Enhanced Simulated Annealing for Automatic Reconfiguration of Multiprocessors in Space

James R. Slagle, Ashim Bose, Perry Busalacchi, Catherine Wee,
University of Minnesota, Minneapolis, Minnesota.

Abstract

This paper describes our recent results in developing enhanced simulated annealing algorithms using a LISP environment. The application is to use simulated annealing for automatic reconfiguration of multiprocessors in space. Our approach to solving this problem involves a combination of object-oriented programming, search strategies, knowledge based reasoning, and an advanced reconfiguration algorithm. The application was developed and is being enhanced on a LISP workstation (Xerox Dandelion) using LOOPS. This environment played an important role in both the initial success of the prototype [9] and the recent development efforts [3] [16] [22].

1 Introduction

Simulated Annealing (SA) is a stochastic optimization technique that was inspired by concepts from statistical mechanics. Problems that lend themselves to a simulated annealing-based solution include NP-complete problems such as the traveling salesman [2] [8] and quadratic assignment problems, among others [1] [4] [5] [6] [7] [12] [11] [13] [14] [17] [19] [18] [20] [21] . Additionally, SA has proven to be a viable method for determining the weights between nodes in a neural network. We will now introduce this sometimes controversial but always interesting algorithm.

Given a combinatorial optimization problem specified by a set C of configurations and by a cost function h defined on all the configurations S , belonging to C , the SA algorithm is characterized by a rule to randomly generate a new configuration with a certain probability, and by a probabilistic acceptance rule according to which the new configuration is accepted or rejected. A parameter T, called the temperature, controls the acceptance rule. The generic simulated annealing algorithm is described as follows:

procedure Simulated Annealing

```
iterations <- i0        (* initial number of iterations *)
T <- T0                 (* initial temperature          *)
S <- S0                 (* initial configuration        *)
repeat
    repeat
        NewS <- Perturb (S)
        if [h(NewS) <= h(S)] or
          [random# < exp((h(S) - h(NewS))/T)] then
          S <- NewS
    until inner loop has been repeated iterations times
    T <- UpdateTemperature(T)
    iterations <- UpdateIterations(iterations)
until outer loop terminating criteria
end
```

In the spaceborne processor array (SPA) environment, S_0 corresponds to an initial assignment of logical processors to physical processors, Perturb to a procedure that proposes a swap of an assignment of a logical processor from one physical processor to another of the same type, and h to an objective function where we want to minimize the total path length in the Module Interconnect Network (MIN). Although simulated annealing methods have proved to be successful for finding near optimal solutions to difficult problems, they often require long run time. We address this further in Section Four.

In the sections that follow we will introduce the application to which we apply the simulated annealing algorithm, the LISP environment, details of the resulting algorithm enhancements, future directions for research, and our conclusions. Simulated annealing is the method we use to solve the logical to physical assignment problem.

2 The Application

Control Data Corporation has developed a multiprocessor system designed for use in space. The system is called the Space Processor Array (SPA). The modules are connected using a simple bi-polar arrangement. The top plane consists of discrete processors of various types not connected. General purpose, signal processors, and symbolic processors are examples of processor types. The bottom plane is the Module Interconnect Network (MIN) that provides the interprocessor communications hardware. For each processor in the top plane there is a corresponding "part" called the Configurable Network Unit (CNU) in the MIN. All interprocessor communication takes place in the MIN, with the path of the communication being determined by the programmable CNU's. Each of four sides of a CNU can be thought of as an edge with the limitation of two paths entering the CNU through any single edge. Initial configuration of an array of processors or reconfiguration after a failure aboard the spacecraft requires a mapping of the logical design, representing the needs of the software application, onto the physical array of processors. The logical processors are grouped into rings, reflecting required dataflow and inherent parallelism in the logical application. The mapping of the logical onto the physical must be done in a way to minimize the total path length required, while at the same time satisfying the constraints of module types and network limitations. Reconfiguring an array requires, in addition to the above, that the system bypasses a failed module.

3 An Example

In this section, a simple example is described to illustrate the working of the SPA program. In this example, the logical design (Figure 2) is to be mapped onto a 3x3 SPA. On comparing modules in the logical design with the modules in the SPA, it is evident that the SPA contains spares for each processor type. Hence, the logical to physical mapping entails determining assignments of logical modules to physical modules and the circuits between the physical modules. The solution determined by the SPA program including the execution time is shown in Figure 1.

4 The Environment

Due to the rapid prototyping capability of the LISP machine and development environment, we developed a prototype simulator in less than three months that used simulated annealing to perform the reconfiguration task. Throughout development we relied heavily

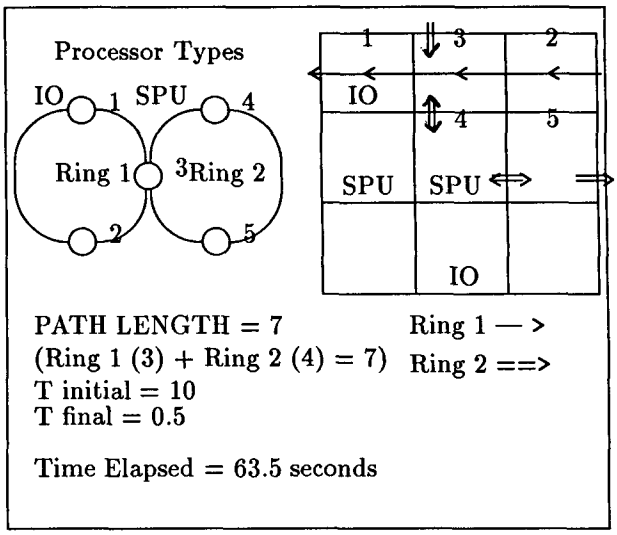

Figure 1: Example 1 – 3x3

on our ability to create a good simulation of the physical system by using the object hierarchy defined by LOOPS. The level of abstraction provided by the object and attribute hierarchy provided data structures that were both meaningful and efficient. Association lists were used to quickly determine the change in the state of the system at different levels. The graphical representations of the object and function hierarchy provide focal points that the developers could view to gain overall system perspective. The graphics provided by the system again proved their worth in providing a visual representation of the SPA hardware system and immediate feedback for changes in the SA algorithm. In this way the swaps proposed for individual assignments and any feasible solutions found by the algorithm were easily displayed. When combined with liberal use of gauges, active values, and inspectors, we were able to watch the algorithm in action, and thereby closely monitor the solution progress. This led to several insights on how the algorithm worked and how it could be improved. The excellent debugging environment allowed for quickly tracing errors to individual functions.

Continuing work has been underway at the University of Minnesota. The LISP environment has been responsible for the short time that was necessary to get the researchers up to speed. The following results were obtained by part-time researchers over a period of five months.

5 Enhancements to the SA algorithm

We have enhanced the SA algorithm by examining strategies for temperature reduction and different penalty functions and their effects.

5.1 Strategies for temperature reduction

We focused our research on three aspects of the temperature portion of the SA process:

- Selecting an Initial Temperature
- Determining the Number of Iterations at a Temperature and;
- Determining the Stopping Criterion.

5.1.1 Selecting an Initial Temperature

The initial temperature is important from two opposing perspectives. The higher the initial temperature, the more randomness that would be allowed, reducing the possibility of stopping at a local minimum. The lower the initial temperature, the faster we arrive at a solution. A key to efficiently solving the reconfiguration problem is finding an acceptable compromise.

Our research included three methods for selecting an initial temperature:

1. Ratio Method;
2. Initial Configuration Based Method, and;
3. Standard Deviation estimated Method.

Ratio Method for Selecting an Initial Temperature.

This was the initial method used to select an initial temperature and is defined by the following formula:

T_0 = 20 * (# of logical modules / # of physical modules)

where:
# of logical modules is the total number of modules of all module types required by the problem or application, and
# of physical modules is the total number of modules in the physical array. This leads to a conservative (high) starting temperature.

Initial Configuration Based Method of Selecting an Initial Temperature.

This method is defined by the following algorithm:

Select an initial configuration;
Take a few one step random transitions from this initial configuration and record all the changes in cost;
if (there are one step random transitions that result in an increase in cost) **then**
Calculate the average increase in cost, C+;
T_0=C+/-ln(P);
where P is the acceptance ratio. The acceptance ratio is the ratio of the number of accepted transitions to the number of proposed transitions.
P = 0.80 if there are more transitions that result in an increase in cost than transitions that result in a decrease or no change in cost.
P = 0.65 if there are an equal number of transitions that result in an increase in cost and transitions that result in a decrease or no change in cost.
P = 0.50 otherwise.
else
T_0 = 20 * Standard Deviation of all the costs obtained (including the cost of the initial configuration).

There are several advantages in calculating the initial temperature this way. The temperature, T plays an important role in whether a random transition is accepted. A random transition is accepted if exp (-C/T) is greater than a generated random number, lying between 0 and 1, where C is the change in cost, i.e. the difference in cost of the new configuration and the current configuration.

Given a certain value P, that represents the probability of acceptance, and C+, the estimated higher cost, initial temperature T_0 is computed. By doing this, we hope to relate T_0 with the initial configuration that is chosen. This may enable the system to start with a lower temperature and obtain the solution faster if a good initial configuration is chosen. If a less satisfactory initial configuration is chosen, the system would start at a higher temperature, and take a longer time to find the solution.

Standard Deviation Estimated Starting Temperature Method.

In this method, a few random transitions are taken and the standard deviation of the cost distribution is computed. The initial temperature is computed as follows:

T_0 = 20 * Standard Deviation of the costs

In this method, T_0 is chosen such that almost all configurations would be accepted. Since T_0 allows almost all configurations to be accepted, it can be considered as the upper bound of the temperatures. Thus, initial temperatures calculated using this method would always be higher than the previous method.

5.1.2 Determining the Number of Iterations at a Temperature

As in the previous section, there are opposing perspectives to the value that this number should take

on. The larger the number, the more thorough and time consuming will be the search. The compromises used in this study include:

- Number of Physical Modules Method;
- Cost-Based Function Method and;
- Configuration-Rejected Method.

Number of Physical Modules Method for Determining Number of Iterations.

This method involved remaining at a temperature for the number of iterations equal to the number of modules in the physical array.

Cost-Based Function Method for Determining Number of Iterations

The number of iterations, N(T), to execute at a particular temperature T, was obtained as:

N(T)=exp((hmax(T)-hmin(T))/T)

where hmax(T) and hmin(T) were the highest and lowest values of the cost function obtained thus far at this temperature. Originally we defined hmax(T) and hmin (T) to be the highest and the lowest value obtained so far during the execution of the system. However, N(T) became too large and the system ran too long.

As the temperature decreases, N(T) may become large, thus we impose an upper bound of three times the number of physical modules.

The above formula was suggested by Remeo and Sangiovanni-Vincentelli [10], when they observed that to obtain a final configuration close to a globally minimal one, there should always be a sufficiently large probability to leave any configuration, possibly a local minimum. A good estimate of the number of iterations required is given by the above formula for N(T).

N(T) above has the property of being small when the temperature is high. As temperature decreases, N(T) would increase correspondingly.

Configurations-Rejected Method for Determining Number of Iterations.

In this method, the number of iterations taken depends on the number of configurations being rejected. At every temperature, a certain number of configurations must be rejected before it can go to another temperature. The number of configurations to be rejected was set equal to the number of physical modules. This method has the property of running many iterations when the temperature was high and running just a few iterations when the temperature decreases.

Since configurations at higher temperatures can be accepted more easily, this method tries to find the solution close to the beginning of the simulation.

5.1.3 Determining the Stopping Criterion

Originally, the system would stop if the temperature was less than the minimum temperature. The minimum temperature was either user-defined or set to be 0.1. The system would also stop if the solution was equal to the lower bound, i.e. the cost as computed from the logical design.

For this study, the system was modified so that it would stop if any one of the following conditions was true.

1. The temperature was less than the minimum temperature of 0.1;
2. The cost was equal to the lowest possible cost;
3. The minimum cost accepted has not changed for four consecutive decreases in temperature. We used this condition as a stopping criterion as the commercial package, TimberWolf, also uses this as its stopping criterion for its SA algorithm implementation;
4. No swap was accepted at a particular temperature. This condition would be applicable only if there were four or more decreases in temperature and may mean that a minimum configuration has been obtained.

However, after doing some runs, we found that conditions 3 and 4 were not suitable for these assignment problems. Condition 3 stopped the system too early, and hardly any run terminated because of condition 4.

As a result, the first two conditions remained as the stopping criterion except that the minimum temperature is set to 0.3.

5.1.4 Tests

To test these alternatives, some representative problems were identified. Each of these test problems was considered for testing against four combinations of temperature strategies identified as Cases A through D.

A. The initial temperature was calculated by taking the average increase in cost of one step transitions from the initial configuration. The number of iterations to be performed before going to another temperature was determined from the temperature and the maximum and minimum cost obtained at that temperature.

B. The initial temperature was calculated by taking the average increase in cost of one step transitions from the initial configuration. The number of iterations to be performed at every temperature depended on the number of swaps rejected.

C. The Standard Deviation-Estimated Method was used to calculate the initial temperature. The number of iterations to be performed before going to another temperature was determined by the temperature and maximum and minimum cost obtained at that temperature.

D. The Standard Deviation-Estimated Method was used to calculate the initial temperature. The number of iterations to be performed at every temperature depended on the number of swaps rejected.

Initial Observations.

After a few initial tests, we confirmed that the Standard Deviation-Estimated Method selected an initial temperature that was too high. On the other hand, the initial configuration-based method is more selective and hence more efficient by using a lower starting temperature for some configurations than others. Thus, cases C and D were eliminated from consideration.

The Configuration-Rejected Method for determining the number of iterations tends to run quite long when higher temperatures are encountered. This is due to the fact that the most swaps are accepted at higher temperatures. When the Configurations-Rejected Method was used in combination with the Standard Deviation-Estimated Starting Temperature Method (Case D), long execution times were observed. Thus, Case D had a second reason for removal from consideration.

Final Test Configurations.

Cases A and B were evaluated for each test problem. Three test problems were defined with each having a different logical definition and associated physical array size. The three physical sizes were 9, 16, and 25 modules.

Each test problem was run using cases A and B a total of five times each. Because the initial configuration is chosen using a random number, the five tests were all different.

Final Observations.

1. Using the Initial Configuration-Based Method for selecting an initial temperature is better than the Standard Deviation-Estimated Method.

2. The relationship between initial temperature (as determined using the Initial Configuration Based Method) and initial cost was observed for different assignment problems. Configurations with the same cost do not necessarily start at the same temperature, since temperature is meant to indicate how close a configuration is to an optimal solution. A configuration with lower cost does not necessarily mean that it is closer to an optimal solution as there may not be any direct transition that leads to an optimal solution.

3. Although initially we thought that case B (Configurations-Rejected Iteration Method) would always be better than case A, (Cost-Based Iteration Function Method), results showed that case A is better. Case A always had shorter run times and managed to find more optimal solutions than case B.

4. Case A was also a more conservative algorithm, since the ratio of accepted swaps to total swaps was lower.

5. What we have done so far (finding a suitable initial temperature, doing the right number of iterations before going to another temperature, and finding the correct time to terminate) is to improve the SA algorithm. In general, we believe that the best approach for solving large problems would be to use, in addition, some problem specific knowledge (heuristics). Improving just the algorithm would not enable an almost optimal solution to be found within a reasonable time.

6. Problem size is related to the number of possible assignments, the number of assignments that are optimal solutions and the number of assignments that would lead to these optimal assignments. Because fewer assignments were optimal, it proved to be more difficult to find an optimal solution for a test problem that was smaller than another in terms of logical and physical modules.

7. A good initial assignment is another important factor for finding an optimal solution. However, it is difficult or even impossible to determine which initial assignment is good. Using parallel processing would be a solution to assignment problems as different initial assignments could be run at the same time and the best solution can then be chosen.

5.2 A study of different penalty functions and their effects

The Simulated Annealing (SA) algorithm implemented in the SPA program relies heavily on the accuracy of the "energy" or "cost" function that is to be

minimized. The cost of a given configuration of the SPA essentially represents the state of the configuration. Hence, the more accurate this representation, the better the results of the SA algorithm. In the current implementation, the cost function is defined as:

Cost for a given SPA configuration = total path length + total penalty
where, the penalty is given by the penalty function (PF) that is defined as:

Penalty for a given SPA configuration

$$\sum_{i=1}^{N} v(i)$$

where N = # of edges along the paths of all rings and where the number v(i) of violations at edge i is:

max[(# of paths through edge i - maximum permissible # of paths through edge i), 0]

Recall that an "edge" can have only two paths through it. The current penalty function appears to be too simplistic because it penalizes all individual edge violations equally irrespective of the number of violations and the control temperature of the configuration. We believe that the search to find an optimal configuration of the SPA can be made more efficient by modifying the penalty function to evaluate a given configuration more accurately.

Several penalty functions have been tried on a diverse range of problems. These will be described in the rest of this section.

5.2.1 Using Arbitrary Numbers as Multipliers to the Edge Violations

Our first study direction for the penalty function involved the use of penalities that varied directly and exponentially with the number of edge violations. Let the total penalty (represented by the modified PF) be defined as:

$$\sum_{i=1}^{N} p(i)$$

where p(i) = penalty for edge i, and N is as defined previously. When the edge penalty varies exponentially as the number of edge violations, the edge penalty is defined as:

$$p(i) = b^{v(i)}$$

Two values of b were tried, 1.5 and 0.75. The reason for using such radical PF's is to study the behavior of the SA algorithm when succeeding edge violations are penalized more heavily and more leniently.

When using b = 1.5, the algorithm becomes extremely conservative in moving to an invalid configuration. This inhibits an exhaustive search, and the algorithm tends to remain close to the initial feasible solution. The percentage of swaps that are rejected increases by an average of 34%. This can be explained by the fact that a change (C) in cost for an infeasible configuration is large causing the probability of acceptance of a swap [P=exp(-C/T)] to be low, reducing the chance that the swap will be accepted. This impedes search, and often the best feasible solution is inferior to that obtained by the use of the original PF.

Using a base of 0.75, the percentage of proposed swaps rejected decreases by an average of 22%. Since an infeasible configuration is not penalized by a significant amount, search is promoted. However, this does not necessarily facilitate the discovery of a better solution. In most cases, the solution is as good as the one obtained by the original PF. The conclusion is that the extra effort involved in the larger search is not worthwhile.

When the edge penalty varies directly as the number of edge violations, p(i) is defined as:

$$p(i) = m * v(i)$$

Two values were tested, 1.5 and 0.75. The results obtained with these multipliers are not significantly different from those obtained by the original PF for relatively large networks (more than 7 logical modules, 2 or more rings, 2 or more ring spanners). There is a difference in the percentage of proposed swaps that are rejected but this can be explained along lines similar to those in the previous paragraph.

Our second study direction for the penalty function involved the use of penalities that were temperature based. The temperature of the SPA is an important property that can be incorporated into the PF. As the temperature decreases, the algorithm is closer to its final solution. It is therefore necessary to limit search by penalizing an infeasible solution more heavily at a lower temperature. This is to make sure that at a low temperature, a swap is not made for an infeasible solution if the number of edge violations is large (because that would make it more difficult to find a better feasible solution given that the end of the search is near). The edge penalty is defined as:

$$p(i) = M * (Tstart/Tcontrol) * v(i)$$

where M is a multiplier (constant or problem-based) of the temperature ratio.

When the multiplier is a constant, 1.5 is selected to ensure that the penalty for an edge violation is greater than unity even when Tstart = Tcontrol. With this PF, consistently better results (than those with the original PF) are obtained.

When the multiplier is problem based, the multiplier is based on the given SPA and the logical design. The parameters from the problem are divided into two categories:

1. parameters for which search should be promoted, and;
2. parameters for which a large search is unnecessary.

Various combinations of the parameters were tried in the PF through the multiplier that is defined as:

M = different combinations of type 2 / different combinations of type 1

The results from this study suggest that a suitable multiplier is of the form

$$M = \prod_{j=1}^{L} S_{l,j}/S_j$$

where $S_{l,j}$ = # of logical modules of type j;
S_j = # of physical processors of type j, and;
L = # of different types of logical modules.

When M is defined in this way, its value for all problems tried is in the range 0.1 to 0.5. Since M < 1, the edge violations are not penalized heavily initially so that configurations that do not look promising initially but that might lead to better configurations later on, are considered. In all the examples tried, this PF gave the best results.

A Case Study.

For the case study, the original PF is compared to the temperature based PF's described in the previous paragraphs.

With the original PF, the path length was 12. The number of swaps proposed was 153. Eighty-six of these were accepted. With the temperature based PF of the form M * Tstart / Tcontrol where M had a weight of 1.5, the path length was 10. 170 swaps were proposed. Seventy-nine of these were accepted. Where M is problem-based of the type as described in the previous paragraph, the path length was 10. The # of swaps proposed was 153. Eighty-six of these were accepted. These results clearly indicate that the temperature based PF's perform better.

For all the examples that have been tried, the highest starting temperature was 15 and the ending temperature was 0.1. If high starting temperatures are being considered (> say 25), then the temperature based PF's should be modified (perhaps by taking the square root of Tstart/Tcontrol) since the ratio Tstart/Tcontrol will become too high for low values of Tcontrol.

6 Conclusions and Future Directions

We conclude here that the LISP environment that we used was a major factor in rapid prototyping. In addition, new researchers were able to quickly learn the concepts used in the algorithm and able to use this knowledge to develop working enhancements to the original prototype. While SA is easy to understand and flexible, much is required to tune it to a particular application. The environment we used made this possible for the logical to physical assignment problem encountered in reconfiguring multiprocessors.

Our future research in this area is proceeding in two directions:

1. We are investigating implementing the algorithm in a distributed environment and;
2. We are investigating using an expert system approach to solve this problem.

References

[1] Prithviraj Banerjee and Mark Jones. A parallel simulated annealing algorithm for standard cell placement on a hypercube computer. *Proc. IEEE Int. Conf. Computer-Aided Design (ICCAD-86)*, November, 1986.

[2] E. Bonomi and J. Lutton. The n-city traveling salesman problem: statistical mechanics and the metropolis algorithm. *Siam Review*, 26(4):551–568, 1984.

[3] Ashim Bose, James Slagle, and T.R. Fennel. A study of the penalty function in the space processor array (SPA) program. *Memo UMCS-AI-88-3, University of Minnesota, Minneapolis*, March 28, 1988.

[4] A. Casotto, F. Romeo, and Vincentelli. A.S. A parallel simulated annealing algorithm for the placement of macro-cells. *Proc. Int. Conf. on Computer Aided Design*, November, 1986.

[5] Moon Chung and Kotesh Rao. Parallel simulated annealing for partitioning and routing. *Proc. IEEE Int. Conf. on Computer Design (ICCD-86)*, 238–242, October, 1986.

[6] F Darema, S Kirkpatrick, and V.A. Norton. Parallel algorithms for chip placement by simulated annealing. *IBM J. Res. Develop.*, 31(3), May 1987.

[7] Lawrence Davis. *Genetic Algorithms and Simulated Annealing.* Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.

[8] Edward Felten, Scott Karlin, and Steve Otto. The traveling salesman problem on a hypercubic, mimd computer. *IEEE Proceedings of the 1985 Parallel Processing Conference*, 6–10, August, 1985.

[9] T.R. Fennel. Teaching a computer to fix itself - when there's no alternative. *International Computers in Engineering Conference*, 1988.

[10] M. Huang, F. Romeo, and A. Sangiovanni-Vincentilli. An efficient general cooling schedule for simulated annealing. *Department of EECS, University of California, Berkeley*, 1984.

[11] S. Kirkpatrick. Optimization by simulated annealing: quantitative studies. *Journal of Statistical Physics*, 34(5/6):975–986, 1984.

[12] S Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671 – 680, May, 1983.

[13] Saul Kravitz. Multiprocessor-based placement by simulated annealing. *Private communication by the Semiconductor Research Corporation*, February 1986.

[14] Saul Kravitz. Multiprocessor-based placement by simulated annealing. *Proc. 23rd Design Automation Conference*, 567–573, June, 1986.

[15] S. Nahar, S. Sahni, and E. Shragowitz. Simulated annealing and combinatorial optimization. *Technical Report Number 85-56 University of Minnesota*, December, 1985.

[16] Byung Joon Park, James Slagle, and T.R. Fennel. Experiments with simulated annealing. *Memo UMCS-AI-88-2, University of Minnesota, Minneapolis*, March 28, 1988.

[17] Rob Rutenbar and Saul Kravitz. Multiprocessor-based placement by simulated annealing. *Proc. IEEE Int. Conf. on Computer Design (ICCD-86)*, 434–437, October, 1986.

[18] C. Sechen and A.S. Vincentelli. Timberwolf 3.2 : a new standard cell placement and global routing package. *Proc. 23rd Design Automation Conference*, 432–439, June, 1986.

[19] C. Sechen and A.S. Vincentelli. The timberwolf placement and routing package. *Proc. Custom Integrated Circuits Conf.*, 522–527, May, 1984.

[20] E. Sontag and H. Sussman. Image restoration and segmentation using the annealing algorithm. *Proceedings of the 24th Conference on Decision and Control*, December, 1985.

[21] M. Vecchi and S. Kirkpatrick. Global wiring by simulated annealing. *IEEE Transactions on Computer-Aided Design*, CAD-2(4):215–222, October, 1983.

[22] Catherine Wee, James Slagle, and T.R. Fennel. Strategies for temperature reduction in simulated annealing. *Memo UMCS-AI-88-4, University of Minnesota, Minneapolis*, March 28, 1988.