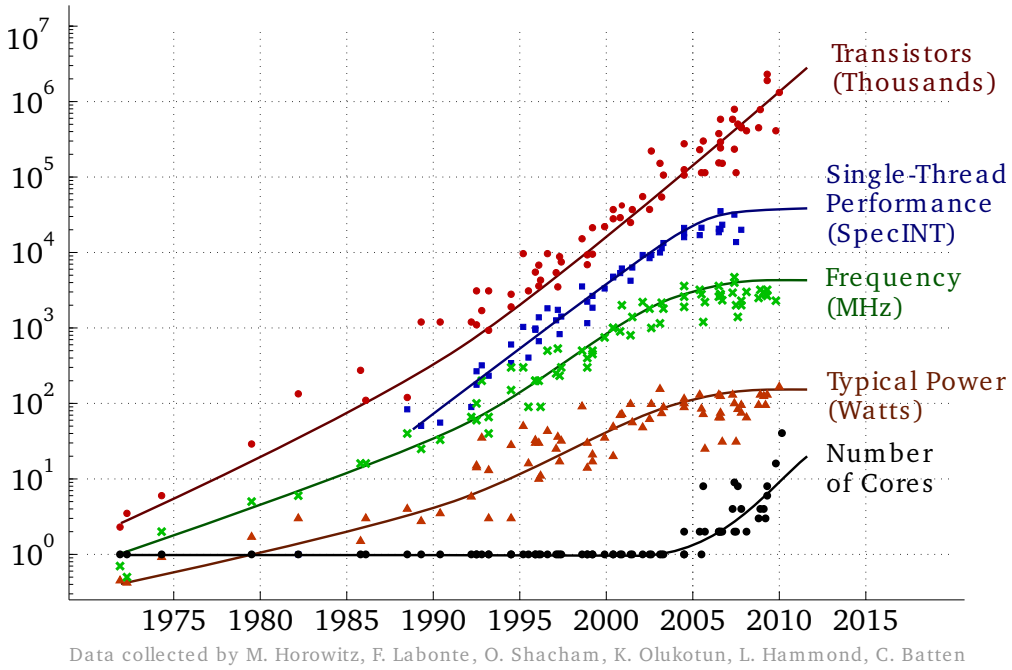


Figure 1.1: Performance measures of processors over the decades. Plotted by Christopher Batten[7].

to find $x^* \in \mathcal{S}$, such that

$$f(x^*) = \min_{x \in \mathcal{S}} f(x). \tag{1.1}$$

To emphasis the search of x^* over the value of $f(x^*)$ itself, the problem can be written as

$$x^* = \arg\min_{x \in \mathcal{S}} f(x). \tag{1.2}$$

Generally speaking, in the realm of machine learning and other models for complex systems, the cost function has form

$$f(x) = \sum_{i=1}^{N} w_i d(M_i, D_i) + p(x), \tag{1.3}$$

a weighted sum of some distance measurement $d(\cdot, \cdot)$ between the prediction M_i and data D_i, plus a penalty or regularization term $p(x)$. For applications like neural networks where both the prediction and cost function have closed form for themselves and their first derivative,

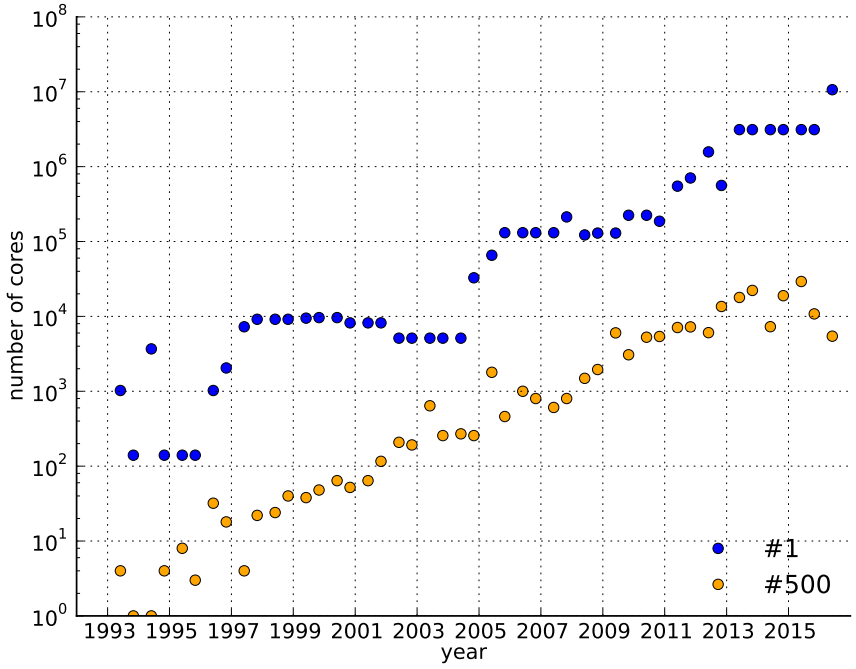


Figure 1.2: Number of cores of the 1st and 500th entries on the top500 list from June 1993 to June 2016. Data from top500.org[65].

gradient based methods such as stochastic gradient descent and their parallelization are highly effective. For problems and models that do not afford such luxury, a general purpose stochastic optimization method that relies only on the values of the cost functions become necessary. The simulated annealing algorithm is one such algorithm that is particularly powerful in solving difficult optimization problems.

This thesis documents the attempts and results in developing a scalable parallel simulated annealing algorithm. The structure of this thesis is as follows. Chapter 2 will offer a brief review of the literature in both serial and existing parallel simulated annealing algorithm. The end of the chapter will discuss a basis for a scalable parallel simulated annealing algorithm. Chapter 3 starts a pilot program on the new algorithm using the Rastrigin function as a test problem. In chapter 4, two more real world problems will be used as test problems. In chapter 5, I will discuss the method of studying the structure of the search space.

Finally, chapter 6 provides some discussions regarding the source and possible remedies of the speedup limitation.

CHAPTER 2

RELATED WORK

The physical annealing process starts from a system in equilibrium at some high temperature T. According to statistical mechanics, the probability P_E that such a system in a state with energy E is determined by the Boltzmann distribution of the ensemble at temperature T

$$P_E = g(E)\frac{\exp\left(-\frac{E}{k_B T}\right)}{Z(T)}. \tag{2.1}$$

where $g(E)$ is the number of states having energy E and $Z(T)$ is the partition function. Then, by cooling the temperature infinitesimally slowly, the system undergoes a quasi-equilibrium process and keeps the Boltzmann distribution at each corresponding temperature. When T reaches 0, the Boltzmann distribution collapses to the global minimum or minima.

The simulated annealing algorithm introduced by S. Kirkpatrick is a stochastic optimization algorithm that mimics the physical process of annealing[36]. By introducing an artificial temperature T into the system, this algorithm makes an analogy between the energy of a system and the value of the cost function being optimized. At each temperature, the algorithm runs the Metropolis algorithm repeatedly to perturb the state so that the distribution of the energies of the states will gradually approximate the Boltzmann distribution at that temperature T[47]. Thus, after gradually decreasing the temperature, the cost being optimized, or the energy, is expected to converge to its global minima, just as the statistical physics dictates it will in the real annealing process.

The algorithm is shown in pseudocode in Figure 2.1. The perturbation in line 5 generates a random proposed trial solution x_p based on the current solution x_n. This process is called move generation. For a discrete state space, it is common to pick x_p from in the immediate neighborhood x_n. For continuous search space, a predefined distribution, usually Gaussian, centered at x_n is used[21]. In some situation, distributions with unbounded variance like the Cauchy distribution may also be used[63]. The proposed state x_p is accepted according to

```
1  begin
2      initialize T_0, x_0;
3      E_0 ⟵ f(x_0);
4      while not frozen do
5          x_p ⟵ perturb(x_n);
6          E_p ⟵ f(x_p);
7          ΔE ⟵ E_p − E_n;
8          ξ ⟵ uniform(0, 1);
9          if ξ < accept(ΔE, T_n) then
10             x_{n+1} ⟵ x_p;
11             E_{n+1} ⟵ E_p;
12         else
13             x_{n+1} ⟵ x_n;
14             E_{n+1} ⟵ E_n;
15         end if
16         T_{n+1} ⟵ cooling schedule(T_n);
17         n ⟵ n + 1;
18     end while
19 end
```

Figure 2.1: Pseudocode for the simulated annealing algorithm

the Metropolis criterion determined by the energy difference $\Delta E = f(x_p) - f(x_n)$ and the current temperature T[47]. That is, to accept x_p with probability

$$\text{accept}(\Delta E, T) = \begin{cases} \exp\left(-\frac{\Delta E}{T}\right), & \Delta E > 0 \\ 1, & \text{otherwise.} \end{cases} \tag{2.2}$$

Lines 5 through 15 are generally known as the Metropolis algorithm. A cooling schedule controls the pace of the temperature decrease. The system is considered frozen either when it reaches a predefined temperature, usually 0 or a value very close to 0, or when no more moves can be accepted at the temperature.

As simulated annealing is such a powerful optimization algorithm, it is unsurprising to see a large number of studies devoted to its parallelization.

This chapter offers a brief review of the literature in both serial and parallel simulated annealing, and then focuses on two related algorithms in particular, Lam's algorithm and

Chu's parallelization, which leads to the search for a scalable parallel simulated annealing algorithm to be presented in this thesis.

2.1 Serial simulated annealing algorithms

2.1.1 Theoretical analysis

Despite the analogy between the simulated annealing algorithm and the actual physical annealing process, most of the theoretical analyses regarding simulated annealing are based on Markov chains rather than statistical mechanics. These analyses lead to precise cooling schedules and elegant convergence proofs no other meta-heuristics can yet achieve. Unfortunately, these schedules and proofs does not provide adequate directions in practice.

Hastings in 1970 proposed a way to analyze and also generalize the Metropolis algorithm using the Markov chain approach[26]. Consider the sequence of states $\{x_n\}$ and the sequence of energies defined on these states $E_n = f(x_n)$. The Metropolis algorithm implies that the state x_{n+1} is only dependent on its previous state x_n and hence $\{x_n\}$ is a Markov chain. Under a fixed temperature, it is natural to assume the move generation is also fixed. Thus, the process of the Metropolis algorithm can be described using a homogeneous Markov chain. Let

$$P_T(a,b) = \mathcal{P}_T[x_{n+1} = b | x_n = a], \qquad a, b \in \mathcal{S} \tag{2.3}$$

denote the one step transition probability from state a to state b for such a Markov chain $\{x_n\}$ at temperature T. When the number of states in $\mathcal{S}$ is finite, $P_T(a,b)$ form a transition matrix $\mathbf{P}_T$. The transition matrix $\mathbf{P}_T$ can have the Boltzmann distribution as its stationary distribution if the detailed balance equation

$$\frac{P(a,b)}{P(b,a)} = \frac{\exp[-E(b)/T]}{\exp[-E(a)/T]}. \tag{2.4}$$

is satisfied. If an appropriate move generation model is chosen such that $\mathbf{P}_T$ is irreducible,

i.e., $\forall a, b \in \mathcal{S}$, we can find a sequence $x_0 = a, x_1, \ldots, x_{k-1}, x_k = b \in \mathcal{S}$ such that

$$\prod_{i=1}^{k} P_T(x_{i-1}, x_i) > 0, \tag{2.5}$$

then such a Markov chain is also positive recurrent and its stationary distribution is unique. Furthermore, such a Markov chain will converge to the stationary distribution, in this case the Boltzmann distribution, regardless of the initial distribution.

In addition, each step in the Metropolis algorithm consists of first generating a proposed state and then deciding if that state is to be accepted. Since the acceptance probability is only dependent on the energy of the current and proposed states, the one step transition probability can be represented as a product of $G(a, b)$, the probability of proposing b given a, and $A(a, b)$, the probability of accepting such a proposed move:

$$P(a, b) = G(a, b)A(a, b). \tag{2.6}$$

In practice, the move generation $G(a, b)$ usually only depends on the distance of the proposed state from the current state. That is $G(a, b) = G(b, a) = G(r)$ where r is some measure of distance between state a and b. This reduces (2.4) to

$$\frac{A(a, b)}{A(b, a)} = \frac{\exp[-E(b)/T]}{\exp[-E(a)/T]}. \tag{2.7}$$

Assume $E(b) \geq E(a)$, then the Metropolis criterion in (2.2) for the probability of accepting state b given state a at temperature T is $A(a, b) = \exp\{-[E(b) - E(a)]/T\}$ while accepting state a given state b has probability $A(b, a) = 1$. It is thus clear the Metropolis criterion is a special case of (2.7).

When $T = 0$, moves toward a higher energy state will no longer be permitted, and hence the irreducibility condition no longer holds. The theoretical analyses on the behavior of this system as T approaches 0 can be categorized into homogeneous and inhomogeneous Markov

chain approaches[49]. The homogeneous approach constructs a Markov chain with m sub-chains. Each sub-chain runs n steps of the Metropolis algorithm at temperature T_m, where $T_m \to 0$ monotonically as $m \to \infty$. It can be proved that the Markov chain converges to a global minimum when $n \to \infty$ and $T_m \to 0 (m \to \infty)$[2]. To implement this algorithm based on the homogeneous approach, a given temperature has to be held long enough for the energy distribution to become stationary. Doing so is generally infeasible in practice, but the inhomogeneous approach provides a way to cool the temperature every step which still guarantees convergence to the minimum state. Geman and Geman pointed out[21], and later Gidas showed with much more rigorous analysis[22], that for a finite number of states, with a symmetrical and irreducible move generation, a cooling schedule that is slow enough to satisfy

$$T(n) \geq \frac{c}{\log(1+n)}, \tag{2.8}$$

where c is a problem specific constant, will make the algorithm converge in distribution. Hajek later proved that (2.8) is sufficient and necessary for the convergence[25].

From the Markov chain point of view, the convergence is a property of the transition matrices, hence it depends not only on the cooling schedule but on move generation as well. Szu and Hartley exploited this property and assert that, on a continuous state space, any bounded variance distribution as the move generation model can cool no faster than (2.8). In contrast, they demonstrated that using the Cauchy distribution, which has infinite variance, for the move generation model allows a cooling schedule that is inversely linear given by

$$T(n) \geq \frac{c}{1+n}, \tag{2.9}$$

which converges to the global minimum[63].

2.1.2 Serial annealing in practice

Although the cooling schedules derived in the previous section provide theoretical assurance, in practice these methods are too slow and the geometric schedule to be discussed below is usually employed. To see how slow they are, consider a specific example used in *Drosophila* embryo pattern formation models[57]. A typical simulated annealing run in that paper starts at temperature $T_1 = 1000$ and ends at $T_n = 0.0202905$ in $n = 17737100$ steps. To cover the same temperature range using (2.8) requires about 10^{21403} steps. Considering that the age of universe is only on the order of 10^{17} seconds, it is impossible for the logarithmic schedule to be deployed in practice. To address this problem, various schedules without proven convergence are proposed and used in applications with promising results.

The single most simple and widely used schedule is the geometric or exponential cooling schedule

$$T_{n+1} = \alpha T_n \tag{2.10}$$

with $0 < \alpha < 1$. Called simulated quenching by Ingber[29], this schedule gives up the proved convergence property in exchange for rapid execution and the results are largely acceptable. It is so commonly used that in the rest of the paper, all the cooling schedules used in the algorithms studied are geometric unless otherwise specified.

For more difficult problems, the exponential cooling schedule will sometimes suffer from premature freeze and produces less then ideal results. This is a result of overly rapid cooling. Schedules that can overcome this problem generally take advantage of past statistics to dynamically determine the cooling pace.

Aarts and van Laarhoven introduced one such adaptive cooling schedule [2] in the form of

$$\beta_{n+1} = \beta_n + \frac{\lambda}{\sigma(\beta)} \tag{2.11}$$

where $\beta = 1/T$ and $\sigma(\beta)$ denotes the standard deviation of the distribution of values of the cost function of the points in the Markov chain at β.

2.1.3 Lam-Delosme algorithm

The Lam-Delosme algorithm, like many practical algorithms, gives up the theoretical convergence property for a faster cooling schedule. It features an adaptive cooling schedule and explicit move generation control. Given its fast speed and high quality results, it is the ideal starting point for a high performance parallel algorithm.

The cooling schedule in the Lam-Delosme algorithm is an adaptive schedule based on the quasi-equilibrium condition

$$\left|\bar{X}(\beta_n) - \mu(\beta_n)\right| \leq \lambda\sigma(\beta_n), \tag{2.12}$$

where $\bar{X}(\beta_n)$ is the average energy at inverse temperature β_n and $\mu(\beta_n)$ is the mean energy at β_n[38, 39]. The resulting optimal cooling schedule is

$$\beta_{n+1} = \beta_n + \left[\frac{\lambda}{\sigma(\beta_n)}\right]\left[\frac{1}{\beta_n^2\sigma^2(\beta_n)}\right]\left[\frac{4\rho_0(1-\rho_0)^2}{(2-\rho_0)^2}\right], \tag{2.13}$$

where β_n is the inverse of temperature at step n, $\sigma(\beta_n)$ is the variance of the energy at inverse temperature β_n and ρ_0 is the acceptance ratio defined by the ratio of accepted moves to the total number of moves. This equation can be interpreted as follows:

- $\dfrac{\lambda}{\sigma(\beta_n)}$ is the measure of the thermal equilibrium.
- $\dfrac{1}{\beta_n^2\sigma^2(\beta_n)}$ is the inverse of the specific heat.
- $\dfrac{4\rho_0(1-\rho_0)^2}{(2-\rho_0)^2}$ is a measure of the effectiveness of sampling of the search space, and is at a maximum value when $\rho_0 \approx 0.44$

Since during the process of annealing, the temperature is lowered at every step, the estimation of the temperature dependent energy variance $\sigma^2(\beta)$ is a difficult task. The algorithm tries to solve this by using a local inverse linear model for both mean and energy variance that is valid for an interval of τ iterations. After τ iterations, the local model is

updated using the results of the last τ iterations and others form further in the past with exponentially decaying weights. In practice, τ is set to 100 for best performance.

The use of the acceptance ratio ρ_0 connects the cooling schedule to the move generation control. If no moves get accepted, i.e. $\rho_0 = 0$, no new states are being sampled. On the other side, if all moves get accepted $\rho_0 = 1$, the algorithm reduces to random sampling. Historically, the original Metropolis paper reported that the system works best when half of the moves are accepted[47]. Lam offered a detailed model and reaches a more precise optimal value of 0.44.

The algorithm requires a controllable move generation strategy to take advantage of such optimality. By "controllable move strategy", we mean that there exists a parameter θ such that the increase of θ will increase the size of the move and thus the magnitude of the proposed energy change. Such an increase will in turn decrease the acceptance ratio ρ_0, and vice versa. Thus, the control of the acceptance ratio ρ_0 can be achieved by controlling the average θ, $\bar{\theta}$, using proportional feedback control

$$\bar{\theta}_n = \bar{\theta}_{n-1} + K(\hat{\rho}_n - 0.44) \tag{2.14}$$

where $K > 0$ is the gain. Finally, the θ of each step is drawn from the double exponential distribution with mean $\bar{\theta}$.

2.2 Existing parallel simulated annealing algorithms

Since the Markov chain analysis indicates an inherently sequential process at the heart of the simulated annealing algorithm, all parallelization schemes must face a fundamental question of how closely should the parallelized algorithm follow the serial framework. Too close to the serial approach will certainly limit the speedup, while straying too far will lose the good attributes and result in a degradation of quality.

This section classifies the parallelization schemes into two large categories according to

their closeness to the serial algorithm: single Markov chain, and population-based hybrid approaches.

2.2.1 Single Markov chain approach

Some early approaches tried to parallelize simulated annealing while keeping the same theoretical properties as the serial algorithm. These algorithms take advantage of the fact that proposing a new state generally takes much less computation time than deciding whether to accept that state. Thus, multiple states can be proposed and evaluated in parallel. Among these, branch prediction algorithms organize such parallel moves in a decision tree[72, 73], while simultaneous moves algorithms simply generate states directly from current state[61].

Since much of the parallelism is obtained by speculative calculations, evaluations which turned out not to be part of the Markov chain are wasted. Hence, despite some implementations run on as many as 100 processors [61, 73], no speedup over 20 was reported.

2.2.2 Population Based Hybrid Approaches

The family of population based algorithms includes genetic algorithm [28], evolution strategies[6], particle swarm optimization [34] and other variations. Earlier algorithms of this type draw an analogy between the optimization problem and the natural process of evolution. Later examples, like particle swarm or ant colony optimizations, turn to the collective behavior of animals for inspiration.

Regardless the source of inspiration, all of these algorithms maintain a population of states being optimized. At each iteration, these algorithms operate on the population using duplication, elimination and other operations according to each algorithm's metaphor. Most of these operations can be naturally distributed for parallel computation to a number of processors up to the size of the population. However, studies show that the optimal population size may be small even for a relatively difficult problem[3, 33], which limits the effective scalability.

In parallel simulated annealing algorithms, the class of multiple Markov chain approaches is structurally similar to the population based algorithms. These approaches maintain a collection of states, which are periodically duplicated or eliminated. Thus, these algorithms themselves can be seen as hybrid with population based algorithms there the collection of states can be considered as a population and the communication stage performs operations on the population. Such operations can be done every fixed iterations, like the division algorithm proposed by Aarts *et al.*[1] and its variation [27]. There are implementations that make interval length variable[41, 53], or have more sophisticated state selection schemes at the end of each interval [12, 16]. In addition, Thompson *et al.*[64] and Xavier-de-Souza *et al.*[74] proposed generalized acceptance functions to suit the multiple Markov chains.

More explicit hybrid algorithms use concepts from population based algorithms directly. The simplest ones run population based operations to generate initial states which feed to multiple Markov chains in parallel simulated annealing[14, 53]. More sophisticated approaches run evolution style selection periodically[14, 77]. In addition, there are many population based algorithms which uses annealing style temperature control[13, 42, 43, 58, 68].

With respect to performance, most algorithms, including two additional comparison studies[40, 50], show various speedup using up to 32 processors[53, 68, 77], while a few of these algorithms show suspicious superlinear speedup[14, 43]. Superlinear speedup generally suggests either inefficiency in the serial performance or a decrease in the quality of parallel results[23]. The few studies which focus on large number of processors[13, 58] did not report speedup data.

2.2.3 Chu's parallelization

Among the parallelization algorithm reviewed in the previous section, that proposed by Chu *et al.* shows the best speedup [16], but is unable to scale beyond 100 processors due to implementation limitations. This thesis will improve upon the work of Chu *et al.* by examining each component of the algorithm. But before diving into the details of the proposed

algorithm, this section first details the algorithm of Chu *et al.*

The parallelization algorithm proposed by Chu *et al.* tries to preserve the serial structure of the Lam-Delosme algorithm as much as possible. It parallelized the τ loop in the variance estimation by cooling P times faster on a P-core parallel run. Thus, after gathering data from all cores, a τ loop still receives the same number of observations for estimation purpose, and the rest of the estimation and cooling schedule are preserved.

In addition to parallelizing the estimation, the algorithm more importantly resamples the states among processor cores periodically using what is called the "mixing of states" method. Periodically, this method pools the states from all the processor cores, then each core independently draws state i with energy E_i from the pool with probability

$$\frac{e^{-E_i\beta}}{\sum_{j=1}^{P} e^{-E_j\beta}}. \tag{2.15}$$

Then each core adopts the state of its target and carriers on annealing with its new state. When the temperature is high, every state has roughly equal probability being adopted. As the system cools, a better state is more likely to be adopted by multiple cores and hence be replicated across the system, while states with higher (worse) energies are more likely to be dropped. The interval between resamplings is set to $M\tau$, where M is a problem dependent integer predetermined such that, according to Chu *et al.*, it is long enough for the states to be decorrelated, while short enough that the variance of the average energy of the states in each processor remain low.

This algorithm, while apparently the current leader in parallel simulated annealing, suffers from two disadvantages that reduces its scalability and usefulness. Firstly, its method of speedup means it can only run on a number of cores that is an integer divisor of τ. Since τ has been limited to 100 in practice, this greatly limits the scalability of the algorithm. Secondly, the resampling interval has a significant affect on its performance, yet it provides no way to predict an optimal value without actually running through the annealing process.

2.3 Search for a scalable parallel simulated annealing algorithm

To build on top of the best existing algorithm and overcome their disadvantages, this section examine each component of a parallel simulated annealing algorithm and discuss the basis for a scalable parallel simulated annealing algorithm.

2.3.1 Cooling schedule

The adaptive cooling schedule in the Lam-Delosme algorithm, like many other adaptive schedules, relies on the estimation of variance to adjust the cooling speed. Its sophisticated model for estimating the mean and variance of energy as a function of temperature imposes a significant computational burden. Nonetheless, the accuracy of these estimations, especially those in parallel ones, are still lacking. As a matter of fact, estimating variance of energy even in fixed temperature Metropolis algorithm is fundamentally difficult.

The energy values obtained from a fixed temperature Metropolis algorithm with an appropriate move generation scheme can be considered as a realization of a Markov chain, whose stationary distribution is the Boltzmann distribution of the system being sampled at the corresponding temperature. From time series point of view, the sequence of energy values are highly correlated, and the autocorrelation remains non-zero even for fairly large lag values. This means the variance estimation, either by calculating sample variance directly or by using both sample variance and autocovariance, will converge too slowly for any practical purposes (Figure 2.2).

In a parallel situation, the problem with variance estimation gets worse. In addition to the correlation problem inherited from the serial algorithm, the resampling introduces further source of correlation between the energy sampled between processor cores. The scalability limitation of Chu *et al.* can be worked around by the combination of using a better local model and allowing each estimation data point to be calculated with a constant samples per core. However, such improvement cannot completely overcome the difficulty in the variance

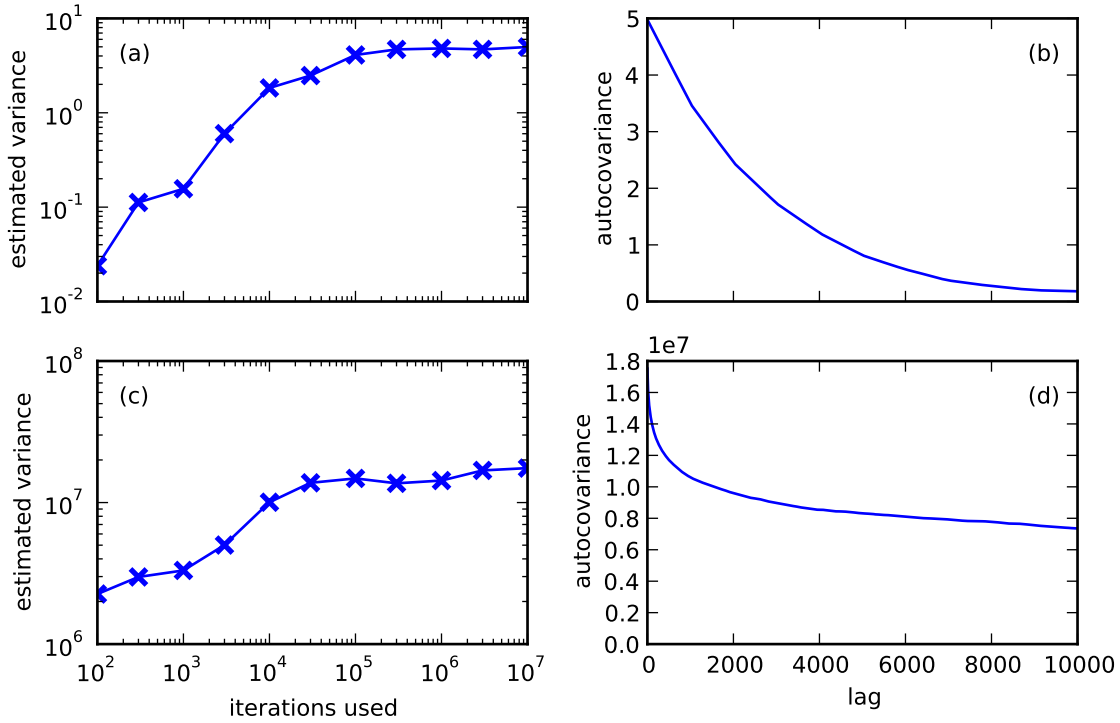


Figure 2.2: Estimated variance and autocovariance from the Rastrigin function and the pattern formation model problems. (a) Variance of 1000-dimension Rastrigin function using different length of Metropolis algorithm at $T = 0.1$. (b) Autocovariance of (a) using the longest sample. (c) Variance of the pattern formation model problem using different length of Metropolis algorithm at $T = 1000$. (d) Autocovariance of (c) using the longest sample.

estimation from samples discussed here.

Accordingly, this thesis use the geometric cooling schedule described in the original Kirkpatrick *et al.* paper [36], which specifies

$$T_{k+1} = (1 - \lambda)T_k, \tag{2.16}$$

where k is the iteration number and λ is a positive number very close to 0 that controls the cooling speed. This cooling schedule frees the algorithm from the limitation of τ.

The mechanism of speedup is similar to that in the Chu algorithm. The algorithm should sample the same number of states over the same temperature span in parallel as in serial. For a parallel setup with P cores, this is achieved by setting

$$\lambda_P = P\lambda_s, \tag{2.17}$$

where λ_s is the base cooling speed in serial.

2.3.2 Move generation

The control of move generation is arguably the most important feature in the Lam-Delosme algorithm [38, 39] and also in the parallelization by Chu *et al.* [16]. It is also vital in practice. Many applications contain cost functions in which parameters have different sensitivities, which naturally requires different average move sizes. For these problems with non-homogeneous parameter space, each parameter is controlled separately.

The details of the move generation and control mechanism are as follows. At each step, a proposed move $x_i^{\text{proposed}} = x_i + \Delta x$ is generated by drawing moves from a Laplace distribution with mean 0 and variance $2\theta^2$, that is

$$\Delta x \sim \text{Lapalce}(0, \theta). \tag{2.18}$$

The average move size θ is adaptively adjusted every I_{mc} steps by feedback move control

$$\ln \theta_{\text{new}} = K(\rho - 0.44) + \ln \theta_{\text{old}} \tag{2.19}$$

where K is the proportional constant and ρ is the acceptance ratio. In parallel, the acceptance data are pooled from all cores before each move control so that the average move size θ is the same across all cores.

2.3.3 Resampling of states

Periodically communicating the states across processor cores is necessary for compensating for the more rapid cooling schedule in parallel. This thesis adopts the "mixing of states" algorithm of resampling proposed by Chu *et al.* detailed in section 2.2.3.

Since the fundamental granularity of τ has been abolished in the cooling schedule, there is no particular reason that the resampling interval should only be an integer multiple of τ. This allowed an aggressively short resampling interval. More details about the resampling interval will be discussed during the next two chapters.

2.3.4 Initial moves

Before the annealing algorithm starts and cools the temperature, it is important to run the Metropolis algorithm at the fixed initial high temperature for a number of iterations. This process is commonly known as the initial moves. When generating a state in the search space at random, the common implementation usually samples from the uniform distribution in the search space. The initial moves make sure that the states are sampled instead from the Boltzmann distribution at the initial temperature. It also helps to set the move control parameters to appropriate values.

For the purpose of simplicity and consistency, this thesis prepares a pool of 1000 initial states for each test problem. Each state in the pool is generated by making the initial moves

long enough such that both the average energy and the average move sizes converge. All the numerical experiments start from a state, or states in parallel cases, randomly chosen from the pool.

2.3.5 Stopping criterion

Finally, an annealing algorithm stops when the system is considered "frozen". The determination of frozen condition is largely problem dependent. For a problem with a discrete state space, the time when no further moves can be accepted is a natural point to consider the system being frozen. For a continuous cost function, there are always moves small enough to be acceptable. Thus, a stopping criterion κ is proposed such that the system is considered frozen if

$$|E_i - E_{i-r}| < \kappa, \tag{2.20}$$

where r is some pre-determined number of iterations that is usually proportional to the number of dimension of the problem. The parallel algorithm periodically checks the stopping criterion on all cores and stops when the first one reaches the stopping criterion.

2.3.6 Measurement of speedup

Once all the basic components have been agreed upon, the algorithm is ready for testing under different test problems. The widely accepted metrics for the performance of a parallel algorithm is the speedup, defined as

$$S(P) = \frac{N_s}{N_P}, \tag{2.21}$$

where N_s and N_P are respectively the number of iterations of the serial algorithm and of the parallel algorithm on P cores. Speedup in time can be similarly calculated. For an optimization problem, we must ensure that parallel algorithm is compared to the most efficient serial algorithm, and the comparison must take into account changes in the quality

of the answer.

For a maximally efficient serial annealing algorithm, the average quality of the answer depends on a single variable, the average number of iterations [16, 39]. For each problem, the tradeoff between the average number of iterations $N_s(E)$ as a function of the average final results E can be empirically measured by adjusting parameters of the algorithm. This relation is sometimes called the serial performance curve of the problem. Thus, the adjusted speedup of a P-core parallel run taking N_P iterations per core to produce a final energy E_P is

$$S(P) = \frac{N_s(E_P)}{N_P}. \tag{2.22}$$

The following chapters will consider the performance of the parallel simulated annealing on different test problems. Chapter 3 will present a pilot program to test the parallel algorithm on the Rastrigin function. Chapter 4 will further test the algorithm on two more real world optimization problems.

CHAPTER 3

PILOT ALGORITHM WITH RASTRIGIN FUNCTION

The following chapter first appeared as Parallel Simulated Annealing Using an Adaptive Resampling Interval *by Zhihao Lou and John Reinitz in* Parallel Computing *Vol 53 pp. 23–32.*

3.1 Introduction

Simulated annealing, introduced by Kirkpatrick *et al.* [36], is one of the most widely used general purpose global optimization algorithms. Certain applications in developmental biology and systems biology rely heavily on this particular method of optimization [18, 31, 35, 45, 57]. It mimics the physical process of annealing by treating the cost function as an "energy" E and sampling the value of E according to the Boltzmann distribution at some artificial temperature T using the Metropolis algorithm [47]. At each iteration, the algorithm proposes a perturbed state (a "move") with energy E_{new} from the current state with energy E_{old}. If $E_{\text{new}} < E_{\text{old}}$, the new state is accepted. Otherwise the new state is accepted with probability $\exp[-(E_{\text{new}} - E_{\text{old}})/T]$. During the annealing process, T is decreased slowly and uphill moves become less and less likely. If the move generating scheme and schedule for decreasing T are chosen correctly, then as T approaches zero, E will converge to its global minimum. Convergence to the global minimum has been proved for a schedule in which the temperature at the kth iteration $T_k \propto 1/\ln(k)$ and moves are drawn from a Gaussian distribution [21, 25], and also for a schedule where $T_k \propto 1/k$ and moves are drawn from a Cauchy distribution [63]. The structure of the proofs reveal that the correct choice of both distribution of the moves and the cooling schedule are important for good performance. In practice, a much faster cooling schedule without a convergence proof was used in both the original Kirkpatrick paper, and most applications. In this schedule, $T_k \propto e^{-\lambda k}$ where λ is sometimes adjusted adaptively based on sampling statistics [2]. Adaptive control of move generation together

with cooling rate was introduced by Lam and Delosme [38, 39]. This innovation turned out to be essential for applications in the natural sciences because the parameters of such problems typically have widely differing characteristic scales [55, 56, 57, 59].

As computing architectures shift toward more cores and massively parallel computers become more accessible, there have been many attempts to parallelize the simulated annealing algorithm [5]. Because the Metropolis algorithm is a Markov process, early parallelization attempts focused on preserving the single Markov chain in the parallel algorithm by using variants of branch prediction [61, 72, 73]. By the nature of these algorithms, they do not scale well as the number of processors increases. The majority of parallelization strategies allow the algorithm to follow multiple Markov chains. These include the division algorithm [1] and variants [27, 40, 41, 53], as well as resampling of states [12, 16]. There are also algorithms which change the acceptance criterion in the Metropolis algorithm to better suit the parallelization [64, 74]. Since Rudolph [58] pointed out the equivalence of simulated annealing and the evolutionary algorithm with a population size of one, explicit hybridization with population based algorithms has been proposed, either by running these algorithms side by side [14, 53, 77], or by blending in the concepts of evolutionary algorithms with the move generation strategy [15, 43, 58, 69]. Unfortunately, most published works did not report speedup at all [15, 58, 64, 69, 74, 77], or only reported speedup for up to 16 processors, a small number by today's standards [12, 40, 41, 53, 72]. Among those that reported speedup for 32 processors or more, some show mediocre speedup [61, 73] while others give a suspiciously super-linear speedup [43, 77]. Super-linear speedup suggests either inefficiency in the serial performance or a decrease in the quality of parallel results [23].

The most promising recent development in parallel simulated annealing is reported in recent work on GPUs [20]. In GPUs, it is possible to run an simulated annealing problem on a high number of threads that may exceed the number of cores. The calculation of speedup is made retrospectively by rerunning the problem on a single thread under identical cooling conditions. This leaves the efficiency of the serial annealing process uncontrolled, making

comparison with extant methods difficult. We take a different approach in this work, which may profitably combined with architecture-specific methods such as this one in the future.

Among these methods, that proposed by Chu *et al.* [16] demonstrated practical utility in solving certain estimation problems that arise in systems biology and operations research on the nuclear fuel cycle [4, 8, 9, 17, 18, 24, 30, 31, 35, 37, 45, 51, 52]. This method is based on the serial Lam-Delosme algorithm [38, 39], which features an adaptive cooling schedule in which the rate of cooling is a function of the variance and the proportion of accepted moves, together with feedback move generation control. The inner loop of the serial algorithm executes the Metropolis algorithm over τ steps, after which the variance is re-estimated. The mechanism of speedup is to let each processor cool P times faster so that it samples the same number of states in parallel as the serial algorithm would in τ steps over the same temperature interval. To retain the estimation structure of the Lam-Delosme algorithm by parallizing only over the τ loop, the Chu algorithm faces the limitation that it can only work on P processors, where P is an integer divisor of τ, and has a hard limit on scalability at τ processors. In practice the algorithm works the best at $\tau = 100$ and delivers good speedup for up to 50 processors. This method, while apparently the leader among parallel simulated annealing algorithms, thus faces an inherent scalability barrier.

In this work, we examine each component of the algorithm to explore the possibility of scaling beyond 100 processors while delivering good results. We will show that the control of the move distribution is essential in breaking this barrier while the adaptive cooing components of the Chu algorithm are dispensable. We demonstrate our findings using the Rastrigin function as a test problem. The Rastrigin function provides a useful testbed for these investigations because it is difficult to optimize but does not have the heavy CPU requirements found in real world applications.

The rest of this paper is organized as follows. We start with details about the test problem, parameter set, and performance metrics used in this study. We introduce a key communication event in which states can multiply or be annihilated by resampling, and

survey the effects that changing the frequency of this resampling step has on the parallel simulated annealing algorithm. Based on these findings, we propose an algorithm to determine the interval of resampling adaptively, and analyze the performance of the proposed algorithm using the metrics we introduced. Finally, we conclude the paper with a discussion about implications and future work.

3.2 Methods

In this section we discuss the test function, details of the implementation, and how the performance is measured.

3.2.1 The Rastrigin Function

The n-dimensional Rastrigin function [48]

$$f(x) = 10n + \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i)] \tag{3.1}$$

is a family of multi-modal analytical test functions which is used in many case studies of simulated annealing algorithms [14, 58, 69, 77]. This family of functions has a global minimum at $f(0, \ldots, 0) = 0$ and the number of local minima grows exponentially with n. Since actual scientific applications [31, 35] take tens of millions of iterations to find a good result, we choose $n = 5000$ so that similar number of iterations will be required to achieve a good result.

3.2.2 Implementation Details

A complete simulated annealing algorithm consists of a cooling schedule, a move generation strategy, and a stopping criterion. A parallel version requires, in addition, a parallel strategy. Below we examine all the components of the Chu *et al.* implementation and explain how

they were altered to produce the implementation reported here.

The adaptive schedule in the Lam-Delosme algorithm [38, 39] relies on the variance and acceptance ratio of proposed moves to adjust the cooling rate. Because the temperature decreases at each step, variance is described by a local temperature dependent model that is valid over τ iterations. After τ iterations, the local model is updated using the results of the last τ iterations and others from further in the past with exponentially decaying weights. Chu's algorithm maintains the precise serial structure of the serial Lam algorithm with the sole exception that that the τ iterations of Metropolis are distributed over P processors. Computations outside the τ loop are, in the Chu algorithm, performed by each processor in lockstep using identical pooled data.

Escaping the hard limit of $P = \tau$ processors turns out to be incompatible with Lam and Delosme's method of estimating the variance in the face of continuously decreasing temperature, because we find that pooling energy statistics from different processors does not yield consistent estimates of variance in any test problem. This observation applies to parallel annealing in general, although our attention was originally drawn to the possibility of eliminating adaptive cooling by the observation that Lam's temperature dependent variance estimation procedure fails for the Rastrigin problem, even in serial.

Accordingly, in this work we use the geometric cooling schedule described in the original Kirkpatrick *et al.* paper [36], which specifies

$$T_{k+1} = (1 - \lambda)T_k. \tag{3.2}$$

where k is the iteration number and λ is a positive number very close to 0 that controls the cooling speed. The larger λ is, the faster the system cools. The adaptive schedule has a parameter with the same name and a similar role as a user defined control of overall cooling speed, but λ in an adaptive schedule is multiplied by factors dependent on annealing statistics and is not commensurate with λ defined above. Giving up adaptive cooling based

on variance will entail some loss of efficiency in serial, but Lam and Delosme show that their method of variance-based adaptive cooling produces a speedup of a factor of 2 to 3 on the Traveling Salesman Problem when the methods being compared both use adaptive move control [39, Table 5]. This loss of efficiency is much less than is gained by parallel speedup.

The central innovation of the method proposed in this work is to eliminate all adaptive cooling based on variance estimation used by Lam and Delosme in serial or Chu in parallel while retaining control of move generation. This is a reasonable strategy, because the control of move generation is arguably the most important feature in the Lam-Delosme algorithm [38, 39], and also in the parallelization by Chu *et al.* [16]. Move control is also vital in practice. Many applications contain cost functions in which parameters have different sensitivities, requiring different average move sizes. For these real world problems, controlling moves for each parameter separately is an absolute requirement [57]. A theoretical analysis shows that the variance of the energies of accepted moves, a measure of the effectiveness of searching parameter space, is greatest when the acceptance ratio $\rho = 0.44$ [38]. This calculation has been confirmed numerically for at least 4 test problems [39, 46], including this one.

The details of the move generation and control mechanism are as follows. At each step, a proposed move $x_i^{\text{proposed}} = x_i + \Delta x$ is generated by drawing moves from a Laplace distribution with mean 0 and variance $2\theta^2$ so that

$$\Delta x \sim \text{Laplace}(0, \theta), \tag{3.3}$$

where the average move size θ is adaptively adjusted by feedback control prescribed by Lam and Delosme [38, 39] such that

$$\ln \theta_{\text{new}} = K(\rho - 0.44) + \ln \theta_{\text{old}}, \tag{3.4}$$

where K is the proportional constant. Doing so will keep the acceptance rate ρ at the desired level of 0.44.

The mechanism of parallel speedup described here is the same as in the Chu algorithm, namely that the algorithm should sample the same number of states over the same temperature span in parallel as in serial. We achieve this with respect to cooling on P processors by setting $\lambda_P = P\lambda_s$, where λ_s is the base cooling speed in serial, whether the serial method is adaptive or otherwise. With respect to move control, in parallel we set $K_P = PK_s$ and ρ is computed from data pooled from all processors so that the average move size θ is the same across all processors.

To compensate for the more rapid cooling schedule in each processor, we resample the states among processors periodically according to the "mixing of states" algorithm proposed by Chu *et al* [16]. At each resampling, each processor receives the energy of every other processor. Then each processor p chooses a target i independently with probability

$$\frac{e^{-E_i/T}}{\sum_j e^{-E_j/T}}. \tag{3.5}$$

Then each processor adopts the state of its target and carries on annealing with its new state. When the temperature is high, every state has roughly equal probability being adopted. As the system cools, a better state is more likely to be adopted by multiple processors and hence be replicated across the system, while states with higher energies are more likely to be dropped. The interval between resampling, R, is discussed in detail in the next section.

An annealing algorithm stops when the system is considered "frozen". The selection of frozen conditions is largely problem dependent. For a problem with a discrete state space, the system freezes when no further moves are accepted. Because the Rastrigin function is a continuous function, there are always moves small enough to be acceptable. Thus, we use a stopping criterion κ such that the algorithm stops if it fails to change more than κ in $5n$ steps, where n is the number of dimensions. The parallel algorithm periodically checks the stopping criteria on all processors and stops when the first processor reaches the stopping criterion.

All numerical experiments start at $T_0 = 50000$ from a state, or states in parallel cases, randomly chosen from a pool of 1000 states. The states in the pool are prepared by running the Metropolis algorithm at T_0 for 100,000 steps, such that their energy distribution converges to a Boltzmann distribution at T_0 and that θ_0 is set to values that give $\rho \approx 0.44$ in each processor.

Both the base serial algorithm and the parallel version are implemented in C++ using templates, making them easy to use on other optimization problems. Communication in the parallel algorithm is implemented using MPI. In particular, the resampling of states uses one sided communication routines specified in MPI v2 and later, while all other communication uses blocking all to all routines. In addition, we use "processor" and "processing core" in an interchangeable fashion.

All the numerical experiments except for the results shown in Figure 3.1 were conducted on Beagle2, a Cray XE-6 system at the Computation Institute of the University of Chicago. The experiments shown in Figure 3.1 were conducted on Beast, a cluster managed by the Ecology and Evolution department of the University of Chicago.

3.2.3 Performance Measurement

The study of parallel efficiency requires an accurate estimation of the speedup, how much faster the computation can be performed in parallel versus serial. Speedup can be measured in terms of number of iterations or time. In this work we use both. For either measure, accurate measurement of speedup in parallel simulated annealing faces two main difficulties. First, we must ensure that parallel annealing is compared to the most efficient possible serial algorithm. Second, the comparison with the serial algorithm must take into account changes in the quality of the answer that come from paralellization.

In a maximally efficient serial annealing algorithm, the average quality of the answer depends on a single variable, the average number of iterations [16, 39]. Because the average number of iterations is not directly adjustable, it is necessary to obtain the desired rela-

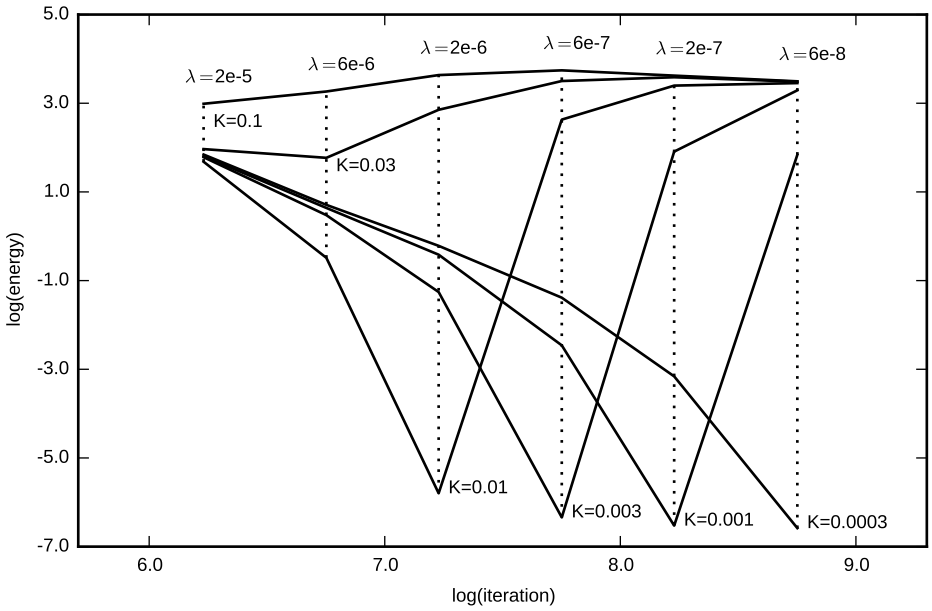


Figure 3.1: Serial performance boundary by varying K and λ. Annealing runs are conducted from a common temperature range from $T_0 = 10^5$ to $T_f = 10^{-10}$. The solid and dashed lines connect annealing runs with the same K and λ, respectively. The results show that the closer to the performance boundary, the more sensitive the results are from change of (K, λ) pair.

tionship by explicitly adjusting other quantities. In this work, those quantities are λ, the cooling speed, κ, the stopping criterion, and K, the move control feedback parameter. In practice, changing λ affects the number of iterations much more strongly than the quality of the answer [16] and so it is useful to set λ to the highest value that gives a reasonably good answer by suitable choice of K (Figure 3.1) and then obtain the serial performance curve of answer quality versus number of iterations N by varying κ (Figure 3.2). Inspection of Figure 3.1 shows that the combination of $\lambda_s = 2 \times 10^{-6}$ and $K_s = 0.01$ contains the largest λ that gives satisfactory results. We use these values for contructing the serial performance curve.

To calculate the speedup based on comparable serial results, we follow the method introduced by Chu *et al.* [16] to empirically measure the trade-off relation between the number of iterations in serial N_s and the final energy E_f. By varying the stopping criterion κ we

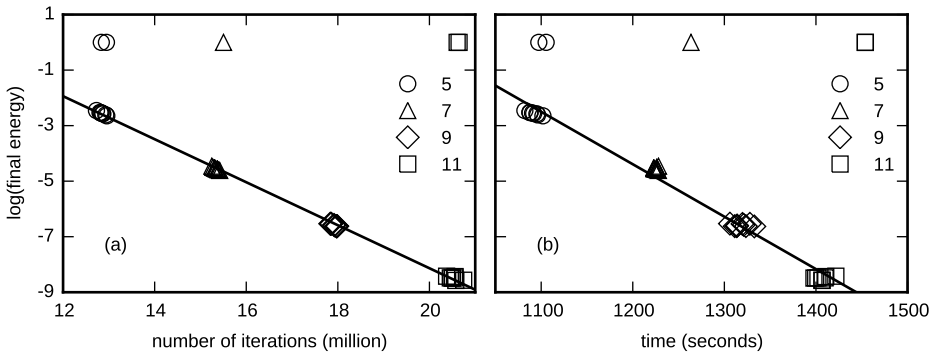


Figure 3.2: Serial performance curve in (a) number of iterations, and (b) time. Legends show the values of $-\log_{10}(\kappa)$. Those results with energy > 0.01 are considered failed and excluded from the fitting of the curves.

obtain the serial performance curve Figure 3.2a as

$$N_s(E_f) = (9.56 \times 10^6) - (5.55 \times 10^5) \ln E_f. \tag{3.6}$$

Once we have N_s as a function of E_f, the adjusted speedup in iteration S_N is given by

$$S_N = \frac{N_s(E_P)}{N_P}, \tag{3.7}$$

where E_P and N_P are the final energy and number of iterations of a parallel run on P processors. In addition to measuring algorithmic speedup in a manner that can be directly compared to the results of Chu *et al.* we also compute the speedup in terms of time. The trade-off between actual running time T_s and final energy E_f can be obtained from Figure 3.2b as

$$T_s(E_f) = 956 - 23.5 \ln E_f \tag{3.8}$$

and we define the adjusted speedup in time S_T as

$$S_T = \frac{T_s(E_P)}{T_P}. \tag{3.9}$$

3.3 Resampling Interval

In this section, we discuss the effect the resampling interval R has on the parallel simulated annealing algorithm and propose a new way to determine R adaptively.

3.3.1 Searching for an Optimal Interval

We first introduce the concept of success rate. Final energies from multiple annealing runs typically exhibit bimodal behavior, with the two peaks separated by multiple orders of magnitude. The higher energy peak represents a quenched state in which annealing has failed [16]. Given that the lowest quenched states in the higher peak have energies ≈ 0.995, and in practice the lower peak never goes beyond 0.01, we introduce a cutoff at $E_f = 0.01$ such that any runs better than the cutoff are considered successful. The success rate is defined as the number of successful runs divided by the total number of runs at each setting.

We characterize the effect of R in parallel simulated annealing by observing the success rates over a range of R for a variety of numbers of processors P. Figure 3.3a shows the success rate is high when R is small, and drops to 0 as R grows larger. The transition appears to become sharper and moves toward the origin as P increases. At $P = 192$ only $R = 1$ produces successful runs under the conditions described above. Resampling at $R < 1$ is undefined, and we saw no no successful runs at $P = 384$. For this particular test problem, $P = 192$ appears to be a hard limit.

For the successful runs, we calculate adjusted speedup both in terms of iterations and time according to (3.7) and (3.9). The speedup in iteration decreases slightly as R increases under the same P, but this decrease is small when compared over different Ps (Figure 3.3b). On the other hand, the speedup in time increases significantly as R increases, as one would expect from less frequent communication, until the success rate drops to 0 (Figure 3.3c). When selecting the best R for each case, the parallel algorithm shows good scalability in iterations up to 192 processors. The speedup in time, on the other hand, reaches a maximum at 21.3

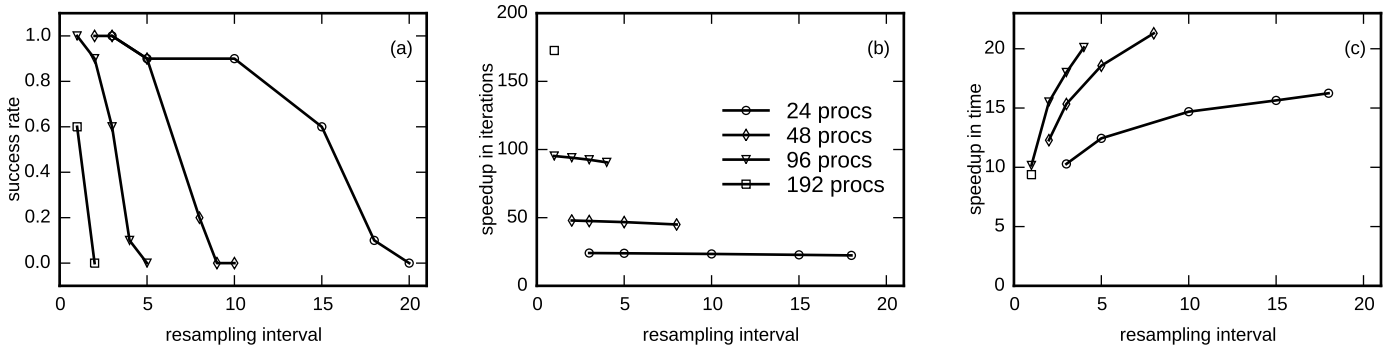


Figure 3.3: Effect of different resampling intervals on (a) success rate, (b) speedup in iterations, and (c) speedup in time. Legends are the same for all three panels.

on 48 processors and then decreases. This happens because, as P increases, the algorithm demands more frequent communication as Figure 3.3a shows, and each communication takes longer because of its all-to-all nature. In addition, the Rastrigin function requires only a moderate amount of computation (about 8 μs per call) for each evaluation. More expensive cost functions may generate a better speedup in time.

From the discussion above, it is clear that the selection of R is crucial for the performance of the parallel simulated annealing algorithm. It appears that R should decrease as P increases, and it is possible that the optimal value of R changes in the course of an annealing run. We addressed these issues by devising a method to adaptively determine a suitable R during each run in order to eliminate the necessity to tune yet another parameter in the algorithm.

3.3.2 Adaptive Resampling Interval

The idea behind the resampling of states is to allow the parallel algorithm to sample the search space more efficiently. As the temperature cools, the processors progressively abandon regions that are less likely to produce a good result and move to refine more promising areas of the state space. It is important to propagate a good state to other processors in a timely fashion to reduce wasteful computation. On the other hand, frequent communication will

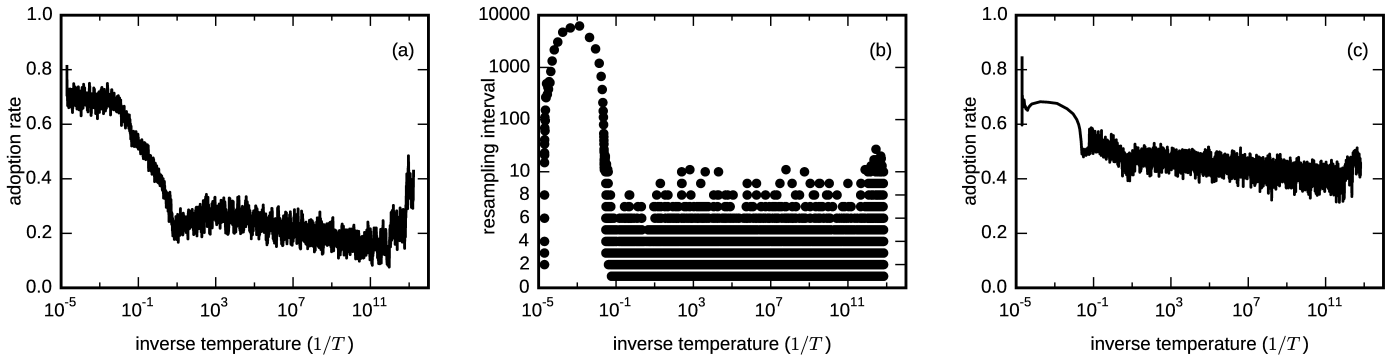


Figure 3.4: (a) Adoption rate for $R = 3$ on $P = 96$. (b) Resampling intervals in an adaptive algorithm on $P = 96$. (c) Adoption rate for adaptive algorithm. Data in (a) and (c) are smoothed by exponentially weighted moving average. Y-axis of (b) is linear between 0 and 10 and logarithmic from 10 above.

increase the overhead and reduce the performance of the parallel algorithm. In addition, maintaining diversity among states is also beneficial. In this section, we introduce a method of determining the resampling interval R by controlling the adoption rate.

The adoption rate A_R is defined as the number of distinct states that are adopted at a resampling divided by the total number of processors. $A_R = 1$ when all the states are adopted after a resampling, a case where states across the processors are reshuffled. In general, a high A_R means the states have roughly equal energies and hence may be insufficiently diverse. On the other hand, a low A_R means a small subset of states are substantially better than the others and hence that many processors are wasting computation. $A_R = 1/P$ takes the lowest possible value when all states adopt the same state after a resampling. Figure 3.4a shows typical A_R values over the course of a parallel annealing run.

From these arguments, it appears that the adoption rate will be a good indicator for determining the resampling interval. When A_R is large, we would increase R, and vice versa. Hence, we control the resampling interval using feedback control in a fashion similar to the move generation so that

$$\ln R_{n+1} = \ln R_n + C(A_R - A_0). \tag{3.10}$$

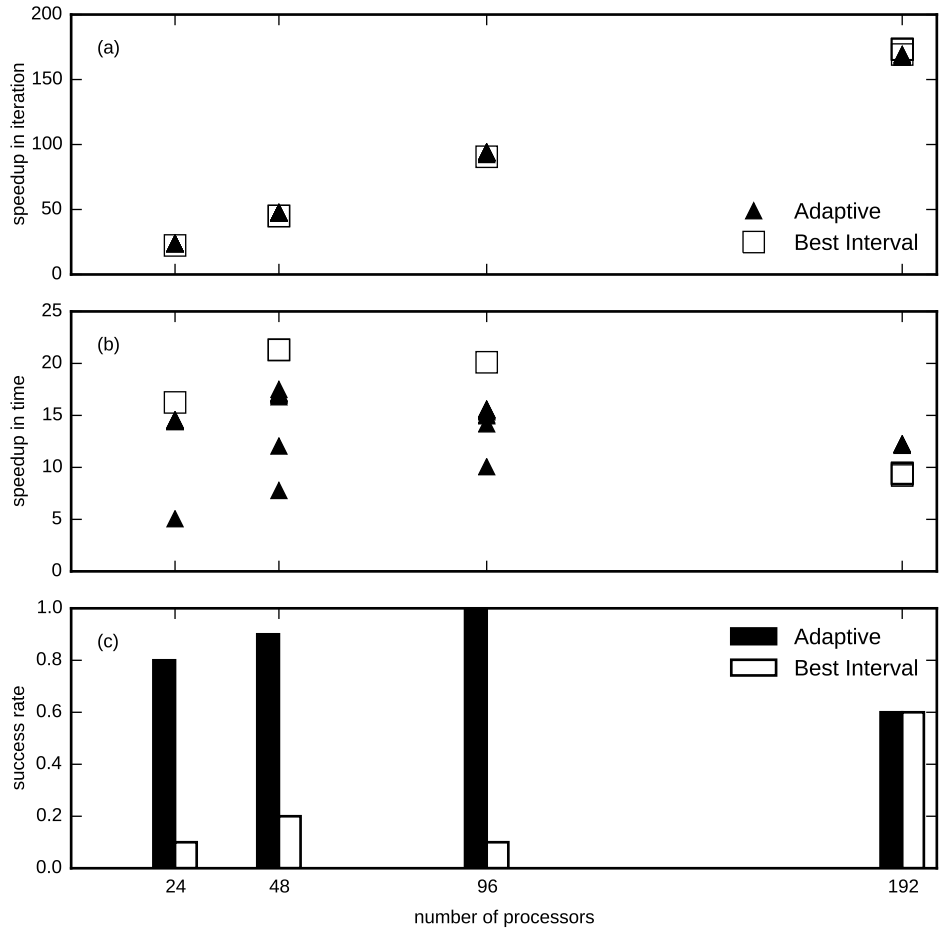


Figure 3.5: Comparison of adaptive resampling and the best resampling interval in (a) speedup in iteration, (b) speedup in time, and (c) success rate. The best intervals are 18 for 24 processors, 8 for 48 processors, 4 for 96 processors and 1 for 192 processors.

The target adoption rate A_0 was set to 0.5, and control constant $C = 2 \ln 2$ set such that $A_R = 1$ will cause R to double and $A_R = 1/P$ will almost half R. Figure 3.4b and c show the values of R and the resulting A_R respectively over the course of a typical parallel simulated annealing run with adaptive intervals.

The results show that such an adaptive algorithm performs well. The adjusted speedup in iteration of the adaptive interval is indistinguishable from the best R of the fixed interval algorithm (Figure 3.5a). Although the speedup in time lags the corresponding best R scheme for 96 processors and under (Figure 3.5b), it shows significant improvement in terms of success rate (Figure 3.5c).

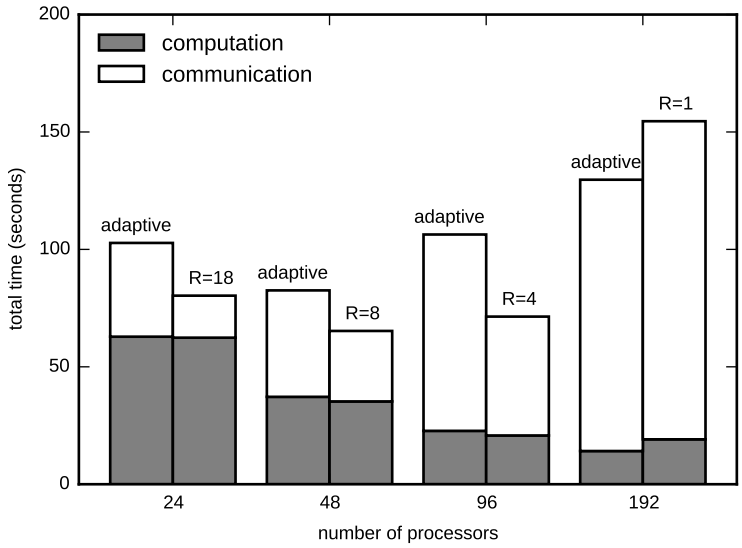


Figure 3.6: Comparison of time spent in computation and communication for adaptive resampling and the best resampling interval. The best intervals are 18 for 24 processors, 8 for 48 processors, 4 for 96 processors and 1 for 192 processors.

3.4 Discussion

We have demonstrated that it is possible to achieve 90% parallel efficiency up to 192 processors in terms of iterations for a high-dimensional non-convex optimization problem under a strictly defined speedup test. This is achieved by using a nonadaptive geometric cooling schedule along with Lam's feedback move generation control, while aggressively reducing the resampling interval. Speedup in real time is smaller, exhibiting parallel efficiency that declines with the number of processors. This behavior is expected when communication costs increase. To confirm this explanation, we measured the computation time and communication time for 24, 48, 96, and 192 processors in both the adaptive and best R scheme in Figure 3.6. For a real application which is more CPU intensive, the impact from the communication overhead will be less severe. Moreover, communication overhead is architecture dependent. Our chief aim in this work is to demonstrate the architecture-independent properties of the algorithm.

We have also demonstrated that it is possible to control the resampling interval adaptively

based on the adoption rate of the last resampling. Such an adaptive method results a high success rate and does not adversely affect speedup in iteration. Speedup in time is slightly worse compared to the best fixed interval method for 96 processors and under, but is marginally better for 192 processors.

In addition, we have shown that the use of success rate as a primary metric is beneficial. Although the strongly bimodal behavior of the final energies and the concept of success rate have been discussed previously [16, 33], performance analysis in these works is still largely focused on the average final energy over successful runs. By promoting the success rate to a primary metric, we identified a significant transition of performance over a small range of resampling intervals, while other measurements such as the average final energy and adjusted speedup do not change much.

The major drawback of this algorithm is that it performs best in a narrow region of (λ, K) pairs. Such regions can only be found by empirical experimentation. The sensitivity of the performance according to the (λ, K) pair is a side effect of operating the simulated annealing at its efficiency boundary. When using smaller (λ, K) pairs, the sensitivity goes down as well as the efficiency of the algorithm. Fortunately, in applications the simulated annealing algorithm is used to minimize a series of closely related cost functions, typically arising from a closely related set of models for the same set of data. Over this set of cost functions, the same (λ, K) pair performs roughly the same. Hence in practice, only one round of empirical experiments are needed to obtain a satisfactory (λ, K) pair that can be used repeatedly. Nevertheless, automatic determination of suitable (λ, K) pair, ideally from a limited sample of the search space, is a direction worth exploring.

The results reported here show that the inherent scalability limitations of the Chu algorithm can be overcome by eliminating adaptive cooling but retaining control of move generation. These results were obtained using a test problem chosen for small computation cost. While advantageous for rapid experimentation, small computation costs led to high communication overhead. The scalability limitation to 192 processors that was observed be-

cause resampling cannot be performed more frequently than once each iteration appears to be related to the choice of test problem. Despite these limitations, we believe that our results open the door to the removal of scalability limitations across a wide range of applied problems by using the same methods. The choice of a small test problem was important because the large CPU demands of global optimization problems make methodological investigations of this type computationally expensive at minimum. If key components of the algorithm, such as estimators of variance, also fail to converge, such methodological investigations are also inconclusive. Future work will show if the methods developed here can be effective in real applications.

CHAPTER 4

APPLICATIONS IN REAL PROBLEMS

The previous chapter demonstrates the performance and scalability of the parallel simulated annealing using a adaptive resampling interval on the pilot test problem of the Rastrigin function. Although it is highly non-convex, the Rastrigin function still features a additive cost function with homogeneous parameter space. A real optimization problem often does not afford such luxuries. In practice, an optimization problem usually has a non-homogeneous, non-additive parameter space. Moves on different parameters will have different but inter-dependent effects on the cost function. This chapter discusses how the parallel simulated annealing algorithm performs facing these challenges from two real life optimization problems.

4.1 Problem Description

4.1.1 The Drosophila *pattern formation model*

The history and motivation of this project is directly linked to the *Drosophila* pattern formation model[31]. In this model, a gene circuit is represented by the element T^{ab} of a matrix indicating the regulatory effects of gene b on gene a. The model is written as a system of ordinary differential equations (ODEs)

$$\frac{dv_i^a}{dt} = R_a g_a \left(\sum_{b=1}^{N} T^{ab} v_i^b + m^a v_i^{bcd} + h^a \right) + D^a(n) \left[(v_{i-1}^a - v_i^a) + (v_{i+1}^a - v_i^a) \right] - \lambda_a v_i^a \quad (4.1)$$

where v_i^a is the concentration of the protein corresponding to the gene a at nucleus i, R_a is the regulatory coefficient, D^a is the diffusion coefficient, λ_a is the decay coefficient and g_a is the threshold function

$$g_a(u) = \frac{1}{2} \left[\frac{u}{\sqrt{u^2 + 1}} + 1 \right]. \quad (4.2)$$

The cost function to be minimized is given by

$$E = \sum_{\text{all data points}} (v_i^a(t) - v_i^a(t)_{\text{data}})^2 + \begin{pmatrix} \text{penalty} \\ \text{terms} \end{pmatrix} \tag{4.3}$$

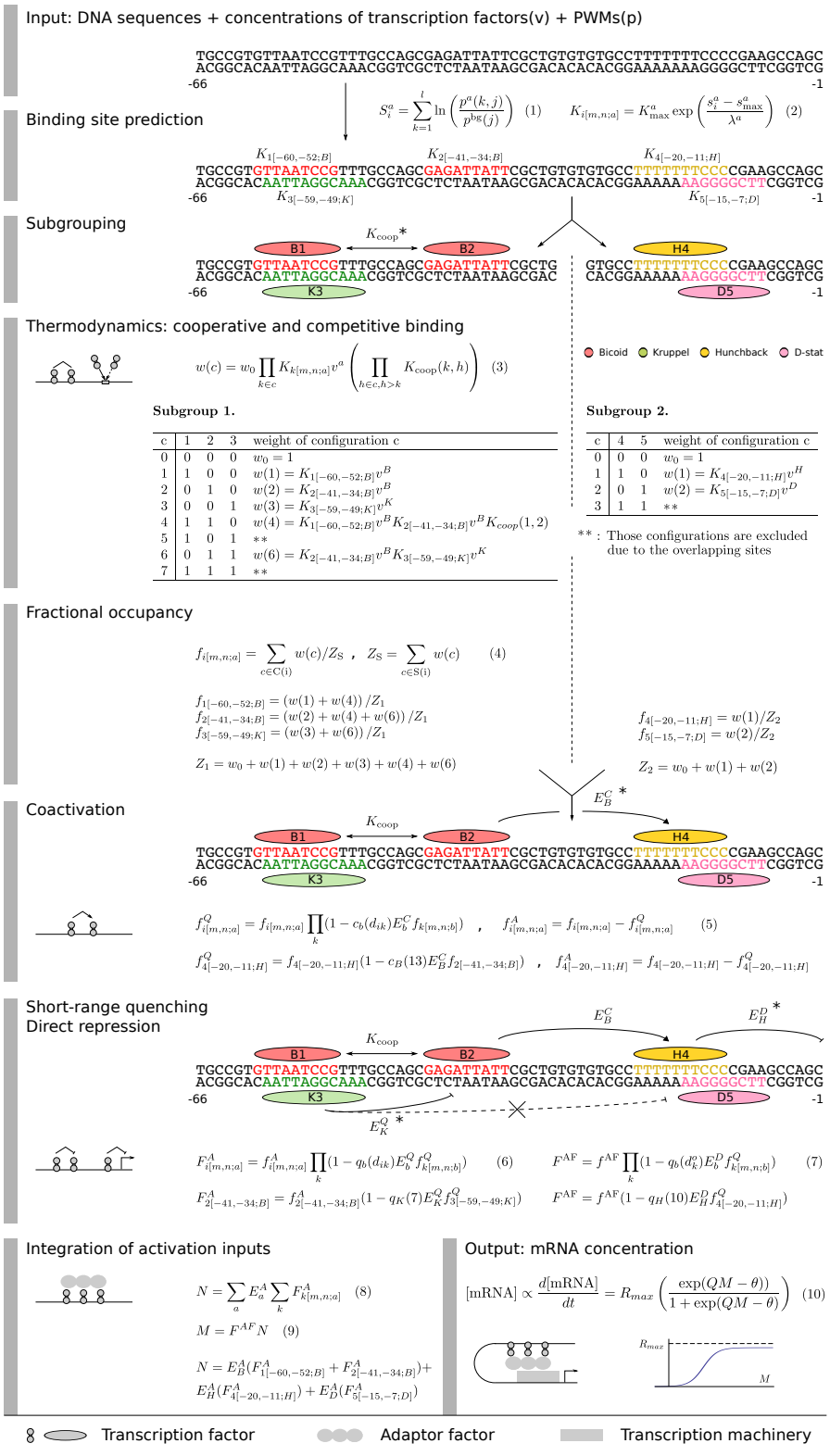
where the first term sums the squared deviation of the solutions to (4.1) for all a, i, and t for which the data exists and the penalty terms defines the boundary of the search space. For a problem concerning N genes on M number of nuclei, this model contains $N \times M$ simultaneous equations with $O(N^2)$ free parameters.

This thesis uses a test problem taken directly from the 2009 study of canalization of gap genes by Manu *et al* [44, 45]. It models the interaction between 4 genes using 40 parameters of different physical and biochemical meanings. The model prediction is then compared to about 2000 experimental data points to generate the score of the cost function. On average, each cost function evaluation of the problem takes about 2 milliseconds running on a single core of a Intel Xeon E5-2670 CPU at 2.3GHz.

4.1.2 The transcription model

The transcription model goes deeper into the understanding of the same phenomena as the pattern formation model. It models the effects of protein–DNA binding in the control region of a gene and the effect of the configuration of bound proteins on the transcription of mRNA by that gene. Thus, the transcription rate is calculated from a multi-layer, feed forward equation [32, 35, 54]. Figure 4.1 gives a detailed explanation of the model equation. The cost function is then defined as the sum of the squared difference between the model outputs and the experimental data.

This model can be used as a different optimization test problem as its mathematical structure is unrelated to the pattern formation model. The calculation of its cost function involves a straightforward function evaluation, albeit a complicated one. It will have a different CPU intensiveness pattern than the pattern formation model. The cost function



c	1	2	3	weight of configuration c
0	0	0	0	$w_0 = 1$
1	1	0	0	$w(1) = K_{1[-60,-52;B]} v^B$
2	0	1	0	$w(2) = K_{2[-41,-34;B]} v^B$
3	0	0	1	$w(3) = K_{3[-59,-49;K]} v^K$
4	1	1	0	$w(4) = K_{1[-60,-52;B]} v^B K_{2[-41,-34;B]} v^B K_{coop}(1,2)$
5	1	0	1	**
6	0	1	1	$w(6) = K_{2[-41,-34;B]} v^B K_{3[-59,-49;K]} v^K$
7	1	1	1	**

c	4	5	weight of configuration c
0	0	0	$w_0 = 1$
1	1	0	$w(1) = K_{4[-20,-11;H]} v^H$
2	0	1	$w(2) = K_{5[-15,-7;D]} v^D$
3	1	1	**

Figure 4.1: Detailed explanation of the transcription model. Adapted from Figures 2 and 3 of A.-R. Kim *et al.* (2013) [35].

does not contain a penalty term. Instead, each parameter has a predetermined range such that an out of range state will be rejected immediately. In addition, the use of threshold in the binding site prediction makes the energy response to the change of the corresponding parameters highly nonlinear. In particular, there are lots of states in the search space that will result an energy that corresponds to an output with 0 mRNA concentration.

This thesis uses the problem taken from the 2013 genomic *cis*-regulatory logic study by Kim *et al* [35] as a second test problem. It has 46 parameters, and the model output is compared to 406 observations. On average, each cost function takes about 4 ms for each cost function evaluation on a single core of Intel Xeon E5-2670 CPU.

4.2 Performance measurement

Before studying the performance and scalability of the parallel simulated annealing algorithm on these test problems, it is necessary to define how the performance is measured. This section details the equations for the adjusted speedup, and thresholds for the success rate for each problem, similar in spirit to that defined in section 3.2.3 and 3.3.1, yet capable of alleviate the difficulties originating from the use of real world problems.

4.2.1 Serial performance curve

The application of performance adjusted speedup (3.7) requires empirical measurements of the problem specific serial performance curve that represents the trade-off between the number of iterations and the quality of final results. Generally speaking, at the efficiency boundary, the closer the final energy E is from the minimum, the more iterations an optimization algorithm would take for the same amount of energy improvement. Thus, the construction of the trade-off curves for these two test problems incorporates a measurement

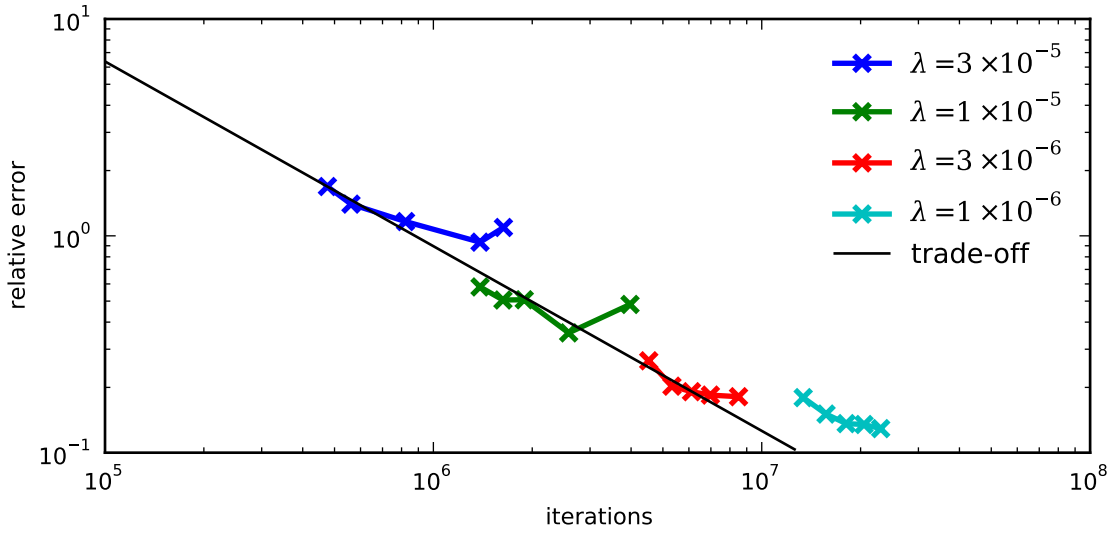


Figure 4.2: Serial performance curve for the pattern formation model problem. Each data point corresponds to the median number of iterations and relative error over 100 independent runs. For each line with the same λ value, the data points reflect the effects of κ from left to right being 100, 10, 1, 0.1, and 0.01.

of the closeness to the global minimum by using the relative error, which defined as

$$E_r = \frac{E - E_0}{E_0}. \tag{4.4}$$

However, the global minimum E_0 for these two test problems, like many real world optimization problems, are not known in advance. In addition, although the global minimum for each problem should in theory exist, due to the noise presented in the data, a collection of states whose scores are close to the minimum usually has superior value than the minimum itself. According to these considerations, it is reasonable to use the minimal value over all results obtained as an approximation of the global minimum value E_0.

Under this measurement, the construction of the serial performance curve for the pattern formation model problem means finding the trade-off relation between the number of iteration N_s and the final relative error E_r when the serial annealing algorithm is maximally efficient. Since neither of these values are directly adjustable, it is necessary to obtain the relation indirectly by varying the cooling speed λ and stopping criterion κ simultaneously.

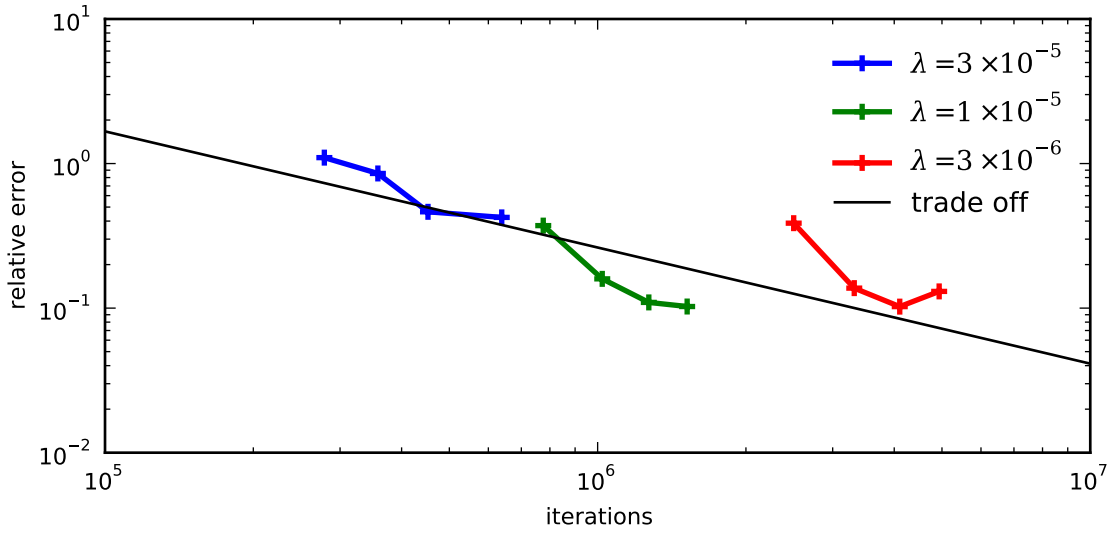


Figure 4.3: Serial performance curve for the transcription model problem. Each data point corresponds to the median number of iterations and relative error over 100 independent runs. For each line with the same λ value, the data points reflect the effects of κ from left to right being 100, 10, 1, and 0.1.

Experiments reveal that within the range $\lambda \in [3 \times 10^{-6}, 3 \times 10^{-5}]$ and $\kappa \in [0.1, 100]$, a change in either parameter will make results move alone a common curve in the N_s-E_r plot (Figure 4.2). Pairs of (λ, κ) out of the range all fall on the upper-right side of the curve, which means they are less efficient. Thus, fitting the median N_s and E_r for each pair of (λ, κ) within the range gives the serial performance curve

$$N_s(E_r) = (8.79 \times 10^5) E_r^{-1.17}. \tag{4.5}$$

A similar procedure can apply to the transcription model problem (Figure 4.3). The trade-off curve for this problem is

$$N_s(E_r) = (8.10 \times 10^4) E_r^{-1.16}. \tag{4.6}$$

4.2.2 Success threshold

Although the statistics of a series of independent annealing runs are stable, for a fixed configuration, the results of individual runs could vary over orders of magnitudes. Under many situations, a few unfortunate runs will have very poor results, causing an out-sized weight in the average. In practice, however, those runs rarely matter. Most of the problems will be run many times and only those which perform adequately well will be used. In the literature, there are many case studies employ a threshold to denote the concept of a successful runs and either use the ratio of successful runs[14, 19, 64] or take the average on those runs[11, 12, 16]. Similar to section 3.3.1, this thesis combines both methods and considers the success rate and the average speedup for those successful runs simultaneously.

For the Rastrigin function, the threshold is obvious as both the global minimum and the second lowest minimum values are known from the cost function. For the test problems in the chapter, such an approach is not possible. Instead, an empirical threshold can be obtained by observing the histogram of the results of each problem under various configurations. Figures 4.4 and 4.5 demonstrate this for both problems. Although these relative errors spread over multiple orders of magnitude, their histograms consistently show two-peak distributions with a low density valley in between that persists over different settings. Such structure suggests some intrinsic barriers in the search space. Hence, it is reasonable to set a threshold for each problem at above described valley. For the pattern formation model problem, the threshold is at $E_r = 1.5$ and for the transcription model problem, it is at $E_r = 0.5$.

Once the threshold is defined, runs with relative errors below their corresponding threshold can be considered successful. The success rate can be defined as the number of successful runs divided by the total number of runs at a particular setting. The adjusted speedup for that setting is the average adjusted speedup over all successful runs.

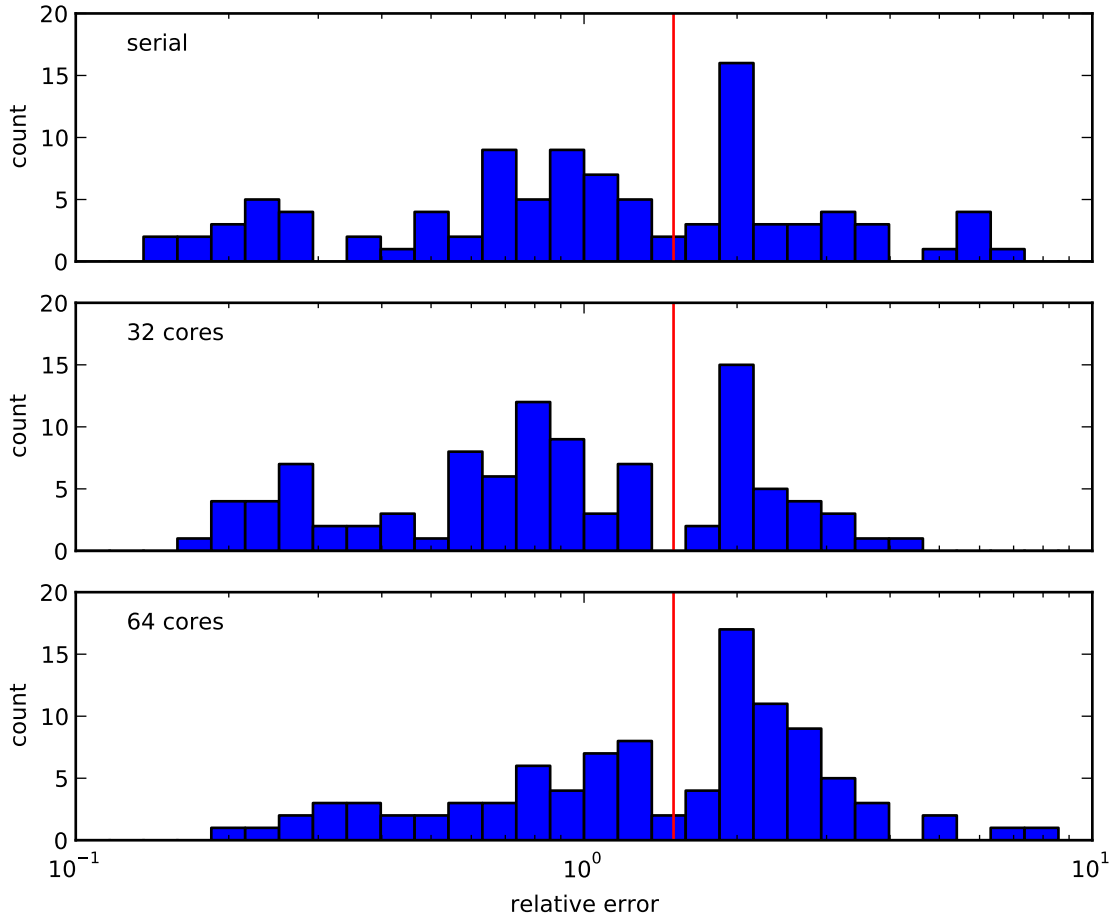


Figure 4.4: Distribution of the relative errors of the pattern formation model problem over 100 runs each for serial, 32 cores, and 64 cores. The serial runs have $\lambda_{\text{serial}} = 3 \times 10^{-5}$, while the parallel runs have $\lambda_{32} = 9.6 \times 10^{-5}$ and $\lambda_{64} = 1.92 \times 10^{-4}$. The vertical red line is the proposed threshold at 1.5.

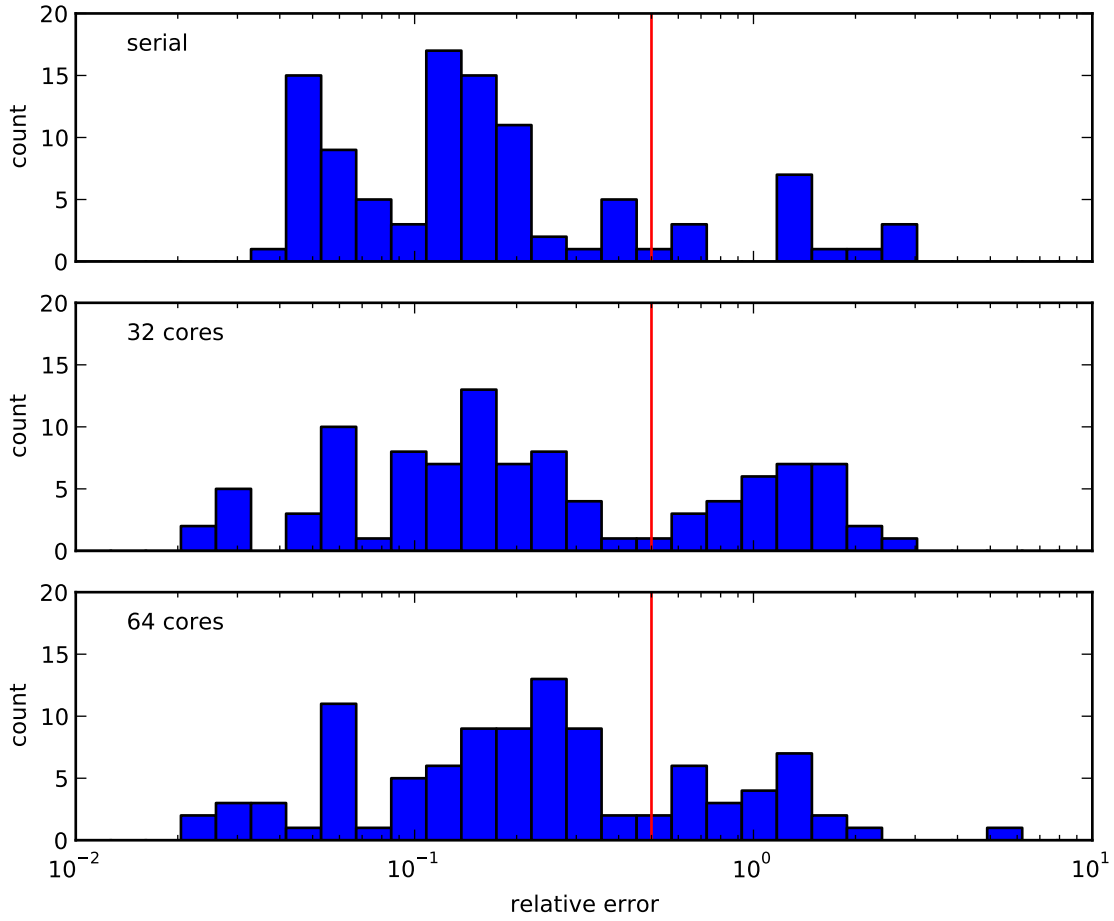


Figure 4.5: Distribution of the relative errors of the transcription model problem over 100 runs each for serial, 32 cores, and 64 cores. The serial runs have $\lambda_{\text{serial}} = 3 \times 10^{-6}$, while the parallel runs have $\lambda_{32} = 9.6 \times 10^{-5}$ and $\lambda_{64} = 1.92 \times 10^{-4}$. The vertical red line is the proposed threshold at 0.5.

4.3 Challenges in move generation and control

As stated in section 3.2.2, move generation control is one of the central components in a scalable parallel simulated annealing algorithm. Naturally, over the annealing process, as the temperature cools down, the average move sizes are expected to shrink accordingly. However, since the current move control scheme only relies on the acceptance ratio of a number of previous steps, it is not directly aware of the temperature or the cooling speed. As a result, when the cooling speed increases along with the number of cores used, the move control may not be able to keep up. In the test on the Rastrigin function, this is exactly what happened, and is solved by setting the move control parameter for P processor cores $K_P = P \cdot K_s$, P times larger than that in the serial case K_s.

For real problems, this simple scaling is not sufficient. Since their search spaces are inhomogeneous, each dimension of a search space has to control and maintain its own average move size. Therefore, the move generation algorithm needs to propose moves and accumulate acceptance statistics on each dimension in turn. One such loop over all parameters is called a "sweep". For a d-dimensional problem conducting move control every I_{mc} sweeps, it totals dI_{mc} iterations between two successive move control events. In practice, I_{mc} is set to 100, which means it takes 4000 and 4600 iterations respectively between move controls for the pattern formation model problem and the transcription model problem, regardless of the cooling speed. For parallel runs, that many iterations means the acceptance statistics are collected over a much wider temperature range as the number of processor cores increases. The resulting acceptance ratio will become less accurate. Coupled with a larger K from the scaling strategy, such inaccuracy will cause instability in the feedback control of the average move size once P is large (Figure 4.6a), and cause a severe drop in the quality of the results.

One way to work around such scaling problem is to reduce I_{mc}. In parallel, all the acceptance statistics are pooled together ahead of a move control. The actual data available for the move control to act upon for each dimension is $P \cdot I_{\text{mc}}$ for a P-core parallel annealing run. As P goes up, it is no longer necessary to keep I_{mc} constant. Reducing I_{mc} to 10 and

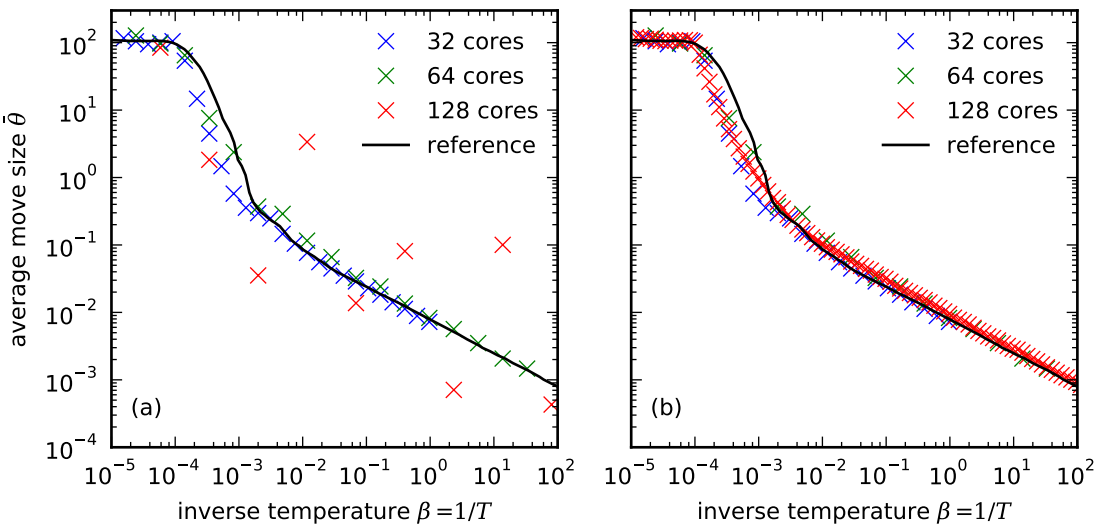


Figure 4.6: Demonstration of the effects of the move control interval I_{mc} has on the average move size over the annealing process. Figures show the average move size of the *bicoid* coactivation in the transcription model problem on 32, 64, and 128 cores compared to the reference line calculated by averaging 20 successful runs in serial. The move control gain parameter K is set to $0.1P$ where P is the number of cores. (a) All runs are set to $I_{\text{mc}} = 100$. (b) Same figure but 128-core case is now set to $I_{\text{mc}} = 10$.

correspondingly, reducing K_P value to 1/10 of the previous value results an average move size curve which is in line with these from smaller Ps, including the reference curve made by averaging multiple runs in serial (Figure 4.6b). Certainly, this process cannot continue indefinitely, as smaller I_{mc} demands more frequent communication and $I_{\text{mc}} = 1$ will be the minimal possible value.

After the change of I_{mc}, it is important to re-investigate the effect of K on the parallel algorithm. This investigation is carried out by examining both the speedup and success rate for runs with various K values over 2 orders of magnitudes on different Ps under both $I_{\text{mc}} = 10$ and 100 cases(Figure 4.7 and 4.8). For simplicity, all the numerical experiments are conducted using an adaptive resampling interval. As the next section will show, this method is as good as or better than the best fixed resampling interval. The results shows that, the case in $I_{\text{mc}} = 10$ results in both better speedup and higher success rates under almost all of the K values tested for both problems. The two problems, however, behave

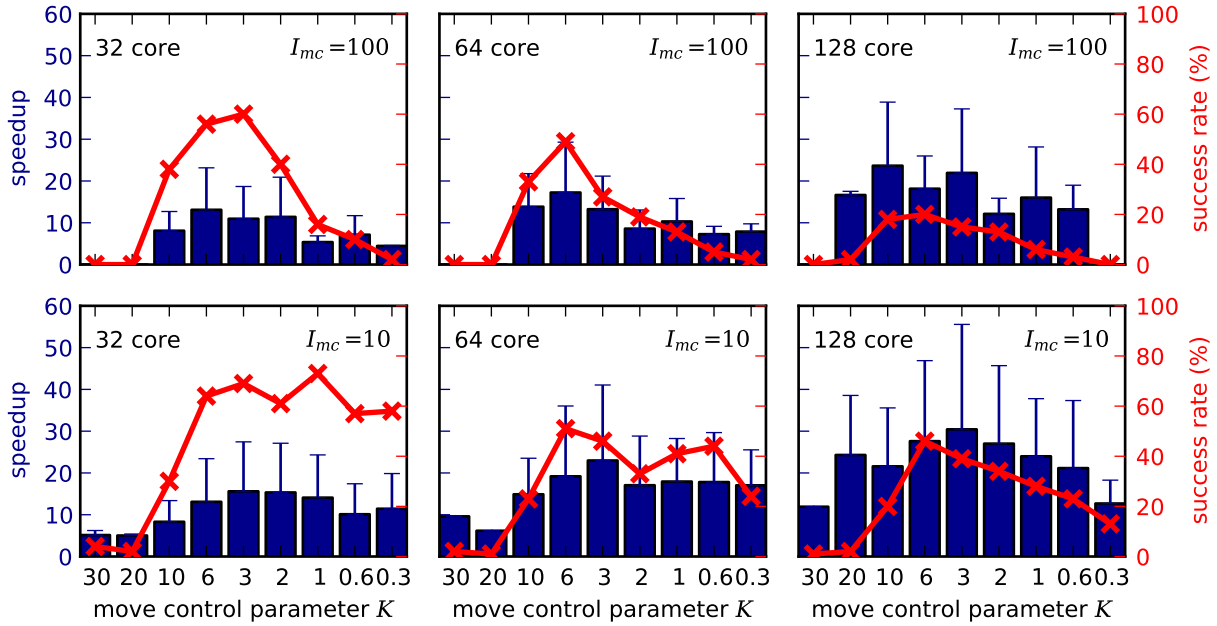


Figure 4.7: The effects of different move control parameter K on the pattern formation model. The results are presented for the mean and standard deviation of adjusted speedup (blue, left axis) and success rate (red, right axis) for 32, 64, and 128 core runs over move control intervals $I_{\mathrm{mc}} = 100$ (top) and $I_{\mathrm{mc}} = 10$ (bottom). Cooling speed is set to $\lambda_P = P\lambda_s$ where $\lambda_s = 3 \times 10^{-6}$. All runs were performed using adaptive resampling intervals.

differently when considering the optimal K values as a function of P under $I_{\mathrm{mc}} = 10$. For the pattern formation model problem, $K = 3$ provides the best speedup while keeping the highest or second highest success rate in all P values tested (Figure 4.7). On the other hand, the optimal K values for the transcription model problem grow slowly as P increases (Figure 4.8). It is possible to approximate this relation as $K = 0.3\sqrt{P}$, which falls back to $K = 0.3$ when $K = 1$, the optimal value for the serial case at $I_{\mathrm{mc}} = 10$.

The following performance measurement will follow the conclusion of this section. For both test problems, I_{mc} is set to 10. For the pattern formation model problem, $K = 3$, and for the transcription model problem, $K = 0.3\sqrt{P}$ rounded to the nearest integer.

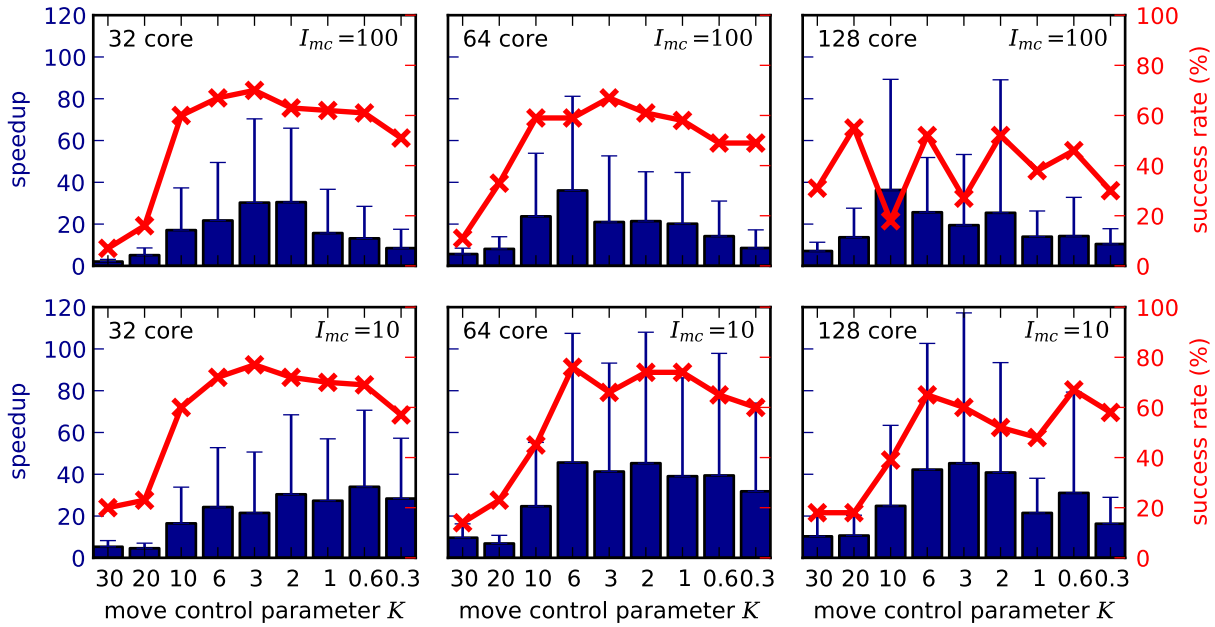


Figure 4.8: The effects of different move control parameter K on the transcription model. The results are presented for the mean and standard deviation of adjusted speedup (blue, left axis) and success rate (red, right axis) for 32, 64, and 128 core runs over move control intervals $I_{\mathrm{mc}} = 100$ (top) and $I_{\mathrm{mc}} = 10$ (bottom). Cooling speed is set to $\lambda_P = P\lambda_s$ where $\lambda_s = 3 \times 10^{-6}$. All runs use using adaptive resampling intervals.

4.4 Performance of adaptive resampling intervals

After determining the move control parameters, parallel performance can be compared between different fixed resampling interval Rs and the adaptive resampling intervals over various number of cores for both problems. The implementation details for the adaptive resampling interval is exactly the same as described in section 3.3.2, including the values of A_0 and C used in (3.10).

For the pattern formation model problem, the effect of different fixed resampling intervals is moderate (Figure 4.9). For a fixed P, setting the resampling interval R to the lower end of the range results in similar speedup and success rate. These performance metrics only decline when R become large. Nonetheless, the adaptive resampling interval method is able to perform as well as the best fixed intervals for each cases.

The effect the resampling intervals have to the performance is more apparent in the transcription model problem, especially in cases where P is large (Figure 4.10). At 32 cores, the speedup is less sensitive to the change in R, similar to those in the pattern formation model. The speedup become more sensitive to R at 64 and 128 cores. The difference in speedup between the best and the worst cases are much larger, and the speedup as a function of R become much steeper as well. In addition, there is a clear shift of the optimal R value from large to small as P increases. It is clear that the performance response to different R values is very different in the transcription model problem from the pattern formation model. Nonetheless, the adaptive resampling interval method, with the same set of parameters, is still able to perform with comparable speedup against the best case in the fixed resampling intervals, with a slightly reduced success rate that is still higher than 60%.

Finally, scaling the parallel algorithm with adaptive resampling intervals up to 256 cores for both problems tests the scalability of the algorithm in the real world problems. On the pattern formation model problem, it is able to sustain a moderate speedup growth while the success rates drops as the number of cores increases (Figure 4.11). On the transcription model problem, the algorithm is able to result a much higher success rate yet the speedup

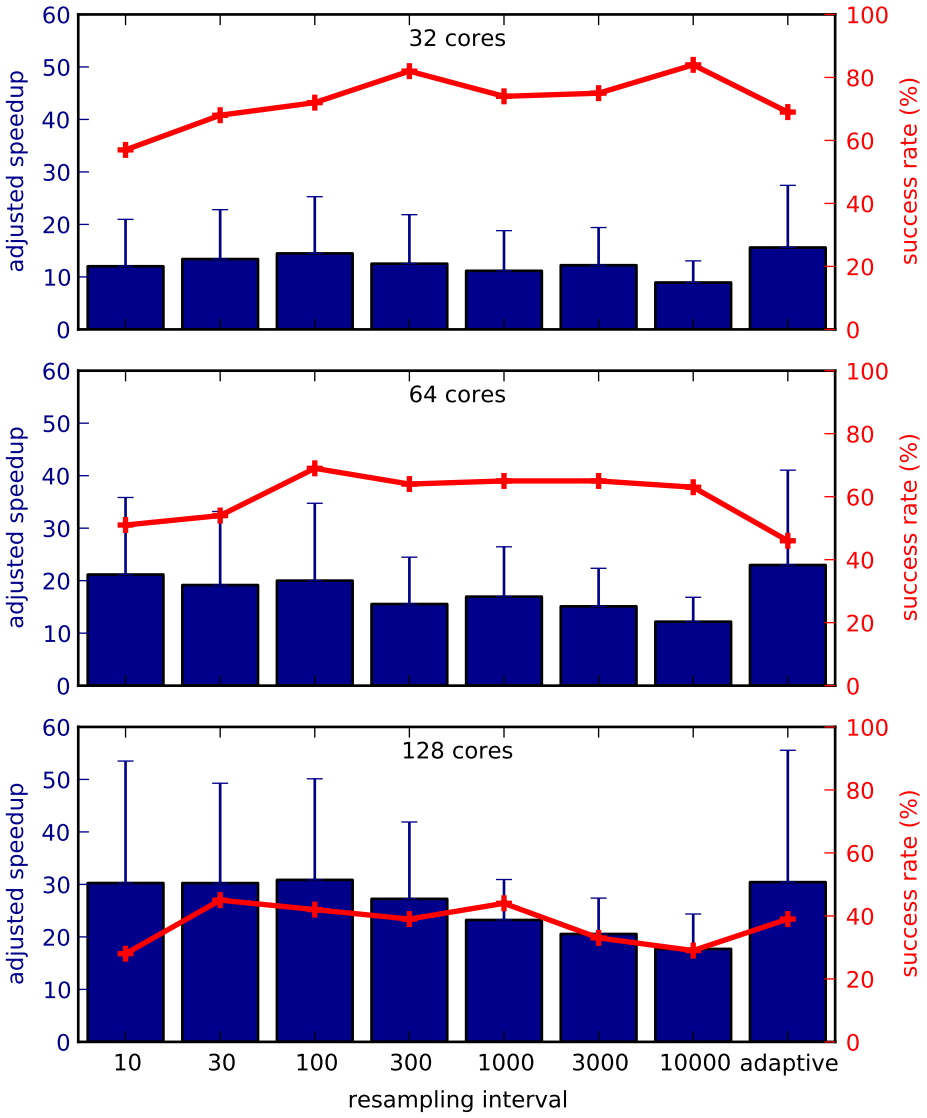


Figure 4.9: Comparison of fixed and adaptive resampling interval for the pattern formation model. The results is presented for the mean and standard deviation of adjusted speedup (blue, left axis) and success rate (red, right axis) over fixed and adaptive resampling intervals on 32, 64, and 128 cores.

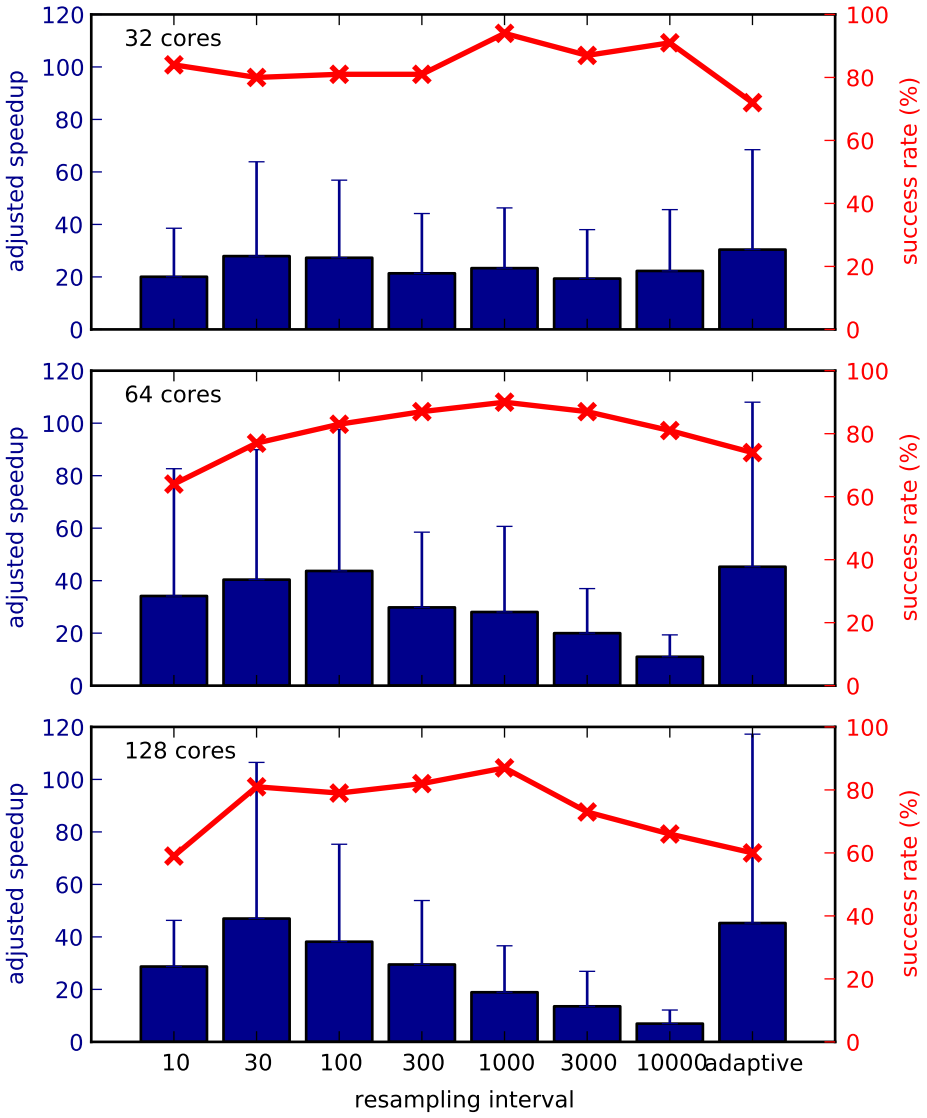


Figure 4.10: Comparison of fixed and adaptive resampling interval for the transcription model. The results is presented for the mean and standard deviation of adjusted speedup (blue, left axis) and success rate (red, right axis) over fixed and adaptive resampling intervals on 32, 64, and 128 cores.

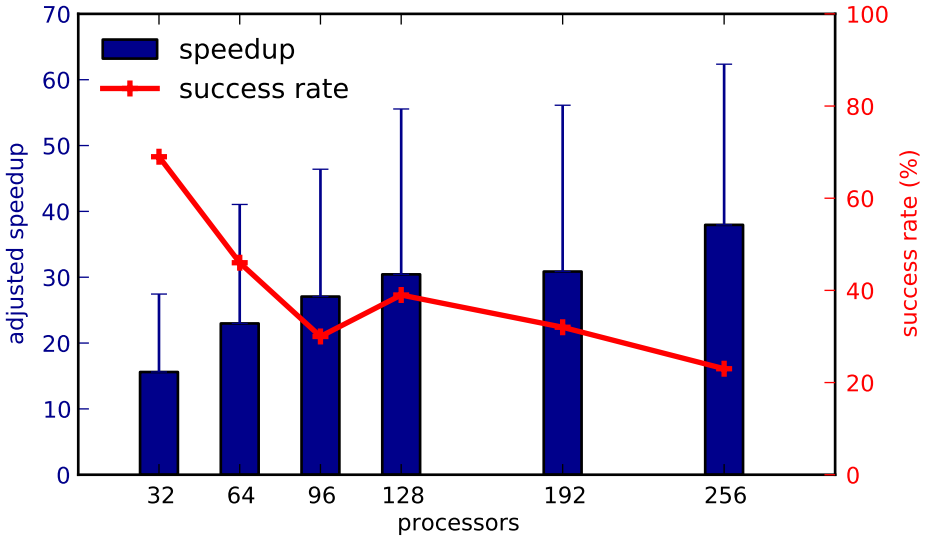


Figure 4.11: Scalability of the parallel simulated annealing in the pattern formation model.

stagnates at around 40 starting from 64 cores.

4.5 Conclusion

This chapter has demonstrated in two different real world test problems that using adaptive resampling intervals is able to consistently match or exceed the performance of those using a fixed resampling interval. Since the parameters used for determining the intervals adaptively are the same for both test problems, in addition to the Rastrigin function problem described in chapter 3, this algorithm effectively removes the resampling interval as a tunable parameter for the annealing algorithm while delivers the best performance previously only available by a laborious tuning method.

The results from the two test problems also demonstrate that it is possible to achieve non-trivial speedup using parallel simulated annealing on real world, high-dimension, non-linear optimization problems. Some may argue that the original Chu algorithm is able to report higher speedup despite its limitations on the number of cores can be used [16]. However, that experiment was tested on a simplified model problem with 13 parameters. The speedup reported was only achieved by selecting the best performing resampling intervals

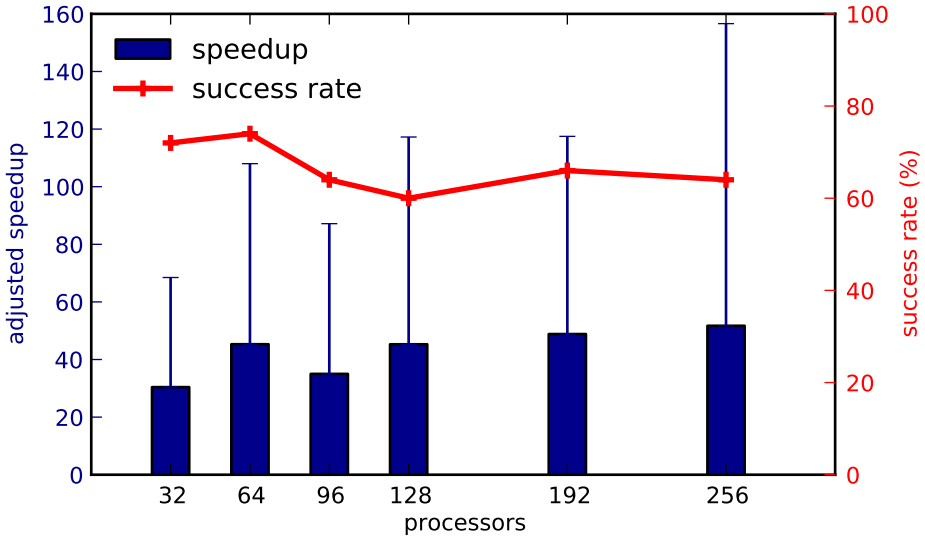


Figure 4.12: Scalability of the parallel simulated annealing in the pattern formation model

over repeated trials of different values. In contrast, the problems tested in this chapter are full models directly adapted from systems biology research, each problem has a higher number of dimensions, and the resampling intervals are determined adaptively with no tuning required.

In addition, the study in this chapter shows that the move generation and control algorithm has a profound impact on the performance of the simulated annealing algorithm. In parallel, especially when using a larger number of cores, predicting an optimal move control parameter before actually running the annealing algorithm is difficult. Considering that move generation is problem dependent, an optimal move control scheme has to depend on the structure of the search space of the problem being optimized. Thus, this difficulty demands an algorithmic method of understanding the structure of the search space. The next chapter will discuss the attempts in achieving this goal.

CHAPTER 5

DENSITY OF STATES ESTIMATION

5.1 Motivation

Previous chapters explored the application in parallel simulated annealing algorithm using the adaptive resampling intervals on multiple test problems. It can produce near perfect speedup for simple test problems and respectable speedup even for difficult real world optimization problems. However, it also proves that it is hard to find appropriate values for the move control interval I_{mc} and gain parameter K. In fact, the optimal K as a function of the number of cores P varies from linear in the Rastrigin function to the square root of P in the transcriptional model problem and became constant in the pattern formation model problem. Since the move control is obviously problem dependent, addressing these problems beyond laborious parameter tuning will require some programmatic method of studying the search space of the problem being optimized.

As the simulated annealing algorithm is inspired by the physical annealing process, it is natural to look for inspiration in the realm of thermodynamics. Consider a continuous optimization problem whose search space has $\Omega(E)\,\mathrm{d}E$ states between energy $[E, E + \mathrm{d}E)$. When each state in the search space is considered as a distinct microstate, the canonical ensemble at inverse temperature $\beta = 1/T$ has the partition function

$$Z(\beta) = \int \Omega(E) e^{-\beta E}\,\mathrm{d}E. \tag{5.1}$$

Once the partition function is obtained, there are a variety of thermodynamic variables that can be calculated which offer insight into the structure of the search space of the problem being optimized. In particular, the mean and variance of the energy of the system at inverse

temperature β are

$$\langle E(\beta)\rangle = -\frac{\partial \ln Z(\beta)}{\partial \beta} \tag{5.2}$$

$$= \frac{1}{Z(\beta)} \int E\Omega(E) e^{-\beta E}\, \mathrm{d}E, \tag{5.3}$$

and

$$\sigma^2(\beta) = \frac{\partial^2 \ln Z(\beta)}{\partial \beta^2} \tag{5.4}$$

$$= \frac{1}{Z(\beta)} \int E^2\Omega(E) e^{-\beta E}\, \mathrm{d}E - \langle E(\beta)\rangle^2, \tag{5.5}$$

and the heat capacity of the system can also be obtained by

$$C_v = \beta^2 \sigma^2(\beta). \tag{5.6}$$

Recall from section 2.1.3 that the Lam-Delosme algorithm uses the inverse of the heat capacity as a part of their adaptive cooling schedule. However, the variance is difficult to estimate in parallel, and the adaptive cooling schedule is hence removed from the parallel algorithm, as discussed in section 2.3.1. The discussion above shows that once the density of states $\Omega(E)$ of a problem is obtained, the variance and heat capacity can be calculated from $\Omega(E)$ instead. This opens the door to reviving the adaptive cooling schedule without the difficulties associated with the direct estimation of variance. The rest of this chapter details the method for such estimation, the application of this estimation to the test problems presented in the earlier chapters, and what can be inferred from the study.

5.2 Methods of estimation

In statistical mechanics, the study of molecular simulations, especially those for the study of phase transitions and critical phenomena, frequently requires the study of the energy

landscape. This field is one of the frontier areas where the latest methods for estimation of density of states are developed. Traditionally, such estimates relied on running the Metropolis algorithm directly to generate a canonical distribution at a given temperature. Latter, Berg *et al.* has introduced multi-canonical ensemble method for efficient sampling for difficult energy landscape [10]. More recently, Wang and Landau proposed an approach for direct calculation of the density of states using a random walk in energy space [66, 67]. The method was developed for a lattice system with discrete energy levels. Later studies expanded the method into continuous systems with a joint density of states either directly [75, 76], by using a distribution [78], or a family of orthogonal basis functions [70, 71]. Singh *et al.* provided a nice review on this topic [60].

Although the later algorithms generally perform better than the original Wang-Landau sampling, they all relied on some physical properties like free energy or force field that do not exist on the optimization problems. In contrast, the original Wang-Landau sampling is much more friendly for adaptation to the optimization problems due to its simplicity and sparse assumptions. The rest of this section details both the original Wang-Landau sampling and the adaptation to estimating the density of states in non-physical systems.

5.2.1 Wang-Landau Sampling

The Wang-Landau sampling is based on the observation that for a system with density of states $\Omega(E)$, if states are sampled with probability $\propto \frac{1}{\Omega(E)}$, then the resulting histogram for the energy distribution will be flat. To achieve this, the algorithm maintains $g(E)$, a running estimate of $\Omega(E)$, and samples the states with probability $\frac{1}{g(E)}$ using the Metropolis-Hastings algorithm. The estimated density of states $g(E)$ is updated every step. The update factor is reduced gradually over the estimation process and stops when the factor become smaller than some predetermined threshold.

If there is some *a priori* knowledge about the density being estimated, $g(E)$ can start from the corresponding distribution. Otherwise, in most cases, $g(E)$ is set to the uniform

distribution, or if starting from where a previous run left off, the distribution resulting from the previous run. At step i, the algorithm proposes a new state with energy E_p from the current state with energy E_i. The new state is accepted with probability

$$P(E_{i+1} = E_p) = \min\left[\frac{g(E_i)}{g(E_p)}, 1\right] \tag{5.7}$$

Otherwise, the old state is kept and $E_{i+1} = E_i$. Regardless of the acceptance, the estimated density at E_{i+1} is updated by a factor of f

$$g(E_{i+1}) \leftarrow f g(E_{i+1}). \tag{5.8}$$

The modification factor f is set to start with $f_0 = e = 2.71828\ldots$, which will allow for fast exploration of the energy landscape. Note that since the acceptance is updated every step and hence is dependent on all the previous steps, this process is not a Markov chain.

During the estimation, the algorithm also maintains a histogram $H(E)$, the number of visits at each energy E. When $H(E)$ is considered sufficiently flat, the modification factor is updated

$$f_{i+1} = \sqrt{f_i}, \tag{5.9}$$

and $H(E)$ is reset to 0 for another sweep of simulation. The flatness of the histogram $H(E)$ is defined when all entries $H(E)$ is not less than $x\%$ of the mean of the histogram, where x is a problem dependent value. The algorithm stops when f is sufficiently small.

In practice, for numerical stability, the algorithm maintains the logarithm of density instead, and the update (5.8) becomes

$$\ln[g(E_{i+1})] \leftarrow \ln[g(E_{i+1})] + \ln f. \tag{5.10}$$

In thermodynamics, the entropy is defined as

$$S(E) = k_B \ln \Omega(E) \tag{5.11}$$

where k_B is the Boltzmann constant. In optimization problems without a thermodynamic unit, we can set $k = 1$ and consider the log-density being estimated, $\ln g(E)$, as the approximation of the entropy.

Note that in the Wang-Landau sampling, only the ratio between two density values $g(E_1)/g(E_2)$ is used in the calculation. Thus, the resulting estimates is only accurate up to a normalization constant. In the log-density case, that means different estimates may differ by an additive constant.

5.2.2 Density of States estimation in optimization problems

Since the original Wang-Landau sampling is developed for lattice systems with discrete energy values, there are multiple strategies to adapt it to a continuous energy system. The simplest way is to divide energies into equal width bins and all the rest of the aspects of the Wang-Landau sampling can be preserved.

Alternatively, we can consider the estimated log-density at the discrete energy E_i up to step t as a weighted sum over all iterations up to time t given by

$$\ln g(E_i; t) = \sum_{\tau<t} W(\tau)\delta(E_i, E(\tau)), \tag{5.12}$$

where $W(\tau) = \ln f$ in step τ, $E(\tau)$ is the energy visited at step τ, and δ is the Kronecker delta. Thus, if the delta function is replaced by some function $I(E, E_\tau)$ that peaks at $E = E\tau$ and rapidly decreases as $|E - E_\tau|$ grows larger, a continuous estimate of the log-density can be obtained as

$$\ln g(E; t) = \sum_{\tau<t} W(\tau)I(E, E_\tau). \tag{5.13}$$

The common choices of I are Gaussian [78], rescaled Legendre polynomials $L_n(x)$ [70], or rescaled Hermitian polynomials [71].

The continuous density function offers advantages in analysis, but that comes with a heavy toll on the computation. It requires the storage of all the past visited energies. Density calculation is also proportional to the number of past visits in the energy vicinity, which could be huge in high density region. Hence, the density of states study of the optimization problems are simply dividing the energies into equal width bins.

When running alongside parallel simulated annealing, it is natural to parallelize the Wang-Landau sampling as well. The parallelization is based on the additive nature of $\ln g$ as shown in (5.12). Since $\ln g$ serves both as the running estimate of the log-density as well as a histogram of visited energies, the parallelization can separate the estimate of $\ln g$ into a global part and a local part

$$\ln g(E) = \ln g_{\text{global}}(E) + \ln g_{\text{local}}(E). \tag{5.14}$$

Each core updates its own local part of $\ln g$ while the estimated log-density are calculated from the sum of local and global densities. Periodically, the global value is updated by adding all the local log-density to each bin and then broadcasting it back to all cores. Each local estimation is then reset to 0 for the next sampling.

5.3 Analyses of Test Problems

5.3.1 Rastrigin function

One of the benefits of using Rastrigin function as a pilot problem is that it has an explicit cost function. That means for an n-dimensional Rastrigin function, its partition function

can be written in the closed form

$$Z(\beta) = e^{-10n\beta} \left(\int_{-5.12}^{5.12} e^{-\beta[x^2 - 10\cos(2\pi t)]} \, \mathrm{d}x \right)^n. \tag{5.15}$$

Many of its theoretical properties can be obtained from this partition function. In particular, its mean energy, energy variance, and heat capacity at any inverse temperature can be calculated numerically using (5.2) and (5.4). These provide a reference in studying the correctness of the density of states estimation.

Although the calculation of density of states is difficult even for those with explicit cost functions, for the 1-dimensional Rastrigin function there is still some structural information about the density that can be inferred from the cost function. Consider a quadratic cost function $E = x^2$, the length over the region such that $x^2 < E$ is $2\sqrt{E}$. That is,

$$\int_0^E \Omega(E) \, \mathrm{d}E = 2\sqrt{E}. \tag{5.16}$$

Differentiation under the integral sign gives

$$\Omega(E) = \frac{1}{\sqrt{E}}, \tag{5.17}$$

which means

$$\lim_{E \to 0} \Omega(E) = \infty. \tag{5.18}$$

The local minima and maxima of the 1-dimensional Rastrigin function are approximately quadratic. Hence the density at those corresponding energy values approach infinity. Figure 5.1 indeed shows spikes at corresponding energy values in the estimated log-density.

Since the original Kirkpatrick paper, the heat capacity or specific heat became an indicator in helping determine the cooling schedule [36]. In that paper, the specific heat is calculated by taking the derivative with respect to temperature of the average energy. From the estimated density of states, heat capacity can be calculated from (5.5) and (5.6) instead.

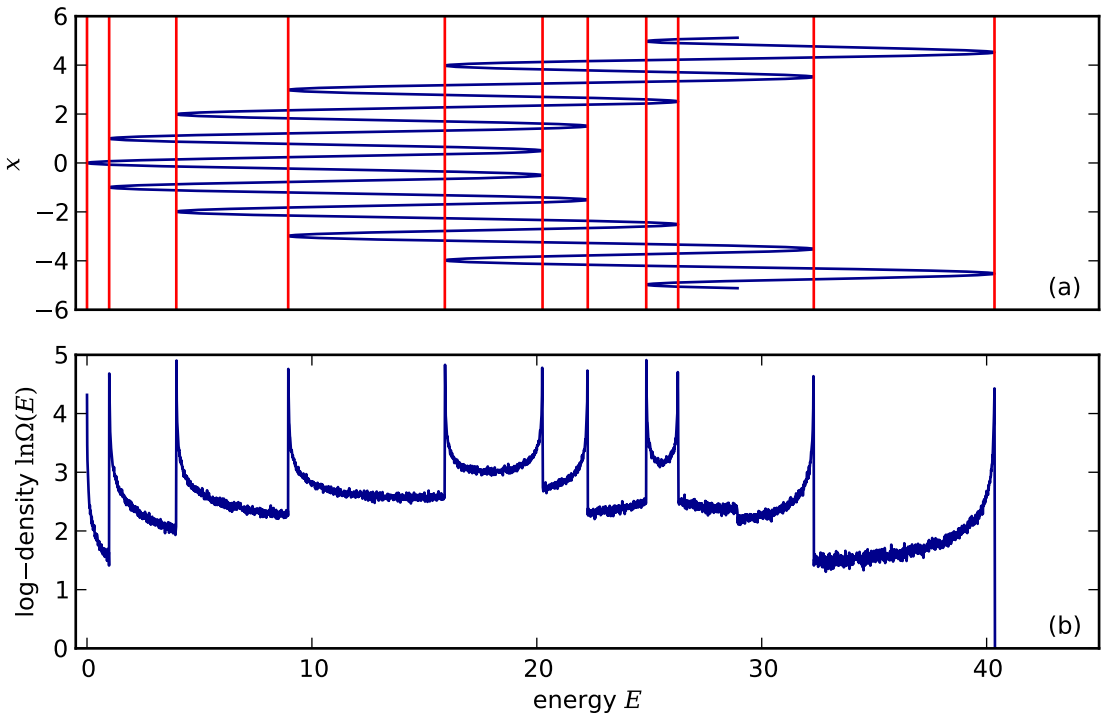


Figure 5.1: (a) The 1-dimensional Rastrigin function with energy on the horizontal axis and x on the vertical axis. (b) Estimated entropy for 1-dimensional Rastrigin function. The entropy estimate is conducted on 10 cores for a total of 10 million iterations. Note the peaks in (b) correspond to the local minima and maxima in (a).

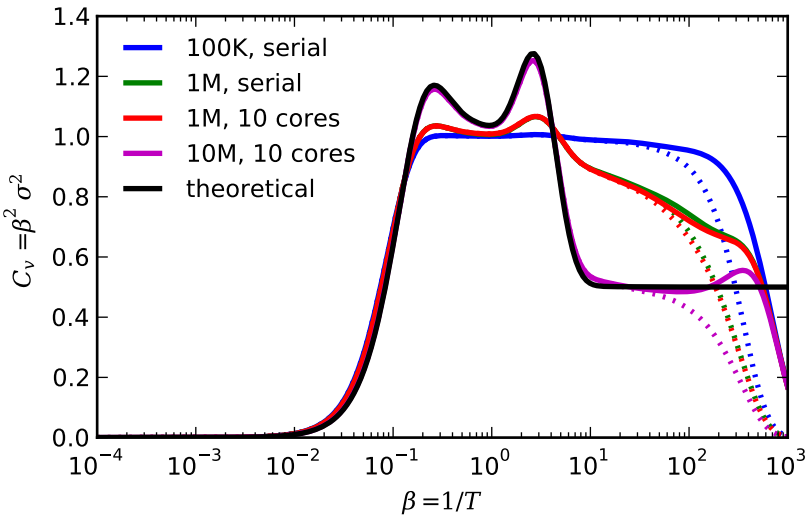


Figure 5.2: Estimated heat capacity for the 1-dimensional Rastrigin function compared to its theoretical value. Estimates are calculated from density of state estimates of 100 thousand and 1 million iterations in serial and 1 million and 10 million iterations in parallel using 10 cores. During the numerical integration, dashed lines are calculated using the rectangular method while the solid lines of the same color are using the trapezoidal method on the same data. Note the curve for 1 million serial (green) and parallel (red) coincide almost completely, and the curve for 10 million (magenta) overlaps a large portion of the theoretical curve (black).

Figure 5.2 shows heat capacity curves calculated from estimated density agree progressively better to the one calculated from the partition function as the number of iterations invested in the estimation grows larger. Note that all the estimated heat capacity curves approach 0 at low temperature, instead of the constant value obtained from theoretical calculation. This artifact is likely caused by the inaccuracy introduced during the numerical calculation. Changing the numerical integration method from the rectangular method to the trapezoidal method when applying (5.5) will make the best estimated curve (magenta curve in Figure5.2) agree with the theoretical curve for an additional order of magnitude as the temperature is lowered. In addition, the curve representing 1 million samples in serial and the one with 1 million samples in total over 10 cores overlap almost completely. This suggest the parallel estimation is as accurate as the serial one.

For a Rastrigin function of a higher dimension, estimating heat capacity directly from sampled density of states will become difficult. Since the number of states grows exponentially with the number of dimensions, samples of any reasonable size can only cover a small subset of the actual energy range (Figure 5.3a). The calculated heat capacity will deviate from the theoretical value as soon as the mean energy drops outside of the sampled energy range (Figure 5.3b). In this situation, some help from modeling will be necessary. We can consider the value of the 1-dimensional Rastrigin function as a random variable with a probability density proportional to $\Omega_1(E)$. Because of its additive nature, the value of the n-dimensional Rastrigin function can thus be considered as the sum of n independent and identically distributed random variables. When written in iterative form, we have

$$E_n = E_{n-1} + E_1, \tag{5.19}$$

where E_1, E_{n-1}, and E_n are the values of 1-dimensional, $(n-1)$-dimensional, and n-dimensional Rastrigin functions respectively. Thus, the density of states of the n-dimensional

Rastrigin function at energy level E is

$$\Omega_n(E) = \int \Omega_{n-1}(E-x)\Omega_1(x)\,\mathrm{d}x, \tag{5.20}$$

which is a convolution of the lower dimensional ones, so that

$$\Omega_n(E) = \Omega_{n-1}(E) * \Omega_1(E). \tag{5.21}$$

As n increases, the spikes which appeared in Figure 5.1 will be smoothed out, and the density $\Omega_n(E)$ converges to a normal distribution as $n \to \infty$ according to the central limit theorem. Figure 5.3a shows that the estimated log-density for $n = 5000$ case is indeed quadratic, which corresponds to a normal density. However, calculating the mean and variance of the density directly from implied mean energy and energy variance at infinite temperature ($\beta = 0$) turns out to be less accurate. Instead, fitting the mean and variance in log-space using a least square fit can show a better performance.

Although both the theory and the estimated density of states strongly suggest a normal density for the 5000-dimension Rastrigin function, modeling a optimization problem using a normal density function will likely miss many important structural features. Consider a density of states in the form of $N(\mu, \sigma)$. Its Boltzmann distribution at β is

$$\begin{aligned} \Omega(E)e^{-\beta E} &\propto e^{-\frac{(E-\mu)^2}{2\sigma^2} - \beta E} \\ &\sim N(\mu - \sigma^2\beta, \sigma). \end{aligned} \tag{5.22}$$

This means when using normal distribution, the resulting model will have constant variance over all temperature. In particular, the heat capacity will increase monotonically as temperature decreases and will not suggest any phase transition behavior. That is because the normal distribution is symmetrical and does not have a finite minimum. On the other hand, the Gamma distribution behaves similar to the normal distribution near the mode while

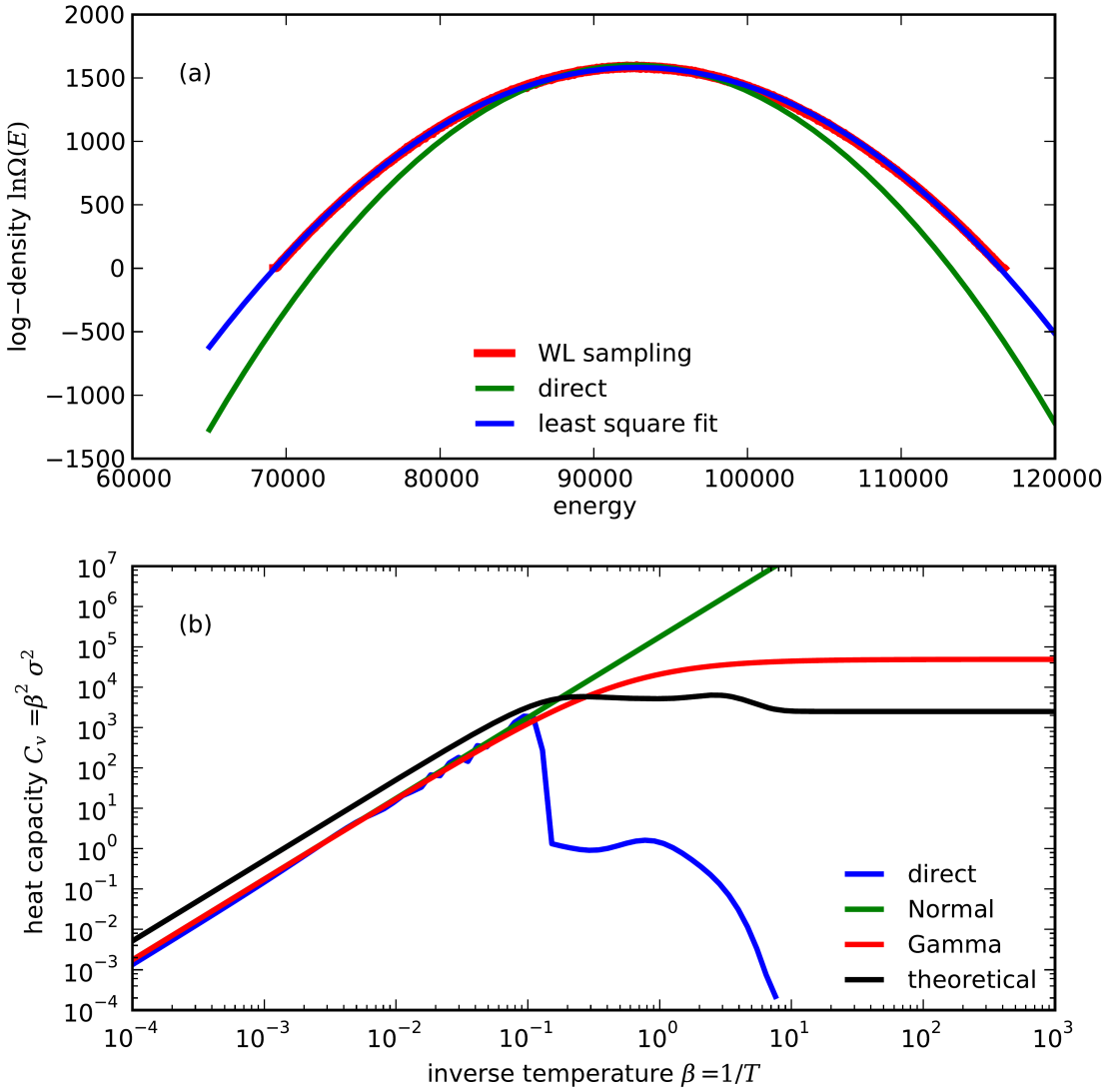


Figure 5.3: (a) Estimated log-density for 5000-dimension Rastrigin function. The estimate was based on 12 cores over 12 billion iterations in total. The raw data are compared to the density modeled as a normal distribution from both direct calculation of mean and variance and also by fitting a least square fit. (b) Calculated heat capacity of the 5000-dimension Rastrigin function over a range of temperature points. The figure shows the value from direct calculation using (5.5) and also by modeling the density as normal or Gamma distributed with same mean and variance.

has a fixed finite minimum on the left side. Hence, if the density of states is modeled as a Gamma distribution $\Omega(E) \sim \Gamma(a, b)$, its Boltzmann distribution at β will be

$$\begin{aligned} \Omega(E)e^{-\beta E} &\propto E^{a-1}e^{-bE-\beta E} \\ &\sim \Gamma(a, b+\beta) \end{aligned} \tag{5.23}$$

That is, the variance at β is $\frac{a}{(b+\beta)^2}$. Figure 5.3b shows that a Gamma distribution with the same mean and variance at infinite temperature as a normal distribution would behave similar at higher temperature while its heat capacity will gradually levels out as its temperature lowers. This largely falls in line with the fact that the Rastrigin function at low temperature behaves like a single quadratic minimum, which have a constant heat capacity. Though not perfectly accurate, the heat capacity predicated by the Gamma distribution is able to identify the temperature regions where the heat capacity is growing, and where the heat capacity stabilizes at a high level. When applying the Lam-Delosme cooling schedule (2.13) that sets the cooling speed to be proportional to the inverse of heat capacity, data from the method described above will generate a cooling schedule more efficient than the constant cooling speed in the exponential schedule.

5.3.2 Application in Gene Regulation Problems

Once the method is validated on the Rastrigin functions, it is natural to apply the method to analyze the real world optimization problems raised from the gene regulation models, the pattern formation model and the transcriptional model. The log-density estimated for each problem reveals its own structure.

The log-density of the pattern formation model problem shown in Figure 5.4a is a smooth curve going upwards as the energy increases. This increase at high energy states corresponds to the region where the penalty term overwhelms the overall score. The further a state lies from the desired region, the larger the energy, but also the larger the equal-penalty ellipsoid. Hence the increase in the states in these cases. The lower energy end shows some distinct

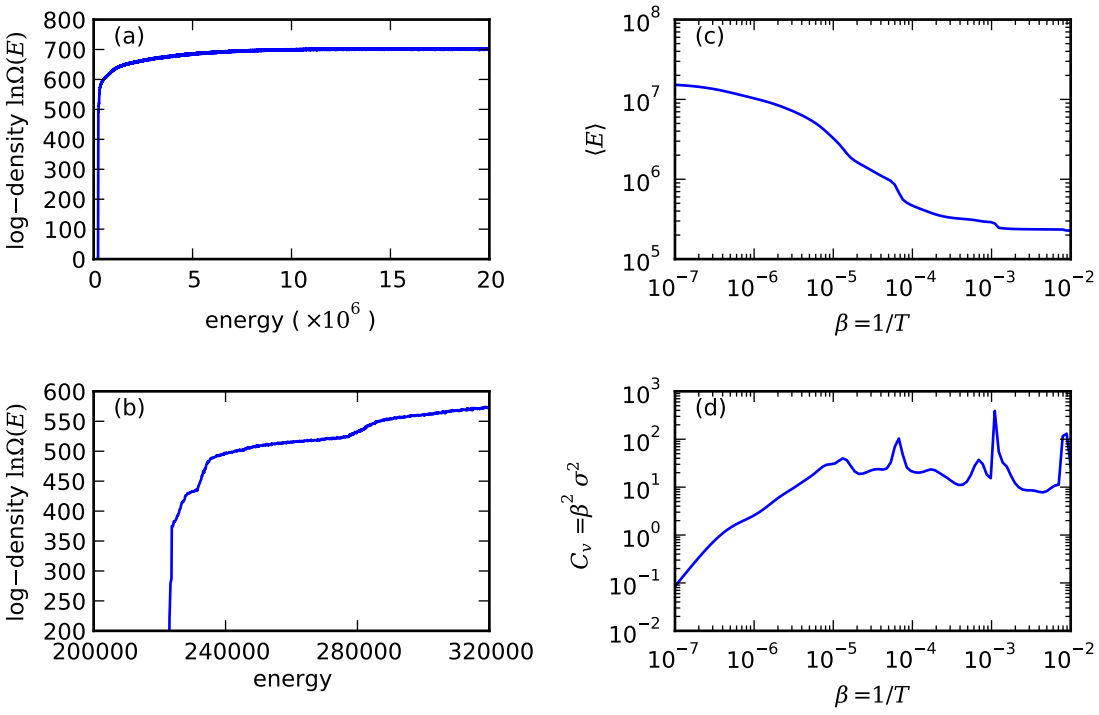


Figure 5.4: (a) Estimated entropy for the pattern formation model problem over 700 million iterations on 10 cores. (b) Zoom-in plot of the entropy at the low energy end. (c) Calculated mean energy over a range of temperature points. (d) Calculated heat capacity over the same temperature range.

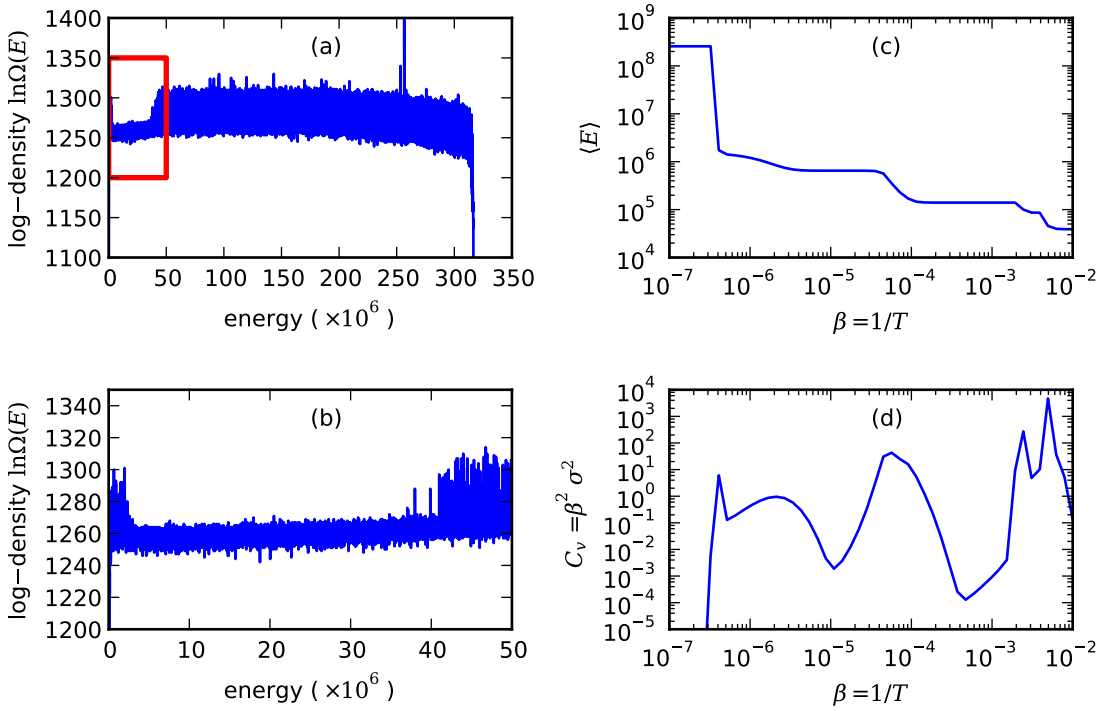


Figure 5.5: (a) Estimated entropy for the transcriptional model problem over 400 million iterations on 4 cores. (b) Zoom in plot of the red box in (a). (c) Calculated mean energy over a range of temperature points. (d) Calculated heat capacity over the same temperature range.

non-smooth density (Figure 5.4b). This non-smoothness is largely responsible for the sudden energy drop shown in Figure 5.4c and the spikes in the heat capacity in Figure 5.4d. In particular, the heat capacity spike near $\beta = 10^{-3}$ is large which suggest a phase transition around that temperature.

As a comparison, the estimated log-density for the transcriptional model (Figure 5.5a) is much more noisy and has an entirely different structure. Since this problem has a hard limit on each parameter instead of a penalty term, it has a fixed maximum energy and hence the hard drop at the high energy end. In addition, there is a spike near the higher end of the energy range corresponding to the score where there is 0 mRNA concentration due to the threshold as discussed in section 4.1.2. This energy apparently dominates the mean energy at high temperature (Figure 5.5c). At lower energy range, there is some intricate structure

(Figure 5.5b) that leads to the spike in heat capacity at lower temperature (Figure 5.5c).

The results in this section demonstrate that for two different real world optimization problems, the density of states estimation is able to reveal the unique features of the search space for each problem. In particular, it is able to identify regions of temperature where each problem has high or low heat capacity. As is the case with to the Lam-Delosme algorithm, this information can be used to determine the cooling schedule in the annealing process.

Because of the usefulness of the density of states information, it is desirable to have the estimation ahead of simulated annealing runs. It is possible to run the density of states estimation in place of the initial moves in traditional simulated annealing. Moreover, the algorithmic similarity between simulated annealing and the density of state estimation means the two algorithm can run side by side, and many of the existing simulated annealing infrastructure can be reused in the density of state estimation. In addition, the main purpose of the initial warm up is to decorrelate the state from the initial state. Since the density of states estimation tends to sample the states uniformly in energy, it is clear that the end result will decorrelate from the initial states.

Naturally, this method has its own drawbacks. The most prominent one is that a reasonable quality of the estimate requires hundreds of millions of iterations for all test problems. That is about one or two orders of magnitude larger than the annealing algorithm would take for each corresponding problem. Although all these estimates can be finished within reasonable time frame, the length required for the estimates seems to defeat the purpose that the density of states could provide guidance for the annealing algorithm. Fortunately, for the purpose of the cooling schedule, estimating the density of states before an annealing run does not need to be a complete Wang-Landau sampling to be useful. Instead, running the sampling for a small number of iterations can provide heat capacity at high temperature with reasonable accuracy. Since most optimization problems have small heat capacity at high temperature, this is the region where constant cooling speed in the exponential schedule is most wasteful. Improving the cooling speed at this region alone can provide

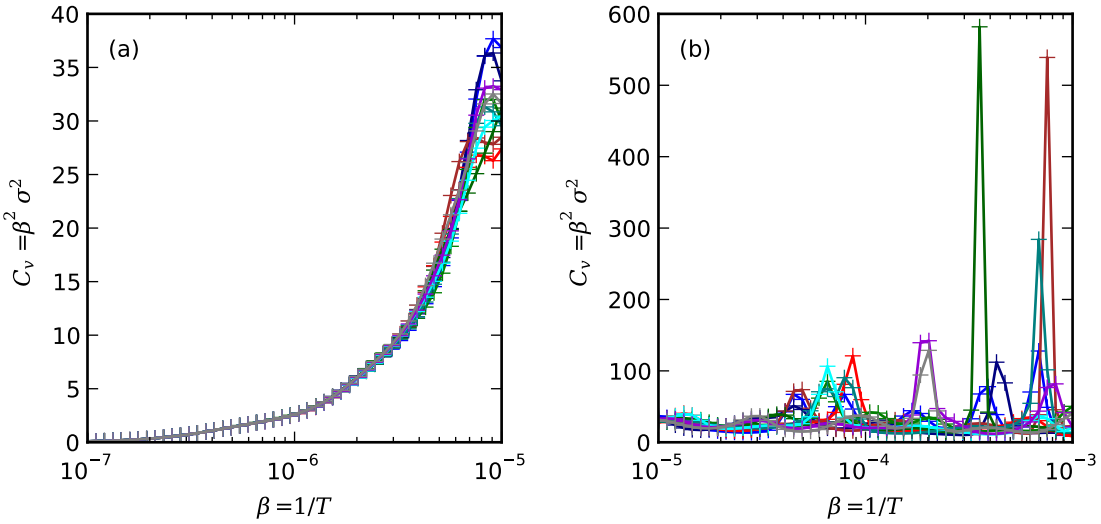


Figure 5.6: Calculated heat capacity curves of the pattern formation model problem from 10 independent density of states estimations. Each estimate, represented by one color, is generated over 700 million iterations. (a) At high temperature (low inverse temperature), and (b) At low temperature (high inverse temperature). Note the different y-scale on two subplots.

substantial benefit to the annealing process. In addition, most real applications will run multiple times, either under the same settings or a collection of closely related ones. Thus, the estimated density of states can be reused and improved upon incrementally over these runs and gradually produces more accurate estimates.

Another issue is that, even after the long estimation process, the accuracy of the estimated heat capacity at a lower temperature is not satisfactory. The heat capacity of the 5000-dimension Rastrigin function from the direct calculation differs from the theoretical value substantially in the low temperature region (Figure 5.3b). For problems where no theoretical values are available, multiple independent estimates must be generated so that consistency among estimates can be used for cross-validation. The example in the pattern formation model showed that the estimated heat capacity is reasonably consistent at high temperature (Figure 5.6a), while the low temperature values vary substantially (Figure 5.6b). Nevertheless, upon inspection, such inaccuracies can be worked around. The heat capacity of the 5000-dimension Rastrigin function can be approximated by the Gamma distribution

as discussed in section 5.3.1. For the pattern formation model, inspection of the location of peaks for each individual estimate allows one to conclude that the problem has two peaks in the heat capacity between inverse temperature 10^{-5} and 10^{-3} despite the variability for individual estimates.

The discussion in this chapter shows that the density of states estimation can provide valuable information about the search space structure of the problem being optimized. The heat capacity calculated from the density of states is able to provide insight into the desired cooling speed of the annealing process. Although the computational cost of the algorithm means an isolated annealing run may not benefit from the full potential of the estimates, for most practical applications the accumulated data will provide an estimate which will give an advantage over the exponential schedule. Future improvements should focus on the efficiency of the estimation, so that it can enable on-the-fly information of not only the cooling speed, but also other aspects of the search space structure.

CHAPTER 6

DISCUSSION

Over the previous chapters, this thesis has demonstrated a novel scalable parallel simulated annealing algorithm. By removing the adaptive cooling schedule and the associated variance estimation model used by the Chu's parallelization, this algorithm is able to show near perfect speedup for the Rastrigin function up to 192 cores and speedup over 40 for two production scale real world problems. Such performance has never been achieved before. The adaptive resampling interval method introduced in this thesis is able to perform as well or better than the best fixed resampling interval in all three test problems. This method hence removes the resampling interval as a tunable parameter. The density of states estimation introduced to the optimization problems is able to provide insights into the search space structure and infer an efficient cooling schedule without the burden of live variance estimation. In addition, the numerical experiments showed the importance the scaling of move control was in the performance of the parallel simulated annealing.

Despite the success reported in the previous chapters, the potential number of processor cores available for a modern application demands even more performance. This chapter investigates in which direction these potential for future improvements might lie.

6.1 Difficulties in move control

Among the components in a parallel simulated annealing algorithm, the move generation and control demonstrates an outsized influence on the performance. In the Rastrigin function, Figure 3.1 shows that a change in the move control gain K by a factor of 3 can affect the result by multiple orders of magnitudes. For the two real world test problems, the combination of K and the move control interval I_{mc} has a larger impact on the parallel performance than the resampling interval does (Figure 4.7 and 4.8).

These observations indicate that appropriate move generation and control is essential to

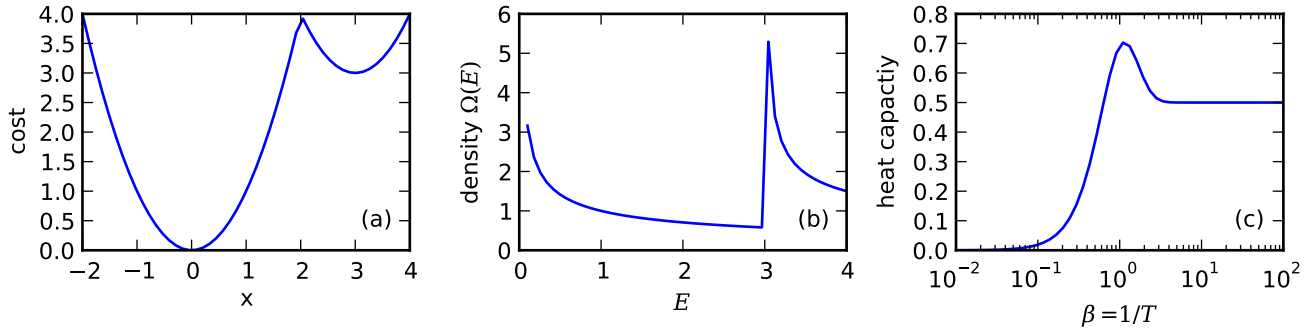


Figure 6.1: Demonstration of a simple optimization problem with cost function (6.1). (a) Cost function. (b) Density of states. (c) Heat capacity.

a successful simulated annealing algorithm. The move control algorithm prescribed in (2.19) performs reasonably under a moderate number of cores and corresponding cooling speed. For larger number of cores, the results in Chapter 4 suggest that this move control method has scaling problems when cooling speed become aggressive. The rest of this section offers some insights into this question by making a detailed analysis on a very simple optimization problem.

Consider an optimization problem with a 1-dimension piecewise smooth cost function given by

$$E(x) = \begin{cases} x^2, & x \leq 2 \\ (x-3)^2 + 3, & 2 < x \leq 4 \end{cases}. \tag{6.1}$$

It has a local minimum at $x = 3$ and a global minimum at $x = 0$ (Figure 6.1a). Its density of states can be obtained analytically as

$$\Omega(E) = \begin{cases} \dfrac{1}{\sqrt{E}}, & 0 \leq E < 3 \\ \dfrac{1}{\sqrt{E}} + \dfrac{1}{\sqrt{E-3}}, & 3 \leq E \leq 4 \end{cases}. \tag{6.2}$$

Due to its two quadratic minima, its density has two spikes that goes to infinity (6.1b). Its

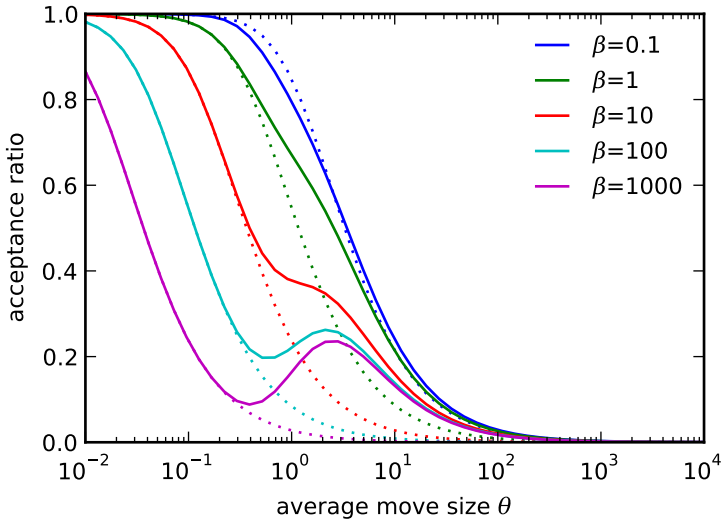


Figure 6.2: Relation between average move size and acceptance ratio for cost function (6.1) at different inverse temperature values. Solid lines are acceptance ratios calculated assuming the current state is at the local minimum where $x = 3$. Dotted lines of the same color represent the acceptance ratio of the same temperature starting from the global minimum $x = 0$.

partition function at inverse temperature β is

$$\begin{aligned} Z(\beta) &= \int_0^4 \Omega(E) e^{-\beta E} \, \mathrm{d}E \\ &= \sqrt{\frac{\pi}{\beta}} \left[\mathrm{erf}(2\sqrt{\beta}) + e^{-3\beta} \mathrm{erf}(\sqrt{\beta}) \right], \end{aligned} \tag{6.3}$$

where

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^t e^{-t^2} \, \mathrm{d}t \tag{6.4}$$

is the error function. Since the error function is non-elementary, the statistics of the problem need to be obtained numerically. In particular, its mean energy at infinite temperature $\langle E(\beta = 0) \rangle = 2$, and its heat capacity shows a distinct peak around $\beta = 1$ which suggests a phase transition at that temperature (Figure 6.1c).

Under this setting, it will be interesting to learn the relationship between the average move size θ and the resulting acceptance ratio under various temperatures in the scenario

when the current state is stuck at the local minimum at $x = 3$ versus when it is at the global minimum at $x = 0$. For states starting from these two minima, the corresponding acceptance ratios are straightforward to calculate. Assuming an out of bounds move will always be rejected, the acceptance ratio at inverse temperature β for the local one is

$$\begin{aligned}\rho_L(\theta;\beta) = 2\int_0^1 \frac{1}{2\theta}e^{-\frac{t}{\theta}-t^2\beta}\,\mathrm{d}t + \int_{3-\sqrt{3}}^{3+\sqrt{3}} \frac{1}{2\theta}e^{-\frac{t}{\theta}}\,\mathrm{d}t \\ + \left(\int_1^{3-\sqrt{3}} + \int_{3+\sqrt{3}}^5\right)\frac{1}{2\theta}e^{-\frac{t}{\theta}-[(3-t)^2-3]\beta}\,\mathrm{d}t,\end{aligned} \tag{6.5}$$

and the global one is

$$\rho_G(\theta;\beta) = 2\int_0^2 \frac{1}{2\theta}e^{-\frac{t}{\theta}-t^2\beta}\,\mathrm{d}t + \int_2^4 \frac{1}{2\theta}e^{-\frac{t}{\theta}-[(3-t)^2+3]\beta}\,\mathrm{d}t. \tag{6.6}$$

Numerical integration at 5 different β values from both sides of the phase transition shows distinct contrast between two cases (Figure 6.2): while ρ_G response curves are monotonic, reverse "S" shaped, and evenly spaced as β decreases, ρ_L ones have a distinct bump at around $\theta = 2.6$ for cases where $\beta > 1$. The location of the bump in ρ_L relates to the distance required for a state at the local minimum to move over the barrier towards the global minimum, and is independent of the temperature. The height of the bump is lower than 0.5 because lower energy states appear only one side of the current state. It is conceivable that if the cost function is instead constructed such that the metastable local minimum is between two identical deep minima, the height of the bump in acceptance ratio will go over 0.5.

As distinct as it is, this behavior in the responsive curve of ρ_L can only be revealed by investigating individual starting states. In traditional analysis where the complete information about the cost function is assumed, the response curve of ρ to θ is generally calculated by averaging over all possible states weighted in corresponding Boltzmann distribution. At high temperature, the response of ρ from global and local minima behaves very similarly.

At low temperature, the global minimum state will have an overwhelming weight relative to any local minima. Thus, the weighted average global response curve will behave like ρ_G at all temperature.

From the move generation point of view, the ideal strategy at low temperature will be using small θ for searching small improvements nearby, or "exploitation" as some literature calls it, when the state is near the global minimum while using large θ for searching new regions of interest, or "exploration", when the state is stuck at a local minimum. However, in practice, determining whether current state can lead to global minimum is nearly impossible. Going back to this example, the response curves ρ_L and ρ_G coincide a large part at low temperature. Most statistics obtained in the annealing algorithm, like variance and acceptance ratio, will be identical between two cases. Hence, paradoxically, without knowing the value and location of the global minimum, the very things the algorithm is used for, it is impossible to distinguish between exploitation versus exploration.

In theoretical analyses, such distinction is not necessary for the simulated annealing algorithm. When the system is kept at or near equilibrium, by definition the probability the system is in a region not lead to the global minimum is increasingly small. Based on this assumption, most of the existing move generation and control schemes, including the one this thesis uses, are derived from on the global response curve. In particular, the move control prescribed in (2.19) implies the acceptance ratio ρ has to decrease monotonically as θ increases.

In practice, as discussed in section 2.1.2, cooling is much faster and equilibrium is rarely reached. In this situation, move control based on the global response curve will keep working as long as the algorithm can reach the "correct" region by the phase transition temperature. If, however, the annealing run is stuck at the local minimum after the phase transition, instead of making large moves to get out of the local minimum, such a move control scheme will tend to make the move size smaller.

In the parallel setting, this problem is amplified by multiple factors. As the cooling speed

is faster in the parallel algorithm, the system is further away from equilibrium. That makes the probability of states being stuck in local minima larger. In addition, as discussed in section 4.3, the frequency of move control cannot keep up with the increased cooling speed. On the surface, this appears to make the effective move size larger for the temperature. However, it will also lead to lower acceptance ratio and cause the move control to over correct to a smaller θ. Furthermore, the parallel algorithm will explore the search space less efficiently than the serial counterpart. Although the parallel algorithm is supposed to have the same number of iterations over the same temperature span as the serial algorithm, the correlation will be higher due to replicated states during the resampling of states algorithm. The combination of above points is likely the cause for the limited performance of the parallel algorithm.

6.2 Scalability and the physical metaphor

Throughout the study of the parallel simulated annealing algorithm, the potential scalability of an ideal algorithm is always a fundamental question. In the cooling of a physical object, all parts of that object are being cooled simultaneously. Compared to the potential number of internal states in the process, physical cooling is a very fast process. This leads to the belief that the simulated annealing, as an imitation of the physical cooling, should be highly scalable as well. However, as both the results in the previous chapters and in the literature reviewed in chapter 2 show, such premise is not true. This section tries to identify a few of the potential sources of the discrepancies between the expectation and reality.

First of all, although the cooling in the colloquial sense may be considered fast, achieving the lowest energy configuration is not necessarily an easy process, especially when phase transition is involved. Take the common household activity of making ice cubes as an example. Simply putting water into the freezer usually will result in cloudy ice cubes. That is because the gasses dissolved in the water get trapped during the cooling and form tiny air pockets. To make a clear ice cube, one either has to cool the water very slowly or to use

water without dissolved gases. When considered from the optimization point of view, this observation means that to reach the minimum energy over a phase transition, the system has either to be cooled very slowly or started from a particular state, despite the parallelism in the process.

Secondly, how fast a physical object can cool without being stuck at some metastable state is shape dependent. Imagine two bodies of water of identical volume such that one is flat and thin while the other is a cube. It is easy to see that the flat one will be easier to freeze into ice. This echoes the observation from experience with simulated annealing that the maximum cooling speed is problem dependent. Since scaling in parallel is linked to the cooling speed, this also suggests the scalability is problem dependent.

Finally, the parallelism in the physical cooling is different from the parallelism we are seeking out of the optimization algorithms. The influence on the internal energy one atom has on another that is far away is negligible. Although the internal energy is not perfectly additive, the action of cooling on different parts of the object is roughly independent. Thus, cooling all the atoms at the same time is similar to parallelizing the calculation of the cost function. Such parallelism is not necessarily desirable. For many optimization problems, each individual step of cost function evaluation is much smaller than the whole optimization. In this situation, the parallelization suggested by the physical cooling is not an appropriate strategy for the optimization problem.

Overall, the parallelism in the physical cooling process is limited for many difficult situation, and faces similar challenges as the simulated annealing algorithm. Hence, the physical cooling process does not inspire any new direction for the simulated annealing beyond what is already embodied in the algorithm.

6.3 Future work

Facing the difficulties discussed in the previous section, this thesis speculates a few directions for potential improvement in and around the move generation and control in both serial and

parallel annealing.

The closeness of the current state to the equilibrium will be a useful measurement. Closer to the equilibrium will allow the move generation to pivot toward exploitation while being further away means more exploration will be necessary. In addition to the beneficial in the move control, such measurement can also be used in dynamically determining the cooling speed. A system closer to equilibrium can afford faster cooling, and vice versa. The implementation of such measurement will likely depend on a combination of current statistics and density of states estimation.

A better move control algorithm should be aware of the cooling speed. While the accumulation of the acceptance statistics takes time, the move control should be able to anticipate the cooling process and proactively adjust the move size between move controls. The implementation can extrapolate the optimal move size as a function of the temperature using past data. This method can make up for the less frequent move control caused by faster cooling in large parallel runs.

Additionally, in parallel algorithm, multiple states are maintained at the same temperature. A parallel move generation algorithm should be able to take advantage of this fact. In particular, the distances between states can reflect the density of the local minima and other properties of the search space. These statistics can be considered in move generation for a better exploration of the search space.

Each one of the above described direction should be able to solve a portion of the difficulties discussed in the previous section. As a whole, they are likely to improve the performance of parallel simulated annealing on top of what is possible today.

REFERENCES

[1] E. Aarts and J. Korst. *Simulated annealing and boltzmann machines: A stochastic approach to combinatorial optimization and neural computing.* John Wiley., 1990.

[2] E. H. L. Aarts and P. J. M. van Laarhoven. Statistical cooling algorithm: A general approach to combinatorial optimization problems. *Philips Journal of Research*, 40:193–226, 1985.

[3] J. T. Alander. On optimal population size of genetic algorithms. In *CompEuro '92 . Computer Systems and Software Engineering, Proceedings.*, pages 65–70, 1992.

[4] M. Ashyraliyev, K. Siggens, H. Janssens, J. Blom, M. Akam, and J. Jaeger. Gene circuit analysis of the terminal gap gene *huckebein*. *PLOS Computational Biology*, 5:e1000548, 2009.

[5] M. Emin Aydin and Vecihi Yiğit. Parallel simulated annealing. In *Parallel Metaheuristics: A New Class of Algorithms*, page 267. Wiley and Sons, New York, 2005.

[6] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1991.

[7] C. F. Batten. *Simplified vector-thread architectures for flexible and efficient data-parallel accelerators*. PhD thesis, Massachusetts Institute of Technology, 2010.

[8] D. C. Bauer and T. L. Bailey. Optimizing static thermodynamic models of transcriptional regulation. *Bioinformatics*, 25:1640–1646, 2009.

[9] Kolja Becker, Eva Balsa-Canto, Damjan Cicin-Sain, Astrid Hoermann, Hilde Janssens, Julio R. Banga, and Johannes Jaeger. Reverse-engineering post-transcriptional regulation of gap genes in Drosophila melanogaster. *PLoS Computational Biology*, 9:e1003281, 2013.

[10] B. A. Berg and T. Neuhaus. Multicanonical ensemble: A new approach to simulate first-order phase transition. *Physical Review Letters*, 68:9–12, 1992.

[11] A. Bevilacqua. A methodological approach to parallel simulated annealing on an smp system. *Journal of Parallel and Distributed Computing*, 62:1548–1570, 2002.

[12] Yang-Lang Chang, Kun-Shan Chen, Bormin Huang, Wen-Yen Chang, J.A. Benediktsson, and Lena Chang. A parallel simulated annealing approach to band selection for high-dimensional remote sensing images. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 4:579–590, 2011.

[13] C. K. Chen, R. P. Kühnlein, K. G. Eulenberg, S. Vincent, M. Affolter, and R. Schuh. The transcription factors Knirps and Knirps related control cell migration and branch morphogenesis during *Drosophila* tracheal development. *Development*, 125:4959–4968, 1998.

[14] Ding-Jun Chen, Chung-Yeol Lee, Cheol-Hoon Park, and Pedro Mendes. Parallelizing simulated annealing algorithms based on high-performance computer. *Journal of Global Optimization*, 39:261–289, 2007.

[15] H Chen, NS Flann, and DW Watson. Parallel genetic simulated annealing: A massively parallel simd algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9:126–136, 1998.

[16] K. W. Chu, Y. Deng, and J. Reinitz. Parallel simulated annealing by mixing of states. *The Journal of Computational Physics*, 148:646–662, 1999.

[17] Anton Crombach, Monica A. Garcia-Solache, and Johannes Jaeger. Evolution of early development in dipterans: Reverse-engineering the gap gene network in the moth midge *Clogmia albipunctata* (psychodidae). *BioSystems*, 123:74–85, 2014.

[18] Anton Crombach, Karl R. Wotton, Damjan Cicin-Sain, Maksat Ashyraliyev, and Johannes Jaeger. Efficient reverse-engineering of a developmental gene regulatory network. *PLoS Computational Biology*, 8:e1002589, 07 2012. doi:10.1371/journal.pcbi.1002589.

[19] F. Darema, S. Kirkpatrick, and V. A. Norton. Parallel algorithms for chip placement by simulated annealing. *IBM Journal of Research and Development*, 31:391–402, 1987. doi:10.1147/rd.313.0391.

[20] A. M. Ferreiro, J. A. García, J. G. López-Salas, and C. Vázquez. An efficient implementation of parallel simulated annealing algorithm in GPUs. *Journal of Global Optimization*, 57:863–890, 2013.

[21] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions On Pattern Analysis And Machine Intelligence*, 6:721–741, 1984.

[22] B. Gidas. Nonstationary Markov chains and convergence of the annealing algorithm. *Journal of Statistical Physics*, 39(1):73–131, 1985.

[23] D. R. Greening. Parallel simulated annealing techniques. *Physica D: Nonlinear Phenomena*, 42:293–306, 1990.

[24] Vitaly V Gursky, Lena Panok, Ekaterina M. Myasnikova, Manu, Maria G. Samsonova, John Reinitz, and Alexander M Samsonov. Mechanisms of gap gene expression canalization in the *Drosophila* blastoderm. *BMC Systems Biology*, 5:118, 2011. doi:10.1186/1752-0509-5-118 PMCID: PMC3398401.

[25] Bruce Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13:311–329, 1988.

[26] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[27] J. S. Higginson, R. R. Neptune, and F. C. Anderson. Simulated parallel annealing within a neighborhood for optimization of biomechanical systems. *Journal of Biomechanics*, 38:1938–1942, 2005.

[28] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[29] Lester Ingber. Adaptive simulated annealing (asa): Lessons learned. *Control and Cybernetics*, 25:33–54, 1996.

[30] J. Jaeger, M. Blagov, D. Kosman, K. N. Kozlov, Manu, E. Myasnikova, S. Surkova, C. E. Vanario-Alonso, M. Samsonova, D. H. Sharp, and J. Reinitz. Dynamical analysis of regulatory interactions in the gap gene system of *Drosophila melanogaster*. *Genetics*, 167:1721–1737, 2004.

[31] J. Jaeger, S. Surkova, M. Blagov, H. Janssens, D. Kosman, K. N. Kozlov, Manu, E. Myasnikova, C. E. Vanario-Alonso, M. Samsonova, D. H. Sharp, and J. Reinitz. Dynamic control of positional information in the early *Drosophila* embryo. *Nature*, 430:368–371, 2004.

[32] H. Janssens, S. Hou, J. Jaeger, A. R. Kim, E. Myasnikova, D. Sharp, and J. Reinitz. Quantitative and predictive model of transcriptional control of the *Drosophila melanogaster even skipped* gene. *Nature Genetics*, 38:1159–1165, 2006.

[33] L. Jostins and J. Jaeger. Reverse engineering a gene network using an asynchronous parallel evolution strategy. *BMC Systems Biology*, 4:17, 2010.

[34] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[35] A. R. Kim, C. Martinez, J. Ionides, A. F. Ramos, M. Z. Ludwig, N. Ogawa, D. H. Sharp, and J. Reinitz. Rearrangements of 2.5 kilobases of noncoding DNA from the *Drosophila even-skipped* locus define predictive rules of genomic *cis*-regulatory logic. *PLoS Genetics*, 9:e1003243, 2013. PMCID: PMC3585115.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[37] David J. Kropaczek. COPERNICUS: A multi-cycle optimization code for nuclear fuel based on parallel simulated annealing with mixing of states. *Progress in Nuclear Energy*, 53:554–561, 2011. Conference on Advances in Nuclear Fuel Management IV (ANFMIV), Hilton Head Isl, SC, APR 12-15, 2009.

[38] J. Lam and J.-M. Delosme. An efficient simulated annealing schedule: Derivation. Technical Report 8816, Yale Electrical Engineering Department, New Haven, CT, September 1988.

[39] J. Lam and J.-M. Delosme. An efficient simulated annealing schedule: Implementation and evaluation. Technical Report 8817, Yale Electrical Engineering Department, New Haven, CT, September 1988.

[40] Soo-Young Lee and Kyung Geun Lee. Synchronous and asynchronous parallel simulated annealing with multiple markov chains. *IEEE Transactions on Parallel and Distributed Systems*, 7:993–1008, 1996.

[41] N. Li, J. Cha, and Y. Lu. A parallel simulated annealing algorithm based on functional feature tree modeling for 3d engineering layout design. *Applied Soft Computing*, 10:592–601, 2010.

[42] S.W. Mahfoud and D.E. Goldberg. A genetic algorithm for parallel simulated annealing. *Parallel problem solving from nature*, 2:301–310, 1992.

[43] S.W. Mahfoud and D.E. Goldberg. Parallel recombinative simulated annealing: a genetic algorithm. *Parallel computing*, 21:1–28, 1995.

[44] Manu, S. Surkova, A. V. Spirov, V. Gursky, H. Janssens, A. Kim, O. Radulescu, C. E. Vanario-Alonso, D. H. Sharp, M. Samsonova, and J. Reinitz. Canalization of gene expression and domain shifts in the *Drosophila* blastoderm by dynamical attractors. *PLoS Computational Biology*, 5:e1000303, 2009. doi:10.1371/journal.pcbi.1000303 PMCID: PMC2646127.

[45] Manu, S. Surkova, A. V. Spirov, V. Gursky, H. Janssens, A. Kim, O. Radulescu, C. E. Vanario-Alonso, D. H. Sharp, M. Samsonova, and J. Reinitz. Canalization of gene expression in the *Drosophila* blastoderm by gap gene cross regulation. *PLoS Biology*, 7:e1000049, 2009. doi:10.371/journal.pbio.1000049 PMCID: PMC2653557.

[46] Carlos A. Martinez, Kenneth A. Barr, Ah-Ram Kim, and John Reinitz. A synthetic biology approach to the development of transcriptional regulatory models and custom enhancer design. *Methods*, 62:91–98, 2013. PMCID: PMC3924567.

[47] N. Metropolis, A. Rosenbluth, M. N. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087–1092, 1953.

[48] H Mühlenbein, M Schomisch, and J Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17:619–632, 1991.

[49] Alexander G. Nikolaev and Sheldon H. Jacobson. Simulated annealing. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 1–39. Springer US, 2010. 10.1007/978-1-4419-1665-5_1.

[50] E. Onbaşoğlu and L. Özdamar. Parallel simulated annealing algorithms in global optimization. *Journal of Global Optimization*, 19:27–50, 2001.

[51] Keith E. Ottinger and G. Ivan Maldonado. BWROPT: A multi-cycle BWR fuel cycle optimization code. *Nuclear Engineering and Design*, 291:236–243, 2015.

[52] Ricardo Pariona-Llanos, Raphael Souza Pavani, Marcelo Reis, Vincent Noel, Ariel Mariano Silber, Hugo Aguirre Armelin, Maria Isabel Nogueira Cano, and Maria Carolina Elias. Glyceraldehyde 3-Phosphate Dehydrogenase-Telomere association correlates with redox status in Trypanosoma cruzi. *PLoS ONE*, 10:e0120896, 2015.

[53] D. J. Ram, T. H. Sreenivas, and K. G. Subramaniam. Parallel simulated annealing algorithms. *Journal of Parallel and distributed Computing*, 37:207–212, 1996.

[54] J. Reinitz, S. Hou, and D. H. Sharp. Transcriptional control in *Drosophila*. *ComPlexUs*, 1:54–64, 2003.

[55] J. Reinitz, D. Kosman, C. E. Vanario-Alonso, and D. H. Sharp. Stripe forming architecture of the gap gene system. *Developmental Genetics*, 23:11–27, 1998.

[56] J. Reinitz, E. Mjolsness, and D. H. Sharp. Cooperative control of positional information in *Drosophila* by *bicoid* and maternal *hunchback*. *The Journal of Experimental Zoology*, 271:47–56, 1995.

[57] J. Reinitz and D. H. Sharp. Mechanism of *eve* stripe formation. *Mechanisms of Development*, 49:133–158, 1995.

[58] G. Rudolph. Massively parallel simulated annealing and its relation to evolutionary algorithms. *Evolutionary Computation*, 1:361–383, 1993.

[59] D. H. Sharp and J. Reinitz. Prediction of mutant expression patterns using gene circuits. *Biosystems*, 47:79–90, 1998.

[60] Sadanand Singh, Manan Chopra, and Juan J. de Pablo. Density of statesbased molecular simulations. *Annual Review of Chemical and Biomolecular Engineering*, 3:369–394, 2012.

[61] A Sohn. Generalized speculative computation of parallel simulated annealing. *Annals of Operations Research*, 63:29–55, 1996.

[62] Y. Sun, G. Zheng, C. Mei, E. J. Bohm, J. C. Phillips, L. V. Kale, and T. R. Jones. Optimizing fine-grained communication in a biomolecular simulation application on cray xk6. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 1–11, 2012.

[63] H. Szu and R. Hartley. Fast simulated annealing. *Physics Letters A*, 122:157–162, 1987.

[64] D.R. Thompson and G.L. Bilbro. Sample-sort simulated annealing. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 35:625–632, 2005.

[65] TOP500.ORG. TOP500 Sublist Generator. http://top500.org/statistics/sublist/.

[66] F. Wang and D. P. Landau. Determining the density of states for classical statistical models: A random walk algorithm to produce a flat histogram. *Physical Review E*, 64:056101, 2001.

[67] F Wang and D. P. Landau. Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical Review Letters*, 86:2050–2053, 2001.

[68] T. Wang and G. D. Stormo. Identifying the conserved network of *cis*-regulatory sites of a eukaryotic genome. *Proceedings of the National Academy of Sciences USA*, 102:17400–17405, 2005.

[69] ZG Wang, YS Wong, and M Rahman. Development of a parallel optimization method based on genetic simulated annealing algorithm. *Parallel Computing*, 31:839–857, 2005.

[70] Jonathan K. Whitmer, Chi-cheng Chiu, Abhijeet A. Joshi, and Juan J. de Pablo. Basis function sampling: A new paradigm for material property computation. *Physical Review Letters*, 113:190602, 2014.

[71] Jonathan K. Whitmer, Aaron M. Fluitt, Lucas Antony, Jian Qin, Michael McGovern, and Juan J. de Pablo. Sculpting bespoke mountains: Determining free energies with basis expansions. *The Journal of Chemical Physics*, 143:044101, 2015.

[72] E.E. Witte, R.D. Chamberlain, and M.A. Franklin. Parallel simulated annealing using speculative computation. *Parallel and Distributed Systems, IEEE Transactions on*, 2:483–494, 1991.

[73] KL Wong and AG Constantinides. Speculative parallel simulated annealing with acceptance prediction. In *Computers and Digital Techniques, IEE Proceedings-*, volume 143, pages 219–223. IET, 1996.

[74] Samuel Xavier-de-Souza, Johan A. K. Suykens, Joos Vandewalle, and Desire Bolle. Coupled simulated annealing. *IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics*, 40:320–335, 2010.

[75] Qiliang Yan and Juan J. de Pablo. Fast calculation of the density of states of a fluid by monte carlo simulations. *Physical Review Letters*, 90:035701, 2003.

[76] Qiliang Yan, Roland Faller, and Juan J. de Pablo. Density-of-states monte carlo method for simulation of fluids. *The Journal of Chemical Physics*, 116:8745–8749, 2002.

[77] L. Yong, K. Lishan, and D.J. Evans. The annealing evolution algorithm as function optimizer. *Parallel Computing*, 21:389–400, 1995.

[78] C. Zhou, T. C. Schulthess, S. Torbrügge, and D. P. Landau. Wang-landau algorithm for continuous models and joint density of states. *Physical Review Letters*, 96:120201, 2006.