Chapter 2

New Simulated Annealing Algorithm

2.1 Introduction

Simulated annealing is a technique for finding an optimal or near-optimal solution for *combinatorial optimization* problems, or problems which have discrete variables. It was proposed by Kirkpatrick, Gelatt, and Vecchi in 1983 and has been successfully applied to circuit partitioning, placement, and routing in the physical design of integrated circuits.

The goal of a combinatorial optimization algorithm is to find the state of lowest cost (or energy) from a discrete space of admissible configurations *S*. For each problem, a cost function must be defined which maps each state to a real number denoting its cost. For many problems, the number of possible states grows exponentially with the size of the input. Optimizing becomes the process of *searching* for the state of lowest cost in a hyper-dimensional space. With a large number of possible states to visit, the brute force method of visiting all configurations becomes impractical. Clearly, we need a search strategy to uncover the lowest cost solution in the jungle of states.

For many problems, the states of the configuration space are related. A problem exhibits *optimal substructure* if an optimal solution to the problem contains within it optimal solutions to subproblems. These cases may be solved by either a *greedy* or a *dynamic programming* algorithm. In a *greedy-choice* problem, a globally optimal solution can be found by making a locally optimal (greedy) decision. The best choice is made at each moment; at each step, we solve the ramifications of the previous choice. The choice made by a greedy algorithm can not depend on future decisions or solutions to subproblems. In dynamic programming, a choice is made at each step which may depend on the solutions to the subproblems. Solving an optimal substructure problem will require a greedy strategy or dynamic programming depending on the nature of the problem [44].

Unfortunately, the physical design problems described in Chapter 1 do not exhibit optimal substructure; each step of physical design has been shown to be NP-complete. If we apply the greedy algorithm search strategy, we will usually get stuck in a local minimum. This means that

$$c(i) \geq c(j_{min}), \forall i \in S(j_{min}) \tag{2.1}$$

where j_{min} is the local minimum state, and $S(j_{min})$ is the set of states reachable from the state j_{min}. In many cases, there is a large disparity between the local minimum and the global minimum cost. We need a search strategy which avoids local minima and finds the global minimum. Simulated annealing is such a search strategy.

At the heart of the simulated annealing algorithm is the Metropolis Monte Carlo procedure which was introduced to provide an efficient simulation of a collection of atoms in equilibrium at a given temperature [151]. The Metropolis procedure is the inner loop of the simulated annealing algorithm as shown in Figure 2.1. While the greedy algorithm forbids changes of state that increase the cost function, the Metropolis procedure allows moves to states that increase the cost function. Kirkpatrick, et al. suggested that the Metropolis Monte Carlo method can be used to simulate the physical annealing process and to solve combinatorial optimization problems [112]. They suggested adding an outer loop which lowers the temperature from a high melting temperature in slow stages until the system *freezes*, and no further changes occur. At each temperature, the simulation must proceed long enough for the system to reach a steady state. The sequence of temperatures and the method to reach equilibrium at each temperature is known as an *annealing schedule*. They showed that this same technique can be applied to combinatorial optimization problems if a cost function is used in place of energy, and the temperature is used as a control parameter.

It has been shown that the simulated annealing algorithm, when started in an arbitrary state and given an appropriate annealing schedule, will eventually converge to a global optimum [153]. Although these results required an infinite amount of computation time for the convergence guarantee, in practice, simulated annealing has been extremely successful when applied to circuit partitioning and placement problems [195][196][198][200][201]. It has outperformed all other known algorithms.

The essential elements of the simulated annealing algorithm are summarized below in Figure 2.1. The algorithm consists of two loops: Each execution of the inner loop gener-

```
Algorithm simulated_annealing()
1   do
2       do
3           $j$ = generate($i$)
4           if accept($\Delta C$, $T$) then
5               $i = j$
6       until cost is in equilibrium
7       reduce($T$)
8   until cost cannot be reduced any further
```

Figure 2.1 The basic simulated annealing algorithm.

ates new configurations to be evaluated at constant temperature. The acceptance of a new configuration *j* depends on the current temperature *T* and the change in cost between the current configuration *i* and the proposed configuration *j* as presented in Figure 2.2. All configuration changes which do not increase the cost are accepted as in any iterative improvement algorithm, but moves with $\Delta C > 0$ are accepted depending on the value of ΔC and the value of *T*. The Boltzmann distribution ($\exp\left(\frac{-\Delta C}{T}\right)$) which governs physical annealing is used as the criteria for determining acceptance of states with increased cost. In this simple formulation of simulated annealing, we designate that the inner loop is repeated until the average value of the cost appears to have converged. As *T* is lowered from a high value, large uphill moves are mostly rejected. As *T* is lowered further, moves

```
Algorithm accept($\Delta C$, $T$)
1   if $\Delta C \le 0$ then /* new cost is less than or equal to the old cost */
2       return(ACCEPT) /* accept the new configuration */
3   else
4       randomly generate a number $r$ between 0 and 1
5       if $r < \exp\left(\frac{-\Delta C}{T}\right)$ then return(ACCEPT)
6       else return(REJECT)
```

Figure 2.2 The acceptance function for the simulated annealing algorithm.

with yet lower values of $\Delta C > 0$ become largely rejected. In some sense, critical decisions are made for those values of $\Delta C > 0$ which are on the order of the value of *T*. Hence, simulated annealing operates in a pseudo-hierarchical fashion with respect to $\Delta C > 0$ values as *T* is decreased.

2.2 Previous Work

Although the Metropolis method is simple, effective and easily programmed, it has a major drawback: at low temperatures, the running time is very long because many candidates for moves are rejected before each move to a different configuration. To remedy this inefficiency, various approaches have been proposed to speed the algorithm such as parallel implementations [1][6][26][65][122][178]. Lam studied the problem and proposed a statistical annealing schedule [123][124][125][126].

Lam's schedule is based on the observation that annealing is successful if the system is kept close to thermal equilibrium as the temperature is lowered. However, to keep the system in equilibrium at all times requires that the temperature decrements be infinitesimal; a *long* time would have passed before the system is frozen, and annealing is stopped. From a practical standpoint, a good annealing schedule must, therefore, achieve a compromise between the quality of the final solution and the computation time. To determine when the system is in equilibrium so that the temperature could be lowered, we need an equilibrium criterion [123]. A system is close to equilibrium at temperature T if the condition

$$\mu(s) - \lambda\sigma(s) \le \bar{c} \le \mu(s) + \lambda\sigma(s) \qquad (2.2)$$

is satisfied, where $\bar{c}$ is the average cost of the system, $s = 1 / T$ is the inverse temperature, and $\mu(s)$ and $\sigma(s)$ are the mean and standard deviation of the cost if the system were in thermal equilibrium at temperature T. The parameter λ, which can be made as small as desired to ensure a good approximation of equilibrium, realizes the compromise between the quality of the final solution and the computation time: the smaller the λ; the better the quality of the final solution; the longer the computation time.

Simulated annealing has been applied to the placement and routing problem in the TimberWolf system. Complete accounts of the implementations of simulated annealing for earlier versions of the TimberWolf placement programs have been published [195][196][198][200][201]. Improvements in the implementations of the cost function C, the generation of new configurations (function *generate* in line 3 of Figure 2.1), and the inclusion of the results of a theoretically derived statistical annealing schedule have been responsible for the very significant reduction in the CPU time required by TimberWolf. The next section presents the details of these algorithmic improvements.

2.3 New Annealing Schedule

We are now utilizing the results of a theoretically derived statistical annealing schedule developed by Lam [125] [126] in a recent Ph.D. dissertation. In his work, Lam showed that the optimum acceptance rate of proposed new configurations is approximately 44 percent. In Lam's algorithm, a *range limiter window* (first described in [199]) is used to keep the acceptance rate (denoted as α) as close as possible to 44 percent. (The *range limiter window* bounds the magnitude of the perturbation (or move distance) from the current state. The range limiter window size is designed to increase the acceptance rate at a given temperature. Changes in cost (ΔC) are on the order of the move distance. Therefore, reducing the move distance yields smaller ΔC values and hence an elevated acceptance rate.) In the beginning of the execution of this algorithm, the temperature T is set to a very high value (effectively infinity). Even with the range limiter dimensions encompassing the entire chip, the acceptance rate α approaches 100 percent. Since a further increase in range limiter dimensions cannot decrease α, there clearly must be a region of operation for the algorithm in which α is above the ideal value of 44 percent. Also, as T gets sufficiently low, the range limiter dimensions reduce to their minimum values. Then, as α drops below 44 percent, there is no way for it to return to a higher level. It is therefore apparent that there is a region of operation in which α falls from 44 percent toward zero as T approaches zero. The anticipated three regions of operation (α above 0.44, α equals 0.44, and α below 0.44) are illustrated in Figure 2.3.

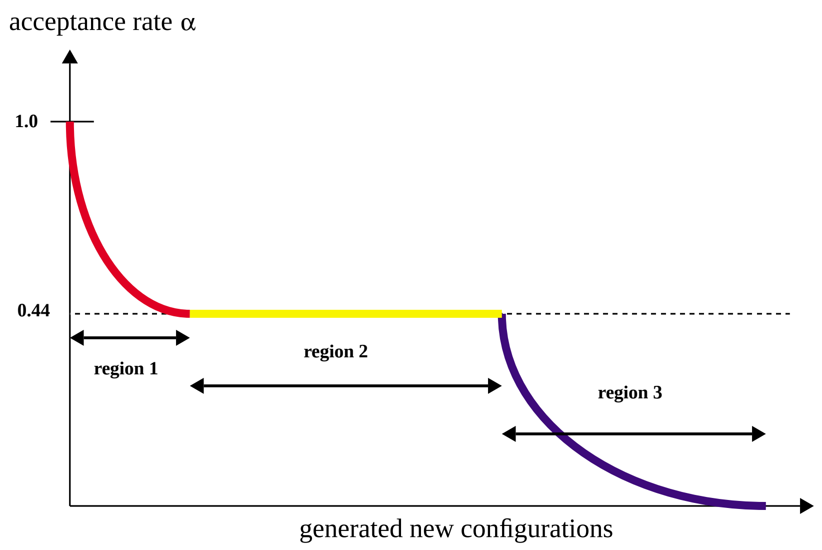

Figure 2.3 Anticipated plot of the acceptance rate versus generated new configurations.

One disadvantage of the schedule developed by Lam is its inability to accurately predict when the execution of the algorithm will end from the beginning of the run. That is, it is not known how many new configurations will be generated during the course of the execution of the algorithm. In an effort to gain a different perspective on Lam's theory, we measured α versus generated new configurations for executions on several industrial circuits. One objective was to determine the percentage of the run (that is, the percentage of the total new configurations generated) devoted to each of the three regions of operation. These percentages were remarkably similar for the very wide range of circuit sizes which were tested. A typical plot is shown in Figure 2.4.

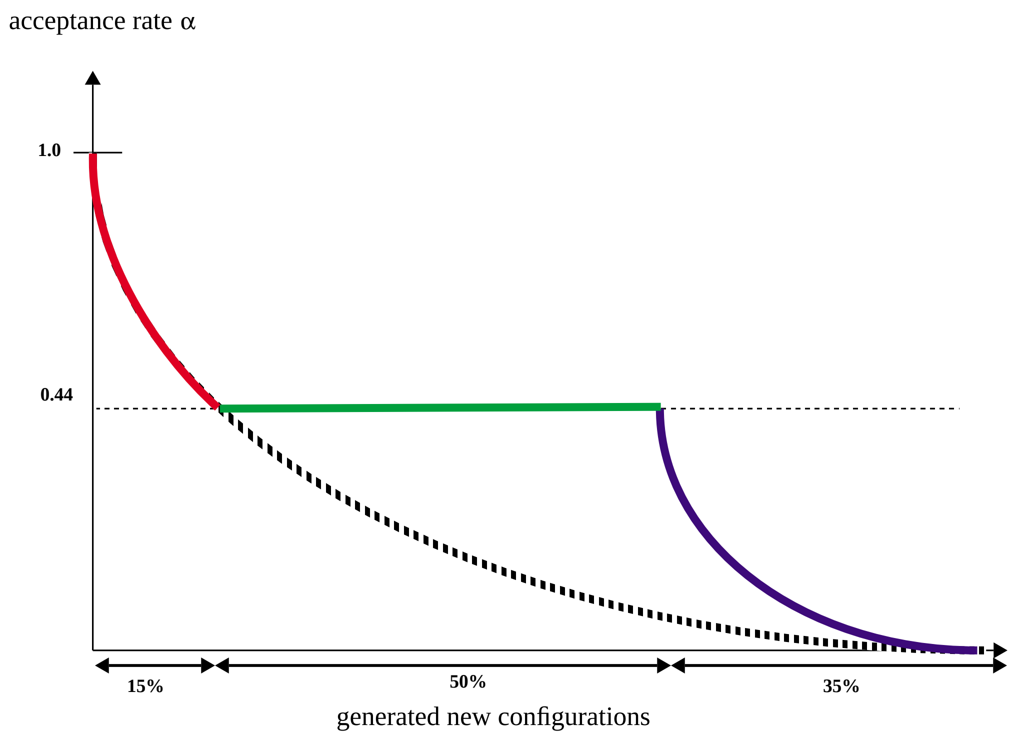

Figure 2.4 Typical measured acceptance rate versus generated new configurations as obtained from experiments conducted on several industrial circuits, showing the percentage of the run spent in each region of operation.

We discovered that for region 1 (which encompasses approximately 15% of the run) α versus generated new configurations could be modeled by an exponential function. This function has a peak value of 1.0, and passes through the point where α first reduces to 0.44. Furthermore, we found that region 3 could also be modeled by an exponential function with peak value 0.44 and minimum value 0.0. In region 2, the acceptance rate is flat, but we discovered that the decrease in the range limiter window dimensions as a function of generated new configurations can also be modeled by exponential functional form. That all three regions can be modeled by exponential functions is not surprising in light of the use of the (exponential) Boltzmann-like factor used to govern acceptance or rejection of new configurations.

We define an iteration (represented by I where $1 \le I \le I_{max}$) to correspond to an interval along the horizontal axis in Figure 2.4. That is, N_{max} new configurations are generated during iteration I. An iteration defines a set of N_{max} moves during which the range limiter window dimensions remain constant.

In simulated annealing, the more new configurations generated during the course of a run, the higher the probability of achieving a better solution. However, our extensive experimentation suggested the existence of a diminishing return on the number of new configurations generated. Therefore, a default number of moves can be determined for which the best results can be obtained with high probability. The default total number of moves during a run is set to:

$$\text{total\_moves} = 1500 N_c^{4/3} \tag{2.3}$$

where N_c is the number of cells. In our implementation, we set I_{max} equal to 150 iterations. Therefore:

$$N_{max} = 10 N_c^{4/3} \tag{2.4}$$

Note that the range limiter dimensions are actually changed 50% of 150 times, or 75 times during the course of a run (that is, its dimensions only change during region 2 of the operation of the annealing algorithm).

Since we know that the acceptance rate behavior described in Figure 2.3 along with the default values of I_{max} and N_{max} yield close to the best possible results for simulated annealing, we force the algorithm to strictly obey that acceptance rate behavior. That is, for each iteration I (I varies from 1 to I_{max}), we compute the target acceptance rate (α_I^T) as shown in Figure 2.5. To ensure that significant further reductions in the cost are not possible, we set the target acceptance rate to be below one percent at the last iteration (I_{max}).

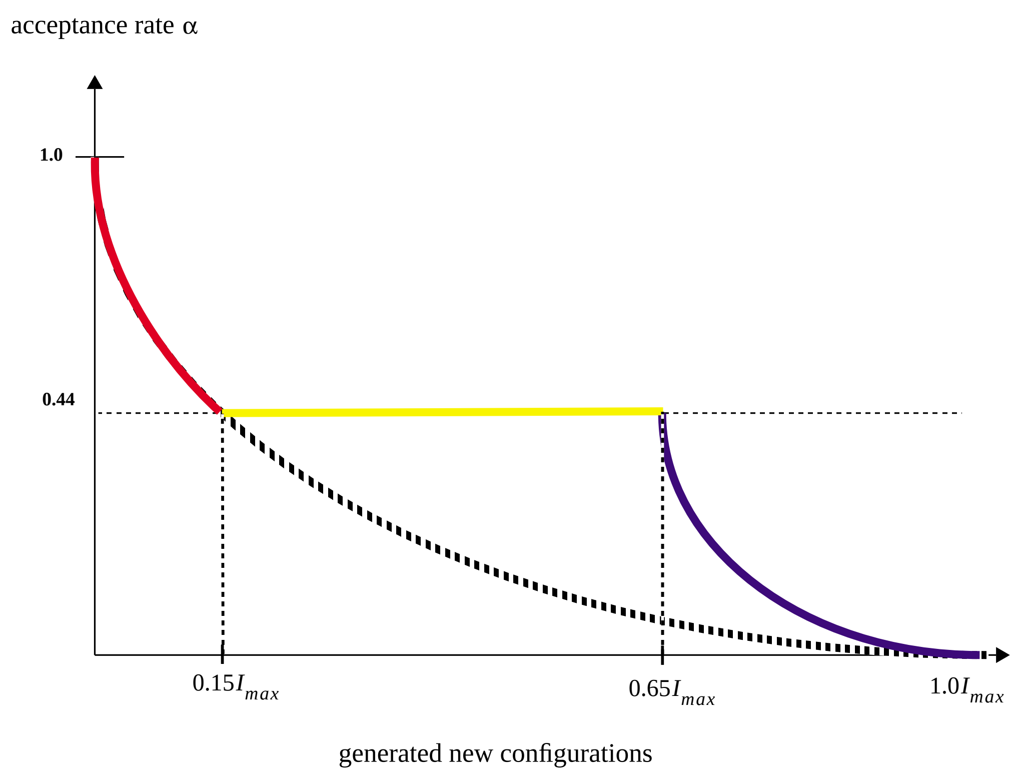

Figure 2.5 Target acceptance rate versus iteration.

We force the actual acceptance rate to track the target acceptance rate by using negative feedback control on the temperature *T*:

$$T = \left[1 - \frac{\alpha_I - \alpha_I^T}{K}\right] T \tag{2.5}$$

where *K* is a damping constant used to stabilize the control of the value of T (in our implementation, a very suitable value of *K* is 40). *T* is updated every *update_limit* moves (as defined in the description of our simulated annealing algorithm in Figure 2.6). Note that *T* can *increase* as well as decrease as the execution of the algorithm proceeds, and the range limiter window dimensions decrease exponentially as a function of the number of itera-

```
Algorithm simulated_annealing($X_0$)
1  $X \leftarrow X_0$                  /* set current configuration equal to initial configuration */
2  $T \leftarrow$ set_initial_T()      /* sufficiently sample configuration space to ascertain value of T
                                        yielding an initial acceptance rate $\alpha_1$ slightly below 100 percent */
3  $I \leftarrow 1$
4  while $I \le I_{max}$ do
5      $N \leftarrow 0$                /* N is the number of moves attempted so far during iteration I */
6      set_range_limiter_size($I$)         /* sets range limiter window dimensions */
7      $up \leftarrow 0$                   /* update counter */
8      while $N < N_{max}$ do
9          $up \leftarrow up + 1$
10         $N \leftarrow N + 1$
11         if $up$ = update_limit then     /* we need to update the temperature T */
12             $up \leftarrow 0$           /* reset the counter */
13             if $\alpha_I < \alpha_I^T$ then
14                 raise_temp($T$)
15             else if $\alpha_I > \alpha_I^T$ then
16                 lower_temp($T$)
17         $Y$ = generate($X$)             /* propose a new configuration */
18         $\Delta C = C(Y) - C(X)$        /* compute the cost change */
19         if accept($\Delta C$, $T$) then
20             $X \leftarrow Y$            /* accept the new configuration to be the current config. */
21     $I \leftarrow I + 1$
```

Figure 2.6 Our simulated annealing algorithm. The cumulative measured acceptance rate during iteration *I* is α_I ; the target acceptance rate at iteration *I* is α_I^T . The algorithm will execute a total of I_{max} iterations.

tions. In Lam's schedule by contrast, *T* decreases monotonically but the range limiter window dimensions fluctuate up or down. Clearly these two parameters are closely related. It is sufficient to dictate the functional form for either one, and let the other parameter adapt to monitored conditions.

Based on extensive measurements, our new interpretation of Lam's schedule did not show a difference in placement quality for a given execution time as compared to Lam's original version. Our new approach generates a fixed number of moves for a circuit of a given size, and therefore, the number of iterations is known *a priori*.

2.4 Conclusions

We have presented a new simulated annealing algorithm which is based on a theoretically derived annealing schedule. The new annealing schedule yields results which are comparable to those obtained with Lam’s statistical schedule. This algorithm is the basis for the placement and partitioning algorithms presented in this thesis.