

An Efficient Simulated Annealing Schedule: Implementation and Evaluation

Jimmy Lam and Jean-Marc Delosme

Report 8817

September 1988

Department of Computer Science, Yale University.
Department of Electrical Engineering, Yale University.

Abstract

We present an implementation of an efficient general simulated annealing schedule and demonstrate experimentally that the new schedule achieves speedups often exceeding an order of magnitude when compared with other general schedules currently available in the literature. To assess the performance of simulated annealing as a general method for solving combinatorial optimization problems, we also compare the method with efficient heuristics on well-studied problems: the traveling salesman problem and the graph partition problem. For high quality solutions and for problems with a small number of close to optimal solutions, our test results indicate that simulated annealing outperforms the heuristics of Lin and Kernighan and of Karp for the traveling salesman problem, and multiple executions of the heuristic of Fiduccia and Mattheyses for the graph partition problem.

This research was supported by the Army Research Office under contract DAAL03-86-K-0158, by the National Science Foundation under grant ECS-8314750, and by the Office of Naval Research under contracts N00014-84-K-0092 and N00014-85-K-0461.

1. Introduction

We assume the reader is familiar with the simulated annealing method [12]. The physical analogy on which the method is based suggests that, to achieve good quality results, the system should be kept close to thermal equilibrium as the temperature is lowered. Most of the practical simulated annealing schedules have been designed following this heuristic. While the first schedules were problem specific, constructed from experiments, schedules of much wider applicability have been derived recently [1], [8], [14]. The starting point in the derivation of these general schedules is an approximate equilibrium criterion that expresses formally when the system is sufficiently close to equilibrium so that the temperature can be lowered.

Let $s_n \equiv 1 / T_n$ be the *inverse temperature* at step n (after a proposed move is either accepted or rejected) and let $\bar{X}(s_n)$ be the average energy of the system, or cost of the current solution, at step n . We consider the system in quasi-equilibrium at inverse temperature s_n if it satisfies the relation

$$\left| \bar{X}(s_{n-1}) - \mu(s_n) \right| \leq \lambda \sigma(s_n), \quad (1)$$

where λ is a user specified parameter, and $\mu(s_n)$ and $\sigma(s_n)$ are, respectively, the equilibrium average and standard deviation of the energy at inverse temperature s_n [14]. To get good quality results we seek a schedule, called a λ -*schedule*, that lowers the temperature at every step (i.e. $s_{n+1} \geq s_n$) while keeping the system in quasi-equilibrium at all times. Higher quality results can always be obtained by reducing λ so that the new λ -schedule ensures a closer approximation of equilibrium. However this improvement comes at the expense of increased computation time.

For a given value of λ and a given move generation strategy (or move selection process), the λ -schedule

$$s_{n+1} = s_n + \lambda \frac{\rho_2(s_n)}{2\sigma^3(s_n)}, \quad (2)$$

where $\rho_2(s_n)$ is the variance of the actual energy change in one step at inverse temperature s_n , is shown in [14] to give at every step the minimum average energy among all λ -schedules. Such a schedule is called an *efficient λ -schedule*; there is one such schedule for each move generation strategy. Since the move generation strategy affects only the factor $\rho_2(s_n)$ in the efficient λ -schedule, if a move generation strategy maximizes $\rho_2(s_n)$ for all inverse temperatures s_n , the associated efficient λ -schedule minimizes the temperature at every step while satisfying the quasi-equilibrium condition (1) and hence minimizes the average energy at every step.

The factor $\rho_2(s_n)$ cannot be made arbitrarily large. In fact its value is quite restricted and we show in [14], using models for the energy density function (or density of states) and the move generation strategy, that it is uniquely determined once the acceptance ratio $\rho_0(s_n)$ (the probability that a proposed move is accepted) is known. The substitution of $\rho_2(s_n)$ by its expression in terms of $\rho_0(s_n)$ gives the final formula for the efficient λ -schedules

$$s_{n+1} = s_n + \left[\frac{\lambda}{\sigma(s_n)} \right] \left[\frac{1}{s_n^2 \sigma^2(s_n)} \right] \left[\frac{4\rho_0(s_n)(1 - \rho_0(s_n))^2}{(2 - \rho_0(s_n))^2} \right]. \quad (3)$$

The increment in inverse temperature is the product of three factors. The first, which also appears in the schedules of Aarts and van Laarhoven [1] and Huang *et al.* [8], reflects what is common to the quasi-equilibrium criteria. The second is the inverse of the specific heat; its

presence is in agreement with the physical intuition that the larger is the specific heat, the slower should be the cooling. The third shows how the temperature decrement is affected by the move generation strategy, through a function of the acceptance ratio only. Since this factor is maximized if the acceptance ratio is equal to 44%, the move generation strategy should be controlled to keep the acceptance ratio constant, equal to that value. The possibility of such a regulation of the acceptance ratio provides a basis for selection between move generation strategies; for instance, if a strategy manages to generate a higher proportion of downhill moves at low temperatures than another strategy, the associated acceptance ratio will not drop as far below the 44% target value and that strategy should be preferred in the low temperature range. The active control of the move generation to regulate the acceptance ratio is an integral part of our schedule, working in concert with the inverse temperature update formula (3).

In the next section, a procedure for the estimation and control of the parameters in the schedule is described together with other practical issues. In Section 3, the performance of the schedule is assessed and the performance of simulated annealing as a general combinatorial optimization method is discussed. Finally, concluding remarks are given in Section 4.

2. Estimation and control of simulated annealing parameters

Since the inverse temperature update formula (3) depends on the acceptance ratio, $\rho_0(s_n)$, and the variance of the energy, $\sigma^2(s_n)$, successful application of the schedule requires proper estimation of these quantities. This section is devoted to these estimation problems and to the method of move generation control, which strongly influences the performance of the schedule.

2.1. Estimation of the standard deviation of the energy

In general, the values of $\rho_0(s_n)$ and $\sigma(s_n)$, needed to update the inverse temperature, are not known a priori and must be estimated. Since the acceptance ratio varies slowly from one inverse temperature to another, the ratio of the number of accepted moves to the number of proposed moves in the last τ steps, with τ between 100 to 1,000, provides a good approximation for the current ρ_0 . (We have also experimented with a more elaborate scheme for estimating ρ_0 , as a weighted average of the last few measured ratios, and did not observe any significant difference between the final results of the two methods.) Unfortunately, such a simple procedure cannot be used for the variance of the energy, $\sigma^2(s)$. If the inverse temperature is fixed and the system is in equilibrium, the estimator

$$\hat{\sigma}^2(s) = \frac{1}{\tau} \sum_{i=1}^{\tau} (x_i - \mu(s))^2,$$

where x_i is the energy at step i and $\mu(s)$ is the average energy, gives an unbiased estimate of $\sigma^2(s)$ provided that $\mu(s)$ is known. However $\mu(s)$ is not known and worse, the system is never in equilibrium since with the efficient λ -schedule the inverse temperature is updated at every step. Therefore, the preceding formula cannot be used by itself to estimate $\sigma^2(s)$. We need an estimator that gives the current value of $\sigma^2(s)$ using measurements made at previous values of the inverse temperature. Such an estimator can be constructed using the energy density model introduced in [14].

If the energy density function, $P(x)$, were known we could compute the mean and variance of the energy at different inverse temperatures with

$$\mu(s) = -\frac{1}{Z(s)} \frac{\partial Z(s)}{\partial s}, \quad (4)$$

and

$$\sigma^2(s) = -\frac{\partial \mu(s)}{\partial s}, \quad (5)$$

where

$$Z(s) = \int_{-\infty}^{\infty} P(x) e^{-sx} dx$$

is known as the *partition function*. However $P(x)$ is unknown and, moreover, it is very hard to estimate the parameters of a model of $P(x)$ on-line. The data gathered up to step n may give a fairly good description of $P(x)$ for values of x above and around $\mu(s_n)$ but are too scanty to enable the accurate description of the lower tail of $P(x)$ which, as can be seen from the expression of $Z(s)$, is most important to estimate $\sigma^2(s)$ for $s > s_n$. A fixed model for $P(x)$, usable at all temperatures, would require a large number of parameters and that sheer number, combined with the scarcity of the data in the lower tail, would make the on-line estimation of these parameters exceedingly difficult. Thus we seek instead a model with a small number of parameters that represents well $P(x)e^{-sx}$ for s close to the current temperature s_n . The price to pay for that smaller number of parameters is that the parameters vary with the temperature; in order to enable parameter estimation this variation should be slow, hence the quality of the model should be sufficiently high, and this imposes a lower bound on how few parameters one can use. The theory developed in [14] suggests a model for which that lower bound turns out to be quite low in practice. The idea is to model the density function $Q_s(x)$, which is proportional to $\sqrt{P(x)}e^{-sx}$, as a weighted sum of gamma density functions

$$x_0 + \sum_{i=0}^M q_i(s_n) G(N(s_n)+i, b(s_n)) e^{-sx},$$

where x_0 is the global minimum and

$$G(N, b) = \frac{b^N}{\Gamma(N)} (x - x_0)^{N-1} e^{-b(x-x_0)}, \quad x > x_0.$$

For typical problems this model is shown experimentally to be quite satisfactory with $M = 0$ [14]. The induced model for $P(x)e^{-sx}$ is of the same form, with M , $N(s_n)$ and $b(s_n)$ replaced by $\bar{M} = 2M$, $\bar{N}(s_n) = 2N(s_n) - 1$ and $\bar{b}(s_n) = 2b(s_n)$ [13]. Note that $\bar{M} = 0$ if $M = 0$.

Using this model we obtain the approximation for the partition function:

$$Z(s) = \int_{-\infty}^{\infty} \left[\sum_{i=0}^{\bar{M}} p_i(s_n) G(\bar{N}(s_n)+i, \bar{b}(s_n)) \right] e^{-sx} dx.$$

There is a whole family of such approximations since, according to the value of the inverse temperature during the annealing process, one would use the model with the corresponding values $\bar{N}(s_n)$ and $\bar{b}(s_n)$. Evaluation of the integral yields

$$Z(s) = \sum_{i=0}^{\bar{M}} p_i(s_n) e^{-sx_0} \bar{b}(s_n)^{\bar{N}(s_n)+i} (\bar{b}(s_n)+s)^{-\bar{N}(s_n)-i},$$

which can be substituted into the expression for $\mu(s)$ in (4) to obtain

$$\mu(s) = x_0 + \frac{\sum_{i=0}^{\bar{M}} p_i(s_n) (\bar{N}(s_n) + i) \bar{b}(s_n)^{\bar{N}(s_n)+i} (\bar{b}(s_n)+s)^{-\bar{N}(s_n)-i-1}}{\sum_{i=0}^{\bar{M}} p_i(s_n) \bar{b}(s_n)^{\bar{N}(s_n)+i} (\bar{b}(s_n)+s)^{-\bar{N}(s_n)-i}}.$$

Multiplying both the numerator and the denominator of the above expression by $(\bar{b}(s_n)+s)^{\bar{M}+1}$, expanding powers of $\bar{b}(s_n)+s$ into powers of s , and grouping terms according to their powers, we arrive at

$$\mu(s) = x_0 + \frac{\sum_{i=0}^{\bar{M}} a_i(s_n) s^i}{\sum_{i=0}^{\bar{M}+1} b_i(s_n) s^i}. \quad (6)$$

(Since there are $2\bar{M} + 3$ coefficients and only $\bar{M} + 3$ independent variables, namely $\bar{N}(s_n)$, $\bar{b}(s_n)$, and $p_i(s_n)$ for $i = 0, 1, \dots, \bar{M}$, the coefficients are not independent of each other for $\bar{M} \geq 1$.) If these coefficients can be found, the variance of the energy, $\sigma^2(s)$, can be computed using (5) to get

$$\sigma^2(s) = \frac{\sum_{i=0}^{\bar{M}} a_i(s_n) s^i \sum_{i=1}^{\bar{M}+1} i b_i(s_n) s^{i-1} - \sum_{i=1}^{\bar{M}} i a_i(s_n) s^{i-1} \sum_{i=0}^{\bar{M}+1} b_i(s_n) s^i}{\left(\sum_{i=0}^{\bar{M}+1} b_i(s_n) s^i \right)^2}. \quad (7)$$

Since $\mu(s)$ and $\sigma^2(s)$ are obtained from successive derivatives of an approximation to $Z(s)$, the relative approximation errors can be expected to grow larger as one goes from $Z(s)$ to $\mu(s)$ and then to $\sigma^2(s)$. Unfortunately it is the last element in that chain that interests us. To appreciate the increase in relative error when going from $\mu(s)$ to $\sigma^2(s)$ observe that, if $a_i(s)$ and $b_i(s)$ were known functions, a better estimate of $\sigma^2(s)$ would be found by taking the derivative of the coefficients with respect to s when applying (5) at $s = s_n$; hence one source of inaccuracy when employing (7) with the coefficients a_i and b_i equal to those obtained for the approximation (6) to $\mu(s)$ is that the variation of the coefficients is not taken into account. To circumvent this difficulty we shall merely use (6) and (7) to provide the functional form of $\mu(s) - x_0$ and $\sigma^2(s)$, as rational functions of s of numerator degrees \bar{M} and $2\bar{M}$, respectively, and denominator degrees $\bar{M} + 1$ and $2(\bar{M} + 1)$, respectively. The parameter \bar{M} is selected sufficiently large such that the coefficients vary slowly with s_n . To simplify the estimation task and also make it less sensitive to modeling assumptions, the coefficients of the two rational functions will be assumed independent within the functions as well as between the functions. By estimating $\sigma(s)$ independently, without differentiating $\mu(s)$, one can get good quality estimates without knowing how the coefficients of the approximation to $\mu(s)$ vary with s .

In order to estimate the coefficients in the rational model for $\sigma^2(s)$, we use the method of weighted least-squares. First we have to measure $\sigma^2(s_i)$. Given λ , we select a τ such that $\mu(s)$ changes very little for the last τ steps. Then, we can use

$$u_j = \frac{1}{\tau} \sum_{i=j-\tau+1}^j x_i \quad (8)$$

as the measured average energy of the system at step j . However, if we went on to use

$$\dot{v}_i^2 = \frac{1}{\tau} \sum_{j=i-\tau+1}^i (x_j - u_j)^2 \quad (9)$$

as the measured energy variance we would grossly underestimate it because the sequence of measurements for x_j is highly correlated. Indeed, the estimator (9) is biased and its expected value is given by

$$E \{ \dot{v}_i^2 \} = \sigma^2(s_i) [1 - \{1 + 2 \sum_{j=1}^{\tau-1} (1 - j/\tau) r_j(s_i)\} / \tau],$$

where $r_j(s_i)$ is the j^{th} autocorrelation coefficient of the energy at inverse temperature s_i [2, p. 448]. Since the sequence of measurements is highly correlated with $r_1(s_i)$ typically close to 0.99 and $r_j(s_i) \approx [r_1(s_i)]^j$, the resulting underestimation can be very large. Therefore, we need to replace u_j in (9) by a better estimate of $\mu(s_j)$ in order to get good quality "measurements" of $\sigma^2(s_i)$.

To estimate $\mu(s_j)$, we can use the rational model (6) for $\mu(s)$ and find the coefficients a_i and b_i such that the weighted squared error

$$\sum_{k=0}^l w_k (u_{k\tau} - \mu(s_{k\tau}))^2, \quad l = 1, 2, 3, \dots, \quad (10)$$

is minimized. The weights increase with k , to give more importance to the most recent data and thus track the coefficients as they vary. The model with the coefficients estimated at time $l\tau$ is used to predict $\mu(s_j)$ for the following τ steps. Denoting that estimate $\hat{\mu}(s_j)$, the measured variance of the energy, v_i^2 , is obtained using the expression

$$v_i^2 = \frac{1}{\tau} \sum_{j=i-\tau+1}^i (x_j - \hat{\mu}(s_j))^2. \quad (11)$$

From the set of measurements $\{v_{k\tau}^2, k \leq l\}$, $\sigma^2(s_i)$ may be predicted for the τ steps after step $l\tau$ by finding the coefficients of the rational model for $\sigma^2(s)$ via the same weighted least-squares method as for $\mu(s)$ (except that the rate of increase of the weights could differ).

The weighted least-squares problems for the coefficients in the models of $\mu(s)$ and $\sigma^2(s)$ are *non-linear*. Methods developed for these problems, such as Levenberg-Marquard's [16, pp. 526-528], are computationally intensive and converge to a local minimum of the squared-error function but not necessarily to the global minimum. Since experiments on a variety of problems indicate that selecting $\bar{M}=0$ results in a satisfactory model for $P(x)e^{-sx}$ [13], we shall use the rational models with $\bar{M}=0$; this enables us, after a slight modification to the squared-error criterion, to find the model coefficients by solving a *linear* least-squares problem. More precisely, if $\bar{M}=0$, model (6) becomes

$$\mu(s) = x_0 + \frac{1}{b_0(s_n) + b_1(s_n)s}, \quad (12)$$

to be used for the τ steps after $n = l\tau$. To simplify the estimation task, we approximate this model by the two-parameter model

$$\mu(s) = \frac{1}{A(s_n)s + B(s_n)}, \quad (13)$$

where, letting $c(s_n) = (1 + b_0(s_n)x_0 + b_1(s_n)x_0s_n)^{-1}$,

$$A(s_n) = c(s_n)b_1(s_n)[1 - c(s_n)x_0] \text{ and } B(s_n) = c(s_n)[b_0(s_n) + c(s_n)b_1(s_n)x_0s_n].$$

(This approximation is valid if $|c(s_n)b_1(s_n)x_0(s - s_n)| \ll 1$.) Because the expressions for $A(s_n)$ and $B(s_n)$ contain factors where s_n appears as a multiplier, these parameters will tend to vary faster than $b_0(s_n)$ and $b_1(s_n)$. This is a small price to pay since we avoid estimating x_0 and, furthermore, we can estimate the parameters by solving a linear least-squares problem: minimize the weighted squared error on $1 / \mu(s)$,

$$\sum_{k=0}^l w_k \left(\frac{1}{u_{k\tau}} - \frac{1}{\mu(s_{k\tau})} \right)^2, \quad l = 1, 2, 3, \dots \quad (14)$$

With $\bar{M}=0$ model (7) for $\sigma^2(s)$ reduces to the form

$$\sigma^2(s) = \left(\frac{1}{D(s_n)s + E(s_n)} \right)^2,$$

yielding the model for $\sigma(s)$, applicable for the τ steps after $n = l\tau$,

$$\sigma(s) = \frac{1}{D(s_n)s + E(s_n)}, \quad (15)$$

where $D(s_n)$ and $E(s_n)$ vary slowly (more slowly than $A(s_n)$ and $B(s_n)$). Since this expression has the same form as the expression in (13), the parameters $D(s_n)$ and $E(s_n)$ will be found by solving the linear least-squares problem: minimize

$$\sum_{k=0}^l w'_k \left(\frac{1}{v_{k\tau}} - \frac{1}{\sigma(s_{k\tau})} \right)^2, \quad l = 1, 2, 3, \dots \quad (16)$$

Straightforward manipulations yield expressions for the estimates of $\mu(s)$ and $\sigma(s)$ that minimize the weighted squared errors in (14) and (16), for $n < i \leq n + \tau$ with $n = l\tau$,

$$\hat{\mu}(s_i) = \frac{1}{A(s_n)s + B(s_n)}$$

and

$$\hat{\sigma}(s_i) = \frac{1}{D(s_n)s + E(s_n)}$$

with

$$A(s_n) = \frac{\sum_{k=0}^l w_k \sum_{k=0}^l w_k \frac{s_{k\tau}}{u_{k\tau}} - \sum_{k=0}^l w_k s_{k\tau} \sum_{k=0}^l w_k \frac{1}{u_{k\tau}}}{\sum_{k=0}^l w_k \sum_{k=0}^l w_k s_{k\tau}^2 - \sum_{k=0}^l w_k s_{k\tau} \sum_{k=0}^l w_k s_{k\tau}}, \quad B(s_n) = \frac{\sum_{k=0}^l w_k \frac{1}{u_{k\tau}} - A(s_n) \sum_{k=0}^l w_k s_{k\tau}}{\sum_{k=0}^l w_k},$$

$$D(s_n) = \frac{\sum_{k=0}^l w'_k \sum_{k=0}^l w'_k \frac{s_{k\tau}}{v_{k\tau}} - \sum_{k=0}^l w'_k s_{k\tau} \sum_{k=0}^l w'_k \frac{1}{v_{k\tau}}}{\sum_{k=0}^l w'_k \sum_{k=0}^l w'_k s_{k\tau}^2 - \sum_{k=0}^l w'_k s_{k\tau} \sum_{k=0}^l w'_k s_{k\tau}}, \quad E(s_n) = \frac{\sum_{k=0}^l w'_k \frac{1}{v_{k\tau}} - D(s_n) \sum_{k=0}^l w'_k s_{k\tau}}{\sum_{k=0}^l w'_k}.$$

As is usually done in adaptive filtering [4], we employ exponential weights $w_k = \alpha^{l-k}$ and $w'_k = \beta^{l-k}$, where α and β are positive constants smaller than 1, determining the effective

memory of the parameter estimation algorithm. The smaller the values of α and β , the faster the estimators adapt, but also the larger the variance of the estimates becomes. The computational formulas for the efficient λ -schedule are summarized in Table 1 in Section 2.3. A description of how to use the formulas is also given in that section. The estimators have been tested by applying the efficient λ -schedule in Table 1 to a highly correlated sequence of random numbers; with the proper choice of α and β the relative estimation error on $\sigma(s)$ does not exceed 10% [13].

2.2. Move generation control

Now we turn to the problems of design and control of move generation strategies. The techniques we present work well in practice, especially when paired with the efficient λ -schedule, but are by no means the only possible ones.

To propose a move in simulated annealing is to perturb the current solution to the optimization problem to obtain a new one. This perturbation is highly problem specific; different optimization problems have different methods for obtaining new solutions. For example, in a traveling salesman problem one may modify sections of a tour to obtain a new tour, while in a graph partition problem one may move a vertex from one set to another. In order to apply the move generation control method described in this section, we classify the move generation strategies into *controllable* and *non-controllable* strategies. A controllable move generation strategy is one for which there exists a parameter θ that increases with the *average magnitude of the proposed energy change*. For example, in our implementation of the N -city traveling salesman problem, each city maintains a list of the neighboring cities sorted in ascending order according to their distances from that city. A move is proposed by picking two cities A and B , and modifying the tour as shown in Fig. 1. City A is always picked randomly with equal probability from the N cities while city B is picked from the sorted list of neighbors of city A . Since in general the *average magnitude of the proposed energy change increases with the distance between cities A and B* , we choose the index value of the sorted list from which city B is picked as the parameter θ . Thus, the bigger is the index, the further apart is city B from city A , and, most likely, the larger is the average magnitude of the proposed energy change. By increasing (or decreasing) θ , we increase (or decrease) the average magnitude of the proposed energy change and, therefore, decrease (or increase) the acceptance ratio ρ_0 .

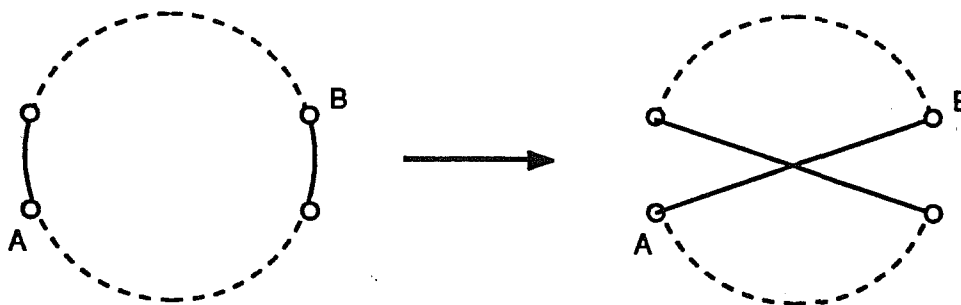


Figure 1: How a tour is modified in the traveling salesman problem.

Controllable move generation strategies are in general preferable to non-controllable move generation strategies, for ρ_0 can be regulated to decrease the execution time of simulated annealing while maintaining result quality. However, for some optimization problems, it is very hard to find a controllable move generation strategy. In those cases, we have to settle with no control on the average magnitude of the proposed energy change and, hence, the acceptance ratio.

Once a controllable move generation strategy has been selected, the problem of move generation control becomes the problem of θ control. Since the goal of move generation control is to achieve a desired acceptance ratio of 0.44, we would like to control θ in order to achieve that goal. Let the parameter $\bar{\theta}$ be picked according to the formula

$$\theta = -\bar{\theta} \log(\xi),$$

where ξ is a random number uniformly distributed between 0 and 1. This function is chosen because of its simplicity and the fact that any positive θ has a non-zero probability of getting picked. Using this formula, the probability that a given θ will be picked is specified by the exponential density function with an average value of $\bar{\theta}$,

$$p(\theta) = \frac{1}{\bar{\theta}} e^{-\frac{\theta}{\bar{\theta}}}. \quad (17)$$

(In practice there is an upper limit θ_{\max} for θ . If the generated θ exceeds θ_{\max} , a situation occurring mostly when $\bar{\theta}$ is large, θ is set to $\xi * \theta_{\max}$ and thus is drawn according to a uniform distribution, consistent with the fact that $p(\theta)$ is almost uniform.)

Since the acceptance ratio should be kept close to 0.44, comparison of the acceptance ratio measured for the last τ steps,

$$\hat{\rho}_0(s_n) = \frac{\text{number of accepted moves in the last } \tau \text{ steps}}{\tau}, \quad (18)$$

with the desired acceptance ratio indicates how $\bar{\theta}$ should be modified. If $\hat{\rho}_0(s_n)$ is less than .44, we decrease $\bar{\theta}$ and, hence, the average magnitude of the proposed energy change, which should lead to an increase in the acceptance ratio. Likewise, if $\hat{\rho}_0(s_n)$ is greater than .44, we increase $\bar{\theta}$. We use a simple proportional feedback controller for $\bar{\theta}$ in order to regulate the acceptance ratio about 0.44

$$\bar{\theta}_n = \bar{\theta}_{n-\tau} + K(\hat{\rho}_0(s_n) - 0.44),$$

where the positive gain K , which is problem as well as implementation dependent, controls the rate at which $\bar{\theta}$ could be changed. Since $\bar{\theta}$ cannot be negative because of the exponential density function in (17), we refine that control law to

$$\bar{\theta}_n = \max(\bar{\theta}_{n-\tau} + K(\hat{\rho}_0(s_n) - 0.44), \bar{\theta}_{\min}),$$

where $\bar{\theta}_{\min}$ is a positive constant. Care must be taken in choosing the gain K . Too large a gain makes the acceptance ratio oscillate around 0.44 while too small a gain makes its evolution sluggish. The proper gain is found experimentally for each optimization problem.

2.3. Application procedure and practical issues

We have discussed the parameter estimation and move generation control problems separately. Now, we give a complete picture of the interaction among the estimators, the

controller and the rest of the system and cover the practical issues associated with the use of the efficient λ -schedule. To use the schedule in Table 1, we first initialize the inverse temperature, s_0 , to 0 and $\bar{\theta}$ to its maximum value, $\bar{\theta}_0 = \bar{\theta}_{\max}$. The system is allowed to run at this inverse temperature until it is sufficiently randomized and initial measurements of the average and the standard deviation of the energy can be computed using the formulas

$$u_0 = \frac{1}{m} \sum_{i=1}^m x_i$$

and

$$v_0 = \left[\frac{1}{m} \sum_{i=1}^m (x_i - u_0)^2 \right]^{1/2},$$

where m is the number of steps executed at s_0 . These measurements are used to compute the initial values of the parameters A , B , D , and E using the parameter initialization formulas in Table 1. (The formulas are also applicable if one desires to start at a lower temperature, $s_0 > 0$. They are obtained under the assumption that the energy density function is a gamma density function and the global minimum is $x_0 = 0$ [14]; if a better lower bound \hat{x}_0 were available, u_0 should be replaced by $u_0 - \hat{x}_0$ in the formulas.) Then, after every step, the new inverse temperature is computed using the temperature updating formulas while after every τ steps, the parameters A , B , D , and E are adjusted using the parameter updating formulas, $\hat{\rho}_0$ is recomputed using the measurement formula, and $\bar{\theta}_n$ is updated using the move generation control formula. This process is repeated until the measured average energy, u_n , remains unchanged for the last f τ steps, where f is a small integer. At this point, the system is considered frozen and annealing is stopped. (We have also experimented with the more elaborate frozen detection schemes proposed in [1] and [8]. Although these two schemes are logically sound, they do not lead to a significant difference in performance.) For an implementation of the efficient λ -schedule in the C programming language, the reader is referred to [13].

The inverse temperature $s_1 = 1 / (2\sigma(0))$ in Table 1 is chosen such that it is high enough to ensure a smooth transition from infinite temperature to finite temperature. Since the efficient λ -schedule formula (3) is valid for $s \geq 7 / \sigma(0)$ [14], the system undergoes sub-optimal cooling within the range

$$\frac{1}{2\sigma(0)} \leq s \leq \frac{7}{\sigma(0)}.$$

This is tolerable since the system spends only a very small percentage of its time within this range. Also note that m is selected such that the total number of steps before reaching $s = 7 / \sigma(0)$ is close to what can be expected from application of (2) with typical values of $\rho_2(s)$, hence the proper level of initial randomization is attained.

The weight factors α and β in the parameter updating formulas are computed as $\alpha = 1 - \tau/L_a$, and $\beta = 1 - \tau/L_b$, where L_a and L_b are the so-called ‘‘memory lengths’’ of the estimators for the average and the standard deviation of the energy, respectively [4, Chapter 3]. Smaller memory lengths give smaller weight factors which, in turn, enable the estimators to track faster varying parameters.

Occasionally, instead of starting at zero inverse temperature, we may want to start at a higher inverse temperature. This is accomplished easily by setting s_0 to any desired inverse

Parameter initialization formulas	
$A(s_0) = \frac{v_0^2}{u_0^2}$	$D(s_0) = \frac{v_0}{u_0}$
$B(s_0) = \frac{1}{u_0} - A(s_0)s_0$	$E(s_0) = \frac{1}{v_0} - D(s_0)s_0$
Temperature updating formulas $n < i \leq n + \tau$	
$s_1 = \frac{1}{2\hat{\sigma}(s_0)}$	$s_{i+1} = s_i + \lambda \frac{4\hat{\rho}_0(s_n)(1 - \hat{\rho}_0(s_n))^2}{s_i^2(2 - \hat{\rho}_0(s_n))^2\hat{\sigma}^3(s_i)}$
$\hat{\mu}(s_i) = \frac{1}{A(s_n)s_i + B(s_n)}$	$\hat{\sigma}(s_i) = \frac{1}{D(s_n)s_i + E(s_n)}$
Parameter updating formulas $n = l\tau, l = 1, 2, \dots$	
$A(s_n) = \frac{f(1)f(\frac{s}{u}) - f(s)f(\frac{1}{u})}{f(1)f(s^2) - f(s)f(s)}$	$D(s_n) = \frac{g(1)g(\frac{s}{v}) - g(s)g(\frac{1}{v})}{g(1)g(s^2) - g(s)g(s)}$
$B(s_n) = \frac{f(\frac{1}{u}) - A(s_n)f(s)}{f(1)}$	$E(s_n) = \frac{g(\frac{1}{v}) - D(s_n)g(s)}{g(1)}$
$f(z) = \sum_{k=0}^l z_{k\tau} \alpha^{l-k}$	$g(z) = \sum_{k=0}^l z_{k\tau} \beta^{l-k}$
Measurement formulas $n = l\tau, l = 1, 2, \dots$	
$u_n = \frac{1}{\tau} \sum_{i=n-\tau+1}^n x_i$	$v_n = \left[\frac{1}{\tau} \sum_{i=n-\tau+1}^n [x_i - \hat{\mu}(s_i)]^2 \right]^{1/2}$
$\hat{\rho}_0(s_n) = \frac{\text{number of accepted moves in the last } \tau \text{ steps}}{\tau}$	
Move generation control formula $n = l\tau, l = 1, 2, \dots$	
$\bar{\theta}_n = \max(\bar{\theta}_{n-\tau} + K(\hat{\rho}_0(s_n) - 0.44), \bar{\theta}_{\min})$	

Table 1: Computational formulas for the efficient λ -schedule.

temperature instead of 0. Alternatively, we may want to specify the initial acceptance ratio rather than the initial inverse temperature and compute the inverse temperature for that ratio as described in [1, p. 212]. Yet another way is to measure the effective inverse temperature of the current solution and use that as the initial inverse temperature, as described in [17].

As noted earlier, the user specified parameter λ realizes the trade-off between the quality of the final solution and the computation time. Ideally, we would like to have a procedure that computes the proper λ for either a given quality of final solution or an allotted computation time. Unfortunately, such a procedure cannot be found analytically. Since λ controls the quasi-stationarity criterion at different inverse temperatures, its effect on the quality of the final solution and the computation time is indirect. Nevertheless, we observe the following empirical relation:

$$N_m = \frac{aN^b}{\lambda^c}, \quad (19)$$

where N is the size of a problem, N_m is the total number of proposed moves, and a , b and c are problem dependent constants. We suggest the following method to find these constants. First, experiment with different values of λ and N and measure the resulting total number of proposed moves in each case. Then, find the constants by solving the linear least-squares problem in $\log a$, b and c

$$\log N_m = \log a + b \log N - c \log \lambda.$$

With the empirical relation in (19), we can compute the proper λ for a given total number of proposed moves and, with a little extra effort, for an allotted computation time.

We observe from Table 1 that the efficient λ -schedule uses floating point computations heavily. To reduce the number of floating point operations, we approximate the efficient λ -schedule by computing the new inverse temperature after every ten steps instead of one,

$$s_{i+10} = s_i + 10\lambda \frac{4\rho_0(1-\rho_0)^2}{s_i^2(2-\rho_0)^2\sigma^3(s_i)}.$$

This speeds up the efficient λ -schedule by a few percent without degrading the quality of the final solutions.

We also pre-compute the values of the exponential function, which are needed in deciding whether a proposed move should be accepted or rejected, as in [9, pp. 39-40], and the values of the logarithm function, which are used to generate exponentially distributed random numbers. These values are stored in tables and looked up when needed. Since the evaluation of the exponential and the logarithm functions occurs quite often, a significant portion of computation time is typically saved.

3. Performance of the efficient annealing schedule

The goals of this section are to assess both the performance of the efficient λ -schedule, when compared with annealing schedules available in the literature, and the performance of simulated annealing as a general method for solving combinatorial optimization problems.

Among the general annealing schedules, Aarts's schedule [1] and Huang's schedule [8] stand out as schedules reported to give good average results, while the logarithmic schedule derived by Hajek [7] and others was shown to give solutions that converge to the global optima; see [14] for a short survey. To assess how well the efficient λ -schedule performs—where it stands among the competition, we compare it with the best of the three annealing schedules, selected on the basis of preliminary results on the traveling salesman problem. The test cases for the comparison cover two classical problems: the traveling salesman problem and the graph partition problem.

Most of the practical applications of simulated annealing have been in complicated problem domains, where other algorithms either did not exist or performed poorly. To assess the performance of simulated annealing as a general method for solving combinatorial optimization problems, we have to compare the method with efficient heuristics on well-studied problems. The traveling salesman and the graph partition problems are chosen for this reason; they have been studied extensively for two decades resulting in a number of well-known heuristics. Among these heuristics, we selected to compete with simulated annealing the ones by Lin and Kernighan [15],

and by Karp [10] for the traveling salesman problem, and the one by Fiduccia and Mattheyses [5] for the graph partition problem.

The test results given in this section were measured on a Sun 3/280-s8 with MC68881 floating point option. Unless stated otherwise, they are average results of *at least eight* executions, with the number of executions chosen to bring the standard deviations of the measurements down to an acceptable level. All programs were implemented in the C programming language.

3.1. Traveling salesman problem

The traveling salesman problem is a classical NP-complete combinatorial optimization problem [6]. The goal of this problem is to minimize the length of a tour starting from any city, visiting each of the N cities once and only once, and returning to the original place of departure. We restrict our attention to problems where the cities lie in the plane, and the distance is the Euclidean distance. We first cover the implementation details and then discuss the test results.

3.1.1. Implementation details

The cost to be minimized is the length of a tour. In this implementation, each city maintains a list of up to M neighboring cities sorted in ascending order according to their distances from that city. A move is proposed by first picking two cities A and B , then modifying the tour as shown in Fig. 1. City A is always picked randomly with equal probability from the N cities while city B is picked using one of the two following methods. In the first method called *standard control*, city B is picked randomly with equal probability from the first θ cities on the neighbor list of city A . The parameter θ , which roughly controls the maximum allowable distance between A and B , is lowered as a function of the inverse temperature, s , according to the empirical formula

$$\theta = 0.5N \left[\log_{10} \left(10 + \frac{1}{s\sqrt{N}} \right) - 1 \right]^2.$$

This method implements the widely held idea that the acceptance ratio should be allowed to drop almost linearly as a function of the total number of proposed moves [18] [8]. In the second method called *feedback control*, city B is the θ^{th} entry on the neighbor list of city A with

$$\theta = -\bar{\theta} \log(\xi),$$

where ξ is a random number uniformly distributed between 0 and 1, and $\bar{\theta}$ is a control parameter between 2 and N . Because ξ is uniformly distributed, θ will be exponentially distributed on $[0, \infty)$ with an average value of $\bar{\theta}$. If θ exceeds M , θ is set to ξN , to result in a uniform distribution. The control parameter $\bar{\theta}$ is adjusted after every τ moves according to the formula

$$\bar{\theta}_n = \max(\bar{\theta}_{n-\tau} + 100(\hat{\rho}_0(s_n) - 0.44), 2),$$

where $\hat{\rho}_0(s_n)$ is the measured acceptance ratio for the last τ moves. The value of $\bar{\theta}$ is lowered if $\hat{\rho}_0(s_n)$ is less than 0.44 and raised if $\hat{\rho}_0(s_n)$ is greater than 0.44. This updating procedure enables us to keep the acceptance ratio close to the *desired* acceptance ratio of 44% for most of the course of simulated annealing (one cannot completely avoid that $\rho_0(s)$ be close to 1 for high temperatures and to 0 for low temperatures).

Care must be taken in setting the lower limit for $\bar{\theta}$, $\bar{\theta}_{\min}$. We are tempted to let $\bar{\theta} \rightarrow 0$ as $s \rightarrow \infty$ so that the acceptance ratio can be kept at 44% for a longer period of time. However, if the value of $\bar{\theta}$ is too close to 0, the value of θ will be restricted to a small set. This restriction is undesirable since the number of possible moves is so small that simulated annealing may terminate prematurely.

For the sake of efficiency, the distances between any two cities are pre-computed and stored in a table. Their values are looked up when needed. This $O(N^2)$ storage requirement is usually not a problem for a small number of cities, but is prohibitive for a large number of cities. For problems in which the cities amount to thousands, the table lookup scheme cannot be used; we have to compute the distance every time we need it.

The settings of the λ -schedule parameters for the traveling salesman problem are $\lambda L_a = 600$, $\lambda L_b = 30,000$, $\tau = 100$, and, for the frozen detection parameter, $f = 5$. Note the fast rate of adaptation required to track the mean parameters, A and B , and the slow rate for the standard deviation parameters, D and E . Ideally, M , the number of cities in the neighbor list, should increase as N increases if we can afford the $O(N^2)$ storage requirement. But for $N = 10,000$, this $O(N^2)$ storage requirement is too high. We, therefore, set the value of M to the minimum of $N - 1$ and 250.

Huang's annealing schedule was implemented as in [8]. Due to the difference in move generation controls, a few values of Huang's maximum generation limit parameter have been tried. We settled on a value of $N\theta/2$, corresponding to the number of possible moves when standard control is used.

3.1.2. Test results

To assess the performance of the efficient λ -schedule, we need to compare it with the best general annealing schedule available in the literature. This best schedule is selected from the

Size	Tour Length (10^4)			Total Moves (10^5)		
	Huang	Aarts	Log.	Huang	Aarts	Log.
100	7.99	8.51	8.28	2.08	2.29	2.02
200	11.45	12.12	11.60	4.07	4.09	4.02
300	13.12	14.20	13.40	8.33	7.92	8.02
400	15.24	15.99	15.41	14.20	14.10	14.00

Table 2: Test results of the three annealing schedules on the traveling salesman problem with 100 to 400 uniformly distributed cities. Feedback control is used with all three annealing schedules.

Size	Speedup			
	$\delta = 3.6\%$	$\delta = 2.9\%$	$\delta = 2.2\%$	$\delta = 1.5\%$
100	2.09	3.39	6.00	8.35
200	2.54	4.09	7.50	10.61
300	2.90	4.38	10.86	24.20
400	3.37	7.77	17.61	>21.00

Table 3: Comparison between the efficient λ -schedule with feedback control and Huang's annealing schedule with standard control. The cities are uniformly distributed.

logarithmic schedule, the schedules by Huang *et al.*, and by Aarts and van Laarhoven, based on a preliminary experiment on the traveling salesman problem with 100 to 400 uniformly distributed cities—the coordinates of such a city are a pair of random numbers uniformly distributed in the interval of 0 to 10,000. To keep the competition fair, we picked one instance of the traveling salesman problem for each problem size and tested the different annealing schedules and heuristics on that same instance. From the test results shown in Table 2, we observe that Huang’s annealing schedule performs significantly better than the others; we, therefore, select Huang’s annealing schedule to compete with the efficient λ -schedule.

We compare the efficient λ -schedule and Huang’s annealing schedule on the traveling salesman problem with 100 to 400 uniformly distributed cities. The discrepancy in quality of the solutions from both schedules is kept within 0.2% so that the speedup can be measured. The experimental results are displayed in Table 3 in which δ , representing the average quality of the final solution, is the percentage of the average cost above the estimated best cost. This best cost was found by carrying out a sequence of careful annealing executions with the run-time doubled after every few executions until the average cost was stabilized.

The speedups associated with a particular δ as the number of cities increases are shown along each column in Table 3 while the speedups associated with a particular problem size as the cost improves are shown along each row. The speedup is defined as the ratio of the CPU time used by Huang’s annealing schedule with standard control to the CPU time used by the efficient λ -schedule with feedback control. Although the efficient λ -schedule is 10% slower per move than Huang’s, we still observe a speedup of up to 24 for the 300-city traveling salesman problem!

To isolate the effect of feedback control from the efficient λ -schedule, we experimented on both annealing schedules with both controls. The speedups, computed as the CPU time spent by the efficient λ -schedule with standard control over the CPU time spent by the same schedule with

Size	Speedup			
	$\delta = 3.6\%$	$\delta = 2.9\%$	$\delta = 2.2\%$	$\delta = 1.5\%$
100	1.74	3.27	3.43	3.87
200	1.37	1.75	2.23	2.76
300	1.94	1.76	3.00	5.00
400	1.37	1.87	2.82	4.90

Table 4: An experiment with move generation control. Comparison between the efficient λ -schedule with feedback control and the same schedule with standard control. The cities are uniformly distributed.

Size	Speedup			
	$\delta = 3.6\%$	$\delta = 2.9\%$	$\delta = 2.2\%$	$\delta = 1.5\%$
100	1.95	1.99	2.04	2.21
200	2.19	2.02	2.06	2.26
300	2.55	2.26	2.36	2.70
400	2.55	2.27	2.80	3.01

Table 5: Comparison between the efficient λ -schedule and Huang’s annealing schedule with feedback control used in both cases. The cities are uniformly distributed.

feedback control, are displayed in Table 4, while the speedups, computed as the CPU time spent by Huang's annealing schedule over the CPU time spent by the efficient λ -schedule when feedback control was employed in both cases, are displayed in Table 5. We observe that the use of feedback control speeds up the efficient λ -schedule by a factor of up to 5. However, even when the same control method is used with both annealing schedules, the efficient λ -schedule still outperforms Huang's by a factor of up to 3.

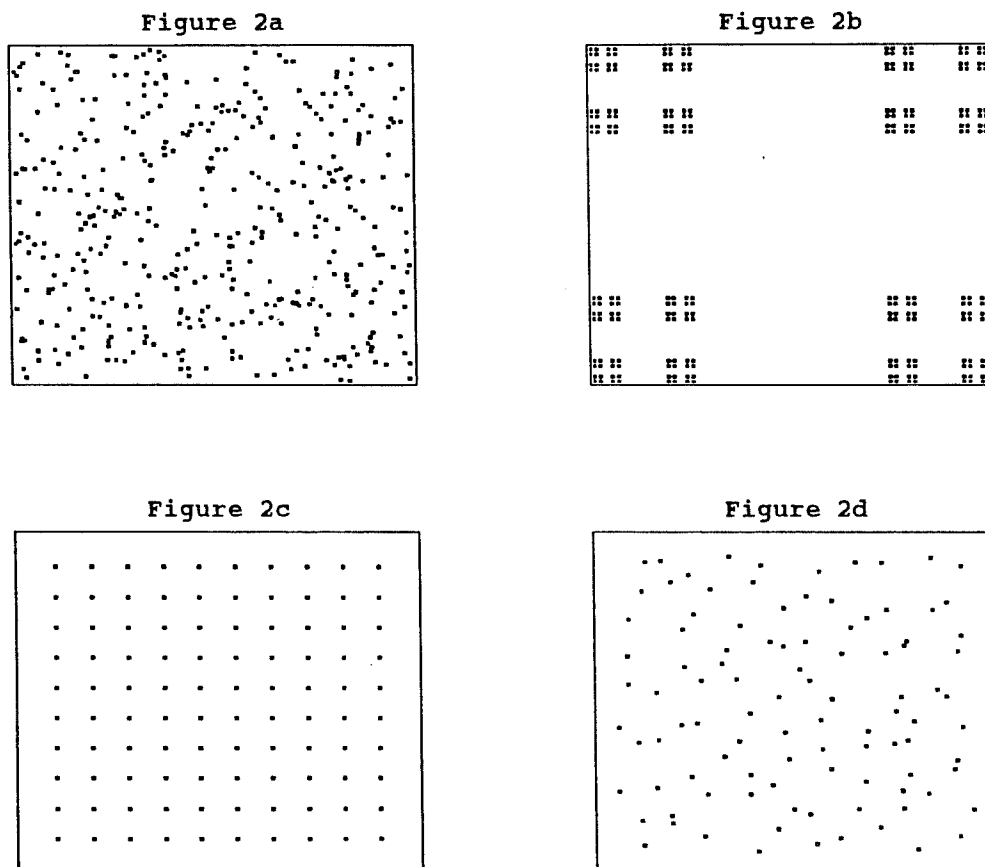


Figure 2: Examples of city distributions for the traveling salesman problem. Figure 2a: 400 uniformly distributed cities. Figure 2b: 256 super-clustered cities (clusters of four). Figure 2c: 100 cities on a lattice. Figure 2d: 100 cities on a lattice with perturbation (up to 50%).

We also compare simulated annealing, using the efficient λ -schedule and feedback control, with other heuristics for the traveling salesman problem. We chose the heuristics by Lin and Kernighan [15], and by Karp [10] as the competitors. The comparison with Lin and Kernighan's heuristic covers four types of city distributions: uniformly distributed cities, super-clustered cities, cities on a lattice, and cities on a lattice with small perturbation. In the case of uniformly distributed cities, an instance for each problem size was selected randomly as the test case. Examples of city distributions are shown in Fig. 2. All results tabulated are the average of *at least eight* executions. In the case of Lin and Kernighan's heuristic, each execution consists of

three iterations and within each iteration, the local optima found in the preceding iterations are used to speed up the search at the current iteration.

The speedups, defined as the ratio of the CPU time used by Lin and Kernighan's heuristic, t' , to the CPU time used by simulated annealing, t , are shown in Table 6 to Table 9. In Table 6 and Table 9, where the optimal costs are not known, δ' and δ are the percentages above the estimated best costs respectively for Lin and Kernighan's heuristic and for simulated annealing while in Table 7 and Table 8, where the optimal costs are known, δ' and δ are the percentages above the optimal costs.

From these tables, we observe that Lin and Kernighan's heuristic performs very poorly on super-clustered cities and does a very good job on cities on a lattice. This comes as no surprise since Lin and Kernighan's heuristic produces 3-optimal solutions—an n -optimal solution is a local optimum with respect to moves that exchange n sections of a tour instead of only two sections as in Fig. 1. In the case of super-clustered cities, the number of locally optimal solutions that are 2-optimal is large. Therefore, Lin and Kernighan's heuristic spends lots of time going from one 2-optimal solution to another until it reaches a 3-optimal solution. In the case of cities on a lattice, the number of globally optimal solutions is large: any tour formed by connecting cities either vertically or horizontally is a globally optimal solution. Consequently, the probability of finding one of these optimal solutions is high and the heuristic terminates quickly.

Size	CPU time (sec.)			Tour length	
	t'	t	t'/t	$\delta'(\%)$	$\delta(\%)$
100	69.8	52.9	1.32	1.01	1.01
200	296.0	101.6	2.91	1.25	1.21
300	686.1	197.5	3.47	1.32	1.32
400	1492.7	330.8	4.51	1.46	1.40

Table 6: Comparison with Lin and Kernighan's heuristic on the traveling salesman problem for uniformly distributed cities.

Size	CPU time (sec.)			Tour length	
	t'	t	t'/t	$\delta'(\%)$	$\delta(\%)$
64	57.9	9.3	6.23	0.58	0.58
128	428.2	17.4	24.61	1.13	1.06
256	2113.6	61.5	34.37	2.52	2.58
512	13883.1	296.3	46.85	4.11	4.03

Table 7: Comparison with Lin and Kernighan's heuristic on the traveling salesman problem for super-clustered cities.

Size	CPU time (sec.)			Tour length	
	t'	t	t'/t	$\delta'(\%)$	$\delta(\%)$
100	13.0	78.2	0.17	0.20	0.31
200	70.7	492.0	0.14	0.15	0.67
300	190.3	1338.4	0.14	0.10	1.09
400	415.6	3125.5	0.13	0.25	1.09

Table 8: Comparison with Lin and Kernighan's heuristic on the traveling salesman problem for cities on a lattice.

To check that the number of globally optimal solutions influences the speed of Lin and Kernighan's heuristic, we perturb the coordinates of the cities on a lattice by up to P percent of the normal inter-neighbor distance so that the number of globally optimal solutions is small. The new coordinates of a city (x', y') are computed from the old coordinates (x, y) according to

$$(x', y') = (x, y) + M_P(\xi_x, \xi_y)$$

where M_P is $P\%$ of the inter-neighbor distance and ξ_x, ξ_y are uniformly distributed random numbers between -1 and 1. The test results for a 200-city problem with different perturbation levels are shown in Table 9. We observe that the CPU time spent by Lin and Kernighan's heuristics increases from 71 seconds to 181 seconds with only a 5% perturbation, but does not increase further with larger perturbations. This observation confirms our conjecture that the abundance of globally optimal solutions speeds up Lin and Kernighan's heuristic. We also experimented with different numbers of cities on a lattice with up to 50% perturbation. As expected, the speedups shown in Table 10 fall in between those for the uniformly distributed cities and the cities on a lattice.

In contrast to Lin and Kernighan's heuristic, simulated annealing is relatively insensitive to the structure and regularity of an optimization problem; the CPU time spent by simulated annealing for different city distributions stays fairly constant for a given quality of solution. This combination of the insensitivity of simulated annealing towards the structure of a problem and the sensitivity of Lin and Kernighan's heuristic towards the number of global optima and the number of 2-optimal solutions explains the drastic speedup (up to 46) for problems with super-clustered cities as well as the superiority of Lin and Kernighan's heuristic over simulated annealing on problems with cities on a lattice.

As mentioned earlier, only one instance for each problem size was chosen for uniformly distributed cities to obtain the test results shown in Table 6. In order to check that similar

P	CPU time (sec.)			Tour length	
	t'	t	t' / t	$\delta'(\%)$	$\delta(\%)$
0%	70.7	492.0	0.14	0.15	0.67
5%	180.8	491.1	0.37	0.21	1.79
10%	179.2	491.1	0.36	0.33	2.27
20%	154.6	297.2	0.52	0.67	0.72
50%	182.3	205.6	0.89	0.73	0.70

Table 9: Comparison with Lin and Kernighan's heuristic on a 200-city traveling salesman problem for cities on a lattice with up to P percent perturbation.

Size	CPU time (sec.)			Tour length	
	t'	t	t' / t	$\delta'(\%)$	$\delta(\%)$
100	32.7	104.9	0.31	0.70	0.70
200	182.3	205.6	0.89	0.73	0.70
300	510.6	270.8	1.89	1.42	1.38
400	978.8	417.5	2.34	1.43	1.42

Table 10: Comparison with Lin and Kernighan's heuristic on the traveling salesman problem for cities on a lattice with up to 50% perturbation.

speedups can be expected for any random instance, we carried out the following experiment. We first randomly picked eight instances of a 200-city traveling salesman problem. Then, Lin and Kernighan’s heuristic was used to solve each instance once, resulting in a total of eight executions. Let us refer to these eight executions as a group. Since the CPU time spent by Lin and Kernighan’s heuristic is controlled by the number of iterations, we can vary the number of iterations to obtain different groups. Similarly, we can obtain different groups of executions for these eight instances using simulated annealing by controlling the value of λ and, hence, the CPU time. The speedups obtained are similar to those shown in Table 6 [13].

Except for the case of cities on a lattice, simulated annealing offers a substantial speedup over Lin and Kernighan’s heuristic for high quality solutions. This observation indicates that for problems where globally optimal solutions are sparse and high quality solutions are desired, simulated annealing coupled with a good move generation strategy is an efficient and highly competitive method.

Lin and Kernighan’s heuristic is only good for small problems since its computation time becomes unacceptable for large problems. Karp designed a heuristic that handles large problems. The comparison with Karp’s heuristic was performed on a traveling salesman problem with 10,000 cities whose coordinates are integers uniformly distributed in the interval of 0 to 1,000,000. Karp’s heuristic consists of three stages: partition a problem into M smaller size sub-problems, solve the sub-problems using another heuristic, and patch the resulting solutions to form a tour. In this experiment, we used *three* iterations of Lin and Kernighan’s heuristic to solve the partitioned sub-problems.

Two modifications are needed in order for simulated annealing to run on a 10,000-city traveling salesman problem. Ideally, we would like to let $M = 9,999$ and perform table lookup for the distances between cities. This is, however, impossible due to the $O(N^2)$ storage requirement. As a result, we compute the distances as the need arises and store the distances of only 250 nearest neighbors. The result of the comparison between simulated annealing and Karp’s heuristic is shown in Table 11 while the pictures of tours obtained via both methods are shown in Fig. 3. The estimated best cost for this problem is based on the formula [3]

$$\lim_{N \rightarrow \infty} \frac{C_{opt}}{\sqrt{NI^2}} = 0.749, \tag{20}$$

where C_{opt} is the optimal tour length and $I = 1,000,000$ is the interval from which the

M	CPU time (sec.)			Tour length	
	t'	t	t' / t	$\delta'(\%)$	$\delta(\%)$
128	4850	11050	0.44	6.98	6.66
64	10330	16070	0.64	3.52	3.41
32	26460	21610	1.22	2.03	1.97

Table 11: Comparison with Karp’s heuristic coupled with Lin and Kernighan’s heuristic on the traveling salesman problem with 10,000 uniformly distributed cities. In this table, M is the number of sub-problems in the partition, t' and t represent the CPU time for Karp’s heuristic and simulated annealing, respectively, while δ' and δ represent the percentages above the best cost estimated from formula (20). These are the average results of *four* executions.

coordinates of the cities are picked. Table 11 is constructed by varying the number of partitions used in Karp's heuristic and comparing the resulting solutions with simulated annealing for different values of λ .

Figure 3a

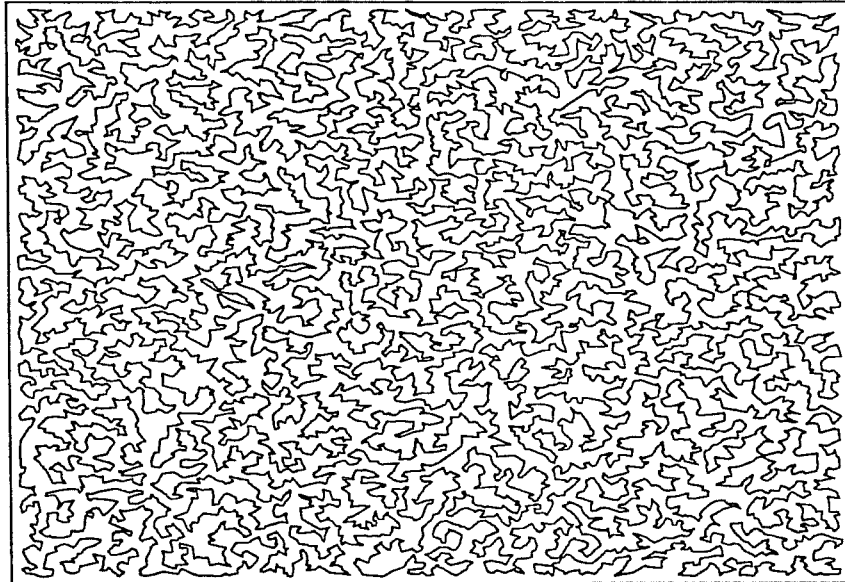


Figure 3b

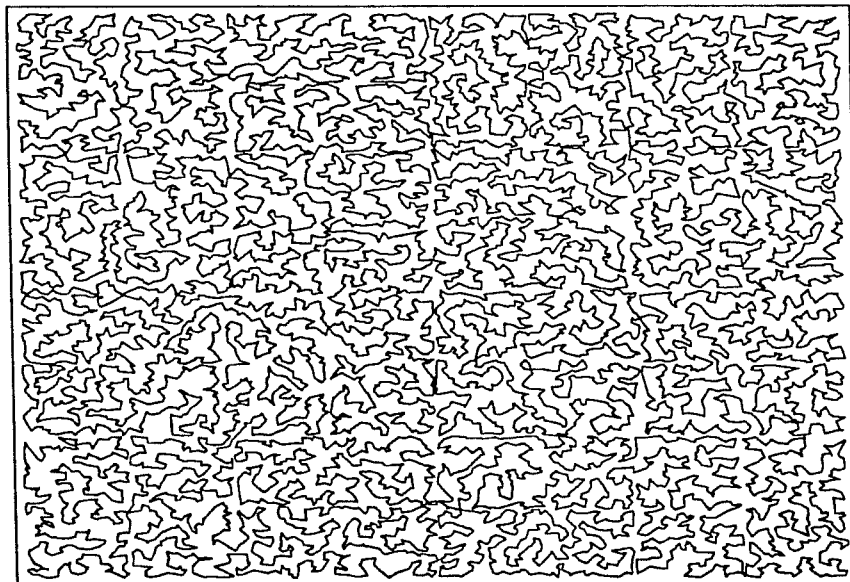


Figure 3: Tours of a 10,000-city traveling salesman problem. Figure 3a: Tour found using simulated annealing, with length 7.59×10^7 . Figure 3b: Tour found using Karp's heuristic, with length 7.64×10^7 ; the edges of the 64 squares in the partition can be made out from the tour.

From this table we observe that simulated annealing out-performs Karp's heuristic for high quality solutions. This evidence strongly supports our belief that an efficient annealing schedule coupled with a good move generation strategy makes simulated annealing an extremely effective method.

3.2. Graph partition problem

The graph partition problem (or graph partitioning) is NP-complete [6]. The goal of this problem is to partition the vertices of a graph into two equal size vertex sets, V_A and V_B , so as to minimize the number of edges with end-points in both sets. We first describe the implementation details of the move generation strategy for simulated annealing, followed by a discussion of the test results. Our results are similar to those obtained by Johnson *et al.* [9] with one essential difference: by using the efficient λ -schedule and a good move generation strategy, we are able to speed up simulated annealing significantly. Thus, for the same high quality solution, the CPU time spent by simulated annealing compares favorably with the CPU time spent by multiple executions of Fiduccia and Mattheyses's heuristic [5]. The first part of the discussion covers the comparison between the efficient λ -schedule and Huang's annealing schedule while the second part covers the comparison between simulated annealing and Fiduccia and Mattheyses's heuristic.

3.2.1. Implementation details

The cost to be minimized in a graph partition problem is the size of the cutset—the number of edges with end-points in both vertex sets. In order to increase flexibility in move generation, we follow the suggestion of Johnson *et al.* and introduce an additional imbalance factor into the cost function:

$$Cost = cutsize + \xi(|V_A| - |V_B|)^2,$$

where *cutsize* is the size of the cutset, ξ is the imbalance factor, and $|V_A|$ and $|V_B|$ are, respectively, the number of vertices in sets V_A and V_B . Ideally, we would allow $|V_A|$ to differ from $|V_B|$ in order to get more potential moves and, thus, increase the chance of climbing out from a local optimum. However, if this size difference is not controlled, we may end up with a cutsize of zero but with all vertices in one set. The introduction of the imbalance factor serves two purposes. The first one is to control the difference in size (the number of vertices) between V_A and V_B . A big difference in size is unlikely to occur at low temperature because moves that reduce this difference are likely to be accepted while moves that increase this difference are likely to be rejected. The second one is to increase flexibility in move generation by allowing moves that result in a difference between $|V_A|$ and $|V_B|$. This increase in flexibility increases the number of available moves and, hence, the chance of climbing out from a local optimum. Since these two purposes are conflicting, we must choose the imbalance factor carefully: too small a value will unduly favor the first purpose while too large a value will unduly favor the second purpose. In our experiments, $\xi = 0.005$ is used for problems in which the average edge degree is low while $\xi = 0.02$ is used for problems in which the average edge degree is high.

A move is proposed in two steps involving two vertices. First, a vertex is picked from an array of double link lists and moved to the other vertex set. Then, a second vertex is picked independently from the array and moved to the other vertex set. Since the two vertices are

picked independently, we may end up moving two vertices from the same set to the other, a perfectly acceptable operation. The reason for moving two vertices instead of one is as follows. The desired acceptance ratio of 44% is difficult to achieve at high temperature because almost all proposed moves are accepted at this temperature. Thus, we would like to move a larger number of vertices at high temperature since moving a larger number of vertices is likely to result in a larger proposed energy change and, hence, a smaller acceptance ratio. As the temperature is lowered, the acceptance ratio decreases. When it is below 44%, we would like to move a smaller number of vertices to obtain a smaller proposed energy change and, hence, a larger acceptance ratio. However, moving a constant number of vertices instead of a variable number has the advantage of reducing the program complexity. Furthermore, although moving a large number of vertices is desirable at high temperature, the computation time per move is unacceptable. We, therefore, achieve a compromise by moving two vertices per move.

In order to understand how move generation is controlled, we must first know how the double link lists are organized. Associated with each vertex, there is a variable called *gain* that keeps track of the change in cutsizes if this vertex is moved from its current vertex set to the other. All vertices with the same absolute gain are grouped to form a double link list. These lists can then be indexed by an array in which the n^{th} element of the array points to the double link list of vertices with absolute gain n . We shall call *head vertex* of the list the vertex to which the array points.

Move generation is controlled using either of the two following methods. In the first method, *standard control*, the two vertices are picked randomly and independently with equal probability from the head vertices of the first θ double link lists. After the head vertex of a list is picked, it loses its head position to the next vertex in the list even if the proposed move is rejected. The parameter θ is lowered as a function of the inverse temperature, s , according to the empirical formula

$$\theta = M \left[\log_{10} \left(10 + \frac{0.0005}{sC_i} \right) - 1 \right]^2,$$

where M is the maximum absolute gain possible and C_i is the initial cutsize. The objective of this method is to control the proposed energy change in such a way that the acceptance ratio drops almost linearly as a function of the total number of proposed moves. In the second method, *feedback control*, the two vertices are picked independently from the head vertices of two double link lists selected using the formula

$$\theta = -\bar{\theta} \log(\xi),$$

where ξ is a random number uniformly distributed between 0 and 1, $\bar{\theta}$ is a control parameter and θ denotes the θ^{th} list in the array. If θ exceeds M , θ is set instead to ξM . The control parameter $\bar{\theta}$ is adjusted after every τ moves according to the formula

$$\bar{\theta}_n = \max(\bar{\theta}_{n-\tau} + 5(\hat{\rho}_0(s_n) - 0.44), 1.5),$$

where $\hat{\rho}_0(s_n)$ is the measured acceptance ratio for the last τ moves. This procedure enables us to keep the acceptance ratio in the neighborhood of 44% for most of the course of the annealing.

The settings of the λ -schedule parameters for the graph partition problem are $\lambda L_a = 400$, $\lambda L_b = 20,000$, $\tau = 100$, and $f = 5$.

Huang's annealing schedule was implemented as in [8]. Due to the difference in move generation controls, a few maximum generation limits have been tried. We settled on a value of $0.04N^2$, where $N = |V_A| + |V_B|$ is the number of vertices in the graph.

In the implementation of Fiduccia and Mattheyses's heuristic, we considered two procedures in which the size of the two vertex sets was allowed to differ either by at most 1 or by at most $0.1N$. This size difference is eliminated at the end of an execution by moving vertices that lead to the smallest increase in cutsize from the larger vertex set to the smaller one. Preliminary experimentation did not indicate any significant difference between the final results of the two procedures. We, therefore, restrict the maximum size difference of the two vertex sets to 1.

3.2.2. Test results

To assess the performance of the efficient λ -schedule, we compare it with Huang's annealing schedule on the graph partition problem. The experiment was conducted on *random graphs* with $N = 500$ or $N = 1,000$ vertices and an average edge degree, d , of 5 or 20. These graphs are random in the sense that the probability that any pair of vertices constitutes an edge is given by $d / (N - 1)$. Since for any vertex, there are $N - 1$ such pairs, the average edge degree is d . Four instances of random graphs are chosen in the test, with the following N and d values: $N = 500, d = 5$; $N = 500, d = 20$; $N = 1,000, d = 5$; $N = 1,000, d = 20$. The experimental results are shown in Table 12 in which γ is the excess of the found cutsize, C , over the estimated best cutsize, C_0 , i.e., $\gamma = C - C_0$. This best cutsize was found by carrying out a sequence of careful annealing executions as described for the traveling salesman problem.

The speedups associated with a particular γ as the number of vertices or the average edge degree increases are shown along each column in Table 12 while the speedups associated with a

N	d	Speedup		
		$\gamma = 12$	$\gamma = 8$	$\gamma = 5$
500	5	3.2	4.6	4.2
1000	5	2.5	2.7	4.8
500	20	9.8	13.7	>100.0
1000	20	13.1		

Table 12: Comparison between the efficient λ -schedule with feedback control and Huang's annealing schedule with standard control on random graphs. The excess of the cutsize, C , over the estimated best cutsize, C_0 , is denoted by γ , i.e., $\gamma = C - C_0$.

N	d	CPU time (sec.)		Cutsizes	
		t'	t	γ'	γ
500	5	65.7	8.3	26	17
1000	5	371.4	24.4	61	41
500	20	87.3	22.6	43	29
1000	20	424.2	57.0	83	65

Table 13: Comparison between Kernighan and Lin's heuristic and the best of 10 executions of Fiduccia and Mattheyses's heuristic. The results from Kernighan and Lin's heuristic are denoted by t' and γ' , while the results from Fiduccia and Mattheyses's heuristic are denoted by t and γ .

particular problem size as the cutsize improves are shown along each row. The speedup is defined as the ratio of the CPU time used by Huang's annealing schedule with standard control to the CPU time used by the efficient λ -schedule with feedback control. We observe a speedup of up to 100 on a random graph with $N = 500$ and $d = 20$.

We also compare simulated annealing using the efficient λ -schedule and feedback control with another heuristic for the graph partition problem. Our choice of heuristics consists of Kernighan and Lin's heuristic [11], and multiple executions of Fiduccia and Mattheyses's heuristic (a sped up version of the Kernighan and Lin's heuristic). Since Fiduccia and Mattheyses's heuristic is very fast (linear time), and Kernighan and Lin's heuristic gives slightly better results than Fiduccia and Mattheyses's heuristic, we first explored whether the best of multiple executions of Fiduccia and Mattheyses's heuristic gives better results than Kernighan and Lin's heuristic. Based on the preliminary results in Table 13, we chose multiple executions of Fiduccia and Mattheyses's heuristic as the competitor. Three types of graphs were used in the comparison between simulated annealing and Fiduccia and Mattheyses's heuristic: random graph, geometric random graph, and hierarchical graph. The construction of random graphs has already been described. To construct a geometric random graph with N vertices and an average edge degree d as described in [9], we first independently pick N pairs of random numbers uniformly distributed in the interval of 0 to 1. These N pairs of numbers are the coordinates of the N vertices. Then, we add an edge between any two vertices whose distance apart is less than r with $r = \sqrt{d / N \pi}$. This formula for r is obtained by noting that all N vertices are contained in an area of 1. Hence, the average area per vertex is $1 / N$. To find an average of d vertices whose distances are less than r apart from a given vertex (i.e., in an area of πr^2), we let $r = \sqrt{d / N \pi}$. See Fig. 4a for the picture of a geometric random graph with $N = 500$ and $d = 5$. To construct a hierarchical graph with 4^k vertices where $k = 1, 2, \dots$, we apply the following algorithm.

1. let $i = k$.
2. partition the 4^i vertices into 4^{i-1} groups with four vertices in each group.
3. add four edges to each group to make a cycle.
4. select one vertex from each of these groups to obtain a total of 4^{i-1} vertices.
5. let $i = i - 1$ and repeat step 2 to step 5 on these 4^i vertices until the graph is connected.

A hierarchical graph with 64 vertices is depicted in Fig. 4b. It is easy to see that the minimum cutsize for these graphs is always 2.

The test results for the three types of graphs are displayed in Table 14 to Table 16 in which C_0 is the best cutsize found and γ' and γ are the cutsizes in excess of C_0 for Fiduccia and Mattheyses's heuristic, respectively. The speedup, t' / t , is defined as the ratio of the CPU time used by Fiduccia and Mattheyses's heuristic, t' , to the CPU time used by simulated annealing, t . All results tabulated for Fiduccia and Mattheyses's heuristic are the average results of *at least eight* groups, where the CPU time of a group is the cumulative CPU time for 100 executions and the cutsize of a group is the best cutsize found in those 100 executions. All results tabulated for simulated annealing are the average results of *at least eight* executions.

Figure 4a

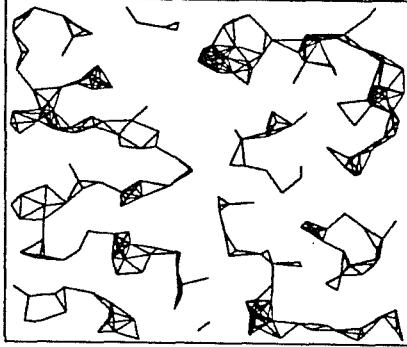


Figure 4b

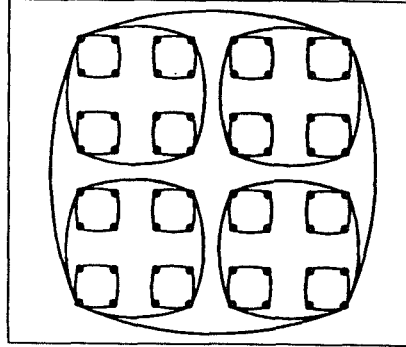


Figure 4a: A geometric random graph with 500 vertices and an average edge degree of 5. Figure 4b: A hierarchical graph with 64 vertices.

From Table 14 we observe a speedup of up to 5 for a random graph with 1,000 vertices and an average degree of 20. To better understand the relationship between the speedup and the quality of the final solution, we picked a random graph with $N = 500$ and $d = 5$ and varied the

N	d	CPU time (sec.)			Cutsizes		
		t'	t	t'/t	γ'	γ	C_0
500	5	64.7	41.5	1.56	10	10	247
1000	5	154.0	42.1	3.66	31	32	463
500	20	209.7	49.4	4.24	11	12	1706
1000	20	487.4	91.7	5.32	45	47	3374

Table 14: Comparison with Fiduccia and Mattheyses's heuristic on random graphs.

N	d	CPU time (sec.)			Cutsizes		
		t'	t	t'/t	γ'	γ	C_0
500	5	66.4	1050.1	0.06	6	11	5
1000	5	204.2	905.7	0.23	14	14	20
500	20	174.5	2823.5	0.06	0	49	100
1000	20	388.5	1315.3	0.30	2	369	181

Table 15: Comparison with Fiduccia and Mattheyses's heuristic on geometric random graphs.

N	CPU time (sec.)			Cutsizes	
	t'	t	t'/t	γ'	γ
256	20.1	248.3	0.08	0	0
1024	128.6	126.3	1.02	7	7
4096	488.7	161.4	3.03	43	43

Table 16: Comparison with Fiduccia and Mattheyses's heuristic on hierarchical graphs. The minimum cutsizes C_0 for hierarchical graphs is always 2.

execution time. The test results on such a graph with the number of executions within a group ranging from 10 to 2,000 are shown in Table 17. We observe a speedup of up to 15 for high quality solutions.

In contrast, simulated annealing performs relatively poorly on geometric random graphs. We observe from Table 15 that simulated annealing runs up to 17 times more slowly and gives worse results than Fiduccia and Mattheyses's heuristic. The superiority of Fiduccia and Mattheyses's heuristic over simulated annealing on geometric graphs may be related to the special structure of these graphs. As shown in Fig. 4a, all edges of a geometric random graph are between pairs of vertices that are close by. Simulated annealing, being fairly insensitive to structure, performs poorly on this type of graph when compared with Fiduccia and Mattheyses's heuristic, which takes advantage of this structure.

Another kind of graph for which simulated annealing does not perform well is hierarchical graphs. The test results for hierarchical graphs are displayed in Table 16. Although simulated annealing performs poorly on this kind of graph, Fiduccia and Mattheyses's heuristic does even worse. We observe a speedup of up to 3 on a hierarchical graph with 4,096 vertices. Furthermore, by comparing Table 16 with Table 18, we notice that the quality of the solutions obtainable by simulated annealing is much higher than that obtainable by Fiduccia and Mattheyses's heuristic if we increase the allotted CPU time. Both methods find the optimal cutsizes for $N = 256$. However, as the number of vertices increases, the problem becomes increasingly difficult. Except for specially designed heuristics that exploit this kind of structure, finding high quality solutions for hierarchical graphs is a very difficult problem because of the small optimal cutsizes.

Exec.	CPU time (sec.)			Cutsizes	
	t'	t	t'/t	γ'	γ
10	8.6	24.0	0.36	13	13
100	64.7	41.5	1.56	10	10
500	308.4	76.4	4.04	7	6
2000	1226.4	82.0	14.96	5	5

Table 17: Detailed comparison with Fiduccia and Mattheyses's heuristic on a random graph with 500 vertices and an average edge degree of 5. The best cutsizes found, C_0 , for this problem is 247.

N	CPU time (sec.)		Cutsizes	
	t'	t	γ'	γ
256	20.1	248.3	0	0
1024	2574.2	1745.7	5	1
4096	9890.5	2755.5	42	18

Table 18: The best average results obtained (with reasonable amount of CPU time) by simulated annealing and Fiduccia and Mattheyses's heuristic on hierarchical graphs with 256 to 4,096 vertices. The CPU time tabulated for Fiduccia and Mattheyses's heuristic is the cumulative CPU time of all the executions in a group. By varying the number of executions in a group, we vary the total CPU time. The minimum cutsizes is always 2.

4. Conclusion

We have presented an implementation for the efficient λ -schedule and we have shown that the schedule coupled with a good move generation strategy achieves substantial speedups when compared with schedules available in the literature. Since the efficient λ -schedule uses statistical quantities only, it is applicable to general combinatorial optimization problems. In addition, for problems that have little structure, simulated annealing has been shown to uncover high quality solutions faster than tailored heuristics for the traveling salesman and the graph partition problems.

Acknowledgements

This research was supported by the Army Research Office under contract DAAL03-86-K-0158, by the Office of Naval Research under contracts N00014-84-K-0092 and N00014-85-K-0461, and by the National Science Foundation under grant ECS-8314750.

References

- [1] E. Aarts and P. van Laarhoven, "Statistical Cooling Algorithm: A General Approach to Combinatorial Optimization Problems," *Philips J. of Res.*, Vol. 40, No. 4, 193-226, 1985.
- [2] T. Anderson, *The Statistical Analysis of Time Series*, John Wiley and Sons, Inc., 1971.
- [3] J. Beardwood, J. Halton, and J. Hammersley, "The Shortest Path Through Many Points," *Proc. Cambridge Phil. Soc.*, Vol. 55, 299-327, 1959.
- [4] C. Cowan, and P. Grant, *Adaptive Filters*, Prentice-Hall, 1985.
- [5] C. Fiduccia and R. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. 19th ACM/IEEE Design Automation Conf.*, 175-181, 1982.
- [6] M. Garey and D. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman and Company, 1979.
- [7] B. Hajek, "A Tutorial Survey of Theory and Applications of Simulated Annealing," *Proc. 24th IEEE Conf. on Decision and Control*, 755-760, 1985.
- [8] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," *Proc. IEEE Int. Conf. on CAD, ICCAD-86*, 381-384, 1986.
- [9] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon, "Optimization by Simulated Annealing: an Experimental Evaluation (Part I)," *Draft*, 1987.
- [10] R. Karp, "Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane," *Math. of Oper. Res.*, Vol. 2, No. 3, 209-224, 1977.
- [11] B. Kernighan, and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell. Sys. Tech. J.*, Vol. 49, 291-307, 1970.
- [12] S. Kirkpatrick, C.D. Gelatt Jr., and M. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, 671-680, 1983.
- [13] J. Lam, "An Efficient Simulated Annealing Schedule," Ph.D. Dissertation, Department of Computer Science, Yale University, 1988.

- [14] J. Lam, and J.-M. Delosme, "An Efficient Simulated Annealing Schedule: Derivation," Technical Report 8816, Department of Electrical Engineering, Yale University, 1988.
- [15] S. Lin and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Oper. Res.*, Vol. 21, 498-516, 1973.
- [16] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes*, Cambridge University Press, 1987.
- [17] J. Rose, W. Klebsch, and J. Wolf, "Equilibrium Detection and Temperature Measurement of Simulated Annealing Placement," *Proc. of the MCNC Int. Workshop on Placement and Routing*, 1988.
- [18] C. Sechen, and A. Sangiovanni-Vincentelli, "The TimberWolf Placement and Routing Package," *IEEE J. of Solid-State Circuits*, Vol. 20, No. 2, 510-522, 1985.