HHS Public Access

Author manuscript

Parallel Comput. Author manuscript; available in PMC 2017 April 01.

Published in final edited form as:

Parallel Comput. 2016 April 1; 53: 23–31. doi:10.1016/j.parco.2016.02.001.

Parallel Simulated Annealing Using an Adaptive Resampling Interval

Zhihao Lou[a] and **John Reinitz**[b]

Zhihao Lou: zhlou@uchicago.edu; John Reinitz: reinitz@galton.uchicago.edu

[a]Department of Computer Science, the University of Chicago, Chicago, Illinois, USA

[b]Department of Statistics, Department of Ecology and Evolution, Department of Molecular Genetics and Cell Biology, Institute of Genomics and Systems Biology, the University of Chicago, Chicago, Illinois, USA

Abstract

This paper presents a parallel simulated annealing algorithm that is able to achieve 90% parallel efficiency in iteration on up to 192 processors and up to 40% parallel efficiency in time when applied to a 5000-dimension Rastrigin function. Our algorithm breaks scalability barriers in the method of Chu *et al.* (1999) by abandoning adaptive cooling based on variance. The resulting gains in parallel efficiency are much larger than the loss of serial efficiency from lack of adaptive cooling. Our algorithm resamples the states across processors periodically. The resampling interval is tuned according to the success rate for each specific number of processors. We further present an adaptive method to determine the resampling interval based on the adoption rate. This adaptive method is able to achieve nearly identical parallel efficiency but higher success rates compared to the fixed interval one using the best interval found.

Keywords

parallel algorithm; global optimization; Rastrigin function; evolutionary computation; stochastic optimization; MPI

1. Introduction

Simulated annealing, introduced by Kirkpatrick *et al.* [1], 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 [2, 3, 4, 5, 6]. 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

Correspondence to: Zhihao Lou, zhlou@uchicago.edu.

Disclosure Statement

The authors are not aware of any affiliations or funding that might be perceived as affecting the objectivity of this paper.

artificial temperature T using the Metropolis algorithm [7]. At each iteration, the algorithm proposes a perturbed state (a "move") with energy E_{new} from the current state with energy E_{old}. If $E_{new} < E_{old}$, the new state is accepted. Otherwise the new state is accepted with probability $\exp[-(E_{new} - E_{old})/T]$. 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 [8, 9], and also for a schedule where $T_k \propto 1/k$ and moves are drawn from a Cauchy distribution [10]. 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 [11]. Adaptive control of move generation together with cooling rate was introduced by Lam and Delosme [12, 13]. 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 [2, 14, 15, 16].

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 [17]. 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 [18, 19, 20]. 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 [21] and variants [22, 23, 24, 25], as well as resampling of states [26, 27]. There are also algorithms which change the acceptance criterion in the Metropolis algorithm to better suit the parallelization [28, 29]. Since Rudolph [30] 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 [24, 31, 32], or by blending in the concepts of evolutionary algorithms with the move generation strategy [30, 33, 34, 35]. Unfortunately, most published works did not report speedup at all [28, 29, 30, 31, 34, 35], or only reported speedup for up to 16 processors, a small number by today's standards [18, 22, 24, 25, 27]. Among those that reported speedup for 32 processors or more, some show mediocre speedup [19, 20] while others give a suspiciously super-linear speedup [31, 33]. Super-linear speedup suggests either inefficiency in the serial performance or a decrease in the quality of parallel results [36].

The most promising recent development in parallel simulated annealing is reported in recent work on GPUs [37]. 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.* [26] demonstrated practical utility in solving certain estimation problems that arise in systems biology and operations research on the nuclear fuel cycle [3, 4, 5, 6, 38, 39, 40, 41, 42, 43, 44, 45, 46]. This method is based on the serial Lam-Delosme algorithm [12, 13], 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.

2. Methods

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

2.1. The Rastrigin Function

The n-dimensional Rastrigin function [47]

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

is a family of multi-modal analytical test functions which is used in many case studies of simulated annealing algorithms [30, 31, 32, 35]. 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 [3, 6] 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.

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 [12, 13] 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 [1], which specifies

$$T_{k+1}=(1-\lambda)T_k. \quad (2)$$

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 [13, 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 [12, 13], and also in the parallelization by Chu *et al.* [26]. 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 [2]. 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 $\rho = 0.44$ [12]. This calculation has been confirmed numerically for at least 4 test problems [13, 48], 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

$$\Delta x \sim \text{Laplace}(0, \theta), \quad (3)$$

where the average move size θ is adaptively adjusted by feedback control prescribed by Lam and Delosme [12, 13] such that

$$\ln\, \theta_{\text{new}} = K(\rho - 0.44) + \ln\, \theta_{\text{old}}, \quad (4)$$

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* [26]. At each resampling, each processor receives the energy of every other processor. Then each processor p chooses a target i independently with probability

$$\frac{e^{-E_i/T}}{\sum_j e^{-E_j/T}}. \quad (5)$$

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 1 were conducted on Beagle2, a Cray XE-6 system at the Computation Institute of the University of Chicago. The experiments shown in Figure 1 were conducted on Beast, a cluster managed by the Ecology and Evolution department of the University of Chicago.

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 parallelization.

In a maximally efficient serial annealing algorithm, the average quality of the answer depends on a single variable, the average number of iterations [13, 26]. Because the average number of iterations is not directly adjustable, it is necessary to obtain the desired relationship 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 [26] and so it is useful to set λ to the highest value that gives a reasonably good answer by suitable choice of *K* (Figure 1) and then obtain the serial performance curve of answer quality versus number of iterations *N* by varying κ (Figure 2). Inspection of Figure 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 constructing the serial performance curve.

To calculate the speedup based on comparable serial results, we follow the method introduced by Chu *et al.* [26] 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 obtain the serial performance curve Figure 2a as

$$N_s(E_f) = (9.56 \times 10^6) - (5.55 \times 10^5) \ln E_f. \quad (6)$$

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

$$S_N = \frac{N_s(E_P)}{N_P}, \quad (7)$$

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 2b as

$$T_s(E_f) = 956 - 23.5 \ln E_f \quad (8)$$

and we define the adjusted speedup in time S_T as

$$S_T = \frac{T_s(E_P)}{T_P}. \quad (9)$$

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.

3.1. 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 [26]. 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 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 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 (7) and (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 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 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 on 48 processors and then decreases. This happens because, as P increases, the algorithm demands more frequent communication as Figure 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.

3.2. 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 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 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

$$\ln R_{n+1} = \ln R_n + C(A_R - A_0). \quad (10)$$

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 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 5a). Although the speedup in time lags the corresponding best R scheme for 96 processors and under (Figure 5b), it shows significant improvement in terms of success rate (Figure 5c).

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 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 [26, 49], 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 because 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.

Acknowledgments

This research was supported in part by NIH under grant R01 OD010936-23A1 (formerly R01 RR07801) and also by NIH through resources provided by the Computation Institute and the Biological Sciences Division of the University of Chicago and Argonne National Laboratory, under grant 1S10OD018495-01.

References

1. Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science. 1983; 220:671–680. [PubMed: 17813860]
2. Reinitz J, Sharp DH. Mechanism of *eve* stripe formation. Mechanisms of Development. 1995; 49:133–158. [PubMed: 7748785]
3. Jaeger J, Surkova S, Blagov M, Janssens H, Kosman D, Kozlov KN, Manu, Myasnikova E, Vanario-Alonso CE, Samsonova M, Sharp DH, Reinitz J. Dynamic control of positional information in the early *Drosophila* embryo. Nature. 2004; 430:368–371. [PubMed: 15254541]
4. Manu, Surkova S, Spirov AV, Gursky V, Janssens H, Kim A, Radulescu O, Vanario-Alonso CE, Sharp DH, Samsonova M, Reinitz J. Canalization of gene expression in the *Drosophila* blastoderm by gap gene cross regulation. PLoS Biology. 2009; 7:e1000049. PMCID: PMC2653557. [PubMed: 19750121]
5. Crombach A, Wotton KR, Cicin-Sain D, Ashyraliyev M, Jaeger J. Efficient reverse-engineering of a developmental gene regulatory network. PLoS Computational Biology. 2012; 8:e1002589. [PubMed: 22807664]

6. Kim AR, Martinez C, Ionides J, Ramos AF, Ludwig MZ, Ogawa N, Sharp DH, Reinitz J. Rearrangements of 2.5 kilobases of noncoding DNA from the *Drosophila even-skipped* locus define predictive rules of genomic *cis*-regulatory logic. PLoS Genetics. 2013; 9:e1003243. PMCID: PMC3585115. [PubMed: 23468638]
7. Metropolis N, Rosenbluth A, Rosenbluth MN, Teller A, Teller E. Equation of state calculations by fast computing machines. The Journal of Chemical Physics. 1953; 21:1087–1092.
8. Geman S, Geman D. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. IEEE Transactions On Pattern Analysis And Machine Intelligence. 1984; 6:721–741. [PubMed: 22499653]
9. Hajek B. Cooling schedules for optimal annealing. Mathematics of Operations Research. 1988; 13:311–329.
10. Szu H, Hartley R. Fast simulated annealing. Physics Letters A. 1987; 122:157–162.
11. Aarts EHL, van Laarhoven PJM. Statistical cooling algorithm: A general approach to combinatorial optimization problems. Philips Journal of Research. 1985; 40:193–226.
12. Lam, J.; Delosme, J-M. An efficient simulated annealing schedule: Derivation, Tech. Rep. 8816. New Haven, CT: Yale Electrical Engineering Department; 1988 Sep..
13. Lam, J.; Delosme, J-M. An efficient simulated annealing schedule: Implementation and evaluation, Tech. Rep. 8817. New Haven, CT: Yale Electrical Engineering Department; 1988 Sep.
14. Reinitz J, Mjolsness E, Sharp DH. Cooperative control of positional information in *Drosophila* by *bicoid* and maternal *hunchback*. The Journal of Experimental Zoology. 1995; 271:47–56. [PubMed: 7852948]
15. Reinitz J, Kosman D, Vanario-Alonso CE, Sharp DH. Stripe forming architecture of the gap gene system. Developmental Genetics. 1998; 23:11–27. [PubMed: 9706690]
16. Sharp DH, Reinitz J. Prediction of mutant expression patterns using gene circuits. Biosystems. 1998; 47:79–90. [PubMed: 9715752]
17. Aydin, ME.; Yigĩt, V. Parallel Metaheuristics: A New Class of Algorithms. New York: Wiley and Sons; 2005. Parallel simulated annealing; p. 267
18. Witte E, Chamberlain R, Franklin M. Parallel simulated annealing using speculative computation. Parallel and Distributed Systems, IEEE Transactions on. 1991; 2:483–494.
19. Wong, K.; Constantinides, A. Computers and Digital Techniques, IEE Proceedings. Vol. 143. IET; 1996. Speculative parallel simulated annealing with acceptance prediction; p. 219-223.
20. Sohn A. Generalized speculative computation of parallel simulated annealing. Annals of Operations Research. 1996; 63:29–55.
21. Aarts, E.; Korst, J. Simulated annealing and boltzmann machines: A stochastic approach to combinatorial optimization and neural computing. John Wiley: 1990.
22. Lee S-Y, Lee KG. Synchronous and asynchronous parallel simulated annealing with multiple markov chains. IEEE Transactions on Parallel and Distributed Systems. 1996; 7:993–1008.
23. Higginson JS, Neptune RR, Anderson FC. Simulated parallel annealing within a neighborhood for optimization of biomechanical systems. Journal of Biomechanics. 2005; 38:1938–1942. [PubMed: 16023483]
24. Ram DJ, Sreenivas TH, Subramaniam KG. Parallel simulated annealing algorithms. Journal of Parallel and distributed Computing. 1996; 37:207–212.
25. Li N, Cha J, Lu Y. A parallel simulated annealing algorithm based on functional feature tree modeling for 3d engineering layout design. Applied Soft Computing. 2010; 10:592–601.
26. Chu KW, Deng Y, Reinitz J. Parallel simulated annealing by mixing of states. The Journal of Computational Physics. 1999; 148:646–662.
27. Chang Y-L, Chen K-S, Huang B, Chang W-Y, Benediktsson J, Chang L. 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. 2011; 4:579–590.
28. Thompson D, Bilbro G. Sample-sort simulated annealing, Systems, Man, and Cybernetics. Part B: Cybernetics, IEEE Transactions on. 2005; 35:625–632.
29. Xavier-de-Souza S, Suykens JAK, Vandewalle J, Bolle D. Coupled simulated annealing. IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics. 2010; 40:320–335.

30. Rudolph G. Massively parallel simulated annealing and its relation to evolutionary algorithms. Evolutionary Computation. 1993; 1:361–383.
31. Yong L, Lishan K, Evans D. The annealing evolution algorithm as function optimizer. Parallel Computing. 1995; 21:389–400.
32. Chen D-J, Lee C-Y, Park C-H, Mendes P. Parallelizing simulated annealing algorithms based on high-performance computer. Journal of Global Optimization. 2007; 39:261–289.
33. Mahfoud S, Goldberg D. Parallel recombinative simulated annealing: a genetic algorithm. Parallel computing. 1995; 21:1–28.
34. Chen H, Flann N, Watson D. Parallel genetic simulated annealing: A massively parallel simd algorithm. IEEE Transactions on Parallel and Distributed Systems. 1998; 9:126–136.
35. Wang Z, Wong Y, Rahman M. Development of a parallel optimization method based on genetic simulated annealing algorithm. Parallel Computing. 2005; 31:839–857.
36. Greening DR. Parallel simulated annealing techniques. Physica D: Non-linear Phenomena. 1990; 42:293–306.
37. Ferreiro AM, García JA, López-Salas JG, Vázquez C. An efficient implementation of parallel simulated annealing algorithm in GPUs. Journal of Global Optimization. 2013; 57:863–890.
38. Jaeger J, Blagov M, Kosman D, Kozlov KN, Manu, Myasnikova E, Surkova S, Vanario-Alonso CE, Samsonova M, Sharp DH, Reinitz J. Dynamical analysis of regulatory interactions in the gap gene system of *Drosophila melanogaster*. Genetics. 2004; 167:1721–1737. [PubMed: 15342511]
39. Bauer DC, Bailey TL. Optimizing static thermodynamic models of transcriptional regulation. Bioinformatics. 2009; 25:1640–1646. [PubMed: 19398449]
40. Ashyraliyev M, Siggens K, Janssens H, Blom J, Akam M, Jaeger J. Gene circuit analysis of the terminal gap gene *huckebein*. PLOS Computational Biology. 2009; 5:e1000548. [PubMed: 19876378]
41. Gursky VV, Panok L, Myasnikova EM, Manu, Samsonova MG, Reinitz J, Samsonov AM. Mechanisms of gap gene expression canalization in the *Drosophila* blastoderm. BMC Systems Biology. 2011; 5:118. PMCID: PMC3398401. [PubMed: 21794172]
42. Kropaczek, DJ. COPERNICUS: A multi-cycle optimization code for nuclear fuel based on parallel simulated annealing with mixing of states. Progress in Nuclear Energy; conference on Advances in Nuclear Fuel Management IV (ANFMIV); 12–15, 2009; Hilton Head Isl, SC, APR. 2011. p. 554-561.
43. Becker K, Balsa-Canto E, Cicin-Sain D, Hoermann A, Janssens H, Banga JR, Jaeger J. Reverse-engineering post-transcriptional regulation of gap genes in Drosophila melanogaster. PLoS Computational Biology. 2013; 9:e1003281. [PubMed: 24204230]
44. Crombach A, Garcia-Solache MA, Jaeger J. Evolution of early development in dipterans: Reverse-engineering the gap gene network in the moth midge *Clogmia albipunctata* (psychodidae). BioSystems. 2014; 123:74–85. [PubMed: 24911671]
45. Pariona-Llanos R, Pavani RS, Reis M, Noel V, Silber AM, Armelin HA, Cano MIN, Elias MC. Glyceraldehyde 3-Phosphate Dehydrogenase-Telomere association correlates with redox status in Trypanosoma cruzi. PLoS ONE. 2015; 10:e0120896. [PubMed: 25775131]
46. Ottinger KE, Maldonado GI. BWROPT: A multi-cycle BWR fuel cycle optimization code. Nuclear Engineering and Design. 2015; 291:236–243.
47. Mühlenbein H, Schomisch M, Born J. The parallel genetic algorithm as function optimizer. Parallel Computing. 1991; 17:619–632.
48. Martinez CA, Barr KA, Kim A-R, Reinitz J. A synthetic biology approach to the development of transcriptional regulatory models and custom enhancer design. Methods. 2013; 62:91–98. PMCID: PMC3924567. doi: [PubMed: 23732772]
49. Jostins L, Jaeger J. Reverse engineering a gene network using an asynchronous parallel evolution strategy. BMC Systems Biology. 2010; 4:17. [PubMed: 20196855]

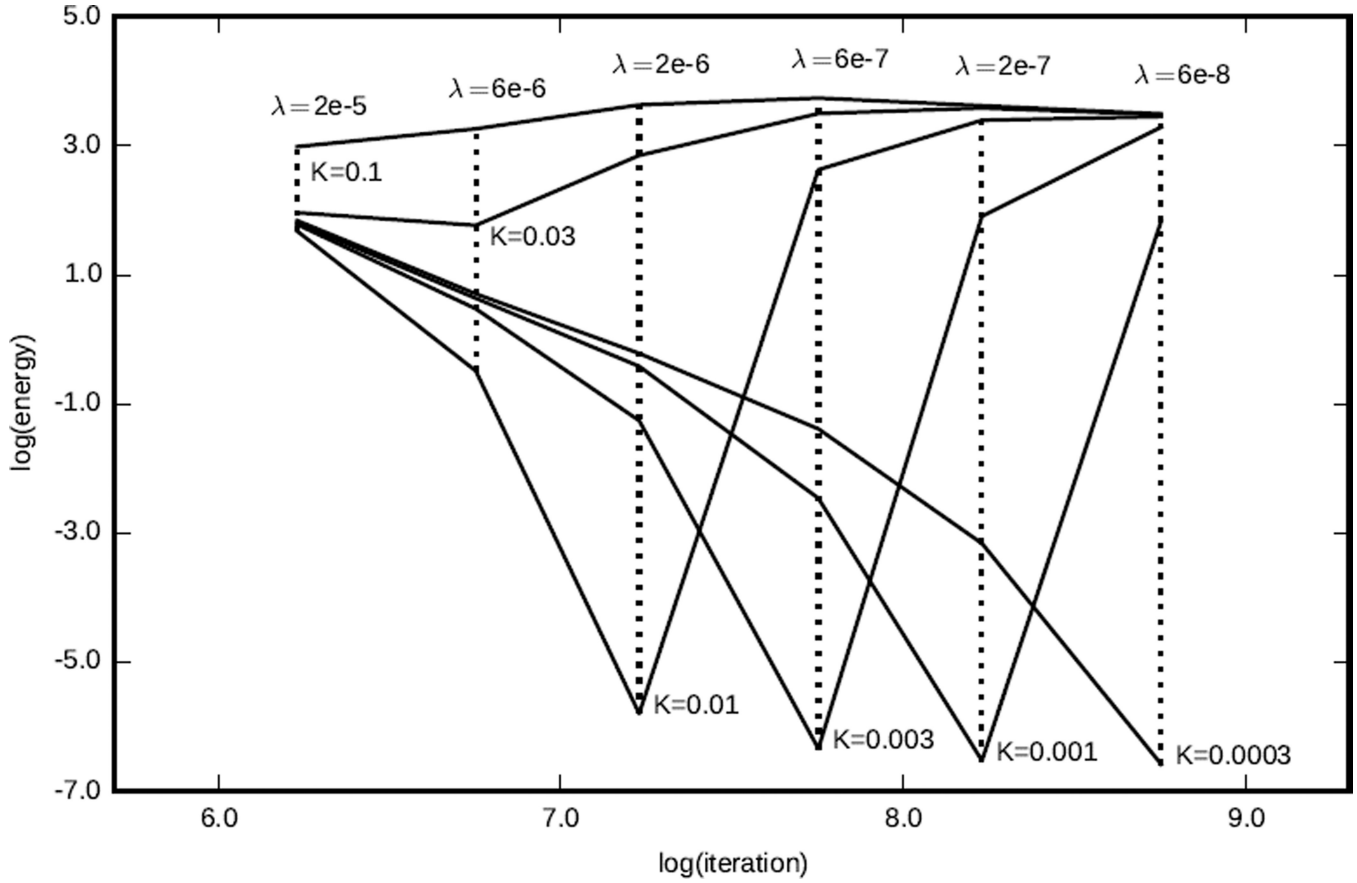

Figure 1.

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.

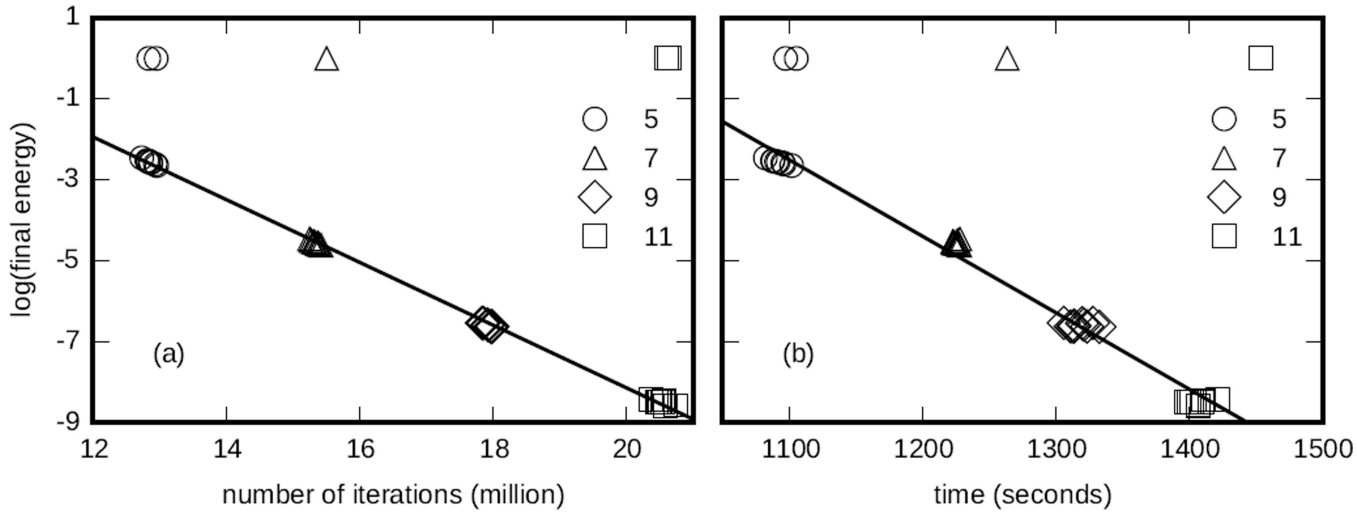

Figure 2.

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.

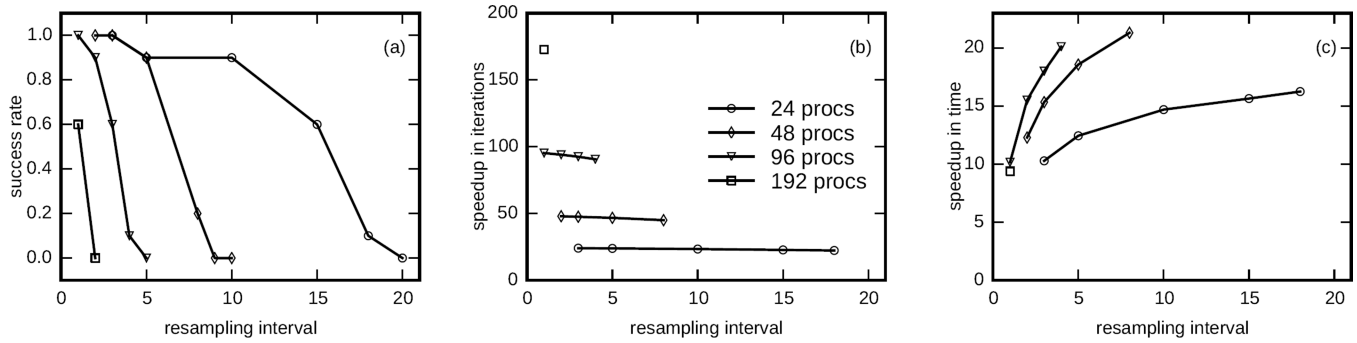

Figure 3.

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.

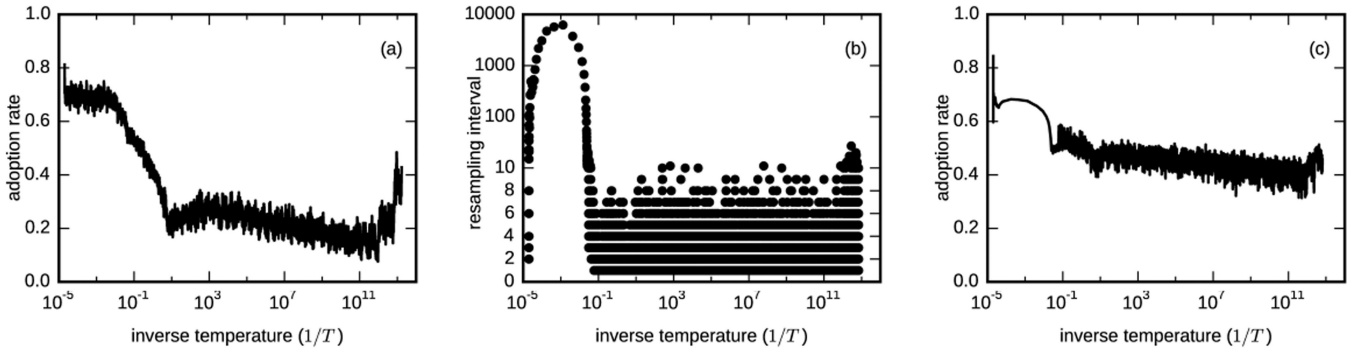

Figure 4.

(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.

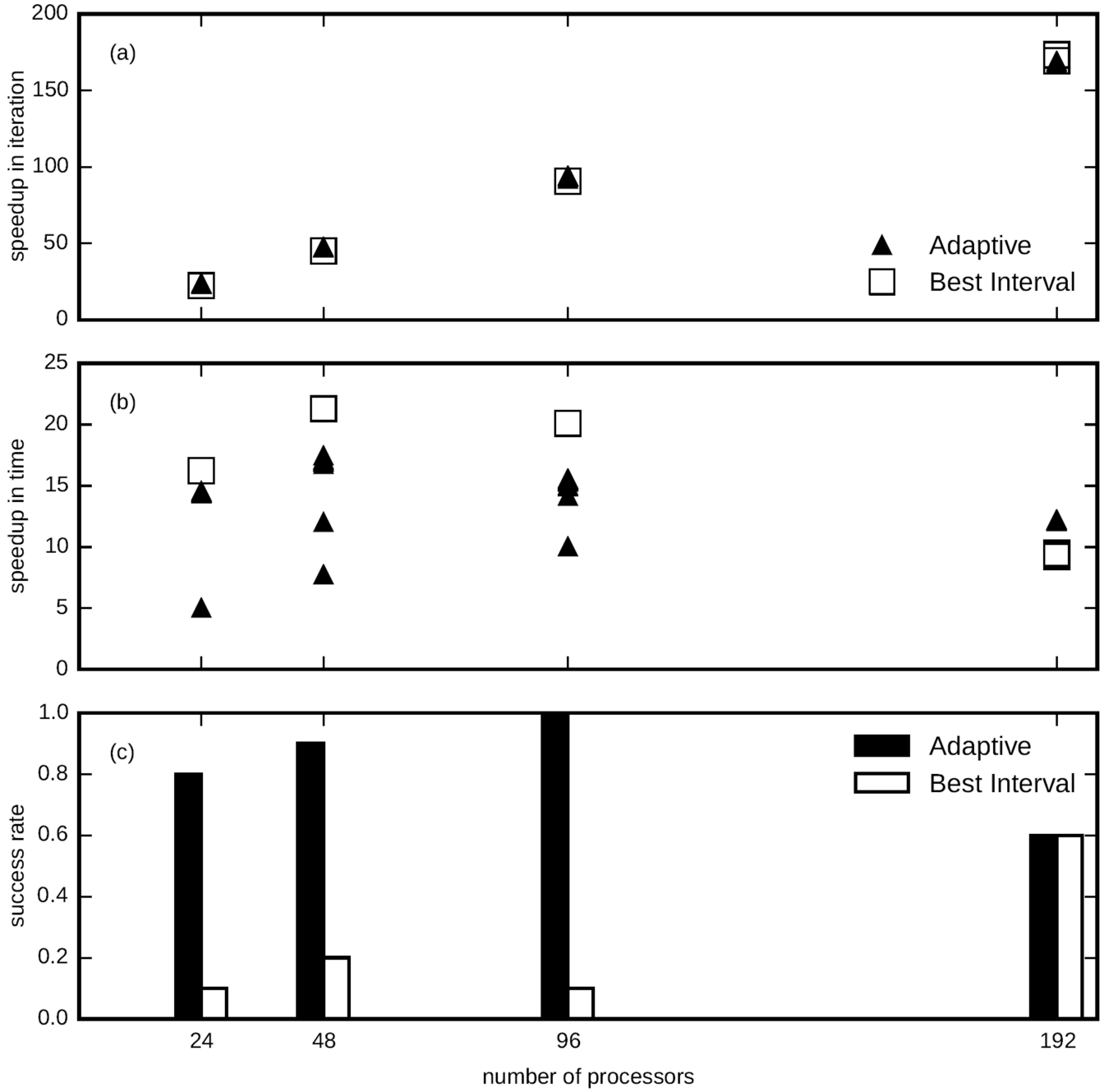

Figure 5.

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.

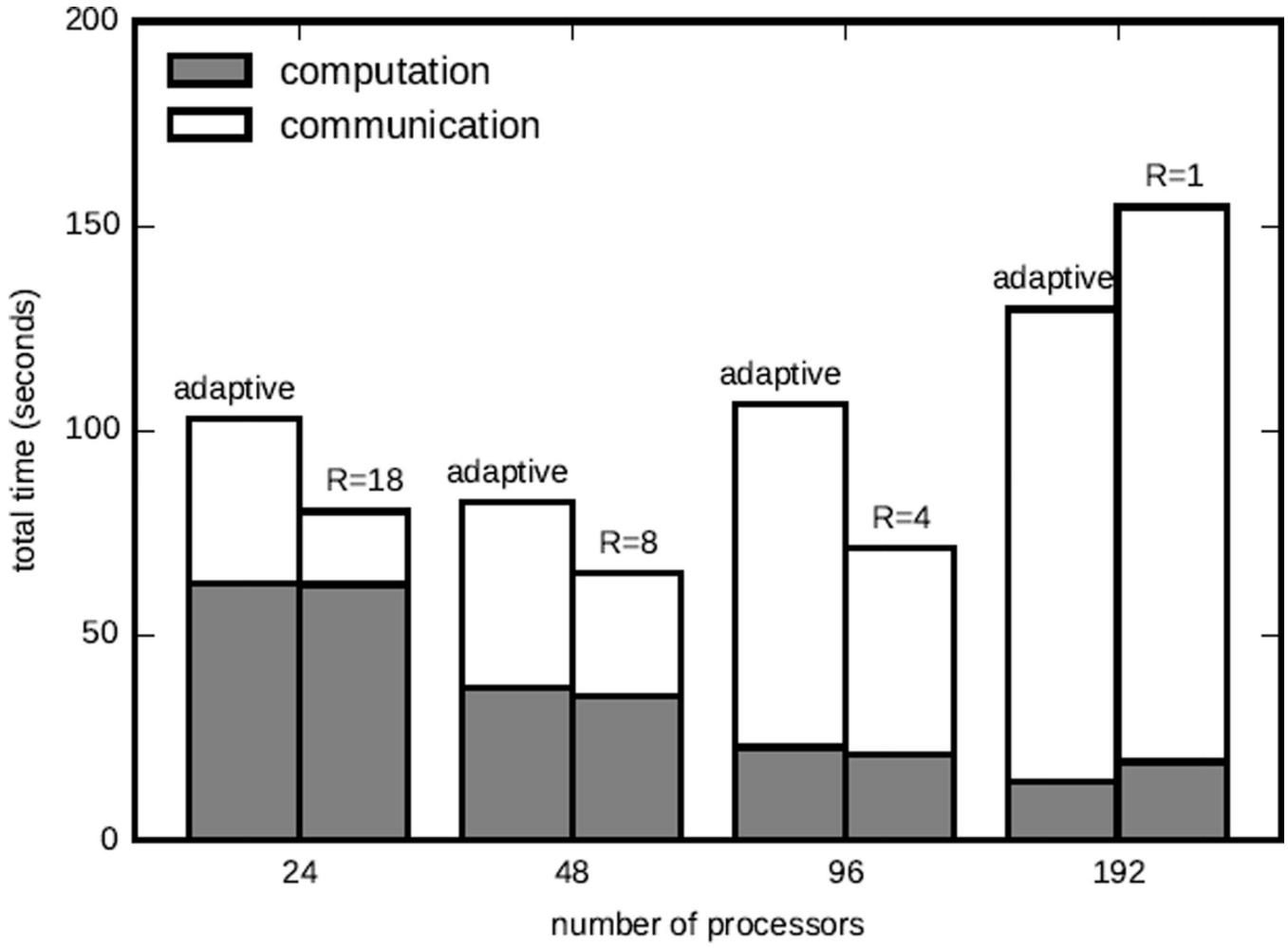

Figure 6.

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.