Image Segmentation by Texture Discrimination
Using Variations of Simulated Annealing

by
Mark Pipes

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 2002

© Mark Pipes, MMII. All rights reserved.

The author hereby grants to MIT permission to reproduce and
distribute publicly paper and electronic copies of this thesis document
in whole or in part.

Author ..........................
Department of Electrical Engineering and Computer Science
May 24, 2002

Certified by ..........................
Leslie P. Kaelbling
Thesis Supervisor

Accepted by ..........................
Arthur C. Smith
Chairman, Department Committee on Graduate Theses
Image Segmentation by Texture Discrimination Using Variations of Simulated Annealing

by

Mark Pipes

Submitted to the Department of Electrical Engineering and Computer Science on May 24, 2002, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

Traditional simulated annealing is compared with two new variations in terms of accuracy of image segmentation based on texture discrimination. The first variation is binary decomposition, which repeatedly divides the image into two texture regions using traditional simulated annealing. The second is dynamic temperature selection, which makes use of the system’s energy gradient to dynamically choose an efficient temperature. Binary decomposition, though not as accurate as traditional simulated annealing, shows promise, while dynamic temperature selection rivals traditional simulated annealing in accuracy and is much faster. The approach to image modeling specified by Hofmann et al. [5] is chosen and implemented, and some improvements are discussed based on experimental evidence.

Thesis Supervisor: Prof. Leslie P. Kaelbling
Acknowledgments

First and foremost, I must thank my parents for bringing me up to be an independent thinker, and supporting me in deed and spirit throughout my twenty-two year career.

Next, I would like to thank Professor Leslie Kaelbling for her guidance and discussion, which were invaluable to the completion of this thesis.

Beracah Yankama’s constant desire and eagerness to discuss my thesis were a great help in allowing me to see every natural process as a variant of annealing.

Vikas Anant’s repeated offers to edit (read “shred”) my thesis were greatly appreciated, though not taken up.

Lastly, I want to thank the old motley crew (Carlos, Chris, Jon, and Murphy) for their lasting and true friendship throughout my college years.
## Contents

1 Introduction ........................................... 11
   1.1 Related Work ........................................ 13
   1.2 Objective ........................................... 13

2 Features, Comparisons, and Objectives .......................................... 15
   2.1 Features ............................................. 15
   2.2 Comparisons .......................................... 18
   2.3 Objectives ............................................ 19
   2.4 Next .................................................. 20

3 Annealing and Variations .................................................. 21
   3.1 Simulated Annealing (SA) ................................ 21
      3.1.1 Related Work on Simulated Annealing ............ 24
   3.2 Variations ............................................. 26
      3.2.1 Single, Low Temperature Annealing (SLTA) ......... 26
      3.2.2 Binary Decomposition (BD) ......................... 27
      3.2.3 Dynamic Temperature Selection (DTS) .............. 28

4 Experimental Results .................................................. 31
   4.1 Single, Low Temperature Annealing (SLTA) .............. 32
   4.2 Simulated Annealing (SA) ................................ 35
   4.3 Binary Decomposition (BD) ................................ 38
   4.4 Dynamic Temperature Selection (DTS) ..................... 41
4.5 Real-World Images ................................................. 42

5 Discussion and Conclusion ........................................ 47
5.1 Visitation Schedules ............................................... 47
5.2 BD ................................................................. 48
5.3 Speckles ............................................................. 49
5.4 Scale and Features .................................................. 51
5.5 Phase Changes ...................................................... 52
5.6 Image Sequences .................................................... 52
5.7 Contributions and Conclusion .................................... 53
List of Figures

2-1 A typical five texture image with the lattice points and their cells overlayed. .......................................................... 16
2-2 The 12 gabor filters used to characterize the images. ............... 17
2-3 A sample histogram \( f_1(\cdot, \bar{x}) \), in which the center-weighted response of \( \bar{x} \)'s cell to filter 1 spikes at a value of 160. ............................... 18
3-1 \( T \) vs. Probability of \( S_{100} \) .................................................. 24
3-2 The Gaussian (dashed) and Cauchy (solid) distributions. The "fatter" tails of the Cauchy distribution allow egress from local minima at lower temperatures (i.e. when the distribution is narrower). .............. 25
4-1 The patterns used to create images with two, three, five, and seven textures. .......................................................... 31
4-2 The image on the left is given the segmentation pattern in the 20x20 grid on the right. ....................................................... 33
4-3 The original image and its segmentation after one, two, three, four, and five epochs. The bottom row contains the corresponding system energy. ................................................................. 34
4-4 A five-texture image and its segmentation using SLTA ............... 35
4-5 Plot of performance using SLTA and SA versus the number of textures. 36
4-6 These are histograms of the final system energy reached, using SLTA and SA on images with two, three, and five textures. ............. 37
4-7 A five-texture image and the plot of its energy using the general-purpose annealing schedule (dotted) and a custom-made annealing schedule (solid). Both perform well (accuracy ≈ 75%) but the custom schedule reaches equilibrium faster.

4-8 A plot of system energy versus time using SA on a five-texture image, and the resulting segmentation. The temperature has been scaled up for easy comparison. It steps through the values 1.1, 1, .8, .6, .4, .2, .1.

4-9 A plot of the average ratio between the heuristic estimate of the minimum energy and the actual minimum energy for 2, 3, 5, and 7 texture images. All values are very close to one, indicating a good estimate.

4-10 Plot of performance using SLTA (solid), SA (dashed), and BD (dotted) versus the number of textures.

4-11 The segmentation on the left is the result of one level of binary decomposition. The one on the right is the result of a second decomposition. Notice the originally misclassified region on the bottom right cannot be corrected. It is modified slightly by light noise reduction performed between decompositions.

4-12 A typical energy curve using DTS, and the corresponding temperature function used to generate it. The bottom plot is the temperature smoothed with a Gaussian.

4-13 Performance plots for 2, 3, 5, and 7 texture images using DTS with $\delta = 3$ (solid), 5 (dashed), 7 (dash-dotted), and 9 (dotted).

4-14 Images of a hallway and living room segmented by DTS.

4-15 The segmentation of a complex image of a residential street.

5-1 Accuracy of regular SA and DTS with the neighbor restriction. Below are two segmentations of the same image. The left segmentation does not make use of the neighbor restriction; the right one does.

5-2 Three textures that are problematic when broken into 20×20 cells.
Chapter 1

Introduction

The history of artificial intelligence has shown us that many of the tasks humans perform with ease are the hardest to solve computationally, while tasks that humans find mentally difficult often turn out to be easier than expected. For example, there is currently no machine capable of reliably extracting the three-dimensional structure of any scene from its two-dimensional image, a task humans perform on a real-time basis throughout the day. At the same time, the current world chess champion is a computer. This suggests that “intelligence” can be split into at least two broad categories:

- **type A**: the intelligence needed to perform seemingly simple everyday human tasks, and

- **type B**: the intelligence needed to perform seemingly difficult mental operations.\(^1\)

Type A intelligence is inseparably linked with the world around us. Humans need this intelligence to survive on a moment-to-moment basis. It is no surprise, then, that we have evolved to be highly proficient in solving type A problems. Furthermore, all other highly-evolved species exhibit advanced type A intelligence, as well.

\(^1\)Both kinds of intelligence are defined relative to human actions because there is no other basis from which we can define them.
Type B intelligence can be regarded as one step removed from the physical world; it involves more abstract thought, which may or may not relate to events in the world. Type B intelligence often has to do with planning ahead or finding the best way to solve a given problem, and is not, therefore, always a matter of immediate survival. As a result, the evolution of type B intelligence has been slower, and few species appear to exhibit it at all.

It seems clear that the extents to which both types of intelligence are found on Earth does not reflect their relative complexities. Nevertheless, the forefathers of artificial intelligence mostly limited their exploration to type B intelligence, which is of little practical significance in the effort to create autonomous, survivable machines.

Today, there are many research groups working on creating type A intelligence. One of the most difficult fields of type A research is computer vision. The goal of this research is to create machines that can process images or sequences of images in a meaningful way. A sub-field of computer vision is image segmentation, the process of breaking an image into homogeneous regions. This is a vague definition because regions could be “homogeneous” in a number of different ways. The issue here is one of feature selection. That is, homogeneous could be defined in terms of color, texture, or something even more high-level like person/not person. The assumption behind any one of these definitions is that meaning is encoded in the chosen features. More specifically, regions with similar feature values tend to be the same at higher levels of abstraction (e.g. they could be part of the same physical object), and regions with different feature values tend not to be the same. This thesis will make the assumption that meaning is encoded in texture, and will be concerned with image segmentation along these lines.

Some applications that make use of image segmentation today are automated product quality assurance, 3D rendering, and radiation therapy. Future applications could be as far-flung as autonomous robot navigation.
1.1 Related Work

The two most common approaches to image segmentation are the graph approach and the clustering approach. In the graph approach, each pixel (or group of pixels) is treated as a node and is linked to other pixels. Some difference metric is used to weight each of the links so that nodes that are less similar are more weakly connected. This set-up allows for the easy use of well-known min-cut algorithms [15]. The idea is that regions of high similarity will remain intact. In practice, however, this method usually results in one large region and one very small region. To solve this problem, Shi and Malik [12] proposed the use of a normalized cut, where the value of a cut is basically normalized by the size of the two resulting regions.

Standard clustering methods (e.g. k-means) have also been used to group sets of pixels into segments [7] [10]. Features used have included Gabor filter responses (which we shall make use of here), average intensity values over a region, and various non-linear combinations of these.

1.2 Objective

Analysis of the methods mentioned in the previous section reveal that most approaches to solving the image segmentation problem involve two stages:

- Feature selection/image modeling
- Finding the optimal segmentation

The objective of this thesis is to introduce a few novel variations of the well-known optimization method of simulated annealing, and compare their performances in the arena of image segmentation. To this end, the feature selection/image modeling approach proposed by Hofmann, et al. [5] is chosen because of its past success. Chapter 2 explains this approach, and our implementation of it, in detail. Chapter 3 describes the traditional simulated annealing algorithm and the variations experimented with in this paper. Chapter 4 presents the results of these experiments, and Chapter 5 contains a discussion of these results along with some suggestions for further work.
Chapter 2

Features, Comparisons, and Objectives

The first problem in any classification task is to choose how to represent the data. Texture is, by definition, not a single pixel property: only a chunk (or cell) of adjacent pixels can have texture. The natural thing is to somehow break the image into cells, which will be given individual texture assignments. These cells of pixels must then be transformed into some feature vector that is useful in characterizing texture. Next, some measure for comparing the dissimilarity between these vectors is needed for clustering purposes. Finally, we need some objective function that measures how good a set of texture assignments is, based on the dissimilarity measures. We shall see later how we use the objective function to iteratively improve the set of assignments. Hofmann et al. [5] offer straight-forward and effective methods for solving these problems. Below is a description of how we implemented their approach.

2.1 Features

We begin by superimposing a rectangular lattice of points on the image.¹ These points serve as the center coordinates of the cells that will serve as our basic unit.

¹We use 20x20 lattices on 400x400 pixel images in almost all of our tests.
Figure 2-1: A typical five texture image with the lattice points and their cells overlayed.

We will concern ourselves with assigning textures only to these cells. The cells are defined as non-overlapping rectangular regions around each lattice point as depicted in Figure 2-1. For the remainder of this paper, cells shall be referenced by the lattice points that define them.

Next, we convert each cell of pixels into a feature vector. We know from our own biology that there are neurons (known as simple cells) in the V1 area of the brain that respond to different combinations of spatial frequency and orientation in the visual field. This conjures the idea of fourier analysis. However, unlike a complete fourier transform, simple cells are locally tuned; that is, they only respond to stimuli local to a small part of the visual field. It turns out that a bank of Gabor filters at different scales and orientations can approximate the simple cell receptive fields [3]. Furthermore, Gabor filters have been used successfully in a variety of vision applications [7] [9] [5]. We decided to use 12 filters at four orientations and three frequencies as depicted in Figure 2-2 and defined as:

\[
GF_r(\vec{x}, \sigma_r, \omega) = \frac{1}{\sigma_r \sqrt{2\pi}} e^{-\vec{x}^2 / 2\sigma_r^2} e^{i\omega \vec{x}},
\]

where \( \sigma_r \) is the standard deviation of the gaussian portion of the filter and depends
on the frequency $\omega$. We used $\omega/\sigma_r$ combinations of 10/5, 20/1.25, 40/.3125, where the units are pixels.

Convolving each of the 12 filters with the image leaves us with 12 filter response images, $I_r(\vec{y})$ for $r \in [1, 12]$ and $\vec{y} \in m \times n$, where $m$ and $n$ are the dimensions of the image. At each lattice point we compute the (gaussian) weighted histogram of its cell pixels’ values for each of the 12 channels.

$$f_r(b, \vec{x}) = \sum_{\vec{y} : \vec{y} \in C(\vec{x}), I_r(\vec{y}) \in R(b, r)} G_r(||\vec{x} - \vec{y}||)/ \sum_{\vec{y} : \vec{y} \in C(\vec{x})} G_r(||\vec{x} - \vec{y}||),$$

where $G_r(x)$ is a gaussian filter with mean zero and standard deviation $\sigma_r$ (the same parameter used in the gaussian part of $r$th gabor filter function), $C(\vec{x})$ is the set of pixels in the cell centered on lattice point $\vec{x}$, and $b$ is a bin in the histogram. $R(b, r)$ is the range of values that fit into the $b$th bin for filter response $r$. We used 40 equally sized bins dynamically adapted to the response range of $I_r$. So, for example, $f_1(\ast, \vec{x})$ might look like the histogram in Figure 2-3. This tells us that lattice point $\vec{x}$’s cell response to filter one strongly concentrated around the value 160.

The idea behind using such a histogram is that the feature vector at each lattice point should be representative of its entire cell, not just the pixel it happens to lie on. So we empirically find the distribution of filter responses in the cell.
2.2 Comparisons

Now that each lattice point has been characterized, a difference metric, \( D(\bar{x}, \bar{y}) \), is needed to compare pairs of lattice points. The \( \chi^2 \) statistic is used for this purpose. We start out by computing the difference, \( D_r(\bar{x}, \bar{y}) \), for each channel, \( r \).

\[
D_r(\bar{x}, \bar{y}) = \sum_b \left( \frac{f_r(b, \bar{x}) - f_r^{mid}(b, \bar{x}, \bar{y})}{f_r^{mid}(b, \bar{x}, \bar{y})} \right)^2, \quad \text{where } f_r^{mid}(b, \bar{x}, \bar{y}) = \frac{f_r(b, \bar{x}) + f_r(b, \bar{y})}{2}.
\]

Next, we combine the metrics for each frequency channel, \( r \), by simply adding them: \( D(\bar{x}, \bar{y}) = \sum_r D_r(\bar{x}, \bar{y}) \). If we made such comparisons between every possible pair of lattice points, we would end up with a symmetric matrix, \( D \), that grows exponentially with the number of lattice points. To avoid this growth problem we choose a unique, fixed-size group, or “clique,” of lattice points for every lattice point. Instead of comparing each lattice point with every other lattice point, we shall only compare each point to every point in its clique. Members of a lattice point’s clique are its four (three or two for border points) adjacent lattice points along with a number of

\(^2\chi^2 \) was chosen over other metrics due to its superior empirical performance [5].
randomly picked lattice points.\footnote{For our usual 20x20 lattice we chose to have cliques of size 20, 5\% of the total number of lattice points. This low number of clique-members was chosen to speed up the testing process.} We shall see later that this can cause problems as the choice of clique-members may not be representative of all the textures in the image.

2.3 Objectives

The goal of the segmentation process is to find the optimal set of texture assignments for the lattice points. We can transform this assignment problem into a minimization problem by formulating an objective function, $H$, that measures the “badness” of a set of assignments. The objective function used here is the weighted sum of the average intra-texture pairwise difference.

Some notation needs to be introduced before we can formulate $H$ precisely. Let $L$ be the set of lattice points and $K$ the set of textures. Next we need some way of indicating which texture a lattice point has been assigned to.

$$M_x(k) = \begin{cases} 1 & \text{if lattice point } x \text{ is assigned to texture } k \\ 0 & \text{otherwise} \end{cases}$$

Let $P(k) = \sum_{x \in L} M_x(k)$ be the number of lattice points that are members of texture $k$;

$$B(k) = \sum_{x \in L} \sum_{\tilde{y} \in Z(x)} M_x(k) M_{\tilde{y}}(k) D(x, \tilde{y}),$$

where $Z(x)$ is the set of points belonging to $x$’s clique. $B(k)$ can be thought of as the sum of differences between lattice points of texture $k$ and their clique-members of texture $k$. Lastly, $Q(k) = \sum_{x \in L} \sum_{\tilde{y} \in C(x)} M_x(k) M_{\tilde{y}}(k)$: the number of pairs $(i, \tilde{j})$ where $\tilde{j}$ is in $i$’s clique and both of them are assigned to texture $k$.\footnote{$Q(k)$ can also be thought of as the number of non-zero terms that are multiplied in $P(k)$.} The hope is that $B(k)/Q(k)$ will yield a good approximation of the average intra-texture difference. That is the average difference between all pairwise combinations of lattice points.
assigned to the same texture. Now, the objective function can be defined like so:

\[ H = \sum_{k=1}^{K} \frac{P(k)B(k)}{Q(k)}, \]

where the approximated average intra-texture difference is scaled by the number of points assigned to each texture. This gives more weight to the differences of textures that claim more image space. Minimizing this function is equivalent to maximizing the intra-texture compactness. Furthermore, Hoffman et al. [5] have shown that it has the useful property of shift invariance under affine transformation of the difference scale. This is a good thing because it means the quality of a set of assignments will be judged based on the relative magnitudes of their difference metrics, not the absolute values.

2.4 Next

The next step is to search for a set of texture assignments for the lattice points that minimizes the objective function \( H \). Assuming that the image model described in this chapter is useful, this set of assignments will correctly segment the image.

The search space of this problem is the set of all possible assignments. Given a typical 20×20 lattice and, say, five textures, there are \((20 \times 20)^5 = 1024 \times 10^{13}\) states to search, much too many for a systematic search. The next obvious method to try might be a greedy search. Unfortunately, there are too many local minima in the objective function for this approach to be useful. Instead, we will investigate the use of simulated annealing, and modifications thereof, to solve this problem.
Chapter 3

Annealing and Variations

In this chapter we describe the traditional simulated annealing algorithm along with the variations with which we experimented.

3.1 Simulated Annealing (SA)

Annealing is the physical process of melting some material and then cooling it slowly so as to allow the molecules to arrange themselves in an energy-minimizing, frozen state: ideally a crystal formation free of dislocations. Successful results depend on an appropriately slow cooling process. The idea is to keep the material in thermal equilibrium as the temperature is lowered. If the energy of the system is allowed to drop too quickly, small groups of molecules may find themselves in stable arrangements that are not globally optimal. This concept has been generalized to solve optimization problems by making the following analogy:

- *molecules* are variables in the optimization problem.

- *molecular states* are the values that each variable can take.

- *system energy* is an objective function that depends on some relation between the variables.

- *annealing the system* is an iterative process in which variables are repeatedly
given a chance to change their value based on the effect this has on the value of the objective function, with the overall goal of minimizing it.

- *temperature* becomes a randomizing parameter that dictates how likely variables are to be given values that don’t minimize the objective function at each iteration.

In our texture classification problem the molecules are the lattice points, \( X = \{ \vec{x}, \text{the coordinates of points in the lattice} \} \), which can take on values \( k \in K \), the set of textures. The energy function is \( H \), defined in the previous chapter. A Markov chain is used to implement the annealing process. Each state in the chain is a unique set of texture assignments for all the lattice points in the image. The only transitions allowed are those that change exactly one or zero assignments. In this way, we can iterate through the image, changing at most one lattice point’s assignment per cycle.

Let us use a notation similar to that from the last section: \( M_t \) is the state of the system at time \( t \), such that

\[
M_{t\vec{x}}(k) = \begin{cases} 
1 & \text{if lattice point } \vec{x} \text{ is assigned to texture } k \text{ at time } t \\
0 & \text{otherwise}
\end{cases}
\]

A visitation schedule, \( v(t) \to \vec{x} \), is used to dictate which lattice point, \( \vec{x} \), may switch assignments at each iteration \( t \). We used a simple visitation schedule that scans through the lattice points top to bottom, left to right. In section XX we discuss some other possibilities. The only constraint on the visitation schedule for annealing to be successful is that all points must be visited an infinite number of times given an infinite amount of time [4].

At each time step we have the same set of transitions \( S_{t\vec{x}}(k) \forall \vec{x} \in L, k \in K \), which returns a new state \( M_{t+1} \) in which all lattice points retain the same assignment as at time \( t \) except \( \vec{x} \) which is assigned to texture \( k \). Now we can define the probability of
each transition as follows:

\[
P(S_{x}(k)) = \begin{cases} 
  \frac{e^{-H(S_{x}(k))/T}}{\sum_{k \in K} e^{-H(S_{x}(k))/T}} & \text{if } x = v(t) \\
  0 & \text{otherwise}
\end{cases}
\]

where \( H \) is the energy function described in the previous section.

The probability of choosing a transition, or texture assignment, for the current lattice point, \( v(t) \), depends on how low the resulting state’s energy is compared to those of other transitions. That is, texture assignments that lower the system’s energy more (or increase it less) will be chosen with a higher probability than others. The temperature parameter, \( T \), regulates the relation between energy difference and probability difference. For example, say there are two valid transitions \( S_{100} \) and \( S_{50} \), where the former results in an energy level of 100 and the latter of 50. Figure 3-1 is a plot of \( P(S_{100}) \) as a function of \( T \). As \( T \) climbs to infinity \( P(S_{100}) \) goes to .5, and since \( P(S_{50}) = 1 - P(S_{100}) \) the difference in probability goes to zero. As \( T \) drops, the probabilities are polarized and the difference between them goes to one. To summarize, at high temperatures transitions that are not as immediately beneficial are more likely to be taken, whereas at low temperatures immediate benefits are given more weight. Looking at the extremes, when the temperature is infinite all transitions are taken with equal probability; as the temperature goes to zero the process turns into hill-climbing. The main concept behind annealing is that if we lower the temperature slowly enough we will not get stuck in local minima, as we would with a purely greedy approach, but will find the global minimum (or something negligibly close to it).

The process of choosing transitions based on the probability formula above is known as **Gibbs Sampling** [4]. Given a constant temperature, \( T \), this process will eventually converge towards the Gibbs Distribution in which the probability of being in state \( M \) at any given time is defined in the familiar way:

\[
P(M) = \frac{e^{-H(M)/T}}{\sum_{M' \in \mathcal{M}} e^{-H(M')/T}},
\]

where \( \mathcal{M} \) is the set of all possible states.
Figure 3-1: $T$ vs. Probability of $S_{100}$

There are other methods for simulated annealing as well, the prominent of which are mentioned below.

### 3.1.1 Related Work on Simulated Annealing

The concept of simulated annealing was first embodied in the Metropolis algorithm [11], which works as follows. For each point on the visitation schedule, choose a random transition $S$. If the state resulting from taking this transition, $S(M)$, has less energy than the current state, $M$, take it. Otherwise, take it with probability $e^{-\Delta E/(cT)}$ where $\Delta E = H(S(M)) - H(M)$ and $c$ is a constant.

In the time since the Metropolis algorithm was introduced, much progress has been made in speeding up the annealing process. Improvements have usually involved using a different distribution with which to sample the state space. Initially, simulated annealing was performed using something similar to Gibbs sampling. As mentioned above, Geman and Geman [4] showed that to guarantee convergence in probability to the uniform distribution on the global minima using Gibbs sampling, it is necessary to use a logarithmic annealing schedule $T(t) = c/(1 + \log t)$. The annealing schedule dictates how quickly the process settles down into its steady state, and logarithmic
Figure 3-2: The Gaussian (dashed) and Cauchy (solid) distributions. The “fatter” tails of the Cauchy distribution allow egress from local minima at lower temperatures (i.e. when the distribution is narrower).

Szu and Hartley [13] found that using the Cauchy distribution to sample states allowed them to speed up the annealing schedule to \( T(t) = c/(1+t) \) without sacrificing the convergence guarantee. Intuitively, this is due to the “fatter” tail of the Cauchy distribution in comparison with the Gibbs distribution (see Figure 3-2), which allows the system to escape local minima at lower temperatures [6]. This method has been dubbed Fast Annealing.

Inefficiencies arise when variables become multidimensional, where values in each dimension inhabit a different range and exhibit different behavior in relation to the objective function. For this type of problem Very Fast Annealing was created, in which each dimension has its own temperature and range variables. These modifications allow for an annealing schedule of \( T_i(t) = T_0e^{-t^{1/D}} \), where \( D \) is the number of dimensions and \( i \in (1, ..., D) \).

More recently, work has gone into speeding up the annealing process by replacing slow stochastic operations with quick deterministic ones, yielding methods that come under the category of deterministic annealing. One specific approach uses the mean
field approximation to calculate the free energy of the system, which is minimized by
the Gibbs distribution, for the purpose of quickly finding the set of assignments that
minimizes this energy [5].

The modifications to simulated annealing proposed in this paper are orthogonal
to the improvements described above. So, for the sake of simplicity, we shall only
compare our results to those of traditional simulated annealing.

3.2 Variations

Along with traditional simulated annealing experiments, we performed a number
of experiments using variations on the basic algorithm above in the hope of better
understanding annealing behavior and improving performance. Below we describe the
variations with which we experimented, the motivation behind them, and a summary
of our results. In the next chapter we describe our results in detail.

3.2.1 Single, Low Temperature Annealing (SLTA)

Although it makes intuitive sense to start with some high temperature and bring
it down slowly so the system can settle into its global minimum, it is wise to use
a standard approach to derive a baseline with which to compare our results. We
decided to perform Gibbs Sampling using a single, low temperature until the system
settled into its steady state. Technically, this is not annealing at all because the
temperature never changes, but the idea is to combine hill-climbing with a slight
amount of randomness to help the system avoid getting stuck in local minima.

We found that this method works well for images with two textures, but perfor-
mance drops quickly with more textures. Intuition for understanding these results
can be arrived at by thinking about the complexity of the solution space. If there are
$k$ textures and $l$ lattice points then the total number of possible states is $l^k$. With
more states comes the possibility for more local minima. Since using a fixed, low
temperature provides only a limited ability to escape these minima, the chance of
getting caught in one increases, possibly exponentially, with the number of textures.
3.2.2 Binary Decomposition (BD)

Ideally, we would like to create a method that combines the speed and accuracy of SLTA on two texture images with good performance on more textures. In this spirit we experiment with a Binary Decomposition algorithm. This works by treating the image as if it contains only two textures and running SLTA on it, producing two sub-regions corresponding to each of the two postulated textures. We can recursively apply this process to the sub-regions as many times as we want, creating an arbitrary number of disjoint texture regions.

This approach raises two difficulties which are related. The first, how do we choose which region to decompose? And the second, how do we know when to stop the decomposition process? An absolute measure of fitness for a region would allow us to decompose those regions that were not fit enough, and stop the entire process when no such regions existed. We came up with a heuristic, $H_{\min}(R)$, which estimates the minimum energy of a region, $R$. Our absolute measure of fitness is the simple test $H(R) \leq H_{\min}(R) + \epsilon$, where $\epsilon$ is a small error term. If the test is positive then we assume that this region is one homogeneous texture, and we leave it intact. The heuristic we use is based on the assumption that most lattice points will be the same texture as their direct neighbors. That is, we expect more lattice points to be positioned inside a homogeneously textured region than on the border of one, and being on the border is more likely than being surrounded by points of a different texture. Recall that our energy function $H$ returns the sum of average intra-cluster difference scaled by the number of points assigned to each cluster. We can estimate the optimal average intra-cluster difference $D(\bar{x}, \bar{y})$ as the average difference between each lattice point and its direct neighbors. Multiplying our estimate of the intra-cluster difference by the total number of lattice points gives us an estimate of the minimum energy. We can combine the averaging and scaling like so:

$$H_{\min} = \sum_{\bar{x} \in L} \frac{\sum_{\bar{y} \in N} D(\bar{x}, \bar{y})}{\sum_{\bar{y} \in N} 1},$$

where $N$ is the set of $\bar{x}$'s direct neighbors.
The most obvious drawback of BD comes from the fact that we are building a binary decision tree where the branches separate the regions and the leaves are unique texture assignments. If a lattice point is assigned to the wrong region at a higher branch it will never be correctly classified because its correct assignment will be at a leaf on the other side of the tree. Another way to think of it is that once points are separated into two regions, no point from one region will ever be assigned to the same texture as a point from the other region. This problem does not arise in SA because each lattice point is given the chance to switch to any assignment every time it is visited.

3.2.3 Dynamic Temperature Selection (DTS)

Fundamental to annealing is the use of a temperature, to introduce randomness, that is lowered in a controlled manner. It is no surprise, then, that the fundamental difficulty with annealing is choosing the temperature at each iteration. A policy governing the change in temperature is called an annealing schedule. It has been shown [4] that to guarantee convergence in probability to the uniform distribution on the global minima using Gibbs Sampling, we need a logarithmic annealing schedule $T(t) = \frac{c}{1 + \log t}$. This is considered very slow and most users of simulated annealing run their applications with a faster schedule that they have found reliable for their purposes. We found that it is possible to choose the temperature adaptively at each step in the annealing process based on the history of the system energy, as well as a few other parameters. This Dynamic Temperature Selection (DTS) method is a general solution to the annealing schedule problem, and may be widely applied.

We took our inspiration from the material sciences from whence we borrowed annealing in the first place. It is well known that when annealing a physical system, the temperature must be lowered most slowly as the system transitions from one phase to another (e.g. from liquid to solid). It is during these periods that the system's energy drops the most precipitously, and so care must be taken to avoid the formation of dislocations (equivalent to getting caught in local minima). It is difficult to apply this knowledge directly to simulated annealing problems because if there are
any well-defined phase changes, they are not known \textit{a priori}. However, we may apply the knowledge that the energy should not drop too quickly. We do this by choosing a desired rate of energy decrease and then enforcing it.

We define the desired rate of energy decrease like so:

\[ r_{\text{desired}} = \frac{H_{\text{init}} - H_{\text{min}}}{\delta} \]

where \( H_{\text{init}} \) is the initial system energy, \( H_{\text{min}} \) is the minimum energy heuristic, introduced earlier, applied to the whole lattice, and \( \delta \) is a constant that can be adjusted based upon the complexity of the solution space. A more complex solution space will tend to have more local minima and will require more careful annealing. In the next chapter we demonstrate the results of using different values for \( \delta \).

Since the rate of energy decrease is influenced by the temperature, we can choose the temperature at each step based on the current rate: \( T(t+1) = T(t) - \alpha (r_{\text{desired}} - r_{\text{current}}) \). The feedback loop adjusts the temperature dynamically so as to keep the energy dropping at the desired rate. This method would work for physical systems with well-defined phase changes, but it has the power to deal with more complicated systems. This power is derived from the fact that the temperature is adapted at every iteration of the process. Taking this concept to its extreme, it is possible to maintain a separate temperature parameter for each molecule. For our purposes, however, we found that a single, changing temperature works quite well, with accuracy on par with traditional annealing, along with a huge drop in the time required to reach equilibrium.
Chapter 4

Experimental Results

In this section we present and compare the results of using our four variations: single, low temperature annealing (SLTA), traditional simulated annealing (SA), binary decomposition (BD), and dynamic temperature selection (DTS), on different classes of data. The majority of the experiments were performed on images composed of textures taken from the Brodatz texture library [1]. They are all made of two, three, five, and seven textures arranged in the patterns shown in Figure 4-1. Each algorithm was run ten times on five different images of two, three, five, and seven textures for a total of 50 tests per algorithm/number of textures combination. At the end of this chapter we present some results using real-world images as well.

We begin all our tests by giving each lattice point a random assignment $k \in K$, the set of textures. $K$ is user-specified, but there are methods for automatically choosing a good value of $K$ [5] [7] [10] that we do not discuss here. The set of initial assignments is then iteratively updated by whatever algorithm is being tested until a

Figure 4-1: The patterns used to create images with two, three, five, and seven textures.
stable\(^1\) set of assignments is reached. All algorithms use a top-to-bottom, left-to-right visitation schedule to scan over an image’s lattice points.

Algorithm performance is measured by how accurately images are segmented. Accuracy is defined as the fraction of correctly assigned lattice points. This measure is only attainable in artificial images where the correct segmentation is known in advance. For real-world images the segmentation is subjective and no hard measure of performance is used.

### 4.1 Single, Low Temperature Annealing (SLTA)

We begin by using the simplest algorithm in our bag: SLTA. The results on two-texture images are quite good. That is, multiple passes with a single low temperature are sufficient for the system to settle into a consistent low-energy state, there are no blatantly wrong segmentations, and there are very few misclassified lattice points. Figure 4-2 shows one typical test image along with the segmentation mask. This particular example was chosen because of the obviously misclassified “speckles.” At first we considered these speckles to be the result of noisy data, but upon further inspection we found that the majority of these misclassified lattice points (e.g. \(\bar{x}\)) have few or no clique-members belonging to the texture (e.g. \(k\)) to which they are assigned. This means that they make no contribution to the system energy level since \(B(k) = \sum_{\bar{x} \in L} \sum_{\bar{y} \in \mathbb{Z}(\bar{x})} M_{\bar{x}}(k) M_{\bar{y}}(k) D(\bar{x}, \bar{y})\) remains the same whether \(\bar{x}\) is assigned to \(k\) or some other texture. So, if assigning them to their correct texture raises the system energy slightly, they will most likely be assigned to the wrong texture, \(k\).

The problem boils down to having a clique whose members are not evenly spread amongst all the textures. This is more of an issue for images with a greater number of textures, as it becomes less likely that the randomly chosen clique-members will be representative of all the textures. For later experiments, we forbid assignments that are not shared by at least two clique-members. Since cliques are composed of a lattice point’s direct neighbors, along with a number of random points, this modification

\(^1\)Stable, here, is used to mean that the system energy is no longer changing appreciably.
biases points towards their direct neighbors' assignments. This is a good thing if an image is mostly composed of chunks of homogeneously textured regions.

Figure 4-2: The image on the left is given the segmentation pattern in the 20×20 grid on the right.

So far no evidence has been given to show that $H$, our energy function, is at all indicative of a good segmentation. Figure 4-3 shows the strong correlation between the two as SLTA is run on a three-texture image. This empirical evidence justifies our choice of features, difference metric, and energy function.

The curve plotted in Figure 4-3 is the general shape of the system energy versus time at any constant temperature. But, within this general shape there is a lot of variation depending upon the system and the temperature. This will be more apparent in the next section.

As the number of textures in the image increases, SLTA's performance decreases. Figure 4-4 shows a typical segmentation of a five-textured image: three distinct regions are identified instead of five. We shall use SLTA's performance as a baseline with which to calibrate performance in the next few sections.
Figure 4-3: The original image and its segmentation after one, two, three, four, and five epochs. The bottom row contains the corresponding system energy.
4.2 Simulated Annealing (SA)

Traditional simulated annealing works well with all the artificial images in the test suite. There was a decrease in performance as the number of textures increased, but not nearly as much as with SLTA (see Figure 4-5). The Kolmogorov-Smirnov test was used to make sure that the differences measured are significant.²

A non-parametric test was preferred over, say, a Gaussian one because the distribution of the various results is unknown, and clearly not Gaussian. There are certain values at which the system energy tends to get stuck. These correspond to local minima in the $H$ function, which in turn depend on the particular image being processed. Figure 4-6 contains histograms of the final energy reached using SLTA and SA. We see a few strong peaks for two-texture images, and an increased number of peaks as the number of textures increases. This is because the increased complexity of the solution space creates more minima in which the system can get trapped. The histograms show that SA copes better with the increased complexity than does SLTA, resulting in generally lower system energy levels.

The advantage of SA over SLTA comes from the gentle drop in temperature and

²All differences between results’ distributions mentioned in this paper can be assumed significant according to Kolmogorov-Smirnov test to the .05 level, unless otherwise stated.
the consequent gentle drop in system energy. Figure 4-8 illustrates a typical example of this: the system finds its equilibrium over 42 epochs\(^3\), as opposed to the typical nine using SLTA. As mentioned in the previous chapter, guaranteeing good performance means using a slow annealing schedule. The purpose of the slow schedule is to ensure that the system energy does not fall too quickly. However, we found that this could be guaranteed by choosing an image-specific schedule with the “right” sequence of temperatures. Figure 4-7 contrasts the energy curve resulting from this customized schedule with the general, slow schedule that works well for most images most of the time. Both resulted in an accuracy of about 75%. Using a customized schedule allows us to speed up the annealing process by about 25% in this case. However, choosing a schedule by hand for each image is not a practical solution. This is the motivation behind DTS, a method for creating a customized annealing schedule on-the-fly.

This is what we see here: 42 epochs were used to attain these results, as opposed to the nine epochs needed by SLTA to attain the results described in the last section. Furthermore, the fixed annealing schedule we came up with does not always result in an accurate segmentation; the performance varies with the system image and the

---

\(^3\)We found that using the annealing schedule \(T = [1.1, 1, .8, .6, .4, .2, .1]\), where each temperature is maintained for six epochs, to be most effective.
Figure 4-6: These are histograms of the final system energy reached, using SLTA and SA on images with two, three, and five textures.
random starting state. We try to address these problems in the next two sections.

4.3 Binary Decomposition (BD)

Binary Decomposition takes the speed and efficacy of SLTA on two-texture images and tries to apply it to images with more textures. Experiments demonstrated the usefulness of using the $H_{\text{min}}$ heuristic in choosing which region to decompose next; there were extremely few situations in which a homogeneous region was chosen for decomposition. Figure 4-9 shows how close $H_{\text{min}}$ is to the actual minimum energy.\(^4\) Unfortunately, the heuristic was not accurate enough on an absolute scale to be useful in determining when to stop the decomposition process. Varying the error term, $\epsilon$, helped the algorithm stop at the right time in some cases, and made it stop too early in others. So, we reverted to the usual practice of specifying the number of textures in the image by hand.

As expected, BD performs identically to SLTA on two-texture images since it is

\(^4\)We are assuming that the true minimum energy is achieved when images are segmented correctly.
Figure 4-8: A plot of system energy versus time using SA on a five-texture image, and the resulting segmentation. The temperature has been scaled up for easy comparison. It steps through the values 1.1, 1, .8, .6, .4, .2, .1.

Figure 4-9: A plot of the average ratio between the heuristic estimate of the minimum energy and the actual minimum energy for 2, 3, 5, and 7 texture images. All values are very close to one, indicating a good estimate.
doing the same thing. With five textures, there is a marked improvement over SLTA as shown in Figure 4-10. However, BD’s performance is, on average, below that of SA for three, five, and seven textures, and is insignificantly better than SLTA’s at seven textures. Examining the segmentation results reveals that chunks of homogeneously textured regions are often misclassified. This turns out to be the result of error propagation in the binary tree. That is, the correct segmentation at any given node in the tree depends on the correctness of its ancestors. We can see this in Figure 4-11, which shows the two decompositions in a three-texture image. The error in the first decomposition prevents the subsequent one from correctly classifying the three chunks on the bottom right. These results suggest that the minimum-energy segmentation using a texture set, $K$, smaller than the true number of textures does not always separate the textures cleanly into $K$ groups, and increasing $K$ does not simply add more partitions, it changes the previous optimal ones.

The speckle problem experienced using SLTA is even more of an issue using BD because only those clique-members that are in the same region as the lattice point in question are used to calculate $H$. This means that segmentations deeper in the tree are more inaccurate, and, consequently, performance on images with more textures
Figure 4-11: The segmentation on the left is the result of one level of binary decomposition. The one on the right is the result of a second decomposition. Notice the originally misclassified region on the bottom right cannot be corrected. It is modified slightly by light noise reduction performed between decompositions.

is worse, which is why performance is near that of SLTA on seven-texture images.

4.4 Dynamic Temperature Selection (DTS)

The first noticeable difference between DTS and the previous techniques is in the shape of the energy curve. Instead of a series of dips, the energy falls in a linear manner until the minimum temperature is reached, at which point the energy flattens out at its minimum value. Figure 4-12 illustrates an example of this along with a plot of the temperature at every point in the process, which is pretty chaotic. The wild variations in the temperature indicate that different lattice points are losing energy at (wildly) different rates throughout the process. Adapting the temperature a little for each lattice point gives us much greater control over our process in which all the variables are not homogeneous as they would be in a physical system. One group of lattice points may be almost optimally assigned (demanding a lower temperature), while another could be far from its minimum energy state (requiring a higher temperature). This is analogous to a physical system in which there are different materials, with different melting/boiling points. A good annealing schedule would have to take into account the state of each material at every point in the process. It is clearly not possible to create such a schedule a priori if the characteristics of the materials are not known. This is exactly the case in our texture problem: there is no way of
knowing how each lattice point will behave in advance. So, the dynamic approach makes most sense.

Looking at a smoothed version of the temperature plot tells us the overall state of the system. In these data there appears to be one distinct “phase change,” during which the temperature decreases minimally, between epochs 2.5 and 6. A non-adaptive approach would have no way of predicting these in advance and would have to use a slow (perhaps even logarithmic) schedule to guarantee proper handling of these phase changes.

DTS' performance is dependent on the desired energy drop rate, \( r_{\text{desired}} = \frac{H_{\text{init}} - H_{\text{min}}}{\delta} \). Choosing a rate that is too high is effectively the same as using SLTA, as the algorithm will choose the lowest possible temperature in its effort to maintain the drop rate. Choosing a drop rate that is very slow will give us the best accuracy, but the algorithm could take just as long as SA. Figure 4-13 shows the accuracy of DTS with different values of delta. Once again, all results with two textures were close to perfect, but the difference between using \( \delta = 3 \) and \( \delta = 5 \) is significant for more texture images. The differences between \( \delta = 5, 7, \) and \( 9 \) are not significant given the number of data points acquired. Furthermore, the difference between the best DTS accuracy (using \( \delta = 5, 7, 9 \)) and SA accuracy is also negligible, so no higher values for \( r_{\text{desired}} \) need be examined. These results are significant because we were able to attain the equivalent performance of 48 epochs of SA using only 13 epochs of DTS.

### 4.5 Real-World Images

Performance on real-world images can be difficult to judge. While the segmentation for some images is obvious, others are quite ambiguous. Another difference between the artificial images we used in the previous sections and real-world images is that the scale of texture varies more widely in the real-world. This can wreak havoc with the results if the features are not adjusted accordingly.

DTS was first tested on simple images. Figure 4-14 shows the segmenting results on an image of a hallway and an image of a living room with a couch and table in
Figure 4-12: A typical energy curve using DTS, and the corresponding temperature function used to generate it. The bottom plot is the temperature smoothed with a Gaussian.
Figure 4-13: Performance plots for 2, 3, 5, and 7 texture images using DTS with $\delta = 3$ (solid), 5 (dashed), 7 (dash-dotted), and 9 (dotted).

front of a window. The hallway image is segmented nicely into wall and not wall, and it is easy to imagine an indoor mobile robot using this information to correct its course.

The picture of the couch seems simpler than it is because both the floor and the table are quite specular, which means they can take on a different object’s texture depending on the angle of observation. For this image, segmenting with more than two textures did not improve performance.

The last real-world image is of a residential street (see Figure 4-15). Although there appears to be a fair bit of noise in the segmentation, careful examination reveals that there are three distinct and sensible regions. The sky is one region (along with a smooth piece of the road). Next, there appears to be a region for the sides of the houses, which are composed of horizontal slats. Lastly, there is a window region, which also encompasses regions of (what appear to the author to be) unclear texture.
Figure 4-14: Images of a hallway and living room segmented by DTS

Figure 4-15: The segmentation of a complex image of a residential street.
Chapter 5

Discussion and Conclusion

This section presents some observations and issues which arose during the experimentation, as well as proposals for future work in the area of image segmentation using simulated annealing.

5.1 Visitation Schedules

The visitation schedule dictates how often each lattice point is given a chance to change assignments. Originally, experiments were performed with a random schedule, but with 400 lattice points it took an extremely long time for every point to be visited even once. The next schedule we tried (and the one we ended up keeping) consisted of scanning the image point-by-point from top-to-bottom, left-to-right. Although this seemed to work well we tried a variant of this: scanning the image by visiting every alternate column. We did not perform a formal analysis of the results, but we observed that this schedule took longer to settle down. We believe that the column-by-column schedule is more expedient than the skip-a-column schedule because the points in each column affect the probability distribution of the points in the next column. In this way, a sort of chain reaction occurs, speeding things along. For example, say column $A$ and $B$ are adjacent and both the same texture. If column $A$ is largely classified as texture 1, then it is more likely that points in column $B$ will be classified as texture 1, since every point in column $B$ has a clique-member in column
A to which it is quite similar. The skip-a-column schedule prevents this propagation of information.

We did not examine the visitation problem in detail, and there may prove to be a more expedient schedule than the one used here. For example, lattice points could be chosen dynamically (in conjunction with DTS) based on the temperature at which they last transitioned: a group of high temperature points are probably further away from their optimal configuration and could be visited more often. Work on this topic speed up the annealing process even further.

5.2 BD

The problem with Binary Annealing is that errors made early on in the process are never corrected. One possible solution is to perform some regular annealing on the BD results in the hope that, once given a chance, the misclassified lattice points will choose the right assignment.

Another solution could be to allow updates to all lattice points at each stage of the decomposition. This is, in effect, regular annealing while increasing the assumed number of textures periodically. Below is the pseudo-code for this algorithm.

\[
\begin{align*}
\text{for } k = 2 : K \\
\text{ % assume there are } k \text{ textures in the image.} \\
\text{ find region } R \text{ such that } H(R)/H_{\min}(R) \text{ is greatest;} \\
\text{ randomize the assignment of all lattice points in } R \text{ across } k \text{ values;} \\
\text{ perform some variant of annealing on the entire image assuming } k \text{ textures} \\
\text{ \quad \rightarrow } k \text{ texture regions;} \\
\text{end;}
\end{align*}
\]

A third possible improvement for BD could come from using a more sophisticated annealing method. SLTA has proven to be less than robust, and SA is slow, so the obvious thing would be to use DTS to regulate BD's annealing schedule. Ideally, this would benefit from the robustness of DTS and the speed of BD.
5.3 Speckles

As mentioned in Section 4.1, “speckles” of misclassified points in a field of correctly classified points are quite common. This is due to cliques whose members do not fairly represent all the textures. As a result, the system energy can sometimes be “tricked” into falling by assigning a lattice point to a texture whose members it has no way of comparing with itself. The comparison is done by using the difference metric $D(\vec{x}, \vec{y})$ where $\vec{x}$ is the lattice point in question and $\vec{y}$ is one of its clique-members belonging to the texture of comparison. If there are no clique-members belonging to a given texture, $k$, then the system assumes a difference of zero and this measurement does not affect the energy, $H$. This is only half the story: we have seen that switching to an unrepresented texture will not increase the energy, but we have not seen how it will decrease the energy. The decrease comes from releasing its assignment to its old texture, $k’$. This occurs when the lattice point appears to be an outlier; the lattice point’s intra-cluster difference, $\sum_{\vec{y} \in Z(\vec{x})} M_{\vec{x}}(k’)M_{\vec{y}}(k’)D(\vec{x}, \vec{y})$, is greater than the average global intra-cluster difference, $\sum_{\vec{x} \in L} \sum_{\vec{y} \in Z(\vec{x})} M_{\vec{x}}(k’)M_{\vec{y}}(k’)D(\vec{x}, \vec{y})/|L|$. This could happen if the point really is an outlier, or if the clique-members are not representative of the texture.

The original objective in creating the cliques was to speed up the annealing process by fixing the number of pairwise comparisons that needed to be done. We want to make sure not to lose this advantage in solving the speckles problem. To this end we decided to use the assumption behind our trusty $H_{\text{min}}$ heuristic one more time. Namely, that a lattice point should belong to the same texture of at least one of its neighbors. Using this restriction in conjunction with DTS improves the accuracy of segmentation for three, five, and seven textures to levels above those of SA, as shown in Figure 5-1. In the same figure are side-by-side comparisons of the segmentation quality with and without the added restriction. The improvement is clear.

Another possible method of solving the speckle problem might be to intelligently choose clique-members. Instead of randomly assigning a clique, perhaps a few prototypical points that seem most representative could be chosen from each texture.
Figure 5-1: Accuracy of regular SA and DTS with the neighbor restriction. Below are two segmentations of the same image. The left segmentation does not make use of the neighbor restriction; the right one does.
Cliques might then have to change members as the system cooled. Besides the obvious advantage of guaranteeing good cliques, this method lends itself to smaller cliques where each member plays a more important part. The main disadvantage of this approach is that differences will have to be calculated on-the-fly for new clique-members. Future work in this area could be fruitful.

5.4 Scale and Features

In Chapter 2 we describe our method for transforming an image into a set of feature vectors. Specifically, we describe how three scales of Gabor filters are chosen based on the lattice size. However, there is no mention of what effect the scale, and therefore lattice size, has on the segmentation process. Clearly, the process depends on having useful features with which to classify the points, and our success has validated our choice of features. However, there are certain textures for which performance is lower. Figure 5-2 contains three troublesome textures. All three of these textures share the characteristic of having a larger scale than fits in a single cell of the 20×20 lattice used here. That is, the texture patterns are on a larger scale than the cells. For example, there are cells in the first image that contain almost all white pixels, while in other cells the pattern is much more “dense.” For the features to be useful, they need to characterize the full texture pattern, and this can only be done if the pattern is contained in each cell. As a result, the lattice dimensions should depend on the texture. In our experiments we hand-picked a cell size that fits most of our texture patterns, but work has been done to automatically find the best features given an image [14]. One method is to choose a combination of filters from a library to represent texture with minimal entropy [2] [16]. Lewicki and Olshausen [8] have come up with a method for finding an efficient set of filters, and use a Bayesian approach to choosing the best combination of these filters to encode an image. These efforts could be integrated with annealing to produce a more general and robust image segmentation algorithm.
5.5 Phase Changes

Using DTS, it was common to find stages in the annealing process where the temperature had to be kept almost constant for the energy to drop at the desired rate. The relation between these stages and phase changes in physical systems has not been studied. Perhaps further work will allow us to generalize more knowledge from the material sciences, and deepen our understanding of optimization problems.

5.6 Image Sequences

The methods for image segmentation discussed in this thesis could be easily extended to work for movies (sequences of images) by replacing the 2d lattice with a 3d one, encompassing all the frames. Segments would now be regions in time and space. For example, footage of a basketball game could be segmented, resulting in one region for the floor, one region for the crowds, and one region for the players (perhaps
identified by the texture of their jerseys or their legs, which are mostly vertical lines).
The movement of different parts of the player segment could be used to distinguish different players.

The difficulty with this extension lies in the fact that the number of lattice points grows exponentially, and so it becomes even more difficult to choose a small number of clique-members to represent them. This is, essentially, the same difficulty that manifests itself in the speckles problem discussed earlier.

5.7 Contributions and Conclusion

We have described a specific method of image texture modeling and some variations on the annealing method of image segmentation. Experiments using traditional simulated annealing (SA), binary decomposition (BD), and dynamic temperature selection (DTS) revealed the weaknesses of each, as well as the frailties of the image model proposed in [5]. SA, though accurate, is very slow. BD is a fast algorithm with promise, but will require some substantial research before it reaches the accuracy level of SA. DTS is quite fast in comparison with SA and performs just as well.
Bibliography


