Approximate inference methods for grid-structured MRFs

by

Keith Battocchi

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

June 2004

© Keith Battocchi, MMIV. 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 20, 2004

Certified by ...........................................................

Leslie Kaelbling
Associate Professor
Thesis Supervisor

Accepted by .........................................................

Arthur C. Smith
Chairman, Department Committee on Graduate Students
Approximate inference methods for grid-structured MRFs

by

Keith Battocchi

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I compared the mean field, belief propagation, and graph cuts methods for performing approximate inference on an MRF. I developed a method by which the memory requirements for belief propagation could be significantly reduced. I also developed a modification of the graph cuts algorithm that allows it to work on MRFs with very general potential functions. These changes make it possible to use any of the three algorithms on medical imaging problems. The three algorithms were then tested on simulated problems so that their accuracy and efficiency could be compared.

Thesis Supervisor: Leslie Kaelbling
Title: Associate Professor
Acknowledgments

I would like to thank Leslie Kaelbling for giving me the opportunity to do this work. I would also like to thank Kevin Murphy for his many helpful comments and suggestions for this document, as well as for providing direction to my research. Finally, I want to thank my family and friends for their constant support.
Chapter 1

Introduction

Graphical models are a widely used tool for analyzing systems in many fields [7]. The nodes in a graph represent variables, which may either be hidden or observed, and the edges between nodes confer conditional probability distributions. Given such a model and the values for the observed variables, the problem is to determine either the most likely assignment of the hidden variables or their probability distribution. For very simple graphical models, exact inference is possible. That is, it is possible to exactly determine the probability distribution of the unobserved nodes. However, the complexity of the algorithms used to exactly compute the distributions goes up exponentially with the number of variables, so that approximate inference is often the only practical solution.

Graphical models are used frequently in diverse fields, from physics to machine vision. Because of their wide applicability, much work has been dedicated to algorithms that approximate the solutions to the probability equations.

In applications that deal with images, a particularly useful type of graphical model, the Markov Random Field, or MRF, is used. Essentially, this model consists of a two-dimensional array of observed nodes (or a three-dimensional array, in the case of volume imaging), representing the pixels (or vowels) of the image, backed by an identically shaped array of hidden nodes, representing the actual state of the system being imaged at that point in space. We assume that our observed pixel values are corrupted to some degree by noise, so that given a particular actual state, there is a
probability distribution for what pixel value we actually observe.

Note that the hidden nodes do not necessarily have the same domain as the pixel values do. Let us take an MRI brain scan as an example. In this case we might have a gray-scale image, with pixel values between 0 and 1, but the hidden variables might take on different classes of brain tissue as values, so that there is some probability distribution for the color of a pixel given that the underlying tissue is gray matter. We then want to determine, given an image, what the most likely underlying tissues are at each point. This is more or less exactly what medical image segmentation is about: determining where in an image various types of tissue are.

It is clear that real world images have constraints other than just those between an observed color and what is actually there. For instance, there is clearly a spatial correlation: if there is a patch of black pixels with one gray pixel amongst them, most likely the gray came about due to noise, and not because there is a tiny amount of a different substance at that point. These constraints are vital to successfully viewing an image, and they are enforced in the graphical model by adding conditional probabilities between close hidden nodes. It is somewhat surprising, but even if a hidden node is only dependent on the direct neighbors to the north, south, east, and west (and up and down, if it is a three-dimensional image), very good de-noising of images is possible.

Finding the exact solution that maximizes the probability of the observed nodes of a MRF takes time exponential in the number of pixels, so for an image of a useful size, approximations need to be made. Fortunately, many algorithms have been developed for doing approximate inference on graphical models. Two important algorithms are the mean field approximation (MF) and belief propagation (BP). It can be shown that on a graphical model with no loops, belief propagation always converges to the correct solution, while the mean field approximation does not [11]. However, on graphical models with loops (which typical MRFs are), neither method is guaranteed to converge to the correct solution, although experimental results indicate that belief propagation generally performs fairly well, and better than the mean field approximation. Weiss compared the results of BP and MF on several graphical
models, and found that BP was consistently better [12].

A relatively new technique for doing approximate inference is graph cuts, which has been gaining popularity for applications from stereo vision to image restoration [9]. This technique has been compared favorably to BP by Tappen et al. in [10]. In Chapter 5, we bridge the work of Weiss and Tappen by comparing MF, BP, and graph cuts on image segmentation problems.

One area where image segmentation is especially important is in analyzing medical images, including MRIs. For example, it is often extremely important to be able to differentiate between different brain structures for studies on brain disease. Figure 1-1 shows a slice of a brain MRI segmented into white and gray matter. Unfortunately, manually segmenting an image is an extremely time consuming process, and with complex structures, even experts in the field can have reasonably large differences in their segmentations. Brain MRIs are generally 256 × 256 × 124 voxels large, and depending on the detail, there could potentially be over one hundred different labels, or tissue types. This makes automatic segmentation an extremely attractive goal. In addition, automatic segmentation could present entirely new opportunities for researchers, since it would be possible to segment an entire brain, whereas manual segmentation is too time-consuming to allow segmenting more than a small region of interest.

Current automatic segmentation programs use a variety of algorithms, but many
scientists in the medical community are unaware of recent advances in the machine vision community. This means that while some programs use the mean field approximation [8], a search through the literature reveals that none use belief propagation to do inference. Of course, since Weiss found BP to outperform MF, it seems like these algorithms would ideally be converted to use BP. Unfortunately, BP has much steeper memory requirements than MF does, and MRIs are extremely large, so that using BP may not be practical. In Chapter 3 we introduce a new method of reducing BP’s memory requirements dramatically that may make the use BP more feasible for these large image applications.

We would also like to use graph cuts for the medical image segmentation problem, but graph cuts only works for specific kinds of data models. We introduce a generalization of graph cuts that should be able to handle medical image segmentation in Chapter 4.
Chapter 2

An Introduction to the Mean Field and Belief Propagation Methods

In this chapter we will describe the basic theory behind the Mean Field and Belief Propagation algorithms. To describe the problem at hand, consider the general idea of a medical image (for instance, a brain MRI). Then, we have an observation $y$ (perhaps a gray-scale image) consisting of a set of independent observations $y_i$ (the pixels or voxels of the image). We then want to determine what tissue class each pixel actually represents in the patient’s body (in the brain scan example, perhaps we want to differentiate between white matter, gray matter, bone, etc.).

We assume that the problem can be represented by a pairwise Markov random field (MRF). Specifically, this requires that the probability of an underlying assignment $x$ is

$$
\Pr(x|y) = \frac{1}{Z} \prod_{(i,j) \in N} \psi_{ij}(x_i, x_j) \prod_i \phi_i(x_i, y_i)
$$

(2.1)

We will now quickly define each term in the preceding equation. The constant $Z$ is merely a normalizing factor so that the probabilities add up to one. The set $N$ contains all pairs of neighbors in the graph. We will generally use the nodes directly to the north, south, east and west (and up and down, in a 3D image) of a given node to define the neighborhood relation. We will also use the notation $i \in N(j)$ to mean $(i, j) \in N$ for a fixed $j$. The function $\psi_{ij}$ is called a compatibility function.
This function describes in some sense the likelihood that pixels $i$ and $j$ take on values $x_i$ and $x_j$. We expect, for instance, that it is very likely that the pixel next to a pixel representing gray matter also represents gray matter. This would correspond to a high value of $\psi_{ij}$. Finally, the function $\phi_i$ encodes the probability that pixel $i$ represents tissue class $x_i$ given that we observed the value $y_i$ at that pixel. We will often treat the observed data as a given, talking about $Pr(x)$, and subsuming $y_i$ into the definition of $\phi_i(x_i)$ for notational simplicity. This defines the basic framework with which we will be working.

### 2.1 The Mean Field Method

In [12], Weiss provides the following sketch of the update rules for the mean field (MF) method. We start by assuming that the probability of an assignment can be completely factored over the nodes of the lattice, that is $Pr(x) = \prod_i b_i(x_i)$ for some parameters $b_i$. These parameters can be acquired by minimizing the mean-field energy

$$E_{MF}\{b_i\} = -\sum_{(i,j)\in N} b_i(x_i)b_j(x_j) \ln \psi_{ij}(x_i, x_j) + \sum_i \sum_{x_i} b_i(x_i)[\ln b_i(x_i) - \ln \phi_i(x_i)]$$

subject to $\sum_i b_i(x_i) = 1$. Setting the derivative to zero gives [12]:

$$b_i(x_i) = K_1\phi_i(x_i) \exp \left( \sum_{j\in N(i)} \sum_{x_j} b_j(x_j) \log \psi_{ij}(x_i, x_j) \right),$$

where $K$ is a normalization constant chosen so that the constraint $\sum_{x_i} b_i(x_i) = 1$ is satisfied.

The MF method consists of defining the beliefs $b_i(x_i)$ in terms of iterating Equation 2.3. The beliefs are initialized to some starting value (often $b_i(x_i) = \phi_i(x_i)$), and the new beliefs at any given step are derived from Equation 2.3. The algorithm terminates when the beliefs converge, or when a set number of iterations is reached (see Chapter 3 for some comments on convergence). In order to run this algorithm, we must store our old beliefs for each state at each vertex in addition to the current
values of these beliefs, requiring the storage of $2nl$ variables, where $n$ is the number of pixels or voxels and $l$ is the number of labels.

### 2.2 The Belief Propagation Algorithm

A derivation of the belief propagation (BP) update rules is too lengthy to include here, so we will instead just give the update equations. The interested reader is referred to [12] for more details. The belief propagation update rules are more complicated than those for mean field, since the current beliefs are not calculated from previous beliefs, but rather from messages being passed from node to node. The messages themselves are updated according to

$$m_{ij}(x_j) \leftarrow K \sum_{x_i} \psi_{ij}(x_i, x_j) \phi_i(x_i) \prod_{k \in N(i) \setminus j} m_{ki}(x_i), \quad (2.4)$$

where again $K$ is a normalization constant.

In essence, each node sends a message to each neighbor telling it how likely it believes the neighbor is to be in each state. In turn, a node calculates its belief from the messages it receives according to the equation

$$b_i(x_i) \leftarrow K \phi_i(x_i) \prod_{j \in N(i)} m_{ji}(x_i). \quad (2.5)$$

Equation 2.4 is the “sum-product” or MPE version of the message update rule, which will lead to beliefs $b_i(x_i)$ that approximate the value of $p(x_i)$ at each node. There corresponding “max-product” or MAP formulation of the message update rules gives beliefs that approximate the simultaneous assignment of all nodes that has the maximum a posteriori probability of occurring. This alternative message update rule is

$$m_{ij}(x_j) \leftarrow K \max_{x_i} \psi_{ij}(x_i, x_j) \phi_i(x_i) \prod_{k \in N(i) \setminus j} m_{ki}(x_i). \quad (2.6)$$

For an analysis of the comparative advantages of the MAP and MPE versions of BP, see Chapter 5.
Because a message needs to be passed from each node to each of its neighbors for each possible label, we require $4nl$ variables to be stored for messages. Because the updated values of the messages leaving a node are calculated based on the values of the messages coming into the node in the previous iteration, we also need to store $4nl$ old messages, for a total of $8nl$ messages total (in the case of a 3D graph, there are 6 neighbors for any interior node, for a total of $12nl$ messages stored). We also have to store the beliefs ($l$ per node), giving a total memory requirement of $9nl$ variables needing to be stored (or $13nl$ for a 3D image). Because $n$ can be quite large in practice ($256 \times 256 \times 124$ is a standard MRI size, yielding an $n$ of over eight million), the amount of storage required is often impractical. For several different techniques for reducing memory requirements, please see Chapter 3.
Chapter 3

Improvements to Mean Field and Belief Propagation

The MF and BP algorithms are quite straightforward to implement, but both (and especially BP) require fairly large amounts of storage space. There are techniques that can be used to reduce the requirements without sacrificing any performance, and there are several more that can be used to trade off space and time performance. We now provide an overview of several available techniques.

3.1 Asynchronous updates

A particularly simple way to cut storage requirements essentially in half is to use an asynchronous update schedule. In the standard synchronous update schedule, we need separate storage space for all old beliefs (in the case of MF) or messages (in the case of BP), and we use this old data to generate all of the new data at once. With an asynchronous update schedule, we generate new data one node at a time, using the most recently available data from the neighboring nodes in the update equations.

As a consequence of the different update schedule, the results of this modified version of the algorithm will obviously not generally be exactly the same as those from the standard version. However, as pointed out in [4, 10] (although these sources only address BP), the algorithm using an asynchronous update schedule typically
Figure 3-1: This figure demonstrates that a lattice is bipartite. White nodes only send messages to black nodes, and vice versa.

converges in fewer iterations that that using synchronous updates. The likely cause for this behavior is also given in [10]: information can travel further with asynchronous updates. This is because, for synchronous updates, the information at that node can only spread to its neighbors in each iteration. Thus, after \( T \) iterations, the information from a given node can only have reached a node \( T \) steps or less away in the lattice. On the other hand, with an asynchronous update schedule, information from the first node goes to the second, then to the third, and eventually all the way to the last, in each iteration. This also suggests that the order in which we proceed in each iteration of the asynchronous updates should be varied, so that information from the last node travels all the way back to the first during even iterations.

3.2 Bipartite graphs

In [5], Felzenswalb and Huttenlocher give a simple way to reduce the size and time requirements of BP on a lattice by a factor of two. The key observation that they make is that lattice graphs are bipartite (that is, the set of nodes can be broken into two sets such that all edges go from a node in one set to a node in the other—see Figure 3-1).
Let the two sets of nodes in the graph be $S_1$ and $S_2$ (the black and white nodes in Figure 3-1). We will use the notation $m_S$ to indicate the set of messages leaving a set of nodes $S$. Then, in any given iteration (assuming again that we're using synchronous updates), the messages leaving the nodes in $S_1$ at iteration $T+1$ depend only on the old messages leaving $S_2$ at iteration $T$ (and vice versa). In effect, we can completely decouple the two sets of nodes. If we start with the messages for $S_1$ at iteration 1, we can generate the messages for $S_2$ at iteration 2, the messages for $S_1$ at iteration 3, etc. We can cut our space requirements for messages in half, since the new messages don't overlap the old ones and can be calculated in place, and because we only calculate half the messages at any iteration, the time required is also halved. After $T$ iterations, the results will be exactly the same as those from the standard algorithm for one of the sets of nodes and will lag one iteration on the other set. For instance, after two iterations, the messages for $S_2$ will be correct, but the messages for $S_1$ will be the same as the results of standard algorithm after one iteration. A more detailed proof of this fact is given in [5].

There are many consequences of the bipartite nature of a lattice that the authors of [5] do not address. For instance, it is slightly troubling that by running the algorithm as described above, the initial messages for half of the graph are not even taken into account! This observation also allows us to prove a perhaps surprising result: for any MRF with at least two local minima, there is a set of initial messages (or, in the case of MF, initial beliefs) such that BP (respectively, MF) never converges when we compute the messages in the traditional way. The basic idea is that we will set the messages on $S_1$ to correspond to one minimum (call it $M$), and those on $S_2$ to correspond to the other ($N$). Then, after one iteration, the new messages for $S_1$ will depend only on the old messages for $S_2$, and will therefore correspond to $N$. Similarly, the new messages on $S_2$ will now correspond to $M$. After each iteration, the messages on $S_1$ and $S_2$ will switch which minima they correspond to, and the algorithm will never converge.

Of course, the preceding analysis leaves unanswered the question of whether it is possible for these methods to fail even if we do calculate the messages one set at
a time. As far as we have observed, BP and MF *always* converge when using the techniques of this section (or for that matter, using asynchronous updates). In other words, when using the initial framework for synchronous updates, BP and MF fail to converge not because they do not find a local minimum, but rather only because they find two local minima, which alternate “fields” with each iteration.

This behavior explains why odd artifacts often occur when BP or MF fail to converge. One such illustration is provided in Figure 3-2. To generate this example, we generated a simple two-label image. We then randomly flipped 30% of the labels and ran the mean field method on the resulting lattice. We found that there were strange checkerboard patterns that alternated labels from one iteration to the next. The “true minima” were generated by taking the even pixels in one image combined with the odd pixels from the other. This resulted in labelings that appeared self-consistent.

The types of artifacts found in Figure 3-2 occur far more frequently with MF than with BP. One inference we can draw from this is that MF has more local minima than BP. After all, the more local minima there are, the more likely it is that each field’s messages are initially set to converge to a different minimum. In turn, this difference in the number of local minima for each technique explains why mean field has often been found to converge to poorer solutions than BP: there are more bad solutions to converge to.

### 3.3 The “Strips” modification

For the mean field method, reducing the memory requirements to calculating in place (as the algorithms in either of the last two section will allow us to do) is generally about the best we can hope for, since we have to store the beliefs somewhere anyway. On the other hand, belief propagation requires the storage of messages in addition to beliefs, so calculating in place will still require several times as much space as is strictly needed. Because its memory requirements are so high, and because medical images can be quite large, we would like to be able to reduce memory constraints.
Figure 3-2: This figure demonstrates the result of running the mean field method on a highly degraded image. Rather than converging, the beliefs alternate between the images in the middle row (note that the checkerboard patterns are reversed from one image to the other). This demonstrates how the actual minima shown in the bottom row have been “interlaced”. A similar phenomenon sometimes occurs when running belief propagation.
still further, even if it is at the expense of computation time. In this section we will present a novel technique for exactly such a space reduction, under the assumption that the number of iterations required is known, and significantly less than the largest lattice dimension.

We again assume that we are using a synchronous update schedule. Then in $T$ iterations, information can only travel $T$ nodes in any direction. For instance, as long as the width of the image is greater than $T$, the beliefs for a pixel at the rightmost edge after $T$ iterations will not depend in any way on the messages passed from any nodes on the left edge. In other words, for the purposes of calculating the belief of a node on the right edge, we can treat the lattice as though the left edge weren’t even there.

This observation inspires us to use something similar to a windowing technique. We only need to hold part of the data in memory to compute the messages and beliefs for pixels towards the edge. For instance, we can split the lattice approximately in half (with a little bit of an overlap). If the lattice is $w$ nodes wide, we can calculate the messages in columns $1, \ldots, w/2$ using only the columns $1, \ldots, w/2 + T$, and those in columns $w/2, \ldots, w$ using only columns $w/2 - T, \ldots, w$. Then, we only need $1/2 + T/w$ as much memory as we did using the straightforward algorithm, and it will take $1 + 2T/w$ times as long to run. If $T \ll w$, this is almost a savings of half the memory required for messages, for only a very small penalty in time.

Of course, there’s no reason to cut the graph in half, rather than into smaller strips. We can calculate the final beliefs for thirds, fourths, or any fraction (again using an overlap, of course, although now we need to have an overlaps for each side of an interior strip). We then essentially slide our window over the lattice, calculating the beliefs only for those nodes $T$ pixels or further into the window. If we use a fraction $\rho$ of the columns, it will require around $\rho + T/w$ times as much memory as the regular algorithm, and take about $1 + 2T/(\rho w)$ times as much time. We can go all the way until each strip only calculates the beliefs one pixel wide, but the tradeoff in time becomes a serious hit (the algorithm will take $2T + 1$ times as long!).

It is therefore best to determine how much of a memory-time tradeoff is best
for any particular application in advance. Fortunately, even more space savings are possible, since this technique can be combined with that from the previous section, cutting the space requirements in half again, and reducing the time requirement by half as well.
Chapter 4

Graph Cuts

Recently, techniques have been developed that can easily minimize functions of the form \( \sum_i D_i(x_i) + \sum_{(i,j) \in N} V_{ij}(x_i, x_j) \). If we want to find the labeling that maximizes the a posteriori probability

\[
\Pr(x) = \frac{1}{Z} \prod_{(i,j) \in N} \psi_{ij}(x_i, x_j) \prod_i \phi_i(x_i), \tag{4.1}
\]

we can instead minimize the function

\[
- \ln(Z \Pr(x)) = - \left[ \sum_i \ln(\phi_i(x_i)) + \sum_{(i,j) \in N} \ln(\psi_{ij}(x_i, x_j)) \right], \tag{4.2}
\]

leading to the obvious assignments \( D_i(x_i) = -\ln(\phi_i(x_i)) \) and \( V_{ij} = -\ln(\psi_{ij}(x_i, x_j)) \).

The function \( D \) is referred to as the data penalty and the function \( V \) is referred to as the interaction penalty.

Graph cut techniques have been used extensively in applications using MRFs ranging from stereo vision to image restoration to minimize functions having this form. This research has been quite encouraging, but many of the results that have been attained are not immediately applicable to the realm of image segmentation because of differences in the types of compatibility functions used.

The idea behind the graph cuts approach is as follows: in order to calculate the minimum of the energy function, we will create a graph such that the minimum cut
of the graph also minimizes the energy. The minimum cut is then calculated using a max-flow algorithm \[9\]. Given a directed graph \( G = (\mathcal{V}, \mathcal{E}) \) with two special vertices \( s \) and \( t \) and non-negative edge weights, a cut is a partition of \( \mathcal{V} \) into two sets \( S \) and \( T = \mathcal{V} \setminus S \) such that \( s \in S \) and \( t \in T \). The cost of the cut is defined to be the sum of all edge weights (costs) from vertices in \( S \) to those in \( T \),

\[
c(S, T) = \sum_{a \in S, b \in T} c(a, b).
\]

The Max-flow min-cut theorem \([3, \text{p. 593}]\) shows that the value of the minimum cut is the same as the value of the maximum flow in the graph. Many algorithms have been developed that make use of this fact to calculate the minimum cut in polynomial time \([3, 9]\).

### 4.1 Image restoration using graph cuts

A typical problem that demonstrates the usefulness of graph cuts is restoring an image that has become corrupted with noise. In this type of problem, we are given a corrupted image, and the goal is to recover as accurately as possible the original image. The formulation of this problem is essentially the same as for that of image segmentation, except that the classes we are trying to segment the image into are exactly the image intensity levels themselves. In addition, we expect noise to affect the corrupted pixels by sending true intensity levels to different levels that are "nearby" in some sense.

As stated previously, this problem is just a particular type of image segmentation problem. Furthermore, we expect each corrupted intensity to be mapped to a value that is near to the true intensity. Because of the highly constrained nature of the problem and the fact that we often want an algorithm that can run be run on all types of images (as opposed to only brain scans, e.g.), it makes sense to use a fairly simple compatibility matrix which does not vary over sets of neighbors. Also, because of the wide variety of images that may be used, rather than using empirical data to
determine the compatibility matrix, a simple analytic form is often used (for example, $V_{ij}(x_i, x_j) = |x_i - x_j|$). Because the potential is the same for all pairs, for notational simplicity we will drop the subscripts. The previous example potential then becomes $V(\alpha, \beta) = |\alpha - \beta|$.

Given the nature of this problem, there are several natural assumptions on the form of compatibility function which may be made [2]:

$$V(\alpha, \beta) = 0 \Leftrightarrow \alpha = \beta,$$  \hspace{1cm} (4.3)

$$V(\alpha, \beta) = V(\beta, \alpha) \geq 0,$$  \hspace{1cm} (4.4)

$$V(\alpha, \beta) \leq V(\alpha, \gamma) + V(\gamma, \beta).$$  \hspace{1cm} (4.5)

Following [2], we will call a potential satisfying conditions 4.3 and 4.4 a semi-metric, and a potential satisfying all three conditions a metric.

In [2], Boykov et al. provide two methods for estimating MAP assignments. The first method, the $\alpha$-$\beta$-swap, works for either metrics or semi-metrics. The second method, the $\alpha$-expansion, works only for metrics. We will now provide a quick overview of the graphs created for each method, adapted from [2].

4.1.1 The $\alpha$-$\beta$-swap

At the beginning of the $\alpha$-$\beta$-swap algorithm, each pixel is assigned an arbitrary label. On each iteration, a pair of labels $\alpha$ and $\beta$ is chosen (usually in a predetermined order). We would like to consider the set of labelings where all of the pixels not set to either of these values remain fixed, but any subset of the pixels assigned to $\alpha$ and $\beta$ can switch labels. A graph is constructed such that there is a one-to-one correspondence between cuts and assignments of these pixels to $\alpha$ and $\beta$, and such that the cost of the cut is the same as the energy of the corresponding assignment. The pseudocode outline for this procedure provided in [2] is included in Figure 4-1.

The graph used to calculate step 3.2 is not too difficult to generate. Let $f_i$ be the label currently assigned to pixel $v_i$. Let $P_{\alpha\beta} = \{v_i | f_i \in \{\alpha, \beta\}\}$. In addition to the
1. Start with an arbitrary labeling $f$
2. Set success := 0
3. For each pair of labels $\alpha, \beta \in \mathcal{L}$
   3.1. Find $\hat{f} = \arg\min E(f')$ among $f'$ within one $\alpha$-$\beta$ swap of $f$
   3.2. If $E(\hat{f}) < E(f)$, set $f := \hat{f}$ and success := 1
4. If success = 1 goto 2
5. Return $f$

Figure 4-1: Pseudocode description of $\alpha$-$\beta$-swap algorithm, taken from [2].

<table>
<thead>
<tr>
<th>edge</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_i^\alpha$</td>
<td>$D_i(\alpha) + \sum_{j \in N(i) \setminus \mathcal{P}<em>{\alpha \beta}} V</em>{ij}(\alpha, f_j)$</td>
</tr>
<tr>
<td>$e_i^\beta$</td>
<td>$D_i(\beta) + \sum_{j \in N(i) \setminus \mathcal{P}<em>{\alpha \beta}} V</em>{ij}(\beta, f_j)$</td>
</tr>
<tr>
<td>$e_{ij}$</td>
<td>$V_{ij}(\beta, \alpha)$</td>
</tr>
</tbody>
</table>

Table 4.1: Edge weights for the $\alpha$-$\beta$-swap algorithm

source node $s$ and the sink node $t$, our graph will have one node for each vertex $v_i$ in $\mathcal{P}_{\alpha \beta}$. Besides an edge $e_i^\alpha$ from the source and an edge $e_i^\beta$ to the sink, each vertex will also have an edge $e_{ij}$ going to each neighboring vertex in $\mathcal{P}_{\alpha \beta}$. The actual costs associated with each edge are shown in Table 4.1. For a proof of the correctness of this construction, see [2]. Note that for any given iteration, the graph has at most $|\mathcal{P}| + 2$ nodes. Also, there are edges connecting each vertex to the source and sink, and at most one edge from each vertex to each neighbor. Thus, for 2D images, there are at most $(2 + 4)|\mathcal{P}| = 6|\mathcal{P}|$ edges, and for 3D images, there are at similarly at most $8|\mathcal{P}|$ edges. Calculating the max flow on graphs of this size is generally possible in a very reasonable amount of time, and in practice, these upper bounds are reasonably generous. See Chapter 5 for a comparison of the speed of this method to MF and BP.

4.1.2 The $\alpha$-expansion

Boykov et al. also provide another algorithm for finding the minimum energy assignment of an MRF, the $\alpha$-expansion algorithm. Because the graph construction for this algorithm is much more complicated than for the $\alpha$-$\beta$-swap, we will merely provide
1. Start with an arbitrary labeling $f$
2. Set success := 0
3. For each label $\alpha \in \mathcal{L}$
   3.1. Find $\hat{f} = \text{arg min}_{f'} E(f')$ among $f'$ within one $\alpha$-expansion of $f$
   3.2. If $E(\hat{f}) < E(f)$, set $f := \hat{f}$ and success := 1
4. If success = 1 goto 2
5. Return $f$

Figure 4-2: Pseudocode for the $\alpha$-expansion algorithm, taken from [2]

an overview of how the algorithm works. For the full details, the reader is referred to
[2].

In the $\alpha$-expansion algorithm, rather than allowing pixels with two different labels
to swap, at each iteration any set of pixels in the whole graph can switch to a given
label, although any pixels currently assigned to that label cannot change. Pseudocode
from [2] outlining the algorithm can be found in Figure 4-2.

In an MRF with only two labels, it is clear that either the $\alpha$-3-swap or the $\alpha$-
expansion algorithm will find the labeling with the lowest possible energy. Unfortu-
nately, for any number of labels higher than 3, both algorithms merely provide
approximations to the maximum a posteriori labeling, although these approxima-
tions tend to be quite good because of the large size of the sets of moves considered in
each iteration. For the $\alpha$-expansion algorithm, [2] shows that the minimum produced
is within a known factor of the global minimum, while no such result exists for the
$\alpha$-3-swap algorithm.

Of course, there are several significant differences in the two algorithms. One
large restriction is that the $\alpha$-expansion algorithm is only valid for a compatibility
function that is a metric, while the $\alpha$-3-swap algorithm works for all semi-metrics.
Also, the graph construction for the $\alpha$-expansion algorithm is more complicated than
that for the $\alpha$-3-swap algorithm (although [9] gives a much simplified algorithm).
Also, only nodes in $P_{\alpha\beta}$ are required for an iteration of the $\alpha$-3-swap algorithm, while
all nodes in the graph are needed for each iteration of the $\alpha$-expansion algorithm.
On the other hand, this is offset somewhat by the fact that each cycle of the $\alpha$-$\beta$-swap algorithm requires looping through all pairs of labels, while each cycle of the $\alpha$-expansion algorithm only requires looping through each label.

For real-world problems, [2] found that the speed of the $\alpha$-expansion and $\alpha$-$\beta$-swap are approximately equal, as are the results generated (although the swap algorithm found a minimum with slightly lower energy, despite the lack of a guarantee in the accuracy of the algorithm). For our own results using the modified versions of these algorithms explained in the upcoming sections, please see Chapter 5.

### 4.2 Graph Cuts for Image Segmentation

With a few modifications, the methods used for image restoration can also be used to do image segmentation. This should come as no surprise, since image restoration is essentially a special case of image segmentation, as described in the last section. However, several of the assumptions made for the image restoration case need to be relaxed. Several of these changes require only slight changes to the framework developed in the last section. For instance, dropping the requirement that the compatibility function be symmetric requires only the use of a directed graph instead of an undirected graph. Similarly, allowing different potentials for different node pairs presents no real challenges.

Unfortunately, other changes require significant structural changes to the graph constructions used in the previous section. This is because is some ways segmentation is a very different kind of problem. For instance, in a brain scan, we are trying to use the intensities of the image to determine the most likely tissue class at each different part of the brain. The problem is now much different than trying to determine the true intensity of each pixel: there can be multiple tissue classes that have the same intensity distribution, but are likely to be found in different parts of the brain. Likewise, in image restoration contexts, it makes perfect sense to have compatibility potentials penalize subtle differences in intensity less severely than stronger differences. However, in medical image segmentation, this concept is impossible to apply,
since the idea of “subtle” differences between tissue classes is nonsensical. If we are looking to segment white and gray matter, we are looking for a complete partition, and the idea of middle ground makes little sense. Therefore, there is no “gradient” of tissue types, and compatibility matrices should be empirically defined, not generated according to some analytic function.

One large consequence of this shift is that we need to be able to allow $V_{ij}(\alpha, \alpha) \neq V_{ij}(\beta, \beta)$. Fortunately, graph cut techniques have recently been developed which allow us to drop the requirement that $V$ be a metric [9]. The technique developed there essentially provide for a framework that perform an $\alpha$-expansion with any potential satisfying

$$V_{ij}(\alpha, \beta) + V_{ij}(\gamma, \alpha) > V_{ij}(\alpha, \alpha) + V_{ij}(\gamma, \beta).$$

This is an awkward looking condition, but it reduces to the triangle inequality for $V_{ij}(\alpha, \alpha) = 0$. Nonetheless, for an arbitrary empirically derived $V$, it is unclear that this condition is reasonable. Building on the work in [9], we will demonstrate a modified $\alpha$-$\beta$-swap graph construction that merely requires that

$$V_{ij}(\alpha, \beta) + V_{ij}(\gamma, \alpha) > V_{ij}(\alpha, \alpha) + V_{ij}(\beta, \gamma),$$

a condition that will hold as long as

$$V_{ij}(\alpha, \beta) > V_{ij}(\alpha, \alpha)$$

holds for all $\alpha$ and $\beta$. In other words, as long as neighboring pixels are penalized less for having the same state as for differing, this condition will be met. The potentials used for all practical medical imaging applications will satisfy this condition.

We will now give a brief description of the construction required for this new $\alpha$-$\beta$-swap algorithm. For the corresponding $\alpha$-expansion algorithm, and for a more detailed overview of how this general technique works, see [9].
4.2.1 The revised $\alpha$-$\beta$-swap

The basic algorithm is the same as that for the $\alpha$-$\beta$-swap algorithm outlined in 4.1.1:

1. Start with an arbitrary labeling $f$
2. Set success := 0
3. For each pair of labels $\alpha, \beta \in \mathcal{L}$
   3.1. Find $\hat{f} = \arg\min_{f'} E(f')$ among $f'$ within one $\alpha$-$\beta$ swap of $f$
   3.2. If $E(\hat{f}) < E(f)$, set $f := \hat{f}$ and success := 1
4. If success = 1 goto 2
5. Return $f$

Although the steps involved in this revised version of the algorithm are the same, the graph we create is now distinctly different. The construction used relies on what Kolmogorov and Zabih call “the additivity theorem” [9], which basically allows us to create the correct graph via the superposition of edges of several smaller graphs. The graph consists of the source node $s$, the sink node $t$, and one node for each vertex $v$ in $\mathcal{P}_{\alpha\beta}$. Because we no longer have the positivity assumptions of a metric or semi-metric, we need to be slightly more careful when creating the graph than we were in 4.1.1.

If a vertex $v$ is directly connected to the source and sink, then exactly one of these edges must be cut, since a cut must separate the source and sink sets. If we would like the cut of the edge between the source and $v$ to add $w_s$ and the edge between the sink and $v$ to have weight $w_t$, we can replace these weights by $w_s + c$ and $w_t + c$ for any constant $c$ without affecting which edge will be cut, since the value of the cut will be increased by $c$ either way. Then, by choosing $c = -\min(w_s, w_t)$, the weights $(e_{s-v}, e_{v-t})$ become either $(w_s - w_t, 0)$ or $(0, w_t - w_s)$ depending on the sign of $w_s - w_t$, so that there are no negative edges in the graph.

As in 4.1.1, we would like to add an edge $s \rightarrow v_i$ with weight $w_s = D_i(\alpha) + \sum_{j \in N(i) \setminus \mathcal{P}_{\alpha\beta}} V_{ij}(\alpha, f_j)$ and an edge $v_i \rightarrow t$ with weight $w_t = D_i(\beta) + \sum_{j \in N(i) \setminus \mathcal{P}_{\alpha\beta}} V_{ij}(\beta, f_j)$. However, since one (or both) of these quantities could be negative, we will use the result of the previous paragraph. Therefore we add edges with weights $(e_s, e_t)$ of
either \((w_s - w_t, 0)\) or \((0, w_t - w_s)\) depending on the sign of \(w_s - w_t\), guaranteeing that there are no edges with negative weight.

We now need to add edges representing the contributions of the compatibility function for neighbors that are both in \(\mathcal{P}_{\alpha \beta}\). See Figure 4-3 for an illustration of the edges we would add for vertices \(v_i\) and \(v_j\). We have edge weights \(e_{s \rightarrow v_i} = V_{ij}(\alpha, \beta) - V_{ij}(\beta, \beta)\), \(e_{v_j \rightarrow t} = V_{ij}(\alpha, \beta) - V_{ij}(\alpha, \alpha)\), and \(e_{v_i \rightarrow v_j} = V_{ij}(\alpha, \beta) + V_{ij}(\beta, \alpha) - V_{ij}(\alpha, \alpha) - V_{ij}(\beta, \beta)\). Equation 4.6 implies that each of these edges is positive.

With the edges shown in Figure 4-3, we can have four different cuts involving these two vertices, depending on whether each vertex is cut on the source edge or sink edge. The four cases are listed in Table 4.2 (note that cutting the \(s \rightarrow v_j\) edge also requires cutting the \(v_i \rightarrow v_j\) edge to separate \(v_j\) from the source). Notice that, as we would expect, the value of each cut is simply the potential for that assignment of variables plus a constant (in this case, \(V_{ij}(\alpha, \beta) - V_{ij}(\alpha, \alpha) - V_{ij}(\beta, \beta)\)).

When we superimpose edges from the graphs of all neighboring pairs in \(\mathcal{P}_{\alpha \beta}\) and the edges representing the individual nodes in \(\mathcal{P}_{\alpha \beta}\), the additivity theorem shows that the minimum cut for the resulting graph will also be the cut that gives the
Table 4.2: The four different cuts for the graph in Figure 4-3. The first two columns indicate whether the vertices are in the s or t set according to the cut (notice that this is the opposite of which edge is cut). The third column indicates which edges are cut, the fourth column indicates the total value of the cut, and the last column indicates the value of the cut plus \( K = V_{ij}(\alpha, \alpha) + V_{ij}(\beta, \beta) - V_{ij}(\alpha, \beta) \).

minimum energy \( \alpha-\beta \)-swap. For a more rigorous proof of the corresponding fact for an \( \alpha \)-expansion, see [9].
Chapter 5

Results

5.1 Relative accuracy

As mentioned in Chapter 4, for MRFs with only two labels, both graph cut methods produce the exact MAP labeling. It makes sense, then, to test MF and BP against the graph cuts methods. When the number of labels is greater than two, the graph cuts methods are not guaranteed to find the global minimum, and in fact, the minima discovered by the two graph cuts methods may differ. Nonetheless, for problems of a scope that is of practical use, it is impossible in general to find the global minimum in a reasonable amount of time. Therefore, we will still use the graph cuts solution as a baseline for measuring accuracy (and specifically, we will use the $\alpha$-expansion method, since there are guarantees on the quality of its solution).

Figure 5-1 summarizes our accuracy results. We generated data for four different models. In the first three (a-c in the figure) we generated a $40 \times 40$ lattice with a background consisting of one label and a foreground of rectangles of either one, two, or three additional labels, and then randomly switched the assigned labels with a variable probability $p$ (this is the x-axis “noise” parameter). In the fourth example (d), we again generated a foreground of rectangles of two labels, and this time used a Gaussian noise model. For each model, we generated 20 sets of data for each noise level, calculated the maximum labeling according to MF, max-product BP, sum-product BP, and the $\alpha$-\(\beta\)-swap version of graph cuts, and calculated the percentage
of pixels that were classified the same as the $\alpha$-expansion version of graph cuts. The graph cuts methods used original code to generate the graph edges, but used code from [1] to calculate the actual minimum flow (this code calculates the minimum flow in time $O(mn^2|C|)$, where $m$ is the number of edges in the graph and $|C|$ is the weight of the minimum cut, but in practice is faster than the theoretical bound [1]). All of the code used in this section can be found at http://web.mit.edu/kvb/www/.

We find that the $\alpha$-$/\beta$-swap and $\alpha$-expansion algorithms produce nearly identical results, regardless of the number of states or the amount of noise added. It is also quite obvious that the mean field method is the worst performer, and with larger amounts of noise, the performance becomes disproportionately worse. Neither of these results is particularly surprising. It intuitively seems like the $\alpha$-$/\beta$-swap and $\alpha$-expansion algorithms should be able to obtain similar minima (although in pathological cases, this is not the case [2]). Also, the mean field method is based on the most simple model of any of our algorithms, so it comes as no surprise that it performs poorly. On the other hand, the performance of two belief propagation algorithms is quite surprising. Given that the graph cuts algorithms are finding the MAP estimate, we would expect the max-product formulation of belief propagation to come much closer than the sum-product version, but in the graphs it is clear that both methods yield extremely similar results.

This is particularly surprising given recent research findings such as those in [6] that indicate that the MAP estimator is not ideal for vision problems. One reason that we would expect MAP to perform poorly (at least compared to MPE) is illustrated in Figure 5-2. When an image is extremely noisy, the single most likely represented image becomes an image with all pixels assigned to the same label. On the other hand, the most probable label for individual pixels is not necessarily the same for all pixels, as can be seen in the figure. As [6] mention, the result is that, although the MAP solution may have the lowest overall probability, it is an uncharacteristic solution, and therefore is visually implausible. Despite the high level of noise, we can still visually make out the figure in the foreground, whereas the $\alpha$-$/\beta$-swap algorithm assigns all pixels to the background label. The BP algorithm’s minimum also distinguishes
Figure 5-1: Plots comparing the accuracy of MF, both varieties of BP, and the \( \alpha-\beta \)-swap to the \( \alpha \)-expansion method. Plots a-c compare the accuracy of the methods on a \( 40 \times 40 \) 2-dimensional lattice with 2, 3, and 4 labels, respectively with a fixed probability of any pixel being misclassified (this is the "noise" axis parameter). Plot d compares the methods on a \( 40 \times 40 \) lattice with 3 labels under a Gaussian noise model. These graphs were created by averaging 20 runs together. In every case, the \( \alpha-\beta \)-swap generates almost exactly the same labelings as the \( \alpha \)-expansion, while MF is clearly the worst performer. Surprisingly, the MAP version of BP did not perform significantly differently than the MPE version according to this metric.
between the foreground and background figures, even when running the MAP version. More than anything else, this indicates that BP does a much worse job of finding a local minimum than the \( \alpha\beta \)-swap algorithm does.

5.2 Efficiency

Figure 5-3 shows some basic timing results. These results were generated by running the algorithms on toy problems similar to those described in the last section and averaging the time to convergence over ten runs. Because we used the bipartite graph versions of the algorithms, MF and BP converged in each run (although slowly, at times). It is clear that the graph cut algorithms are both comparable, and far faster than any other algorithm. The fact that the \( \alpha\beta \)-swap algorithm has to go through all pairs of labels, while the \( \alpha \)-expansion algorithm only has to go through each label is balanced by the fact that the graphs created by the \( \alpha\beta \)-swap algorithm can be much smaller than the corresponding graphs for the \( \alpha \)-expansion algorithm. In the first graph, the MAP BP result on the far right is likely uncharacteristically large because of a few outliers, since there is no reason for the algorithm to take longer for an \( 80 \times 80 \) lattice than for a \( 120 \times 120 \) lattice.

It is not surprising that the mean field method is faster than belief propagation for a small number of labels. Both methods require \( O(nl^2) \) calculations per iteration, where \( n \) is the number of nodes in the lattice and \( l \) is the number of labels, but the iterative equations for MF are extremely quick to calculate, while BP’s are a bit more involved. However, as the number of labels increases, the number of iterations required for MF to converge over BP ends up outweighing the relative simplicity of the update rule.

There is no reason to expect a priori that either the max-product or sum-product version of belief propagation should be faster, but the sum-product MPE) version of belief propagation consistently beat the max-product version in the tests. When coupled with the fact that the solutions offered by the max-product version are hardly closer to the true MAP solution, it would seem to make sense to always use the sum-
Figure 5-2: Demonstration that the MAP solution is not always the "best" solution in terms of how we process information. The α-β-swap solution on the lower right has a lower energy than the sum-product BP solution on the lower left, even though the BP solution "looks" better.
Figure 5-3: Comparison of timing of our methods on different lattice sizes with different numbers of labels.
product version, regardless of the application. It is also worth noting that in separate tests, the factor of two in time saved by using the bipartite speedup outweighed the speedup provided by asynchronous updates.
Chapter 6

Conclusions

In this paper we have demonstrated that belief propagation outperforms the mean field method when used for image segmentation on lattices. In addition, we have introduced a technique to allow an almost arbitrary reduction in the amount of space required for BP to run by trading off space and time requirements. This technique, when combined with existing techniques, can reduce the amount of memory needed for BP to the point that it can be competitive with the MF and graph cuts methods in terms of both space and time performance.

We have also created a new graph cut technique based on Boykov’s $\alpha-\beta$-swap that works on the most flexible set of compatibility functions possible. This is an important advancement, since previous techniques focused on stereo vision or image restoration, and therefore used a fairly narrow class of compatibility functions. This new method is flexible enough to handle even empirically derived compatibility functions for use in image segmentation, as long as they follow a few easily satisfiable conditions.

As always, there will be tradeoffs involved when considering which algorithm to use. Graph cuts has been shown to be faster and provides a better approximation to the MAP assignment. On the other hand, it is unclear whether the accurate approximation to the MAP assignment that graph cuts provides is more or less useful than the somewhat less accurate approximation of the MPE assignment that belief propagation can be used to calculate.

Whether belief propagation, graph cuts, or some altogether new technique ulti-
mately proves to be the best fit for medical image segmentation, it is clear that the use of the mean field method should be phased out. Other than a space advantage over BP (most of which can be mitigated using the techniques in Chapter 3), there is nothing to recommend MF over either belief propagation or graph cuts. Its performance may be adequate in low noise images, but there is a wealth of evidence that demonstrates that BP is a better tool for general inference problems, and this document has provided further support by showing that for the specific problem of image segmentation BP is demonstrably better as well. Unfortunately, as mentioned in Chapter 2, the inconsistency of medical image segmentations done manually by experts makes it nearly impossible to compare the methods on actual data, since ground truth is impossible to determine (segmentations done by MF are as consistent with human segmentations as human segmentations are to each other). Nonetheless, the growing evidence demonstrating the dominance of BP and graph cuts, combined with the fact that graph cuts generally outperforms MF in terms of both storage space and speed indicate that one of these more sophisticated tools should be used.
Bibliography

[1] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-
cut/max-flow algorithms for energy minimization in vision. In Energy Min-

[2] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy mini-


[4] Christopher Crick and Avi Pfeffer. Loopy belief propagation as a basis for com-
munication in sensor networks. In Proceedings of the 19th Annual Conference on 
Uncertainty in Artificial Intelligence (UAI-03), pages 159–166, San Francisco, 


[6] C. Fox and G. Nicholls. Exact MAP states and expectations from perfect sam-
pling: Greig, Porteous and Seheult revisited. In AIP Conference Proceedings, 


