Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

5-1-2000

Global optimization: techniques and applications

Nathan Cahill

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Cahill, Nathan, "Global optimization: techniques and applications" (2000). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

Global Optimization: Techniques and Applications

Nathan D. Cahill
May 2000

Rochester Institute of Technology
Rochester, NY 14623

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in the field of
Industrial and Applied Mathematics

Thesis Committee:
Prof. Richard Orr, Chair
Dr. Maurino Bautista
Dr. Seshavadhani Kumar
Dr. Theodore Wilcox

This thesis fulfills the project/thesis requirement
for the degree of Master of Science in the field of
Industrial and Applied Mathematics
at Rochester Institute of Technology

Prof. Richard Orr, Chair

5/10/2006

Dr. Maurino Bautista

Dr. Seshavadhani Kumar

5/10/2006

Dr. Theodore Wilcox

RELEASE PERMISSION FORM

Rochester Institute of Technology

Global Optimization: Techniques and Applications

I, Nathan D. Cahill, hereby grant permission to any individual or organization to reproduce this thesis in whole or in part for non-commercial and non-profit purposes only.

Nathan D. Cahill

5/10/2000

Date

Abstract

Optimization problems arise in a wide variety of scientific disciplines. In many practical problems, a *global* optimum is desired, yet the objective function has multiple *local* optima. A number of techniques aimed at solving the global optimization problem have emerged in the last 30 years of research. This thesis first reviews techniques for local optimization and then discusses many of the stochastic and deterministic methods for global optimization that are in use today. Finally, this thesis shows how to apply global optimization techniques to two practical problems: the image segmentation problem (from imaging science) and the 3-D registration problem (from computer vision).

Table of Contents

Introduction

Optimization problems arise in nearly every area of science and engineering. A chemist may want to determine the minimum energy structure of a certain polymer, a biologist may need to investigate optimal dosages and scheduling of drugs administered to a cancer patient, a civil engineer may need to decide how many elevators to place in an office building so that waiting time is minimized, and an image scientist may wish to be able to automatically locate people in an image. In introductory calculus classes, the most complicated optimization problems students are faced with usually sound something like, "What is the maximum area that can be enclosed with a fence of length P?" Even though these problems teach an important lesson, rarely can real-world problems be solved solely by the analytical determination of points that satisfy first- and second-order optimality conditions.

A typical optimization problem takes the form:

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in D \end{array}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathbf{T}}$ is a vector of parameters, and f is the *objective function*. In other words, find $\mathbf{x}^*$ such that $f(\mathbf{x}^*) \leq f(\mathbf{x})$ for all $\mathbf{x} \in D$. Note that maximization problems can be posed in this manner by negating f. If D is a discrete set, the problem is referred to as a *combinatorial optimization* problem. If D is closed and f is continuous, this is a *continuous optimization* problem. This thesis will not focus on non-smooth optimization problems where the objective function is discontinuous, or non-differentiable, on a compact set (for an analysis of these types of problems see Chapter 14 of Fletcher [15]).

A variety of methods exist to determine local solutions ($\mathbf{x}'$ is a local solution if $f(\mathbf{x}') \leq f(\mathbf{x})$ for all $\mathbf{x} \in N \cap D$, where N is a neighborhood of $\mathbf{x}'$) for both combinatorial and continuous optimization problems. For combinatorial problems, a simple choice is the *iterative improvement* method (a deterministic descent method). More sophisticated methods, such as the branch and bound method (a method that solves a series of continuous optimization problems), are also available. For continuous problems, a variety of local minimization methods exist that require function values (simplex), function values and derivatives (conjugate gradient and quasi-Newton methods), or function values, first and second derivatives (Newton and restricted step methods). If D is compact, it can be represented by the intersection of equality and inequality constraints. Such constrained optimization problems arise frequently in practice and can be solved in a variety of ways; many of which are based on the idea of Lagrange multipliers.

The problem of global optimization is much more complicated. Theoretically, global minimization is as simple as finding the minimum of all local minima and boundary points. In application, this can be impractical. All local optimization techniques require an initial feasible point $\mathbf{x}^{(1)} \in D$ from which to begin the algorithm. In order to find a given local minima, it is necessary to choose a feasible point that will converge to that minima; however, determining such a feasible point can be as difficult as solving the optimization problem itself! Global optimization can be categorized as stochastic or deterministic. Stochastic or Monte Carlo methods (e.g., random search, Metropolis algorithm, simulated annealing, genetic algorithms, etc.) randomly sample D in an attempt to locate feasible points that are local minima (or lie sufficiently close to local minima). Deterministic methods carry out some non-random search procedure to weed through local minima and find the global minimum. For example, branching techniques set up simple sub-problems to solve, sub-energy tunneling iteratively transforms the objective function to remove local minima, and path following indirectly locates local minima by solving a system of differential equations.

This thesis is broken down into five chapters. Chapter 1 gives an overview of local optimization methods for combinatorial and continuous problems. Chapters 2 and 3 discuss Monte Carlo and deterministic techniques for global optimization, respectively. Chapters 4 and 5 illustrate two difficult imaging science problems and how they can be attacked with global optimization algorithms. Chapter 4 discusses the thresholding problem in image segmentation, and Chapter 5 probes the 3-D registration problem.

Chapter 1: Local Optimization

1.1 Combinatorial Optimization

The combinatorial optimization problem has the form:

$$\underset{\mathbf{x}\in D}{\text{minimize}} \qquad f(\mathbf{x}) \tag{1.1.1}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)^{\mathbf{T}}$ is a vector of parameters. We can safely assume that D is a finite set (if not, we can impose suitable restrictions on the x_i), and that D is of sufficient size that a brute force search is impractical. Ignoring multiple instances of the same function value, we can establish by the Well Ordering Principle that a global minimum $\mathbf{x}^*$ exists. This global minimum is not necessarily unique, but it satisfies:

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in D \tag{1.1.2}$$

In order to attempt to solve (1.1.1), we may design a simple algorithm that can be described in the following steps:

Algorithm (1.1.1): Combinatorial Steepest Descent

(i) Set $k = 1$. Choose $\mathbf{x}^{(1)}$.

(ii) Find $\mathbf{y}$ such that $f(\mathbf{y}) < f(\mathbf{x})$ for all $\mathbf{x} \in N_k$.

(iii) If $\mathbf{y} = \mathbf{x}^{(k)}$, terminate. If not, set $\mathbf{x}^{(k+1)} = \mathbf{y}$.

(iv) Set $k = k + 1$. Go to (ii).

In step (ii), N_k is the neighborhood of $\mathbf{x}^{(k)}$ (the set including $\mathbf{x}^{(k)}$ and all of its one-step transitions). This algorithm is a steepest descent algorithm; that is, it iterates by descending in function value as fast as possible. When the algorithm terminates at $\mathbf{x}^{(m)}$, we notice that $\mathbf{x}^{(m)}$ satisfies:

$$f(\mathbf{x}^{(m)}) \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in N_m \tag{1.1.3}$$

Since $\mathbf{x}^{(m)}$ does not necessarily satisfy (1.1.2), it may not be the global minimum; rather it is a local minimum.

The steepest descent method described in Algorithm (1.1.1) is a specific type of *iterative improvement* algorithm. On each iteration, it selects the value that most improves the objective function. In general, iterative improvement algorithms only require that the objective function improves somewhat on each iteration. Such an algorithm is categorized as a descent method and can be carried out deterministically (by searching through the neighborhood in a certain order) or stochastically (by randomly choosing points in the neighborhood) until an improvement is found.

1.2 Continuous Optimization

The continuous optimization problem has the same form as the combinatorial optimization problem in (1.1.1). In this context though, D is some continuous domain. If D is $\Re^n$ (or $\mathbb{C}^n$), then (1.1.1) is unconstrained. If D is a compact set, then it can be represented as the intersection of a set of equality and inequality constraints, and the optimization problem takes the form:

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & c_i(\mathbf{x}) = 0, & i \in E \\ & c_i(\mathbf{x}) \geq 0, & i \in I \end{array} \tag{1.2.1}$$

where E is the set of equality constraints and I is the set of inequality constraints. Analogous to (1.1.3), a point $\mathbf{x}'$ is considered a local minimum if there exists an $\varepsilon > 0$ such that:

$$f(\mathbf{x}') \leq f(\mathbf{x}) \qquad \forall \mathbf{x} \in B(\mathbf{x}', \varepsilon) \cap D \tag{1.2.2}$$

where $B(\mathbf{x}', \varepsilon)$ is the open ball of radius ε centered at $\mathbf{x}'$. If the inequality is strict, then we say $\mathbf{x}'$ is a strict local minimum.

1.2.1 Unconstrained Minimization

For the standard unconstrained optimization problem, we will assume that $D = \Re^n$ (nearly everything in this section generalizes to the case $D = \mathbb{C}^n$ as well). In order to explain the methods used to find local minima, we must understand the properties of these minima. If $\mathbf{x}'$ is a local minimum of (1.1.1), then there are no descent directions at $\mathbf{x}'$. Mathematically speaking, $f_{\mathbf{s}}(\alpha) = f(\mathbf{x}' + \alpha \mathbf{s})$ has zero slope and non-negative curvature at $\mathbf{x}'$ for any direction $\mathbf{s}$. The zero slope condition is satisfied when

$$\mathbf{g}' = \mathbf{0}, \tag{1.2.3}$$

where $\mathbf{g} = \nabla f(\mathbf{x})$ is the *gradient* of f and is given by:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial}{\partial x_1} f(\mathbf{x}), \frac{\partial}{\partial x_2} f(\mathbf{x}), \cdots, \frac{\partial}{\partial x_n} f(\mathbf{x}) \right)^{\mathbf{T}} \tag{1.2.4}$$

The curvature constraint is satisfied when

$$\mathbf{s}^{\mathbf{T}} \mathbf{G}' \mathbf{s} \geq 0 \qquad \forall \mathbf{s} \neq \mathbf{0}, \tag{1.2.5}$$

where $\mathbf{G} = \nabla^2 f(\mathbf{x})$ is the *Hessian* matrix of f and is given by:

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1 \partial x_1} f(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2} f(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} f(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1} f(\mathbf{x}) & \frac{\partial^2}{\partial x_2 \partial x_2} f(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n} f(\mathbf{x}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f(\mathbf{x}) & \frac{\partial^2}{\partial x_n \partial x_2} f(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_n \partial x_n} f(\mathbf{x}) \end{bmatrix} \tag{1.2.6}$$

Conditions (1.2.3) and (1.2.5) state that at any local minimum, the gradient vanishes and the Hessian is positive semi-definite. If the Hessian is positive definite; i.e.:

$$\mathbf{s}^{\mathbf{T}} \mathbf{G}' \mathbf{s} > 0 \qquad \forall \mathbf{s} \neq \mathbf{0}, \tag{1.2.7}$$

then the minimum is strict. Therefore, (1.2.3) and (1.2.5) are necessary conditions for a local minimum, and (1.2.3) and (1.2.7) are sufficient. (1.2.7) is not easy to test in practice; any of the following conditions are equivalent to positive definiteness and can be easier to test:

(i) all eigenvalues of $\mathbf{G}'$ are positive
(ii) all principal minors of $\mathbf{G}'$ are positive
(iii) Cholesky factors $\mathbf{G}' = \mathbf{L}\mathbf{L}^{\mathbf{T}}$ exist with l_{ii} positive

Descent methods with exact line search were some of the earliest methods for local optimization (Curry [10]). These methods rely on the fact that if $\mathbf{x}^{(k)}$ does not satisfy (1.2.3) and (1.2.5), then there must

exist a direction $\mathbf{s}^{(k)}$ such that $\mathbf{s}^{(k)\mathrm{T}}\mathbf{g}^{(k)} < 0$. Then $\mathbf{s}^{(k)}$ is a direction along which f can be reduced (a descent direction). An exact line search is performed to locate the minimum of f along $\mathbf{s}^{(k)}$, and the resulting point becomes next iterate $\mathbf{x}^{(k+1)}$. Descent methods with line search can be described in the following steps:

Algorithm (1.2.1): Descent methods with line search

(i) Set $k = 1$. Choose $\mathbf{x}^{(1)}$.

(ii) Terminate if $\mathbf{x}^{(k)}$ satisfies (1.2.4) and (1.2.5).

(iii) Choose $\mathbf{s}^{(k)}$ such that $\mathbf{s}^{(k)\mathrm{T}}\mathbf{g}^{(k)} < 0$. (1.2.8)

(iv) Find $\alpha^{(k)}$ that minimizes $f\left(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{s}^{(k)}\right)$. (1.2.9)

(v) Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{s}^{(k)}$.

(vi) Set $k = k + 1$. Go to (ii).

In general, it is not easy to solve (1.2.9) exactly; therefore, an approximation to $\alpha^{(k)}$ must suffice. A comprehensive treatment of the line search sub-problem is given in Chapter 2 of Fletcher [15]. At a high-level, an inexact line search is comprised of a *bracketing* phase, where an acceptable interval of points is located, and a *sectioning* phase, where the bracket is partitioned into arbitrarily small sections until a minimum is found to within a preset tolerance.

The choice of $\mathbf{s}^{(k)}$ in (1.2.8) can radically affect the convergence properties of a descent method. An intuitive choice is the *steepest descent* direction, the direction $\mathbf{s}^{(k)}$ that leads to the most rapid initial descent in f. The steepest direction at $\mathbf{x}^{(k)}$ is that of the negative gradient $-\mathbf{g}^{(k)}$. Therefore, the steepest descent method with line search is given by Algorithm (1.2.1) with (iii) replaced by:

(iii) Set $\mathbf{s}^{(k)} = -\mathbf{g}^{(k)}$.

In practice, however, steepest descent is a poor method in that it can exhibit oscillatory behavior. In fact, it can be shown that in certain instances, steepest descent with exact line search applied to a simple quadratic function does not terminate in a finite number of iterations!

In light of the poor convergence properties of steepest descent, another class of line search methods was developed that attempt to improve convergence while maintaining some notion of descent. These methods, called *conjugate gradient* methods, are based on the conjugacy property of vectors. The vectors $\mathbf{s}^{(1)}$ and $\mathbf{s}^{(2)}$ are conjugate with respect to $\mathbf{G}$ (or $\mathbf{G}$-conjugate) if:

$$\mathbf{s}^{(1)\mathrm{T}}\mathbf{G}\mathbf{s}^{(2)} = 0 \tag{1.2.10}$$

For a simple quadratic function, a conjugate gradient method can be constructed by choosing $\mathbf{s}^{(k+1)}$ to be the component of $-\mathbf{g}^{(k+1)}$ conjugate to $\mathbf{s}^{(1)}, \mathbf{s}^{(2)}, \ldots, \mathbf{s}^{(k)}$ and performing a line search as in Algorithm (1.2.1). The Fletcher-Reeves conjugate gradient method (Fletcher and Reeves [16]) uses the specific formula:

$$\mathbf{s}^{(1)} = -\mathbf{g}^{(1)}, \qquad \mathbf{s}^{(k+1)} = -\mathbf{g}^{(k+1)} + \frac{\left\|\mathbf{g}^{(k+1)}\right\|_2}{\left\|\mathbf{g}^{(k)}\right\|_2}\mathbf{s}^{(k)} \tag{1.2.11}$$

This method preserves descent and terminates in n or fewer iterations for a n-dimensional quadratic function. Note that a positive definite quadratic function $q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{G}\mathbf{x} + \mathbf{g}^{\mathrm{T}}\mathbf{x} + \mathbf{b}$ can be minimized analytically by solving the system $\mathbf{G}\mathbf{x}^{*} = -\mathbf{g}$, but for large-scale problems conjugate gradient methods do not require the storage of $O\left(n^2\right)$ elements.

Another family of local minimization methods is based on iteratively minimizing quadratic Taylor series approximations to f. Given a point $\mathbf{x}^{(k)}$, Newton's method constructs the truncated Taylor series approximation $q^{(k)}(\mathbf{s})$ to f at $\mathbf{x}^{(k)}$:

$$f\left(\mathbf{x}^{(k)}+\mathbf{s}\right) \approx q^{(k)}(\mathbf{s}) = \tfrac{1}{2}\mathbf{s}^{\mathrm{T}}\mathbf{G}^{(k)}\mathbf{s} + \mathbf{g}^{(k)^{\mathrm{T}}}\mathbf{s} + f\left(\mathbf{x}^{(k)}\right) \tag{1.2.12}$$

A new iterate is computed by finding the minimum of $q^{(k)}(\mathbf{s})$:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)}, \tag{1.2.13}$$

$$\text{where } \mathbf{G}^{(k)}\mathbf{s}^{(k)} = -\mathbf{g}^{(k)} \tag{1.2.14}$$

The system (1.2.14) can be solved by a variety of different methods. Solution by Cholesky factorization enables the positive definiteness of $\mathbf{G}^{(k)}$ to be checked. (1.2.13) can be replaced with a line search as in steps (iv) and (v) in Algorithm (1.2.1) if desired. If $\mathbf{x}^{(k)}$ is sufficiently close to the local minimum $\mathbf{x}'$, then Newton's method exhibits second order convergence. In general, this is much more efficient than conjugate gradient methods, but Newton's method (or Quasi-Newton methods) may be impractical for large-scale problems. Newton's method can also be used in conjunction with a line search carried out along $\mathbf{s}^{(k)}$.

The main difficulty with Newton's method is that the Hessian matrix may only be positive definite in a small neighborhood of the solution. This difficulty can be overcome by adding a multiple of the unit matrix and solving the system:

$$\left(\mathbf{G}^{(k)} + \nu\mathbf{I}\right)\mathbf{s}^{(k)} = -\mathbf{g}^{(k)} \tag{1.2.15}$$

in place of (1.2.14). This idea appeared originally in Levenberg [29] and Marquardt [31]. By selecting ν so that $\mathbf{G}^{(k)} + \nu\mathbf{I}$ is positive definite, (1.2.15) uses the quadratic information present in (1.2.12) to find the next iterate and eventually converges to the local solution. *Restricted step* methods carry this modification even further by constraining the solution to (1.2.15) to lie within some *trust region*, which is generally an ellipsoid or hyperparallelepiped centered at $\mathbf{x}^{(k)}$.

When second derivatives are difficult (or inefficient) to compute, another family of methods (Quasi-Newton methods) can be used which approximate the inverse Hessian matrix $\mathbf{G}^{(k)^{-1}}$ by a positive definite matrix $\mathbf{H}^{(k)}$ and then carry on a line search as in Newton's method. The matrix $\mathbf{H}^{(k)}$ is updated iteratively (starting with some initial positive definite matrix such as $\mathbf{H}^{(1)} = \mathbf{I}$) by incorporating second derivative information obtained at each iteration. In order to do this, we will first expand the gradient (1.2.4) about $\mathbf{x}^{(k)}$:

$$\nabla f\left(\mathbf{x}^{(k)} + \mathbf{u}^{(k)}\right) = \nabla f\left(\mathbf{x}^{(k)}\right) + \left[\nabla^2 f\left(\mathbf{x}^{(k)}\right)\right]\mathbf{u}^{(k)} + o\left(\left\|\mathbf{u}^{(k)}\right\|\right). \tag{1.2.16}$$

This can be rewritten as:

$$\mathbf{v}^{(k)} = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)} = \mathbf{G}^{(k)}\mathbf{u}^{(k)} + o\left(\left\|\mathbf{u}^{(k)}\right\|\right). \tag{1.2.17}$$

Therefore, $\mathbf{H}^{(k)}$ can be updated so that

$$\mathbf{H}^{(k+1)}\mathbf{v}^{(k)} = \mathbf{u}^{(k)}. \tag{1.2.18}$$

A rank-two updating method was proposed by Davidon [12], Fletcher and Powell [13]. A rank two update is given by:

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + \alpha_1 \mathbf{a}\mathbf{a}^{\mathrm{T}} + \alpha_2 \mathbf{b}\mathbf{b}^{\mathrm{T}}. \tag{1.2.19}$$

Since the quasi-Newton condition (1.2.18) must be satisfied, we have

$$\mathbf{u}^{(k)} = \mathbf{H}^{(k)}\mathbf{v}^{(k)} + \alpha_1 \mathbf{a}\mathbf{a}^{\mathrm{T}}\mathbf{v}^{(k)} + \alpha_2 \mathbf{b}\mathbf{b}^{\mathrm{T}}\mathbf{v}^{(k)}. \tag{1.2.20}$$

If the choices $\mathbf{a} = \mathbf{u}^{(k)}$ and $\mathbf{b} = \mathbf{H}^{(k)}\mathbf{v}^{(k)}$ are made, then (1.2.20) reduces to the *DFP formula*:

$$\mathbf{H}_{DFP}^{(k+1)} = \mathbf{H}^{(k)} + \frac{\mathbf{u}^{(k)}\mathbf{u}^{(k)\mathrm{T}}}{\mathbf{u}^{(k)\mathrm{T}}\mathbf{v}^{(k)}} - \frac{\mathbf{H}^{(k)}\mathbf{v}^{(k)}\mathbf{v}^{(k)\mathrm{T}}\mathbf{H}^{(k)}}{\mathbf{v}^{(k)\mathrm{T}}\mathbf{H}^{(k)}\mathbf{v}^{(k)}}. \tag{1.2.21}$$

Broyden [5], Fletcher [14], Goldfarb [18], and Shanno [41] introduced another updating formula that is the dual of the DFP formula. The *BFGS formula* is given by:

$$\mathbf{H}_{BFGS}^{(k+1)} = \mathbf{H}^{(k)} + \left(1 + \frac{\mathbf{v}^{(k)\mathrm{T}}\mathbf{H}^{(k)}\mathbf{v}^{(k)}}{\mathbf{u}^{(k)\mathrm{T}}\mathbf{v}^{(k)}}\right)\frac{\mathbf{u}^{(k)}\mathbf{u}^{(k)\mathrm{T}}}{\mathbf{u}^{(k)\mathrm{T}}\mathbf{v}^{(k)}} - \left(\frac{\mathbf{u}^{(k)}\mathbf{v}^{(k)\mathrm{T}}\mathbf{H}^{(k)} + \mathbf{H}^{(k)}\mathbf{v}^{(k)}\mathbf{u}^{(k)\mathrm{T}}}{\mathbf{u}^{(k)\mathrm{T}}\mathbf{v}^{(k)}}\right). \tag{1.2.22}$$

In fact, an infinite number of updating formulas exist that satisfy the quasi-Newton condition. The *Broyden* family of methods is given by

$$\mathbf{H}_{\phi}^{(k+1)} = \phi\mathbf{H}_{BFGS}^{(k+1)} + (1-\phi)\mathbf{H}_{DFP}^{(k+1)}. \tag{1.2.23}$$

If we choose $\phi = \dfrac{\mathbf{u}^{(k)\mathrm{T}}\mathbf{v}^{(k)}}{\mathbf{u}^{(k)\mathrm{T}}\mathbf{v}^{(k)} - \mathbf{v}^{(k)\mathrm{T}}\mathbf{H}^{(k)}\mathbf{v}^{(k)}}$, then we have a rank one update to $\mathbf{H}^{(k)}$. Both the DFP and BFGS formulae exhibit desirable properties; they preserve the positive definiteness of $\mathbf{H}^{(k)}$ so a descent direction is always chosen, they generate conjugate directions and terminate in at most n iterations for quadratic functions, and they exhibit superlinear convergence for general functions. It has been shown in the literature that the BFGS method globally converges when inexact line searches are performed (under certain conditions); consequently, it has become the quasi-Newton method of choice.

When it becomes difficult (or extremely inefficient) to compute first derivatives, any of the previous methods can be used with finite difference approximations to the gradient vector (and the Hessian matrix, for Newton's method). Formula (1.2.24) shows how to approximate each element of the gradient vector with centered differences:

$$g_i(\mathbf{x}) \approx \frac{f(\mathbf{x} + \delta\mathbf{e}_i) - f(\mathbf{x} - \delta\mathbf{e}_i)}{2\delta}. \tag{1.2.24}$$

where $\mathbf{e}_i$ is the i^{th} column of the identity matrix. A finite difference approach is probably the best no-derivative methods for unconstrained optimization; however, it still may be too costly to perform $2n$ function evaluations each iteration. If this is the case, other no-derivative methods can be used. The most popular ones are based on the simplex method (Spendley et al. [43]).

In the simplex method, f is evaluated at a set of $(n+1)$ equidistant points in $\Re^n$. A new simplex is formed by reflecting the point with the largest function value through the hyperplane containing the other n points. When a point $\mathbf{x}$ has been in a simplex for a given number of iterations, the simplex is contracted by relocating the remaining n points half the distance to $\mathbf{x}$. Termination generally occurs when the simplex occupies a smaller volume than some threshold. The Nelder-Mead method (Nelder and Mead [34]) is a simplex method that allows for irregularly shaped simplices, and for expansions as well as contractions based on the local geometry of f. In general, the simplex method can be a poor choice because it does not guarantee convergence to a local minimum. However, it can be very useful when attempting to optimize noisy objective functions.

1.2.2 Constrained Minimization

For the standard constrained optimization problem, the set D is compact and can be described by the intersection of the equality and inequality constraints of (1.2.1). In unconstrained minimzation, any local minimum satisfies conditions (1.2.3) and (1.2.5). In constrained minimization, these conditions need not hold. All that is necessary for a local minimum is that no *feasible* descent direction exists. In order to describe this mathematically, let us first consider the case where D contains no inequality constraints (i.e., $I = \Phi$). The following development is based on that given in Chapter 9 of Fletcher [15].

Suppose $\mathbf{x}'$ is a local minimizer of (1.2.1). Expanding the constraints about $\mathbf{x}'$, we have:

$$c_i(\mathbf{x}' + \delta) = c_i(\mathbf{x}') + \delta^{\mathbf{T}} \mathbf{a}_i' + o(\|\delta\|). \tag{1.2.25}$$

where $\mathbf{a}_i = \nabla c_i(\mathbf{x})$ is the i^{th} column of the *Jacobian matrix*. If we assume that δ is a feasible step, then $c_i(\mathbf{x}' + \delta) = c_i(\mathbf{x}') = 0$, so we know that

$$\delta^{\mathbf{T}} \mathbf{a}_i' = 0 \qquad \forall i \in E. \tag{1.2.26}$$

If the condition

$$\delta^{\mathbf{T}} \mathbf{g}' < 0 \tag{1.2.27}$$

holds, then δ is a descent direction at $\mathbf{x}'$. Since this contradicts the assumption that $\mathbf{x}'$ is a local minimizer of (1.2.1), it must be true that no direction δ satisfies (1.2.26) and (1.2.27) simultaneously.

If we say that

$$\mathbf{g}' = \sum_{i \in E} \lambda_i' \mathbf{a}_i' = \mathbf{A}' \boldsymbol{\lambda}', \tag{1.2.29}$$

then we can see that $\delta^{\mathbf{T}} \mathbf{g}' = \sum_{i \in E} \lambda_i' \delta^{\mathbf{T}} \mathbf{a}_i' = 0$, so (1.2.29) does not allow for (1.2.26) and (1.2.27) to be simultaneously satisfied for any δ. In fact, it can be shown that (1.2.29) *must* hold at a local minimum, so (1.2.29) becomes a *necessary* condition for a local minimum.

The condition (1.2.29) states that at a local minimum, the gradient must be a linear combination of the constraint gradients. The coefficients (λ_i's) of the linear combination are referred to as *Lagrange multipliers*. A simple way to restate the first order necessary conditions arises from the *Lagrangian function*:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \sum_i \lambda_i c_i(\mathbf{x}). \tag{1.2.30}$$

The condition (1.2.29) is equivalent to:

$$\nabla_x L(\mathbf{x}', \boldsymbol{\lambda}') = \mathbf{0}, \tag{1.2.31}$$

and satisfaction of the constraints in (1.2.1) is equivalent to:

$$\nabla_\lambda L(\mathbf{x}', \boldsymbol{\lambda}') = \mathbf{0}. \tag{1.2.32}$$

Second-order conditions can be derived from the Lagrangian function. It turns out that if the necessary conditions are satisfied, the condition

$$\delta^{\mathbf{T}}\nabla_x^2 L(\mathbf{x}',\boldsymbol{\lambda}')\delta > 0 \tag{1.2.33}$$

for all $\delta \neq 0$ that satisfy (1.2.26) is a sufficient condition for a local minimum.

When D contains inequality constraints, we are only concerned with the *active* constraints at a given minimum; that is, the set of constraints that satisfy $c_i(\mathbf{x}') = 0$. We will denote the active set of constraints at $\mathbf{x}'$ by $\mathcal{A}'$, and the set of active inequality constraints at $\mathbf{x}'$ by $I' = \mathcal{A}' \cap I$. If we now observe the Taylor series expansion of the constraints at $\mathbf{x}'$ given by (1.2.25), we see that $c_i(\mathbf{x}') = 0$ and $c_i(\mathbf{x}' + \delta) \geq 0$, so we know that

$$\delta^{\mathbf{T}}\mathbf{a}_i' \geq 0 \qquad \forall i \in I'. \tag{1.2.34}$$

It can be shown that the following conditions must hold for (1.2.34) and (1.2.26) to be satisfied while (1.2.27) is not satisfied:

$$\mathbf{g}' = \sum_{i \in \mathcal{A}'} \lambda_i' \mathbf{a}_i' = \mathbf{A}'\boldsymbol{\lambda}', \tag{1.2.35}$$

$$\lambda_i' \geq 0 \qquad \forall i \in I'. \tag{1.2.36}$$

Therefore, any active inequality constraints must have positive multipliers. By convention, we say that any inactive inequality constraints correspond to zero multipliers.

The first-order necessary conditions for minimization can be restated as follows:

(i)	$\nabla_x L(\mathbf{x}',\boldsymbol{\lambda}') = \mathbf{0}$	
(ii)	$c_i(\mathbf{x}') = 0,$	$\forall i \in E$
(iii)	$c_i(\mathbf{x}') \geq 0,$	$\forall i \in I$
(iv)	$\lambda_i' \geq 0,$	$\forall i \in I$
(v)	$\lambda_i' c_i(\mathbf{x}') = 0,$	$\forall i$

These conditions are known as the first-order KT conditions (Kuhn and Tucker [27]). The first two conditions follow directly from (1.2.31) and (1.2.32). The third condition follows from (1.2.1), the fourth from (1.2.36), and the fifth from the convention that inactive constraints have zero multipliers.

The theoretical approach to solving constrained minimization problems is to locate all points that satisfy the KT conditions and see which, if any, satisfy the second-order conditions. This approach is known as the *method of Lagrange multipliers*. Like the theoretical approach to unconstrained minimization, it is frequently difficult, if not impossible, to apply this method to a given "practical" problem. Because of this, a number of alternate algorithms have arisen based on the structure of the objective function and constraint functions.

If the objective function and constraint functions are linear, (1.2.1) is a *linear programming* problem. A LP is most readily solved by either the *Dantzig-Wolfe simplex* method (Dantzig and Wolfe [11]) or the *active set* method. Both methods rely on the fact that in a LP problem, any extreme value must take place at a vertex of the set of constraints. At a high level, they search through different vertices of the feasible region, swapping constraints in and out of the active set until a minimum is found. For *quadratic programming* problems, where the objective function is quadratic and the constraints are linear, a similar active set approach is employed that solves the equality constrained QP at each iteration by transforming it into an unconstrained QP (via QR decomposition or generalized elimination; see Fletcher [15]). General linearly constrained problems can be solved in the same manner, by iteratively transforming into an unconstrained problem, and solving each unconstrained problem by a Quasi-Newton method, for example. The most complicated constrained minimization problems are called *nonlinear programming* problems, where neither the objective function nor the constraints are linear. NLP problems can be solved by penalty (Hestenes [20]) or barrier methods, or by solving a series of QP sub-problems (also known as Sequential Quadratic Programming (SQP), Wilson [45] and Schittowski [40]).

Chapter 2: Monte Carlo Global Optimization

2.1 From Local to Global Optimization

The task of global minimization can be daunting compared to that of local optimization. The theoretical solution to the global optimization problem is to determine the set of all points that satisfy (1.1.3) or (1.2.2), sort them by increasing function value, and take the first element (or elements, if the function value is not unique) of the set to be the global minimum. This process can rarely be carried out analytically, unless the objective function and constraints are simple in nature. In general, though, even the simplest *looking* problems can be extremely difficult to optimize. It is no wonder, then, that many algorithms based on Monte Carlo techniques have emerged in recent years.

One way to set up a simple Monte Carlo algorithm is to represent the search space D as the union of a finite number of (not necessarily mutually exclusive) sets S_k, $k = 1,\ldots,N$. In the case where D is unbounded, suitable lower and upper bounds on the parameters must be chosen. A set of points $\left\{\mathbf{x}^{(k)} \mid \mathbf{x}^{(k)} \in S_k, k = 1,\ldots,N\right\}$ is randomly chosen, and each $\mathbf{x}^{(k)}$ used as an initial point in one of the local optimization algorithms described in the previous chapter. We will illustrate this technique by minimizing the following function:

$$f(\mathbf{x}) = \sin(\pi x_1)\cos(4\pi x_2) + \cos\left(4\pi^2 x_1 x_2\right) \qquad (2.1.1)$$
$$0 \le x_1 \le 1, \quad 0 \le x_2 \le 1$$

As can be seen in Figure (2.1.1), $f(\mathbf{x})$ has many local minima in this region. The global minimum occurs at approximately $\mathbf{x}^* = (0.5293, 0.7513)^{\mathbf{T}}$.

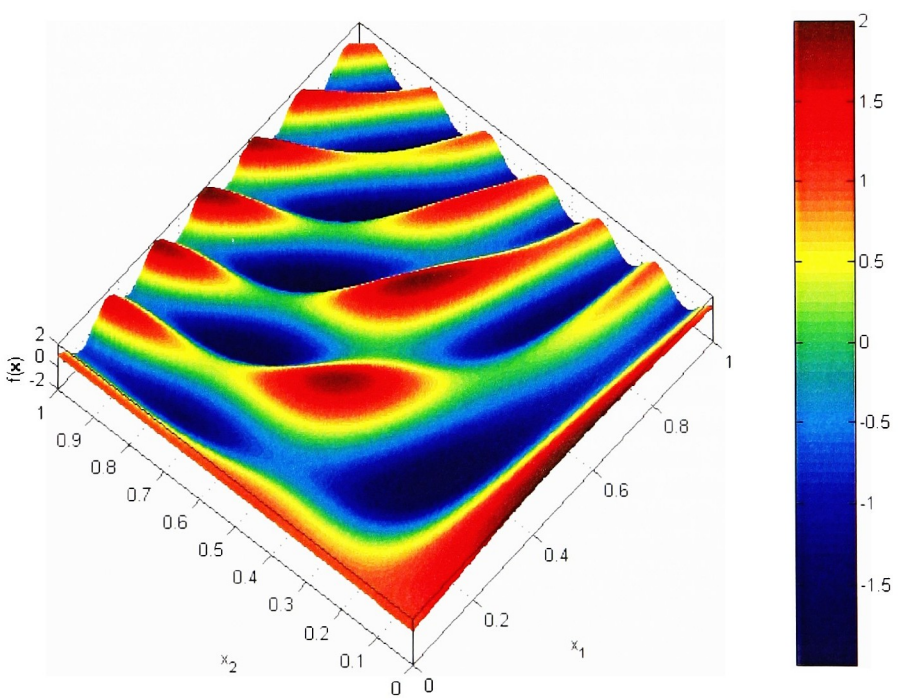

Figure (2.1.1): Surface plot of $f(\mathbf{x})$

We can apply a Monte Carlo algorithm by dividing the feasible region into a 5 x 5 grid. We then choose a random point from each patch, and apply the SQP method (Schittowski [40]) iteratively with each random point as a starting point. The left half of figure (2.1.2) shows the 5 x 5 grid with a random point

chosen in each square, and the right half shows the set of local minima found by different runs of the SQP method. Note that in this case, all local minima (including the global minimum) were found.

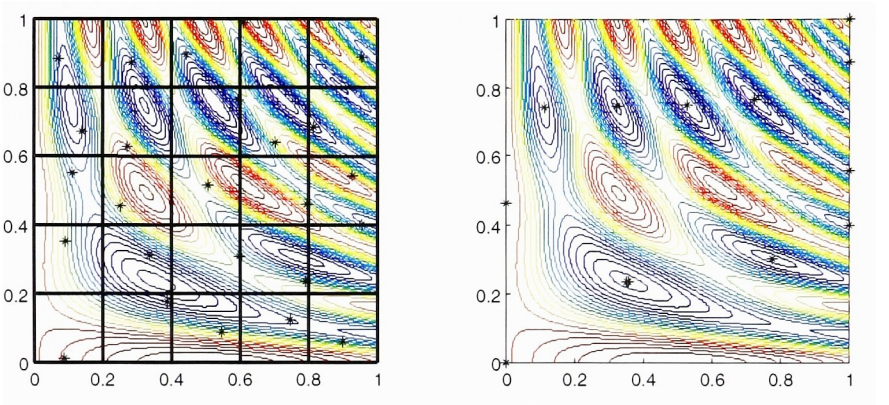

Figure (2.1.2): Monte Carlo algorithm for minimizing $f(\mathbf{x})$

In order for a Monte Carlo algorithm like this one to be robust, the covering sets S_k should be small enough that they can be contained within basins of attraction of local minima. If the S_k are too large, the global minimum may not be found. On the other hand, if the S_k are too small, the algorithm can become very inefficient. One major difficulty with this type of algorithm is that the only way to construct an "optimal" collection of covering sets requires knowledge of the basins of attraction of the local minima.

Another major problem with this Monte Carlo algorithm is that the required number of covering sets S_k increases exponentially with the dimension of the problem. One way to combat this problem is to keep track of the sets that have been visited during successive calls to the local optimizer. If a local optimizer enters a set that has previously been visited, the local optimizer can be prematurely terminated. However, the S_k must be sufficiently small (and hence there must be a large number of them) in order to reap the benefits of premature termination.

In some optimization problems, it may be very expensive to compute function values, gradient vectors, and Hessian matrices, so performing local minimizations from a succession of points may be out of the question. In these cases, the only recourse (given current processor speeds) is to search for minima based on function values alone. One way of doing this is the simplex method, as described in Chapter 1. However, since the simplex method is a local minimization method, global minimization still requires that it be carried out from a variety of initial points. More efficient methods can be based on random search techniques. The rest of this chapter discusses pure and adaptive random searches, simulated annealing, and genetic algorithms.

2.2 Random Searches

The simplest random search technique is the *pure random search*. The algorithm is very straightforward: randomly generate N feasible points, and choose the point with the smallest objective function value for the global minimum. The pure random search is described as a descent method in Algorithm (2.2.1). If refinement is needed, a local minimization can be performed from the optimal point found by the search. The pure random search does not yield an efficiency improvement over the algorithm of Section 2.1; in addition, it requires that N grow exponentially with the dimension of the problem in order to achieve a given level of accuracy.

Algorithm (2.2.1): Pure Random Search

(i) Set $k = 0$, $c = 0$, $y = \infty$.

(ii) Randomly generate $\mathbf{x}^{(k+1)} \in D$ (D is the feasible region)

(iii) If $f\left(\mathbf{x}^{(k+1)}\right) < y$, set $y = f\left(\mathbf{x}^{(k+1)}\right)$ and $k = k+1$.

(iv) Set $c = c+1$. If $c < N$, go to (ii).

Patel, Smith, and Zabinsky [36] and Zabinsky and Smith [47] describe a *pure adaptive search* algorithm that only requires N to grow linearly with the dimension of the problem. The PAS algorithm can be described in the following way:

Algorithm (2.2.2): Pure Adaptive Search

(i) Set $k = 0$, $y = \infty$.

(ii) Randomly generate $\mathbf{x}^{(k+1)} \in S_{k+1} = \{\mathbf{x} \mid \mathbf{x} \in D \cap f(\mathbf{x}) < y\}$.

(iii) Set $y = f\left(\mathbf{x}^{(k+1)}\right)$. If $k < N$, go to (ii).

Even though this search method is theoretically more efficient than the PRS, it may be difficult, if not impossible, to generate the nested sets in step (ii).

Romeijn [37] describes an adaptive search method that differs from the PAS algorithm in that iterates are chosen from the feasible region (i.e., $S_k = D$ in step (ii)). The adaptive nature of Romeijn's method is that the probability distribution used to generate $\mathbf{x}^{(k+1)}$ changes on each iteration (not like the PRS, in which the distribution is generally uniform). The sequence of distributions is chosen to converge to the stationary distribution of a Markov Chain. In the next section, simulated annealing is introduced; this is a specific type of Romeijn's adaptive search method.

2.3 Simulated Annealing

Since being introduced by Kirkpatrick, Gelatt, and Vecchi [26] and Cerny [7], simulated annealing has quickly become one of the best general purpose Monte Carlo global optimization algorithms. It is based on the physical model of *annealing*, where a solid is coaxed into a minimum energy crystalline state by slowly reducing its temperature. If the temperature is reduced too quickly, metastable structures result that have higher energy than the crystalline state. The fundamental building block of simulated annealing is the Metropolis algorithm, introduced some 30 years earlier by Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller [33], which simulates the energy configuration of a thermally equilibrating solid. This section will develop simulated annealing in a combinatorial setting and then show how it can be extended to the continuous case.

When a solid is in thermal equilibrium, the probability that the solid is configured in a given energy state is governed by the *Boltzmann distribution*:

$$P_T(\mathbf{x} = i) = \frac{1}{Z} e^{-E_i/T}, \tag{2.3.1}$$

where i is a state with energy E_i (i can be thought of as a vector of parameters and E_i the corresponding value of the objective function), Z is a partition function, and T is a parameter that acts like temperature. When T approaches zero, the Boltzmann distribution concentrates its mass in the low energy states. In the limit, the only states with non-zero probability are the minimum-energy states.

Metropolis illustrated that a combinatorial optimization scheme similar to a stochastic iterative improvement algorithm can be constructed so that the long-term probabilities of the occurrence of energy states approach a Boltzmann distribution. The difference between the Metropolis algorithm and a stochastic iterative improvement algorithm is that Metropolis allows transitions to higher-energy states with positive probability, whereas, an iterative improvement algorithm only allows transitions to lower-energy states. Because of this, the Metropolis algorithm has the potential to "jump" out of a local minimum to

explore other feasible areas of the state space. The probability that a higher energy transition is accepted is given by:

$$P_{accept}(\mathbf{x}_{k+1} = j \mid \mathbf{x}_k = i, E_j > E_i) = e^{-(E_j - E_i)/T}. \tag{2.3.2}$$

This is known as the *Metropolis criteria*. We can encapsulate the probability of accepting a transition to any state by:

$$A_{ij}(T) = P_{accept}(\mathbf{x}_{k+1} = j \mid \mathbf{x}_k = i) = e^{\min\{-(E_j - E_i)/T, 0\}}. \tag{2.3.3}$$

We can now state the Metropolis algorithm (note the similarities to an iterative improvement algorithm):

Algorithm (2.3.1): Metropolis algorithm

(i) Set $k = 1$. Choose $\mathbf{x}^{(1)}$.

(ii) Randomly choose $\mathbf{x}^{(k+1)} \in N_k$ (the neighborhood of one-step transitions)

(iii) Accept $\mathbf{x}^{(k+1)}$ with probability $A_{ij}(T)$. If not accepted, set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$.

(iv) Set $k = k + 1$. Go to (ii).

No termination criteria are explicitly stated in Algorithm (2.3.1). In practice, the algorithm can be terminated when no substantial decrease in function value has been made over the last n iterations.

It is interesting to note that any given run of a Metropolis algorithm is a realization of a Markov Chain. In order to construct the corresponding probability transition matrix, we must not only consider the transition acceptance probabilities given by (2.3.3), but the probability that a particular state is generated (as in step (ii) of Algorithm (2.3.1)). Since each iterate is randomly chosen from the set of possible one-step transitions from the previous iterate, the generation probabilities are given by:

$$G_{ij} = P_{choose}(\mathbf{x}_{k+1} = j \mid \mathbf{x}_k = i) = \frac{1}{\#\text{ of states in } N_k}. \tag{2.3.4}$$

The probability transition matrix $P(T)$ is found by considering both the generation and acceptance probabilities:

$$P_{ij}(T) = \begin{cases} G_{ij} A_{ij}(T), & j \neq i \\ 1 - \sum_{r \neq i} G_{ir} A_{ir}(T), & j = i \end{cases}. \tag{2.3.5}$$

Aarts and van Laarhoven [1] prove that the stationary distribution of this Markov Chain is indeed Boltzmann. Ross [38] shows that the vector of stationary probabilities is a left eigenvector of $P(T)$ with corresponding eigenvalue $\lambda = 1$. Therefore, one way to solve the optimization problem is to form $P(T)$ and find the appropriate left eigenvector. However, forming $P(T)$ requires that the objective function be evaluated for every possible state, which is equivalent to a brute force search. It still behooves us to perform the analysis on a small problem.

Consider a combinatorial optimization problem with only four states. The possible transitions are indicated by arrows in the following diagram:

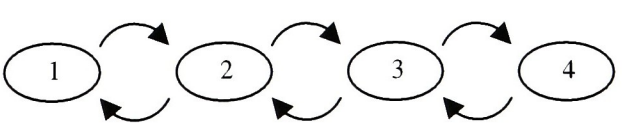

Figure (2.3.1): Diagram of states and possible transitions

We will assign objective function values to each state: $f(1)=E_1=1$, $f(2)=E_2=0$, $f(3)=E_3=2$, and $f(4)=E_4=1$. The generation and acceptance probability matrices can easily be constructed:

$$\mathbf{G}=\begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{A}(T)=\begin{bmatrix} 1 & 1 & 0 & 0 \\ e^{-1/T} & 1 & e^{-2/T} & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & e^{-1/T} & 1 \end{bmatrix}. \tag{2.3.6}$$

From (2.3.5), we can construct the transition matrix:

$$\mathbf{P}(T)=\begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2}e^{-1/T} & 1-\frac{1}{2}e^{-1/T}-\frac{1}{2}e^{-2/T} & \frac{1}{2}e^{-2/T} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & e^{-1/T} & 1-e^{-1/T} \end{bmatrix}. \tag{2.3.7}$$

It can be verified that $\pi(T)=\left(\frac{e^{-1/T}}{2e^{-2/T}+2e^{-1/T}+2}, \frac{1}{e^{-2/T}+e^{-1/T}+1}, \frac{e^{-2/T}}{e^{-2/T}+e^{-1/T}+1}, \frac{e^{-1/T}}{2e^{-2/T}+2e^{-1/T}+2}\right)$ is a left eigenvector of $\mathbf{P}(T)$ with unit eigenvalue and that the elements of $\pi(T)$ are the masses of a Boltzmann distribution. Also, it is clear that $\lim_{T\to 0}\pi(T)=(0,1,0,0)$, so the Metropolis algorithm applied to this problem can identify the global minimum if a sufficiently large number of iterations are performed and if T is small.

One of the most difficult aspects of performing combinatorial optimization with the Metropolis algorithm is the choice of an appropriate temperature parameter. If T is too large, the stationary Boltzmann distribution may not be concentrated enough at the global minimum. If T is too small, the number of iterations required to converge to the stationary distribution can be tremendous. There is definitely a trade-off that must be made, but the decision of how to make it can be very difficult. *Simulated annealing* utilizes a series of Metropolis algorithms, each with smaller T, so as to converge more rapidly to a stationary distribution highly concentrated at the global minimum. In terms of the physical analogy, a solid is cooled slowly enough so that thermal equilibrium is achieved at each temperature, resulting in a minimum energy crystalline structure. Therefore, much of the success of a simulated annealing algorithm depends on the *cooling schedule*, or the choice of decreasing values of T.

Simulated annealing can be described by Algorithm (2.3.2):

Algorithm (2.3.2): Simulated annealing

(i) Set $m=1$. Choose $\mathbf{x}^{(1)}$, $T^{(1)}$.

(ii) Run Algorithm (2.3.1) starting at $\mathbf{x}^{(m)}$ with $T=T^{(m)}$.

(iii) Set $\mathbf{x}^{(m+1)}$ to be the state at which (ii) was terminated.

(iv) Generate $T^{(m+1)}$ according to cooling schedule.

(v) Set $m=m+1$. Go to (ii).

As was the case in the Metropolis algorithm, termination of step (ii) and overall termination of Algorithm (2.3.2) can be chosen to occur by a variety of means. For example, the user could choose to perform a single iteration at step (ii) and have a large number of slowly-changing temperatures in the cooling schedule. In this case, a run of the algorithm would be a realization of an inhomogeneous Markov Chain. For another problem, the user could choose a relatively coarse cooling schedule and terminate each Metropolis algorithm after a large number of iterations.

Because the success and efficiency of simulated annealing greatly depends on the cooling schedule, it is important to pick a schedule that works well. A simple geometric schedule was proposed by Kirkpatrick, Gelatt, and Vecchi [26], and many variants of this method have appeared in the literature since then. The initial value of T is experimentally determined so that the *acceptance ratio* (ratio of acceptable transitions to possible transitions at a given point) is close to one (usually around 0.95). Then the first Metropolis algorithm is almost a random walk through the state space. New values of T are generated

geometrically; i.e., $T^{(m+1)} = \alpha T^{(m)}$, with α typically between 0.9 and 0.99. Termination of the algorithm can occur if no substantial improvement is observed over a number of iterations.

Johnson et al. [25] describe some other simple cooling schedules. Besides the geometric schedule, they describe a linear schedule, where T decreases linearly, and a logarithmic schedule, where T decreases according to:

$$T^{(m)} = \frac{C}{1+\ln(m)}, \tag{2.3.8}$$

where C is a constant. They conclude that none of these schedules yield a dramatic advantage over the geometric schedule.

Aarts and van Laarhoven [1] describe a more sophisticated cooling schedule. Like the geometric schedule, the initial value of T is determined so that the acceptance ratio is close to one. New values of T are given by the iteration:

$$T^{(m+1)} = \frac{T^{(m)}}{1+\dfrac{T^{(m)}\ln(1+\delta)}{3\sigma^{(m)}}}, \quad \delta > 0, \tag{2.3.9}$$

where δ is small and $\sigma^{(m)}$ is the standard deviation of function values found during the m^{th} execution of step (ii) of Algorithm (2.3.2). If δ is sufficiently small, then succeeding Markov Chains have "nearby" stationary distributions, so the number of iterations required for each Metropolis algorithm to converge is small.

All of the previous schedules monotonically decrease the temperature. In some instances, however, it may be valuable to allow a non-monotonic schedule. Osman and Christofides [35] describe a "strategic oscillation" in which T is repeatedly decreased according to a geometric cooling schedule until progress halts, and then increased to half of its previous initial value.

As an example of how simulated annealing can be applied to a combinatorial optimization problem, let us consider the *traveling salesman* problem. This problem can be stated in the following way: a salesman leaves home, visits n different houses, and returns home. How can the salesman construct his route so that the distance he travels is minimized? In order to solve this problem using simulated annealing, we must appropriately define a state or configuration and a method for transitioning to a nearby state.

Each state can be considered as a possible path for the salesman to travel, and can be represented as a cyclic permutation vector. For example, if there are 5 houses, the state (1 3 4 5 2) represents a path from house 1 to house 3 to house 4 to house 5 to house 2 and back to house 1. In general, for n states, there are $(n\text{-}1)!/2$ cyclic permutations of these states, and hence $(n\text{-}1)!/2$ possible paths. Clearly, there are too many paths to perform a brute force search. We will define a nearby state as one that can be reached by a *2-change*. In a 2-change, a single element of the permutation vector is chosen, and the element directly to the right is exchanged with the element directly to the left.

Figure (2.3.2) shows the results of applying simulated annealing to the traveling salesman problem with n = 96 randomly chosen points. T was chosen according to a geometric cooling schedule with α = 0.98, and 500 iterations were performed. At each iteration, the Metropolis algorithm was iterated approximately 200 times. The plots in Figure (2.3.2) show samples of the initial path (upper left) and the paths after the 100^{th} (upper middle), 200^{th} (upper right), 300^{th} (lower left), 400^{th} (lower middle), and 500^{th} (lower left) iterations of a single simulated annealing run with an initial random path. Even though the solution is not necessarily the global minimum (we would have to iterate infinitely to guarantee this), it is a good solution. Upon performing simulated annealing five times for this particular problem, different optimal paths were found each time, but all of the solutions were close in total travel distance required. Out of the five trials, the best solution required a travel distance of 8.1914 units, whereas the worst required 8.4797 units.

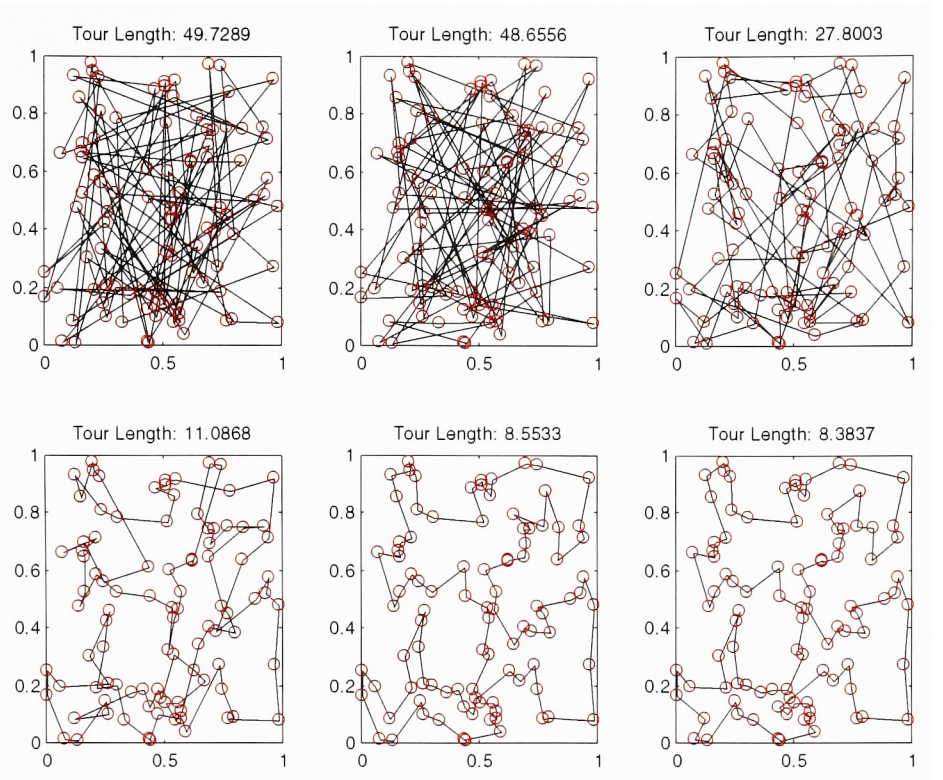

Figure (2.3.2): Paths produced by simulated annealing after the 0^{th} (upper left), 100^{th} (upper middle), 200^{th} (upper right), 300^{th} (lower left), 400^{th} (lower middle), and 500^{th} (lower right) iterations.

Simulated annealing can also be applied to continuous optimization problems. In order to extend the paradigm from combinatorial problems, we must appropriately define the generation mechanism and neighborhood in step (ii) of Algorithm (2.3.1). Corana, Marchesi, Martini, and Ridella [9] describe an algorithm where the neighborhood N_k is a hypercube centered at $\mathbf{x}^{(k)}$, and a candidate point $\mathbf{x}^{(k+1)}$ is chosen from a uniform distribution over N_k. If the candidate point is infeasible due to equality or inequality constraints, a new candidate point is chosen. In order to improve the efficiency and convergence properties of the algorithm, Corana et al. update the size of the hypercube at each iteration. If the ratio τ of the number of accepted moves to the number of rejected moves is large, the search is progressing too slowly, so the neighborhood size should be increased. If this ratio is small, computational effort is being wasted, so the neighborhood size should be decreased. Ideally, τ should be 0.5. A suggested scheme for updating the radius is given by:

$$r^{(k+1)} = \begin{cases} r^{(k)}\left(1+\beta\frac{5\tau-3}{2}\right), & \tau > 0.6 \\ \frac{r^{(k)}}{1+\beta\frac{2-5\tau}{2}}, & \tau < 0.4 \\ r^{(k)}, & \text{otherwise} \end{cases}, \tag{2.3.10}$$

where β is an experimentally determined constant.

The mechanism for generating iterates can also greatly influence the efficiency of simulated annealing. For example, if a minimum lies in a long, narrow valley, it is generally more fruitful to search

along the axis through the valley than normal to it. One way to control the direction of search is outlined in Vanderbilt and Louie [46]. They describe a scheme for generating steps according to a probability distribution whose covariance matrix is proportional to the inverse Hessian matrix. A growth factor similar to that of Corana et al. is also used to control the size of the search neighborhoods.

We will illustrate the use of simulated annealing for continuous optimization problems by minimizing the function (2.1.1). (We know that its global minimum is $\mathbf{x}^* = (0.5293, 0.7513)^T$). The cooling schedule used to solve this problem was geometric, with α = 0.98 and initial temperature T = 10. Simulated annealing was performed at 200 temperature levels; at each level, 100 points were randomly chosen and tested using the Metropolis criteria. Figure (2.3.3) shows the result of one simulated annealing run. The initial point is inside the thick circle, and the best point that was found (which is near the true optimum) is inside the thick square. All of the other points correspond to the best point found at each temperature level. Note that simulated annealing did a good job of exploring the entire feasible region before settling around a local minimum. In 100 similar runs, the solution yielded by simulated annealing was in the same basin of attraction as the global minimum 89 times.

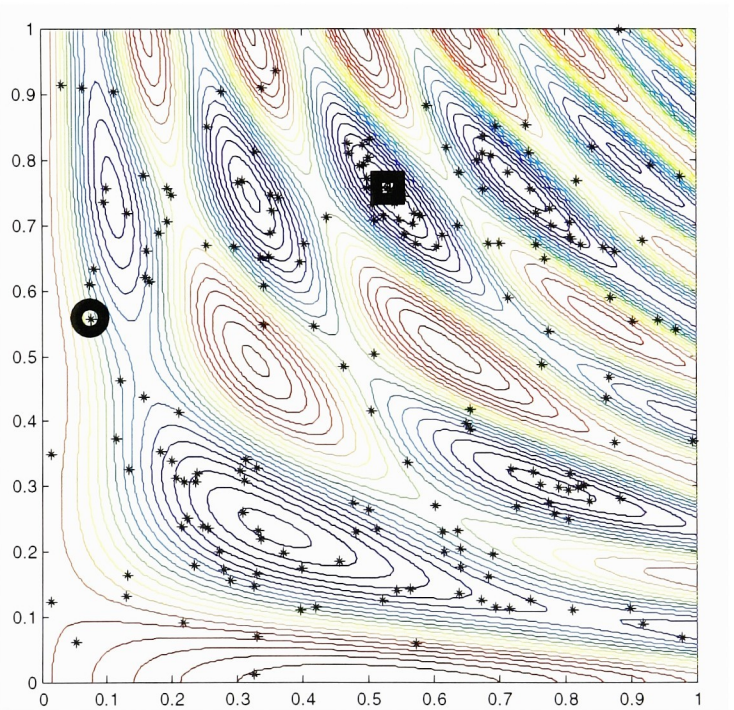

Figure (2.3.3): Simulated annealing applied to Equation (2.1.1). Initial point is in thick circle, solution is in thick square. Other points are best points at each temperature level.

There is one final note we need to make in regards to simulated annealing. Because of the stochastic nature of this algorithm, it can take a huge number of iterations to yield results of arbitrary precision. Therefore, it is recommended that the simulated annealing solution is used as the initial point in a local optimizer. This is a much simpler way of "fine-tuning" the result.

2.4 Genetic Algorithms

Genetic algorithms were introduced by John Holland [21] in order to mimic the theory of natural selection; that is, the development over time of a species adapting to its environment. In sexual reproduction, combinations of the genetic traits of parents are passed along to their offspring. The probability that the offspring will be viable (sexually reproduce) is dependent on the degree to which it can

adapt to its environment. Over generations, this dependence manifests itself in a population whose level of adaptation (or *fitness*) continually improves.

In order to understand the basic genetic algorithm, we will first introduce some terminology. A population of *chromosomes* (or vectors) is changed over time by sexual reproduction. Each chromosome contains a certain number of traits, or *genes*, and each gene can take on certain values, or *alleles*. The *fitness* of a chromosome can be evaluated by assessing its particular combination of genes. The mechanism of reproduction is *crossover*, where all the genes up to a random position are taken from one parent, and all the rest of the genes are taken from the other parent. Occasionally *mutations* take place, where one or more alleles are randomly perturbed.

The basic genetic algorithm updates a population of chromosomes by sexual reproduction. Initially, the fitness of every chromosome is determined; then, two chromosomes are selected to reproduce. Offspring are produced by applying the crossover mechanism; the offspring may then undergo some degree of mutation. Once the number of offspring produced is the same as the number of initial chromosomes, a new *generation* is formed. The new generation is treated as the initial population, and the process is repeated until an entire generation exhibits a predetermined level of fitness. Figure (2.4.1) illustrates the flow of a basic genetic algorithm.

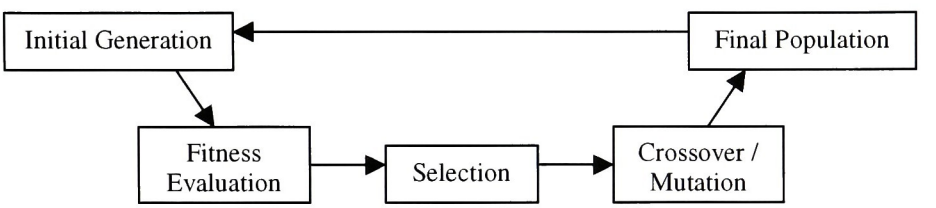

Figure (2.4.1): Basic genetic algorithm flowchart.

In the initial generation C, there are n individual chromosomes (typically anywhere from 50-200). Each chromosome can be represented as a vector of length m, where m is the number of genes. If $\mathbf{x} = (x_1, x_2, \ldots, x_m)^{\mathbf{T}}$ is a chromosome, then x_i is the particular allele of the i^{th} gene. We will define D_i as the set of possible alleles of the i^{th} gene, so that $x_i \in D_i$. We will also define $D = D_1 \oplus D_2 \oplus \cdots \oplus D_m$ as the set of all possible chromosomes. The fitness of a chromosome $\mathbf{x}$ is given by the scalar-valued function $f(\mathbf{x})$. In the selection step, the probability that a chromosome is chosen is determined by its fitness. Mathematically speaking, if $\mathbf{S}_1$ is a random variable representing the vector chosen for selection, the distribution of $\mathbf{S}_1$ is given by:

$$p_{\mathbf{S}_1}(\mathbf{x}) = \frac{f(\mathbf{x})}{\sum_{\mathbf{y} \in C} f(\mathbf{y})}. \tag{2.4.1}$$

Because two chromosomes are selected for reproduction, they can be chosen with replacement (using the distribution in (2.4.1)) or without replacement. If they are chosen without replacement, the first vector chosen is $\mathbf{S}_1$, and the second vector, say $\mathbf{S}_2$, has conditional distribution:

$$p_{\mathbf{S}_2|\mathbf{S}_1}(\mathbf{x}) = \frac{f(\mathbf{x})}{\sum_{\mathbf{y} \in C, \mathbf{y} \neq \mathbf{S}_1} f(\mathbf{y})}. \tag{2.4.2}$$

Once two chromosomes have been selected, the crossover mechanism is applied with probability p_c to yield two new chromosomes. In the simplest form of crossover, an element r is randomly chosen from $\{1,2,\ldots,m\}$ according to some discrete distribution $c(x)$ (usually uniform). A new chromosome $\mathbf{z}$ is formed from the first r genes from the first parent $\mathbf{x}$ and the last m-r genes from the second parent $\mathbf{y}$, and a new chromosome $\mathbf{w}$ is formed by the dual operation. Figure (2.4.2) illustrates the single-point crossover mechanism.

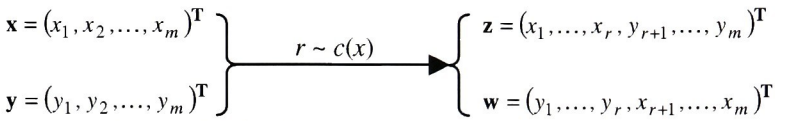

Figure (2.4.2): Single-point crossover mechanism.

In many instances it is helpful to apply the crossover mechanism at more than one point. An n-point crossover mechanism can be defined by choosing n distinct elements from $\{1,2,\ldots,m\}$, say r_1, r_2, …, r_n, and alternating copying genes from each parent at each index r_i. Figure (2.4.3) illustrates the n-point crossover mechanism.

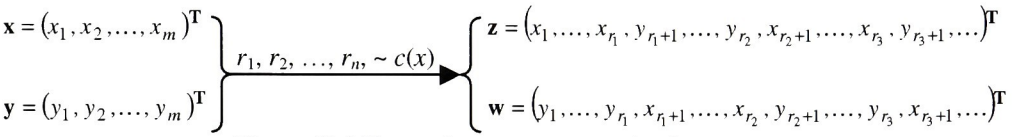

Figure (2.4.3): n-point crossover mechanism.

After a chromosome has been generated by crossover, each allele is mutated according to some probability distribution; that is, $z_i = \mathbf{Z}_i$, where $p_{\mathbf{Z}_i}(z)$ is the probability mass (or density) function of Z. In general, $p_{\mathbf{Z}_i}(z) << 1$ for $z \neq z_i$ and $p_{\mathbf{Z}_i}(z) = 0$ if $z \notin D_i$. In its simplest case (each chromosome is a binary vector), the mutation operator will change an element of the vector to its complement with small probability (generally $p_m = 0.001$ in practice).

In order to illustrate the use of genetic algorithms in global optimization, we will attack the bin-packing problem with a GA. In one of its incarnations (Mathworks [32]), it can be expressed in the following way: You are given a list of songs with their lengths, and the maximum length of a CD. You must choose which songs to place on the CD so as to minimize the "empty" time, or gap, at the end of the CD.

This problem can be solved by a brute-force search; however, this search is an exponential time algorithm (a list of q songs yields 2^q possible combinations). For even moderate values of n, a brute-force search is infeasible on today's processors. By attacking this problem with a genetic algorithm, we may not be able to find the exact solution, but we will easily be able to find "good" combinations of the songs (those combinations that have small gaps). Let us examine the following problem. Given a list of 30 songs, their lengths (in minutes) are given by:

i	1	2	3	4	5	6	7	8	9	10	11
l_i	1.0979	3.5540	1.8627	0.0849	3.2110	3.6466	4.8065	3.2717	0.1336	2.5008	0.4508
i	12	13	14	15	16	17	18	19	20	21	22
l_i	3.0700	3.1658	0.8683	3.5533	3.7528	2.7802	4.2016	1.6372	9.6254	1.3264	0.3160
i	23	24	25	26	27	28	29	30			
l_i	4.3212	3.0192	0.7744	2.3970	1.7416	2.4751	1.0470	1.9091			

We want to choose songs from this list to place on a CD that is 45 minutes long. By brute force, we find that if we eliminate songs 1, 7, 12, 16, 20, 23, 24, and 30 from the list, the rest of the songs fill 45 minutes exactly, leaving zero gap. This brute force technique is not very efficient, however; it took just over 2 hours to solve this problem on a 400MHz Pentium II.

To solve this problem with a genetic algorithm, we first need to define a chromosome in an appropriate context. We will define each chromosome $\mathbf{x}$ to be a binary vector of length 30, where $x_i = 0$ implies that song i is excluded from the CD, and $x_i = 1$ implies that song i is included on the CD. Given this definition, we can compute the gap $g(\mathbf{x})$ for any chromosome:

$$g(\mathbf{x}) = 45 - \sum_{i=1}^{30} x_i l_i \ . \tag{2.4.3}$$

In order to determine a fitness function $f(\mathbf{x})$ that will work well for this example, we will first describe some of its properties. We want $f(\mathbf{x})$ to take on its maximum value as $g(\mathbf{x})$ approaches zero. When $g(\mathbf{x})$ is less than zero, we have an unacceptable solution; however, the chromosome may have valuable segments that can be passed along to the next generation. Therefore, we define the fitness function to be a symmetric one as follows:

$$f(\mathbf{x}) = 1 - \left| \frac{g(\mathbf{x})}{g^*} \right|, \qquad g^* = \max_{\mathbf{y} \in D} |g(\mathbf{y})| \tag{2.4.4}$$

Initially choosing a random population of 50 chromosomes, a probability of crossover p_c = 0.7, and a probability of mutation p_m = 0.001, a typical GA run (for 200 generations) takes approximately 30 seconds on the same 400MHz Pentium II. The following figure shows the results of one such run. The top axis shows the maximum (blue x), median (green +), and average (red o) fitness values in each generation, and the bottom axis shows the gap of the chromosome of maximum fitness for each generation.

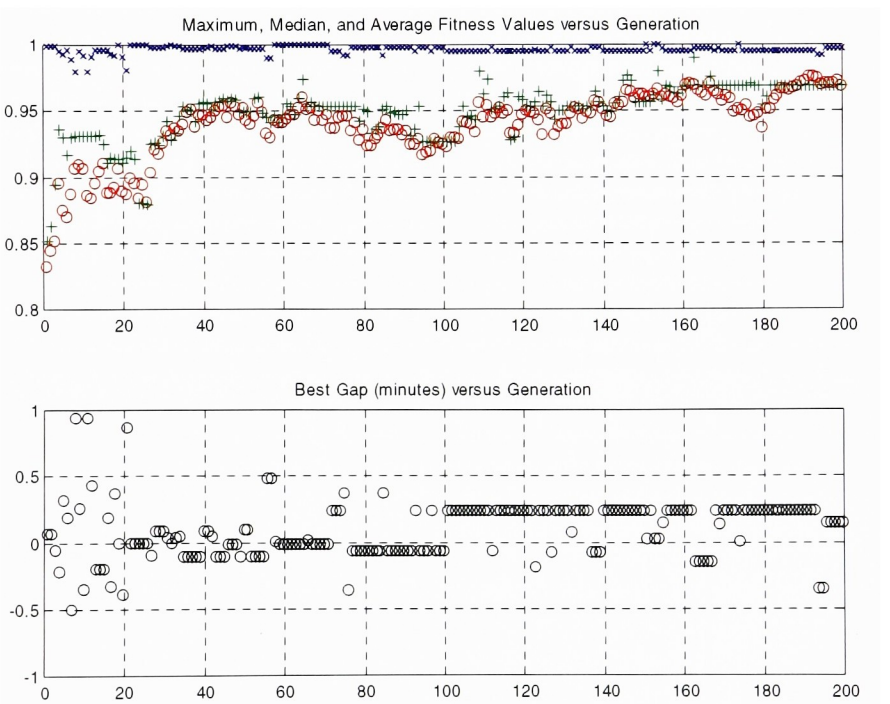

Figure (2.4.4): Top: fitness values in each generation; Bottom: best gap in each generation

Even though this run of the GA did not find the global solution, it found the vector **x** = 1000111001101010110100001110 10 in the 66th generation. Adding up the lengths of the songs according to this chromosome, we find that these songs span 44.9999 minutes, leaving a gap of 0.006 seconds on the CD (hardly annoying to the listener). When I performed 200 runs of this GA and plotted a histogram of the best gaps (in seconds), I found that the histogram could be modeled with an exponential distribution of rate $\lambda \approx 3.2054$. According to this model, the expected best gap for each run is $1/\lambda \approx 0.312$ seconds (the standard deviation is also $1/\lambda \approx 0.312$ seconds). The worst of the 200 runs yielded a solution with a 2.502 second gap, while three of the runs yielded the global solution of zero gap.

In a single run of the GA, 200 generations of 50 chromosomes requires 10000 gap evaluations. Considering that a brute force search could require $2^{30} \approx 10^9$ gap evaluations, the fact that our GA can find a solution with a gap of less than one third of a second on average while performing 10^5 times fewer gap evaluations is a tremendous feat!

If the GA does not yield satisfactory results, it can be "tweaked" in many ways. For example, experimenting with crossover and mutation probabilities may be fruitful. Multi-point crossover can be used instead of single-point crossover. A variety of other shapes of fitness functions can be examined that may yield smaller gaps in fewer generations. A cornucopia of possibilities exist; in general, it is sufficient to use a GA that yields good results without worrying about finding the optimal GA for a problem.

Genetic algorithms can also be somewhat effective when applied to continuous global optimization problems. In order to apply a GA to a function defined over a continuous domain, we must appropriately define chromosomes. This can be done, for example, by defining each chromosome to be a string (vector) composed of bit-string representations of each parameter. In order to minimize a function of two variables using a GA, a chromosome might be defined as a bit string with 64 elements; two concatenated 32-bit blocks form single precision floating point representations of the parameters.

When choosing a chromosome representation for a particular problem, it is important to pay attention to the effect of the crossover mechanism. If single-point crossover is used in the previously mentioned example, offspring chromosomes differ from their parents along only one coordinate direction. If the chromosome is defined by interleaving the two 32-bit strings instead of concatenating them, then offspring chromosomes will be located (in the two-coordinate space) extremely close to a parent chromosome because the "energy" distribution of each 32-bit string is not uniform (perturbing bits at the front of the string has a radical impact on the value of the variable, but perturbing bits at the end of the string has little impact). One way to overcome this problem is to modify the distribution used to generate crossover points; e.g., let $c(x)$ be geometric instead of discrete uniform. However, this may not be adequate. Even though choosing crossover points geometrically is an appropriate way to combat the non-uniform energy distribution of a bit string, we still expect that an offspring will differ from its parent in a given way half of the time. Therefore, binary representations of the parameters leave much to be desired.

Another way to define chromosomes is to discretize the coordinates and let a chromosome with k ones in the first m positions represent the k^{th} value of the first coordinate (the bits could be interleaved as well). Encoding chromosomes in this manner eliminates the problem of a non-uniform energy distribution; however, the number of bits required using this encoding scheme grows exponentially with respect to the number of bits required in the encoding scheme of the previous paragraph. Another drawback to this representation is that the mutation operator is insignificant; it only moves a chromosome to a neighboring bin.

Since the binary representation of parameter values yields the best representation in terms of length, the coordinate discretization yields the best representation in terms of energy uniformity, but both have significant drawbacks, better representations may be constructed by using ideas from each. For example, consider the following matrix:

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The matrix H can be thought of as a generator matrix for a (15,8) binary code. If each parameter is quantized and represented by an 8-bit integer, say $\mathbf{x}$, then a chromosome $\mathbf{y}$ representing that parameter can be constructed by solving $H\mathbf{y} = \mathbf{x}$ (there are $2^7 = 128$ possible solutions). Besides thinking about this representation in terms of coding theory, we can also recognize that each chromosome is a wavelet representation; each column of H is a shifted or scaled version of the Haar wavelet. Again, once each parameter is encoded as a 15-bit string, the bit strings can be concatenated or interleaved.

The number of bits required for the "Haar" chromosome grows linearly with the number required for a strict binary representation, so it is much better than the coordinate discretization method in that regard. Also, even though the energy distribution of the Haar chromosome is not uniform, it is much better that that of the chromosome with the strict binary representation.

To illustrate the solution of continuous global optimization problems with GA's, we will minimize the function (2.1.1). As mentioned in Section 2.1, the global minimum occurs at approximately $\mathbf{x}^* = (0.5293, 0.7513)^{\mathbf{T}}$. For this problem, we will construct the chromosomes by interleaving Haar representations of each parameter. Therefore, each chromosome will contain 30 bits, and we will be able to resolve points to within ~0.005. An initial population of 100 chromosomes will be iterated over 50 generations. Crossover will occur with probability $p_c = 0.7$ and mutation will occur with probability $p_m = 0.001$. Note again that the mutation operator in this case will generate new chromosomes which lie along one coordinate direction. Because of this, clusters of chromosomes will generally arise along one coordinate direction from other clusters, as can be seen in Figure (2.4.5):

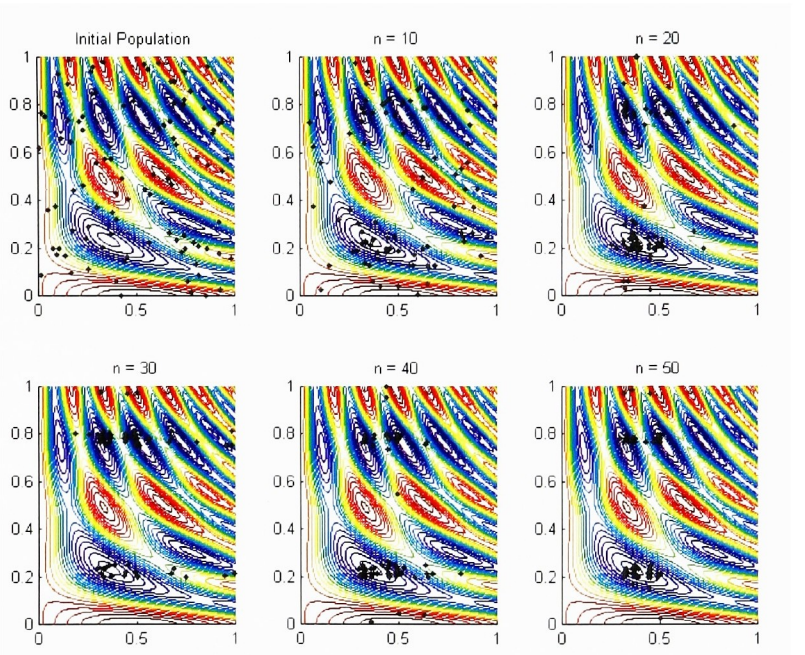

Figure (2.4.5): Genetic algorithm applied to $f(\mathbf{x})$; chromosomes at various generations

As is shown in Figure (2.4.6), we can use a genetic algorithm to find local maxima simply by negating the fitness function:

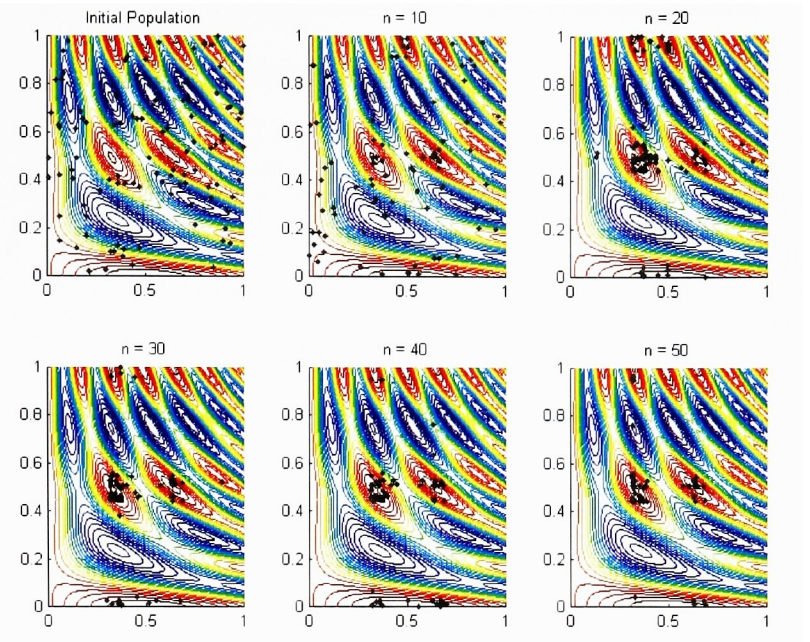

Figure (2.4.6): Genetic algorithm applied to $-f(\mathbf{x})$; chromosomes at various generations

As can be seen in the previous two figures, all chromosomes do not dwell near the global minimum. Rather, sub-populations emerge in the regions of attraction of some of the local minima. This suggests that in problems where the objective function is continuous, genetic algorithms could be used to make intelligent guesses at starting points for local optimizers (Newton/quasi-Newton/restricted step methods). Instead of randomly choosing n points from which to run local optimizers, feed those n points into a genetic algorithm. Iterate the genetic algorithm for a number of generations, throw away all points below the 90^{th} percentile in fitness value, and run local optimizers from the remaining points.

For any given optimization problem, there is no "black-box" solution offered by genetic algorithms. In some instances, determining a good combination of chromosome representation, selection mechanism, crossover mechanism, and mutation probability can be more difficult than the actual optimization the objective function! (Maybe a genetic algorithm is needed to determine the optimal parameters of the genetic algorithm!) The most advisable course of action to take is to use experience as a guide; if faced with a problem similar in nature to one that has already been solved by a genetic algorithm, it cannot hurt to attack the new problem in a similar way.

Chapter 3: Deterministic Methods for Global Optimization

3.1 Integer/Mixed Integer Programming

Deterministic algorithms for global optimization of combinatorial problems are scarce. One could construct methods based on simulated annealing or tabu search that analyze different areas of the feasible region for a particular problem; however, constructing a similar general purpose deterministic method has not been done. In general, Monte Carlo methods are much preferred for combinatorial problems. One useful deterministic method does exist, though. The branch and bound algorithm can be used for a specific class of problems; namely, those that can be posed as *integer programming* problems (Beale [3]). Integer programming problems are really continuous problems of the form (1.2.1) with the additional constraint that $x_i \in \mathbf{Z} \quad \forall i$, where $\mathbf{Z} = \{0,\pm1,\pm2,\ldots\}$. If this constraint does not necessarily hold for all i, but only for some i, then the problem is called a *mixed integer programming* problem.

Many combinatorial optimization problems can be posed as mixed integer programming problems by mapping each configuration in the combinatorial problem to a set of integer variables. A smooth (at least twice differentiable) objective function must then be constructed so that at integer coordinates, it takes on values corresponding those of the original objective function from the combinatorial problem. In order to illustrate the branch and bound technique, we will assume that the combinatorial optimization problem has been posed as a mixed integer program (MIP), even though we recognize that this can be an extremely difficult task.

We will restate the MIP as a continuous optimization problem P_I with constraints:

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & c_i(\mathbf{x}) = 0\,, & i \in E \\ & c_i(\mathbf{x}) \geq 0\,, & i \in I \\ & x_i \text{ integer.} & i \in U \end{array} \tag{3.1.1}$$

In the branch and bound method, initially, the integer constraints are removed from (3.1.1) and the resulting constrained optimization problem P is solved, yielding a solution $\mathbf{x}'$. If x_i' is integer valued for all $i \in U$, then $\mathbf{x}'$ solves (3.1.1). If not, then there must exist some $j \in U$ such that x_j' is not integer valued, so we branch (form two sub-problems to solve) at x_j'. The two sub-problems we must solve are both MIP's as well. P_- is given by:

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & c_i(\mathbf{x}) = 0\,, & i \in E \\ & c_i(\mathbf{x}) \geq 0\,, & i \in I \\ & x_j \leq \lfloor x_j' \rfloor, & \\ & x_i \text{ integer,} & i \in U \end{array} \tag{3.1.2}$$

where $\lfloor y \rfloor$ denotes the largest integer less than or equal to y. P_+ is given by:

$$\begin{array}{lll} \text{minimize} & f(\mathbf{x}) & \\ \text{subject to} & c_i(\mathbf{x}) = 0\,, & i \in E \\ & c_i(\mathbf{x}) \geq 0\,, & i \in I \\ & x_j \geq \lfloor x_j' \rfloor + 1\,, & \\ & x_i \text{ integer.} & i \in U \end{array} \tag{3.1.3}$$

Since both sub-problems are MIP's, the same method applied to (3.1.1) is used to solve (3.1.2) and (3.1.3). It is easily seen that this process iteratively branches out, generating many sub-problems that must be solved. Figure (3.1.1) shows a tree structure that illustrates the branching process.

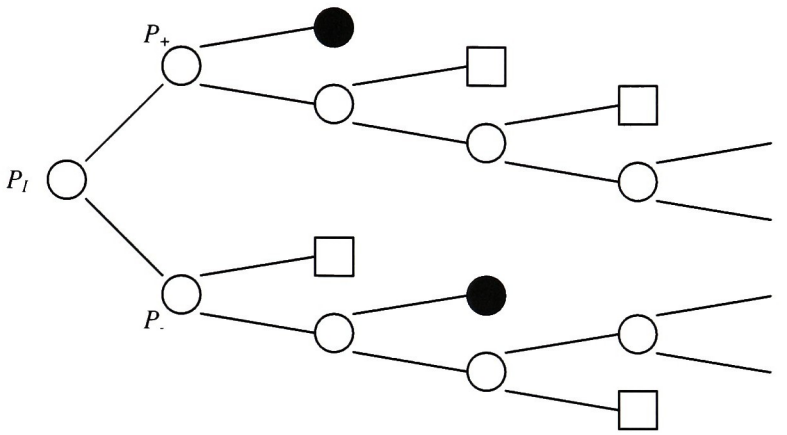

Figure (3.1.1): Tree structure of branch and bound algorithm.

Each ○ represents a branch point, or *node*. If one of the sub-problems generated at a particular node is infeasible, it is denoted by ●. If the solution to the sub-problem (without the integer constraints) satisfies the integer constraints, it is denoted by □.

Even if the MIP is bounded, the size of the tree of Figure (3.1.1) can still be infinite. Also, it can take an exhorbitant amount of time to construct nodes of the tree because each node requires the execution of a minimization routine. The "bound" portion of the branch and bound method allows us to get away with only partially constructing the tree. The key fact that allows us to do this is that the optimal function value at a node serves as a lower bound for the optimal values at each of its sub-nodes. Therefore, if we keep track of the best optimal value at the □ nodes that have already been constructed, we can eliminate further branching at any ○ nodes whose optimal values are greater than that value.

As an example, we will solve the problem

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) = x_1^4 + x_2^4 + 16\left(x_1 x_2 + (4 + x_2)^2\right) \\ &\text{subject to} && x_1 \text{ , } x_2 \text{ integer} \end{aligned} \tag{3.1.4}$$

using the branch and bound method (Chapter 13 Exercises, Fletcher [15]). The solution to the unconstrained problem is $\mathbf{x}' = (2.2062, -2.6846)^{\mathbf{T}}$, so we will begin by branching on x_1. First we will examine the sub-problem

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) = x_1^4 + x_2^4 + 16\left(x_1 x_2 + (4 + x_2)^2\right) \\ &\text{subject to} && x_1 \le 2 \\ & && x_1 \text{ , } x_2 \text{ integer} \end{aligned} \tag{3.1.5}$$

The solution to (3.1.5) without the integer constraints is $\mathbf{x}' = (2, -2.6565)^{\mathbf{T}}$. If we now branch on x_2, we have the further sub-problem

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) = x_1^4 + x_2^4 + 16\left(x_1 x_2 + (4 + x_2)^2\right) \\ &\text{subject to} && x_1 \le 2 \text{ , } x_2 \le -3 \\ & && x_1 \text{ , } x_2 \text{ integer} \end{aligned} \tag{3.1.6}$$

The non-integer constrained solution to (3.1.6) is $\mathbf{x}' = (2, -3)^{\mathbf{T}}$ with $f(\mathbf{x}') = 17$, and this is integer feasible. Going back up the tree, we will now solve the other sub-problem of (3.1.5), namely,

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) = x_1^4 + x_2^4 + 16\left(x_1 x_2 + (4 + x_2)^2\right) \end{aligned} \tag{3.1.7}$$

subject to $x_1 \le 2\,,\ x_2 \ge -2$

$x_1\,,\ x_2$ integer

The non-integer constrained solution to (3.1.7) is $\mathbf{x}' = (2,-2)^{\mathbf{T}}$ with $f(\mathbf{x}') = 32$, and this is integer feasible. We can now go two steps up the tree and solve the other sub-problem of (3.1.4), which is

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}) = x_1^4 + x_2^4 + 16\left(x_1 x_2 + (4 + x_2)^2\right) \\ &\text{subject to} && x_1 \ge 3 \\ & && x_1\,,\ x_2 \text{ integer} \end{aligned} \tag{3.1.8}$$

The non-integer constrained solution to (3.1.8) is $\mathbf{x}' = (3,-2.7888)^{\mathbf{T}}$ with $f(\mathbf{x}') = 31.0977$. Since the best optimal value so far is 17 (at $(2,-3)^{\mathbf{T}}$), we know that none of the sub-problems of (3.1.8) can do better than this, so we can terminate our search. The solution to (3.1.4) is therefore $\mathbf{x}' = (2,-3)^{\mathbf{T}}$.

The tree structure of this problem can be seen in Figure (3.1.2):

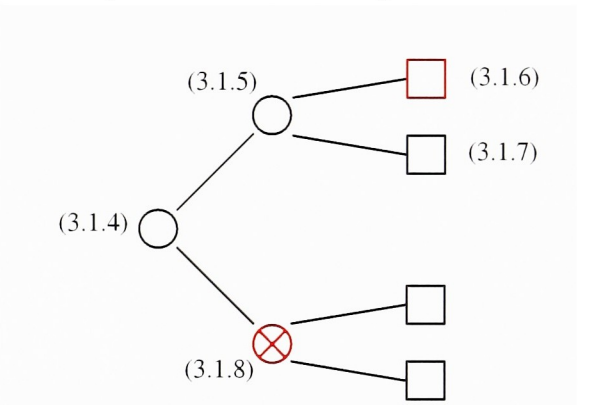

Figure (3.1.2): Tree structure of branch and bound method applied to (3.1.4).

The red □ indicates the sub-problem that yielded the global minimum, and the red ⊗ indicates the start of a path we did not need to search because the lower bound was higher than the best previously reported integer feasible minimum. Its sub-nodes are shown so that we can see the entire tree for this problem.

Aside of the difficulties in casting a combinatorial optimization problem as a MIP, the largest drawback of the branch and bound method is probably that it requires finding the global optimum of each sub-problem. This is straightforward if the objective function is simple (linear or quadratic). If the objective function is nonlinear, each sub-problem must be globally optimized, for instance, by one of the Monte Carlo methods of the previous chapter or one of the deterministic methods of the next two sections. Therefore, unless a simple objective function can be constructed, this author recommends a Monte Carlo approach to global optimization for combinatorial problems.

3.2 Unconstrained Optimization: Transformation Methods

Any deterministic approach to global optimization of a continuous problem undoubtedly entails iteratively finding and escaping local minima until the global minimum is found. For unconstrained problems (or bound constraints that are not strongly active at any local minimum), *transformation* methods have emerged. In these methods, when a local minimum is found, the objective function is transformed so that the current solution is no longer a local minimum. The transformations allow an easy way to search for the global minimum by "hopping" or "tunneling" from local minimum to local minimum.

The first transformation method to appear in the literature was the *tunneling method* (Levy and Gomez [30]) for optimization of functions of a single variable, which places poles at local minima. Later on, Cetin, Barhen, and Burdick [8] introduced terminal repeller unconstrained sub-energy tunneling (*TRUST*), which transforms the objective function by flattening it in areas where the function value is larger

than that of the current minimum and by adding a repeller term. Both of these methods are only guaranteed to yield a global minimum for univariate functions; however, they can be extended to the multivariate case (but with less than desirable characteristics, in the opinion of this author). A third transformation method, *basin hopping* (Wales and Doye [44]), has been developed to find low energy configurations of macro-molecular systems. Basin hopping transforms the objective function by making it constant in the neighborhood of a local minimum. This approach is questionable, as it introduces discontinuities in the objective function, rendering many local optimization methods useless. The remainder of this section will describe the tunneling and TRUST methods in further detail.

Levy and Gomez [30] introduced the tunneling method as an iterative two-phase approach to finding the global minimum of a function of one variable. The first phase is the minimization phase, where a local minimum, say $\mathbf{x}^*$, is found. Then the second phase, the tunneling phase, searches for zeros of the tunneling function

$$T(\mathbf{x},\mathbf{x}^*)= f(\mathbf{x})- f(\mathbf{x}^*), \tag{3.2.1}$$

where $f(\mathbf{x})$ is the objective function. If a zero is found, say $\mathbf{x}'$, and $\mathbf{x}' \neq \mathbf{x}^*$, then the two-phase approach is performed again starting at $\mathbf{x}'$. If no such zero can be found, then $\mathbf{x}^*$ is the global minimum. Graphically, it can be seen in Figure (3.2.1) that when a local minimum is found, this algorithm tunnels under other areas of the function until a zero of (3.2.1) is found.

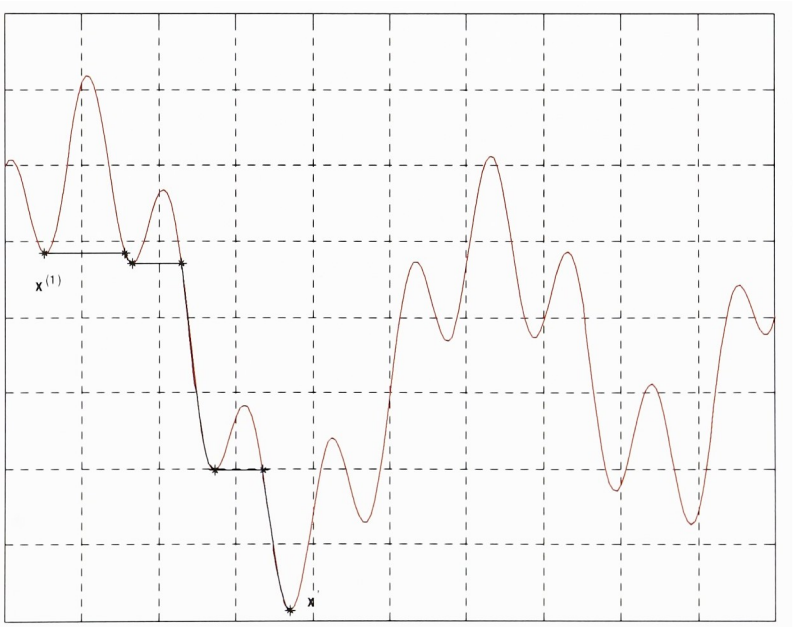

Figure (3.2.1): Example of minimization and tunneling phases of the tunneling algorithm.

The problem with the tunneling phase is that $\mathbf{x}^*$ is a zero of (3.2.1), so it may be difficult for rootfinding software to locate a zero $\mathbf{x}' \neq \mathbf{x}^*$. In order to remedy this situation, a pole is added to remove the zero at $\mathbf{x}^*$. This can be done by changing the tunneling function to

$$T(\mathbf{x},\mathbf{x}^*,\alpha,\lambda)= \frac{f(\mathbf{x})- f(\mathbf{x}^*)}{\alpha\left[(\mathbf{x}-\mathbf{x}^*)^T(\mathbf{x}-\mathbf{x}^*)\right]^\lambda}, \tag{3.2.2}$$

where α and λ are constant scalars that affect the strength of the pole. To illustrate this graphically, we show a function on the left side of Figure (3.2.2) with a local minimum denoted by the black asterisk. On the right side of Figure (3.2.2) is the original function (red dotted line) and a constant offset of the tunneling function (3.2.2) (red solid line). The black dotted line denotes the position of the pole, and the blue dotted line is the zero line of the tunneling function. We can see $\mathbf{x}^*$ is not a zero of (3.2.2), and any point that is a zero of (3.2.2) will allow us to find a better local minimum.

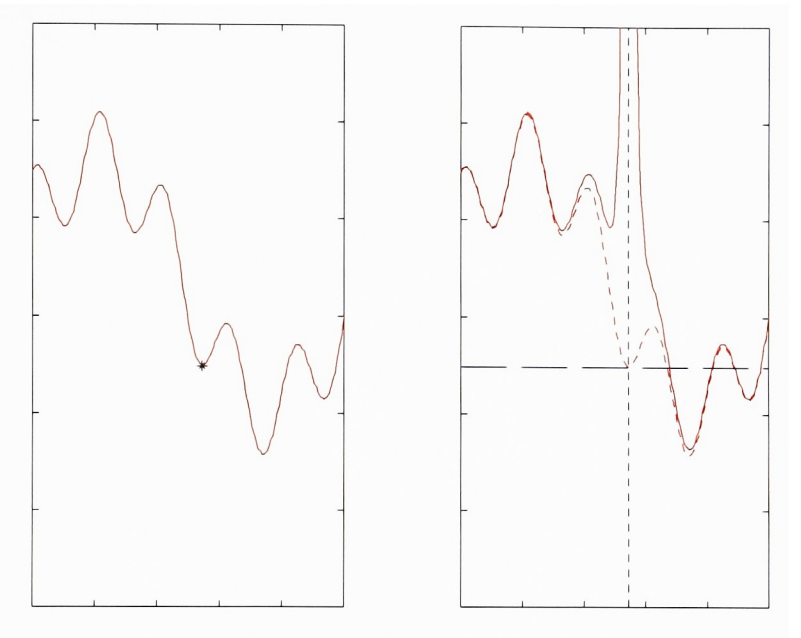

Figure (3.2.2): Original function (left); pole introduced (right)

The parameters α and λ are somewhat arbitrary; they must remove the zero of the tunneling function at $\mathbf{x}^*$, but they must not distort the function so much that other local minima are removed as well. In practice, it may be necessary to execute the tunneling phase at various combinations of the values of α and λ, hindering the efficiency of the tunneling method. Another problem is that if more than one local minima have the same function value, it may be necessary to introduce multiple poles.

Cetin, Barhen, and Burdick's [8] TRUST method for global optimization attempts to alleviate these problems by introducing a different type of transformation. Their approach still entails a local minimization phase and a tunneling phase. Instead of introducing poles at local minima, they transform the objective function by flattening it in areas where the function value exceeds that at the local minimum, and by introducing a terminal repeller to force the next local minimizer away from the current iterate. The transformation is given by

$$T(\mathbf{x},\mathbf{x}^*,\alpha,\lambda) = T_{sub}(\mathbf{x},\mathbf{x}^*,\alpha) - \lambda T_{rep}(\mathbf{x},\mathbf{x}^*)\mu(f(\mathbf{x}) - f(\mathbf{x}^*)), \tag{3.2.3}$$

where the sub-energy function T_{sub} is given by

$$T_{sub}(\mathbf{x},\mathbf{x}^*,\alpha) = \log\left(\frac{1}{1+e^{-(f(\mathbf{x})-f(\mathbf{x}^*)+\alpha)}}\right), \tag{3.2.4}$$

the repeller function T_{rep} is

$$T_{rep}(\mathbf{x},\mathbf{x}^*)=\tfrac{3}{4}(\mathbf{x}-\mathbf{x}^*)^{1/3}, \tag{3.2.5}$$

u is the unit step function, and α and λ are constant scalars. Cetin et al. show that for most univariate problems, suitable choices for α and λ are $\alpha = 2$ and $\lambda > \left[1/(\mathbf{x}-\mathbf{x}^*)^{1/3}\right](\partial f(\mathbf{x})/\partial \mathbf{x})(1/[1+\exp(\alpha)])$ for all $\mathbf{x}$. The sub-energy function preserves location and ordering of local minima, and the repeller function ensures that any local minimization routine will escape from the current local minimum. Figure (3.2.3) illustrates the transformation induced by the sub-energy function. The initial local minimum is denoted by the black asterisk. Note that the sub-energy function (blue dashed line) is flat in areas where the objective function (red solid line) is greater than the value at that minimum.

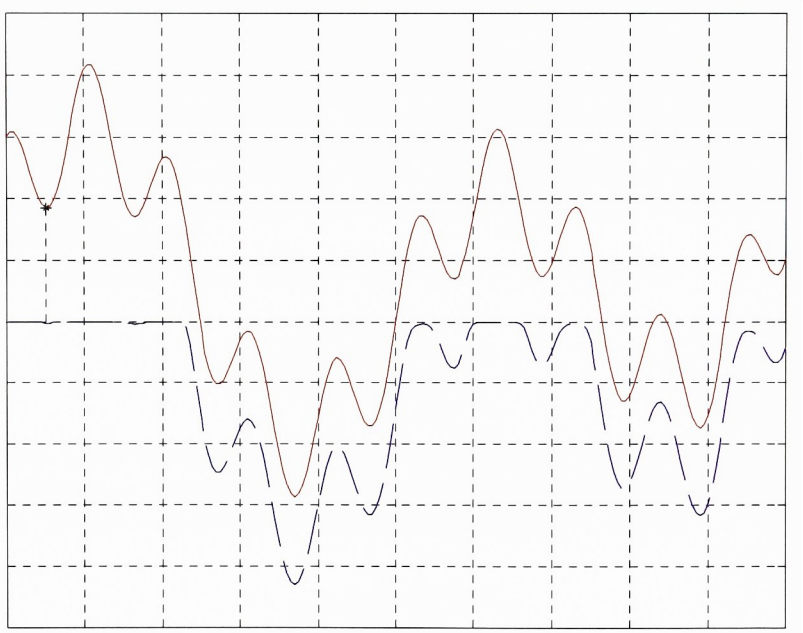

Figure (3.2.3): Objective function (red solid line); sub-energy function (blue dashed line)

One problem with the TRUST method is that the flatness induced by the sub-energy function depends on the scale of the objective function. This can be simply remedied by adding a scale parameter in the exponential portion of (3.2.4); however, choosing an appropriate scaling parameter may be difficult. Another problem with this method is that in order for convergence to be guaranteed, the search for the global minimum must "flow" in a single direction. In other words, TRUST should be started at the lower bound, and each local minimum that is found should be located to the right of the previous minimum. The problem arises in extending TRUST to multiple dimensions. One proposed method is to redefine the repeller function as:

$$T_{rep}(\mathbf{x},\mathbf{x}^*)=\tfrac{3}{4}\sum_{i=1}^{n}(x_i-x_i^*)^{1/3}, \tag{3.2.6}$$

and carry out TRUST in the same was as the univariate case. Because of the restriction of maintaining a positive (or negative) flow, it is highly likely that the global minimum will be missed if it does not lie near the main diagonal of the hyperparallelpiped defining the lower and upper bounds of the variables. Another

proposed method, called *hyperspiral embedding*, attempts to solve the multivariate global optimization problem by generating a one-dimensional path that adequately covers the high-dimensional space, and carrying out the univariate TRUST algorithm along this path. This takes the ad-hoc cake!

The idea of transforming an objective function to temporarily remove local minima while searching for a global minimum is a good one; however, none of the transformation algorithms published to date are described in general enough detail to apply to problems of arbitrary scale and dimension without much ado. As echoed in the previous section, this author recommends a Monte Carlo approach above any current transformation technique, because Monte Carlo methods can be more easily applied to arbitrary problems.

3.3 Constrained Optimization: Path Following

One final global optimization approach worth mentioning is the *homotopy*, or *path following*, approach. Path following has been used in other types of problems, such as ordinary differential equations, integral equations, matrix eigenvalue problems, and determinant evaluations. All of these problems are solved by identifying the solution of a nearby simple problem, then "bending" the nearby problem and solution into the problem at hand. Garcia and Zangwill [17] describe how the homotopy approach can be used to solve nonlinear systems of equations. They further explain how this method can be extended to certain constrained optimization problems by casting the Kuhn-Tucker first-order conditions (Section 1.2) as a system of nonlinear equations to be solved. This approach is valuable in that it can identify multiple local minima if done properly. The remainder of this section gives a brief overview of path following as presented by Garcia and Zangwill.

Consider the nonlinear system $\mathbf{f}(\mathbf{x})=\mathbf{0}$, where $\mathbf{f}:\Re^2 \to \Re^2$ is given by:

$$\mathbf{f}(\mathbf{x})=\begin{pmatrix} x_1^3 - 3x_1^2 + 8x_1 + 3x_2 - 36 \\ x_1^2 + x_2 + 4 \end{pmatrix}. \tag{3.3.1}$$

The solution to $\mathbf{f}(\mathbf{x})=\mathbf{0}$ can be computed numerically by the Newton-Raphson method (more generally, the Gauss-Newton method, see Fletcher [15]), but we can use a homotopy approach to come up with an analytical solution. We can construct another nonlinear system $\mathbf{g}(\mathbf{x})=\mathbf{0}$, where

$$\mathbf{g}(\mathbf{x})=\begin{pmatrix} x_1^3 + 8x_1 + 3x_2 \\ x_2 \end{pmatrix}, \tag{3.3.2}$$

and note that the only real solution to this "simple" system is $\mathbf{x}^* = \mathbf{0}$. We now construct a homotopy function $\mathbf{h}(\mathbf{x},t)$ in the following manner:

$$\mathbf{h}(\mathbf{x},t)=t\mathbf{f}(\mathbf{x})+(1-t)\mathbf{g}(\mathbf{x}). \tag{3.3.3}$$

This is known as the *linear homotopy*, because it is a linear combination of the functions $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$. As t increases from 0 to 1, the system $\mathbf{h}(\mathbf{x},t)=\mathbf{0}$ changes, or bends, from $\mathbf{g}(\mathbf{x})=\mathbf{0}$ to $\mathbf{f}(\mathbf{x})=\mathbf{0}$. If we say that $\mathbf{x}(t)$ is the solution to $\mathbf{h}(\mathbf{x},t)=\mathbf{0}$, then we can find the solution to $\mathbf{f}(\mathbf{x})=\mathbf{0}$ by starting at $\mathbf{x}(0)=\mathbf{x}^*$ and following the path $\mathbf{x}(t)$ until $t=1$. For $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ given by (3.3.1) and (3.3.2), we can easily determine $\mathbf{x}(t)$. From (3.3.3), we have

$$x_1^3 + 8x_1 + 3x_2 - t\left(3x_1^2 + 36\right)= 0 \tag{3.3.4}$$

$$x_2 + t\left(x_1^2 + 4\right)= 0 \tag{3.3.5}$$

Substituting for x_2 into (3.3.4) and factoring, we have

$$(x_1 - 6t)(x_1^2 + 8) = 0, \tag{3.3.6}$$

so the path is given by $\mathbf{x}(t) = (6t, -36t^3 - 4t)^{\mathbf{T}}$. Then $\mathbf{x}(1) = (6, -40)^{\mathbf{T}}$ is the solution to our original system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$.

The previous example was obviously picked because it is easy to determine $\mathbf{x}(t)$ in closed form; however, this will rarely be possible. In general, the path can be determined by solving a system of first order differential equations known as the *basic differential equations* (BDE) with respect to the path length p. The BDE are given by:

$$\begin{aligned} \dot{x}_i &\equiv \frac{\partial x_i}{\partial p} = (-1)^i \det \nabla \mathbf{h}_{-i}, \quad i = 1, \ldots, n \\ \dot{t} &\equiv \frac{\partial t}{\partial p} = (-1)^{n+1} \det \nabla \mathbf{h}_{-(n+1)} \end{aligned}, \tag{3.3.7}$$

where $\nabla \mathbf{h}_{-j}$ is the nxn matrix formed by removing the j^{th} row of the Jacobian matrix of $\mathbf{h}(\mathbf{x}, t)$.

To give an example of how the BDE are used to solve a system of nonlinear equations, consider the system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, where $\mathbf{f} : \Re^2 \to \Re^2$ is given by:

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} x_1^2 + x_1 x_2 - 3 \\ 2\sin x_1 + x_2 - 7 \end{pmatrix}. \tag{3.3.8}$$

We will use the Newton homotopy to solve this problem. The Newton homotopy is given by $\mathbf{h}(\mathbf{x}, t) = \mathbf{f}(\mathbf{x}) - (1-t)\mathbf{f}(\mathbf{x}')$, where $\mathbf{x}'$ can be any initial vector. (For the Newton homotopy, the "simple" nearby system $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ is given by $\mathbf{g}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}')$, and it has solution $\mathbf{x}'$). Choosing $\mathbf{x}' = \mathbf{0}$, we have:

$$\mathbf{h}(\mathbf{x}, t) = \begin{pmatrix} x_1^2 + x_1 x_2 - 3t \\ 2\sin x_1 + x_2 - 7t \end{pmatrix}. \tag{3.3.9}$$

The Jacobian matrix of $\mathbf{h}(\mathbf{x}, t)$ is:

$$\nabla \mathbf{h}(\mathbf{x}, t) = \begin{pmatrix} 2x_1 + x_2 & 2\cos x_1 \\ x_1 & 1 \\ -3 & -7 \end{pmatrix}. \tag{3.3.10}$$

Computing determinants and substituting into (3.3.7), we now have the BDE for this problem:

$$\begin{aligned} \dot{x}_1 &= 7x_1 - 3 \\ \dot{x}_2 &= -14x_1 - 7x_2 + 6\cos x_1 \\ \dot{t} &= -2x_1 - x_2 + 2x_1 \cos x_1 \end{aligned}. \tag{3.3.11}$$

Since we have chosen $\mathbf{x}' = \mathbf{0}$ at $t = 0$, the initial conditions for the BDE are $x_1(0) = x_2(0) = t(0) = 0$. If we solve this system of differential equations numerically, we find that $t = 1$ approximately when $x_1 = -8.6884$ and $x_2 = 8.3432$, so a solution to $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ is $\mathbf{x}^* = (-8.6884, 8.3432)^{\mathbf{T}}$. The following figure shows the path $\mathbf{x}(t)$ given by the solution to the BDE. The circle denotes the initial point $\mathbf{x}(0)$, and the asterisk is the solution $\mathbf{x}^* = \mathbf{x}(1)$.

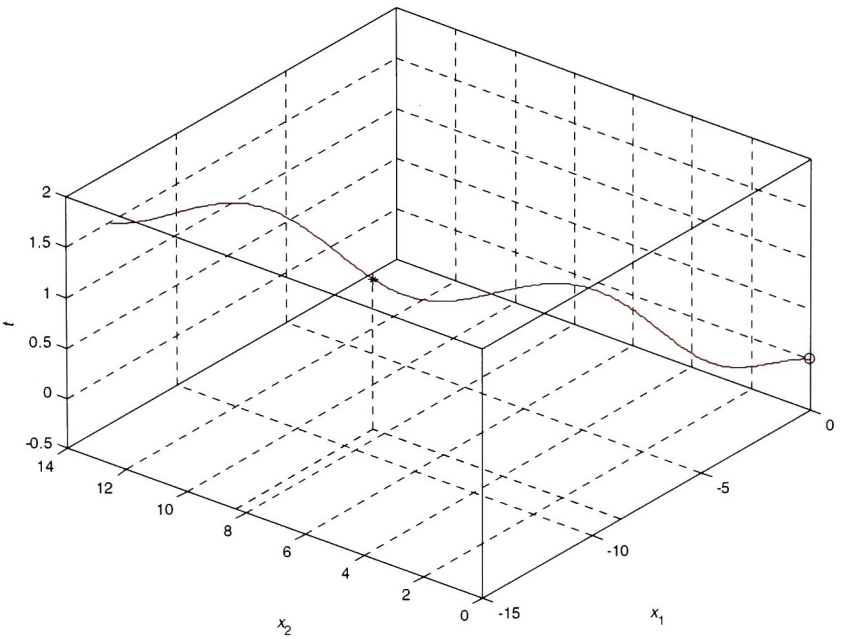

Figure (3.3.1): Path to the solution of $\mathbf{f}(\mathbf{x})=\mathbf{0}$, found by solution of BDE.

This method of contructing the Newton homotopy and solving the BDE is useful when our goal is to find any solution to $\mathbf{f}(\mathbf{x})=\mathbf{0}$, but we are more interested in finding all possible solutions. If path-following is to be applied to constrained optimization problems, we would like it to identify *all* points that satisfy the Kuhn-Tucker conditions. One way of approaching this is to perform some sort of Monte Carlo scheme (like that described in section 2.1) to generate a collection of points, treat each point as a solution to a nearby system $\mathbf{g}(\mathbf{x})=\mathbf{0}$, and follow every path. Another way of generating all solutions is to use a different homotopy that allows the paths to be complex.

To solve a given system $\mathbf{f}(\mathbf{x})=\mathbf{0}$, where $\mathbf{f}$ and $\mathbf{x}$ are real-valued, we must first consider the system $\mathbf{f}(\mathbf{z})=\mathbf{0}$, where $\mathbf{z}$, and hence $\mathbf{f}$, are complex. We then construct the All-Solutions homotopy $\mathbf{h}(\mathbf{z},t)$, where the k^{th} element is given by

$$h_k(\mathbf{z},t)=(1-t)\left(z_k^{q_k}-1\right)+tf_k(\mathbf{z}). \tag{3.3.12}$$

Each q_k is a positive integer, chosen so that if $f_k(\mathbf{z})=z_k^{q_k}+\phi_k(\mathbf{z})$, then $\lim_{\|z_k\|\to\infty}\phi_k(\mathbf{z})/z_k^{q_k}=0$ and $\lim_{\|z_k\|\to\infty}f_k(\mathbf{z})/\left(z_k^{q_k}-1\right)$ is not pure real and negative. The nearby "simple" problem $\mathbf{h}(\mathbf{z},0)=0$ has $\prod_k q_k$ solutions; the j^{th} entry of each solution is one of the q_j^{th} roots of unity. Once the All-Solutions homotopy has been constructed, we construct an equivalent homotopy $\hat{\mathbf{h}}(\mathbf{w},t)$, where

$$\begin{aligned}\hat{h}_{2k-1}(\mathbf{w},t)&=\mathrm{Re}(h_k(\mathbf{z},t))\\ \hat{h}_{2k}(\mathbf{w},t)&=\mathrm{Im}(h_k(\mathbf{z},t))\end{aligned}, \tag{3.3.13}$$

and $\mathbf{w}$ is given by

$$\begin{aligned}w_{2k-1}&=\mathrm{Re}(z_k)\\ w_{2k}&=\mathrm{Im}(z_k)\end{aligned}. \tag{3.3.14}$$

Now if $\mathbf{h}(\mathbf{z},t)=\mathbf{0}$ is a length n complex system of $n+1$ complex variables, then $\hat{\mathbf{h}}(\mathbf{w},t)=\mathbf{0}$ is an equivalent length $2n$ real system of $2n+1$ real variables. If we start at each of the $\prod_k q_k$ solutions to $\mathbf{h}(\mathbf{z},0)=0$ (cast into the form of (3.3.14)) and follow the paths given by the BDE (3.3.7) applied to the homotopy $\hat{\mathbf{h}}(\mathbf{w},t)$, we will identify all solutions to the original system $\mathbf{f}(\mathbf{x})=\mathbf{0}$.

To illustrate this process, let us again observe the system $\mathbf{f}(\mathbf{x})=\mathbf{0}$ given by (3.3.8). The All-Solutions homotopy for this problem (with $q_1 = 2$ and $q_2 = 1$) is

$$\mathbf{h}(\mathbf{z},t)=\begin{pmatrix}(1-t)(z_1^2-1)+t(z_1^2+z_1z_2-3)\\(1-t)(z_2-1)+t(2\sin z_1+z_2-7)\end{pmatrix}. \tag{3.3.15}$$

The system $\mathbf{h}(\mathbf{z},0)=0$ has two solutions; namely, $(1,1)^{\mathbf{T}}$ and $(-1,1)^{\mathbf{T}}$. We can now define $\mathbf{w}$ so that $z_1 = w_1 + w_2 i$ and $z_2 = w_3 + w_4 i$. Equation (3.3.15) then becomes

$$\mathbf{h}(\mathbf{w},t)=\begin{pmatrix}(1-t)((w_1+w_2i)^2-1)+t((w_1+w_2i)^2+(w_1+w_2i)(w_3+w_4i)-3)\\(1-t)((w_3+w_4i)-1)+t(2\sin(w_1+w_2i)+(w_3+w_4i)-7)\end{pmatrix}. \tag{3.3.16}$$

Separating the real and imaginary parts, we can construct the equivalent homotopy (3.3.13):

$$\hat{\mathbf{h}}(\mathbf{w},t)=\begin{pmatrix}w_1^2-w_2^2-1+t(w_1w_3-w_2w_4-2)\\2w_1w_2+t(w_1w_4+w_2w_3)\\w_3-6t-1+t(e^{-w_2}+e^{w_2})\sin w_1\\w_4-t(e^{-w_2}-e^{w_2})\cos w_1\end{pmatrix}. \tag{3.3.17}$$

The solutions to $\hat{\mathbf{h}}(\mathbf{w},0)=\mathbf{0}$ are then given by $(1,0,1,0)^{\mathbf{T}}$ and $(-1,0,1,0)^{\mathbf{T}}$. The Jacobian matrix of $\hat{\mathbf{h}}(\mathbf{w},t)$ is:

$$\nabla\hat{\mathbf{h}}(\mathbf{w},t)=\begin{pmatrix}2w_1+tw_3 & 2w_2+tw_4 & t(e^{-w_2}+e^{w_2})\cos w_1 & t(e^{-w_2}-e^{w_2})\sin w_1\\-2w_2-tw_4 & 2w_1+tw_3 & t(e^{w_2}-e^{-w_2})\sin w_1 & t(e^{-w_2}+e^{w_2})\cos w_1\\tw_1 & tw_2 & 1 & 0\\-tw_2 & tw_1 & 0 & 1\\w_1w_3-w_2w_4-2 & w_1w_4+w_2w_3 & (e^{-w_2}+e^{w_2})\sin w_1-6 & (e^{w_2}-e^{-w_2})\cos w_1\end{pmatrix}. \tag{3.3.18}$$

We can now solve the BDE to follow the path from each starting point. In doing so, we find that following the path from $(-1,0,1,0)^{\mathbf{T}}$ yields the solution $\mathbf{w}^* = (-8.6884,0,8.3432,0)^{\mathbf{T}}$, which corresponds to the previously found solution $\mathbf{x}^* = (-8.6884,8.3432)^{\mathbf{T}}$ to $\mathbf{f}(\mathbf{x})=\mathbf{0}$. Following the path from $(1,0,1,0)^{\mathbf{T}}$ yields the solution $\mathbf{w}^* = (0.4563,0,6.1188,0)^{\mathbf{T}}$; therefore $\mathbf{x}^* = (0.4563,6.1188)^{\mathbf{T}}$ also solves $\mathbf{f}(\mathbf{x})=\mathbf{0}$.

As the reader can tell from our example, this process can be extremely tedious. Our example would have been much easier to solve had we substituted for x_2 in f_1 and used a numerical rootfinding technique. Even if an elimination strategy would not work, applying the Newton-Raphson method at a random selection of starting points would probably be more efficient. Constructing $\hat{\mathbf{h}}(\mathbf{w},t)$ is a very arduous task even if the dimension of the problem and the maximum q_k is relatively small. This method's impracticality rivals its aesthetic beauty. Even so, it is useful to see how the KT conditions can be cast as a nonlinear system so that constrained optimization problems can be solved by path following.

Recall from section 1.2 that for an optimization problem of the form

minimize $f(\mathbf{x})$
subject to $c_i(\mathbf{x})=0$, $i \in E$

$$c_i(\mathbf{x}) \geq 0, \qquad i \in I$$

then the first-order KT conditions are given by

(i)	$\nabla_x L(\mathbf{x}', \boldsymbol{\lambda}') = \mathbf{0}$	
(ii)	$c_i(\mathbf{x}') = 0,$	$\forall i \in E$
(iii)	$c_i(\mathbf{x}') \geq 0,$	$\forall i \in I$
(iv)	$\lambda_i' \geq 0,$	$\forall i \in I$
(v)	$\lambda_i' c_i(\mathbf{x}') = 0,$	$\forall i,$

where L is the Lagrangian function and the λ_i's are the Lagrange multipliers. If the constraint functions are well behaved (i.e. at least one point exists that satisfies all constraints, and a regularity assumption holds), then we can construct the following set of equations:

$$\begin{aligned} \nabla f(\mathbf{x}') - \sum_{i \in I} \alpha_i^+ \nabla c_i(\mathbf{x}') - \sum_{i \in E} \alpha_i \nabla c_i(\mathbf{x}') &= \mathbf{0} \\ c_i(\mathbf{x}') - \alpha_i^- &= 0, \qquad \forall i \in I \\ c_i(\mathbf{x}') &= 0, \qquad \forall i \in E, \end{aligned} \tag{3.3.19}$$

where $\alpha^+ = [\max(0, \alpha)]^k$, $\alpha^- = [\min(0, -\alpha)]^k$, and k is a positive integer that makes α^+ and α^- $(k\text{-}1)$-continuously differentiable everywhere. Equations (3.3.19) are called the *KT equations*. If $(\mathbf{x}', \boldsymbol{\alpha})$ solve the KT equations, then $(\mathbf{x}', \boldsymbol{\lambda}')$ satisfy the KT conditions, where

$$\lambda_i' = \begin{cases} \alpha_i^+, & i \in I \\ \alpha_i, & i \in E \end{cases}. \tag{3.3.20}$$

In order to solve a constrained optimization problem using the path following approach, one would construct the KT equations as in (3.3.19), construct an appropriate homotopy (such as the linear homotopy or Newton's homotopy), and then set up and solve the BDE. All possible solutions to the KT equations must be found, so either a Monte Carlo approach or the use of the All-Solutions homotopy is required. Once all KT points have been found, the global minimum can be found by determining which KT point has the smallest objective function value (and satisfies the second-order conditions). Since this process is very complicated for even the simplest of problems, it is recommended that a Monte Carlo scheme wrapped around the Newton-Raphson method is used to locate KT points.

Chapter 4: Histogram Thresholding for Image Segmentation

4.1 The Image Segmentation Problem

Segmenting grayscale images is a useful task in imaging science. Image segmentation can be used directly for image compression or as a preprocessing step for background removal, feature detection, or pattern recognition algorithms. A wide variety of segmentation techniques have emerged in the literature over the last two decades; for an excellent survey, see Sahoo, Soltani, and Wong [39]. The simplest methods (and the most efficacious to representation as a global optimization problem) are *point-based*, where each pixel is quantized based on thresholds determined by analysis of the histogram of the image. More sophisticated (and time consuming) are the *region-based* methods, which take into account local spatial characteristics of the image while quantizing each pixel.

In point-based segmentation methods, we assume that the value $I(u,v)$ at each pixel is drawn from one of n sub-populations (generally normal distributions). The histogram of the image will then appear to be a mixture of the sub-populations, and can be described by the following model:

$$h(x,\boldsymbol{\beta},\boldsymbol{\mu},\boldsymbol{\sigma}) = \sum_{i=1}^{n} \beta_i p_i(x), \tag{4.1.1}$$

where

$$\sum_{i=1}^{n} \beta_i = 1 \tag{4.1.2}$$

and

$$p_i(x) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-(x-\mu_i)^2/2\sigma_i^2}. \tag{4.1.3}$$

Without loss of generality, we will sort the indices so that $\mu_1 < \mu_2 < \cdots < \mu_n$. An actual image histogram is composed of a set of m ordered pairs $(\hat{x}_j, \hat{h}_j)$. The optimal parameters $\boldsymbol{\beta}^*$, $\boldsymbol{\mu}^*$, and $\boldsymbol{\sigma}^*$ solve the nonlinear least squares problem:

$$\underset{\boldsymbol{\beta},\boldsymbol{\mu},\boldsymbol{\sigma}}{\text{minimize}} \quad \sum_{j=1}^{m} \left[h(\hat{x}_j,\boldsymbol{\beta},\boldsymbol{\mu},\boldsymbol{\sigma}) - \hat{h}_j \right]^2. \tag{4.1.4}$$

Given the solution to (4.1.4), we can then calculate an optimal set of thresholds $\mathbf{T}$ by minimizing classification error. The $n = 2$ case is derived in Gonzalez and Wintz [19]; the general case is a simple extension. We will define $\mathbf{T} = (T_1, T_2, \ldots, T_{n-1})^{\mathbf{T}}$ and assume that $T_0 < T_1 < T_2 < \ldots < T_{n-1} < T_n$, where $T_0 = \min_j \hat{x}_j$ and $T_n = \max_j \hat{x}_j$. Given a set of thresholds, an image I is segmented to form a new image $\hat{I}$ in the following way:

$$\hat{I}(u,v) = \begin{cases} \phi(1), & T_0 \le I(u,v) < T_1 \\ \phi(2), & T_1 \le I(u,v) < T_2 \\ \vdots & \vdots \\ \phi(n) & T_{n-1} \le I(u,v) \le T_n \end{cases}, \tag{4.1.5}$$

where ϕ is an arbitrary one-to-one function such as $\phi(i)=i$ or $\phi(i)=\mu_i$.

To determine the optimum set of thresholds given $\boldsymbol{\beta}^*$, $\boldsymbol{\mu}^*$, and $\boldsymbol{\sigma}^*$, we must first introduce two random variables. Let **X** be the index of the sub-population to which a given pixel belongs, and let **C** be the index of the segment into which it is classified. Then the optimal **T** vector should maximize the probability that a pixel is correctly classified (or minimize the probability that a pixel is misclassified). Then

$$\begin{aligned}\theta(\mathbf{T}) &= P\{\text{misclassification}\} = 1 - P\{\text{correct classification}\} \\ &= 1 - \sum_{i=1}^{n} P\{\mathbf{C}=i, \mathbf{X}=i\} \\ &= 1 - \sum_{i=1}^{n} P\{\mathbf{C}=i \mid \mathbf{X}=i\} P\{\mathbf{X}=i\}\end{aligned}.$$

We know that $P\{\mathbf{X}=i\}=\beta_i^*$ and $P\{\mathbf{C}=i \mid \mathbf{X}=i\}=\int_{T_{i-1}}^{T_i} p_i(x)dx$, so we have:

$$\theta(\mathbf{T}) = 1 - \sum_{i=1}^{n} \int_{T_{i-1}}^{T_i} \beta_i^* p_i(x)dx. \tag{4.1.6}$$

Any point that minimizes (4.1.6) satisfies $\nabla\theta(\mathbf{T})=\mathbf{0}$. From the Fundamental Theorem of Calculus, we find that:

$$\nabla\theta(\mathbf{T}) = \begin{pmatrix} \beta_2^* p_2(T_1) - \beta_1^* p_1(T_1) \\ \beta_3^* p_3(T_2) - \beta_2^* p_2(T_2) \\ \vdots \\ \beta_n^* p_n(T_{n-1}) - \beta_{n-1}^* p_{n-1}(T_{n-1}) \end{pmatrix}. \tag{4.1.7}$$

Therefore, any optimal vector $\mathbf{T}^*$ satisfies $\beta_i^* p_i(T_i^*) = \beta_{i+1}^* p_{i+1}(T_i^*)$ for $i=1,\ldots,n-1$. Taking logarithms of both sides of these equations and simplifying yields quadratic equations in $\mathbf{T}^*$:

$$\begin{aligned}&\left(\sigma_i^{*2} - \sigma_{i+1}^{*2}\right) T_i^{*2} + 2\left(\mu_i^* \sigma_{i+1}^{*2} - \mu_{i+1}^* \sigma_i^{*2}\right) T_i^* + \mu_{i+1}^{*2} \sigma_i^{*2} \\ &\quad - \mu_i^{*2} \sigma_{i+1}^{*2} + \sigma_i^{*2} \sigma_{i+1}^{*2} \ln\left(\frac{\sigma_i^* \beta_i^*}{\sigma_{i+1}^* \beta_{i+1}^*}\right) = 0.\end{aligned} \tag{4.1.8}$$

Therefore, to determine a set of optimal thresholds given the population mixture model, we would solve the quadratic equations given by (4.1.8). It can be shown that any solution to (4.1.8) that satisfies $\mu_1^* < T_1^* < \mu_2^* < T_2^* < \cdots < T_{n-1}^* < \mu_n^*$ also satisfies the second-order conditions ($\nabla^2\theta(\mathbf{T})$ positive definite), and hence is a local minimizer of (4.1.6). The global minimizer can be found by finding the best of all the local minima.

Since we know that the determination of thresholds is a straightforward process, the difficulty in point-based methods is determining the population mixture model, or the solution to (4.1.4). Many local minima exist for this problem, so attacking it with simple local optimizers will not be adequate. Since the problem can potentially have high dimensionality, none of the deterministic methods of Chapter 3 look promising. However, some of the Monte Carlo techniques of Chapter 2 can be used quite easily to solve (4.1.4). In the remainder of this chapter, we will review a tree-annealing technique (similar to simulated annealing) proposed by Snyder, Bilbro, Logenthiran, and Rajala [42] that solves (4.1.4) for fixed n (Section 2), a simulated annealing heuristic proposed by Brunelli [6] that does not solve (4.1.4) but allows for

variable n (Section 3), and a genetic algorithm proposed by this author that solves (4.1.4) when n is not chosen a priori (Section 4).

4.2 Estimating Population Mixtures

Let us examine the nonlinear least squares problem (4.1.4) that estimates the population mixture parameters for the image histogram. At first glance, we may consider solving it using the Gauss-Newton method (Fletcher [15]), possibly from a random sampling of starting points. However, we can simplify the problem. For fixed $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, (4.1.4) is a linear least squares problem in terms of $\boldsymbol{\beta}$. Taking into account the equality constraint (4.1.2) and the inequality constraints $\boldsymbol{\beta} \geq \mathbf{0}$, we have a standard LSEI problem (Lawson and Hanson [28]) that can be solved easily for $\boldsymbol{\beta}$. Therefore, we can solve (4.1.4) by *partitioned least squares*, using Gauss-Newton to update estimates for the nonlinear parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and solving the resulting LSEI problem for $\boldsymbol{\beta}$ at each iteration. This effectively reduces the dimensionality of the nonlinear least squares problem by 1/3.

It is extremely difficult to find the global minimum of (4.1.4), even if the problem is partitioned. If there are n modes in the population, (4.1.4) describes a function in $3n$-dimensional space ($2n$ for the partitioned problem) that has a plethora of local minima. Because of the relatively high dimensionality of this problem, deterministic global optimization techniques are out of the question. Therefore, Monte Carlo techniques are our only resort. A multistart method like that of Section 2.1 might work well; however, this might not be very efficient because we know nothing about the basins of attraction of local minima. Another possibility is simulated annealing; this problem seems to lend itself well to such an approach.

Snyder, Bilbro, Logenthiran, and Rajala [42] describe a solution to (4.1.4) by tree annealing, a variant of simulated annealing. In tree annealing, the search space is described by a tree, where each level of the tree describes some binary partition of one of the parameters. For this problem, the search space can be described by $\mathbf{0} \leq \boldsymbol{\mu} \leq \mathbf{1}$, $\alpha_1 \leq \boldsymbol{\sigma} \leq \alpha_2$, and $\mathbf{0} \leq \boldsymbol{\beta} \leq \mathbf{1}$, where α_1 and α_2 are suitable lower and upper bounds on the standard deviations. The annealing procedure takes place by randomly traveling down one branch of the tree and then comparing the "leaf" point with the current iterate according to the Metropolis criterion. If the leaf point is chosen as the next iterate, another node is placed at that leaf to increase the resolution of the tree. If the Metropolis criterion is changed according to some cooling schedule, over time the tree will be highly resolved in areas of local minima.

There are two major drawbacks to the tree annealing approach:

(1) The size of the tree becomes enormous and storage becomes impractical, and
(2) It is not readily apparent how to satisfy the equality constraint (4.1.2)

The first drawback can be slightly alleviated by solving the partitioned problem. Even though the dimensionality of the problem is somewhat reduced, it does not become small enough to disregard the issue of tree size. The second drawback can be ignored if we use a standard elimination or transformation method to remove the equality constraint; however, the transformed parameters will have no intuitive significance.

For these reasons, this author recommends that if a simulated annealing approach is used to solve (4.1.4), it should be a continuous version of simulated annealing such as that of Corana et al. [9] described in Section 2.3. In addition, simulated annealing should be applied to the partitioned problem to reduce dimensionality and move satisfaction of (4.1.2) into the evaluation of the objective function.

To show an example of how simulated annealing performs for histogram thresholding, we will consider the two images in Figure (4.2.1). On the left is a professional portrait, and on the right is a cluttered scene of a room. By quickly observing the histograms of each image (see Figure (4.2.2)), we will assume that there are three modes or sub-populations in the left image and four modes in the right image. After normalizing each histogram, simulated annealing is used to search for optimal $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ vectors within the region defined by $\mathbf{0} \leq \boldsymbol{\mu} \leq \mathbf{1}$ and $0.01 \leq \boldsymbol{\sigma} \leq 0.5$ ($\boldsymbol{\beta}$ is computed for each combination by solving the corresponding LSEI problem). A geometric cooling schedule is used with initial temperature $T_0 = 100$ and $\alpha = 0.98$, and the system is cooled for 200 iterations. During each iteration, a metropolis algorithm is performed 100 times. Once the optimal set of parameters is found for the mixture model, the vector of

thresholds $\mathbf{T}^*$ is found according to (4.1.8). Figure (4.2.3) shows each normalized histogram (red dotted line), the individual sub-populations (blue dashed lines), the full population model (black solid line), and the optimal thresholds (magenta solid vertical lines). The original images are then segmented via (4.1.5) with $\varphi(i) = \mu_i^*$. The segmented images are shown in Figure (4.2.4).

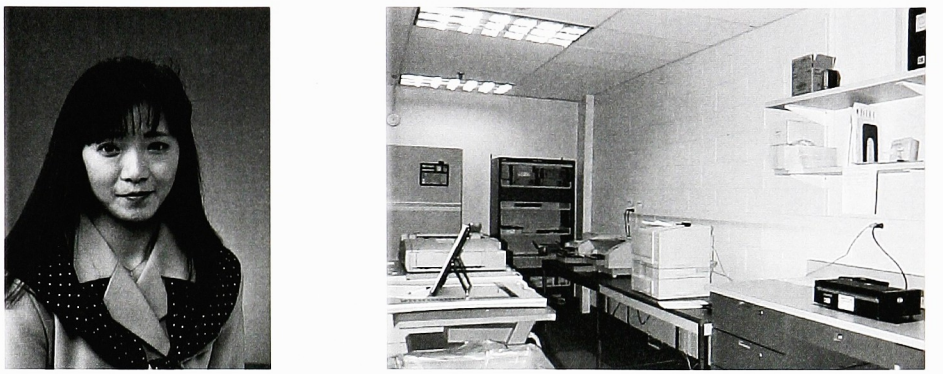

Figure (4.2.1): Image of woman (left); image of room (right).

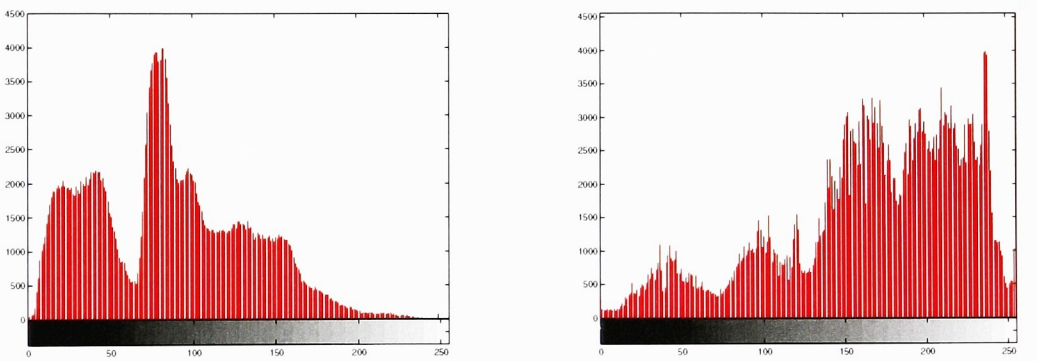

Figure (4.2.2): Histograms of woman and room images.

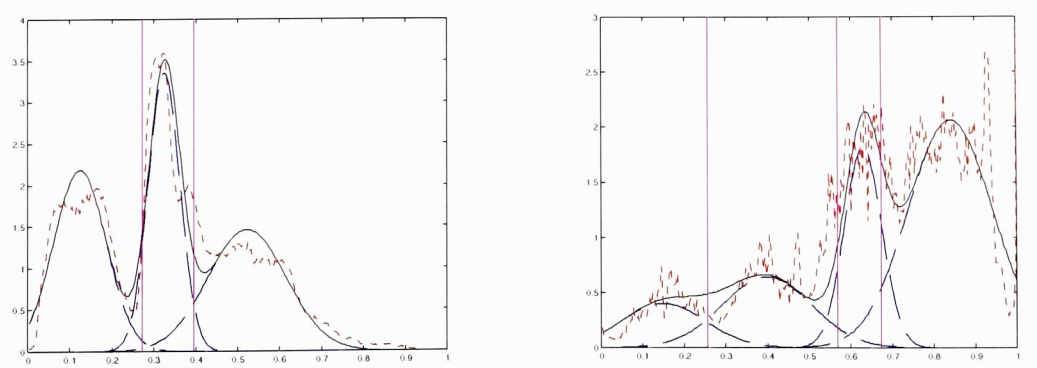

Figure (4.2.3): Population mixture models: histogram (red dotted), individual sub-populations (blue dashed), population model (black solid), optimal thresholds (magenta vertical solid)

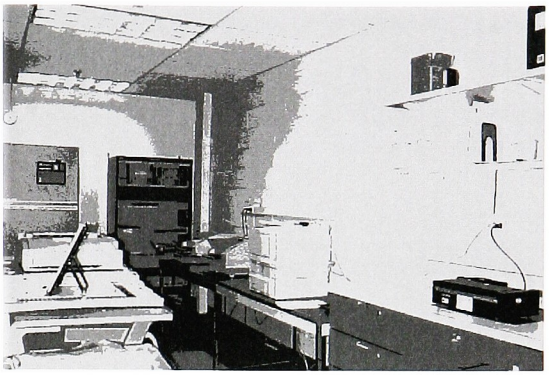

Figure (4.2.4): Thresholded images; woman at three levels, room at four levels.

4.3 Simulated Annealing for an Indeterminate Number of Thresholds

Another interesting technique for histogram thresholding is given by Brunelli [6]. He does not attempt to fit a population model to the histogram; rather, he solves a combinatorial optimization problem via simulated annealing that yields the thresholds directly. The benefit of using this method is that it is not required (although it is useful) to determine the number of modes/thresholds prior to performing the optimization. On the flip side, Brunelli's method does not yield any useful information for estimating sub-population parameters. At any rate, it is of interest here because it provides a novel approach to solving the image segmentation / histogram thresholding problem by global optimization.

In Brunelli's method, the image histogram is preprocessed to locate every local minimum. A node is defined the n local minima that are found. Seven pieces of information are stored at each node: node location, histogram value, and cumulative distribution value; location, histogram value, and cumulative distribution value of the nearest local maximum to the right of the node; and a binary flag. A configuration $\mathbf{x}$ is defined as the n-vector of all binary flags. Every element of $\mathbf{x}$ that is a 1 is considered to be a threshold value. There are 2^n possible configurations, one of which describes the optimal set of thresholds. The objective function we must minimize is given by:

$$f(\mathbf{x}) = -\sum_s \frac{\min\left(\left|m_s - k_{s-}\right|, \left|m_s - k_{s+}\right|\right)}{m_s}\left(1 - \frac{A_s}{m_s d_s}\right), \tag{4.3.1}$$

where s is a slice of the current configuration (a segment of the histogram corresponding to the interval containing a node $x_i = 1$ and the nearest node to its right with $x_j = 1$), m_s is the maximum histogram value of the slice, k_{s-} is the histogram value of the left node, k_{s+} is the histogram value of the right node, A_s is the area of the slice, and d_s is the width of the slice. Since the nearest maximum and the cumulative distribution values are stored at each node, computing m_s and A_s is simple and efficient. According to Brunelli, configurations that minimize (4.3.1) have the best "goodness of shape" between each threshold.

If the number of modes of the histogram is known a priori (say m modes), (4.3.1) can be transformed to favor configurations with m active by defining a new objective function:

$$\hat{f}(\mathbf{x}) = \frac{f(\mathbf{x})}{\alpha\left|\sum_{i=1}^{n} x_i - m\right| + 1}, \tag{4.3.2}$$

where α is a weighting factor.

Simulated annealing can now be used to minimize (4.3.1) or (4.3.2). New configurations in a "neighborhood" of the current configuration can be generated by flipping the active status of a uniformly distributed (from 1 to n) random number of bits. Brunelli uses both the geometric cooling schedule and an

exponentially decaying cooling schedule, performing the Metropolis algorithm 10000 times at each temperature level. Figure (4.3.1) shows the histograms (red dotted) and optimal thresholds (magenta solid) found by Brunelli's annealing method for the images shown in Figure (4.2.1). The resulting segmented images, thresholded to the average pixel value in each slice, are shown in Figure (4.3.2).

It is worthwhile to note that it may be possible to tune a genetic algorithm to solve (4.3.1) or (4.3.2) more quickly than simulated annealing. The way in which configurations are defined lends itself nicely to this approach.

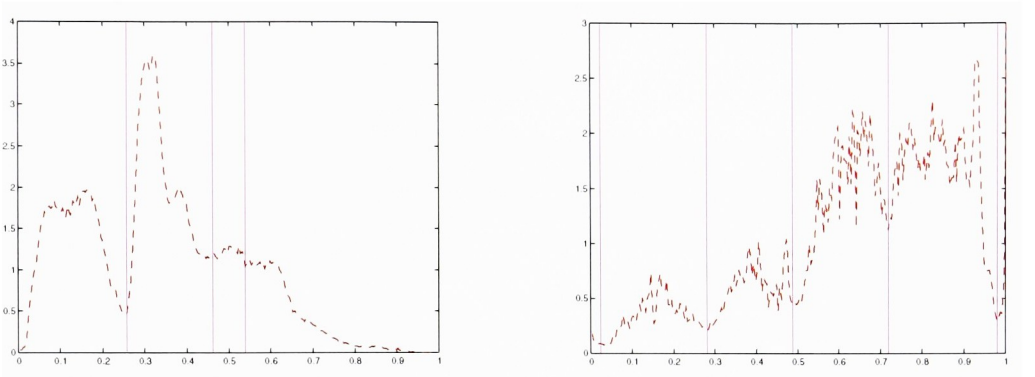

Figure (4.3.1): Image histograms (red dotted), optimal thresholds (magenta solid).

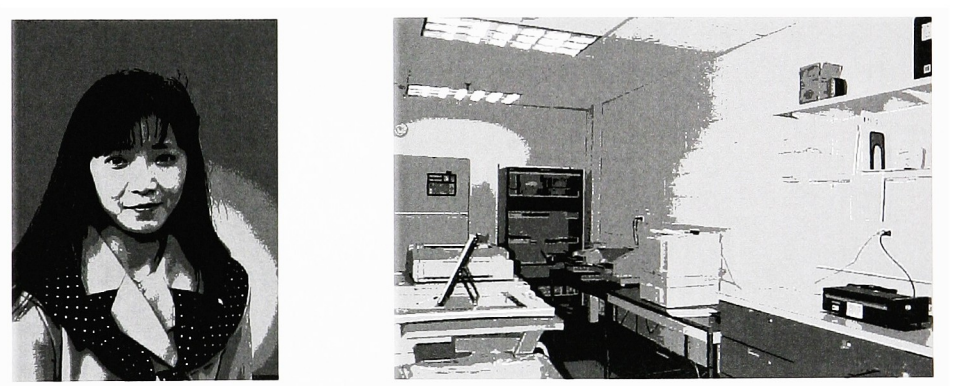

Figure (4.3.2): Thresholded images; woman at four levels, room at six levels.

4.4 Genetic Algorithms for an Indeterminate Number of Modes

Both Snyder's method and Brunelli's method for histogram thresholding have their strengths and weaknesses. Snyder's method (extended to solve the partitioned least squares problem) does a good job at finding the optimal configuration of parameters for the model (4.1.1), but it requires a priori knowledge of the number of sub-populations. Brunelli's method performs reasonably well when the number of sub-populations or desired thresholds is not known, but the notion of "goodness of shape" that it relies on is sketchy. This author has developed an image segmentation strategy that uses genetic algorithms to find the optimal set of parameters for the histogram model (4.1.1) without requiring that the number of sub-populations n be predetermined. The remainder of this section describes this new GA technique for histogram thresholding by describing the encoding of chromosomes, the crossover, mutation, and selection operators, and the fitness function.

We will begin by defining chromosomes: each chromosome is a length b vector that defines a collection of gaussians, where b is the number of histogram bins. For a given chromosome $\mathbf{x}$, if $x_i \neq 0$, then we say that this configuration contains a gaussian with mean located at the i^{th} bin. (If $x_i = 0$, then we say that this configuration contains no gaussian centered at the i^{th} bin). The value of x_i is the standard deviation

of that gaussian. Therefore, if we define an appropriate range for standard deviation values (say $0.01 \le \sigma \le 0.5$) and specify a vector of c linearly spaced values spanning this range, then there exist $b(c+1)$ possible chromosomes that provide good coverage of the μ and σ spaces for any $n \le b$. $\boldsymbol{\beta}$ can be found by solving the LSEI problem arising from (4.1.4), so $\boldsymbol{\beta}$ is implicitly stored for each chromosome.

For this problem, single-point crossover (as shown in Figure (2.4.2) with probability of crossover p_c = 0.7) will suffice. After two chromosomes are crossed over, the mutation operator is applied to the offspring. Each allele (element of **x**) is mutated with probability p_m = 0.001. Since the chromosomes are not binary vectors, we will define the mutation of an allele by the function $\omega(x)$:

$$\omega(x) = \begin{cases} \kappa, & x = 0 \\ 0, & x \neq 0 \end{cases}, \tag{4.4.1}$$

where κ is a random variable uniformly chosen from the possible values of σ. (4.4.1) allows a new sub-population to be entered into the mixture or a current sub-population to be removed. Given a current generation of chromosomes, selection occurs by probabilities weighted by the inverse of the relative fitness of the chromosomes (the inverse is taken since our aim is to minimize, not maximize, the objective function). An initial population of N chromosomes (N = 25 or 50 is a good choice) must be chosen so that it provides adequate coverage of parameter space. Since we make the assumption that each histogram has at least two sub-populations, we can generate an initial chromosome **x** by randomly choosing the number of modes n and then randomly choosing n elements of **x** in which to place nonzero values. The nonzero values are chosen uniformly from the set of possible standard deviations. The random choice of n can be achieved by assigning $n = 2 + \rho$, where $\rho \sim \text{Poisson}(\lambda)$. A good choice for the mean of ρ is $\lambda = 3$.

To fully describe this technique for histogram thresholding, all that remains is the definition of the fitness function. For this problem, we must design a fitness function that not only takes residual error into account, but also the number of sub-populations required to adequately explain variation. In order to do this, we divide the fitness computation into two parts. When a given chromosome is created, we calculate the residuals of the corresponding population model and determine the sum squared error:

$$sse(\mathbf{x}) = \sum_{j=1}^{m} \left[h(\hat{x}_j, \boldsymbol{\beta}_\mathbf{x}, \boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\sigma}_\mathbf{x}) - \hat{h}_j \right]^2. \tag{4.4.2}$$

where $\boldsymbol{\beta}_\mathbf{x}$, $\boldsymbol{\mu}_\mathbf{x}$, and $\boldsymbol{\sigma}_\mathbf{x}$ are the parameters of the histogram model (4.1.1) associated with the chromosome **x**, and the m ordered pairs $(\hat{x}_j, \hat{h}_j)$ compose the histogram. This calculation is performed for each chromosome in a given generation. Once the entire generation has been created, a master chromosome **M** that contains every unique (μ,σ) pair that exists in any chromosome of the generation is formed. (Note that it may not be possible to explicitly code **M** as a chromosome because there may be multiple gaussians with the same mean. We do not need to do this, however; all we need is to store $\boldsymbol{\mu}_\mathbf{M}$ and $\boldsymbol{\sigma}_\mathbf{M}$, the means and standard deviations of all gaussians described by **M**.) Given $\boldsymbol{\mu}_\mathbf{M}$ and $\boldsymbol{\sigma}_\mathbf{M}$, we can solve the LSEI problem (4.1.4) for $\boldsymbol{\beta}_\mathbf{M}$. In the statistical sense, we can think of $h(x, \boldsymbol{\beta}_\mathbf{M}, \boldsymbol{\mu}_\mathbf{M}, \boldsymbol{\sigma}_\mathbf{M})$ as the "full" model and $h(x, \boldsymbol{\beta}_\mathbf{x}, \boldsymbol{\mu}_\mathbf{x}, \boldsymbol{\sigma}_\mathbf{x})$ as a reduced model of the histogram. Using an F test, we can now define a fitness function $f(\mathbf{x})$ that accounts for both residual error and for the ability to explain variation:

$$f(\mathbf{x}) = \frac{sse(\mathbf{x})}{F_{\alpha,k-n,m-(k+1)}} \left[\frac{(sse(\mathbf{x}) - sse(\mathbf{M}))(m-(k+1))}{sse(\mathbf{M})(k-n)} \right], \tag{4.4.3}$$

where k is the number of modes in **M** and $F_{\alpha,k-n,m-(k+1)}$ is the critical value for an F distribution with parameters $\alpha(= 0.01)$, k-n, and m-(k+1). The effect of (4.4.3) is to increase (4.4.2) when we would reject the hypothesis that the reduced model is acceptable and to decrease (4.4.2) when we would not reject this hypothesis. (4.4.3) will be driven to a minimum when both the residuals are small and n is not deemed inacceptable. Two problems can potentially arise. First, the number of modes in **M** can exceed the number

of data points m extracted from the histogram. If this happens in a given generation, we can resample the histogram so that we extract as many points are needed. The second problem is that if two or more modes in **M** are similar (means and standard deviations are close), the predictor matrix in the LSEI problem can be poorly scaled. If this is the case, we can safely remove modes from **M** that are nearly linearly dependent on other modes.

The following figure shows the best histogram model and set of thresholds determined by the genetic algorithm for the images in Figure (4.2.1). Interestingly enough, the GA determined that the image of the woman should contain three modes and that the image of the room should contain four modes (which exactly corresponds to the number of modes we chose when we ran Snyder's algorithm). Figure (4.4.2) shows the results of segmenting these images based on the optimal thresholds found by the GA.

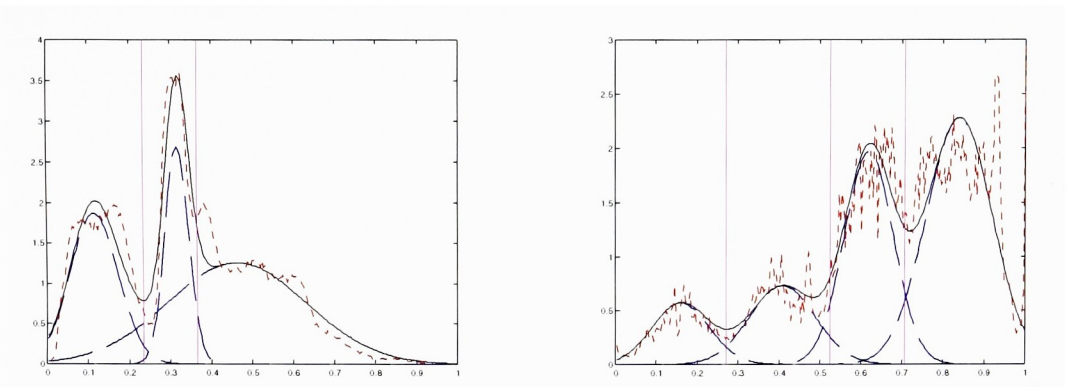

Figure (4.4.1): Population mixture models: histogram (red dotted), individual sub-populations (blue dashed), population model (black solid), optimal thresholds (magenta vertical solid)

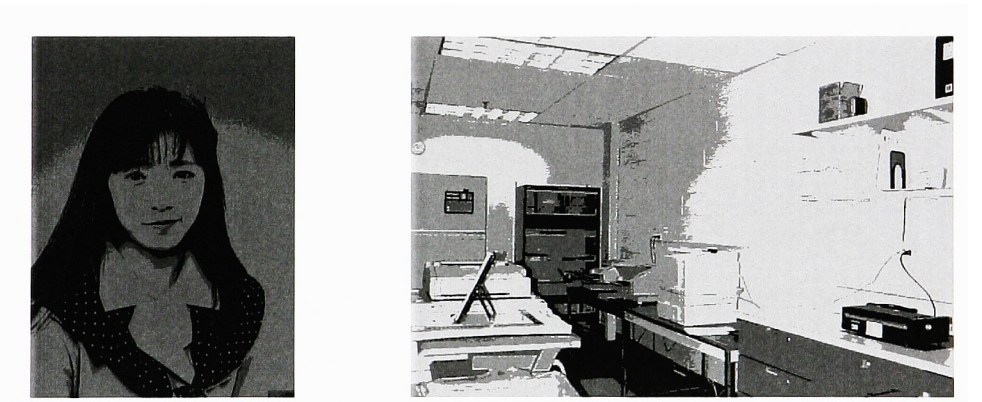

Figure (4.2.4): Thresholded images; woman at three levels, room at four levels.

Chapter 5: 3-D Registration of Point Sets

5.1 The 3-D Registration Problem

Another standard global optimization problem that arises in imaging science and computer vision is the 3-D registration problem. Given two sets of three-dimensional data, we want to determine the rotation and translation parameters that will register the two sets. This problem manifests itself in a variety of ways. For example, we may wish to design a robot that is capable of recognizing certain objects in its environment. In this case, we may use a stereo correspondence technique (by using two cameras as the "eyes" of the robot) to construct a three-dimensional image. If we have a 3-D model of the object we are looking for, we can solve the 3-D registration problem to locate that object in the image. In another example, we may wish to align multiple three-dimensional images of a scene that were captured/constructed from different viewpoints.

This chapter will describe how the 3-D registration problem can be solved for point sets using global optimization techniques. If point correspondence between the two data sets is known, the problem becomes much simpler. Section 2 outlines two algorithms for this case. In Section 3, we will examine the Iterative Closest Point (ICP) algorithm (a local optimizer that can be used when point correspondence is unknown). We will also describe how simple Monte Carlo techniques (such as multistart) can be used to initialize the ICP algorithm.

5.2 Algorithms for Known Point Correspondence

Two methods have arisen for solving the 3-D registration problem given a known correspondence between two point sets. Arun, Huang, and Blostein [2] derive an SVD algorithm that works under low to moderate noise, and Horn [1987] derives a similar method using quaternions that has a higher tolerance for noise. In order to describe the two algorithms, we will first state the 3-D registration problem mathematically: Given two 3-D point sets $X = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}\right\}$ and $Y = \left\{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(n)}\right\}$ where each $\mathbf{x}^{(k)}$ and $\mathbf{y}^{(k)}$ are corresponding 3 x 1 vectors, we describe a transformation $Y = \mathbf{f}(X)$ in the following way:

$$\mathbf{f}\left(\mathbf{x}^{(k)}\right) = \mathbf{R}\mathbf{x}^{(k)} + \mathbf{t} + \boldsymbol{\varepsilon}^{(k)}, \tag{5.2.1}$$

where $\mathbf{R}$ is a 3 x 3 rotation matrix ($\mathbf{R}$ unitary with $\det \mathbf{R} = 1$), $\mathbf{t}$ is a 3 x 1 translation vector, and $\boldsymbol{\varepsilon}^{(k)}$ is a noise vector. The solution to the 3-D registration problem is the <$\mathbf{R}$,$\mathbf{t}$> pair that minimizes

$$g(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{n} \left[\mathbf{y}^{(i)} - \left(\mathbf{R}\mathbf{x}^{(i)} + \mathbf{t}\right)\right]^{\mathrm{T}} \left[\mathbf{y}^{(i)} - \left(\mathbf{R}\mathbf{x}^{(i)} + \mathbf{t}\right)\right] \tag{5.2.2}$$

over the entire space of rotation matrices and translation vectors. Huang, Blostein, and Margerum [23] show that this least squares problem can be reduced by first minimizing

$$h(\mathbf{R}) = \sum_{i=1}^{n} \left[\mathbf{z}^{(i)} - \mathbf{R}\mathbf{w}^{(i)}\right]^{\mathrm{T}} \left[\mathbf{z}^{(i)} - \mathbf{R}\mathbf{w}^{(i)}\right] \tag{5.2.3}$$

over the space of rotation matrices, where

$$\mathbf{w}^{(i)} = \mathbf{x}^{(i)} - \frac{1}{n}\sum_{j=1}^{n} \mathbf{x}^{(j)} \quad \text{and} \quad \mathbf{z}^{(i)} = \mathbf{y}^{(i)} - \frac{1}{n}\sum_{j=1}^{n} \mathbf{y}^{(j)}. \tag{5.2.4}$$

Once $\mathbf{R}$ has been determined by minimizing (5.2.3), $\mathbf{t}$ can be computed by:

$$\mathbf{t} = \frac{1}{n}\left(\sum_{j=1}^{n} \mathbf{y}^{(j)} - \mathbf{R} \sum_{j=1}^{n} \mathbf{x}^{(j)} \right). \tag{5.2.5}$$

This simplification is somewhat intuitive; because a translation is a shift in the centroid of a point set, we can solve (5.2.2) by shifting X and Y so their centroids are at the origin, minimizing (5.2.3) to determine **R**, and shifting back.

Arun et al. show that if we only constrain **R** to be unitary (with determinant ±1), then $\mathbf{R} = \mathbf{V}\mathbf{U}^{\mathbf{T}}$ is the solution to (5.2.3), where $\mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$ is the singular value decomposition of the cross-covariance matrix $\sum_{i=1}^{n} \mathbf{w}^{(i)}\mathbf{z}^{(i)\mathbf{T}}$. Given this result, we now have an SVD algorithm for 3-D registration; namely:

Algorithm (5.2.1): SVD algorithm for 3-D registration

(i) Compute $\mathbf{w}^{(i)}$ and $\mathbf{z}^{(i)}$ from (5.2.4),

(ii) Construct $\mathbf{A} = \sum_{i=1}^{n} \mathbf{w}^{(i)}\mathbf{z}^{(i)\mathbf{T}}$,

(iii) Decompose **A** into $\mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$,

(iv) Set $\mathbf{R} = \mathbf{V}\mathbf{U}^{\mathbf{T}}$,

(v) Solve (5.2.5) for **t**.

Note that this algorithm fails if the determinant of **R** as computed in (iv) is -1. If this is the case, then it is more appropriate to model $Y = \mathbf{f}(X)$ as a reflection than a rotation. This failure mode can arise if there is significant noise in the data. Even if there is little noise in the data, symmetries in the point sets can cause the SVD algorithm to produce reflection rather than rotation matrices.

Horn [22] provides a quaternion approach to solving this problem that does not fail under high noise levels or symmetry. A quaternion vector $\mathbf{q} = (q_1, q_2, q_3, q_4)^{\mathbf{T}}$ satisfies $q_1 \geq 0$ and $\mathbf{q}^{\mathbf{T}}\mathbf{q} = 1$, and it specifies the rotation matrix $\mathbf{R}(\mathbf{q})$ given by:

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 & 2(q_2q_3 - q_1q_4) & 2(q_2q_4 + q_1q_3) \\ 2(q_2q_3 + q_1q_4) & q_1^2 - q_2^2 + q_3^2 - q_4^2 & 2(q_3q_4 - q_1q_2) \\ 2(q_2q_4 - q_1q_3) & 2(q_3q_4 + q_1q_2) & q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{bmatrix}. \tag{5.2.6}$$

Replacing **R** in (5.2.2) with $\mathbf{R}(\mathbf{q})$, we see that the solution to the 3-D registration problem is the <**q**,**t**> pair that minimizes

$$g(\mathbf{q}, \mathbf{t}) = \sum_{i=1}^{n} \left[\mathbf{y}^{(i)} - \left(\mathbf{R}(\mathbf{q})\mathbf{x}^{(i)} + \mathbf{t} \right) \right]^{\mathbf{T}} \left[\mathbf{y}^{(i)} - \left(\mathbf{R}(\mathbf{q})\mathbf{x}^{(i)} + \mathbf{t} \right) \right] \tag{5.2.7}$$

subject to $q_1 \geq 0$ and $\mathbf{q}^{\mathbf{T}}\mathbf{q} = 1$. By shifting each point set, we can reduce the problem to that of minimizing

$$h(\mathbf{q}) = \sum_{i=1}^{n} \left[\mathbf{z}^{(i)} - \mathbf{R}(\mathbf{q})\mathbf{w}^{(i)} \right]^{\mathbf{T}} \left[\mathbf{z}^{(i)} - \mathbf{R}(\mathbf{q})\mathbf{w}^{(i)} \right] \tag{5.2.8}$$

subject to $q_1 \geq 0$ and $\mathbf{q}^{\mathbf{T}}\mathbf{q} = 1$, where $\mathbf{w}^{(i)}$ and $\mathbf{z}^{(i)}$ are computed from (5.2.3). Horn proves that the solution $\mathbf{q}^*$ to (5.2.8) is the unit eigenvector corresponding to the maximum eigenvalue of **Q**, where

$$\mathbf{Q} = \begin{bmatrix} tr(\mathbf{A}) & \mathbf{b}^{\mathbf{T}} \\ \mathbf{b} & \mathbf{A} + \mathbf{A}^{\mathbf{T}} - tr(\mathbf{A})\mathbf{I}_3 \end{bmatrix}, \tag{5.2.9}$$

and where **A** is the cross-covariance matrix $\sum_{i=1}^{n} \mathbf{w}^{(i)}\mathbf{z}^{(i)\mathbf{T}}$ and $\mathbf{b} = (A_{23} - A_{32}, A_{31} - A_{13}, A_{12} - A_{21})^{\mathbf{T}}$. Given this result, we now have a quaternion algorithm for 3-D registration:

Algorithm (5.2.2): Quaternion algorithm for 3-D registration

(i) Compute $\mathbf{w}^{(i)}$ and $\mathbf{z}^{(i)}$ from (5.2.4),

(ii) Construct $\mathbf{A} = \sum_{i=1}^{n} \mathbf{w}^{(i)}\mathbf{z}^{(i)\mathbf{T}}$,

(iii) Construct $\mathbf{b} = (A_{23} - A_{32}, A_{31} - A_{13}, A_{12} - A_{21})^{\mathbf{T}}$,

(iv) Form **Q** as in (5.2.9),

(v) Set **q** to be the unit eigenvector corresponding to the largest eigenvalue of **Q**,

(vi) Compute **R**(**q**) as in (5.2.6)

(vii) Solve (5.2.5) for **t**.

Because of its robustness, the quaternion algorithm is preferred for 3-D registration. The SVD algorithm is useful, though, because it can be extended to higher dimensions; whereas, the quaternion algorithm can not be extended.

To illustrate 3-D registration graphically, consider the corresponding 3-D point sets in Figure (5.2.1). Since there is relatively low noise, we use the SVD algorithm to find the transformation from the second set to the first set. Figure (5.2.2) shows the result of applying this optimum transformation.

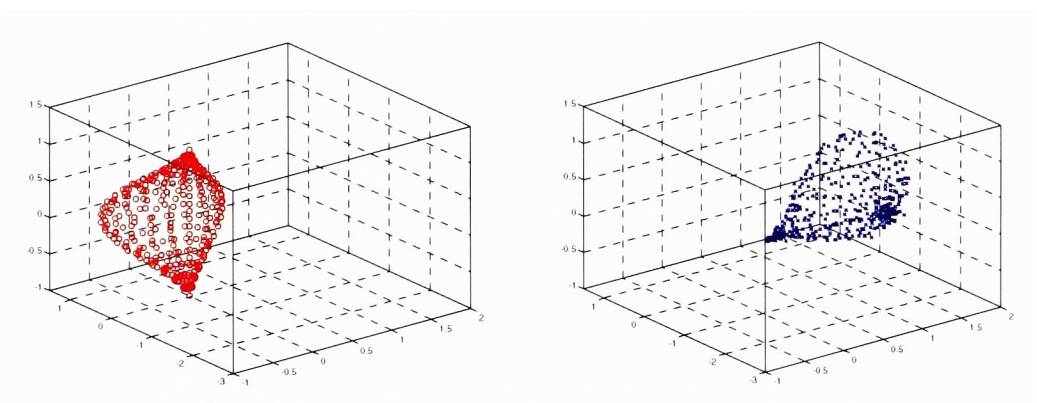

Figure (5.2.1): Corresponding 3-D point sets, set *Y* on left, set *X* on right

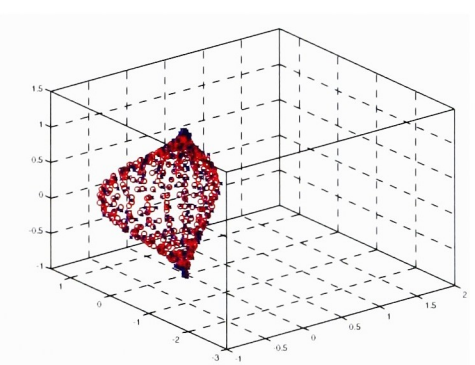

Figure (5.2.2): Result of applying transformation found by SVD algorithm.

5.3 Global Solutions for Unknown Point Correspondence

In more realistic 3-D registration problems, the correspondence between the point sets is not known. In fact, in many cases, there is no one-to-one correspondence! The number of points in each set may differ (imagine one point set describes a fine sampling of a model surface, and the other set describes a coarse sampling of an object), and after registration, only portions of the point sets may overlap. Clearly the SVD and quaternion algorithms (in their current form) are not suitable for this type of registration problem. Probably the most powerful and efficient algorithm available today that handles unknown point correspondence is the Iterative Closest Point (ICP) algorithm, described by Besl and McKay [4]. Part of the power of this algorithm is that it can be extended to model registration between sets of line segments, parametric curves, implicit curves, triangles, parametric surfaces, and implicit surfaces.

In its basic form (modeling the transformation between two point sets, X and Y), the ICP algorithm is a local optimizer that runs the quaternion algorithm with a series of "guesses" at the correspondence between the two point sets. The guesses are made by determining for each $\mathbf{x}^{(i)}$ in X the closest point $\mathbf{y}^{(i)}$ in Y in terms of euclidean distance. Once a closest point in Y has been found for every point in X, these closest points are collected into a set $\hat{Y}$, and the quaternion algorithm is performed to determine the appropriate rotation/translation to transform X into $\hat{Y}$. The transformation found by the quaternion algorithm is applied to X to form $\hat{X}$. If the sum of squared errors (5.2.7) or (5.2.8) is beneath some preset tolerance, the ICP algorithm terminates. If not, then another iteration of determining closest points, etc., is performed with X replaced by $\hat{X}$. Here is a description of the ICP algorithm:

Algorithm (5.3.1): Iterative Closest Point algorithm for 3-D registration

(i) Choose an initial transformation $<\mathbf{q}^{(0)},\mathbf{t}^{(0)}>$.

(ii) Set $k = 1$, apply $<\mathbf{q}^{(0)},\mathbf{t}^{(0)}>$ to X to generate $X^{(1)}$.

(iii) For each $\mathbf{x}^{(i)} \in X^{(k)}$, find $\mathbf{y}^* \in Y$ that minimizes $\left\|\mathbf{x}^{(i)} - \mathbf{y}\right\|_2$. Define $\hat{\mathbf{y}}^{(i)} = \mathbf{y}^*$.

(iv) Construct $\hat{Y} = \left\{\hat{\mathbf{y}}^{(i)}\right\}$, $i = 1,\ldots,|X|$.

(v) Run Algorithm (5.2.2) to determine the optimal transformation $<\mathbf{q}^{(k)},\mathbf{t}^{(k)}>$ from $X^{(k)}$ to $\hat{Y}$.

(vi) Apply $<\mathbf{q}^{(k)},\mathbf{t}^{(k)}>$ to $X^{(k)}$ to form $X^{(k+1)}$.

(vii) If $g(\mathbf{q}^{(k)},\mathbf{t}^{(k)}) < \varepsilon$ ((5.2.7) with $X=X^{(k)}$, $Y=\hat{Y}$), go to (viii). If not, set $k = k + 1$, go to (iii).

(viii) Compute overall rotation $\mathbf{R} = \mathbf{R}\left(\mathbf{q}^{(k)}\right)\mathbf{R}\left(\mathbf{q}^{(k-1)}\right)\cdots\mathbf{R}\left(\mathbf{q}^{(0)}\right)$ from X to Y.

(ix) Compute overall translation $\mathbf{t} = \mathbf{t}^{(k)} + \sum_{i=0}^{k-1}\left[\prod_{j=i+1}^{k}\mathbf{R}\left(\mathbf{q}^{(j)}\right)^{\mathbf{T}}\right]\mathbf{t}^{(i)}$ from X to Y.

The major difficulty with the ICP algorithm is that it is only a local optimization technique. If the initial transformation $<\mathbf{q}^{(0)},\mathbf{t}^{(0)}>$ is sufficiently close to the global minimizer, then convergence to that global minimum will occur rapidly. As is the case in all global optimization problems, we are left with only a vague notion of what "sufficiently close" means in a practical sense. Clearly none of the deterministic global optimization methods are worthwhile here; simulated annealing and/or genetic algorithms can be used, but they are extremely costly. The only methods that provide us with any hope of finding the global minimum in a reasonable amount of time are the Monte Carlo method of Section 2.1 or a Random Search technique like those described in Section 2.2. Besl and McKay suggest a random search method, where initial quaternion vectors are chosen from a uniform distribution on the 3-sphere, and initial translation vectors are uniformly chosen within some bounding box. However, when both point sets describe the same portions of the same 3-D object (even if they are at different resolutions), we can design a multistart technique that requires fewer starting points.

If the point sets in our 3-D registration problem comprise the same portions of the same object (like that shown in Figure (5.2.1)), we would expect that there is a correspondence between the eigenvectors of the covariance matrix of each point set. Assuming we have already adjusted the centroids of each point set, these eigenvectors $\mathbf{v}_{\mathbf{W}}^{(1)}$, $\mathbf{v}_{\mathbf{W}}^{(2)}$, $\mathbf{v}_{\mathbf{W}}^{(3)}$, $\mathbf{v}_{\mathbf{Z}}^{(1)}$, $\mathbf{v}_{\mathbf{Z}}^{(2)}$, and $\mathbf{v}_{\mathbf{Z}}^{(3)}$ can be computed most readily by

noting that they are the left singular vectors of $\mathbf{W}^{\mathbf{T}} = \left[\mathbf{w}^{(1)} : \mathbf{w}^{(2)} : \cdots\right]^{\mathbf{T}}$ and $\mathbf{Z}^{\mathbf{T}} = \left[\mathbf{z}^{(1)} : \mathbf{z}^{(2)} : \cdots\right]^{\mathbf{T}}$. If we collect the eigenvectors so we have $\mathbf{V_W} = \left[\mathbf{v}_{\mathbf{W}}^{(1)} \quad \mathbf{v}_{\mathbf{W}}^{(2)} \quad \mathbf{v}_{\mathbf{W}}^{(3)}\right]$ and $\mathbf{V_Z} = \left[\mathbf{v}_{\mathbf{Z}}^{(1)} \quad \mathbf{v}_{\mathbf{Z}}^{(2)} \quad \mathbf{v}_{\mathbf{Z}}^{(3)}\right]$, then we can generate a set of 24 initial rotations. Each rotation has the form:

$$\mathbf{R} = \mathbf{P}\mathbf{D}\mathbf{V}_{\mathbf{Z}}\mathbf{V}_{\mathbf{W}}^{\mathbf{T}}, \tag{5.3.1}$$

where **P** is a 3 x 3 permutation matrix (there are six of these) and **D** is a 3 x 3 diagonal matrix with entries ± 1 (there are eight of these). **D** is needed because the eigenvectors of the covariance matrices are ambiguous up to sign. **P** provides for the possibility of swapping the eigenvectors; this may be needed if the eigenvalues are close (although the closer they are, the less we can rely on this set of initial conditions as being adequate). Note that there are 48 matrices given by (5.3.1) that span all the possible rotations and reflections. Since we are only concerned with rotations, we keep only the 24 matrices with determinant +1. We can use this set of 24 matrices and the translation that aligns the centroids as a set of initial states for the ICP algorithm. Even though Algorithm (5.3.1) specifies the need for an initial quaternion vector, we can apply the initial rotation matrices directly in step (2).

In Figure (5.3.1), we see the result of applying the ICP algorithm to the two point sets in Figure (5.2.1) (assuming we do not know point correspondence) from two different starting rotations. In the blue point set *X*, half of the points have been removed, so there is no longer a strict correspondence between *X* and *Y*. This is meant to mimic practical problems where a 3-D object and a 3-D model might be sampled at different resolutions. On the left, the algorithm has converged into a local minimum; on the right, the algorithm has found the global minimum. Each initial transformation came from one of the 24 rotations found by (5.3.1).

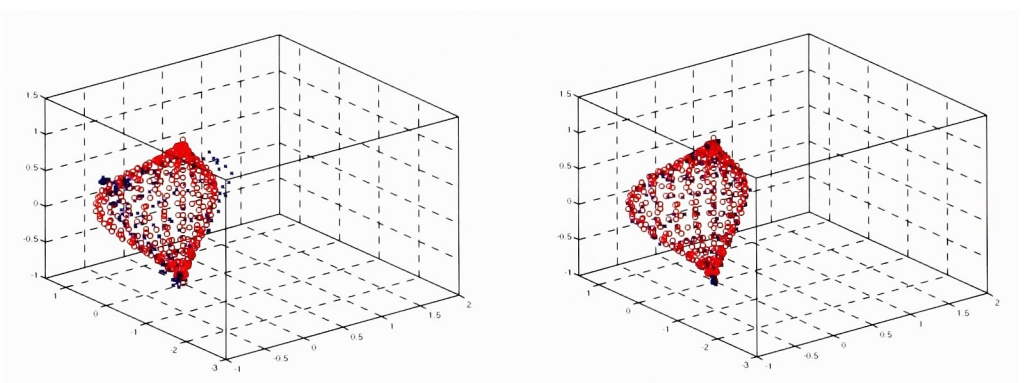

Figure (5.3.1): Two runs of the ICP algorithm: convergence into a local minimum (left), and convergence into the global minimum (right).

Because of the cost of objective function evaluations, most of the other global optimization techniques discussed in this thesis (and in the literature) are not effective for this problem. The only way to avoid random searches is to obtain some knowledge of the basin of attraction of the global minimum. This knowledge can be obtained by performing an exhaustive search of the transformation space, but this can only be applied to problems where such a search has been performed for an extremely similar 3-D object. Another way to improve the chances of finding the global minimum in a reasonable amount of time is to run a coarse surface-matching algorithm to generate initial guesses for rotation and translation. One such surface matching algorithm is described in Johnson [24]. A set of local 2-D histograms (called "spin images") of 3-D object or point set are computed and then correlated using standard image processing techniques. Johnson's algorithm, or any other suitable surface matching algorithm, could provide an efficient method for coming up with a good approximation to the transformation between two objects (presumably near the global minimum); the ICP algorithm could then be employed to refine the approximation.

Acknowledgments

There are a number of people who have helped me specifically with this thesis and in general with my Master's Degree, and I would like to thank them:

Richard Orr, for being my advisor, and for his thoughtful insights,

John Spence, for guiding me in appropriate research directions, and for many thought-provoking discussions about optimization,

John D'Errico, for mentoring, getting me interested in numerical linear algebra and optimization, serving as a sounding board for ideas, and teaching me MATLAB,

Larry Ray and Hsien-Che Lee, for helpful conversations about histogram thresholding,

Chris and Bev Cahill, for instilling in me the desire for the pursuit of knowledge, and

Kathy Embt, for patience and support.

Bibliography

[1] Aarts, E. and van Laarhoven, P. Simulated Annealing: an Introduction. *Statistica Neerlandica*, 43(1): 31-52, 1989.

[2] Arun, K. S., Huang, T. S., and Blostein, S. D. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 9(5):698-700, September 1987.

[3] Beale, E. M. L. Integer Programming. In *The State of the Art in Numerical Analysis* (ed. Jacobs, D. A. H.), Academic Press, 1978.

[4] Besl, P. and McKay, N. D. A Method for Registration of 3-D Shapes. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(2):239-256, February 1992.

[5] Broyden, C. G. The Convergence of a Class of Double Rank Minimization Algorithms, Parts I and II. *J. Inst. Maths. Applns.*, 6:76-90 and 222-231, 1970.

[6] Brunelli, R. Optimal Histogram Partitioning Using a Simulated Annealing Technique. *Pattern Recognition Letters*, 13:581-586, 1992.

[7] Cerny, V. Thermodynamical Approach to the Travelling Salesman Problem: An Efficient Simulation Algorithm. *J. Opt. Theo. Applns.*, 45:41-51, 1985.

[8] Cetin, B., Barhen, J., and Burdick, J. Terminal Repeller Unconstrained Subenergy Tunneling (TRUST) for Fast Global Optimization. Journal of Optimization Theory and Applications, 77(1):97-126, April 1993.

[9] Corana, A., Marchesi, M., Martini, C., and Ridella, S. Minimizing Multimodal Functions of Continuous Variables with the "Simulated Annealing" Algorithm. *ACM Transactions on Mathematical Software*, 13(3):262-280, September 1987.

[10] Curry, H. The Method of Steepest Descent for Nonlinear Minimization Problems. Quart. Appl. Math., 2:258-261, 1944.

[11] Dantzig, G.B. and Wolfe, P. A Decomposition Principle for Linear Programs. *Operations Res.*, 8:101-111, 1960.

[12] Davidon, W. C. Variable Metric Method for Minimization. AEC Res. and Dev. Report ANL-5990 (revised), 1959.

[13] Fletcher, R. and Powell, M. J. D. A Rapidly Convergent Descent Method for Minimization. *Computer J.*, 6:163-168, 1963.

[14] Fletcher, R. A New Approach to Variable Metric Algorithms. *Computer J.*, 13:317-322, 1970.

[15] Fletcher, R. *Practical Methods of Optimization*. John Wiley & Sons, 1987.

[16] Fletcher, R. and Reeves, C. M. Function Minimization by Conjugate Gradients. *Computer J.*, 7:149-154, 1964.

[17] Garcia, C. B. and Zangwill, W. I. *Pathways to Solutions, Fixed Points, and Equilibria*. Prentice-Hall, 1981.

[18] Goldfarb, D. A Family of Variable Metric Methods Derived by Variational Means. *Maths. Comp.*, 24:23-26, 1970.

[19] Gonzalez, R. C. and Wintz, P. *Digital Image Processing*. Addison-Wesley, 1977.

[20] Hestenes, M. R. Multiplier and Gradient Methods. *J. Opt. Theo. Applns.*, 4:303-320, 1969.

[21] Holland, J. *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975.

[22] Horn, B. P. K. Closed-Form Solution of Absolute Orientation Using Unit Quaternions. *J. Opt. Soc. Amer. A*, 4(4):629-642, April 1987.

[23] Huang, T. S., Blostein, S. D., and Margerum, E. A. Least-Squares Estimation of Motion Parameters from 3-D Point Correspondences. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, June 1986.

[24] Johnson, A. E. *Spin-Images: A Representation for 3-D Surface Matching*. Ph.D. dissertation, Carnegie Mellon University, CMU-RI-TR-97-47, 1997.

[25] Johnson, D. S., et al. Optimization by Simulated Annealing: An Experimental Evaluation - Part I (Graph Partitioning). *Operations Research*, 37:865-892, 1989.

[26] Kirkpatrick, S., Gelatt, C., and Vecchi, M. Optimization by Simulated Annealing. *Science*, 220(4598):671-680, 13 May 1983.

[27] Kuhn, H. W. and Tucker, A. W. Nonlinear Programming. *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, University of California Press, 1951.

[28] Lawson, C. L. and Hanson, R. J. *Solving Least Squares Problems*. SIAM, 1995.
[29] Levenberg, K. A Method for the Solution of Certain Non-linear Problems in Least Squares. Quarterly of Applied Mathematics, 2(2):164-168, July 1944.
[30] Levy, A., Gomez, S. The Tunneling Method Applied to Global Optimization. *Numerical Optimization 1984* (*Proc. SIAM Conf. on Numerical Optimization*), 1985.
[31] Marquardt, D. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *J. Soc. Indust. Appl. Math*, 11(2):431-441, June 1963.
[32] Mathworks. *The Record Company Contest*, http://www.mathworks.com/contest, 1998.
[33] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.*, 21:1087-1092, 1953.
[34] Nelder, J. A. and Mead, R. A Simplex Method for Function Minimization. *Computer J.*, 7:308-313, 1965.
[35] Osman, I. H. and Christofides, N. Capacitated Clustering Problems by Hybrid Simulated Annealing and Tabu Search. *Int. Trans. Oper. Res.*, 1:317-337, 1994.
[36] Patel, N. R., Smith, R. L., and Zabinsky, Z. B. Pure Adaptive Search in Monte Carlo Optimization. *Mathematical Programming*, 43:317-328, 1988.
[37] Romeijn, H. E. Global Optimization: From Pure Adaptive Search to Simulated Annealing. *Ten Years LNMB, Ph.D. Research and Graduate Courses of the Dutch Network of Operations Research* (eds. Haneveld, W. K. K., Vrieze, O. J., Kallenberg, L. C. M), CWI, 1997.
[38] Ross, S. M. *Introduction to Probability Models*, Academic Press, 1997.
[39] Sahoo, P., Soltani, S., and Wong, A. A Survey of Thresholding Techniques. *Computer Vision, Graphics, and Image Processing*, 41:233-260, 1988.
[40] Schittowski, K. NLQPL: A FORTRAN-Subroutine Solving Constrained Nonlinear Programming Problems. *Annals of Operations Research*, 5:485-500, 1985.
[41] Shanno, D. F. Conditioning of Quasi-Newton Methods for Function Minimization. *Maths. Comp.*, 24:647-656, 1970.
[42] Snyder, W., Bilbro, G., Logenthiran, A., and Rajala, S. Optimal Thresholding - A New Approach. *Pattern Recognition Letters*, 11:803-810, 1990.
[43] Spendley, W., Hext, G. R., and Himsworth, F. R. Sequential Application of Simplex Designs in Optimization and Evolutionary Operation. *Technometrics*, 4:441-461, 1962.
[44] Wales, D. J. and Doye, J. P. K. Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *J. Phys. Chem. A*, 101:5111-5116, 1997.
[45] Wilson, R. B. *A Simplicial Algorithm for Concave Programming*. Ph.D. dissertation, Harvard Univ. Graduate School of Business Administration, 1963.
[46] Vanderbilt, D. and Louie, S. A Monte Carlo Simulated Annealing Approach to Optimization over Continuous Variables. *Journal of Computational Physics*, 56:259-271, 1984.
[47] Zabinsky, Z. B. and Smith, R. L. Pure Adaptive Search in Global Optimization. *Mathematical Programming*, 53:323-338, 1992.