UNIVERSITY OF CALIFORNIA

Los Angeles

Simulated Annealing with Errors

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in Computer Science

by

Daniel Rex Greening

1995

The dissertation of Daniel Rex Greening is approved.

Joseph Rudnick

Lawrence McNamee

Andrew Kahng

Rajeev Jain

Miloš Ercegovac, Committee Chair

University of California, Los Angeles

1995

DEDICATION

To my grandmother, the late V. Mae Teeters, whose integrity, tenacity, resourcefulness, intelligence and love inspired me. I miss you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Dr. Miloš Ercegovac, my advisor at UCLA, supported me in many tangible and intangible ways: money, equipment, friendship, and advice.

Dr. Frederica Darema and the IBM T.J. Watson Research Center supported my simulated annealing research from its inception.

These people reviewed Chapter 2: Dr. Stephanie Forrest of LLNL, Dr. Frederica Darema, Dr. Dyke Stiles of Utah State University, Dr. Steven White of IBM, Dr. Andrew Kahng of UCLA, Richard M. Stein of the Ohio Supercomputer Center, Dr. M. Dannie Durand of Columbia University and INRIA, Dr. James Allwright of the University of Southampton, and Dr. Jack Hodges of CSU, San Francisco.

The following people reviewed portions of Chapter 4: Dr. Geoffrey Sampson of Leeds University, Dr. Thomas M. Deboni of LLNL, Dr. Albert Boulanger of BBN, Dr. John Tsitsiklis of MIT, Dr. Philip Heidelberger of IBM, Todd Carpenter of Honeywell Systems and Research Center, Karlik Wong of Imperial College. I am particularly grateful to Drs. Tsitsiklis and Heidelberger for pointing out a suspicious assumption (now proved), and to Dr. Boulanger for an important citation.

Jen-I Pi of ISI supplied information about Kernighan-Lin graph bisection and Karmarkar-Karp number partitioning.

My friends helped me on my journey: Ron Lussier, Greg Johnson, Verra Morgan, Jack Hodges, Leo Belyaev.

VITA

January 8, 1959	Born, Alma, Michigan
1979–1981	Teaching Assistant University of Michigan, Computer Science Ann Arbor, Michigan
1979–1982	Systems Research Programmer I University of Michigan Computing Center
1982	B.S.E., Computer Engineering, Cum Laude University of Michigan
1982–1984	Project Manager Contel/CADO Business Systems Torrance, California
1985–1986	President (elected) University of California Student Association Sacramento, California
1985–1986	Vice-President External Affairs (elected) UCLA Graduate Student Association Los Angeles, California
1988	M.S., Computer Science University of California, Los Angeles
1988–1991	Student Research Staff Member IBM T.J. Watson Research Center Yorktown Heights, New York
1991	Simulated Annealing Session Chair 13th IMACS Congress on Computation & Applied Math Dublin, Ireland
1991–1994	Director Novell, Inc. Cupertino, California
1994–	President Chaco Communications, Inc. Cupertino, California

PUBLICATIONS

—. "Type-Checking Loader Records." *Proceedings, MTS Development Workshop VI*, Rensselear Polytechnic Institute, Troy, New York, pp. 1271-1284, June 1980.

—. *Modeling Granularity in Data Flow Programs*, Masters Thesis, UCLA, 1988.

E.L. Foo, M. Smith, E.J. DaSilva, R.C. James, P. Reining, — and C.G. Heden. "Electronic Messaging Conferencing on HIV/AIDS." *Electronic Message Systems 88: Conference Proceedings*, pp. 295-305, Blenheim Online Ltd., Middlesex, UK, 1988.

— and M.D. Ercegovac. "Using Simulation and Markov Modeling to Select Data Flow Threads." *Proceedings of the 1989 International Phoenix Conference on Computers and Communications*, pp. 29–33, Phoenix, Arizona, 1989.

— and A.D. Wexelblat. "Experiences with Cooperative Moderation of a USENET Newsgroup." *Proceedings of the 1989 ACM/IEEE Workshop on Applied Computing*, pp. 170-176, 1989.

— and F. Darema. "Rectangular Spatial Decomposition Methods for Parallel Simulated Annealing." In *Proceedings of the International Conference on Supercomputing*, pp. 295–302, Crete, Greece, June 1989.

—. "Equilibrium Conditions of Asynchronous Parallel Simulated Annealing." In *Proceedings of the International Workshop on Layout Synthesis*, Research Triangle Park, North Carolina, 1990. (also as IBM Research Report RC 15708).

—. "Parallel Simulated Annealing Techniques." *Physica D: Nonlinear Phenomena*, **42**(1–3):293–306, 1990.

—. "Review of The Annealing Algorithm." *ACM Computing Reviews*, **31**(6):296–298, June 1990.

—. "Asynchronous Parallel Simulated Annealing." In *Lectures in Complex Systems, volume III*. Addison-Wesley, 1991.

—. "Parallel Simulated Annealing Techniques." In Stephanie Forrest, editor, *Emergent Computation*, pp. 293–306. MIT Press, 1991.

—. "Simulated Annealing with Inaccurate Cost Functions." In *Proceedings of the IMACS International Congress of Mathematics and Computer Science*, Trinity College, Dublin, 1991.

—. "OWL for AppWare." In *Borland International Conference 1994*, Orlando, Florida, 1994.

PRESENTATIONS

— and F. Darema. “Errors and Parallel Simulated Annealing,” *LANL Conference on Emergent Computation*, Los Alamos, New Mexico, May 1989.

—. “Parallel Asynchronous Simulated Annealing.” University of California, Santa Cruz, Computer and Information Sciences Colloquium, Nov 8, 1990.

—. “Parallel Asynchronous Simulated Annealing.” Stanford University, DASH Meeting, Nov 26, 1990.

— et al. “Cross-Platform Development Panel.” *Motif and COSE Conference 1993*, Washington, DC, Nov 1993.

—. “Introduction to ObjectWindows for AppWare Foundation,” 3 sessions, *Novell Brainshare Conference*, Salt Lake City, Utah, Mar 1994.

ABSTRACT OF THE DISSERTATION

Simulated Annealing with Errors

by

Daniel Rex Greening

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1995

Professor Miloš Ercegovac, Chair

Simulated annealing is a popular algorithm which produces near-optimal solutions to combinatorial optimization problems. It is commonly thought to be slow. Use of estimated cost-functions (common in VLSI placement) and parallel algorithms which generate errors can increase speed, but degrade the outcome.

This dissertation offers three contributions: First, it presents a taxonomy of parallel simulated annealing techniques, organized by state-generation and cost function properties.

Second, it describes experiments that show an inverse correlation between calculation errors and outcome quality. Promising parallel methods introduce errors into the cost-function.

Third, it proves these analytical results about annealing with inaccurate cost-functions: 1) Expected equilibrium cost is exponentially affected by γ/T, where γ expresses the cost-function error range and T gives the temperature. 2) Expected equilibrium cost is exponentially affected by $\beta/2T^2$, when the errors have a Gaussian distribution and β expresses the variance range. 3) Constraining range-errors to a constant factor of T guarantees convergence when annealing with a $1/\log t$ temperature schedule. 4) Constraining range-errors to a constant factor of T guarantees a predictable outcome quality in polynomial time, when annealing a fractal space with a geometric temperature schedule. 5) Inaccuracies worsen the expected outcome, but more iterations can compensate.

Annealing applications should restrict errors to a constant factor of temperature. Practitioners can select a desired outcome quality, and then use the results herein to obtain a temperature schedule and an "error schedule" to achieve it.

CHAPTER 1

Introduction

Simulated annealing is a computer algorithm widely used to solve difficult optimization problems. It is frequently used to place circuits in non-overlapping locations on a VLSI chip. Simulated annealing consumes substantial amounts of computation—one or more computation days to place a circuit are not uncommon. In response, researchers have parallelized annealing in many different ways, with mixed results.

The first widely-available publication on simulated annealing, by Kirkpatrick et al [KCV83], provides a brief practical overview. Van Laarhoven and Aarts provide a more complete introductory treatment of simulated annealing [LA87]. I was impressed by the theoretical discussion in Otten and van Ginniken's book [OG89], though it is a difficult read. My review of it appears in [Gre90b].

This chapter provides enough introductory material that the dissertation stands on its own, albeit somewhat abstractly. I construct the simulated annealing algorithm by modifying the greedy algorithm, show how circuit placers use simulated annealing, and describe the relationship between thermodynamics and simulated annealing. Finally, I outline the rest of the dissertation.

1.1 Simulated Annealing

Combinatorial optimization problems present this task: There is a finite set of feasible states S, where each state $s \in S$ is represented by n state-variables, so that $s = (v_1, v_2, \ldots, v_n)$. There is a cost-function $C\colon S \to \mathbb{R}$. Now, find a state with minimum cost. Many such problems are NP-complete or worse. Current algorithms to solve NP-complete problems require exponential time, based on n.

A near-optimal state is often good enough in practice. Several algorithms require only polynomial-time to produce a near-optimal state. One polynomial-time heuristic for these problems is the "greedy algorithm." Although it doesn't always produce a satisfactory outcome, it is the basis for simulated annealing.

1.1.1 The Greedy Algorithm

A greedy algorithm for combinatorial optimization has a generator, which outputs a randomly-chosen state from any input state. The set of output states produced from input state s is called the neighborhood of s.

The algorithm randomly chooses a first state, then starts a loop. The loop calls the generator to obtain a trial-state from the current-state. If the trial-state has lower cost, the algorithm selects the trial-state as its new current-state, otherwise, it selects the old current-state. The greedy algorithm continues its loop, generating trial-states and selecting the next state until some stopping criteria is met, usually when it sees no further improvement for several iterations. The greedy algorithm then returns the current-state as its outcome.

The greedy algorithm has a flaw: many problems have high-cost local minima. When applied to NP-complete problems it will likely return one.

1.1.2 Simulated Annealing

Simulated annealing augments the greedy algorithm with a random escape from local minima. The escape is controlled through a value called "temperature." Higher temperatures make the algorithm more likely to increase cost when selecting a trial-state. In this way, simulated annealing can "climb out" of a local minimum.

```
1.    $T \leftarrow T_0$;
2.    $s \leftarrow$ starting − state;
3.    $E \leftarrow C(s)$;
4.    while not stopping-criteria()
5.        $s' \leftarrow$ generate($s$) with probability $G_{ss'}$;
6.        $E' \leftarrow C(s')$;
7.        $\Delta \leftarrow E' - E$;
8.        if $(\Delta \leq 0) \vee ($random() $< e^{-\Delta/T})$
9.            $s \leftarrow s'$;
10.           $E \leftarrow E'$;
11.       $T \leftarrow$reduce-temperature($T$);
12.       end while;
```

Figure 1.1: Simulated Annealing

Figure 1.1 shows the simulated annealing algorithm. Line 1 sets the initial temperature to T_0. Lines 2 and 3 set the current-state s and its cost E. The loop at lines 4–12 generates a trial-state s', evaluates the change in cost Δ, selects the next current-state, and reduces the temperature until the stopping criteria is met.

Line 8 shows how simulated annealing accepts a trial-state. The first term, $(\Delta \leq 0)$, expresses greed: it always accepts a lower-cost trial state. The `random` function returns a uniformly-distributed random value between 0 and 1. The second term of line 6, $(\texttt{random}() < e^{-\Delta/T})$, expresses the likelihood of accepting a costlier trial-state.

When the stopping criteria is met, simulated annealing returns current-state s as its outcome.

At high initial temperatures, the second term of line 8 lets the algorithm explore the entire state space: it accepts almost all cost-increases. As the temperature drops, it explores big valleys, then smaller and smaller sub-valleys to reach the outcome. This allows it to escape local minima, as illustrated in Figure 1.2.

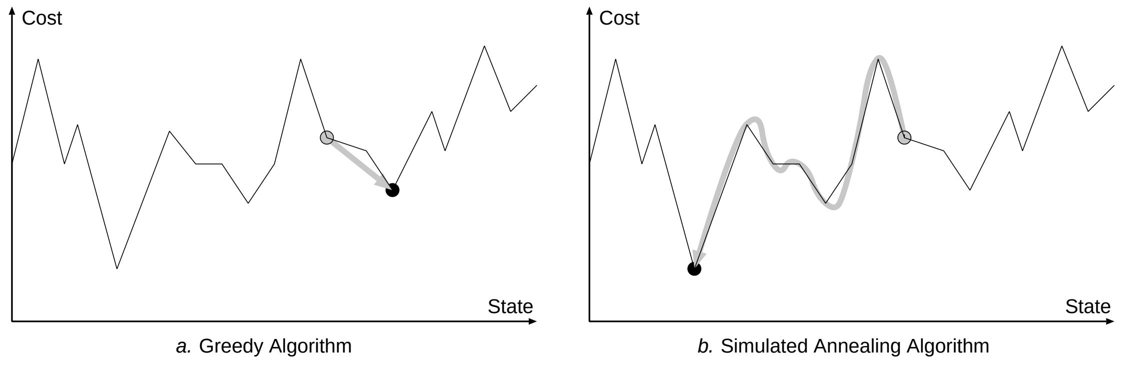


Figure 1.2: Annealing is a Modified Greedy Algorithm

Simulated annealing has a useful property: at a fixed temperature, it "equilibrates." That is, it approaches a stationary probability distribution, or "equilibrium." Temperature changes are usually chosen to keep transient distributions close to equilibrium. Simulated annealing's equilibrium is the "Boltzmann distribution," a probability distribution dependent solely on the cost-function.

These terms—annealing, equilibrium, temperature, Boltzmann distribution, etc.—come from thermodynamics. Though I describe simulated annealing as an algorithm, it behaves like a thermodynamic system. Many publications on simulated annealing appear in physics journals. To help explain simulated annealing, I will discuss thermodynamic annealing in §1.2.

1.1.3 Applications

Simulated annealing has been applied to several combinatorial optimization problems. Laarhoven's summary work [LA87] describes traditional applications. I list a few common applications, and some obscure ones below.

VLSI design: circuit placement and routing [Sec88, KCV83], circuit delay minimization [CD86], channel routing [BB88], array optimization [WL87]. *Hardware Design*: data-flow graph allocation [GPC88], digital filter design [BM89, CMV88], network design [BG89a], digital transmission code design [GM88], error-correcting code design [Leo88]. *Database Systems*: join query optimization [Swa89, SG89], distributed database topology [Lee88]. *Operations Research*: university course scheduling [Abr89], job shop scheduling [LAL88]. *Formal Problems*: the minimum vertex cover problem [BG89b], the knapsack problem [Dre88], sparse matrix ordering [Doy88], maximum matching [SH88]. *Image Processing*: image restoration [BS89, ZCV88], optical phase retrieval [NFN88], nuclear-magnetic resonance pulse optimization [HBO88]. *Organic Chemistry*: protein folding [BB89, Bur88, NBC88, WCM88], other molecular configuration problems [KG88]. *Artificial Intelligence*: perceptron weight-setting [Dod89], neural network weight-setting [Eng88], natural language analysis [SHA89].

Though annealing gives good results for these problems, it requires much computation time. Researchers have made several speed-up attempts: different parallel techniques, temperature schedules, cost functions, and move generation schemes.

1.1.4 Circuit Placement

Circuit placement was one of the first simulated annealing applications [GK85]. Reducing the area of a VLSI chip decreases its fabrication price; shortening total wire length increases its speed. Rearranging the circuits will change both properties. Optimizing this arrangement is “circuit placement.”

Simulated annealing remains popular for automatically placing circuits in chips [KCP94]. It has been used at IBM [KCV83], Thinking Machines [Wal89], and several universities [Sec88]. Commercially available CAD programs provided by Seattle Silicon, Mentor Graphics, and Valid use simulated annealing.

Circuit placement is the primary target problem in this dissertation, chosen for two reasons: first, simulated annealing is widely used for industrial circuit placement; second, placement consumes tremendous computing resources. It has become a major bottleneck in the VLSI-design process, second only to circuit simulation.

Efficient simulated annealing techniques help engineers produce chips more rapidly, with less expensive equipment. Faster algorithms allow chip manufacturers to reduce computation costs and improve chip quality.

1.1.4.1 Variants

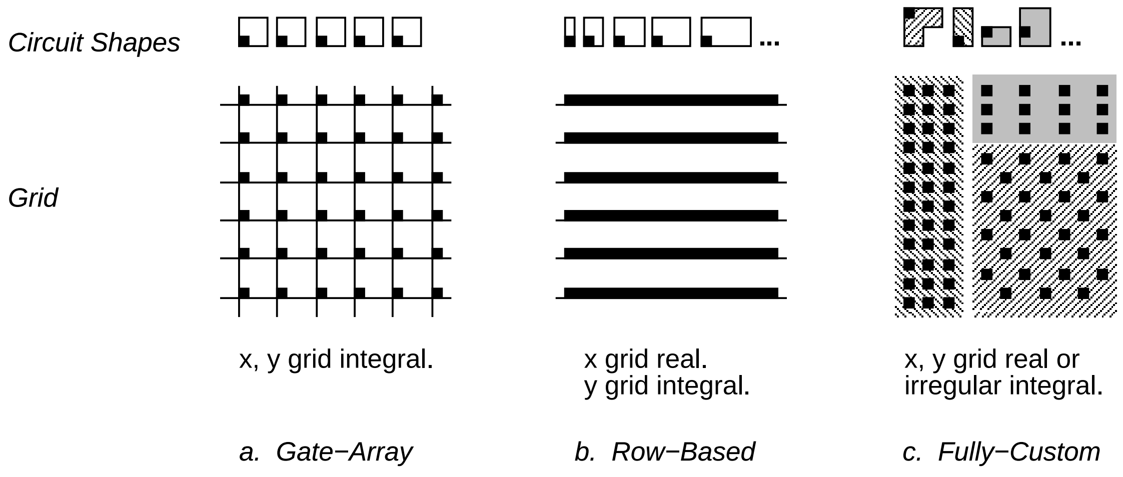


Figure 1.3: Classes of Circuit Placement

Variants of circuit placement fall into three categories. The simplest is "gate-array placement." Circuits have uniform height and width. They must be placed at points on a uniform grid, as shown in Figure 1.3a. The name "gate-array placement" comes from programmable gate-array chips: each gate is at a grid position.

Another variant is "row-based placement." Circuits have integral height (e.g., 1, 2, or 3) and unrestricted width (e.g. 1.34 or 2.53), as shown in Figure 1.3b. This occurs in some CMOS circuits, where restrictions on the size and relative placement of P-wells and N-wells enforces an integral height.

The most complicated variant is "macro-cell" or "fully-custom" placement.

Circuits may take any rectilinear shape. Feasible placement locations might be limited by the user. Each circuit might be assigned a circuit-class, and certain areas designated for particular circuit-classes. Odd shapes and area restrictions can appear in placing bipolar circuits, particularly on "semi-gate array" circuits where some mask steps are always done with the same pattern.

1.1.4.2 Cost Functions and Move Generation

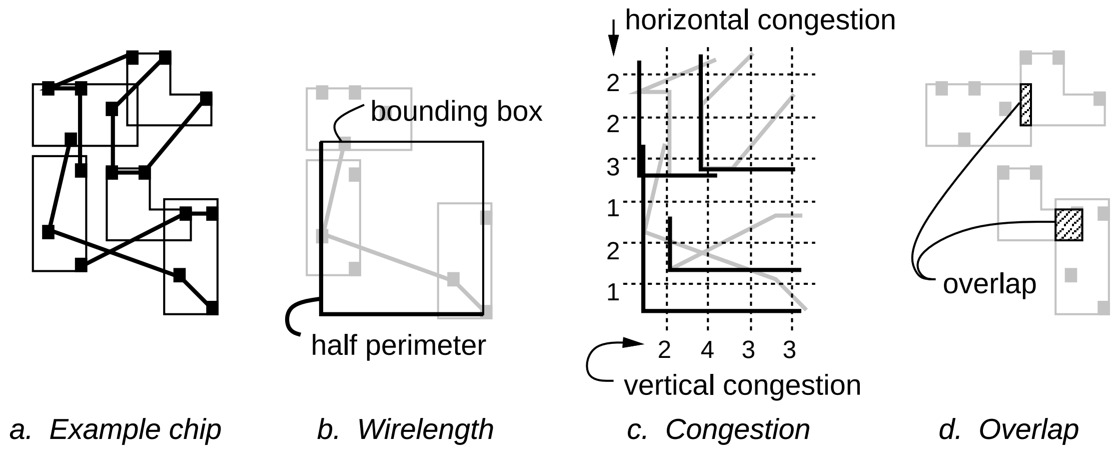


Figure 1.4: Components of Placement Cost Function

Figure 1.4a shows an example chip with five circuits. Each circuit contains pins represented by solid black squares. A wire connects pins on different circuits; this pin collection is a "net." The example has four nets. The wires usually have "Manhattan" orientation: they run horizontally or vertically, never at an angle.

Figure 1.4b shows how circuit placers often estimate the wire length of a net. A bounding-box is constructed around the net—this is the minimum rectangle containing the pins. The wire length to connect the pins is estimated at half the bounding-box perimeter. The half-perimeter is an approximation. §4.1.1 discusses this in detail.

If there are n nets, and each net i has a half-perimeter length of w_i, then the total wire length is $W = \sum_{i=1}^{n} w_i$. The total wire length is loosely related to a chip's speed. The length of the critical path would provide a better measure, but this is expensive to compute. I know of no placement programs that do it.

In VLSI chips, wires have a non-zero width which contributes to the chip's total area. Adding vertical wires to a chip causes the chip to widen. Adding horizontal wires to a chip causes it to heighten. This is "wire congestion." Figure 1.4c shows how wire congestion is estimated.

To obtain the horizontal congestion, divide the chip with uniformly-spaced horizontal cuts. Count the half-perimeters that cross each horizontal cut. This number is called a "crossing count." Suppose there are n horizontal cuts, and cut $i \in \{1, \ldots, n\}$ has a crossing count of h_i. Then the horizontal congestion is computed as $H = \sum_{i=1}^{n} h_i^2$.

To obtain the vertical congestion, follow the same procedure using vertical cuts.

The width added to a chip by vertical wires is a monotonic function of the horizontal congestion. The height is a monotonic function of the vertical congestion. These additions to the chip size increase its cost, its power consumption, and its total wire-length.

In the usual VLSI technologies, real circuits do not overlap. However, a common annealing hack allows overlap, but adds a high penalty in the cost-function. At high temperatures overlaps occur, but as the temperature lowers, the penalty becomes sufficiently overwhelming to eliminate them.

Figure 1.4d shows two overlaps. To compute overlap, choose fixed x and y units. The cost function counts the circuits co-existing in each location (x, y), say l_{xy}. The total overlap is then typically computed as $L = \sum_{x,y}[\min(0, l_{xy} - 1)]^2$.

More elaborate cost function components may be added. For example, pin density may be a concern: too many pins in one area may decrease the chip-yield or enlarge the chip, as does wire congestion. A system may allow a designer to specify capacitance or resistance limits for each wire.

Each cost function component is given a weight and summed. The cost of a placement s is $C(s) = c_w W(s) + c_v V(s) + c_h H(s) + c_l L(s)$.

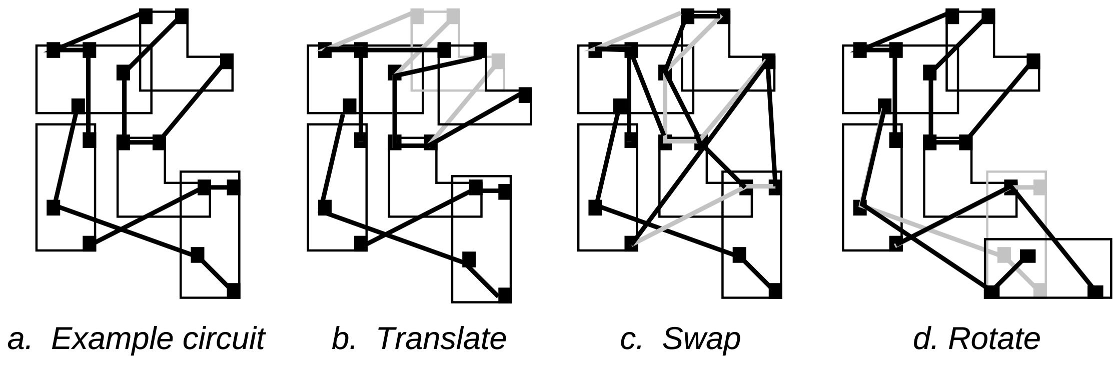


Figure 1.5: Generated Moves

Figure 1.5 shows the typical moves generated in a circuit placement program. In Figure 1.5b, the L-shaped circuit in the upper right corner is translated to a different location. In Figure 1.5c, the two L-shaped circuits are swapped, changing the nets attached to them (overlaps remain unchanged). In Figure 1.5d, the circuit in the lower right corner is rotated 90 degrees.

In short, circuit placers have substantial flexibility in how they compute cost-functions, and how they generate moves. The general structure of the circuit placement problem varies with gate-array, row-based, and macro-cell forms. It is hard to generalize about placement problems. To make matters worse, companies keep many interesting algorithms secret.

The problem richness and variety contributes to the wide use of simulated annealing for circuit placement. Few good algorithms are as tolerant of different cost-functions and move-generation functions as simulated annealing.

1.1.4.3 Experiences

A popular row-based placement and routing program, called TimberWolfSC, uses simulated annealing [SLS89]. In a benchmark held at the 1988 International Workshop on Placement and Routing, TimberWolfSC produced the smallest placement for the 3000-element *Primary2* chip—3% smaller than its nearest competitor. Moreover, it completed earlier than all other entrants. TimberWolfSC also routed *Primary2*; no other entrant completed that task. In most of the competitions given in the 1990 International Workshop on Layout Synthesis, TimberwolfSC 5.4 beat all other entrants [Koz90].

Placement requires substantial execution time. On a Sun 4/260, *Primary2* required approximately 3 hours to place with TimberwolfSC. Execution times for industrial simulated annealing runs have ranged from minutes to days on supercomputers.

1.2 Thermodynamic Annealing

There is no *fundamental* difference between a mechanical system and a thermodynamic one, for the name just expresses a difference of attitude. By a thermodynamic system we mean a system in which *there are so many relevant degrees of freedom that we cannot possibly keep track of all of them. J.R. Waldram*

You can view crystalizing as a combinatorial optimization problem: arrange the atoms in a material to minimize the total potential energy. Crystalizing is often performed with a cooling procedure called "annealing"; the theoretical modelling of this process, applicable to arbitrary combinatorial spaces, is called "simulated annealing."

Since work in simulated annealing sometimes appeals to intuitive extensions of the physical process, I will describe the physics of annealing. I call this "thermodynamic annealing" to distinguish it from the simulated annealing algorithm.

You can observe thermodynamic annealing from two perspectives: From a macroscopic perspective, observable properties of a material change with temperature and time. From a microscopic perspective, the kinetic energies, potential energies, and positions of individual molecules change.

1.2.1 Macroscopic Perspective

In thermodynamic annealing, you first melt a sample, then decrease the temperature slowly through its freezing point. This procedure results in a physical system with low total energy. Careful annealing can produce the lowest possible energy for a sample, e.g., a perfect crystalline lattice.

When a system is cooled rapidly, called *quenching*, the result typically contains many defects among partially-ordered domains.

The function $T(t)$, which relates time to temperature, is called the *temperature schedule*. Usually, $T(t)$ is monotonically decreasing.

A familiar example is silicon crystal manufacture. Annealing converts molten silicon to cylindrical crystals. Using silicon wafers sliced from these cylinders, engineers manufacture integrated circuits.

A material has reached thermal equilibrium when it has constant statistical distributions of its observable qualities, such as temperature, pressure, and internal structure. After fixing the temperature, perfect thermal equilibrium is guaranteed only after infinite time. Therefore, you can only hope to attain approximate thermal equilibrium, called *quasi-equilibrium*.

Temperature should be reduced slowly through *phase-transitions*, where significant structural changes occur. Condensing and freezing represent obvious phase-transitions. Less obvious phase-transitions occur: ice, for example, has several phase-transitions at different temperatures and pressures, each marked by a shift in molecular structure.

A system's specific heat, denoted by H, is the amount of energy absorbed per temperature change. H may vary with temperature. When the H is large, energy is usually being absorbed by radical structural changes in the system. Increases in specific heat often accompany phase-transitions. The specific heat is given by

$$H = \frac{\partial C}{\partial T}, \tag{1.1}$$

where ∂C is the change in equilibrium energy and ∂T is the change in temperature.

In finite time quasistatic processes, which are closely related to annealing processes, maximum order is obtained by proceeding at constant thermodynamic speed [NSA85]. The thermodynamic speed, v, is related to temperature and specific heat by

$$v = -\frac{\sqrt{H}}{T} \cdot \frac{dT}{dt}. \tag{1.2}$$

Thermodynamic speed v measures how long it takes to reach quasi-equilibrium.

Consider a system with constant specific heat regardless of temperature, such as a perfect gas. A temperature schedule which maintains quasi-equilibrium is

$$\ln|T(t)| = -\frac{v}{\sqrt{H}}\, t + a \tag{1.3}$$

or

$$T(t) = e^{-vt/\sqrt{H}+a}, \tag{1.4}$$

where a is a constant. This is precisely the temperature schedule used in many implementations of simulated annealing [KCV83], and in the self-affine (fractal) annealing examples analyzed in §4.6.

When H varies with T, maintaining quasi-equilibrium requires an adaptive temperature schedule. Adaptive schedules slow temperature reductions during phase-transitions.

1.2.2 Microscopic Perspective

Statistical mechanics represents the energy of a multi-body system using a state vector operator, called a Hamiltonian [Sin82]. In combinatorial optimization problems, the cost function has a comparable role.

Annealing seeks to reduce the total energy (effectively, the total potential energy) in a system of interacting microscopic states. The position and velocity of each pair of molecules, participating in the multi-body interaction expressions of a Hamiltonian, contribute to the system energy. Hamiltonians, like the cost functions of optimization problems, often exhibit the property that a particular microscopic state change contributes to bringing the total system energy to a local minimum, but not to the global minimum.

The Hamiltonian for the total energy of a thermodynamic system is shown in equation 1.5.

$$\mathcal{H}(p,x) = \sum_i p_i^2/2m_i + \phi(x_1, x_2, \ldots) \tag{1.5}$$

Here, p_i is the momentum vector, m_i is the mass, and x_i is the position of the ith particle. $\mathcal{H}(p,x)$ includes both the kinetic energy of each component, $p_i^2/2m_i$, and the total potential energy in the system, $\phi(x_1, x_2, \ldots)$. Physical configurations S_i of the system have an associated energy C_i, which is an eigenvalue of $\mathcal{H}(p,x)$.

By the first law of thermodynamics, total energy in a closed system is a constant. Therefore $\mathcal{H}(p,x)$ is a constant.

At the microscopic level, an individual particle's energy level exhibits a probability distribution dependent on the current temperature T. At thermal equilibrium, the probability that a component will be in its ith quantum state is shown in Equation 1.6, the *Boltzmann probability distribution.*

$$\boldsymbol{\pi}_i = \frac{e^{-C_i/kT}}{\sum_j e^{-C_j/kT}}, \tag{1.6}$$

where k is Planck's constant and C_i is the energy in state S_i.

Note that the values of C_i depend on the positions of the particles in the system, according to the $\phi(x_1, x_2, \ldots)$ term of Equation 1.5. That dependency makes exact computer simulations of the microscopic states in large systems impractical.

An approximate equilibrium simulation algorithm, which has achieved success, is called Monte-Carlo simulation or the Metropolis method [MRR53]. The algorithm works as follows: There are n interacting particles in a two-dimensional space. The state of the system is represented by $s = (s_0, \ldots, s_{n-1})$, where s_i is the position vector of particle i. Place these particles in any configuration. Now move each particle in succession according to this method:

$$s'_i = s_i + \alpha\rho \tag{1.7}$$

where s'_i is the trial position, s_i is the starting position, α is the maximum allowed displacement per move, and ρ is a pseudo-random vector uniformly distributed about the interval $[-1, 1] \times [-1, 1]$.

Using the Hamiltonian, calculate the change in energy, Δ, which would result from moving particle i from x_i to x'_i. If $\Delta \leq 0$, accept the move by setting x_i to x'_i.

If $\Delta > 0$, accept the move with probability $e^{-\Delta/(kT)}$. If the move is not accepted, x_i remains as before. Then consider particle $i + 1 \bmod N$.

Metropolis showed that by repeated application of this procedure, the probability distributions for individual particles converge to the Boltzmann distribution (1.6), greatly simplifying equilibrium computations.

Using Markov analysis with the Metropolis method, you can show that the system escapes local energy minima, expressed in the Hamiltonian, by hill-climbing, if the system is brought close to thermal equilibrium before reducing the temperature. From a microscopic perspective, that is the goal of the annealing process.

Convergence cannot be guaranteed unless thermal equilibrium is reached—but one cannot guarantee thermal equilibrium in finite time. This also holds for simulated annealing. To think about convergence behavior in a Markov sense, consider a discretized state space, where N particles can occupy M locations. If the particles are distinguishable and can share locations, there are M^N states.

A single pass of the Metropolis algorithm through all the particles can be represented by a constant state-space transition matrix A, when the temperature is held fixed. The steady state probability vector $\boldsymbol{\pi}$, in

$$\boldsymbol{\pi} = \boldsymbol{\pi} A = \boldsymbol{\pi}^{(0)}\, A\, A\, \ldots, \tag{1.8}$$

where $\boldsymbol{\pi}^{(0)}$ is the initial state distribution, may not be reached in finite time. But this steady state probability is precisely what is meant by thermal equilibrium, so thermal equilibrium may not be reached in finite time.

If A is ergodic,

$$\boldsymbol{\pi} = \lim_{n\to\infty} \boldsymbol{\pi}^{(0)} A^n, \tag{1.9}$$

where n is the number of trials [Kle75]. So with longer equilibrating times, you more closely approximate thermal equilibrium.

1.3 Alternatives to Simulated Annealing

Simulated annealing is an extremely general algorithm, as the variety of application problems illustrates. Other general algorithms exist to solve the same problems.

Genetic algorithms [CHM88, Dav87] start with a random population of solutions, create new populations with "mixing rules," and destroy bad members based on a "fitness function" (similar to simulated annealing's cost function). At the end of the algorithm you choose the best solution in the population. Genetic algorithms do not lend themselves to theoretical understanding: there are no guarantees, even for exponential execution time, except for specific problems [GS87]. However, parallel genetic algorithms have been used for some traditional annealing problems, including circuit placement [KB89].

Other general algorithms include branch-and-bound and related exhaustive techniques, which are guaranteed to find an optimal answer, though not necessarily in an acceptable time [HS78]. Comparison of branch and bound against simulated annealing for network optimization is described in various unpublished technical reports. Branch and bound on realistic networks failed to produce results in acceptable times, while simulated annealing succeeded [Sti94].

Some algorithms perform better than simulated annealing at specific tasks, such as relaxed linear programming for wire-routing [Rag92], min-cut for graph partitioning [LD88, JAM89], and the Karmarkar-Karp heuristic for number partitioning. Other special-purpose algorithms maintain an uneasy superiority, with new move-generators or temperature schedules bringing simulated annealing back into the lead, and then some new modification to the special-purpose algorithm beating it again, etc. This has been the case for the satisfiability problem [Spe95]. Banerjee has documented many parallel algorithms specifically relevant to VLSI problems [Ban94].

Simulated annealing appears to be well-entrenched, if only because it tolerates modified cost-functions with minimal disruption.

1.4 Dissertation Outline

Allowing cost-function inaccuracies in simulated annealing can improve its speed. This dissertation addresses two open analytic problems: what cost-function accuracy is required for an acceptable result? Does the execution time have to change to compensate for these inaccuracies?

This chapter discussed simulated annealing and circuit placement. Simulated annealing is a combinatorial optimization algorithm—a modification of the greedy algorithm. Thermodynamic annealing is the process of cooling a material through its freezing point to regularize its internal structure. Understanding thermodynamic annealing helps explain simulated annealing.

Simulated annealing applies to a wide variety of applications. Its tolerance of different cost-functions and move-generation methods contributes to its common use for circuit placement. Circuit placement has so many variations that it is difficult to create a general placement algorithm except with something broadly applicable, like annealing.

Chapter 2 surveys the field of parallel simulated annealing, arranging different parallel techniques into a taxonomy. It concludes that asynchronous techniques show good performance, but errors must be adequately controlled. That chapter appeared previously as a chapter of *Emergent Computation* [Gre91b].

Chapter 3 describes my experiments using a parallel asynchronous algorithm for gate-array placement. I measured cost-function errors, "mobility," and outcome quality. Not surprisingly, larger errors worsened the outcome. The changes I made in mobility seem to have had less effect than the errors I introduced. Parts of this chapter appeared as a paper in the *Proceedings of the 1989 International Conference on Supercomputing* [GD89].

Chapter 4 presents my efforts to analytically resolve questions raised by my experiments. Though I started this investigation with parallel annealing in mind, I realized later that errors abound in common sequential annealing applications. I show how they arise in both sequential and parallel situations. I assumed that errors could be expressed in two forms: either as "range-errors," where the cost-function can return a value in a fixed range about the true cost, or as "Gaussian errors," where the cost-function is a Gaussian function with the true cost as its mean.

Errors affect constant-temperature properties of simulated annealing, including the equilibrium cost and the conductance. I show how these properties are affected by both error forms. The key problem for me, however, was discovering how errors affect the speed of normal annealing applications, which do not operate at constant-temperature.

On general combinatorial problems, simulated annealing has been proven to operate in exponential time. For practitioners this is hardly exciting: all combinatorial optimization problems can be performed in exponential time by simple enumeration. However, since a large body of theoretical work relies on this result, and since it applies to any combinatorial problem, I investigated. Using the general (and slow) $T(t) = c/\log t$ temperature schedule on any space satisfying the usual constraints, I show that annealing will converge if range-errors are confined to a constant factor of temperature.

Most simulated annealing temperature schedules, however, are similar to the constant specific-heat form (1.3), namely $T(t) = e^{-ct+d}$, or as it is usually described in computer science, $T(t) = c\ T(t-1)$. This is the "geometric temperature schedule." Few have attempted to create analytic models for problems that can be annealed with geometric schedules. One is Greg Sorkin, who modeled these problems with self-affine functions (fractals) [Sor92].

Although it remains to be seen whether fractals are a good models for typical annealing problems, they provided the closest analytic model I knew that resembled common annealing problems, such as circuit placement. I show that in fractals, confining range-errors to a constant factor of the temperature, and using a geometric temperature schedule extended to account for conductance and quality changes, will guarantee the same quality outcome as without errors.

The range-error equilibrium results in Chapter 4 first appeared in *1990 Proceedings of the International Workshop on Layout Synthesis* [Gre90a]. The range-error and Gaussian conductance results first appeared in [Gre91a]. The analysis of $1/\log t$-based schedules first appeared in *1991 Proceedings of the IMACS International Congress on Mathematics and Computer Science* [Gre91c]. The rest has not been previously published.

Chapter 5 presents the conclusions that can be drawn from this work. My practical results apply to self-affine cost-functions, but in most cases it is difficult to determine whether a cost-function is self-affine. Sorkin presents some techniques for analyzing these spaces in [Sor92], but these techniques are often difficult and inconclusive. As with almost every annealing theory applied in practice, annealers should use my results as a guide rather than a mandate.

Errors appear in many annealing applications, either from a cost-function approximation or from parallelism. When errors appear, annealers can obtain the same quality results they would obtain otherwise, if they establish and limit the errors according to the results I present. Annealers may choose to accept lower outcome quality rather than finding a way to limit errors. If so, they can use these results to help predict the outcome quality.

CHAPTER 2

Parallel Techniques

Since a new state contains modifications to the previous state, simulated annealing is often considered an inherently sequential process. However, researchers have eliminated some sequential dependencies, and have developed several parallel annealing techniques. To categorize these algorithms, I ask several questions:

1. How is the state space divided among the processors?
2. Does the state generator for the parallel algorithm produce the same neighborhood as the sequential algorithm? How are states generated?
3. Can moves made by one processor cause cost-function calculation errors in another processor? Are there mechanisms to control these errors?
4. What is the speedup? How does the final cost vary with the number of processors? How fast is the algorithm, when compared to an optimized sequential program?

I will show that the efficiency of parallel processing is a mixed bag. Some experiments show a *decrease* in speed as processors are increased. This is caused by increased interprocessor communication and synchronization costs. Others show "superlinear speedup," meaning that the speed per processor increases when processors are added.

When considering published speedup comparisons in simulated annealing, proceed cautiously, particularly when superlinear speedups appear. Simulated annealing researchers frequently see this suspicious property.

Three causes explain most superlinear speedup observations. First, changes to state generation wrought by parallelism can improve annealing speed or quality [GD89]. If this happens, one can fix the sequential algorithm by mimicking the properties of the parallel version [JB87, FLW86]. Second, a speed increase might come with a solution quality decrease [DKN87]. That property holds for sequential annealing, as well [Lam88]. Third, annealing experimenters often begin with an optimal initial state, assuming that high-temperature randomization will annihilate the advantage. But if the parallel implementation degrades state-space exploration, high-temperature may not totally randomize the state: the parallel program, then, more quickly yields a better solution [BB88].

Knowledge of such pitfalls can help avoid problems. Superlinear speedup in parallel algorithms, such as parallel simulated annealing, should raise a red flag: altered state exploration, degraded results, or inappropriate initial conditions may accompany it.

I found only one comparison of parallel and sequential annealing algorithms which controlled the outcome quality [CRS87] by changing the temperature schedule and move-generation functions. As I will show in Chapter 4, an algorithm can trivially improve its speed by producing a lower-quality result. Even recent comparisons fail to account for this [KCP94]. Because experiments have been so poorly controlled, this chapter is more qualitative than quantitative.

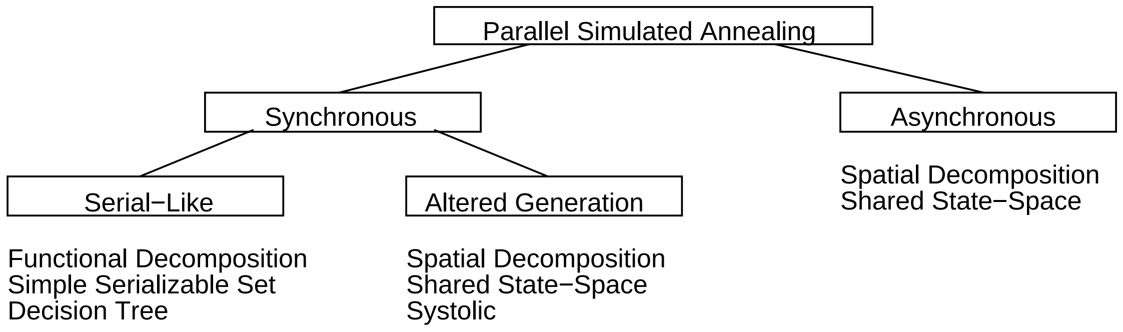


Figure 2.1: Parallel Simulated Annealing Taxonomy

I have categorized parallel simulated annealing techniques in a taxonomy of three major classes shown in Figure 2.1: serial-like, altered generation, and asynchronous. Call an algorithm *synchronous* if adequate synchronization ensures that cost function calculations are the same as those in a similar sequential algorithm. Two major categories, *serial-like* and *altered generation*, are synchronous algorithms. *Serial-like convergence* algorithms preserve the convergence properties of sequential annealing. *Altered generation* algorithms modify state generation, but compute the same cost function. *Asynchronous* algorithms eliminate some synchronization and allow errors to get a better speedup, possibly causing reduced outcome quality.

Table 2.1 shows the criteria used to distinguish parallel annealing categories, comparing them to accurate serial annealing.

Category	Cost	Generate
Serial-Like	$= C$	$= G$
Altered Generation	$= C$	$\neq G$
Asynchronous	$\neq C$	?
Serial, Estimated Cost	$\neq C$	$= G$

Table 2.1: Comparison with Accurate-Cost Serial Annealing

Each category makes a trade-off between cost-function accuracy, state generation, parallelism or communication overhead.

2.1 Serial-Like Algorithms

Three synchronous parallel algorithms preserve the convergence properties of sequential simulated annealing: *functional decomposition*, *simple serializable set*, and *decision tree decomposition*. These are *serial-like* algorithms.

2.1.1 Functional Decomposition

Functional decomposition algorithms exploit parallelism in the cost-function f. In the virtual design topology problem, for example, the cost function must find the shortest paths in a graph. One program computes that expensive cost function in parallel, but leaves the sequential annealing loop intact [BG89a]. Published reports provide no speedup information.

Another program evaluates the cost function for VLSI circuit placement in parallel [KR87]. Simultaneously, an additional processor selects the next state. Figure 2.2, Algorithm FD, shows the details.

```
1.   $m' \leftarrow$ select random state;
2.   loop for $i \leftarrow 0$ to $\infty$
3.       $m \leftarrow m'$;
4.       parallel block begin
5.           $m' \leftarrow$ generate( $m$ );
6.           $E_0 \leftarrow$ block-length-penalty( $m$ );
7.           $E_{1,0} \leftarrow$ overlap for affected circuit $c_0$ before move;
8.           $\cdots E_{1,j} \leftarrow$ overlap for affected circuit $c_j$ before move;
9.           $E_{2,0} \leftarrow$ overlap for affected circuit $c_0$ after move;
10.          $\cdots E_{2,j} \leftarrow$ overlap for affected circuit $c_j$ after move;
11.          $E_{3,0} \leftarrow$ length change for affected wire $w_0$;
12.          $\cdots E_{3,k} \leftarrow$ length change for affected wire $w_k$;
13.          end parallel block;
14.      $\Delta \leftarrow E_0 + (E_{1,0} + \ldots + E_{1,j}) - (E_{2,0} + \ldots + E_{2,j}) + (E_{3,0} + \ldots + E_{3,k})$;
15.      if accept( $\Delta, T$ ) then
16.          parallel block begin
17.              update overlap values;
18.              update blocks and circuits;
19.              update wire $w_0$;
20.              $\cdots$ update wire $w_k$;
21.              end parallel block;
22.          end if;
23.      recompute $T$, evaluate stop criteria, etc.
24.      end loop;
```

Figure 2.2: Algorithm FD, Functional Decomposition for VLSI Placement

One can obtain only a limited speedup from Algorithm FD. Ideally, the parallel section from line 4 to line 13 dominates the computation, each process executes in uniform time, and communication requires zero time. One can then extract a maximum speedup of $1 + 2j + k$, where j is the average circuits affected per move, and k is the average wires affected per move. Researchers estimate a speedup limitation of 10, based on experience with the VLSI placement program TimberWolfSC [SLS89].

Since cost-function calculations often contain only fine-grain parallelism, communication and synchronization overhead can dominate a functional decomposition algorithm. Load-balancing poses another difficulty. Both factors degrade the maximum speedup, making functional decomposition inappropriate for many applications.

2.1.2 Simple Serializable Set

If a collection of moves affect *independent* state variables, distinct processors can independently compute each Δ without communicating. Call this a "serializable set"—the moves can be concluded in any order, and the result will be the same. The simplest is a collection of rejected moves: the order is irrelevant, the outcome is always the starting state.

The *simple serializable set* algorithm exploits that property [KR87]. At low annealing temperatures, the acceptance rate (the ratio of accepted states to tried moves) is often very low. If processors compete to generate one accepted state, most will generate rejected moves. All the rejected moves and one accepted move can be executed in parallel.

Figure 2.3, Algorithm SSS, shows this technique [BAM88]. P processors grab the current state in line 5. Each processor generates a new state at line 7. If the new state is accepted (line 8) *and* the old state has not been altered by another processor (line 10), the move is made. Otherwise the move is discarded.

If the acceptance rate at temperature T is $\alpha(T)$, then the maximum speedup of

```
1.   shared variable $s$, semaphore $sema$;
         $\cdots$
2.   parallel loop for $i \leftarrow 1$ to $P$;
3.       loop for $j \leftarrow 0$ to $\infty$
4.           wait( $sema$ );
5.           $s_{\text{old}} \leftarrow s$;
6.           signal( $sema$ );
7.           $\langle s', \Delta \rangle \leftarrow$ generate( $s_{\text{old}}$ );
8.           if accept( $\Delta, T$ ) then
9.               wait( $sema$ );
10.              if $s_{\text{old}} = s$ then
11.                  $s \leftarrow s'$;
12.                  $T \leftarrow$ new $T$;
13.                  end if;
14.              signal( $sema$ );
15.              end if;
16.          change $T$, evaluate stop criterion, etc.
17.          end loop;
18.      end parallel loop;
```

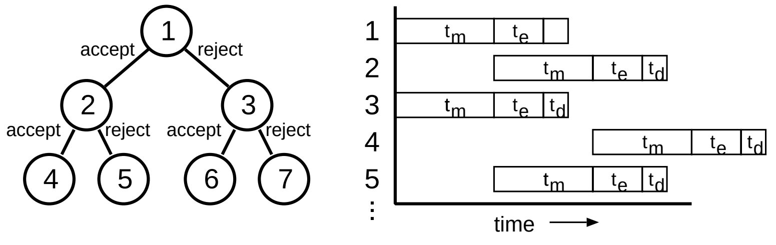
Figure 2.3: Algorithm SSS. Simple Serializable Set Algorithm

this algorithm, ignoring communication and synchronization costs, is $1/\alpha(T)$. At high temperatures, where the acceptance rate is close to 1, the algorithm provides little or no benefit. But since most annealing schedules spend a majority of time at low temperatures, Algorithm SSS can improve overall performance.

Algorithm SSS has limitations. Some recent annealing schedules maintain $\alpha(T)$ at relatively high values, throughout the temperature range, by adjusting the generation function. Lam's schedule, for instance, keeps $\alpha(T)$ close to 0.44 [Lam88]. With that schedule, Algorithm SSS provides a maximum speedup of approximately 2.3, regardless of the number of processors.

2.1.3 Decision Tree Decomposition

A third serial-like algorithm, called decision tree decomposition, exploits parallelism in making accept-reject decisions [CEF88]. Consider the tree shown in Figure 2.4a. If we assign a processor to each vertex, cost evaluation for each suggested move can proceed simultaneously. Since a sequence of moves might be interdependent (i.e., not serializable), however, we *generate* the states in sequence.



a. Annealing Decision Tree b. Functional Dependence

Figure 2.4: Decision Tree Decomposition

Figure 2.4b shows vertex dependencies. A vertex generates a move in time t_m, evaluates the cost in time t_e, and decides whether to accept in time t_d. Note that vertex 2 cannot begin generating a move until vertex 1 generates its move and sends it to vertex 2.

A simple implementation results in predicted speedups of $\log_2 P$, where P is the number of processors. By skewing tree evaluation toward the left when $\alpha(T) \geq 0.5$, and toward the right when $\alpha(T) < 0.5$, researchers predict a maximum speedup of $(P + \log_2 P)/2$ [CEF88].

In numeric simulations, however, the speedups fall flat. With 30 processors and $t_m = 16t_e$, the estimated speedup was 4.7. Unfortunately, in VLSI placement problems $t_m \gg t_e$, and in traveling salesman problems $t_m \approx t_e$. Setting t_m close to t_e leads to a speedup of less than 2.5 on 30 processors. As a result, this approach holds little promise for such applications. This was confirmed by later results [Wit90].

2.2 Altered Generation Algorithms

Even if a parallel annealing algorithm computes cost-functions exactly, it may not mimic the statistical properties of a sequential implementation. Often, state generation must be modified to reduce inter-processor communication. These *altered generation* methods change the pattern of state space exploration, and thus change the expected solution quality and execution time.

2.2.1 Spatial Decomposition

In spatial decomposition techniques, the algorithm distributes state variables among the processors, and transmits variable updates between processors as new states are accepted. Spatial decomposition techniques are typically implemented on message-passing multiprocessors.

In *synchronous* decomposition, processors must either coordinate move generation and communication to avoid errors, or not generate moves that affect other processors' state variables. These two techniques are *cooperating processors* and *independent processors*, respectively.

2.2.1.1 Cooperating Processors

A cooperating processor algorithm disjointly partitions state variables over the processors. A processor that generates a new state notifies other affected processors. Then, those processors synchronously evaluate and update the state. If a proposed move could interfere with another in-progress move, the proposed move is either delayed or abandoned.

One such program minimizes the number of routing channels (the slots where wires lie) for a VLSI circuit [BB88]. The cost is the total number of routing channels that contain at least one wire; two or more wires can share the same routing channel, if they don't overlap.

The program first partitions a set of routing channels across the processors of an iPSC/2 Hypercube; that processor assignment henceforth remains fixed. Processors proceed in a lockstep communication pattern. At each step, all processors are divided into master-slave pairs. The master processor randomly decides among four move classes:

Intra-displace The master and slave each move a wire to another channel in the same processor.

Inter-displace The master processor moves one of its wires to a channel in the slave processor.

Intra-exchange The master and slave each swap two wires in the same processor.

Inter-exchange The master swaps a wire from one of its channels with a wire in the slave.

Experiments indicate superlinear speedups, from 2.7 on 2 processors to 17.7 on 16 processors. These apparently stem from a nearly-optimal initial state and more-constrained parallel moves, making the reported speedups untenable. However, the decomposition method itself is sound.

2.2.1.2 Independent Processors

In independent processor techniques, each processor generates state changes which affect only its own variables. Under this system, a fixed assignment of state variables to processors would limit state-space exploration, and produce an inferior result. The technique requires periodic state variable redistribution.

One such technique optimizes traveling salesman problems [AC89]. A traveling salesman problem (TSP) consists of a collection of cities and their planar coordinates. A tour that visits each city and returns to the starting point forms a solution; the solution cost is its total length.

Construct an initial state by putting the cities into a random sequence: the tour visits each in order and returns to the first city. Stretch this string of cities out like a rubber band, and evenly divide the two parallel tracks among the processors, as shown in Figure 2.5a. The state variables consist of the endpoints of each two-city segment.

Each processor anneals the two paths in its section by swapping corresponding

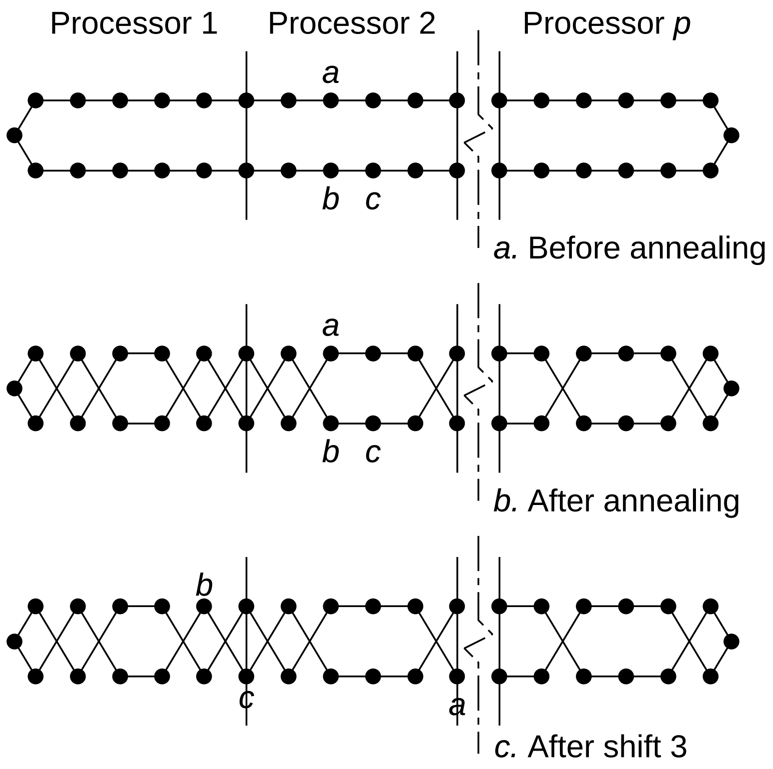


Figure 2.5: Rubber Band TSP Algorithm

endpoints, as shown in Figure 2.5b. After a fixed number of tries in each processor, the total path length is computed, and a new temperature and a shift count are chosen. Each processor then shifts the path attached to its top left node to the left, and the path attached to its bottom right node to the right, by the shift count, as shown in Figure 2.5c. This operation redistributes the state variables, ensuring that the whole state space is explored. Annealing continues until it satisfies the stopping criterion.

In one experiment, the 30 processor versus 2 processor speedup ranged from about 8 for a 243 city TSP, to 9.5 for a 1203 city TSP. Unfortunately, a single processor example was not discussed. The paper does not show final costs; final cost probably increases as the number of processors increases. Other spatial decomposition techniques exhibit similar behavior and speedups [FKO85, DN86].

2.2.2 Shared State-Space

Shared state-space algorithms make simultaneous, independent moves on a shared-memory state-space: no cost-function errors can occur.

One such algorithm optimizes VLSI gate-array placement [DKN87]. Changes in the state generation function, resulting from the locking of both circuits and wires, caused poor convergence. Maximum speedup was 7.1 for 16 simulated RP3 processors, solving a uniform 9×9 grid problem. Improving the parallel algorithm's convergence would reduce its speedup below 7.1.

A similar algorithm for minimizing the equal partition cut-set (see §2.3.2) obtained a dismal speedup close to 1 on 16 processors [Dur89].

Another shared state-space algorithm constructs conflict-free course timetables [Abr89]. Before evaluating a move, the algorithm must lock the instructors, courses and rooms for two time periods, then swap them. If the locks conflict with an in-progress move, the algorithm abandons them and generates another move. Speedup was compared against an optimized sequential algorithm. With 8 processors, a speedup of 3.2 was obtained in scheduling 100 class periods, while 6.8 was obtained in scheduling 2252 class periods.

2.2.3 Systolic

The systolic algorithm exploits the property that simulated annealing brings a thermodynamic system toward the Boltzmann distribution [ABH86, MRR53].

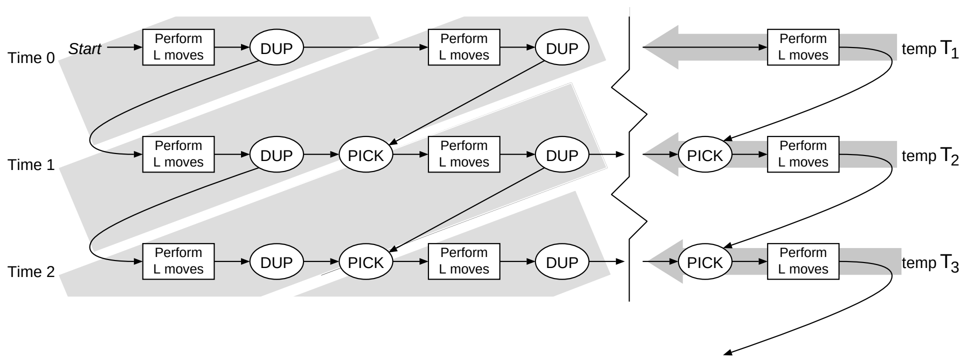


Figure 2.6: Systolic Algorithm

Suppose there are P processors, and the algorithm maintains the same temperature for a sequence of N generated states. We would like to divide these moves into P subchains of length $L = P/N$, and execute them on different processors. Figure 2.6 shows a corresponding data flow graph for this decomposition.

At any PICK node on processor p, the algorithm must decide between state $s_{(n-1,p)}$ computed by processor p at temperature T_{n-1}, and state $s_{(n,p-1)}$ computed by processor $p-1$ at temperature T_n. It makes the choice according to the Boltzmann distribution: the relative probability of picking $s_{(n-1,p)}$ is

$$\rho_0 = \frac{1}{Z(T_{n-1})} e^{[f(s|)-f(s_{(n-1,p)})]/T_{n-1}} \tag{2.1}$$

and the relative probability of picking $s_{(n,p-1)}$ is

$$\rho_1 = \frac{1}{Z(T_n)} e^{[f(s\downarrow) - f(s_{(n,p-1)})]/T_n} \tag{2.2}$$

where S is the entire state space and $s\downarrow$ is a minimum cost state. $Z(T)$ is the *partition function* over the state space, namely

$$Z(T) = \sum_{s \in S} e^{-f(s)/T} \tag{2.3}$$

The PICK node then selects $s_{(n-1,p)}$ and $s_{(n,p-1)}$ with probabilities

$$p(s_{(n-1,p)}) = \frac{\rho_0}{\rho_0 + \rho_1}, \qquad p(s_{(n,p-1)}) = \frac{\rho_1}{\rho_0 + \rho_1} \tag{2.4}$$

If you don't know the minimum cost, you can't evaluate $f(s\downarrow)$. A lower bound must suffice as an approximation. Choosing a lower bound far from the minimum cost will increase execution time or decrease solution quality [Lam88].

The partition function, Z, requires the evaluation of every state configuration. The number of state configurations is typically exponential in the number of state variables, making exact computation of Z unreasonable.

Instead, the systolic method uses an approximate Z. In the temperature regime where the exponential function dominates, ρ_0 and ρ_1 are almost completely determined by their numerators in Equations 2.1 and 2.2. The influence of $Z(T)$ thus becomes small, and it can be approximated by the Gaussian distribution.

How does the algorithm perform? With 8 processors operating on a 15×15 uniform grid of cities, the systolic algorithm obtained a mean path-length of 230, at a speedup of about 6.2, while the sequential algorithm obtained an average of about 228.5. Accounting for the less optimal parallel result, the effective speedup is something less than 6.2.

2.3 Asynchronous Algorithms

Without sufficient synchronization, different processors can simultaneously read and alter dependent state-variables, causing cost-function calculation errors. Such algorithms are *asynchronous.* Imprecise cost-function evaluation accelerates *sequential* simulated annealing under certain conditions [GM89, Gro89]; a similar effect accompanies asynchronous parallel simulated annealing.

These algorithms use a method related to chaotic relaxation—processors operate on outdated information [CM69]. Since simulated annealing randomly selects hill-climbing moves, it can tolerate some error; under the right conditions, annealing algorithms can evaluate the cost using old state information, but still converge to a reasonable solution. This property holds for genetic algorithms, as well [JG87].

Error tolerance provides a great advantage in multiprocessing: when processors independently operate on different parts of the problem, they need not synchronously update other processors. A processor can save several changes, then send a single block to the other processors. The processor sends less control information and compresses multiple changes to a state variable into one, reducing total communication traffic. In addition, if updates can occur out-of-order, synchronization operations are reduced. Asynchronous algorithms require a minimum synchronization: two processors acting independently must not cause the state to become inconsistent with the original problem.

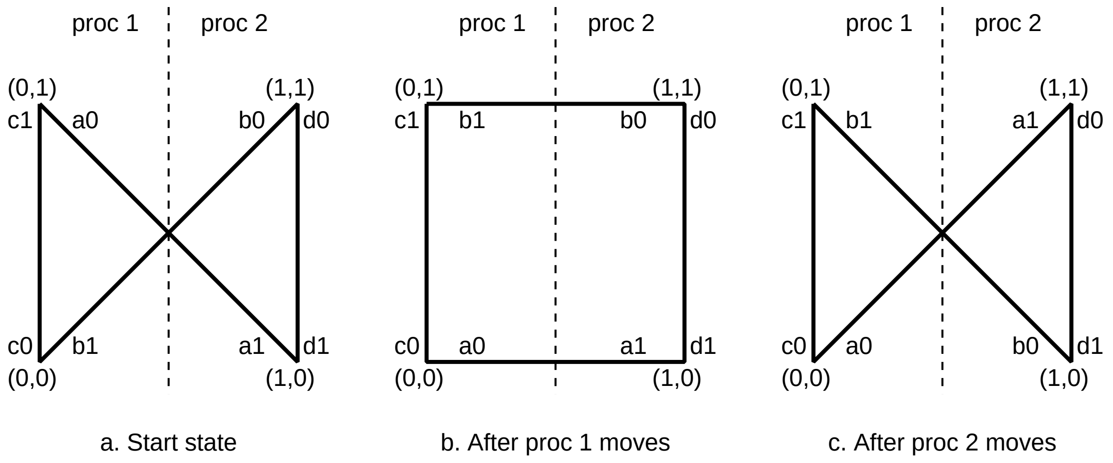


Figure 2.7: Cost-Function Errors in Spatial Decomposition

Figure 2.7 shows how errors arise in a spatially decomposed traveling salesman problem. In the figure, variables a_0 and a_1 denote the endpoints of edge a. The simulated annealing algorithm swaps endpoints to generate a new state. The algorithm partitions the cities over two processors. A processor may only swap endpoints that point to its vertices, ensuring problem consistency. However, to reduce synchronization time, processors do not lock edges while they evaluate the cost-function.

While processor 0 considers swapping endpoint a_0 with b_1, processor 1 considers swapping endpoint a_1 with b_0. Processor 0 sees a path-length change for its move of $\Delta = 2\,(1 - \sqrt{2}) \approx -0.818$. Processor 1 also sees $\Delta \approx -0.818$, for its move.

Processor 0 makes its move, by swapping a_0 and b_1. Now, processor 1 makes its move, thinking its $\Delta \approx -0.818$ (a good move) when the effect is $\Delta \approx +0.818$

(a bad move). At low temperatures, the error will degrade the final result unless corrected by a later move. So, simulated annealing does not have an unlimited tolerance for errors.

Cost-function errors usually degrade convergence quality, when all other factors are fixed: note the contrast with altered state generation. For example, experiments have shown that VLSI placement quality decreases as errors increase [GD89, JD88].

Several authors have conjectured that annealing properties might be preserved when the errors are small. Experimental evidence bears this out [GD89, DKN87, BJ86, Gro86, RSV88]. However, you can easily construct a problem which converges well under sequential simulated annealing, but will *likely* converge to a bad local minimum in an asynchronous program.

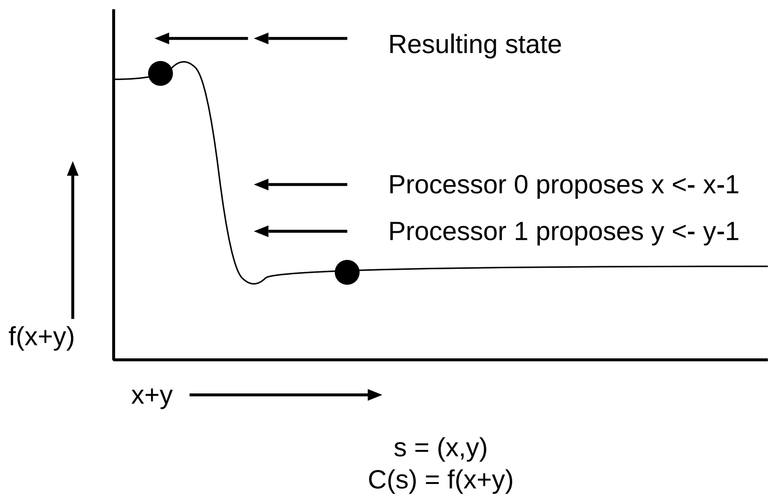


Figure 2.8: Errors Can Cause Annealing Failure

Consider a system with two state variables x and y, so some state $s = \langle x, y \rangle \in$

S. Let the cost-function be $f(x+y)$, where f is the curve in Figure 2.8. Now put x and y on two separate processors. Each processor proposes a move: processor 0 generates $x \leftarrow x-1$, while processor 1 generates $y \leftarrow y-1$. In both cases, $\Delta < 0$, so each move will be accepted.

The cost-function error causes the state to jump to a high local minimum. At low temperatures, the annealing algorithm probably will not escape this trap.

2.3.1 Asynchronous Spatial Decomposition

Asynchronous spatial decomposition methods, like the synchronous methods in §2.2.1, partition state variables across different processors. However, in asynchronous algorithms each processor also maintains read-only copies of state variables from other partitions.

When a processor evaluates a new state, it uses only local copies of state variables. In some programs, when a move is accepted the new state information is immediately sent to other processors [BJ86]. In other programs, a processor completes a fixed number of tries, called a "stream," before transmitting the modifications [GD89, JD88]. Longer streams increase the execution-to-communication ratio, gaining a speedup, but they also increase calculation errors, reducing the solution quality.

2.3.1.1 Clustered Decomposition

The clustered decomposition technique solves two simultaneous optimization problems: the specified target problem and assigning the state variables to processors.

In one example, the target problem is VLSI macro-cell circuit placement, and the assignment problem is circuit partitioning [CRS87]. Overlap penalties in the VLSI cost-function generate the largest errors—when two circuits owned by different processors are moved to the same empty location, neither processor will see an overlap, but the overlap error might be huge. This leads to a clustering problem: divide state variables (circuits) equally among the processors, while putting dependent variables (adjacent or connected circuits) on the same processor.

Compute the assignment cost-function, for VLSI macro-cell placement, as follows. Let A be the set of circuits, let $\mathbf{A} = \{A_1, \ldots, A_P\}$ be the partition of A over P processors, let $\overline{a}$ be a circuit's vector center and let $|a|$ be its scalar area. For each processor p, you can compute the center of gravity X_p of its partition A_p

$$X_p = \frac{1}{\sum_{a \in A_p} |a|} \sum_{a \in A_p} \overline{a} \cdot |a| \tag{2.5}$$

and its inertial moment

$$\Gamma_p = \sum_{a \in A_p} \|\overline{a} - X_p\|^2 \cdot |a| \tag{2.6}$$

The assignment cost-function for partition $\mathbf{A}$ is

$$f_a(\mathbf{A}) = w_a \cdot \sum_{i=1}^{P} \Gamma_i \tag{2.7}$$

where w_a is a weighting factor.

Experiments used the same temperature for both partitioning and placement: independent temperature schedules would probably improve the result. A 30 circuit problem, running on an 8 processor, shared-memory Sequent 8000, reached a speedup of 6.4 against the same algorithm running on a single processor.

Clustering improved convergence. Express a result's *excess cost* as $E_{\text{final}} - E_{\text{min}}$, where E_{final} is the result's cost, and E_{min} is the best solution known (presumably close to optimal). Clustering reduced the excess cost in a 101 circuit, 265 wire problem by about 15%.

2.3.1.2 Rectangular Decomposition

A simpler approach, rectangular decomposition, tries to accomplish the same goals. It divides the grid of a gate-array placement problem into disjoint rectangles, then shifts the boundaries after each stream [GD89]. At low temperatures, interdependent state variables typically share a rectangle.

Different variants were tried: placing restrictions on the minimum width of a rectangle and "fuzzing" the rectangle boundaries. All rectangular decomposition schemes produced small errors and converged close to the minimum. In contrast, random assignment of state variables to processors, on identical problems, produced greater errors and converged to a much higher final cost [JD88].

One rectangular decomposition experiment fixed the number of generated states in a block of PN tries, where N is the stream length and P is the number of processors. Figure 2.9 displays the resulting errors. The error value is $\epsilon_{\mathbf{s}} = |\Delta - \Delta_P|$, where Δ is the actual cost change after completion of a stream, and Δ_P is the sum of the apparent cost changes observed by the processors. Increasing P also increases $\epsilon_{\mathbf{s}}$, as one might expect.

Figure 2.9: Spatial Decomposition, 16 Tries per Block

A more detailed discussion of these experiments appears in Chapter 3.

2.3.2 Asynchronous Shared State-Space

Asynchronous shared state-space algorithms keep all state variables in shared memory. Processors competitively lock state variables, make moves, and unlock. Unlike synchronous algorithms, processors need not lock all affected state variables;

they need only lock those variables required for problem consistency.

One experiment compared synchronous and asynchronous algorithms for VLSI gate-array placement [DKN87]. Under a simulated RP3 environment, three methods were tried. Method A, a synchronous shared state-space algorithm, is described in §2.2.2. Each processor locked two circuits and any attached wires before attempting a swap.

In method B1, an asynchronous algorithm, processors lock only the two circuits in the proposed move, and calculate the new wire length with possibly-changing information. Each processor maintains a local copy of the histogram variables (the wire-congestion information described in §1.1.4.2). A move updates only the local histogram; at the completion of a stream, each processor corrects its histogram with global state information.

Method B2 operates like B1, except that it never corrects the local histograms. Thus, histogram information becomes progressively outdated as the temperature falls.

Method B1 converged well with a maximum of 8 processors. Method B2 converged imperfectly, but surprisingly it converged better than a random spatial decomposition technique [JD88].

Using extrapolated simulation measurements for a 900 circuit placement problem running on 90 processors, researchers estimated a speedup of about 45 for Method A, and 72 for Method B1 and B2 [DKN87].

Another experiment compared synchronous and asynchronous shared state-space algorithms for the equal partition cut-set problem [Dur89]. Given a graph with an even number of vertices, such algorithms partition the vertices into two equal sets, and minimize the number of edges which cross the partition boundary. The synchronous algorithm locked both vertices and edges, while the asynchronous algorithm locked only vertices.

On a 250 vertex graph, the synchronous algorithm ran more slowly than a sequential implementation, except at 16 processors where the speedup was close to 1. The asynchronous algorithm ran faster than the sequential algorithm, yielding 16-processor speedups from 5 on a graph with mean vertex degree 10, to 11 on a graph with mean vertex degree 80.

These experiments indicate that asynchronous execution may be beneficial in simulated annealing.

2.4 Hybrid Algorithms

Hybrid algorithms recognize that different schemes may be more appropriate at different temperatures. This section provides only a cursory review, since previous sections provided algorithmic details.

2.4.1 Modified Systolic and Simple Serializable Set

One hybrid combines a modified systolic algorithm and a simple serializable set algorithm [BAM88]. In the modified systolic algorithm, independent processors copy the current state, then complete a stream of moves at the same temperature.

The PICK operation from Figure 2.6 on page 37 chooses among the results, as per Equations 2.1 and 2.2. Equal temperatures for PICK simplify the computations.

At high temperatures, where most moves are accepted, Algorithm SSS provides little benefit—here only the systolic algorithm is used. As lower temperatures reduce the acceptance rate, the program combines Algorithm SSS with systolic. Finally, at extremely low acceptance rates, the program uses Algorithm SSS exclusively.

Researchers claim this hybrid is slightly faster than the systolic algorithm alone [ABH86].

2.4.2 Random Spatial Decomposition and Functional Decomposition

Another approach combines asynchronous spatial decomposition with functional decomposition [JR87]. This program randomly distributes the state variables across processors in an iPSC hypercube, to perform VLSI macro-cell placement.

With a 20 circuit problem, on a 16 processor iPSC, the algorithm obtained speedups of between 4 and 7.5. Considering the small problem size and the message-passing architecture, the speedup appears very good.

2.4.3 Heuristic Spanning and Spatial Decomposition

One implementation uses heuristic spanning, a non-simulated annealing technique, and asynchronous rectangular decomposition to perform VLSI placement

[RSV88].

The heuristic spanning algorithm chooses several random starting states, and iteratively improves each. For the high-cost regime, heuristic spanning shows better convergence behavior than simulated annealing.

In the low cost regime, rectangular decomposition refines the state space to a lower final cost than heuristic spanning could achieve. The rectangular decomposition method showed a speedup of 4.1 on 5 processors, and an extrapolated speedup of 7.1 on 10 processors. Using the hybrid technique, researchers estimate speedups of 10–13 on 10 processors, when compared to a standard simulated annealing algorithm.

2.4.4 Functional Decomposition and Simple Serializable Set

In another hybrid algorithm, functional decomposition operates at high temperatures, and simple serializable set operates at low temperatures [KR87]. The poor behavior of Algorithm SSS at high temperatures justifies a different algorithm.

In this early work, researchers sought to avoid convergence problems by using only serial-like algorithms—little was known of altered-generation or asynchronous algorithms. On a 100 circuit gate-array placement problem, they achieved a maximum speedup of 2.25 on a 4 processor VAX 11/784.

2.5 Summary

I categorized parallel annealing techniques into serial-like, altered generation, and asynchronous algorithmic classes. Comparisons of these techniques have been limited [GD89, DKN87, KR87, RSV88].

Based on this survey, it appears that asynchronous and altered generation algorithms have provided the best overall speedup, while one serial-like technique, simple serializable set, has been incorporated advantageously at low temperatures.

Although the asynchronous algorithms show promising speedups, the experiments described here make no appeals to theory. This problem is rectified in Chapter 4, where we show how asynchronous algorithms can be controlled to achieve the same outcome quality found in sequential algorithms.

CHAPTER 3

Experiment

In traditional simulated annealing, the algorithm must ensure that it always creates feasible solutions to the original problem, and that it evaluates the cost change accurately. Asynchronous simulated annealing relaxes the second requirement, tolerating some error in the computed cost.

This error tolerance provides an advantage in multiprocessing: when several processors proceed independently on different parts of the problem, they need not synchronously update state information in other processors. A processor can save several updates, then send them in a block to the other processors. The processor sends less control information and compresses multiple moves for a circuit into one. This reduces total communication traffic. With a few limitations, updates can occur out-of-order, decreasing synchronization operations.

A later chapter will determine how much error can be tolerated to achieve a particular goal. This chapter describes my early experiments on simulated annealing, showing empirically how errors affect the outcome quality of a circuit-placement problem. I show that errors can be limited by

1. limiting the number of processors assigned to the problem,

2. changing the way the problem is decomposed,
3. increasing the frequency of interprocessor communication.

3.1 Background

Quantifying the error tolerance of simulated annealing has been a difficult theoretical issue, so most published conclusions are based on experiment. Relying on empirical evidence, Jayaraman claims that simulated annealing can tolerate cost function errors of 10% with limited effect on convergence [JD88]. However, that foregone conclusion depends on the cost-function. An obvious modification of the cost-function in Figure 2.8, with 10% error, would generate a bad outcome under simulated annealing.

Grover states that the largest tolerable error is about half the current annealing temperature [Gro86], but this depends on the structure of the move-space. Rose et al suggest that error reduction is extremely important at high-temperatures, but less important at lower temperatures [RSV88]. Chapter 4 contradicts this notion.

This chapter expands on my experiments, mentioned earlier in §2.3.1.2. I wanted to see how different partitionings of a gate-array placement problem might affect measurable properties of asynchronous parallel simulated annealing. Furthermore, I hoped these experiments would suggest a relationship between the properties and the quality of the outcome. In particular, my experiments illustrate the behavior of errors at different temperatures, and show how parallelism affects those errors.

Furthermore, different partitioning schemes and temperatures change how the algorithm explores the state space. To try to capture this notion, I measured "circuit mobility", the average Manhattan distance a circuit moves when accepted. Circuit mobility in these experiments relates to the more rigorous notion of "mixing speed," discussed and used in Chapter 4. Similar notions have been explored in other works, from White's early work on "move scales" [Whi84] to Lam's attempts to combine an adaptive temperature schedule with an adaptive move-generation algorithm [Lam88, SLS89].

My experiments show that both parameters—errors and circuit mobility—affect the outcome quality in simulated annealing.

3.1.1 Spatial Decomposition

The "spatial decomposition" algorithm divides the gate-array grid into mutually disjoint partitions, either randomly [CRS86, JD88] or using a fixed pattern [Kli87], and assigns each partition to a different processor. A processor randomly selects two circuits within its partition, calculates the cost-function assuming the circuits were swapped, and decides whether to swap the circuits. No locking is required in these algorithms, because the processors operate on disjoint sets of circuits. A processor may complete several swaps, in a "stream," before sending the updated information to a common database.

Each processor acts independently on its partition, assuming that circuits in other partitions are stationary. Cost-function errors result because circuits in other partitions are *not* stationary—they are being modified by different processors. After a number of tries, the processes update a global map, the controlling task repartitions the grid, and the process repeats. Using a longer stream (allowing more swaps between updates) increases calculation errors. A successful algorithm must change partition boundaries between streams, otherwise limitations on state space exploration preclude a good outcome.

3.1.2 Experimental Work

This chapter explores four new rectangular spatial decomposition schemes. The approach divides a circuit grid into rectangular regions—the algorithms differ in where the boundaries can fall. One method, *Sharp* converges better than the others. In one example, the outcome placement quality of parallel *Sharp* was *higher* than standard sequential approaches. Here the partitioning method created a better `generate` function than was found in the sequential algorithm.

These experiments can be interpreted in terms of the increased circuit-mobility and decreased cost-function errors that occur with the rectangular partitioning approaches. Chapter 4 confirms this analytically.

3.2 The Algorithm

This spatial decomposition algorithm first assigns the circuits in a VLSI circuit to random locations on a grid.

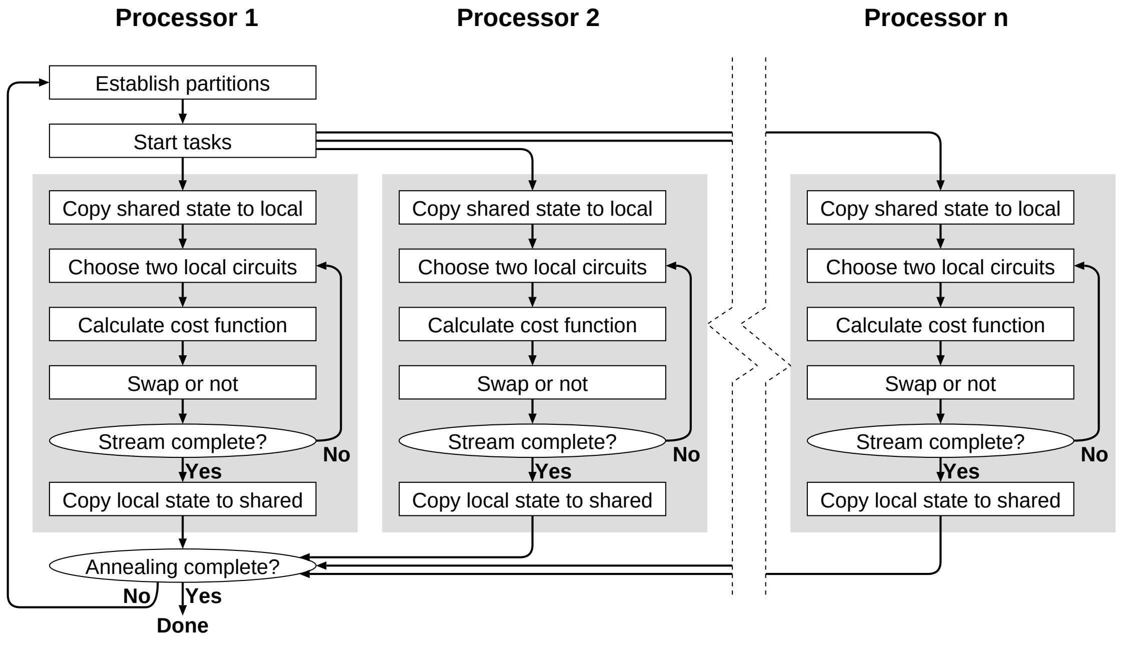


Figure 3.1: Parallel Execution Model

Figure 3.1 shows the parallel execution model. It begins with a supervisor process decomposing the grid into mutually disjoint partitions. All processes then proceed independently to complete a "stream" of trial moves on their respective regions. Each process copies the entire circuit map to its local memory—I optimized this step by copying only *changed* circuits to the local memory.

A process randomly chooses two circuits within its region. If the two circuits have Manhattan distance greater than 3, the process repeatedly retries, randomly selecting another two circuits until it finds two at Manhattan distance less than 4.

It then calculates the cost function based on its local copy of the circuit map. Circuits local to the partition accurately reflect their present position, circuit positions outside the partition become outdated when other processes move their circuits around.

Using the Metropolis algorithm, the process decides whether to swap the two selected circuits. If the circuits are swapped, the local circuit map is changed, but copies of the circuit map in other processors are *not* updated.

Finally, when the number of tries equals the "stream length," the process relinquishes control back to the supervisor process. Otherwise, it again randomly selects two circuits, as above, and the sequence repeats.

When all processes have completed their streams, the supervisor process creates an accurate shared copy of the circuit map, reflecting changes made in individual processors. It determines whether the annealing schedule is complete. If not, it changes the annealing constraints according to the schedule, re-establishes the partitions, and restarts the parallel processes.

3.2.1 Four Rectangular Approaches

To evaluate this class of parallel simulated annealing algorithms, I tried four different variations of rectangular partitioning, and compared them with two other approaches. It will be helpful to refer to Figure 3.2, which illustrates the four rectangular methods.

3.2.1.1 Proportional Rectangles

Proportional rectangles is the simplest of the four techniques. Here I divide the chip into equal rectangles, roughly proportional to the overall chip size. I then

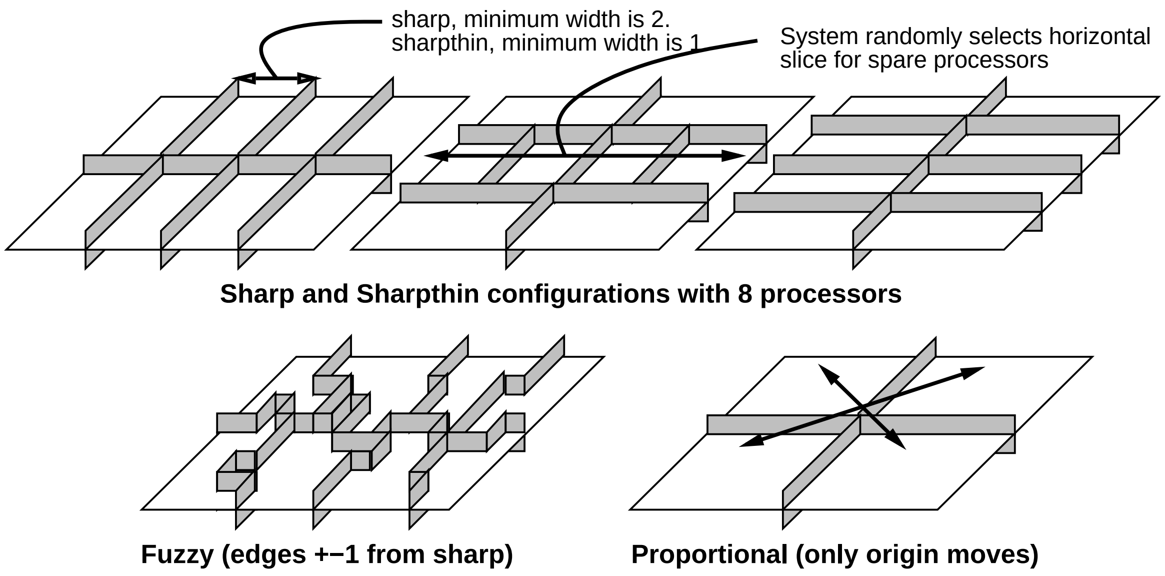

sharp, minimum width is 2.
sharpthin, minimum width is 1
System randomly selects horizontal slice for spare processors
Sharp and Sharpthin configurations with 8 processors
Fuzzy (edges +−1 from sharp)
Proportional (only origin moves)

Figure 3.2: Rectangular Decomposition Schemes

randomly select the origin for this grid. Note that the rectangle shapes in this technique never change.

3.2.1.2 Sharp Rectangles

The *Sharp* technique divides the chip into rectangles with a minimum width and height of 2. Let W be the chip width, H be the chip height, and p be the number of processors. W, H, and p must satisfy the inequality $\lceil 2p/H \rceil + 1 < \min(p, \lfloor W/2 \rfloor)$. If not, the chip size is too small for this partitioning: increase W or H, or decrease p.

Choose the number of columns. Randomly select an integer column width $\lceil 2p/H \rceil < w < \min(p, \lfloor W/2 \rfloor)$. The chip will hold $c = \lfloor W/w \rfloor$ columns.

Select the number of rows to $r = \lfloor p/c \rfloor$, using as many processors as possible with uniform divisions of the chip. The row height is $h = \lfloor H/r \rfloor$.

If $wc < W$ or $hc < H$, then there are too few processors for the columns or rows. To resolve this, widen a randomly chosen column c' to $w_{c'} = W - w(c-1)$ and heighten a randomly chosen row, r' to $h_{r'} = H - h(r-1)$.

If $p < rc$, then $rc - p$ processors are idle. To use the remaining processors, increase the number of rows in column c' to $r_{c'} = p - c(r-1)$. If $hr_{c'} \neq H$, then heighten a randomly selected row, r'', in column c' to $h_{r''} = H - h(r_{c'} - 1)$, in the same manner as for c'.

3.2.1.3 Sharpthin Rectangles

Sharpthin rectangles is the same as *Sharp*, except that it randomly selects $w \in \{\lceil p/H \rceil, \ldots, \min(p, W)\}$. *Sharp* ensures that $w \geq 2$ and $h \geq 2$, while *Sharpthin* allows $w \geq 1$ and $h \geq 1$.

It was thought that rectangles with width or height 1 would significantly increase the calculation error and reduce the quality of the resulting chip. The data reported here appear to confirm these initial thoughts. The same effect probably occurred in [KCP94].

3.2.1.4 Fuzzy Rectangles

Fuzzy rectangles applies the *Sharp* method discussed above, except that circuits lying on rectangle borders are assigned randomly to any rectangle adjacent to the circuit. This creates rectangles with "fuzzy" borders. I hoped to increase the circuit mobility by doing this, but recognized that it might introduce greater errors.

3.2.1.5 Random3

Random3 is a variation of Jayaraman's technique [JD88]. Before a swap stream, each circuit is randomly assigned to a processor. The circuits are divided roughly equally among the processors and no circuit is allocated to more than one processor each time. When the swap stream completes, the circuits are again redistributed to the processors.

The difference between *Random3* and Jayaraman's technique is that *Random3* disallow exchanges of pairs of circuits whose Manhattan distance exceeds 3—serial algorithms typically use this restriction to accelerate the annealing process [KCV83].

I also tried Jayaraman's technique, called *Random* in the figures, hoping to provide some data for comparison, but when the number of processors exceeded 4 annealing runs using unlimited Manhattan distance would not converge in a reasonable time.

3.2.2 Speed vs. Accuracy and Mobility

When a process moves circuits within its region, the global states viewed by the other processes become inconsistent. This introduces a computation error which, uncontrolled, would destroy the quality of the outcome.

To minimize these errors, the stream length must be kept reasonably small.

However, as the stream length decreases, the interprocess communication increases, reducing the benefits of multiprocessing. Any spatial decomposition algorithm must balance these conflicting goals: higher accuracy improves the result, but requires more time.

Partition shape can influence errors. Figure 3.3 shows cost-function errors for different spatial decomposition methods, while annealing the *P9* example with 4 processors and a stream length of 64 tries.

Random assignment can introduce tremendous errors. If circuits are assigned to partitions at random, a circuit's neighbors are not likely to be within the same partition. In a chip where circuits tend to have strong local connectivity (most chips are like this), calculation errors in random assignment are likely to be higher. Swaps within these random partitions would cause dramatic changes in net length, and those nets would probably be connected to circuits in other partitions. The large cost-function errors for *Random3* and *Random* shown in Figure 3.3 bear this out.

Circuit mobility also competes with execution time. Suppose, for example, that partition boundaries never change. Then all swaps will occur within a region, and the circuits in the region will never leave. To allow circuits to travel any location on the grid, the algorithm must periodically redraw the boundaries. This repartitioning increases execution time, because individual processes must copy global state information for their region's circuits before each stream. As the stream length decreases, the program redraws partition boundaries more often—

Figure 3.3: Cost Function Errors vs. Decomposition Methods

circuit mobility increases, but the execution speed decreases. Again, there is a fundamental conflict: higher circuit mobility improves the result, but requires more time.

Partition shape can also affect circuit mobility. Figure 3.4 shows how average Manhattan distance traveled per circuit varies with different spatial decomposition methods—again I show the *P9* example with 4 processors and 64 tries per stream.

Figure 3.5 presents an interesting contrast: Rather than showing the average distance moved per trial (essentially a unit of time), it shows the average distance moved per accepted move. Note that unrestricted random assignment resulted in the worst final placement, yet it has a high mobility per accepted move, throughout

Figure 3.4: Circuit Mobility/Trial vs. Decomposition Method

the temperature range. This suggests that the important issue is mobility *per trial*. Theory also suggests this, as mixing speed is related to mobility per trial [OG89, Sor91].

Suppose that partitions in a spatial decomposition scheme always take on a wide, flat shape—as wide as the entire virtual grid. The non-random spatial decomposition scheme described in [Kli87] follows this approach. Circuits exhibit high mobility in the horizontal direction, but low mobility in the vertical direction. A circuit can travel from grid left to grid right in one stream, but it may take several streams to travel from top to bottom. In terms of mobility, this may

Figure 3.5: Circuit Mobility/Accepted Move vs. Decomposition Method

be suitable for row-based placement problems where horizontal mobility is more important, but is probably inadequate for gate-array or macro-cell placement.

3.3 Empirical Data

To gather statistics about placement quality, I ran the random rectangles algorithms in a simulated IBM RP3 environment, using EPEX C [WN86] on an IBM 3090, an IBM Ace/RT multiprocessor, and several HP 9000/350s. The nature of the algorithm is such that simulation does not affect the placement quality or the errors observed.

I used the classic VLSI placement cost-function [KCV83], namely $C = L + X^2$, where C is the cost (energy) of the system, L is the total wire length of the nets, computed by the bounding box method, and X is the wire congestion. Since the algorithm was simulated, run times for the RP3 cannot be computed.

I obtained speedups, however, on the IBM Ace/RT multiprocessor. Due to frequent bus collisions on this 8 processor, single bus system, I view the speedup values obtained as lower bounds.

To evaluate a simulated annealing algorithm, one must run several trials using different random seeds. The mean final cost over many trials provides a good measure of the quality of a particular annealing technique. For each data point, I ran 50 trials.

At the boundary condition of a single processor, all spatial decomposition techniques are the same. In the tables which follow, I simply duplicated the data from one set of single processor trials in all five categories. Had I made separate runs, the single processor data points would differ slightly due to the random nature of the runs.

3.3.1 The P9 Chip

I tried two chip systems. The first, *P9*, is a uniform 9×9 grid with immediate neighbor circuits connected by two-circuit nets. There are 81 circuits, and 144 nets. It has a known ground state, under the cost-function, of $E = 144$, $L = 144$ and $C = 0$. The ground state appears in Figure 3.6.

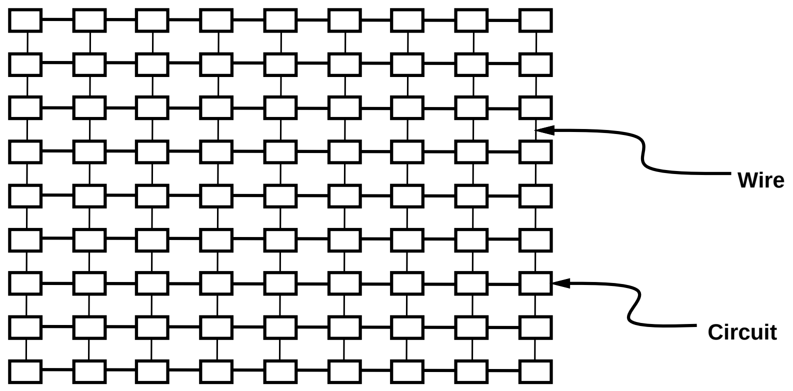


Figure 3.6: P9 Ground State

I evaluated each of the four rectangular techniques with 1, 2, 4, and 8 processors. A fifth set of runs using *Random3* provides a baseline from which to measure improvement. Table 3.1 shows the type of partitioning, the number of processors (*P*), the number of tries per stream (*T/S*), the percentage of runs which reached the ground state cost of 144 (*% G*), the average final cost (*Ave E*), and the standard deviation of the final cost (*Std E*).

	Sharp			Sharpthin			Proportional		
P	*% G*	*Ave E*	*Std E*	*% G*	*Ave E*	*Std E*	*% G*	*Ave E*	*Std E*
1	38%	233.32	97.14	38%	233.32	97.14	38%	233.32	97.14
2	27%	255.32	97.62	38%	236.82	102.61	72%	180.68	60.19
4	26%	252.64	90.85	40%	245.68	106.93	24%	258.66	92.50
8	100%	144.0	0.00	20%	287.92	97.35	18%	316.60	134.22

	Fuzzy			Random3		
1	38%	233.32	97.14	38%	233.32	97.14
2	38%	231.36	86.69	36%	244.72	97.70
4	15%	298.01	98.42	38%	254.92	106.96
8	0%	363.44	74.96	15%	262.28	91.05

Table 3.1: P9, 64 Tries/Stream: Convergence Statistics

Note that as the number of processors increases, most runs show an increasing average cost-function value. The number of runs which result in the ground state of $E = 144$ typically decreases. *Random3* shows the worst degradation as the number of processors increases. Since *Random3* creates the highest calculation errors, that result was expected.

Oddly, *Sharp* on 8 processors shows the best result—all runs reached the ground state. This is assumed to be due to the effects of partitioning on the move space. With the ZA chip, described in the next section, *Sharp* performed better than the other parallel techniques, but did not beat sequential annealing in final cost.

The *Sharp* algorithm was also run on an IBM Ace/RT multiprocessor for a speed trial. The IBM Ace/RT contains 8 IBM RT processors connected by a single high-speed bus. Speedup data are shown in Table 3.2. Speedup is strongly limited by the bus, since a huge amount of information (all changed states) must be transferred on the shared bus at the end of each stream. The low speedups obtained should be interpreted with the hardware limitations in mind.

Annealing on two processors was slower than one. There is no communication overhead in one processor, since state changes do not have to be copied at all. Increasing the processors to two caused a substantial increase in communication overhead, enough to eliminate the extra processing available. Increasing the processors from two to four did not increase communication as much, and the algorithm could then exploit some of the extra processing available.

P	*Trials/Stream*	*Seconds*	*Speedup*
8	16	2008	1.5
4	32	2577	1.16
2	64	3361	0.89
1	128	3007	1

Table 3.2: P9, Sharp: IBM Ace/RT Speeds

3.3.2 The ZA Chip

The other problem I tried, called *ZA*, is a real printed circuit placement problem. All circuits in this problem are uniformly square. There are 359 circuits in *ZA*, some of which are unconnected. There are 50 nets, each with an average of 4.04 attached circuits.

Overall, there are three blocks of interconnected circuits. Two blocks include a few simple 2-circuit nets—circuit swaps in those blocks have minor effects on the cost-function. The third block includes many multiple-circuit nets, typically 9 to 11 circuits per net. Each circuit has numerous connections to other circuits. One swap in this group typically causes dramatic changes in the cost function.

	Sharp		Sharpthin		Proportional		Fuzzy		Random3	
P	*Ave E*	*Std E*	*Ave E*	*Std E*	*Ave E*	*Std E*	*Ave E*	*Std E*	*Ave E*	*Std E*
1	139.08	1.83	139.08	1.83	139.08	1.83	139.08	1.83	139.08	1.83
2	138.74	1.99	138.69	1.93	139.52	2.31	139.30	2.04	139.46	2.39
4	139.58	2.22	139.26	2.49	139.46	2.31	139.28	2.06	139.34	2.15
8	140.40	2.51	140.16	2.59	140.00	2.44	139.70	2.24	142.54	2.50
16	141.02	2.71	141.56	2.26	140.78	2.67	140.34	2.63	150.12	3.22
32	142.02	2.53	142.50	3.05	142.54	2.99	144.48	3.82	166.30	5.93

Table 3.3: ZA, 64 Tries/Stream: Convergence

There is no known ground state to *ZA*. I obtained data for the four rectangular spatial decomposition methods, and for the *Random3* method. I tried each of these partitioning methods with 1, 2, 4, 8, 16, and 32 processors. As with *P9*, data for the single processor case is duplicated in all categories. Table 3.3 shows the results.

As with the *P9* example, Table 3.3 shows the best results in the *Sharp* partitioning method. *Fuzzy* looks promising until the processor count exceeds 32. *Random3*, as in the *P9* case, produces terrible results when the number of processors becomes high.

I did not compare speedups for this circuit. The tests were performed on a simulated RP3 environment that did not give timing information.

3.4 Summary

This chapter presented four rectangular spatial decomposition techniques for parallel simulated annealing. The rectangular techniques use partition shape to help increase circuit mobility and decrease cost-function errors. This allows increased stream length, providing greater parallelism and decreasing execution time on multiprocessors.

One rectangular technique, *Sharp*, appears to perform better than the others. I attribute this to the relatively low errors associated with it.

These experiments show that decreased cost-function errors and increased per-trial mobility are important goals in spatial decomposition methods for simulated annealing. Increased per-move mobility is apparently unimportant. Annealing researchers face a trade-off: increased parallelism can increase cost-function errors or decrease per-trial circuit mobility, resulting in a less desirable annealing result.

Partition shape can affect both circuit mobility and cost-function errors, and should be considered when constructing parallel annealing programs.

CHAPTER 4

Analysis

4.1 Background

Previous chapters discussed different parallel simulated annealing techniques, and described my experiments in the area of parallel simulated annealing. My experiments explored two values: cost-function errors and mobility.

That work drove me to analyze the effects of errors on simulated annealing, a much discussed topic which had a limited theoretical base. This chapter attempts to bring a stronger foundation to the field.

In this chapter I prove these claims: Simulated annealing is tolerant of errors, but errors either reduce outcome quality or require the algorithm to increase the iterations performed. Convergence is guaranteed only if errors are constrainted to a constant factor of the temperature. This is true both for the logarithmic temperature schedule used in many annealing theory texts, when applied to general annealing spaces, and for the common geometric temperature schedule applied to self-affine functions. Finally, I construct a temperature schedule for a self-affine cost-function to show how the results are applied.

4.1.1 Inaccuracies in Cost Functions

Inaccuracy can speed up simulated annealing. For example, circuit placement algorithms often use the half perimeter method, described in §1.1.4.2. If the wires in the net parallel the x or y axis, if the net holds no more than 3 cells, and if the wires encounter no obstructions, the half perimeter method is accurate. Otherwise, it may be inaccurate. To calculate the wire-length more accurately would require a slower cost-function.

Figure 4.1 shows an example. In the first row, *a* shows the half perimeter estimate for a three-cell net. *b* shows the rectilinear wiring use with no obstruction. *c* adds an obstruction. *d* allows the wires to be placed at angles. In the second row, *e* shows the estimate for a four-cell net. *f* shows the rectilinear wiring with no obstruction. Only in case *b* is the estimate correct.

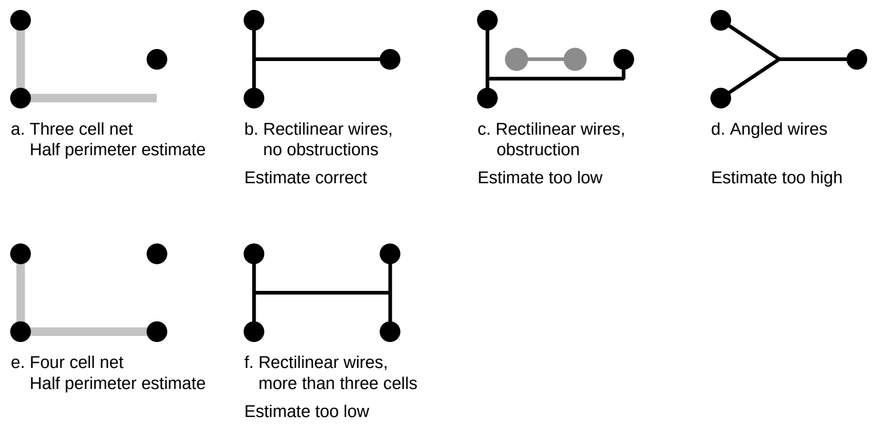


Figure 4.1: Wire Length Estimates

The problem statement itself is an approximation of an ill-defined real problem: place transistors on silicon so that a chip is as fast as possible, while keeping the chip area small. Longer wires slow down the chip, making total net length a plausible cost-function, but long-wires are not the only contributors to chip slowness, and their contribution is not linear. A more accurate cost-function would involve simulating the circuit and measuring the total time-delay, but this would slow the annealing algorithm by several orders of magnitude.

4.1.2 Inaccuracies in Parallel Algorithms

Parallel processing can speed up simulated annealing. The parallel method I focus on partitions ownership of the state variables among several processors. A processor then generates trial states by manipulating only its own state variables.

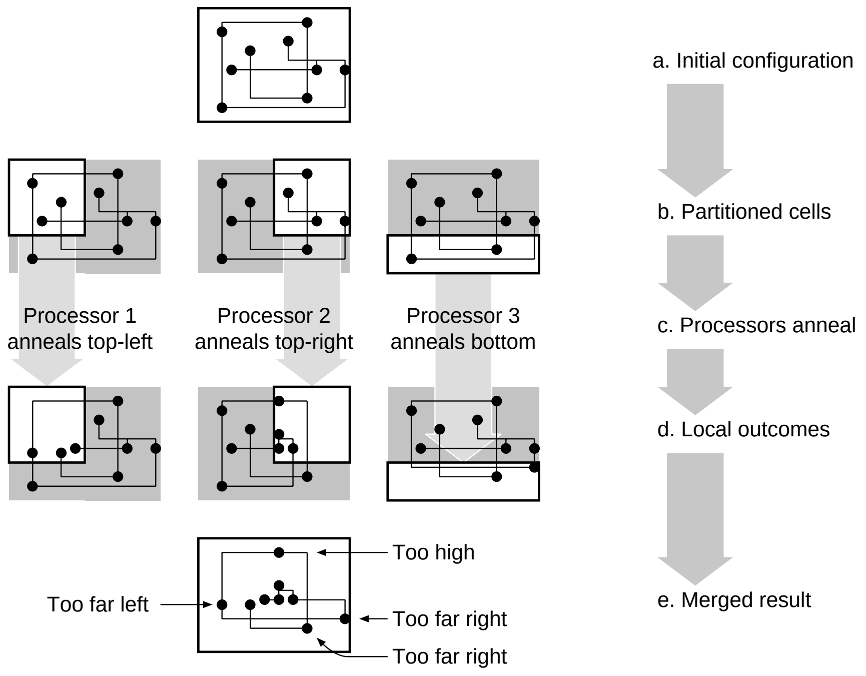


Figure 4.2: Parallel Annealing with Stale State-Variables

Each processor maintains local copies of other processors' state variables. If the value of a local copy of a state variable differs from the value in the owning processor, call the local data "stale." Periodically, the processors reassign cell ownership and refresh local copies.

Figure 4.2 shows a 3-processor gate-array circuit placement example. Row *a* shows the initial configuration. Each dot in the figure is a cell. The lines connecting the dots are nets.

Row *b* shows how the cells have been partitioned among the processors, in this case with some spatial locality.

In row *c*, each processor performs a fixed number of annealing loop iterations and then stops. A processor manipulates only state-variables for cells it owns (in the white area), but can refer to local copies of other state-variables (in the gray area). A processor may move a cell only within the white area.

Row *d* shows the result of each processor's annealing, with the cell locations it used to perform cost-function calculation. Notice that all local positions of unowned cells are stale, and so many of the cost-function results were inaccurate.

Row *e* shows the merged outcome. If the iterations occurred at an infinitesimally low temperature (a greedy algorithm), stale state-variables caused mistakes in moving cells. The diagram points out four in the merged outcome.

The parallel algorithm allows the cost-function to use stale state-variables, and those causes the cost-function to return inaccurate results [Gre91b]. Experiments have characterized these inaccuracies [GD89, JD88, DKN87].

To avoid using stale state-variables, the processors must communicate more frequently, perhaps even by locking state-variables. These operations increase execution time. Thus, as in the half perimeter wire-length estimate, this trades off accuracy for speed.

4.1.3 Errors

It is useful to measure the "error" of a cost-function computation: the difference between a state's true-cost and the value returned by the cost-function (the "observed cost"). But practitioners do not directly measure all errors in commercial applications: that would require computation of the true-cost *and* the inaccurate cost-function. If you can afford to compute the true-cost every time, there is no benefit in performing the inaccurate cost computation. Instead, you can sample the errors or execute the algorithm on a smaller representative problem to get error estimates.

I assumed that errors fall into two categories: range-errors and Gaussian-errors. A cost-function has "range-errors" if the difference between the true-cost $C(s)$ and the cost-function $\dot{C}(s)$ is confined to a fixed range. A cost-function has "Gaussian-errors" if the difference between the true-cost $C(s)$ and the cost-function $\ddot{C}(s)$ is a Gaussian random variable with mean $C(s)$.

I studied range-errors because they are easy to measure, and Gaussian-errors because they are the sums of independent random variables. The annealing algorithm shown in Figure 4.2 generates errors composed of a sum of dependent random variables. As the chip becomes large relative to the number of processors, I suspect the dependencies will decrease and the errors will become closer to Gaussian. Others have hypothesized similarly [GM89]. I draw conclusions about both range-errors and Gaussian-errors, but range-errors receive the most complete discussion.

Throughout this text, symbols with one dot above signify range-errors, such as $\dot{C}$. Symbols with two dots above signify Gaussian errors, such as $\ddot{\boldsymbol{\pi}}_s$. Symbols without dots refer to functions and variables without errors.

4.1.4 The Problem

Experiments show that larger errors increase the outcome cost, but the literature contains contradictions (see discussion in §3.1). There are several unanswered questions. I address this by exploring three general areas:

¶ *Fixed temperature behavior*: If the cost-function errors have known properties, what is the equilibrium distribution at a fixed temperature? What is the expected cost? At what rate does the system equilibrate?

¶ *Decreasing temperature schedules*: If you limit the errors in the cost-function during execution, by changing the frequency of interprocessor communication, by altering the precision of computed values, or by introducing a cost-function with a different accuracy, can you produce as good an outcome as with the true-cost? To get the same expected outcome as with an accurate cost function, how should you alter the temperature? How will the total execution time change?

¶ *Error characteristics*: What types of inaccuracies might you expect? How do you measure them?

4.1.5 Prior Work

Appealing to physical science analogies, Grover presented an early analysis showing the effects of range-errors on the partition function [Gro89]. Durand and White analyzed equilibrium properties for range-errors on a restricted algorithmic class [DW90].

Gelfand and Mitter showed that slowly decreasing state-independent Gaussian errors, under some conditions, do not affect asymptotic convergence under long schedules, for discrete [GM89] and continuous [GM91] state spaces. I do not assume state-independence.

Romeo and Sangiovanni-Vincentelli give conditions on range-errors such that transition probabilities of an inaccurate annealing process converge to those of the accurate process as $T \to 0$ [RS91]. That result can be usefully applied to long schedules where regardless of the starting temperature, the system is guaranteed to approach the equilibrium distribution. Application to shorter schedules seems limited. I extend this result in §4.5.

Experiments have shown that when errors are proportionally constrained to temperature, outcomes improve. Invoking these observations, researchers have modified their algorithms to obtain better outcomes [KCP94, BJS90, CRS87]. The analytic results I obtain confirm that temperature-dependence.

4.1.6 Contents

§4.2 introduces simulated annealing as a Markov process, and presents its fundamental properties.

§4.3 derives equilibrium properties for annealing. §4.3.1 shows the effects of range-errors on expected cost, and §4.3.2 shows the effects of Gaussian errors on expected cost.

§4.4 shows how inaccuracies affect conductance. Conductance is a value that approximates the rate a system approaches equilibrium (the mixing rate). §4.4.1 establishes the conductance for annealing with range-errors. §4.4.2 establishes the conductance for annealing under Gaussian noise.

§4.5 introduces a framework for simulated annealing [Tsi89]. Using this framework, I prove that limiting cost errors proportionally to the temperature guarantees the minimum cost outcome, using a $T(t) = d/\log t$ temperature schedule. The result is only mildly useful, since the execution time is exponential.

§4.6 describes fractal state spaces. Fractal spaces and practical problems are statistically similar [Sor92, MS89, KT85]. Annealing on fractal spaces has a convenient property: use of the geometric $T(t) = c^t T_0$ schedule, used in many commercial annealing programs, produces an outcome of preselected quality. I prove that with a $T(t) = c^t T_0$ temperature schedule on a fractal state space, if errors are strictly limited to values proportional to the temperature, the system obtains the same outcome quality. §4.6.1 analyzes "confined annealing," and §4.6.2 extends the result to "unconfined annealing." In §4.6.3, I show that inaccuracies require you to increase the annealing steps to achieve the same quality.

§4.7 discusses the practical issues involved in measuring errors. In this section, I also show either of two types of observed range-errors, "instantaneous" or "accumulated" errors can be used in my results.

§4.8 summarizes the results and offers practical insights for implementing inaccurate simulated annealing algorithms.

4.2 Properties of Simulated Annealing

First, I formalize simulated annealing. Please refer to Figure 4.3.

```
1.    $T \leftarrow T_0$;
2.    $s \leftarrow$ starting − state;
3.    $E \leftarrow C(s)$;
4.    while not stopping-criteria()
5.        $s' \leftarrow$ generate($s$) with probability $G_{ss'}$;
6.        $E' \leftarrow C(s')$;
7.        $\Delta \leftarrow E' - E$;
8.        if $(\Delta \leq 0) \vee$ (random() $< e^{-\Delta/T}$)
9.            $s \leftarrow s'$;
10.           $E \leftarrow E'$;
11.       $T \leftarrow$reduce-temperature($T$);
12.       end while;
```

Figure 4.3: Simulated Annealing

Let $N \in \mathbb{Z}^+$ be the number of states, and let $S = \{1, \ldots, N\}$ label all states. Define the *generate probability matrix* G so that $G_{ss'}$ is the probability that `generate` will return state s' when passed argument s.

Let T_t be the temperature after t iterations. Define the *acceptance probability matrix at time t*, $A^{(t)}$, so that

$$A^{(t)}_{ss'} = \begin{cases} 1 & \text{if } C(s') \leq C(s) \\ e^{(C(s)-C(s'))/T_t} & \text{otherwise.} \end{cases} \tag{4.1}$$

Define an inhomogeneous Markov chain P corresponding to the annealing loop of steps 3–9 in Figure 4.3 as follows.

$$P^{(t)}_{ss'} = \begin{cases} G_{ss'}A^{(t)}_{ss'} & \text{if } s \neq s' \\ 1 - \sum_{x \neq s'} P^{(t)}_{ss'} & \text{otherwise.} \end{cases} \tag{4.2}$$

Let s_0 be the starting state. $P^{(t)}P^{(t-1)} \cdots P^{(0)}s_0$ gives the state probability vector after t iterations.

If you fix the temperature, then $A^{(t)} = A^{(t+1)} = A$ and $P^{(t)} = P^{(t+1)} = P$, and the annealing algorithm simulates a homogeneous Markov chain.

Even analysis of realistic *accurate* annealing is difficult: a time-dependent acceptance matrix introduces problems, programs often use time-dependent generate functions [Whi84, Lam88], and the temperature schedule may not be monotonic [SK91].

Fortunately, annealing programs attempt to bring state probabilities near equilibrium at each temperature [KCV83]. Thus, you can gain information about annealing behavior by looking at the homogeneous Markov chain.

Call a Markov chain that represents state transitions in an annealing algorithm an "annealing chain." If the temperature is fixed, the annealing chain is homogeneous. A homogeneous annealing chain should be ergodic, otherwise the minimum cost state could be unreachable or periodic. With this property, each state s has an *equilibrium probability* $\boldsymbol{\pi}_s$ independent of the initial state. You can model many annealing applications as ergodic processes. You can guarantee ergodicity if the following properties hold.

$$\begin{aligned} &\text{probability} && \forall s \in S, \sum_{s' \in S} G_{ss'} = 1 && (4.3) \\ &\text{coverage} && \forall s, s' \in S, \exists k \geq 1, [(G^k)_{ss'} \neq 0] && (4.4) \\ &\text{aperiodicity} && \exists s \in S, [P_{ss} \neq 0] && (4.5) \\ &\text{finiteness} && |S| \in \mathbb{Z}^+ && (4.6) \end{aligned}$$

(4.3) simply states that G_s is a probability vector. (4.4) guarantees that the annealing chain is irreducible. (4.5) and (4.6) ensure that the chain is aperiodic and finite. Irreducible, aperiodic, finite Markov chains are ergodic.

Add one more, the symmetry property,

$$\text{symmetry} \qquad \forall s, s' \in S, [G_{ss'} = G_{s's}]. \tag{4.7}$$

Including (4.7) produces three useful results [OG89]: first, the equilibrium distribution is otherwise independent of G; second, the annealing chain is reversible, namely,

$$\forall i, j \in S, \ \boldsymbol{\pi}_i P_{ij} = \boldsymbol{\pi}_j P_{ji}; \tag{4.8}$$

third, the equilibrium distribution is the Boltzmann distribution, namely,

$$\boldsymbol{\pi}_s(T) = \frac{e^{-C(s)/T}}{\sum_{s' \in S} e^{-C(s')/T}}. \tag{4.9}$$

Properties (4.3)–(4.7) are based on those of Otten and van Ginneken [OG89]: however, they ensure aperiodicity by $\forall s \in S, G_{ss} \neq 0$, a requirement not typically satisfied by annealing programs. In contrast, this aperiodicity property, (4.5), is trivially satisfied if $\exists s, s' \in S, C(s) \neq C(s')$.

4.3 Equilibrium Properties

In this section, I compare the expected cost at equilibrium of an inaccurate algorithm and that of an accurate algorithm, based on both range-errors and Gaussian-errors.

When errors appear in the cost-function, I refer to the true-cost, $C(s)$, of state s, and the observed-cost $\dot{C}(s) = C(s) + \epsilon(s)$, of state s. ϵ is a random function dependent on s, thus $\dot{C}(s)$ is a random function.

4.3.1 Range-Errors

If the true-cost of s is $C(s)$, a cost-function with range-errors, $\dot{C}(s)$, satisfies (4.10).

$$\forall s \in S, C(s) + \underline{\epsilon} \ \leq \ \dot{C}(s) \ \leq \ C(s) + \overline{\epsilon} \tag{4.10}$$

Let $C\colon S \to \mathbb{R}$ give the true cost of each state. Let $\dot{C}\colon S \to \mathbb{R}$ be a range-error cost-function. You can use $\underline{\epsilon}$ and $\overline{\epsilon}$ from (4.10) to compare the equilibrium distributions of annealing with and without range-errors. The results I will derive will depend only on $\overline{\epsilon} - \underline{\epsilon}$, so let $\gamma = \overline{\epsilon} - \underline{\epsilon}$.

Theorem 1 *Let $\dot{\boldsymbol{\pi}}_s(T)$ and $\boldsymbol{\pi}_s(T)$ be the equilibrium probabilities of state s at temperature T, with and without range-errors. Then,*

$$e^{-\gamma/T}\boldsymbol{\pi}_s(T) \leq \dot{\boldsymbol{\pi}}_s(T) \leq e^{\gamma/T}\boldsymbol{\pi}_s(T) \tag{4.11}$$

Proof. Pick any state $s \in S$. By (4.9), if you minimize the cost of s and maximize the cost of all other states, as shown in Figure 4.4c, you maximize the equilibrium probability of state s. Thus, let $\dot{C}(s) = C(s) + \underline{\epsilon}$, and $\forall s' \in (S \setminus \{s\}), [\dot{C}(s') = C(s') + \overline{\epsilon}]$.

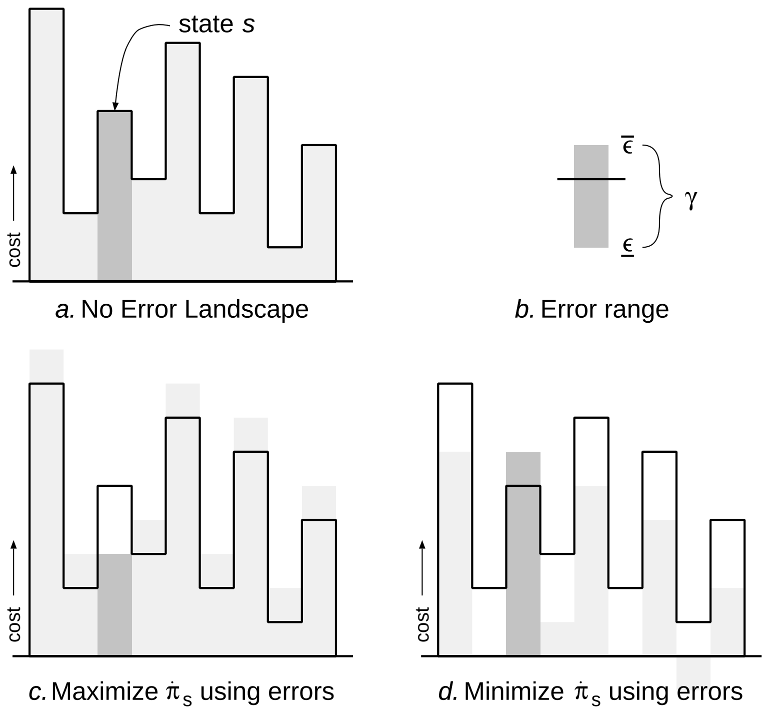


Figure 4.4: Range-Errors and Equilibrium

In fixing these worst-case costs, the inaccurate system satisfies the Boltzmann distribution,

$$\dot{\boldsymbol{\pi}}_s(T) \leq \frac{e^{-(C(s)+\underline{\epsilon})/T}}{e^{-(C(s)+\underline{\epsilon})/T} + \sum_{s' \neq s} e^{-(C(s)+\overline{\epsilon})/T}} \tag{4.12}$$

$$= \frac{e^{-\underline{\epsilon}/T} e^{-C(s)/T}}{e^{-\underline{\epsilon}/T} e^{-C(s)/T} + e^{-\overline{\epsilon}/T} \sum_{s' \neq s} e^{-C(s)/T}} \tag{4.13}$$

Since $e^{-\underline{\epsilon}/T} \geq e^{-\overline{\epsilon}/T}$, then

$$\dot{\boldsymbol{\pi}}_s(T) \leq \frac{e^{-\underline{\epsilon}/T} e^{-C(s)/T}}{e^{-\overline{\epsilon}/T} \sum_{s \in S} e^{-C(s)/T}}. \tag{4.14}$$

Finally, substituting the accurate equilibrium probability $\boldsymbol{\pi}_s(T)$ for its equivalent gives

$$\dot{\boldsymbol{\pi}}_s(T) \leq e^{(\overline{\epsilon} - \underline{\epsilon})/T} \boldsymbol{\pi}_s(T). \tag{4.15}$$

The converse argument supplies the lower bounds.

■

Theorem 2 *Let $\bar{C}[\boldsymbol{\pi}(T)]$ be the expected true-cost C, with equilibrium distribution $\boldsymbol{\pi}$, at temperature T. So,*

$$\bar{C}[\boldsymbol{\pi}(T)] = \sum_{s \in S} C(s) \boldsymbol{\pi}_s(T) \tag{4.16}$$

Then,

$$e^{-\gamma/T} \bar{C}[\boldsymbol{\pi}(T)] \leq \bar{C}[\dot{\boldsymbol{\pi}}(T)] \leq e^{\gamma/T} \bar{C}[\boldsymbol{\pi}(T)] \tag{4.17}$$

Proof. By Theorem 1,

$$e^{-\gamma/T} \sum_{s \in S} C(s) \boldsymbol{\pi}_s(T) \leq \sum_{s \in S} C(s) \dot{\boldsymbol{\pi}}_s(T) \leq e^{\gamma/T} \sum_{s \in S} C(s) \boldsymbol{\pi}_s(T) \tag{4.18}$$

which is equivalent to (4.16).

■

Thus, the difference between a state's true equilibrium probability and its probability under an error-range increases exponentially as the size of the error-range increases and the temperature decreases. The average cost varies similarly. I use these results throughout the rest of the chapter.

4.3.2 Gaussian Errors

Simulated annealing typically operates on discrete cost-functions where errors appear as discrete values. However, as you add state variables and as the maximum number of uncorrected independent moves increases (through increased parallelism or longer stream lengths, for example), the errors can approach a Gaussian distribution.

In many instances, particularly in parallel applications, the probability distribution of the inaccurate cost-function $\dot{C}(s)$ is reflected about the true-cost $C(s)$ [DW90].

Thus, it is reasonable to investigate the effect of Gaussian-error cost-functions, where each value is a Gaussian random variable with mean $C(s)$. I will show that when the variances of the state costs do not differ greatly, annealing converges to a good solution.

Lemma 3 *Let the cost of each state s be $\ddot{C}(s) = C(s) + X_s$, where X_s is an independent random variable. Execute the simulated annealing algorithm in Figure 4.3, with lines 3 and 6 sampling random variables $\ddot{C}(s)$ and $\ddot{C}(s')$ respectively, and with T fixed. If (4.3)–(4.7) are satisfied and $T > 0$, the resulting homogeneous Markov chain $\ddot{P}$ is ergodic and reversible, and the equilibrium probabilities are* *given by*

$$\ddot{\boldsymbol{\pi}}_i = \frac{e^{-C(i)/T}\mathrm{E}[e^{-X_i/T}]}{\sum_{j\in S} e^{-C(j)/T}\mathrm{E}[e^{-X_j/T}]}. \tag{4.19}$$

Proof. For any two states $s, s' \in S$, the acceptance probability $\ddot{A}_{s,s'}$ under random cost-function $\ddot{C}$ satisfies $0 < \ddot{A}_{s,s'} < 1$.

$$(P^k)_{ss'} = \sum_{s_1} \cdots \sum_{s_{k-1}\in S} G_{ss_1}\ddot{A}_{s,s_1} \cdots G_{s_{k-1}s'}\ddot{A}_{s_{k-1},s'}. \tag{4.20}$$

Since (4.4) is satisfied and all terms of A are non-zero, there is some k for which $(\ddot{P}^k)_{ss'}$ is non-zero. Thus, $\ddot{P}$ is irreducible.

(4.3)–(4.6) ensure that $\ddot{P}$ is irreducible, aperiodic, and finite, so $\ddot{P}$ is ergodic. G is symmetric (4.7), so $\ddot{P}$ is reversible and it satisfies the *detailed balance condition* (4.8). Thus, $\ddot{\boldsymbol{\pi}}_i\ddot{P}_{ij} = \ddot{\boldsymbol{\pi}}_j\ddot{P}_{ji}$. If $\ddot{P}_{ij}$ is non-zero, then $\ddot{\boldsymbol{\pi}}_i = \ddot{\boldsymbol{\pi}}_j\ddot{P}_{ji}/\ddot{P}_{ij} = \ddot{\boldsymbol{\pi}}_j G_{ji}\ddot{A}_{ji}/(G_{ij}\ddot{A}_{ij})$. By (4.7), $\ddot{\boldsymbol{\pi}}_i = \ddot{\boldsymbol{\pi}}_j\ddot{A}_{ji}/\ddot{A}_{ij}$.

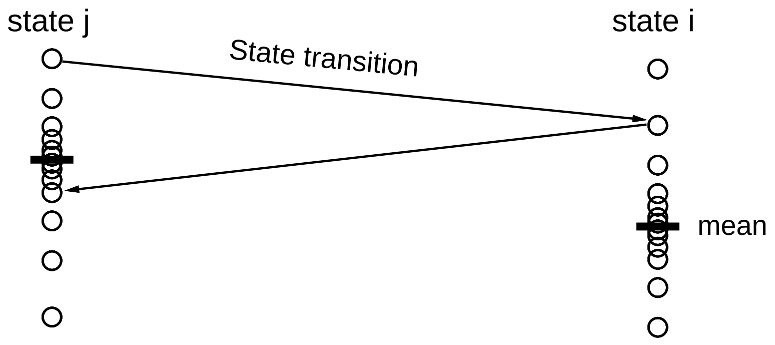


Figure 4.5: State i Sampled and Fixed.

Upon entering state i, sample and fix its cost until the state changes again, as in Figure 4.5. This is consistent with the algorithm in Figure 4.3.

Let $a(C_i, C_j)$ be the probability of accepting a move from a state with cost C_i to a state with cost C_j. By (4.1) this is

$$a(C_i, C_j) = \begin{cases} 1 & \text{if } C_j \leq C_i \\ e^{(C_i - C_j)/T} & \text{otherwise.} \end{cases} \tag{4.21}$$

Let ϕ_i be the probability density function for random variable $\ddot{C}(i)$. By (4.21) this unfolds:

$$\begin{aligned}
\frac{\ddot{A}_{ji}}{\ddot{A}_{ij}} &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \phi_i(x_i)\phi_j(x_j)\frac{a(x_j, x_i)}{a(x_i, x_j)} dx_i dx_j && (4.22)\\
&= \int_{-\infty}^{\infty} \phi_i(x_i)\left[\int_{x_j > x_i} \phi_j(x_j)\frac{1}{e^{(C(i)+x_i-C(j)-x_j)/T}} dx_j\right. && (4.23)\\
&\qquad\qquad \left. + \int_{x_j \leq x_i} \phi_j(x_j)\frac{e^{(C(j)+x_j-C(i)-x_i)/T}}{1} dx_j\right] dx_i \\
&= \int_{-\infty}^{\infty} \phi_i(x_i)e^{(-C(i)-x_i)/T} dx_i \int_{-\infty}^{\infty} \phi_j(x_j)e^{(C(j)+x_j)/T} dx_j && (4.24)\\
&= \frac{e^{C(j)/T}\mathrm{E}[e^{X_j/T}]}{e^{C(i)/T}E[e^{X_i/T}]} && (4.25)
\end{aligned}$$

For any $G_{ij} \neq 0$, the detailed balance equation (4.8) is now

$$\ddot{\boldsymbol{\pi}}_i e^{C(i)/T}\mathrm{E}[e^{X_i/T}] = \ddot{\boldsymbol{\pi}}_j e^{C(j)/T}\mathrm{E}[e^{X_j/T}]. \tag{4.26}$$

By (4.19), $\ddot{\boldsymbol{\pi}}$ is a probability distribution ($\sum_{s\in S}\ddot{\boldsymbol{\pi}}_s = 1$). (4.19) satisfies (4.26) for all state pairs. Since $\ddot{P}$ is ergodic, (4.19) must be the unique probability distribution.

■

Corollary 4 *If all X_s are identically distributed, and the resulting Markov chain is ergodic, then $\ddot{\boldsymbol{\pi}} = \boldsymbol{\pi}$.*

PROOF. This follows directly from the proof of Lemma 3.

■

Lemma 5 *If X is a Gaussian random variable with mean μ and standard deviation σ, then*

$$\mathrm{E}[e^{-X/T}] = e^{\sigma^2/2T^2-\mu/T} \tag{4.27}$$

PROOF. By definition,

$$\begin{aligned}
\mathrm{E}[e^{-X/T}] &= \int_{-\infty}^{\infty} e^{-x/T}\frac{1}{\sqrt{2\pi}\sigma}e^{-(x-\mu)^2/2\sigma^2}dx \\
&= \frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty} e^{-x/T-x^2/2\sigma^2+2x\mu/2\sigma^2-\mu^2/2\sigma^2}dx && (4.28)\\
&= \frac{e^{-\mu^2/2\sigma^2}}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty} e^{-[x^2-2x\mu+2\sigma^2x/T]/2\sigma^2}dx && (4.29)\\
&= \frac{e^{-\mu^2/2\sigma^2}}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty} e^{-[x^2+2x(\sigma^2/T-\mu)+(\sigma^2/T-\mu)^2-(\sigma^2/T-\mu)^2]/2\sigma^2}dx && (4.30)\\
&= \frac{e^{-\mu^2/2\sigma^2+(\sigma^2/T-\mu)^2/2\sigma^2}}{\sqrt{2\pi}\sigma}\int_{-\infty}^{\infty} e^{-(x+\sigma^2/T-\mu)^2/2\sigma^2}dx && (4.31)
\end{aligned}$$

Let $u = (x+\sigma^2/T-\mu)/\sqrt{2}\sigma$ and $du = dx/\sqrt{2}\sigma$. Then

$$\mathrm{E}[e^{-X/T}] = \frac{e^{\sigma^2/2T^2-\mu/T}}{\sqrt{\pi}}\int_{-\infty}^{\infty} e^{-u^2}du. \tag{4.32}$$

By a standard mathematical trick, convert Cartesian to polar coordinates and obtain

$$\mathrm{E}[e^{-X/T}] = e^{\sigma^2/2T^2-\mu/T}. \tag{4.33}$$

■

Theorem 6 *Let $C\colon S \to \mathbb{R}$ be a cost-function with equilibrium distribution $\boldsymbol{\pi}$. Consider a random cost-function $\ddot{C}\colon S \to \mathbb{R}$, where each random variable $\ddot{C}(s)$*

is an independent Gaussian distribution with mean $C(s)$ and variance $\sigma^2(s)$ and equilibrium distribution $\ddot{\boldsymbol{\pi}}$. Then $\ddot{\boldsymbol{\pi}}_s$ is bounded by

$$e^{(\underline{\sigma}^2-\overline{\sigma}^2)/2T^2}\boldsymbol{\pi}_s \leq \ddot{\boldsymbol{\pi}}_s \leq e^{(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}\boldsymbol{\pi}_s \tag{4.34}$$

Proof. By Lemmas 3 and 5,

$$\ddot{\boldsymbol{\pi}}_s = \frac{e^{\sigma^2(s)/2T^2-C(s)/T}}{\sum_{s'\in S} e^{\sigma^2(s')/2T^2-C(s')/T}} \tag{4.35}$$

Fix s. To maximize $\ddot{\boldsymbol{\pi}}_s$, maximize the numerator at $e^{\overline{\sigma}^2/2T^2-C(s)/T}$ and minimize the denominator at $Z = e^{\overline{\sigma}^2/2T^2}e^{-C(s)/T} + \sum_{s'\neq s} e^{\underline{\sigma}^2/2T^2}e^{C(s')/T}$. The inequality

$$Z \geq \sum_{s'\in S} e^{\underline{\sigma}^2/2T^2}e^{C(s')/T} \tag{4.36}$$

produces

$$\ddot{\boldsymbol{\pi}}_s \leq \frac{e^{\overline{\sigma}^2/2T^2}e^{C(s)/T}}{\sum_{s'\in S} e^{\underline{\sigma}^2/2T^2}e^{C(s)/T}} = e^{(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}\boldsymbol{\pi}_s \tag{4.37}$$

A similar argument supplies the lower bound.

■

Theorem 7 *Let $\bar{C}[\boldsymbol{\pi}(T)]$ be the expected true-cost C, with equilibrium distribution $\boldsymbol{\pi}$, at temperature T. Then,*

$$e^{-(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}\bar{C}[\boldsymbol{\pi}(T)] \leq \bar{C}[\ddot{\boldsymbol{\pi}}(T)] \leq e^{(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}\bar{C}[\boldsymbol{\pi}(T)] \tag{4.38}$$

Proof. By Theorem 6,

$$e^{-(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}\sum_{s\in S} C(s)\boldsymbol{\pi}_s(T) \leq \sum_{s\in S} C(s)\ddot{\boldsymbol{\pi}}_s(T) \leq e^{(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}\sum_{s\in S} C(s)\boldsymbol{\pi}_s(T) \tag{4.39}$$

Definition (4.16) gives the result.

■

These results show that under Gaussian errors, the variance range determines equilibrium state probabilities and average cost. As the variance-range increases and temperature decreases, the effect on equilibrium properties increases exponentially.

4.4 Mixing Speed

Mixing speed is the rate at which an ergodic process approaches equilibrium. A higher mixing speed indicates that you can more rapidly reduce the temperature while maintaining near-equilibrium. Errors affect mixing speed, not necessarily for the worse.

To quantify mixing speed, you must first have a measure for the distance from equilibrium. Let P be the transition matrix for the annealing chain, let $\boldsymbol{\pi}$ be the stationary probability vector over state space S. Let t denote time, and $\mathbf{x}(t)$ denote state probabilities at time t, so $\mathbf{x}(t) = P^t\mathbf{x}(0)$. Define the discrepancy vector $\mathbf{d}(t) = \mathbf{x}(t) - \boldsymbol{\pi}$. Define the distance from equilibrium as

$$\|\mathbf{d}(t)\| = \sum_{s\in S} \mathbf{d}_s^2(t). \tag{4.40}$$

If P is a Markov chain and $\boldsymbol{\pi}_i$ is the stationary probability of state i, define the *conductance of a subset*, Φ_V, $V \in S$ as

$$\Phi_V = \frac{\sum_{i\in V}\sum_{j\notin V} \boldsymbol{\pi}_i P_{ij}}{\sum_{i\in V} \boldsymbol{\pi}_i} \tag{4.41}$$

In words, the conductance of a state subset V is the conditional probability that some transition will leave the subset, given that you start there. Figure 4.6 illustrates this concept.

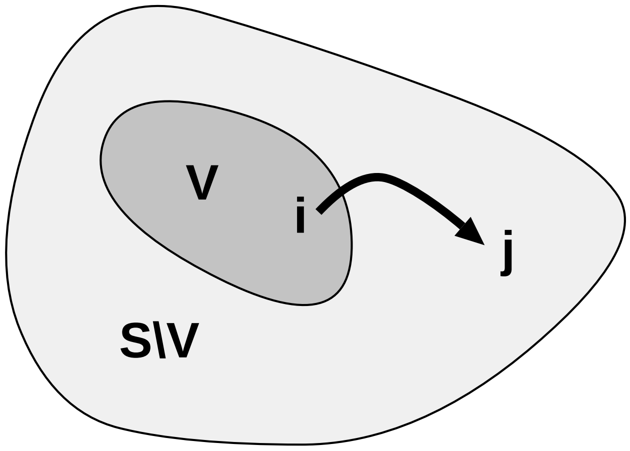


Figure 4.6: Conductance of Subset V

Let $\boldsymbol{\pi}_V = \sum_{v \in V} \boldsymbol{\pi}_v$, and let $\mathbf{S}_{1/2} = \{V \subset S | \boldsymbol{\pi}_V \leq 1/2\}$. Define the *global conductance*, Φ, as the minimum conductance over all subsets with stationary probability below 1/2, thus,

$$\Phi = \min_{V \in \mathbf{S}_{1/2}} \Phi_V \tag{4.42}$$

Sinclair and Jerrum showed that any initial distribution of a strongly aperiodic Markov chain, such as defined in (4.3)–(4.7), with discrepency vector $\mathbf{d}(0)$, satisfies (4.43) [SJ89].

$$\|\mathbf{d}(t)\| \leq (1 - \Phi^2)^t \cdot \|\mathbf{d}(0)\| \tag{4.43}$$

Thus, global conductance provides a good measure for mixing speed.

Another commonly-used mixing speed measure is the “dominant eigenvalue.”

The convergence rate of a simulated annealing algorithm is inversely related to the eigenvalues of its underlying Markov chain [BS87, OG89]. The eigenvalues measure the "ruggedness" of the cost landscape. The second largest eigenvalue (or "dominant eigenvalue"), in particular, provides a first-approximation to the slowness of reaching equilibrium from an arbitrary starting state.

Calculating the eigenvalues for annealing spaces is typically intractable, however a useful bound relates the more easily computed global conductance to the dominant eigenvalue [SJ89]. Sinclair and Jerrum showed that for an ergodic reversible Markov chain, the dominant eigenvalue $\boldsymbol{\lambda}_2$ of the transition matrix satisfies (4.44).

$$\boldsymbol{\lambda}_2 \leq 1 - \frac{\Phi^2}{2}. \tag{4.44}$$

Thus, by raising the global conductance, you reduce the dominant eigenvalue and increase the convergence rate.

Others have examined the effect of adjusting the move spaces to obtain better mixing speed [Sor92, OG89]. In the following two sections, I show how the errors modify mixing speed, using global conductance as a measure.

4.4.1 Mixing Speed Under Range-Errors

Consider two functions: the *true-cost* given by C, and the *observed-cost* given by random function $\dot{C}$. Suppose $\dot{C}$ is bounded relative to C: $\forall s \in S, C(s) + \underline{\epsilon} \leq \dot{C}(s) \leq C(s) + \overline{\epsilon}$. As before, let $\gamma = \overline{\epsilon} - \underline{\epsilon}$. What is the effect of these errors on the global conductance?

Lemma 8 *Consider two annealing chains P and $\dot{P}$, with G and S defined identically. They have cost-functions C and $\dot{C}$, respectively. The two cost-functions are related by $C(s) + \underline{\epsilon} \leq \dot{C} \leq C(s) + \overline{\epsilon}$. Let Φ_V and $\dot{\Phi}_V$ be the corresponding conductances for subset $V \subseteq S$. Then,*

$$e^{-3\gamma/T}\dot{\Phi}_V \leq \Phi_V \leq e^{3\gamma}\dot{\Phi}_V \tag{4.45}$$

Proof. Let $V' = S \setminus V$. The conductance of $V \subset S$, under cost-function $\dot{C}$ is

$$\dot{\Phi}_V = \frac{\sum_{i\in V}\sum_{j\in V'} \dot{\boldsymbol{\pi}}_i G_{ij} \dot{A}_{ij}}{\sum_{i\in V} \dot{\boldsymbol{\pi}}_i} \tag{4.46}$$

Construct an upper bound on Φ_V. For any vertex $i \in V$, you can lower its transition probability to any $j \in V'$ by a factor of at most $e^{-\gamma}$. This follows directly from the definition of A and $\dot{C}$. Minimize the acceptance term thus: $\dot{A}_{ij} = A_{ij}e^{-\gamma/T}$. I assume that the generate probability matrix G does not change. Thus, you have

$$\dot{\Phi}_V \geq e^{-\gamma/T}\frac{\sum_{i\in V}\sum_{j\in V'} \dot{\boldsymbol{\pi}}_i G_{ij} A_{ij}}{\sum_{i\in V} \dot{\boldsymbol{\pi}}_i} \tag{4.47}$$

Using Theorem 1,

$$\dot{\Phi}_V \geq e^{-\gamma/T}\frac{\sum_{i\in V}\sum_{j\in V'} e^{-\gamma/T}\boldsymbol{\pi}_i G_{ij} A_{ij}}{\sum_{i\in V} e^{+\gamma}\boldsymbol{\pi}_i}, \tag{4.48}$$

which implies

$$\Phi_V \leq e^{+3\gamma/T}\dot{\Phi}_V. \tag{4.49}$$

A similar construction supplies the lower bound.

■

Theorem 9 *Let Φ and $\dot{\Phi}$ be the global conductances of annealing chains P and $\dot{P}$. Then,*

$$e^{-3\gamma/T}\Phi \leq \dot{\Phi} \leq e^{+3\gamma/T}\Phi \tag{4.50}$$

Proof. Let $\dot{\mathbf{S}}_{1/2} = \{V \subset S | \dot{\boldsymbol{\pi}}_V \leq 1/2\}$. Suppose

$$\min_{V \subset \dot{\mathbf{S}}_{1/2}} \dot{\Phi}_V = \dot{\Phi}_A \tag{4.51}$$

for some set $W \in \dot{\mathbf{S}}_{1/2}$. There are two cases.

Case 1, $W \in \mathbf{S}_{1/2}$.

By Lemma 8, Theorem 9 holds.

Case 2, $W \notin \mathbf{S}_{1/2}$.

By definition,

$$\dot{\Phi}_W = \frac{\sum_{i \in W} \sum_{j \notin W} \dot{\boldsymbol{\pi}}_i \dot{P}_{ij}}{\sum_{i \in W} \dot{\boldsymbol{\pi}}_i}. \tag{4.52}$$

Let $W' = S \setminus W$. $\dot{P}$ satisfies the detailed balance conditions (4.8), and thus

$$\sum_{i \in W} \sum_{j \notin W} \dot{\boldsymbol{\pi}}_i \dot{P}_{ij} = \sum_{i \in W'} \sum_{j \notin W'} \dot{\boldsymbol{\pi}}_j \dot{P}_{ji}, \tag{4.53}$$

so (4.52) becomes

$$\dot{\Phi}_W = \frac{\sum_{i \in W'} \sum_{j \notin W'} \dot{\boldsymbol{\pi}}_i \dot{P}_{ij}}{\sum_{i \in W} \dot{\boldsymbol{\pi}}_i}. \tag{4.54}$$

$W \in \dot{\mathbf{S}}_{1/2}$ implies that $\Rightarrow \dot{\boldsymbol{\pi}}_W \leq 1/2$. Therefore, $\dot{\boldsymbol{\pi}}_{W'} \geq 1/2$. Thus, $\dot{\boldsymbol{\pi}}_{W'} \geq \dot{\boldsymbol{\pi}}_W$ and

$$\dot{\Phi}_W \geq \frac{\sum_{i \in W'} \sum_{j \notin W'} \dot{\boldsymbol{\pi}}_i \dot{P}_{ij}}{\sum_{i \in W'} \dot{\boldsymbol{\pi}}_i} = \dot{\Phi}_{W'}. \tag{4.55}$$

By Lemma 8, you have

$$\dot{\Phi}_W \geq \dot{\Phi}_{W'} \geq e^{-3\gamma/T}\Phi_{W'}. \tag{4.56}$$

Note that $W' \in \mathbf{S}_{1/2}$. By (4.42), you have $\dot{\Phi}_W \geq e^{-3\gamma/T}\Phi$, and $\dot{\Phi} \geq e^{-3\gamma/T}\Phi$. Thus

$$e^{-3\gamma/T}\Phi \leq \dot{\Phi}. \tag{4.57}$$

A similar proof, starting with $\mathbf{S}_{1/2}$ instead of $\dot{\mathbf{S}}_{1/2}$, supplies the upper bound in (4.50).

■

This result predicts the rate that an inaccurate system equilibrates, if you know the maximum error magnitude and the rate that the accurate system equilibrates.

4.4.2 Mixing Speed Under Gaussian Errors

Now consider the mixing speed of Gaussian cost-functions. Substitute $\ddot{C}$ for $\dot{C}$ in (4.46). The conductance of a subset $V \subset S$ is defined by

$$\ddot{\Phi}_V = \frac{\sum_{i\in V}\sum_{j\notin V} \ddot{\boldsymbol{\pi}}_i G_{ij} \ddot{A}_{ij}}{\sum_{i\in V} \ddot{\boldsymbol{\pi}}_i}. \tag{4.58}$$

Lemma 10 (Conductance Reversibility) *Consider state space S, and subsets $V \subset S$ and $V' = S\backslash V$. Let Φ_V be the conductance of V, and $\boldsymbol{\pi}_V$ be the equilibrium probability of set V. If the underlying Markov space has the reversibility property, then*

$$\Phi_V \boldsymbol{\pi}_{V'} = \Phi_{V'} \boldsymbol{\pi}_V \tag{4.59}$$

PROOF. The conductance of V is

$$\Phi_V = \frac{\sum_{i \in V} \sum_{j \in V'} \boldsymbol{\pi}_i P_{ij}}{\sum_{i \in V} \boldsymbol{\pi}_i}. \tag{4.60}$$

The conductance of V' is

$$\Phi_V = \frac{\sum_{i \in V} \sum_{j \in V'} \boldsymbol{\pi}_j P_{ji}}{\sum_{i \in V'} \boldsymbol{\pi}_i}. \tag{4.61}$$

By the reversibility property, the numerators of (4.60) and (4.61) are equal. Algebra produces (4.59).

■

Lemma 11 *Apply Gaussian-error cost-function to states $i, j \in S$, as described previously, then the acceptance probability $\ddot{A}$ behaves according to the inequality*

$$\ddot{A}_{ij} \leq e^{\overline{\sigma}^2/T^2} A_{ij}. \tag{4.62}$$

PROOF. Let random variables X and Y be defined so $X = \ddot{C}(i)$ and $Y = \ddot{C}(j)$. By the definition of A,

$$\ddot{A}_{ij} = \int_{-\infty}^{\infty} \left[\int_{x}^{\infty} e^{(x-y)/T} \phi_j(y)\, dy \quad + \int_{-\infty}^{x} \phi_j(y)\, dy \right] \phi_i(x) dx. \tag{4.63}$$

Since $y \geq x \Rightarrow e^{(x-y)/T} \leq 1$,

$$\ddot{A}_{ij} \leq \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \phi_j(y) \phi_i(x)\, dy\, dx. \tag{4.64}$$

Separating independent integrations,

$$\ddot{A}_{ij} \leq \int_{-\infty}^{\infty} \phi_i(x)\, dx \int_{-\infty}^{\infty} \phi_j(y)\, dy. \tag{4.65}$$

By Lemma 5,

$$\ddot{A}_{ij} \leq e^{\sigma_i^2/2T^2 + C(i)/T} e^{\sigma_j^2/2T^2 - C(j)/T}, \tag{4.66}$$

and therefore

$$\ddot{A}_{ij} \leq e^{\sigma_i^2/2T^2 + \sigma_j^2/2T^2} e^{(C(i)-C(j))/T}. \tag{4.67}$$

Suppose $C(i) \leq C(j)$, then $e^{(C(i)-C(j))/T} = A_{ij}$ and Equation 4.62 holds. Suppose $C(i) \geq C(j)$ then $A_{ij} = 1$. Since $e^{2\overline{\sigma}^2/T^2} \geq 1$ and since $\ddot{A}_{ij} \leq 1$, Equation 4.62 holds here also.

■

Lemma 12 *Let $\ddot{C}$ be a random function where each random variable $\ddot{C}(s)$ is a Gaussian. $\ddot{C}(s)$ has mean value $C(s)$. Let A_{ij} and $\ddot{A}_{ij}$ be the acceptance probabilities for an annealing transition from state i to state j under cost-functions C and $\ddot{C}$ respectively. Then A and $\ddot{A}$ are related by*

$$\ddot{A}_{ij} \geq \frac{1}{2} A_{ij}. \tag{4.68}$$

PROOF. Examine the transition from state a to state b. There are two cases:

Case 1, $C(i) \geq C(j)$.

In this case, $A_{ij} = 1$. Let ϕ_i and ϕ_j be the probability densities of random variables $\ddot{C}(i)$ and $\ddot{C}(j)$. Restating (4.63), the transition probability under $\ddot{C}$ is

$$\begin{aligned} A_{ij} &= \int_{-\infty}^{\infty} \phi_i(x) \int_{-\infty}^{x} \phi_j(y) \cdot 1 \; dy \; dx \qquad (4.69) \\ &\quad + \int_{-\infty}^{\infty} \phi_i(x) \int_{x}^{\infty} \phi_j(y) e^{-(y-x)/T} dy \; dx \end{aligned}$$

If $C(i) \geq C(j)$ then the first term always exceeds 1/2. Therefore, $\ddot{A}_{ij} \geq \frac{1}{2}A_{ij}$.

Case 2, $C(i) < C(j)$.

Now, $A_{ij} = e^{-(C(j)-C(i))/T}$. Restate (4.69) slightly differently for this case:

$$\begin{aligned} A_{ij} &= \int_{-\infty}^{\infty} \phi_i(x) \int_{-\infty}^{x} \phi_j(y) \cdot 1 \; dy \; dx \\ &+ \int_{-\infty}^{\infty} \phi_i(x) \int_{x}^{x+C(j)-C(i)} \phi_j(y) e^{-(y-x)/T} dy \; dx \\ &+ \int_{-\infty}^{\infty} \phi_i(x) \int_{x+C(j)-C(i)}^{\infty} \phi_j(y) e^{-(C(j)-C(i))/T} dy \; dx \end{aligned} \tag{4.70}$$

Examine the first term of (4.70), and note that

$$\begin{aligned} & e^{-(C(j)-C(i))/T} \int_{-\infty}^{\infty} \phi_i(x) \int_{-\infty}^{x} \phi_j(y) \; dy \; dx \\ \leq & \int_{-\infty}^{\infty} \phi_i(x) \int_{-\infty}^{x} \phi_j(y) \; dy \; dx. \end{aligned} \tag{4.71}$$

Examine the second term of (4.70). I assert that

$$\begin{aligned} & \int_{-\infty}^{\infty} \phi_i(x) \int_{x}^{x+C(j)-C(i)} \phi_j(y) e^{-(C(j)-C(i))/T} dy \; dx \\ & \leq \int_{-\infty}^{\infty} \phi_i(x) \int_{x}^{x+C(j)-C(i)} \phi_j(y) e^{-(y-x)/T} dy \; dx \end{aligned} \tag{4.72}$$

Combining (4.70) with (4.71) and (4.72), obtain

$$\begin{aligned} \ddot{A}_{ij} \;\; \geq \;\; & e^{-(C(j)-C(i))/T} \int_{-\infty}^{\infty} \phi_i(x) \int_{-\infty}^{x+C(j)-C(i)} \phi_j(y) \cdot 1 \; dy \; dx \\ & + \int_{-\infty}^{\infty} \phi_i(x) \int_{x+C(j)-C(i)}^{\infty} \phi_j(y) e^{-(y-x)/T} dy \; dx \end{aligned} \tag{4.73}$$

Since the means of both ϕ_i and ϕ_j are less than $x + C(j) - C(i)$, the first term is no less than $\frac{1}{2}e^{-(C(j)-C(i))/T}$. Thus,

$$\ddot{A}_{ij} \geq \frac{1}{2} A_{ij}. \tag{4.74}$$

■

Lemma 13 *Consider two annealing chains P and $\ddot{P}$, with the same generate probability matrix G and state space S. Chain P has cost-function C, and chain $\ddot{P}$ has Gaussian random cost-function $\ddot{C}$. The two cost-functions are related by* $\mathrm{E}[\ddot{C}(s)] = C(s)$. *The variance of each random variable $\ddot{C}(s)$, σ_s^2, is constrained by $\underline{\sigma}^2 \leq \sigma_s^2 \leq \overline{\sigma}^2$. Then the conductances of a subset $V \in S$ in the two chains are related by*

$$\frac{1}{2}e^{(\underline{\sigma}^2-\overline{\sigma}^2)/T^2}\Phi_V \leq \ddot{\Phi}_V \leq e^{(2\overline{\sigma}^2-\underline{\sigma}^2)/T^2}\Phi_V. \tag{4.75}$$

Proof. By definition (4.42),

$$\ddot{\Phi}_V = \frac{\sum_{i\in V}\sum_{j\notin V}\ddot{\boldsymbol{\pi}}_i G_{ij}\ddot{A}_{ij}}{\sum_{i\in V}\ddot{\boldsymbol{\pi}}_i}. \tag{4.76}$$

By Theorem 6 and Lemma 11,

$$\ddot{\Phi}_V \leq \frac{\sum_{i\in V}\sum_{j\notin V}\boldsymbol{\pi}_i e^{(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}G_{ij}A_{ij}e^{\overline{\sigma}^2/T^2}}{\sum_{i\in S}e^{(\underline{\sigma}^2-\overline{\sigma}^2)/2T^2}\boldsymbol{\pi}_i}. \tag{4.77}$$

By simple algebra obtain

$$\ddot{\Phi}_V \leq e^{(2\overline{\sigma}^2-\underline{\sigma}^2)/T^2}\Phi_V. \tag{4.78}$$

By Theorem 6 and Lemma 12,

$$\frac{\sum_{i\in V}\sum_{j\notin V}\boldsymbol{\pi}_i e^{(\underline{\sigma}^2-\overline{\sigma}^2)/2T^2}G_{ij}\frac{1}{2}A_{ij}}{\sum_{i\in S}e^{(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}\boldsymbol{\pi}_i} \leq \ddot{\Phi}_V \tag{4.79}$$

and by algebra obtain

$$\frac{1}{2}e^{(\underline{\sigma}^2-\overline{\sigma}^2)/T}\Phi_V \leq \ddot{\Phi}_V. \tag{4.80}$$

Combining (4.78) and (4.80), obtain (4.75).

■

Theorem 14 *Consider two annealing chains P and $\ddot{P}$, with the same generate probability matrix G and state space S. Chain P has cost-function C, and chain $\ddot{P}$ has Gaussian random cost-function $\ddot{C}$. The two cost-functions are related by* $\mathrm{E}[\ddot{C}(s)] = C(s)$. *The variance of each random variable $\ddot{C}(s)$, σ_s^2, is constrained by* $\underline{\sigma}^2 \leq \sigma_s^2 \geq \overline{\sigma}^2$. *Then the global conductances of the two chains are related by*

$$\frac{1}{2} e^{(\underline{\sigma}^2 - \overline{\sigma}^2)/T^2} \Phi \leq \ddot{\Phi} \leq e^{(3\overline{\sigma}^2 - 2\underline{\sigma}^2)/T^2} \Phi. \tag{4.81}$$

Proof. By definition,

$$\ddot{\Phi} = \min_{V \in \ddot{\mathbf{S}}_{1/2}} \ddot{\Phi}_V. \tag{4.82}$$

Choose any $V \in \ddot{\mathbf{S}}_{1/2}$. There are two cases.

Case 1, $V \in \mathbf{S}_{1/2}$.

Let $W_V = V$. By Lemma 13,

$$\frac{1}{2} e^{(\underline{\sigma}^2 - \overline{\sigma}^2)/T^2} \Phi_{W_V} \leq \ddot{\Phi}_V. \tag{4.83}$$

This ends case 1.

Case 2, $V \notin \mathbf{S}_{1/2}$.

Let $W_V = S \setminus V$. If $V \notin \mathbf{S}_{1/2}$, then $\boldsymbol{\pi}_V > 1/2$. Therefore, $1 - \boldsymbol{\pi}_V = \boldsymbol{\pi}_{W_V} < 1/2$ and $W_V \in \mathbf{S}_{1/2}$.

By Lemma 10

$$\Phi_V \boldsymbol{\pi}_{W_V} = \Phi_{W_V} \boldsymbol{\pi}_V, \tag{4.84}$$

therefore

$$\Phi_{W_V} \frac{\boldsymbol{\pi}_V}{\boldsymbol{\pi}_{W_V}} = \Phi_V. \tag{4.85}$$

$\boldsymbol{\pi}_{W_V} < 1/2$ and $\boldsymbol{\pi}_V > 1/2$, so

$$\Phi_{W_V}\frac{(1/2)}{(1/2)} < \Phi_V. \tag{4.86}$$

By Lemma 13,

$$\frac{1}{2}e^{(\underline{\sigma}^2-\overline{\sigma}^2)/T^2}\Phi_{W_V} < \ddot{\Phi}_V. \tag{4.87}$$

This ends case 2.

Let $\mathbf{S}_W = \{W_V : V \in \ddot{\mathbf{S}}_{1/2}\}$, and let

$$\Phi_{\mathbf{S}_W} = \min_{W\in\mathbf{S}_W} \Phi_{W_V}. \tag{4.88}$$

Then by (4.83), (4.87), and the definition of $\ddot{\Phi}$,

$$\frac{1}{2}e^{(\underline{\sigma}^2-\overline{\sigma}^2)/T^2}\Phi_{\mathbf{S}_W} < \ddot{\Phi} \tag{4.89}$$

But $\mathbf{S}_W \subseteq \mathbf{S}_{1/2}$, so by definition of $\mathbf{S}_{1/2}$,

$$\frac{1}{2}e^{(\underline{\sigma}^2-\overline{\sigma}^2)/T^2}\Phi \leq \frac{1}{2}e^{(\underline{\sigma}^2-\overline{\sigma}^2)/T^2}\Phi_{\mathbf{S}_W} < \ddot{\Phi} \tag{4.90}$$

This proves the leftmost inequality in (4.81).

Now I prove the rightmost inequality. This second half of the proof follows the reasoning of the first. I provide it because the factors and results are slightly different.

By definition,

$$\Phi = \min_{V\in\mathbf{S}_{1/2}} \Phi_V. \tag{4.91}$$

Choose any $V \in \mathbf{S}_{1/2}$. There are two cases.

Case 1, $V \in \ddot{\mathbf{S}}_{1/2}$.

Let $X_V = V$. By Lemma 13,

$$\ddot{\Phi}_V \leq e^{(2\overline{\sigma}^2 - \underline{\sigma}^2)/T^2} \Phi_V. \tag{4.92}$$

This ends case 1.

Case 2, $V \notin \ddot{\mathbf{S}}_{1/2}$.

Let $X_V = S \setminus V$. If $V \notin \ddot{\mathbf{S}}_{1/2}$, then $\ddot{\boldsymbol{\pi}}_V > 1/2$. Therefore, $1 - \ddot{\boldsymbol{\pi}}_V = \ddot{\boldsymbol{\pi}}_{X_V} < 1/2$ and $X_V \in \ddot{\mathbf{S}}_{1/2}$.

By Lemma 10

$$\ddot{\Phi}_V \ddot{\boldsymbol{\pi}}_{X_V} = \ddot{\Phi}_{X_V} \ddot{\boldsymbol{\pi}}_V, \tag{4.93}$$

therefore

$$\ddot{\Phi}_{X_V} \frac{\ddot{\boldsymbol{\pi}}_V}{\ddot{\boldsymbol{\pi}}_{X_V}} = \ddot{\Phi}_V. \tag{4.94}$$

$\ddot{\boldsymbol{\pi}}_{X_V} < 1/2$ and $\ddot{\boldsymbol{\pi}}_V > 1/2$, so

$$\ddot{\Phi}_{X_V} \frac{(1/2)}{(1/2)} < \ddot{\Phi}_V. \tag{4.95}$$

By Lemma 13,

$$\ddot{\Phi}_{X_V} < e^{(2\overline{\sigma}^2 - \underline{\sigma}^2)/T^2} \Phi_V. \tag{4.96}$$

This ends case 2.

Let $\ddot{\mathbf{S}}_X = \{X_V \colon V \in \mathbf{S}_{1/2}\}$, and let

$$\Phi_{\ddot{\mathbf{S}}_X} = \min_{X \in \ddot{\mathbf{S}}_X} \ddot{\Phi}_{X_V}. \tag{4.97}$$

Then by (4.92), (4.96), and the definition of Φ,

$$\Phi_{\ddot{\mathbf{S}}_X} < e^{(2\overline{\sigma}^2 - \underline{\sigma}^2)/T^2}\Phi \tag{4.98}$$

But $\ddot{\mathbf{S}}_X \subseteq \ddot{\mathbf{S}}_{1/2}$, so by definition of $\ddot{\mathbf{S}}_{1/2}$,

$$\ddot{\Phi} \leq \Phi_{\ddot{\mathbf{S}}_X} < e^{(2\overline{\sigma}^2 - \underline{\sigma}^2)/T^2}\Phi. \tag{4.99}$$

(4.90) and (4.99) prove the theorem.

■

How do I interpret this result? Φ is equivalent to the speed at which the accurate annealing system equilibrates. $\ddot{\Phi}$ is the speed at which the inaccurate system equilibrates. Suppose you have an annealing schedule which keeps the accurate system at temperature T for time t_T. To guarantee the same level of equilibration in the inaccurate case, keep the system at temperature T for time

$$\ddot{t}_T = 2e^{(\overline{\sigma}^2 - \underline{\sigma}^2)/T^2} t_T \tag{4.100}$$

4.5 Convergence on General Spaces

On ergodic spaces, simulated annealing will provably converge monotonically to an optimum, with an appropriate temperature schedule. The most general results don't constrain the state space, other than by (4.3)–(4.7). You pay a time penalty for generality. The $T(t) = d/\log t$ temperature schedule, annealing a reversible graph, converges to a minimum cost state [Haj88], but it takes exponential time. Tsitsiklis generalized this result to non-reversible graphs [Tsi89].

I first present a result by Tsitsiklis without proof: interested readers can refer to the original paper. I then prove that constraining errors to a constant factor of temperature guarantees convergence.

Define global optima set S_0 such that $s \in S_0 \Rightarrow C(s_0) = \min\{C(s')|s' \in S\}$. With state space S, generate probability matrix G, and cost-function C, construct a delta matrix D^0 as follows.

$$D^0_{ij} = \begin{cases} \infty & \text{if } G_{ij} = 0 \\ C(j) - C(i) & \text{if } G_{ij} \neq 0 \text{ and } C(j) \geq C(i) \\ 0 & \text{otherwise} \end{cases} \tag{4.101}$$

If (4.3)–(4.7) are satisfied, D^0 captures enough information to compute equilibrium probabilities at any fixed temperature.

Now, construct the set of *transient states* $\not{R}$ so that

$$\not{R} = \{i \in S | \exists j \in S \text{ such that } D^0_{ij} = 0 \text{ and } D^0_{ji} \neq 0\} \tag{4.102}$$

All other states, $R = S \setminus \not{R}$, are termed *recurrent*. These recurrent states are the local minima. Construct a partitioning over R: for any $i \in R$, its equivalency class is $R_i = \{j \in R | D^0_{ij} = 0\}$.

Figure 4.7 shows a pruning algorithm. At pruning depth $d+1$, eliminate local minima of depth d. Informally, measure the depth by starting at some $i \in R$, and following the shortest path to a lower local minimum j. The difference between the greatest cost reached on the path and $C(i)$, is the depth of i.

Hajek describes this process as "how cups runneth over": the state variable has

Input: $N \times N$ matrix D^d.
Output: $N \times N$ matrices D^{d+1}, sets $\not{R}^d, R^d \subset S$, equivalency sets R_i^d.

1. Find the transient states $\not{R}^d$, recurrent states R^d, and equivalence classes R_i^d of D^d.
2. Let
$$C_{ij} = \begin{cases} D_{ij}^d - 1 & \text{if } i, j \in R^d \text{ and } j \notin R_i^d \\ D_{ij}^d & \text{otherwise} \end{cases} \tag{4.103}$$
3. For all $i \in R^d, j \in R^d$ solve the shortest path problem from i to j over D^d. Let D_{ij}^{d+1} be its length. (Note that $i \in R^d \Rightarrow D_{ii}^{d+1} = 0$.)
4. For all $i \in R^d, j \in \not{R}^d$, let
$$D_{ij}^{k+1} = \min_{k \in R^d} \{D_{ik}^{d+1} + C_{kj}\} = \min_{k \in R^d} \{D_{ik}^{d+1} + D_{kj}^d\} \tag{4.104}$$
5. If $i \in \not{R}^k$, let
$$D_{ij}^{k+1} = \min_{k \in R^d} \{C_{ik} + D_{kj}^{d+1}\} = \min_{k \in R^d} \{D_{ik}^d + D_{kj}^{d+1}\} \tag{4.105}$$

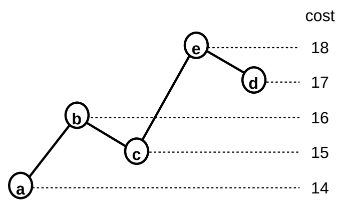
Figure 4.7: *Prune Minima* Algorithm

to "run over" some boundary to escape a local minima. The higher the boundary, the longer the algorithm takes to crest it.

The recurrent state sets constructed by this algorithm have important properties. First, $R^{d+1} \subset R^d$. Second, if the system satisfies (4.3)–(4.7), then some $d' \in \mathbb{Z}^{0+}$ produces $R^{d'} = S_0$, the set of all minimum cost states.

Figure 4.8 shows an example space, and its delta matrix. Prune the annealing space by iterating with *Prune Minima*. The first iteration yields $\not{R}^0 = \not{R}, R^0 = R, C^0$, and D^1.

$$\not{R}^0 = \{b, d\}, \quad R^0 = \{a, c, e\} \tag{4.106}$$



$$D^0 = \begin{array}{c|ccccc|} & a & b & c & d & e \\ a & \infty & 2 & \infty & \infty & \infty \\ b & 0 & \infty & 0 & \infty & \infty \\ c & \infty & 1 & \infty & 3 & \infty \\ d & \infty & \infty & 0 & \infty & 0 \\ e & \infty & \infty & \infty & 1 & \infty \end{array}$$

Figure 4.8: Pruning Example

$$C^0 = \begin{bmatrix} \infty & 2 & \infty & \infty & \infty \\ 0 & \infty & 0 & \infty & \infty \\ \infty & 1 & \infty & 3 & \infty \\ \infty & \infty & 0 & \infty & 0 \\ \infty & \infty & \infty & 1 & \infty \end{bmatrix}, \quad D^1 = \begin{bmatrix} 0 & 2 & 2 & 5 & 5 \\ 0 & 1 & 0 & 3 & 3 \\ 1 & 1 & 0 & 3 & 3 \\ 1 & 1 & 0 & 1 & 0 \\ 2 & 2 & 1 & 1 & 0 \end{bmatrix} \qquad (4.107)$$

The second iteration yields $\not{R}^1, R^1, C^1$, and D^2.

$$\not{R}^1 = \{b, d\}, \quad R^1 = \{a, c, e\} \qquad (4.108)$$

$$C^1 = \begin{bmatrix} 0 & 2 & 1 & 5 & 4 \\ 0 & 1 & 0 & 3 & 3 \\ 0 & 1 & 0 & 3 & 2 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 2 & 0 & 1 & 0 \end{bmatrix}, \quad D^2 = \begin{bmatrix} 0 & 2 & 1 & 4 & 3 \\ 0 & 1 & 0 & 3 & 2 \\ 0 & 1 & 0 & 3 & 2 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad (4.109)$$

The third iteration yields

$$\not{R}^2 = \{b, c, d, e\}, \;\; R^2 = \{a\}, \;\; C^2 = D^2, \;\; D^3 = D^2. \tag{4.110}$$

Further iterations produce no changes. Thus, $R^2 = \{a\}$ is the set of global minima for this example, and the maximum depth of local minima is 1.

Let P^ε be a stochastic matrix (whose ijth entry is denoted by p^ε_{ij}), parameterized by a positive parameter ε. Assume that there exist positive constants C_1, C_2, and a collection $\mathcal{A} = \{\alpha_{ij} \colon 1 \le i, j \le N\}$ such that $\alpha_{ij} \in \mathcal{N}_0 \cup \{\infty\}, \forall i, j$ and such that $p^\varepsilon_{ij} = 0$ whenever $\alpha_{ij} = \infty$, and $C_1 \varepsilon^{\alpha_{ij}} \le p^\varepsilon_{ij} \le C_2 \varepsilon^{\alpha_{ij}}$ whenever $\alpha_{ij} < \infty$. Finally, assume a monotonically nonincreasing function (temperature schedule) $\varepsilon \colon \mathcal{N}_0 \to (0, 1)$.

Consider the Markov chain $X(t)$ with transition probabilities given by $P(X(t+1) = j | X(t) = i) = p^\varepsilon_{ij}$. If you let $\varepsilon = e^{-1/T(t)}$, and if

$$\alpha_{ij} = \begin{cases} \infty & \text{if } G_{ij} = 0 \\ \max\{0, C(j) - C(i)\} & \text{otherwise,} \end{cases} \tag{4.111}$$

then $X(t)$ is the same as simulated annealing. Classify the states as recurrent or transient: State i is transient iff $\exists j \in S, \alpha_{ij} = 0 \wedge \alpha_{ji} \neq 0$. Otherwise, i is recurrent.

Assumption: If at least one of the states i, j, and k are recurrent, then $\alpha_{ik} \le \alpha_{ij} + \alpha_{jk}$.

Theorem 15 (Tsitsiklis) *Assume that for some integer $d \ge 0$,*

$$\sum_{t=0}^{\infty} \varepsilon^d(t) = \infty \tag{4.112}$$

$$\sum_{t=0}^{\infty} \varepsilon^{d+1}(t) < \infty \tag{4.113}$$

Then

1. $\forall i \in S, \lim_{t\to\infty} P(X(t) \in R^d | X(0) = i) \;=\; 1.$

2. *For any* $i \in R^d$, $\limsup_{t\to\infty} P(X(t) = i | X(0) = i) > 0.$

Corollary 16 (Tsitsiklis) *Let the transition probabilities for simulated annealing be given by (4.1) and (4.2). Consider temperature schedules of the form* $T(t) = c/\log t$. *For any initial state, the algorithm converges in probability to the set of global minima,* S_0, *if and only if there exists some* d *such that* $R^d \subset S_0$, *then* c *is larger than or equal to the smallest such* d, *to be denoted by* d^*.

Theorem 17 *Suppose a inaccurate simulated annealing chain satisfies (4.3)–(4.7) and has a cost-function* $\dot{C}(s,t)$, *with time-dependent errors*

$$C(s) + \underline{\epsilon}(t) \le \dot{C}(s,t) \le C(s) + \overline{\epsilon}(t). \tag{4.114}$$

Let S_0 *be the set of minimum cost states in* S.

Let $b_1 T(t) \le \underline{\epsilon}(t) \le \overline{\epsilon}(t) \le b_2 T(t)$, *where* b_1 *and* b_2 *are constants. Let the temperature schedule be of the form* $T(t) = d/\log t$, *where* $d \in \mathbb{Z}^{0+}$. *Then the algorithm converges in probability to the set of global minima if* $R^d \subset S_0$.

Proof. Suppose the transition matrix P^t for some Markovian system at time t is constrained by (4.115).

$$c_1 e^{-D^0_{ij}/T(t)} \le P^t_{ij} \le c_2 e^{-D^0_{ij}/T(t)}, \tag{4.115}$$

where c_1 and c_2 are positive constants. Assume for some integer $d \geq 0$ that (4.116) and (4.117) hold.

$$\sum_{t=0}^{\infty} e^{-d/T(t)} = \infty \tag{4.116}$$

$$\sum_{t=0}^{\infty} e^{-d-1/T(t)} < \infty \tag{4.117}$$

Theorem 16 shows that (4.118) and (4.119) are true.

$$\forall i \in S, \ \lim_{t \to \infty} P[X(t) \in R^d | X(0) = i] = 1 \tag{4.118}$$

$$\forall i \in R^d, \ \limsup_{t \to \infty} P[X(t) = i | X(0) = i] > 0 \tag{4.119}$$

Let $c_1 = e^{-b_1}$ and $c_2 = e^{-b_2}$. These values satisfy (4.114) and (4.115).

Let d satisfy $R^d \subset S_0$. Such a d must exist, since (4.3)–(4.7) are satisfied. Let $T(t) = d/\log t$. This satisfies (4.116) and (4.117). By (4.118) the inaccurate simulated annealing algorithm converges in probability to the set of global minima.

■

This result shows that if errors are constrained above and below by constant factors of the temperature, annealing under a $c/\log t$ temperature schedule will converge to an optimum. This result applies to any state space satisfying (4.3)–(4.7).

The convergence time makes this schedule uninteresting for practical matters. Its primary advantage, and the reason it appears in so many simulated annealing papers, is that it applies generally and yields theoretical results.

4.6 Deterministic Fractal Spaces

Several researchers have postulated that many common annealing spaces have a self-affine (fractal) structure [Sor92, MS89, KT85]. This characterization appeals because a geometric temperature schedule, ubiquitous in annealing programs, provably drives annealing to the optimum in a fractal space. Statistical comparisons suggest that the more closely an optimization space approximates a fractal structure, the more likely simulated annealing will find a good solution [Sor92]. This is also the annealing schedule for a physical system with constant specific heat, as described in equation (1.3).

In this section, I analyze the performance of inaccurate simulated annealing on a deterministic fractal space. Following Sorkin's work, suppose you have integer base $b > 0$, scaling factor $r \in (0, 1)$, and arbitrary, fixed function $F\colon \{0, \ldots, b-1\} \to [0, 1]$. The *a-complement function* of x, $\mathrm{comp}_a(x)$, is defined below.

$$\mathrm{comp}_a(x) = \begin{cases} x & \text{if } a \text{ is even,} \\ 1 - x & \text{otherwise.} \end{cases} \tag{4.120}$$

Construct a cost-function $C\colon [0, 1] \to [0, 1/(1 - r)]$ as follows. Represent state s in base b as $0.s_1 s_2 \cdots$. If state s has two representations, such as $0.0011111\cdots$ and 0.01 for $b = 2$, use the terminating one. Then

$$C(s) = F(s_1) + r \cdot C(\mathrm{comp}_{s_1}(0.s_2 s_3 \cdots)). \tag{4.121}$$

Thus, you have $C\colon [0, 1] \to [0, 1/(1 - r)]$.

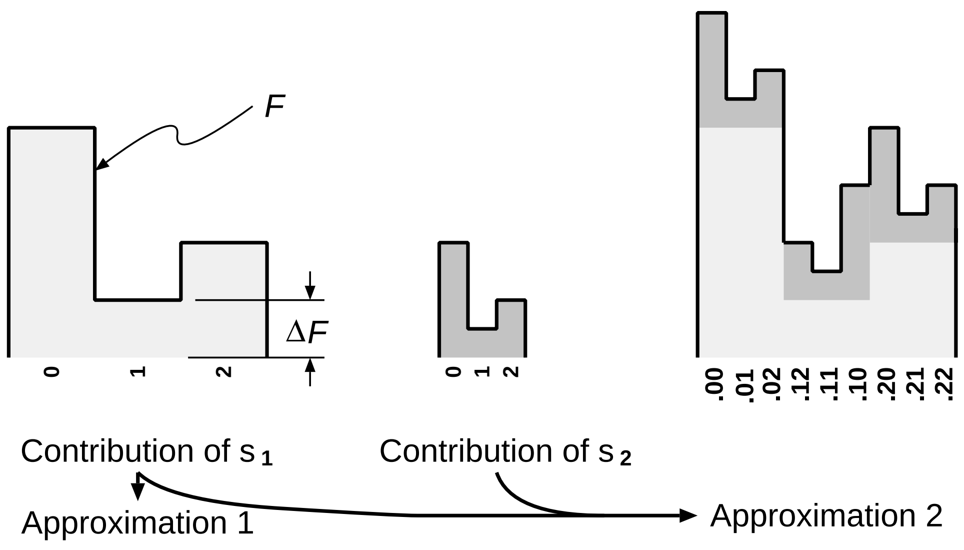


Figure 4.9: Base 3 Fractal Example

Spaces of this form exhibit a deterministic fractal structure. Figure 4.9 shows a simple example with base $b = 3$ and scaling factor $r = 0.5$. The contribution of first digit s_1 is added to the contribution of s_2 to form an approximation for the cost of s. If you compute the approximation of $C(s)$ using digits s_1 through s_k (i.e., presuming $C(\mathrm{comp}_{s_k}(0.s_{k+1}\cdots)) = 0$), the value you obtain is off by no more than

$$\sum_{i=k+1}^{\infty} r^i = \frac{r^k + 1}{1 - r}. \qquad (4.122)$$

(4.121) constructs the state space to anneal, but I have not presented a generate function. This analysis considers two generate functions, for confined and unconfined annealing. Confined annealing is unrealistically simple, but results for confined annealing will apply to unconfined annealing.

Suppose that you can perturb a single digit s_k of state s, by adding or subtracting b^{-k-1}, to generate a new state s'. If the perturbation would cause a carry or borrow into digit s_{k-1}, then move back to the original digit. Formally, the move set for state s at stage k is

$$S_k^{\mathbf{c}}(s) = \begin{cases} \{s - b^{-k-1}, s + b^{-k-1}\} & \text{if } 0 < s_k < b - 1, \\ \{s, s + b^{-k-1}\} & \text{if } s_k = 0, \\ \{s - b^{-k-1}, s\} & \text{if } s_k = b - 1. \end{cases} \tag{4.123}$$

The corresponding generate function is

$$\texttt{generate}_k^{\mathbf{c}}(s) \equiv \text{Choose randomly from } S_k^{\mathbf{c}} \tag{4.124}$$

Using this generate function produces what Sorkin calls confined annealing, so named because moves at stage k are confined by the first $k-1$ digits in the state.

By contrast, unconfined annealing has a less restricted move set,

$$S_k^{\mathbf{u}}(s) = \begin{cases} \{s - b^{-k-1}, s + b^{-k-1}\} & \text{if } b^{-k-1} \leq s \leq 1 - b^{-k-1}, \\ \{s, s + b^{-k-1}\} & \text{if } s < b^{-k-1}, \\ \{s - b^{-k-1}, s\} & \text{if } s > 1 - b^{-k-1}. \end{cases} \tag{4.125}$$

It has the obvious generate function $\texttt{generate}_k^{\mathbf{u}}$. An example showing the differences between confined and unconfined annealing appears in Figure 4.10.

This algorithm changes its generate function, starting with $\texttt{generate}_0$ and incrementing the subscript with each change in temperature. This feature is not exotic: tying the move scale to temperature has been common since the earliest work on simulated annealing [Whi84].

4.6.1 Confined Annealing

First, presume each state exhibits a fixed error. That is, that there is a single-valued function $\varepsilon\colon S \to \mathbb{R}$. Further presume that the accurate state space has a fractal structure. You can transform it into a new problem of the same class.

Definition 18 *The total variational distance between two probability vectors $\boldsymbol{\rho}$ and $\boldsymbol{\rho}'$ is*

$$\|\boldsymbol{\rho}' - \boldsymbol{\rho}\|_{\mathrm{tvd}} = \frac{1}{2}\sum_{v\in V} |\boldsymbol{\rho}'_v - \boldsymbol{\rho}_v| \tag{4.126}$$

or equivalently,

$$\|\boldsymbol{\rho}' - \boldsymbol{\rho}\|_{\mathrm{tvd}} = \sum_{v\in V:\boldsymbol{\rho}'_v>\boldsymbol{\rho}_v} |\boldsymbol{\rho}'_v - \boldsymbol{\rho}_v| \tag{4.127}$$

Lemma 19 *If $\boldsymbol{\rho}$, $\boldsymbol{\rho}'$, and $\boldsymbol{\rho}''$ are probability vectors on the same space, then*

$$\|\boldsymbol{\rho} - \boldsymbol{\rho}''\|_{\mathrm{tvd}} \le \|\boldsymbol{\rho} - \boldsymbol{\rho}'\|_{\mathrm{tvd}} + \|\boldsymbol{\rho}' - \boldsymbol{\rho}''\|_{\mathrm{tvd}} \tag{4.128}$$

Proof. Apply the definition of $\|_{\mathrm{tvd}}$ and the triangle inequality for absolute value.

■

Definition 20 *The minimum non-zero value of function $F\colon V \to \mathbb{R}$ is defined by*

$$\Delta F = \min_{v\in V}\{F(v), F(v) \neq 0\} \tag{4.129}$$

Lemma 21 *Let $\gamma = (\overline{\epsilon} - \underline{\epsilon})$. If $\gamma \le -T\ln(1-\varepsilon)$, then $\|\boldsymbol{\pi} - \dot{\boldsymbol{\pi}}\|_{\mathrm{tvd}} \le \varepsilon$.*

Proof. Let $a_v = \boldsymbol{\pi}_v/\dot{\boldsymbol{\pi}}_v$. By the definition of $\boldsymbol{\pi}$,

$$\|\boldsymbol{\pi} - \dot{\boldsymbol{\pi}}\|_{\mathrm{tvd}} = \frac{1}{2}\sum_{v\in V} |\boldsymbol{\pi}_v - \dot{\boldsymbol{\pi}}_v| \tag{4.130}$$

$$= \frac{1}{2}\sum_{v\in V}|\boldsymbol{\pi}_v - a_v\boldsymbol{\pi}_v| \tag{4.131}$$

$$= \frac{1}{2}\sum_{v\in V}|(1-a_v)\boldsymbol{\pi}_v| \tag{4.132}$$

By Theorem 1, you have $e^{-\gamma/T} \leq a_v \leq e^{\gamma/T}$. To maximize the effect of a_v, reduce some $\boldsymbol{\pi}_v = 1$ by a factor of $e^{-\gamma/T}$ and raise some other $\boldsymbol{\pi}_v = 0$ correspondingly by a factor of $e^{\gamma/T}$. So

$$\|\boldsymbol{\pi} - \dot{\boldsymbol{\pi}}\|_{\text{tvd}} \leq \frac{1}{2}\cdot 2\cdot(1-e^{-\gamma/T}) \tag{4.133}$$

By the premise obtain

$$1 - e^{-\gamma/T} \leq \varepsilon \tag{4.134}$$

(4.134), (4.130), and transitivity of $\leq$ prove the result.

■

Here I state several results from [Sor92] without proof.:

Lemma 22 (Sorkin 3.8.1) *Let $f\colon V \to [f_{\min}, f_{\max}]$, let $f_{\text{range}} = f_{\max} - f_{\min}$, let $\boldsymbol{\rho}$ be an arbitrary probability distribution on V, and let $\boldsymbol{\pi}(T)$ and $\boldsymbol{\pi}(0)$ be the stationary distributions at temperatures T and 0, then*

$$\mathrm{E}[f[\boldsymbol{\rho}]] \leq \mathrm{E}[f[\boldsymbol{\pi}(T)]] + \|\boldsymbol{\rho} - \boldsymbol{\pi}(T)\|_{\text{tvd}}\cdot f_{\text{range}} \tag{4.135}$$

Definition 23 *For a graph G with b vertices let $\hat{T}(\varepsilon,\Delta,b) = \Delta/\ln(b^2/\varepsilon)$. If G is regular, substitute b for b^2.*

Lemma 24 (Sorkin 3.8.4) *Given Δ, let $T \leq \hat{T}(\varepsilon,\Delta,b)$. If $\Delta \leq \varepsilon$ or $\Delta \leq \min\{F(v)|F(v) > \varepsilon\}$, then*

$$\mathrm{E}[F[\boldsymbol{\pi}(T)]] \leq 2\varepsilon \tag{4.136}$$

and if $\Delta \leq \Delta F$ as defined in Definition 20, then

$$\mathrm{E}[F[\boldsymbol{\pi}(T)]] \;\leq\; \|\boldsymbol{\pi}(T) - \boldsymbol{\pi}(0)\|_{\mathrm{tvd}} \;\leq\; \varepsilon. \tag{4.137}$$

Definition 25 *For an annealing graph G with b vertices and for a given temperature T, let*

$$\hat{t}(T, \varepsilon, b) = 2\left(\ln\left(\frac{b^2}{\varepsilon}\right) + \frac{1}{T}\right)\frac{1}{\Phi^2(\infty)}e^{2/T} \tag{4.138}$$

If G is regular, substitute b for b^2. Also, if G is regular you may substitute $1/b$ for $\Phi(\infty)$.

Lemma 26 (Sorkin 3.8.6) *If $t \geq \hat{t}(T, \varepsilon, n)$ then beginning from any distribution $\boldsymbol{\rho}^{(0)}$ and annealing at temperature T for time t, the outcome distribution $\boldsymbol{\rho}^{(t)}$ satisfies*

$$\|\boldsymbol{\rho}^{(t)} - \boldsymbol{\pi}(T)\|_{\mathrm{tvd}} \leq \varepsilon \tag{4.139}$$

You will need to fix the temperature until the total variational distance from the present distribution to the stationary distribution is less than some distance $\varepsilon \in [0, 1]$.

Theorem 27 (Sorkin 3.8.7) *Consider a graph G with b vertices, a cost-function F from G to $\mathbb{R}$ having minimum 0 and maximum 1, and a small value ε. Let Δ be any of ΔF, ε, or $\min\{F(v) \colon F(v) > \varepsilon\}$. With the functions $\hat{T}$ and $\hat{t}$ as defined above, begin from an arbitrary initial distribution $\boldsymbol{\rho}^{(0)}$ and anneal at temperature $T \leq \hat{T}(\varepsilon, \Delta, b)$ for $t \geq \hat{t}(T, \varepsilon, b)$ steps. If the outcome distribution is denoted $\boldsymbol{\rho}^{(t)}$,* *you have:*

$$\mathrm{E}[F[\boldsymbol{\pi}(T)]] \leq 2\varepsilon \tag{4.140}$$

$$\|\boldsymbol{\rho}^{(t)} - \boldsymbol{\pi}(T)\|_{\mathrm{tvd}} \leq \varepsilon \tag{4.141}$$

$$\mathrm{E}[F[\boldsymbol{\rho}^{(t)}]] \leq 3\varepsilon \tag{4.142}$$

If $\Delta = \Delta F$*, the bounds in (4.140) and (4.142) can be improved to* ε *and* 2ε *respectively.*

Lemma 28 *Define*

$$T = \hat{T}_\gamma(\varepsilon, \varepsilon, b) = \hat{T}(\varepsilon, \varepsilon, b), \tag{4.143}$$

per Definition 23, and

$$t = \hat{t}_\gamma(T, \varepsilon, b) = \hat{t}(T, \varepsilon, b) \tag{4.144}$$

per Definition 25. Apply simulated annealing with any starting distribution $\boldsymbol{\rho}^{(0)}$ *for time* t *and temperature* T*. Let* $\boldsymbol{\rho}^{(t)}$ *denote the outcome probability distribution using the accurate cost function, and let* $\dot{\boldsymbol{\rho}}^{(t)}$ *denote the outcome probability distribution under the inaccurate cost function. Then*

$$\|\dot{\boldsymbol{\rho}}^{(t)} - \dot{\boldsymbol{\pi}}(T)\|_{\mathrm{tvd}} \leq \varepsilon, \tag{4.145}$$

$$\mathrm{E}[F[\dot{\boldsymbol{\pi}}(T)]] \leq 3\varepsilon, \quad \text{and} \tag{4.146}$$

$$\mathrm{E}[F[\dot{\boldsymbol{\rho}}^{(t)}]] \leq 4\varepsilon. \tag{4.147}$$

Proof. Let $\Delta = \varepsilon$ in Lemmas 24 and 26. Then (note the absence of the dot here).

$$\mathrm{E}[F[\boldsymbol{\pi}(T)]] \leq 2\varepsilon \tag{4.148}$$

and, the first result,

$$\|\dot{\boldsymbol{\rho}}^{(t)} - \dot{\boldsymbol{\pi}}(T)\|_{\mathrm{tvd}} \leq \varepsilon. \tag{4.149}$$

By Lemma 21,

$$\|\boldsymbol{\pi}(T) - \dot{\boldsymbol{\pi}}(T)\|_{\mathrm{tvd}} \leq \varepsilon \tag{4.150}$$

Combining these using Lemma 19, obtain

$$\|\boldsymbol{\pi}(T) - \dot{\boldsymbol{\rho}}^{(t)}\|_{\mathrm{tvd}} \leq \|\boldsymbol{\pi}(T) - \dot{\boldsymbol{\pi}}(T)\|_{\mathrm{tvd}} - \|\dot{\boldsymbol{\pi}}(T) - \boldsymbol{\pi}(T)\|_{\mathrm{tvd}} \leq 2\varepsilon \tag{4.151}$$

Using Lemma 22, obtain the second and third results,

$$\mathrm{E}[F[\dot{\boldsymbol{\pi}}(T)]] \leq \mathrm{E}[F[\boldsymbol{\pi}(T)]] + \|\boldsymbol{\pi}(T) - \dot{\boldsymbol{\pi}}(T)\|_{\mathrm{tvd}}, \qquad \text{and} \tag{4.152}$$

$$\mathrm{E}[F[\dot{\boldsymbol{\rho}}^{(t)}]] \leq \mathrm{E}[F[\boldsymbol{\pi}(T)]] + \|\boldsymbol{\pi}(T) - \dot{\boldsymbol{\rho}}^{(t)}\|_{\mathrm{tvd}} \leq 4\varepsilon \tag{4.153}$$

■

Definition 29 *Construct a fractal-error cost-function with parameter $\gamma = (\overline{\epsilon} - \underline{\epsilon})$ as follows: Let fixed function F be given for the accurate space. Define $\dot{F}$ as any function such that*

$$F + \underline{\epsilon} \leq \dot{F} \leq F + \overline{\epsilon}. \tag{4.154}$$

Then construct the fractal-error cost-function $\dot{C}$ substituting $\dot{F}$ for F in (4.121).

Theorem 30 *Let deterministic fractal C with base b and scale factor r be given. Let quality factor ε satisfy $0 < \varepsilon < 1$. Compute $\hat{T} = \hat{T}(\varepsilon, \Delta, b)$ and $\hat{t} = \hat{t}(\hat{T}, \Delta, b)$. Apply confined annealing using fractal-error cost-function $\dot{C}$ with cooling schedule $(T_k, t_k) = (r^{k-1}\hat{T}, \hat{t})$ for $k = 1, \cdots, K$ and $K = \lceil \ln \varepsilon / \ln r \rceil$. Limit the errors in each stage by $\gamma_k \leq -T_k \ln(1-\varepsilon)$. Then the outcome has expected relative cost (or "quality"), $\dot{q}_{\text{con}}$, evaluated under the accurate cost-function,*

$$\dot{q}_{\text{con}} = \frac{\mathrm{E}[C[\dot{\boldsymbol{\rho}}^{(\hat{t}K)}]]}{C_{\text{range}}} \leq 5\varepsilon \tag{4.155}$$

and the algorithm consumes run time

$$\dot{t}_{\text{con}} = \lceil \ln \varepsilon / \ln r \rceil \cdot \hat{t} = \hat{t} \cdot K. \tag{4.156}$$

Proof. By Lemma 28, annealing with $(T_1, t_1) = (\hat{T}, \hat{t})$ in generation 1 gives $\mathrm{E}[F[\dot{\boldsymbol{\rho}}^{(\hat{t})}]] \leq 4\varepsilon$. Using the similarity of S_k to S_1, annealing in generation k with

$$(T_k, t_1) = (r_{k-1}\hat{T}, \hat{t}) \tag{4.157}$$

results in a distribution for s_k, $\dot{\boldsymbol{\rho}}^{(\hat{t})}$, satisfying

$$\mathrm{E}[F[\dot{\boldsymbol{\rho}}^{(\hat{t})}]] \leq 4\varepsilon \tag{4.158}$$

Therefore, the expected cost of the outcome satisfies

$$\begin{aligned} \mathrm{E}[C[\dot{\boldsymbol{\rho}}^{(\hat{t}K)}]] &\leq \sum_{k=1}^{K} r^{k-1}\mathrm{E}[F[\dot{\boldsymbol{\rho}}^{(\hat{t})}]] + \sum_{k=K+1}^{\infty} r^{k-1} \cdot 1 && (4.159) \\ &= \frac{1}{1-r}(4\varepsilon + r^K) && (4.160) \end{aligned}$$

Since C ranges from 0 to $\sum_{k=0}^{\infty} r^k = 1/(1-r)$, the relative expected cost is

$$\dot{q}_{\text{con}} = \frac{\mathrm{E}[C[\dot{\boldsymbol{\rho}}^{(\hat{t}K)}]]}{C_{\text{range}}} \leq 4\varepsilon + r^K. \tag{4.161}$$

If $K = \lceil \ln \varepsilon / \ln r \rceil$, then $r^K \leq \varepsilon$. This yields the result.

■

The previous result applies to a state space where both the accurate cost-function and the errors are self-similar. This may have some basis in realistic annealing problems, where reducing the size of the moves reduces the errors.

If the errors are not naturally self-similar, then you can impose artificial adjustments to make them so.

Corollary 31 *Define $\dot{C}$ so that $C(s) + \underline{\epsilon} \leq \dot{C}(s) \leq C(s) + \overline{\epsilon}$. If the errors are limited by $\gamma_k \leq -(T_k^2/\hat{T}) \ln(1 - \varepsilon)$, then the outcome has expected relative cost, $\dot{q}_{\text{con}}$, under the accurate cost-function,*

$$\dot{q}_{\text{con}} = \frac{\mathrm{E}[C[\dot{\boldsymbol{\rho}}^{(\hat{t}K)}]]}{C_{\text{range}}} \leq 5\varepsilon \tag{4.162}$$

and the algorithm consumes run time

$$\dot{t}_{\text{con}} = \lceil \ln \varepsilon / \ln r \rceil \cdot \hat{t}(\varepsilon). \tag{4.163}$$

4.6.2 Unconfined Annealing

This section will show that annealing on an unconfined fractal space has the same outcome distribution as the confined fractal space, appealing to the Sorkin replica model.

The results for confined annealing encourage, but do not satisfy. In real annealing applications, a program cannot constrain moves to a given state segment. If it could, other algorithms would do a better job.

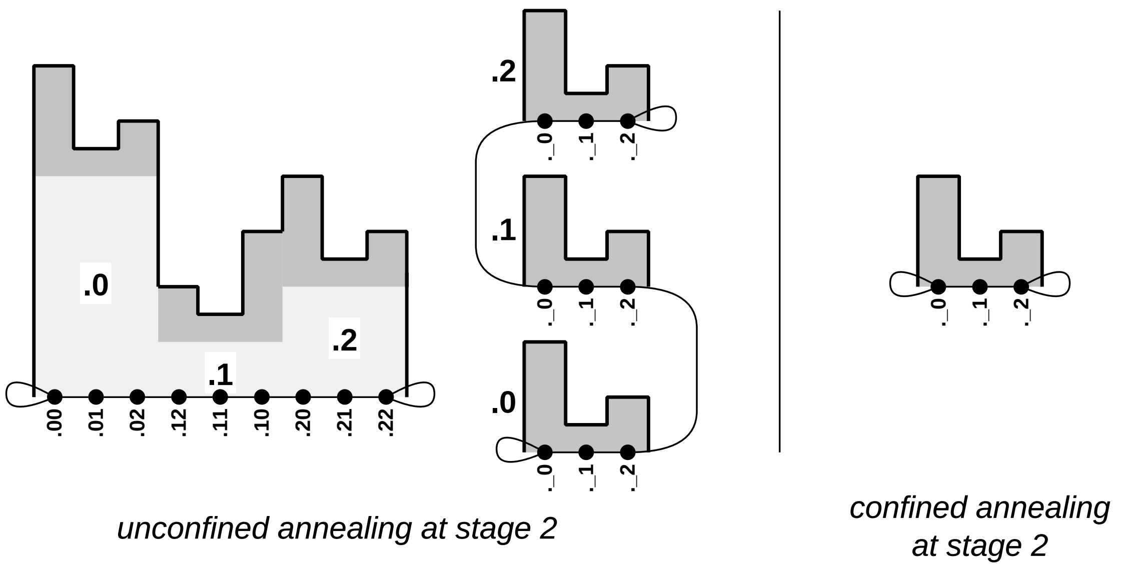

.0
.1
.2
.00
.01
.02
.12
.11
.10
.20
.21
.22
.2
.1
.0
._0
._1
._2
unconfined annealing at stage 2
confined annealing at stage 2

Figure 4.10: Unconfined annealing vs. Confined annealing

As it turns out, allowing the state to stray into an adjacent segment will not affect the result. The algorithm resolves the kth segment to within 5ε of the desired outcome in generation k, as in confined annealing. In subsequent generations, annealing continues on the kth segment with gradually decreasing temperature. The additional processing can only improve the kth segment, and so the worst-case bounds for confined annealing still hold.

The proof is virtually identical to Sorkin's. I refer the reader to §8.4 of [Sor92], and merely state the result:

Theorem 32 *Let value $0 < \varepsilon < 1$ satisfying $\hat{T} \leq T_{\text{crit}}$, and deterministic fractal C with scale factor r be given. Let $\gamma \leq -\ln(1-\varepsilon)$. Compute $\hat{T}$ and $\hat{t}$ as before. Apply unconfined annealing with cooling schedule $(T_k, t_k) = (r^{k-1}\hat{T}, \hat{t})$ for $k = 1, \cdots, K$ and $K = \lceil \ln \varepsilon / \ln r \rceil$. Then the outcome has relative expected cost, $\dot{q}_{\text{uncon}}$, evaluated* *under the accurate cost-function, of*

$$\dot{q}_{\text{uncon}} = \frac{\mathrm{E}[C[\dot{\boldsymbol{\rho}}^{(\hat{t}K)}]]}{C_{\text{range}}} \leq 5\varepsilon \tag{4.164}$$

and the algorithm consumes run time

$$\dot{t}_{\text{uncon}} = \lceil \ln \varepsilon / \ln r \rceil \cdot \hat{t}. \tag{4.165}$$

Observe the proof outline from Figure 4.10. When performing unconfined annealing, transitions may occur from one large-scale feature to another (as from .21 to .11), even during small-scale moves, but these transitions do not affect the annealing process on the smaller scale—they behave identically to the end-loops on confined annealing. Therefore, unconfined annealing can be shown equivalent to confined annealing.

4.6.3 Inaccurate Fractal: Equivalent Quality

I have shown that constraining errors by $\gamma_k \leq -T_k \ln(1-\varepsilon)$ allows an inaccurate fractal system to converge. However, I have produced an outcome inferior to the accurate system: the inaccurate system reaches relative expected cost $\dot{q}_{\text{uncon}} \leq 5\varepsilon$, while the accurate system reaches $q_{\text{uncon}} \leq 3\varepsilon$ [Sor92].

A more useful result determines the number of additional annealing steps needed to bring the inaccurate system to the same quality as the accurate system. I provide specifics below.

Theorem 33 *If t is the total time required to reach $q_{\text{uncon}} \leq 3\varepsilon$ with accurate* *annealing, and t' is the total time to reach $\dot{q}_{\text{uncon}} \leq 3\varepsilon$ under the inaccurate cost-function, then t and t' are related by the approximation*

$$\frac{t'}{t} \approx \left(\frac{5}{3}\right)^{2/\Delta} \tag{4.166}$$

Proof. Let $\varepsilon' = 3\varepsilon/5$. To obtain $\dot{q}_{\text{uncon}} \leq 3\varepsilon$ under inaccurate annealing, anneal for

$$t' = \lceil \ln \varepsilon' / \ln r \rceil \cdot \hat{t} \tag{4.167}$$

steps.

$$\begin{aligned} t' &= \lceil \ln \frac{3\varepsilon}{5} / \ln r \rceil \cdot 2 \left(\ln \frac{5b^2}{3\varepsilon} + \frac{1}{\Delta F / \ln \frac{5b^2}{3\varepsilon}} \right) e^{2 \ln(5b^2/3\varepsilon)/\Delta} && (4.168) \\ &= \lceil \ln \frac{3\varepsilon}{5} / \ln r \rceil \cdot 2 \ln \frac{5b^2}{3\varepsilon} \left(1 + \frac{1}{\Delta F}\right) \left(5b^2/3\varepsilon\right)^{2/\Delta}. && (4.169) \end{aligned}$$

Now, using algebra,

$$\begin{aligned} \frac{t'}{t} &= \frac{\lceil \ln \frac{3\varepsilon}{5} / \ln r \rceil \cdot 2 \ln \frac{5b^2}{3\varepsilon} \left(1 + \frac{1}{\Delta F}\right) (5b^2/3\varepsilon)^{2/\Delta}}{\lceil \ln \varepsilon / \ln r \rceil \cdot 2 \ln \frac{b^2}{\varepsilon} \left(1 + \frac{1}{\Delta F}\right) (b^2/\varepsilon)^{2/\Delta}} && (4.170) \\ &\approx \left\lceil \frac{\ln \varepsilon + \ln \frac{3}{5}}{\ln \varepsilon} \right\rceil \frac{\ln \frac{b^2}{\varepsilon} + \ln \frac{5}{3}}{\ln \frac{b^2}{\varepsilon}} \left(\frac{5}{3}\right)^{2/\Delta}. && (4.171) \end{aligned}$$

Suppose $5/3 \ll b^2/\varepsilon$ and $0 < \ln \epsilon \ll 3/5$, which is likely for a high-quality outcome (otherwise, why use simulated annealing). Then you can further approximate (4.171) as

$$\frac{t'}{t} \approx \left(\frac{5}{3}\right)^{2/\Delta}. \tag{4.172}$$

■

Note that (4.172) has $(\overline{\epsilon} - \underline{\epsilon}) \leq -T \ln(1 - \varepsilon)$ as a premise. Tightening error bounds would bring t'/t closer to one. Likewise, loosening error bounds would increase the ratio.

4.7 Measuring Errors

I have shown how errors and the temperature schedule interact to affect the outcome and speed of simulated annealing. It is rarely practical to analytically determine the error-range, instead you measure errors and estimate the error-range.

Annealing programs may measure errors in one of two ways: First, by subtracting the true-cost from the computed-cost at each iteration. Second, by performing a series of iterations, accumulating a total observed cost-difference, and subtracting the total true cost-difference.

The first, called "instantaneous error," provides more detailed information about each error. The second, called "accumulated error," is more efficient, since the program computes the true-cost once per sequence, rather than once per iteration. Several researchers have performed accumulated error measurements [GD89, BJS90, DKN87, JD88, KCP94]. I know of no published instantaneous error measurements.

Figure 4.11 shows a four iteration annealing sequence performed by two processors. The first column shows the true-state of the system. The second column shows processor 1's moves, with local copies of processor 2 variables shaded. The third column shows processor 2's moves, with local copies of processor 1 variables shaded. The fourth column shows each instantaneous error, in graphical form. The fifth column shows accumulated errors.

The accumulated error column in Figure 4.11 does not account for the "P2

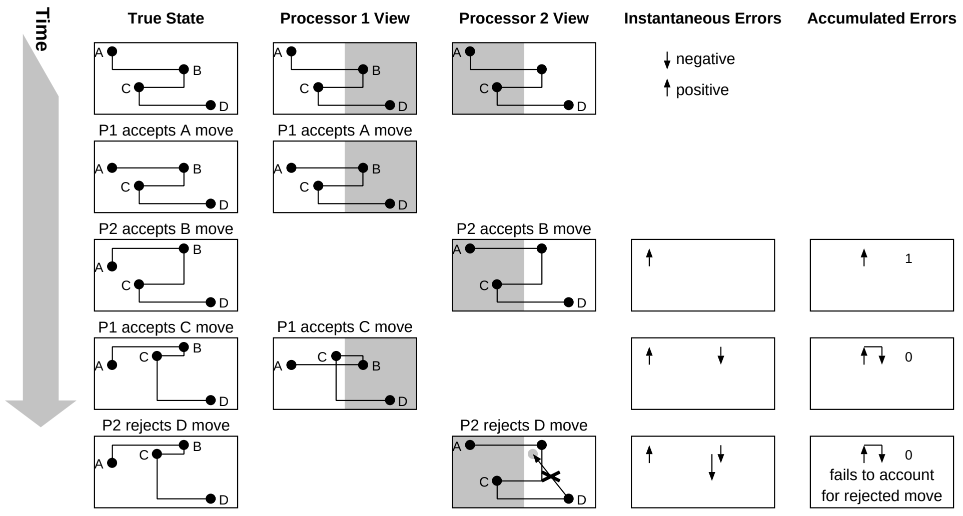


Figure 4.11: Observed Errors

rejects D move." In fact, accumulated errors never include contributions from rejected moves. Although there are two instantaneous errors in the figure's accepted moves, the final accumulated error is zero—the instantaneous errors cancelled each other out.

To measure instantaneous errors you must compute both true and observed-costs of each move, annihilating the speed advantage of inaccurate cost-functions. You might compute instantaneous errors during a test run, to predict the behavior of a class of annealing problems, but not during every production run.

Measuring accumulated errors is more efficient, but it does not help estimate the instantaneous error-range. Dividing a series' accumulated error by its iterations does not produce average instantaneous error—the accumulated error does not include contributions from rejected moves. The average instantaneous error isn't useful anyway, the theory requires a maximum and minimum for the range.

Though accumulated error measurements do not help estimate instantaneous errors, accumulated error-bounds work directly in the results I established. Why? The composed moves in a sequence can be seen as a "big move," and the composed accept/reject decisions as a "big decision." If you performed the "big" annealing algorithm the result would be the same Boltzmann distribution as that of the "little" original. The following results formalize this idea.

Lemma 34 *Let $X = (S, G, C, T)$ be an annealing chain, with state-set S, generate probability matrix G, cost-function C, and fixed temperature T, which satisfies (4.3)–(4.7). Choose a sequence-length n, and assume $G' = \prod_{i=1}^{n} G$ has the coverage property (4.4). Construct another annealing chain $X' = (S, G', C, T)$. Then, X and X' have the same equilibrium probability density.*

PROOF. I first prove that X' satisfies the annealing properties (4.3)–(4.7).

Probability (4.3): G is a transition probability matrix. The composition of transition probability matrices is a transition probability matrix, so G' is a transition probability matrix.

Coverage (4.4): By definition of G'.

Aperiodicity (4.5): If $\exists s, s' \in S, C(s) \neq C(s')$, then (4.2) ensures aperiodicity. Otherwise, all costs are equal, and the acceptance matrix defined by (4.1) must contain all 1s. By (4.2), the Markov chains $P' = G'$ and $P = G$. Therefore, G must itself be aperiodic. The composition of aperiodic matrices is aperiodic, so G' is aperiodic. Therefore, P' is aperiodic.

Finiteness (4.6): By definition of X'.

Symmetry (4.7): The composition of symmetric matrices is symmetric, therefore G' is symmetric.

The annealing constraints are satisfied. Therefore, the equilibrium probability density of chain X' is the Boltzmann distribution (4.9).

Since X and X' have the same cost-function, their distributions are equal.

■

Let the move-sequence $\hat{s} = \langle s_0, \ldots, s_n \rangle$ designate n successively accepted moves. Let the observed cost-difference from state s to s' be denoted $\dot{\delta}(s, s')$ The total observed cost-difference is

$$\dot{\Delta} = \sum_{i=0}^{n-1} \dot{\delta}(s_i, s_{i+1}) \tag{4.173}$$

Knowing the true-cost at the beginning and end of the move-sequence, $C(s_0)$ and $C(s_n)$, you can compute the total true cost-difference as

$$\Delta = C(s_n) - C(s_0). \tag{4.174}$$

The accumulated error for the sequence is

$$\epsilon_{\hat{s}} = \dot{\Delta} - \Delta = \sum_{i=0}^{n-1} \epsilon_i. \tag{4.175}$$

Theorem 35 *The results of §4.3.1 apply to accumulated errors: If the accumulated error is bounded above by $\overline{\boldsymbol{\epsilon}}$ and below by $\underline{\boldsymbol{\epsilon}}$, and $\boldsymbol{\gamma} = \overline{\boldsymbol{\epsilon}} - \underline{\boldsymbol{\epsilon}}$, then*

$$e^{-\boldsymbol{\gamma}/T} \boldsymbol{\pi}_s(T) \leq \dot{\boldsymbol{\pi}}_s(T) \leq e^{\boldsymbol{\gamma}/T} \boldsymbol{\pi}_s(T) \tag{4.176}$$

and

$$e^{-\boldsymbol{\gamma}/T}\bar{C}[\boldsymbol{\pi}(T)] \leq \bar{C}[\dot{\boldsymbol{\pi}}(T)] \leq e^{\boldsymbol{\gamma}/T}\bar{C}[\boldsymbol{\pi}(T)] \tag{4.177}$$

Proof. All moves which can generate instantaneous errors have a non-zero probability of being accepted. Therefore, any sequence of moves $\epsilon_{\hat{s}}$ has a non-zero probability $\prod_{i=0}^{n-1} P_{s_i,s_{i+1}}$ of being accepted in its entirety. Its accumulated error, $\epsilon_{\hat{s}}$, contains instantaneous error contributions from each move.

Lemma 34 shows that the equilibrium properties of annealing with "big moves," created by composing `generate` n times, are equal to those obtained by annealing with "little moves", using `generate` once.

Suppose the instantaneous error of the "big move" cost-function was bounded above by $\overline{\epsilon}$ and below by $\underline{\epsilon}$. Of course, you could then use the results of §4.3.1 to predict the equilibrium properties of the "big move" system.

Note that the instantaneous error of the "big move" cost-function is the accumulated error of the "little moves". Then (4.11) implies (4.176), and (4.17) implies (4.177).

■

The instantaneous and accumulated error upper bounds, $\overline{\epsilon}$ and $\overline{\boldsymbol{\epsilon}}$, are infrequently observed at low temperatures: when errors are above the mean, they decrease the probability of acceptance. Some annealing algorithms have errors symmetric about the mean, and there the mean error is zero. Parallel implementations often have these properties [DW90]. In this case, use the observed lower bound to estimate the upper bound.

4.8 Summary

Calculation errors in simulated annealing, arising from parallelism or estimated cost-functions, affect the expected cost, the algorithm's speed, and its outcome. Revisiting the taxonomy constructed in §2, the properties of $\boldsymbol{\mu}$ and Φ produce the same taxonomy, as shown in Table 4.1.

Category	$\dot{C}$	$\dot{G}$	$\dot{\boldsymbol{\mu}}$	$\dot{\Phi}$
Serial-Like	$= C$	$= G$	$= \boldsymbol{\mu}$	$= \Phi$
Altered Generation	$= C$	$\neq G$	$= \boldsymbol{\mu}$	$\neq \Phi$
Asynchronous	$\neq C$	?	$\neq \boldsymbol{\mu}$	$\neq \Phi$
Serial, Estimated Cost	$\neq C$	$= G$	$\neq \boldsymbol{\mu}$	$\neq \Phi$

Table 4.1: Comparison with Accurate-Cost Serial Annealing (Revisited)

I have addressed several open questions about annealing under inaccurate cost-functions.

When the cost-function has range-errors, the expected true-cost at equilibrium is bounded. The controlling value is $\gamma = \overline{\epsilon} - \underline{\epsilon}$. In this case, the expected true-cost, $\dot{C}$, ranges from $Ce^{-\gamma/T}$ to $Ce^{+\gamma/T}$, where C is the expected true-cost when running annealing with an accurate cost-function. Therefore, the expected true-cost diverges exponentially from that obtained with an accurate cost-function, as temperature decreases. By controlling the errors so that γ is proportional to T, the expected costs will differ no more than a constant factor.

When the errors exhibit a Gaussian distribution, the controlling value is $\overline{\sigma}^2 - \underline{\sigma}^2$.

In this case, the expected true-cost $\ddot{C}$, ranges from $Ce^{-(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}$ to $Ce^{+(\overline{\sigma}^2-\underline{\sigma}^2)/2T^2}$. Again, the expected true-cost diverges exponentially as temperature decreases. Likewise, if $\overline{\sigma}^2 - \underline{\sigma}^2$ is kept proportional to T^2 the equilibrium costs will differ by no more than a constant factor.

To characterize a system with varying temperature, examine the rate at which it approaches equilibrium at each temperature. One convenient measure for the rate of convergence is "conductance."

Like the expected true-cost, the conductance of the inaccurate system diverges exponentially from that of the accurate system, as temperature decreases. The conductance of an inaccurate system with fixed bounds varies from $e^{-3\gamma/T}\Phi$ to $e^{+3\gamma/T}\Phi$, where Φ is the conductance of an equivalent accurate system.

The conductance of a system with Gaussian errors, $\ddot{\Phi}$, is related to the accurate system conductance by $e^{(\underline{\sigma}^2-\overline{\sigma}^2)/T^2}\Phi/2 \leq \ddot{\Phi} \leq e^{(2\overline{\sigma}^2-\underline{\sigma}^2)/T^2}\Phi$. Again, the conductance diverges exponentially as temperature decreases.

By relating the expected cost and conductance of the inaccurate systems to those of similar accurate systems, I obtained useful results: error constraints and annealing-time adjustments which guarantee convergence to the same quality outcome as normal annealing.

The state spaces I considered are mathematical approximations to meaningful problems. They provide direction to be modified with experience. Based on this analytic work, simulated annealers should:

¶ Identify errors or approximations in the cost-function. For example, a common cost-function approximates the length of a wire network as half the bounding-box perimeter. This inaccurate cost-function appears in many VLSI circuit placement programs.

¶ Instrument inaccurate annealing programs to compute and display errors during a run. Note that it is acceptable to measure only accumulated errors, since §4.7 shows that they are equivalent to instantaneous errors. However, the accumulated errors will likely be larger than the instantaneous errors, and several instantaneous moves must be counted as one accumulated move. Both effects will extend the required annealing time.

¶ Introduce algorithms to keep errors within a constant factor of the temperature. Use §4.5 and §4.6 to help estimate the factor.

¶ Increase the number of annealing steps performed at each temperature to compensate for errors. Use §4.6.3 to help estimate the number of steps.

¶ Weigh the performance benefits of parallelism or approximate computations against the increased annealing steps to obtain the same quality. Inaccuracies can even degrade the total execution time, when considered in this light.

CHAPTER 5

Conclusion

Simulated annealing is an optimization algorithm related to a thermodynamic process called annealing. It is similar to the greedy algorithm, but generates cost-increasing moves to escape local minima based on temperature. If a cost-function and a move-generator exist, and appropriate constraints are satisfied, many combinatorial problems are susceptable to simulated annealing. One such problem is circuit placement.

Because the circuit placement problem has many variations, each suitable for different chip technologies or resource constraints, it has resisted special-purpose optimization algorithms. So, the general-purpose simulated annealing algorithm has remained a popular method for placing circuits.

Because annealing is thought to be slow, researchers have explored a variety of parallelizing techniques. Some techniques exploit annealing's claimed error tolerance to eliminate some inter-processor communication required to compute accurate cost-functions. This speeds up the cost-function, but introduces errors. Results have been mixed, but promising.

I wrote parallel circuit-placement annealing programs which measure cost-function errors and "circuit mobility." I then experimented on several small circuit placement problems. When errors increased, the outcome quality decreased. Circuit mobility per-move did not conclusively affect the outcome, while increased mobility per-trial at low temperatures seemed to improve the result.

Dissatisfied with existing claims about simulated annealing's error tolerance, I proved the analytic results of Chapter 4. Errors have long been known to appear in some parallel simulated annealing applications, and many experiments have been performed to understand their impact. However, I showed that errors also appear in common sequential annealing applications. These errors succumb to the same analysis.

I assumed that two forms of error appear in cost-functions: range-errors and Gaussian errors. Two important properties—equilibrium cost and conductance—drive the proofs. It turns out that conductance formally captures my intentions behind measuring "mobility" in the experiments of Chapter 3. I showed how equilibrium cost and conductance change with error constraints and temperature, when the temperature is fixed.

To understand how errors affect the final outcome, you must consider a non-constant temperature schedule. I first evaluated general combinatorial problems under an impractical schedule common in annealing proofs: $T(t) = c/\ln t$. This schedule completes in exponential time, but it applies broadly. I proved that keeping range-errors to a constant factor of temperature produces a high-quality outcome.

A commonly used temperature schedule is the geometric schedule, $T(t) =$

$c\,T(t-1)$, where $0 < c < 1$. In theory, this schedule is not broadly applicable; in practice, variations of it are used everywhere. Sorkin showed that self-affine functions (fractals) can be solved by annealing with geometric schedules. He performed measurements on typical annealing problems showing they compare favorably to fractals.

I showed that fractals containing range-errors can also be solved by geometric temperature schedules, if the range is confined to a constant function of the temperature. It is necessary to extend the temperature schedule to account for slower convergence to equilibrium (i.e., changes in conductance) and to account for the equilibrium cost differences between the inaccurate cost-function and the true-cost.

These results have practical bearing for simulated annealing researchers. Errors may occur in cost-functions even when no parallelism is involved. When errors occur, analyze the problem or measure errors at runtime to establish an error range. Using this range, and the results of the last chapter, modify the algorithm to keep errors within a constant factor of the temperature. Extend the temperature schedule to account for conductance changes and equilibrium cost differences. Using these techniques, researchers can improve the outcome and speed of simulated annealing whenever errors occur.

5.1 Future Research

This work leaves open several areas of research. First, better controlled experiments would bring more understanding to this field. Few published reports make fair comparisons between parallel and sequential annealing outcomes. When the outcome quality of parallel annealing is worse than sequential, speedup claims for parallel annealing aren't valid.

Second, dismal speedup results have continued to appear in publications on parallel simulated annealing. The problem sizes for experiments have been small, and experimenters have not compensated for increased errors and decreased conductance. These contribute to decreasing the outcome quality or increasing the time.

For example, parallel versions of row-based circuit placement have often interleaved the rows on different processors [KCP94, Kli87]. This is not surprising, because the data-structures lend themselves to assignment of a whole row to a processor, but it creates massive errors and small conductance, each contributing to a poor outcome. Attentiveness to these issues will make future experiments much more successful.

Third, measurements of various properties during annealing would help drive adaptive move and error-control schedules for simulated annealing. These should include errors and conductance. In fact, just finding a practical way of estimating conductance for various annealing problems would contribute to the field.

Fourth, adaptive error-control schedules would make both parallel and sequential annealing run better. An early attempt at an error-control schedule appears in [CRS87], but I have seen little since. Some parallel algorithms could be easily improved by using simple serializable set at low temperatures (see §2.1.2). I believe this area has many opportunities.

Fifth, during my work on Chapter 4, I sought to prove a very general statement about errors: that if an annealing space was ergodic, and if *any* particular temperature schedule was guaranteed to obtain an outcome of a particular quality on that space, then keeping the errors within a constant factor of temperature, using the same temperature schedule, would guarantee an outcome of some related quality. This proof eluded me, but I believe it likely.

Many simulated annealing cost-functions include errors, both in sequential and parallel implementations, but annealing remains a generally useful tool. Understanding the consequences of cost-function errors, both in outcome quality and execution time, helps us produce faster and better annealing algorithms.

Bibliography

[ABH86] Emile H.L. Aarts, Frans M.J. de Bont, Erik H.A. Habers, and Peter J.M. van Laarhoven. "A Parallel Statistical Cooling Algorithm." In *Proceedings of the Symposium on the Theoretical Aspects of Computer Science*, volume 210, pp. 87–97, January 1986.

[Abr89] D. Abramson. "Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms." Technical Report TR 112 069, Department of Communication and Electrical Engineering, Royal Melbourne Institute of Technology, Melbourne, Australia, January 1989.

[AC89] James R.A. Allwright and D.B. Carpenter. "A Distributed Implementation of Simulated Annealing for the Travelling Salesman Problem." *Parallel Computing*, **10**(3):335–338, May 1989.

[BAM88] F.M.J. de Bont, E.H.L. Aarts, P. Meehan, and C.G. O'Brien. "Placement of Shapeable Blocks." *Philips Journal of Research*, **43**(1):1–27, April 1988.

[Ban94] Prithviraj Banerjee. *Parallel Algorithms for VLSI Computer-Aided Design.* PTR Prentice-Hall, Englewood Cliffs, NJ, 1994.

[BB88] R. Brouwer and P. Banerjee. "A Parallel Simulated Annealing Algorithm for Channel Routing on a Hypercube Multiprocessor." In *Proceedings of the International Conference on Computer Design*, pp. 4–7, 1988.

[BB89] Henrik Bohr and Søren Brunak. "A Travelling Salesman Approach to Protein Conformation." *Complex Systems*, **3**:9–28, 1989.

[BG89a] Joseph Bannister and Mario Gerla. "Design of the Wavelength-Division Optical Network." Technical Report CSD-890022, UCLA Computer Science Department, May 1989.

[BG89b] Valmir C. Barbosa and Eli Gafni. "A Distributed Implementation of Simulated Annealing." *Journal of Parallel and Distributed Computing*, **6**:411–434, 1989.

[BJ86] Prithviraj Banerjee and Mark Jones. "A Parallel Simulated Annealing Algorithm for Standard Cell Placement on a Hypercube Computer."

In *Proceedings of the International Conference on Computer-Aided Design*, pp. 34–37, November 1986.

[BJS90] Prithviraj Banerjee, Mark Howard Jones, and Jeff S. Sargent. “Parallel Simulated Annealing Algorithms for Cell Placement on Hypercube Multiprocessors.” *IEEE Transactions on Parallel and Distributed Systems*, **1**(1):91–106, January 1990.

[BM89] N. Benvenuto and M. Marchesi. “Digital Filters Design by Simulated Annealing.” *IEEE Transactions on Circuits and Systems*, **36**(3):459–460, March 1989.

[BS87] Andrei Broder and Eli Shamir. “On the Second Eigenvalue of Random Regular Graphs.” In *Proceedings of the 28th IEEE Symposium on the Foundations of Computer Science*, pp. 286–294, 1987.

[BS89] Griff L. Bilbro and Wesley E. Snyder. “Range Image Restoration using Mean Field Annealing.” In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pp. 594–601. Morgan Kaufmann Publishers, San Mateo, 1989.

[Bur88] Edward M. Burgess. “Protein Folding by Simulated Annealing.” *Abstracts of Papers of the American Chemical Society*, **195**:41, June 1988.

[CD86] Pak K. Chan and Gary S. Ditlow. “Power/Timing Optimization with Long and Short-Path Constraints.” Technical Report RC 12126, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, September 1986.

[CEF88] Roger D. Chamberlain, Mark N. Edelman, Mark A. Franklin, and Ellen E. Witte. “Simulated Annealing on a Multiprocessor.” In *Proceedings of the International Conference on Computer Design*, pp. 540–544, 1988.

[CHM88] J.P. Cohoon, S.U. Hegde, W.N. Martin, and D. Richards. “Floorplan Design using Distributed Genetic Algorithms.” In *Proceedings of the International Conference on Computer-Aided Design*, 1988.

[CM69] D. Chazan and W. Miranker. “Chaotic Relaxation.” *Linear Algebra and Its Applications*, **2**(2):199–222, April 1969.

[CMV88] Francky Catthoor, Hugo de Man, and Joos Vandewalle. “SAMURAI: A General and Efficient Simulated-Annealing Schedule with Fully Adaptive Annealing Parameters.” *Integration, the VLSI Journal*, **6**:147–178, July 1988.

[CRS86] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli. "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells." In *Proceedings of the International Conference on Computer-Aided Design*, pp. 30–33. IEEE Computer Society Press, 1986.

[CRS87] Andrea Casotto, Fabio Romeo, and Alberto Sangiovanni-Vincentelli. "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells." *IEEE Transactions on Computer-Aided Design*, **CAD-6**(5):838–847, September 1987.

[Dav87] Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers, Los Altos, CA, 1987.

[DKN87] Frederica Darema, Scott Kirkpatrick, and Alan V. Norton. "Parallel Algorithms for Chip Placement by Simulated Annealing." *IBM Journal of Research and Development*, **31**(3):391–402, May 1987.

[DN86] Srinivas Devadas and A. Richard Newton. "Topological Optimization of Multiple Level Array Logic: On Uni and Multi-processors." In *Proceedings of the International Conference on Computer-Aided Design*, pp. 38–41, Santa Clara, CA, November 1986.

[Dod89] Nigel Dodd. "Graph Matching by Stochastic Optimization Applied to the Implementation of Multi Layer Perceptrons on Transputer Networks." *Parallel Computing*, **10**(2):135–142, April 1989.

[Doy88] John Doyle. "Classification by Ordering a (Sparse) Matrix: A "Simulated Annealing" Approach." *Applications of Mathematical Modeling*, **12**:86–94, February 1988.

[Dre88] A. Drexl. "A Simulated Annealing Approach to the Multiconstraint Zero-One Knapsack Problem." *Computing*, **40**:1–8, 1988.

[Dur89] M.D. Durand. "Cost Function Error in Asynchronous Parallel Simulated Annealing Algorithms." Technical Report CUCS-423-89, Columbia University Computer Science Department, New York, NY, June 1989.

[DW90] M.D. Durand and Steve R. White. "Permissible Error in Parallel Simulated Annealing." Technical Report RC 15487, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1990.

[Eng88] Jonathan Engel. "Teaching Feed-Forward Neural Networks by Simulated Annealing." *Complex Systems*, **2**:641–648, 1988.

[FKO85] Edward Felten, Scott Karlin, and Steve W. Otto. "The Traveling Salesman Problem on a Hypercubic, MIMD Computer." In *Proceedings of* *the 1985 International Conference on Parallel Processing*, pp. 6–10, St. Charles, Pennsylvania, 1985.

[FLW86] V. Faber, Olaf M. Lubeck, and Andrew B. White, Jr. "Superlinear Speedup of an Efficient Sequential Algorithm is Not Possible." *Parallel Computing*, **3**(3):259–260, July 1986.

[GD89] Daniel R. Greening and Frederica Darema. "Rectangular Spatial Decomposition Methods for Parallel Simulated Annealing." In *Proceedings of the International Conference on Supercomputing*, pp. 295–302, Crete, Greece, June 1989.

[GK85] Charles D. Gelatt, Jr. and Edward S. Kirkpatrick. "Optimization of an Organization of Many Discrete Elements." Patent Number 4,495,559, United States Patent Office, January 1985. (application filed November 2, 1981).

[GM88] D.J. Goodman and T.J. Moulsley. "Using Simulated Annealing to Design Digital Transmission Codes for Analogue Sources." *Electronics Letters*, **24**(10):617–619, May 1988.

[GM89] Saul B. Gelfand and Sanjoy K. Mitter. "Simulated Annealing with Noisy or Imprecise Energy Measurements." *Journal of Optimization Theory and Applications*, **62**(1):49–62, July 1989.

[GM91] Saul B. Gelfand and Sanjoy K. Mitter. "Simulated Annealing Type Algorithms for Multivariate Optimization." *Algorithmica*, **6**(3):419–436, 1991.

[GPC88] J.L. Gaudiot, J.I. Pi, and M.L. Campbell. "Program Graph Allocation in Distributed Multicomputers." *Parallel Computing*, **7**(2):227–247, June 1988.

[Gre90a] Daniel R. Greening. "Equilibrium Conditions of Asynchronous Parallel Simulated Annealing." In *Proceedings of the International Workshop on Layout Synthesis*, Research Triangle Park, North Carolina, 1990. (also as IBM Research Report RC 15708).

[Gre90b] Daniel R. Greening. "Review of The Annealing Algorithm." *ACM Computing Reviews*, **31**(6):296–298, June 1990.

[Gre91a] Daniel R. Greening. "Asynchronous Parallel Simulated Annealing." In *Lectures in Complex Systems, volume III.* Addison-Wesley, 1991.

[Gre91b] Daniel R. Greening. "Parallel Simulated Annealing Techniques." In Stephanie Forrest, editor, *Emergent Computation*, pp. 293–306. MIT Press, 1991.

[Gre91c] Daniel R. Greening. "Simulated Annealing with Inaccurate Cost Functions." In *Proceedings of the IMACS International Congress of Mathematics and Computer Science*, Trinity College, Dublin, 1991.

[Gro86] Lov K. Grover. "A New Simulated Annealing Algorithm for Standard Cell Placement." In *Proceedings of the International Conference on Computer-Aided Design*, pp. 378–380. IEEE Computer Society Press, November 1986.

[Gro89] Lov K. Grover. "Simulated Annealing Using Approximate Calculation." In *Progress in Computer Aided VLSI Design, volume 6*. Ablex Publishing Corp., 1989. (also as Bell Labs Technical Memorandum 52231-860410-01, 1986).

[GS87] David E. Goldberg and Philip Segrest. "Finite Markov Chain Analysis of Genetic Algorithms." In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pp. 1–8, 1987.

[Haj88] Bruce Hajek. "Cooling Schedules for Optimal Annealing." *Mathematics of Operations Research*, **13**(2):311–329, May 1988.

[HBO88] C.J. Hardy, P.A. Bottomley, M. O'Donnell, and P. Roemer. "Optimization of Two-Dimensional Spatially Selective NMR Pulses by Simulated Annealing." *Journal of Magnetic Resonance*, **77**(2):233–250, 1988.

[HS78] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, Rockville, Maryland, 1978.

[JAM89] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. "Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning." *Operations Research*, **37**(6):865–892, 1989.

[JB87] Mark Jones and Prithviraj Banerjee. "Performance of a Parallel Algorithm for Standard Cell Placement on the Intel Hypercube." In *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 807–813, 1987.

[JD88] Rajeev Jayaraman and Frederica Darema. "Error Tolerance in Parallel Simulated Annealing Techniques." In *Proceedings of the International Conference on Computer Design*, pp. 545–548. IEEE Computer Society Press, 1988.

[JG87] Prasanna Jog and Dirk Van Gucht. "Parallelisation of Probabilistic Sequential Search Algorithms." In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pp. 170–176, 1987.

[JR87] Rajeev Jayaraman and Rob A. Rutenbar. "Floorplanning by Annealing on a Hypercube Multiprocessor." In *Proceedings of the International Conference on Computer-Aided Design*, pp. 346–349, November 1987.

[KB89] Ralph M. Kling and Prithviraj Banerjee. "ESP: Placement by Simulated Evolution." *IEEE Transactions on Computer-Aided Design*, **8**(3):245–256, March 1989.

[KCP94] Sungho Kim, John A. Chandy, Steven Parkes, Balkrishna Ramkumar, and Prithviraj Banerjee. "ProperPLACE: A Portable Parallel Algorithm for Standard Cell Placement." In *International Parallel Processing Symposium (IPPS 94)*, Cancun, Mexico, 1994.

[KCV83] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing." *Science*, **220**(4598):671–680, 1983.

[KG88] N. Karasawa and W.A. Goddard. "Phase-Transitions in Molymethylene Single Chains from Monte-Carlo Simulated Annealing." *Journal of Physical Chemistry*, **92**(20):5828–5832, 1988.

[Kle75] Leonard Kleinrock. *Queueing Systems, Volume 1: Theory*. John Wiley and Sons, Inc., New York, 1975.

[Kli87] Ralph M. Kling. *"Placement by Simulated Evolution."*. Master's thesis, University of Illinois at Urbana-Champaign, 1987.

[Koz90] Krzysztof Kozminski. "VLSI Placement Competition Results, International Workshop on Layout Synthesis." Unpublished manuscript, MCNC, Research Triangle Park, North Carolina, March 1990.

[KR87] Saul A. Kravitz and Rob A. Rutenbar. "Placement by Simulated Annealing on a Multiprocessor." *IEEE Transactions on Computer-Aided Design*, **CAD-6**(4):534–549, July 1987.

[KT85] S. Kirkpatrick and G. Toulouse. "Configuration Space Analysis of Traveling Salesman Problems." *Journal de Physique*, **46**:1277–1292, 1985.

[LA87] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Boston, 1987.

[LAL88] P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. "Job Shop Scheduling by Simulated Annealing." Technical Report CWI report OS-R8809, Centrum voor Wiskunde en Informatica, July 1988.

[Lam88] Jimmy Lam. *An Efficient Simulated Annealing Schedule*. PhD thesis, Yale University, New Haven, CT, December 1988.

[LD88] Jimmy Lam and Jean-Marc Delosme. "An Efficient Simulated Annealing Schedule: Derivation." Technical Report 8816, Yale University, New Haven, CT, September 1988.

[Lee88] Fu-Hsieng Allisen Lee. *"Parallel Stochastic Optimization of Distributed Database Networks."*. Master's thesis, Utah State University, Logan, Utah, 1988.

[Leo88] Jeffrey S. Leon. "A Probabilistic Algorithm for Computing Minimum Weights of Large Error-Correcting Codes." *IEEE Transactions on Information Theory*, **34**(5):1354–1359, September 1988.

[MRR53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. "Equations of State Calculations by Fast Computing Machines." *Journal of Chemical Physics*, **21**(6):1087–1091, June 1953.

[MS89] Giorgio Mantica and Alan Sloan. "Chaotic Optimization and the Construction of Fractals: Solution of an Inverse Problem." *Complex Systems*, **3**:37–62, 1989.

[NBC88] M. Nilges, A.T. Brunger, G.M. Clore, and A.M. Gronenborn. "Determination of 3-Dimensional Structures of Proteins by Simulated Annealing with Interproton Distance Restraints—Application to Crambin, Potato Carboxypeptidase Inhibitor and Barley Serine Proteinase Inhibitor-2." *Protein Engineering*, **2**(1):27–38, 1988.

[NFN88] M. Nietovesperinas, F.J. Fuentes, and R. Navarro. "Performance of a Simulated-Annealing Algorithm for Phase Retrieval." *Journal of the Optical Society of America A: Optics and Image Science*, **5**(1):30–38, 1988.

[NSA85] J. Nulton, P. Salamon, B. Andresen, and Qi Anmin. "Quasistatic Processes as Step Equilibrations." *Journal of Chemical Physics*, **83**(1):334–338, 1985.

[OG89] R.H.J.M. Otten and L.P.P.P. van Ginneken. *The Annealing Algorithm.* Kluwer Academic Publishers, Boston, 1989.

[Rag92] Prabhakar Raghavan. "Integer Programming in VLSI Design." *Discrete Applied Mathematics*, **40**(1):29–43, 1992.

[RS91] Fabio Romeo and Alberto Sangiovanni-Vincentelli. "A Theoretical Framework for Simulated Annealing." *Algorithmica*, **6**(3):302–345, 1991.

[RSV88] Jonathan S. Rose, W. Martin Snelgrove, and Zvonko G. Vranesic. "Parallel Standard Cell Placement Algorithms with Quality Equivalent to Simulated Annealing." *IEEE Transactions on Computer-Aided Design*, **7**(3):387–396, March 1988.

[Sec88] Carl Sechen. *VLSI Placement and Global Routing Using Simulated Annealing*. Kluwer Academic Publishers, Boston, 1988.

[SG89] Arun Swami and Anoop Gupta. "Optimization of Large Join Queries." In *Proceedings of the ACM/SIGMOD International Conference on Management of Data*, pp. 8–17, June 1989.

[SH88] Galen H. Sasaki and Bruce Hajek. "The Time Complexity of Maximum Matching by Simulated Annealing." *Journal of the ACM*, **35**(2):387–403, April 1988.

[SHA89] Geoffrey Sampson, Robin Haigh, and Eric Atwell. "Natural Language Analysis by Stochastic Optimization: A Progress Report on Project APRIL." to appear in *Journal of Experimental and Theoretical Artificial Intelligence*, 1989.

[Sin82] Ya. G. Sinai. *Theory of Phase Transitions: Rigorous Results*. Pergamon Press, New York, 1982.

[SJ89] Alistair Sinclair and Mark Jerrum. "Approximate Counting, Uniform Generation, and Rapidly Mixing Markov Chains." *Information and Computation*, **82**:93–133, 1989.

[SK91] Philip N. Strenski and Scott Kirkpatrick. "Analysis of Finite Length Annealing Schedules." *Algorithmica*, **6**(3):346–366, 1991.

[SLS89] Carl Sechen, Kai-Win Lee, Bill Swartz, Jimmy Lam, and Dahe Chen. "TimberWolfSC Version 5.4: Row-Based Placement and Routing Package." Technical report, Yale University, New Haven, Connecticut, July 1989.

[Sor91] Gregory B. Sorkin. "Efficient Simulated Annealing on Fractal Energy Landscapes." *Algorithmica*, **6**(3):367–418, 1991.

[Sor92] Gregory B. Sorkin. *Theory and Practice of Simulated Annealing on Special Energy Landscapes*. PhD thesis, University of California, Berkeley, Department of Electrical Engineering and Computer Science, February 1992.

[Spe95] William M. Spears. "Simulated Annealing for Hard Satisfiability Problems." *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 1995. in publication.

[Sti94] Dyke Stiles. "Branch-and-Bound Fails for Large Network Optimization Problems." Personal communication, 1994.

[Swa89] Arun Swami. "Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques." In *Proceedings of the ACM/SIGMOD International Conference on Management of Data*, pp. 367–376, June 1989.

[Tsi89] John N. Tsitsiklis. "Markov Chains with Rare Transitions and Simulated Annealing." *Mathematics of Operations Research*, **14**(1):70–90, February 1989.

[Wal89] Dave Waltz. "Parallel Simulated Annealing at Thinking Machines." Personal communication, May 1989.

[WCM88] S.R. Wilson, W. Cui, J.W. Moskowitz, and K.E. Schmidt. "Conformational-Analysis of Flexible Molecules—Location of the Global Minimum Energy Conformation by the Simulated Annealing Method." *Tetrahedron Letters*, **29**(35):4373–4376, 1988.

[Whi84] Steve R. White. "Concepts of Scale in Simulated Annealing." In *Proceedings of the International Conference on Computer Design*, pp. 646–651. IEEE Computer Society Press, 1984.

[Wit90] Ellen E. Witte. "Results of Experiments on Decision Tree Decomposition." Personal communication, 1990.

[WL87] D.F. Wong and C.L. Liu. "Array Optimization for VLSI Synthesis." In *Proceedings of the ACM/IEEE Design Automation Conference*, 1987.

[WN86] Chang Whei-Ling and Alan Norton. "VM/EPEX C Preprocessor User's Manual Version 1.0." Technical Report RC 12246, IBM T.J.Watson Research Center, Yorktown Heights, NY, October 1986.

[ZCV88] Yi-Tong Zhou, Rama Chellappa, Aseem Vaid, and B. Keith Jenkins. "Image Restoration Using a Neural Network." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **36**(7):1141–1151, July 1988.