TR-673

Time-homogeneous Parallel Annealing Algorithm

by
K. Kimura & K. Taki

August, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Time-homogeneous Parallel Annealing Algorithm

KOUICHI KIMURA AND KAZUO TAKI

Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan

Abstract

We propose a new parallel simulated annealing algorithm. Each processor maintains one solution and performs the annealing process concurrently at a *constant* temperature that differs from processor to processor, and the solutions obtained by the processors are exchanged occasionally in some probabilistic way. An appropriate cooling schedule is automatically constructed from a given set of temperatures that are assigned to the processors. Thus we can avoid the task of carefully reducing the temperature over the course of time, which is essential for the performance of the conventional sequential algorithm.

In this paper we propose a scheme for the probabilistic exchange of solutions and justify it from the viewpoint of probability theory. We have applied our algorithm to a graph-partitioning problem. Results of experiments, and comparison with those by the sequential annealing algorithm and the Kernighan-Lin algorithm, are discussed.

1 Introduction

Many combinatorial optimization problems belong to the class of NP-hard problems [7], *i.e.*, there are no known deterministic algorithms that can solve them exactly and efficiently. In order to obtain approximate solutions for them, many heuristic algorithms have been devised with a great deal of effort. However, they are mostly problem-specific and cannot be applied to other types of problems, and they often have a common drawback: they are likely to stick to the so-called local optimals.

Simulated annealing (SA) is a general and powerful technique to solve these difficult combinatorial optimization problems [12]. It is a stochastic algorithm using pseudo-random numbers. The outline of the algorithm is as follows.

Let X be a solution space and $E : X \to \boldsymbol{R}$ be the objective function which we want to minimize. Given an arbitrary initial solution $x_0 \in X$, the algorithm generates a sequence of solutions $\{x_n\}_{n=0,1,2,\ldots}$ iteratively as follows, and finally outputs x_n for a large enough n.

(i) Perturb the current solution x_n randomly and get a candidate for the next solution x'_n.

(ii) Calculate the change in the objective function: $\Delta E = E(x'_n) - E(x_n)$.

(iii) When $\Delta E \leq 0$, accept the candidate: $x_{n+1} = x'_n$. When $\Delta E > 0$, accept the candidate with probability $p = \exp(-\Delta E/T_n)$, and reject it otherwise: $x_{n+1} = x_n$.

where $T_n > 0$ is a control parameter decreasing with n.

When $T_n \equiv +\infty$, SA is reduced to a blind random search; and when $T_n \equiv +0$, it is reduced to a greedy algorithm (iterative improvement) converging to one of the local optimals at hand. For general $0 < T < +\infty$, it behaves in a manner between these two extreme cases. In the following we refer to a sequence of operations (i)~(iii) as an *annealing step*, or simply, a *step*.

SA is based on the analogy between combinatorial optimization and statistical mechanics. The objective function E is referred to as *energy* and T_n is referred to as *temperature*. We call $\{T_n\}_{n=0,1,\ldots}$ a *cooling schedule*. When the temperature is constant ($T_n \equiv T$), SA simulates the equilibrium states of an imaginary physical system at temperature T, which corresponds to a combinatorial optimization problem. Hence the solution x_n is distributed according to the Boltzmann distribution at temperature T. This Boltzmann distribution converges to the lowest energy states (optimal solutions) as the temperature decreases to zero. Hence one might expect that SA can provide the exactly optimal solutions in principle.

Hajek [9] characterized the necessary and sufficient condition of the cooling schedule for convergence to

the optimal solutions. It is essentially given by

$$T_n \to 0 \qquad \text{and} \qquad T_n \geq \frac{A}{\log n}$$

where $A > 0$ is a constant representing the "roughness" of the "energy landscape". Such a cooling schedule requires an prohibitively long computation time, and hence is inadequate for practical use.

It is well-known that a cooling schedule has a great influence on the performance of SA; a poor schedule may lead to only poor solutions. Here arises the *cooling schedule problem*: how slowly should we decrease the temperature so as to get as better solution as possible within a given number of annealing steps (within a given amount of computation time). However, characterizing the ideal cooling schedule is also a difficult stochastic control problem.

In practice, such a simple cooling schedule as:

$$T_n = \alpha^{\lfloor \frac{n}{K} \rfloor} \cdot T_0, \qquad n = 1, 2, \ldots, N$$
$$0 < \alpha < 1, \qquad K \gg 1$$

is often used [12]. When the initial temperature T_0 and the final temperature T_N are chosen properly and both K and N/K (the number of so-called inner loops and outer loops) are large enough, such a cooling schedule has been known to work well in many applications. In order to further reduce the computation time and improve the quality of the solution, many more elaborate and empirically efficient cooling schedules have been proposed [14, 15].

SA has received much attention since it can provide much better solutions than conventional heuristics, although it often requires lengthy execution time [17]. In order to accelerate its execution, the parallelization of SA has been studied extensively [1, 3, 4, 5, 13]. The most popular approach is the so-called "parallel moves with lazy updates": — the solution data is divided between the processors, which perform annealing processes concurrently to improve different parts of the shared solution while occasionally exchanging the updated information. In order to reduce the costly interprocessor communications which damage the speed-up, minor errors in calculating the energy function are often admitted in practice. However, this harms the soundness of the algorithm from the theoretical point of view.

In this paper we propose a new parallel simulated annealing algorithm, which automatically constructs an appropriate cooling schedule from a given set of temperatures. Hence it partly solves the cooling schedule problem. Namely, it automatically determines *how many steps should be spent at each temperature*: the majority of steps should be spent at some *critical* temperatures around which the energy reduces remarkably. Such a cooling schedule is constructed stochastically based on the same principle as that behind simulated annealing itself. Parallelization is employed in the temperature dimension, and the cooling schedule is embedded in the parallel execution and never manifests itself explicitly. This algorithm is schedule-less or *time-homogeneous* in the sense that there are no time-dependent control parameters.

The organization of this paper is as follows. We present the time-homogeneous parallel annealing algorithm in Section 2, and discuss its convergence properties in Section 3. Results of experiments and comparison with other methods are discussed in Section 4. Finally conclusions are given in Section 5.

2 Time-homogeneous Parallel SA

2.1 Outline of the algorithm

The basic idea is to use parallelism in temperature, to perform annealing processes concurrently at various temperatures instead of sequentially reducing the temperature over the course of time.

The outline of the algorithm is as follows. Each processor maintains one solution and performs the annealing process concurrently at a *constant* temperature that differs from processor to processor. After every k annealing steps, every pair of processors with adjacent temperatures performs a *probabilistic exchange of solutions*: exchange each other's solution with some probability p or do nothing with probability $1 - p$, where p is appropriately defined and calculated for each pair. In order to avoid the possible collisions between the pair-wise exchanges, half of them are performed $k/2$ steps after the other half (Fig.1). The algorithm can be stopped at any time after a large number of steps and we will find a well-optimized solution on the processor that has the lowest temperature. We refer to k as the *period of exchange*, $f = 1/k$ as the *frequency of exchange*, and p as the *probability of exchange*.

Since exchanging the solutions between processors with different temperatures is nothing but changing the temperature for each participant solution, each solution will select its appropriate cooling schedule dynamically through successive competitions with others for lower temperatures. We will define the probability of exchanges so that it is advantageous to the solutions with low energies. Hence, the solution at the lowest temperature, the winner of these competitions, is expected to be the best solution so far. The cooling

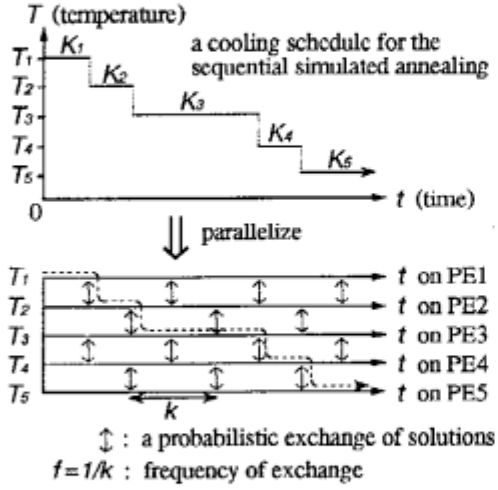


Fig.1. Parallelization in Temperature

schedule adopted by this winner, which is dynamically and stochastically determined, is supposed to be an efficient cooling schedule and is invisibly embedded in the parallel execution (Fig.1).

Exchanges often occur between a quenched solution quickly obtained at a lower temperature and a good solution found by chance at a higher temperature. After such an exchange, the latter will be improved more greedily at the lower temperature, and the former will be annealed again at the higher temperature so that it can escape from the valley of a local optimum for further improvement. Without these exchanges, quenched solutions will stay around the local optima, and good solutions found by chance will be deformed and lost.

An essential question is how to define and calculate the probability of exchange. This will be discussed in the next subsection, and will be subjected to probabilistic analysis in Section 3.

Since this algorithm maintains a set of solutions and competitions between them lead to the optimization, it may share some features with the genetic algorithm [8]. However, they are totally different in principle. The optimization here owes most to the annealing process performed at each processor.

2.2 Probability of exchange

In order to find a proper definition of the probability of exchange, we first investigate the necessary condition which a probabilistic exchange of solutions must satisfy.

Imagine that the annealing process is performed *independently* (*i.e.* without any exchanges) at each processor at a distinct constant temperature. Then the distribution of the solution in each processor converges to the Boltzmann distribution of the respective temperature [14]. The lower the temperature is, the better the solution that will be found, but after a longer time.

Now we introduce probabilistic exchanges of the solutions between the processors and intend to accelerate the convergence so that we can find a better solution at the lowest temperature more quickly.

Let $p(T, E, T', E')$ denote the probability of the exchange between two solutions: one with energy E at temperature T and the other with energy E' at temperature T'. Since we expect a better solution at a lower temperature, we always exchange the solutions if we find the better solution at the higher temperature. Namely, we define:

$$(T - T')(E - E') < 0 \quad \Rightarrow \quad p(T, E, T', E') = 1$$

On the other hand, when $(T - T')(E - E') \geq 0$, $p(T, E, T', E')$ is *uniquely* determined as follows. In order to accelerate the convergence to the equilibrium, a probabilistic exchange of solutions must not disturb the equilibrium — this is only a necessary condition. Namely, the detailed balance equation must hold between the distributions before and after the probabilistic exchange of solutions:

$$\frac{1}{Z(T)} \exp(-\frac{E}{T}) \cdot \frac{1}{Z(T')} \exp(-\frac{E'}{T'}) \cdot p(T, E, T', E')$$
$$= \frac{1}{Z(T)} \exp(-\frac{E'}{T}) \cdot \frac{1}{Z(T')} \exp(-\frac{E}{T'}) \cdot 1$$

$$\therefore \quad p(T, E, T', E') = \exp\left\{-\frac{(T - T')(E - E')}{TT'}\right\}$$

where $Z(T)$ denotes the partition function:

$$Z(T) = \sum_{x \in X} \exp(-\frac{E(x)}{T})$$

Therefore we are led to the definition:

$$p(T, E, T', E') = \begin{cases} 1 & \text{if } \Delta T \cdot \Delta E < 0 \\ \exp(-\frac{\Delta T \cdot \Delta E}{TT'}) & \text{otherwise} \end{cases}$$

where $\Delta T = T - T'$, $\Delta E = E - E'$.

We will confirm in Subsection 3.3 that such a probabilistic exchange of solutions in fact *accelerates* the convergence.

The above derivation uses the same principle as the Metropolis's criterion in the ordinary simulated annealing, although the exponent in the above expression has a similar but distinct form. Exchanging the

solutions *only* in the case $\Delta T \cdot \Delta E < 0$ is a too greedy strategy and would spoil the equilibrium.

Note that $p(T, E, T', E') > 0$ for arbitrary $T, T' > 0$ and $E, E' \in \boldsymbol{R}$. This means that a solution can go through a *non-monotonic* cooling schedule.[1]

This probability has a quite different form from that of choosing a solution-temperature pair in the systolic statistical cooling algorithm by E. Aarts *et al.* [1]. Given two solution-temperature pairs, one with energy E at temperature T and the other with energy E' at temperature T', they defined the probability of choosing the former pair by

$$p = \frac{p_0}{p_0 + p_1}, \quad \text{where} \quad p_0 = \frac{1}{Z(T)} \cdot \exp(-\frac{E}{T})$$
$$\text{and} \quad p_1 = \frac{1}{Z(T')} \cdot \exp(-\frac{E'}{T'})$$

Such a definition requires computing the partition functions $Z(T)$ and $Z(T')$, which is not an easy task. The advantage of our definition is that it does not contain the partition function and hence can be computed easily, and that we can give a rather easy proof of the convergence of the algorithm (*cf.* Section 3).

2.3 Time-homogeneity

A remarkable feature of this algorithm is that it is *time-homogeneous*, since the temperature on each processor remains constant. An appropriate cooling schedule is dynamically constructed from a given set of temperatures, however, it is embedded in the parallel execution and never manifests itself explicitly. In other words, this algorithm automatically decides *how many steps should be spent at each temperature, i.e., at each processor*. Thus, the cooling schedule problem is partly solved.

The time-homogeneity of the algorithm is advantageous when we want to continue the execution until a satisfiable solution is found. We can stop the execution at any time and examine whether a satisfiable solution has already been obtained. If one has not, we can resume the execution again *without any rescheduling* and continue it for a better solution. — In contrast, in the conventional simulated annealing, when an obtained solution is not satisfiable, we have to increase the temperature again and repeat the time-consuming annealing process, since we should not stick to the "quenched" solution that will not make a major improvement.

[1] Strenski *et al.* showed a striking fact that the optimal cooling schedule is non-monotonic in some cases [18].

2.4 Assignment of temperatures

In our algorithm, it is necessary to allocate an appropriate temperature to each processor beforehand. Namely, we have to specify a set of temperatures, from which the algorithm will construct a cooling schedule. This set of temperatures should be chosen large enough from a wide range, since the following may damage the quality of the solution.

(i) If it does not cover the high temperature region, the search will be restricted to a narrow area around the initial solution.

(ii) If it does not cover the low temperature region, the solution will keep being deformed and will never settle down to low-energy states.

(iii) If the neighboring temperatures are too much apart from each other, the probability of exchange will be only slight.

Conversely, if the set of temperatures is too rich, only a certain subset of it may contribute to the optimization effectively. An extra high temperature annealing will just keep generating high-energy solutions and will seldom have influence on the others. An extra low temperature annealing will virtually do nothing. Thus an excessive set of temperatures may waste the processors, but will never increase the execution time.

Getting a necessary and sufficient set of temperatures is the remaining part of the cooling schedule problem, which is not solved here. The estimation of the equilibrium (static) relation between the temperature and the energy, *e.g.*, the concepts of the *scales* [19], will be useful. However, such an estimation cannot directly provide the number of steps that should be devoted to each temperature. It is just empirically believed that they should be determined according to the heat capacity. In general, predicting the heat capacity in lower temperature is difficult, and estimating it on run time is expensive since the estimation itself requires many annealing steps.

3 Convergence Properties

In this section we describe the behavior of the time-homogeneous parallel annealing algorithm as a *Markov chain* and study its convergence properties.

3.1 Representation by Markov chains

In order to fix our basic notations, we first review the Markov chain that describes the behavior of the

ordinary sequential simulated annealing at a constant temperature.

Let X be the *solution space* (a large finite set) and

$$E \ : \ X \longrightarrow \boldsymbol{R} \qquad i \longmapsto E(i) = E_i$$

be the *energy function*. Namely, we search for a solution $i \in X$ that minimizes E_i.

Random perturbations on a solution: $x_n \mapsto x'_n$, as in (i) in Section 1, can be described by the conditional probabilities:

$$s_{ij} = P(\, x'_n = j \ \mid \ x_n = i \,) \qquad \text{for } i, j \in X$$

A stochastic matrix $S = (s_{ij})_{i,j \in X}$ is called the *selection probability matrix*. We assume that S is symmetric and irreducible. Namely, it satisfies:

$$0 \le s_{ij} = s_{ji} \le 1, \qquad \sum_j s_{ij} = 1$$

and there exists $n \ge 1$ such that all elements in S^n are positive. The latter means that any solution in X can be reached from any other solution in X by a sequence of perturbations.

Then the behavior of the simulated annealing at a constant temperature $T > 0$ can be described as a Markov chain over X with a transition matrix A as follows. Let $i_0 \in X$ be the *initial solution* and x_t be the solution after t steps. The distribution of x_t is specified by a row vector $\boldsymbol{p}_t = (P(x_t = i))_{i \in X}$, which is given by:

$$\boldsymbol{p}_0 = (\delta_{ii_0})_{i \in X}, \quad \boldsymbol{p}_{t+1} = \boldsymbol{p}_t A \quad (t = 0, 1, 2, \ldots)$$

where δ_{ij} is Kronecker's delta, and A is a matrix defined by

$$A = A(\beta) = (a_{ij}(\beta)),$$

$$a_{ij} = a_{ij}(\beta) = \begin{cases} s_{ij} e^{-\beta(E_j - E_i)_+} & (j \ne i) \\ 1 - \displaystyle\sum_{k \ne i} s_{ik} e^{-\beta(E_k - E_i)_+} & (j = i) \end{cases}$$

where $w_+ = \max(w, 0)$ for $w \in \boldsymbol{R}$ and $\beta = 1/T$ is the *inverse temperature*.

In the limit $t \to +\infty$, $\boldsymbol{p}_t$ converges to the *Boltzmann distribution* $\pi = \pi(\beta)$.

$$\pi(\beta) = \left(\frac{e^{-\beta E_i}}{Z(\beta)} \right)_{i \in X}, \quad \text{where} \quad Z(\beta) = \sum_{i \in X} e^{-\beta E_i}$$

Now, let us turn to the time-homogeneous parallel annealing algorithm. Let N be the number of processors and $T_n = 1/\beta_n$ be the temperature of the n-th processor such that $0 < \beta_1 < \beta_2 < \cdots < \beta_N$. If there is no probabilistic exchange of solutions and each processor performs annealing independently, its behavior is described as a Markov chain over X^N with a transition matrix $\tilde{A}$, which is given by the tensor product of the above transition matrices. Namely,

$$\tilde{A} = A(\beta_1) \otimes A(\beta_2) \otimes \cdots \otimes A(\beta_N)$$

$\tilde{A}$ corresponds to performing one annealing step at each temperature. In the limit of $t \to \infty$, the equilibrium state is given by a probability vector $\tilde{\pi}$:

$$\tilde{\pi} = \pi(\beta_1) \otimes \pi(\beta_2) \otimes \cdots \otimes \pi(\beta_N)$$

Probabilistic exchanges of solutions can be described by similar transition matrices. Firstly let us consider the case with $N = 2$, $T_1 = T$ and $T_2 = T'$. Then the transition matrix is given by

$$C = C(\beta, \beta') = (c_{(ij)(kl)})$$

$$c_{(ij)(kl)} = \begin{cases} c_{ij} & (i,j) = (l,k) \\ 1 - c_{ij} & (i,j) = (k,l) \ne (j,i) \\ 0 & \text{otherwise} \end{cases}$$

$$c_{ij} = c_{ij}(\beta, \beta') = \exp[\min\{0, (\beta - \beta')(E_i - E_j)\}]$$

$$\beta = 1/T, \qquad \beta' = 1/T'$$

For the general case with $N > 2$, there are two parallel probabilistic exchanges, each of which consists of an exhaustive disjoint set of probabilistic exchanges with adjacent temperatures. Namely, they are expressed by the transition matrices $\tilde{C}_{\text{even}}$ and $\tilde{C}_{\text{odd}}$ respectively:

$$\tilde{C}_{\text{even}} = \begin{cases} C_1 \otimes C_3 \otimes \cdots \otimes C_{N-1} & (N : \text{even}) \\ C_1 \otimes C_3 \otimes \cdots \otimes C_{N-2} \otimes I_X & (N : \text{odd}) \end{cases}$$

$$\tilde{C}_{\text{odd}} = \begin{cases} I_X \otimes C_2 \otimes C_4 \otimes \cdots \otimes C_{N-2} \otimes I_X & (N : \text{even}) \\ I_X \otimes C_2 \otimes C_4 \otimes \cdots \otimes C_{N-1} & (N : \text{odd}) \end{cases}$$

where $C_n = C(\beta_n, \beta_{n+1})$ for $1 \le n < N$ and I_X denotes the unit matrix of degree $|X|$. These two parallel probabilistic exchanges of solutions are repeatedly performed one after another every $k/2$ annealing steps, where k is the period of exchange. For simplicity we assume that k is a positive even integer.

Thus the behavior of the time-homogeneous parallel annealing algorithm after every k annealing steps can be described by a time-homogeneous Markov chain $\mathcal{M}$ with a transition matrix $\tilde{P}$:

$$\tilde{P} = \tilde{A}^{k/2} \tilde{C}_{\text{odd}} \tilde{A}^{k/2} \tilde{C}_{\text{even}}$$

$\tilde{P}$ corresponds to one period of the algorithm, namely, k annealing steps at each processor and one probabilistic exchange of solutions at every pair of processors with adjacent temperatures.

3.2 Convergence in law

Since the behavior of the time-homogeneous parallel annealing algorithm can be described by a time-homogeneous Markov chain $\mathcal{M}$, we can easily establish its convergence property:

$$\lim_{t\to\infty} \tilde{p}_t = \tilde{\pi} \qquad \text{(convergence in law)}$$

where $\tilde{p}_t$ represents the distribution over X^N of the solutions in the processors after t periods (after kt steps at each processor). This follows from the facts that the state space X^N is a finite set and that the transition matrix $\tilde{P}$ is irreducible and aperiodic [6]. These facts are verified immediately.

3.3 Monotone convergence property

In Subsection 2.2, investigating the necessary condition that a probabilistic exchange of solutions must satisfy, we saw that the probability of exchange is uniquely determined. In this subsection, we show that such a probabilistic exchange of solutions in fact *accelerates* the convergence of the algorithm. In order to measure how close the current distribution of solutions approaches the equilibrium distribution, we use Kullback-Leibler divergence. For two probability distribution specified by probability vectors $\boldsymbol{p} = (p_i)$ and $\boldsymbol{q} = (q_i)$, the Kullback-Leibler divergence defined by

$$D(q\|p) = -\sum_i q_i \log \frac{p_i}{q_i} \geq 0$$

represents the discrepancy between them [2]. Here the strict inequality holds unless $\boldsymbol{p} = \boldsymbol{q}$.

To begin with, we note a monotone convergence property of the ordinary sequential simulated annealing at a *constant* temperature. Let X, E, A and $\boldsymbol{p}_t$ be as in the beginning of Subsection 3.1, and denote an arbitrary probability distribution over the solution space X by a $|X|$-dimensional row probability vector $\boldsymbol{p} = (p_i)_{i\in X}$.

Lemma 1

$$D(\boldsymbol{\pi}\|\boldsymbol{p}) \geq D(\boldsymbol{\pi}\|\boldsymbol{p}A) \qquad for\ \ \forall \boldsymbol{p}$$

PROOF: From the definition of $\boldsymbol{\pi}$, we have

$$D(\boldsymbol{\pi}\|\boldsymbol{p}) = -\frac{1}{Z}\sum_i e^{-\beta E_i}\log p_i - \frac{\beta}{Z}\sum_i e^{-\beta E_i}E_i - \log Z$$

From the definition of a_{ij} and the symmetry of S, we have

$$e^{-\beta E_i}a_{ij} = e^{-\beta E_j}a_{ji}$$

Owing to the fact that A is a stochastic matrix and that the logarithmic function is concave, we can derive:

$$\begin{aligned}
&\sum_i e^{-\beta E_i}\log\Big(\sum_j p_j a_{ji}\Big) \\
\geq\ &\sum_i e^{-\beta E_i}\sum_j a_{ij}\{\log p_j + \beta(E_j - E_i)\} \\
=\ &\sum_{i,j} e^{-\beta E_j}a_{ji}\log p_j \\
&+\beta\sum_{i,j}(e^{-\beta E_j}E_j a_{ji} - e^{-\beta E_i}E_i a_{ij}) \\
=\ &\sum_j e^{-\beta E_j}\log p_j
\end{aligned}$$

Thus we obtain the desired result. ∎

For a probabilistic exchange of solutions between two processors, we can establish a similar result as follows. Let $N = 2$ and $C = C(\beta, \beta')$ be as before and denote an arbitrary probability distribution over $X \times X$ of a pair of solutions in the two processors by a $|X|^2$-dimensional row probability vector $\tilde{\boldsymbol{p}} = (p_{ij})_{i,j\in X}$.

Lemma 2

$$D(\tilde{\boldsymbol{\pi}}\|\tilde{\boldsymbol{p}}) \geq D(\tilde{\boldsymbol{\pi}}\|\tilde{\boldsymbol{p}}C) \qquad for\ \ \forall \tilde{\boldsymbol{p}}$$

PROOF: From the definition of $\tilde{\boldsymbol{\pi}}$, we have

$$\begin{aligned}
&D(\tilde{\boldsymbol{\pi}}\|\tilde{\boldsymbol{p}}) \\
=\ &-\frac{1}{Z(\beta)Z(\beta')}\sum_{i,j} e^{-(\beta E_i+\beta' E_j)}\log p_{ij} \\
&-\frac{\beta}{Z(\beta)}\sum_i e^{-\beta E_i}E_i - \frac{\beta'}{Z(\beta')}\sum_j e^{-\beta' E_j}E_j \\
&-\log Z(\beta) - \log Z(\beta')
\end{aligned}$$

From the definition of c_{ij}, we have

$$e^{-(\beta E_i+\beta' E_j)}c_{ij} = e^{-(\beta E_j+\beta' E_i)}c_{ji}$$

Owing to the fact that C is a stochastic matrix and that the logarithmic function is concave, we can derive:

$$\begin{aligned}
&\sum_{i,j} e^{-(\beta E_i+\beta' E_j)}\log\{(1-c_{ij})p_{ij} + c_{ji}p_{ji}\} \\
\geq\ &\sum_{i,j} e^{-(\beta E_i+\beta' E_j)}\{-\beta E_i - \beta'\ E_j \\
&+(1-c_{ij})\log(e^{\beta E_i+\beta' E_j}p_{ij})
\end{aligned}$$

$$
\begin{aligned}
& + c_{ij} \log(e^{\beta E_j + \beta' E_i} p_{ji})\} \\
= & \sum_{i,j} e^{-(\beta E_i + \beta' E_j)} (1 - c_{ij}) \log p_{ij} \\
& + \sum_{i,j} e^{-(\beta E_j + \beta' E_i)} c_{ji} \log p_{ji} \\
& - \sum_{i,j} e^{-(\beta E_i + \beta' E_j)} c_{ij} (\beta - \beta')(E_i - E_j) \\
= & \sum_{i,j} e^{-(\beta E_i + \beta' E_j)} \log p_{ij}
\end{aligned}
$$

Thus we obtain the desired result. ∎

Now we extend these results to the general case with $N > 2$. Let $\tilde{A}$, $\tilde{C}_{\text{even}}$, $\tilde{C}_{\text{odd}}$ and $\tilde{P}$ be as before and denote an arbitrary probability distribution over X^N of the solutions in the N processors by a $|X|^N$-dimensional row probability vector $\tilde{\boldsymbol{p}}$.

Theorem 3 *For arbitrary $\tilde{\boldsymbol{p}}$,*

$$
\begin{aligned}
D(\tilde{\boldsymbol{\pi}} \| \tilde{\boldsymbol{p}}) &\geq D(\tilde{\boldsymbol{\pi}} \| \tilde{\boldsymbol{p}} \tilde{A}) \\
D(\tilde{\boldsymbol{\pi}} \| \tilde{\boldsymbol{p}}) &\geq D(\tilde{\boldsymbol{\pi}} \| \tilde{\boldsymbol{p}} \tilde{C}_{\text{even}}) \\
D(\tilde{\boldsymbol{\pi}} \| \tilde{\boldsymbol{p}}) &\geq D(\tilde{\boldsymbol{\pi}} \| \tilde{\boldsymbol{p}} \tilde{C}_{\text{odd}})
\end{aligned}
$$

where the strict inequalities hold unless $\tilde{\boldsymbol{p}} = \tilde{\boldsymbol{\pi}}$.

This theorem implies that performing a parallel probabilistic exchange of solutions as well as performing one annealing step at each processor in fact *accelerates* the convergence of the algorithm, since applying each of $\tilde{A}$, $\tilde{C}_{\text{even}}$ or $\tilde{C}_{\text{odd}}$ always reduces the Kullback-Leibler divergence of the current distribution to the equilibrium distribution.

PROOF: $\tilde{A}$ can be decomposed as

$$
\begin{aligned}
\tilde{A} &= \tilde{A}_1 \cdot \tilde{A}_2 \cdot \cdots \cdot \tilde{A}_N \\
\tilde{A}_n &= I_X^{\otimes(n-1)} \otimes A(\beta_n) \otimes I_X^{\otimes(N-n-1)}
\end{aligned}
$$

where $I^{\otimes k}$ denotes the k-fold tensor product of I. As in Lemma 1, we have

$$
D(\tilde{\boldsymbol{\pi}} \| \tilde{\boldsymbol{p}}) \geq D(\tilde{\boldsymbol{\pi}} \| \tilde{\boldsymbol{p}} \tilde{A}_n) \qquad (1 \leq \forall n \leq N)
$$

Thus we can cstablish the first claim by induction. Other claims can be established similarly. ∎

Corollary 4 (monotone convergence to the equilibrium distribution)

$$
D(\tilde{\boldsymbol{\pi}} \| \tilde{\boldsymbol{p}}_t) \searrow 0 \qquad as \quad t \to \infty
$$

PROOF: The monotone decreasing property follows from Theorem 3 and the convergence to zero follows from the result of Subsection 3.2. ∎

4 Experimental Results

We have implemented our algorithm for a graph-partitioning problem on the Multi-PSI/V2 [16], an MIMD parallel machine with 64 processors.

(graph-partitioning problem) Given a graph $G = (\mathcal{V}, \mathcal{E})$, define a *label* on the vertices $\lambda : \mathcal{V} \to \{\pm 1\}$ so as to minimize E:

$$
E = - \sum_{(u,v) \in \mathcal{E}} \lambda(u) \lambda(v) + c \cdot (\sum_{v \in \mathcal{V}} \lambda(v))^2
$$

where $c > 0$ is a constant.

This is an NP-hard problem [7]. Kernighan-Lin algorithm efficiently gives its approximate solutions [11].

For a random graph G with 400 vertices and 2004 edges, we compared the results given by our algorithm with those by other methods (Fig.2).

(a) Time-homogeneous parallel annealing: All of 63 processors performed 20,000 annealing steps each at distinct constant temperature. The highest and lowest temperatures are determined empirically, and the other temperatures are determined so that adjacent ones have the same ratio. As for the frequency of exchange f, we examined various values ranging from 1/20,000 to 1/2. Each point represents the average over 30 runs with different sequences of random numbers.

(b) Sequential annealing: The cooling schedule consists of exactly the same sequence of 63 temperatures as those in (a). 20,000 annealing steps are performed, which are divided equally between the 63 temperatures. The final energy is indicated by a dashed line.

(c) Simple parallel annealing: Each of the 63 processors executes the sequential annealing with the cooling schedule described in (b) using a distinct sequence of random numbers. The result is the best solution among those obtained by them, and its energy is indicated by a dashed line.

(d) Kernighan-Lin: Kernighan-Lin algorithm is repeatedly applied several times until convergence. The final energy is indicated by a dashed line.

Fig.2. Energy vs Frequency of Exchange

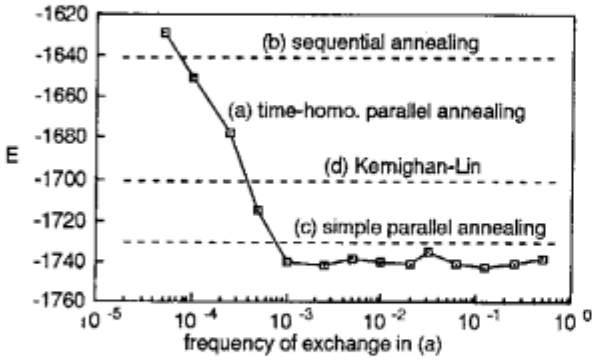
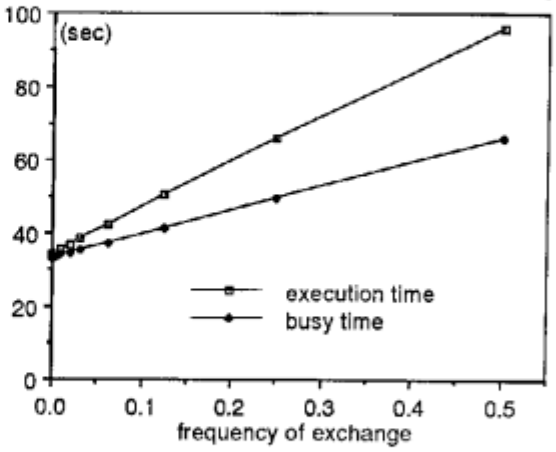


Fig.3. Execution Time vs Frequency of Exchange



We also measured the execution time of the time-homogeneous parallel annealing (a) for different values of f in order to see the overheads associated with the probabilistic exchanges of solutions between processors (Fig.3). Plotted points in Fig.2 and Fig.3 correspond to each other with the same values of f, although the abscissa are scaled differently (logarithmically and linearly). The gap between the execution time and the busy time represents the idle time, which is incurred by synchronization between the processors in exchanging the solutions. The execution time with $f \to 0$ almost agreed with those in (b) and (c).

We made the following observations from these results.

1. (a) gives the best solutions for a wide range of frequency of exchange: $1/1000 \leq f \leq 1/2$ (Fig.2). The overheads grow linearly with the frequency f, although they are rather immaterial for $f \leq 1/50$ (Fig.3). Thus this algorithm is fairly insensitive to the value of f.

2. Since 20,000 annealing steps are relatively small, (b) gives a worse solution than (d). However, in (a), the algorithm probabilistically selects an appropriate cooling schedule with 20,000 steps and gives a better solution.

3. Note that the total number of annealing steps in (a) and that in (c) are the same. (a) outperforms (c) unless f is too small.

5 Conclusions

We have proposed the time-homogeneous parallel annealing algorithm, in which an appropriate cooling schedule is automatically and probabilistically constructed from a given set of temperatures. And the cooling schedule problem is partly solved.

Time-homogeneity of the algorithm is advantageous when we want to prolong the execution for further optimization, since it does not require any re-scheduling.

The behavior of this algorithm is theoretically tractable, since it is described in terms of a *time-homogeneous* Markov chain. In particular, we have proved its monotone convergence property.

We have experimentally observed that this algorithm automatically constructed a better cooling schedule than that which assigned the same number of annealing steps at each temperature. We also observed that this algorithm is fairly insensitive to the choice of the frequency of exchanges.

The following require further investigation.

(i) How do we find the range of optimal frequency of exchange in general?

(ii) Does this algorithm probabilistically select the *theoretically best* cooling schedule, the *theoretically best* assignment of the annealing steps to each temperature?

(iii) How many processors (temperatures) are necessary and sufficient?

(iv) What is the optimal assignment of temperatures to the processors?

The last two questions are concerned with the part of the cooling schedule problem, which is not solved in our algorithm.

6 Acknowledgments

We would like to thank N. Ichiyoshi, K. Rokusawa, and E. Sugino for valuable discussions, and also D. Greening for calling our attention to [18].

References

[1] E.H.L. Aarts *et al.*, "Parallel Implementations of the Statistical Cooling Algorithm," *Integration*, 4, (1984).

[2] S. Amari, "Differential Geometric Methods in Statistics," Lecture Note in Statistics **28**, Springer-Verlag, (1985).

[3] P. Banerjee and M. Jones, "A Parallel Simulated Annealing for Standard Cell Placement on a Hypercube Computer," *Proc. Int. Conf. on CAD* (1986).

[4] A. Casotto *et al.*, "Placement of Standard Cells Using Simulated Annealing on the Connection Machine," *Proc. Int. Conf. on CAD* (1987).

[5] F. Darema, S. Kirkpatrick and V.A. Norton, "Parallel Algorithms for Chip Placement by Simulated Annealing ," *IBM J. Res. Dev.* 31(3), (1987).

[6] W. Feller, "An Introduction to Probability Theory and Its Applications," vol. 1, John Wiley & Sons (1957).

[7] M. Garey and D. Johnson, "Computers and Intractability, A Guide to the Theory of NP-Completeness," Freeman, New York, (1979).

[8] D.E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning," Addison-Wesley, (1989).

[9] B. Hajek, "Cooling Schedule for Optimal Simulated Annealing," *Math. Oper. Res.*, 13 (1988).

[10] M. Huang, F. Romeo and A. Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," *Proc. Int. Conf. on CAD* (1986).

[11] B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell. sys. tech. J.*, 49, (1969).

[12] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecci, "Optimization by Simulated Annealing," *Science*, vol.220, no.4598 (1983).

[13] S.A. Kravitz and R. Rutenbar, "Placement by Simulated Annealing on a Multiprocessor," *IEEE Trans. on CAD* vol.6, no.4 (1987).

[14] P.J.M. van Laarhoven and E.H.L. Aarts, "Simulated Annealing: Theory and Applications", Reidel (1987).

[15] J. Lam and J.-M. Delosme, "Performance of a New Annealing Schedule", *Proc. 25th Design Automation Conf.* (1988).

[16] K. Nakajima *et al.*, "Distributed Implementation of KL1 on the Multi-PSI/V2", *Proc. 6th Int. Conf. on Logic Programming* (1989).

[17] C. Sechen, and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE Journal of Solid-State Circuits*, vol. SC-20, no.2 (1985).

[18] P.N. Strenski and S. Kirkpatrick, "Analysis of Finite Length Annealing Schedule," *Algorithmica*, vol.6, no.3 (1991).

[19] S. R. White, "Concepts of Scales in Simulated Annealing," *Proc. Int. Conf. on Computer Design* (1984).